APBS

1.3

Generated by Doxygen 1.7.2

Wed Oct 20 2010 12:01:33

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

1.2 License

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

Pacific Northwest National Laboratory

Additional contributing authors are listed in the code documentation.

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 instatiated 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 withing 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[][VAPBS_DIM]) 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 boudnary 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 dimII, int simplexType, int faceType[VAPBS_NVS], int vertexType[VAPBS_NVS], int ch 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-Hü 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, Vdata_Format format) 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, double *gradz, double kappa Not implemented yet

Global Vgrid_writeUHBD(Vgrid *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, char *title, double *pvec) This routine does not respect partition information

Global Vpbe_ctor2(Vpbe *thee, Valist *alist, int ionNum, double *ionConc, double *ionRadii, double *ionQ, double T, double soluteDiel, double 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 mxtype[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:

Chapter 6

Data Structure Index

6.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 7

File Index

7.1 File List

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

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

thee	The FEtk geometry manager
externalUpdate	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

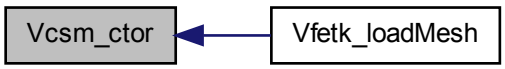
Parameters

alist	List of atoms
gm	FEtk geometry manager defining the mesh

Definition at line 132 of file vcsm.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

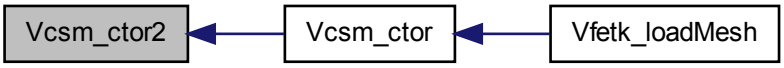
1 if successful, 0 otherwise

Parameters

thee	The Vcsm object
alist	The list of atoms
gm	The FEtk geometry manager defining the mesh

Definition at line 143 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

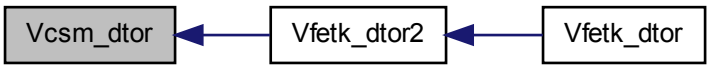
Parameters

thee	Pointer to memory location for Vcsm object

Definition at line 284 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

thee	Pointer to Vcsm object

Definition at line 292 of file vcsm.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

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

Definition at line 73 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

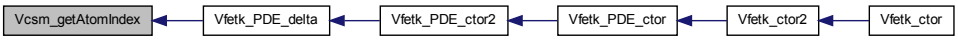
Index of atom in Valist object

Parameters

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

Definition at line 84 of file vcsm.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

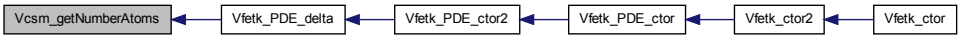
Number of atoms associated with a simplex

Parameters

thee	The Vcsm object
isimp	Simplex ID

Definition at line 65 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

thee	The Vcsm object
iatom	The Valist atom index

Definition at line 95 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

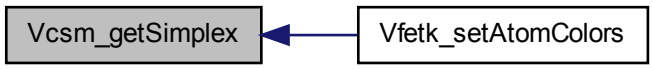
Pointer to simplex object

Parameters

thee	The Vcsm object
isimp	Index of simplex in Vcsm list
iatom	Valist atom index

Definition at line 105 of file vcsm.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

thee	The Vcsm object
isimp	Index of simplex in Vcsm list
iatom	Index of atom in Valist

Definition at line 115 of file vcsm.c.

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

Get atom list.

Author

Nathan Baker

Returns

Pointer to Valist atom list

Parameters

thee	The Vcsm object

Definition at line 58 of file vcsm.c.

8.1.2.13 VEXTERNC void Vcsm_init (Vcsm * *thee*)

Initialize charge-simplex map with mesh and atom data.

Author

Nathan Baker

Note

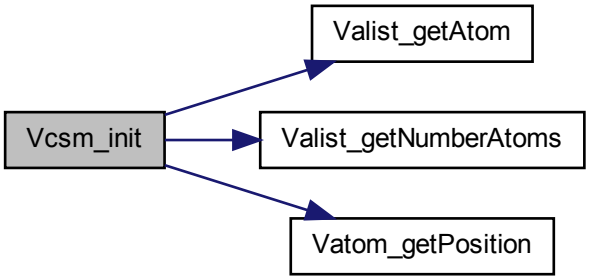
The initial mesh must be sufficiently coarse for the assignment procedures to be efficient

Parameters

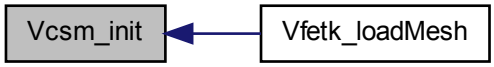
thee	The Vcsm object

Definition at line 166 of file vcsm.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

thee	The Vcsm object

Definition at line 125 of file vcsm.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

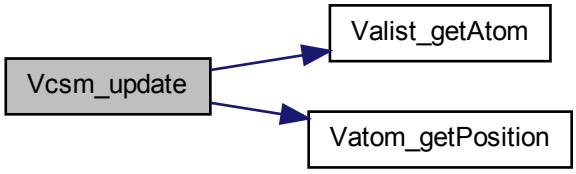
1 if successful, 0 otherwise

Parameters

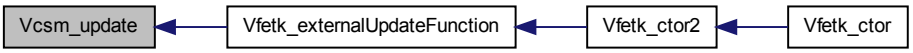
thee	The Vcsm object
simps	List of pointer to newly created (by refinement) simplex objects. The first simplex is expected to be derived from the parent simplex and therefore have the same ID. The remaining simplices are the children and should represent new entries in the charge-simplex map.
num	Number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

Definition at line 318 of file vcsm.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_BT/e) at the finest mesh level.

- VEXTERNC void Vfetk_setParameters (Vfetk *thee, PBEparm *pbeparm, FEMparm *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 frmo 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)

  Intializes 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 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:

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

  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:

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

  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:

  $$\int_\Omega \left[\varepsilon \nabla w \cdot \nabla v + b'(u) w v - f v\right] dx$$

  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:

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

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 127 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 75 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 93 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 110 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 144 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

thee	The matrix to write
fname	Filename for output

Definition at line 958 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

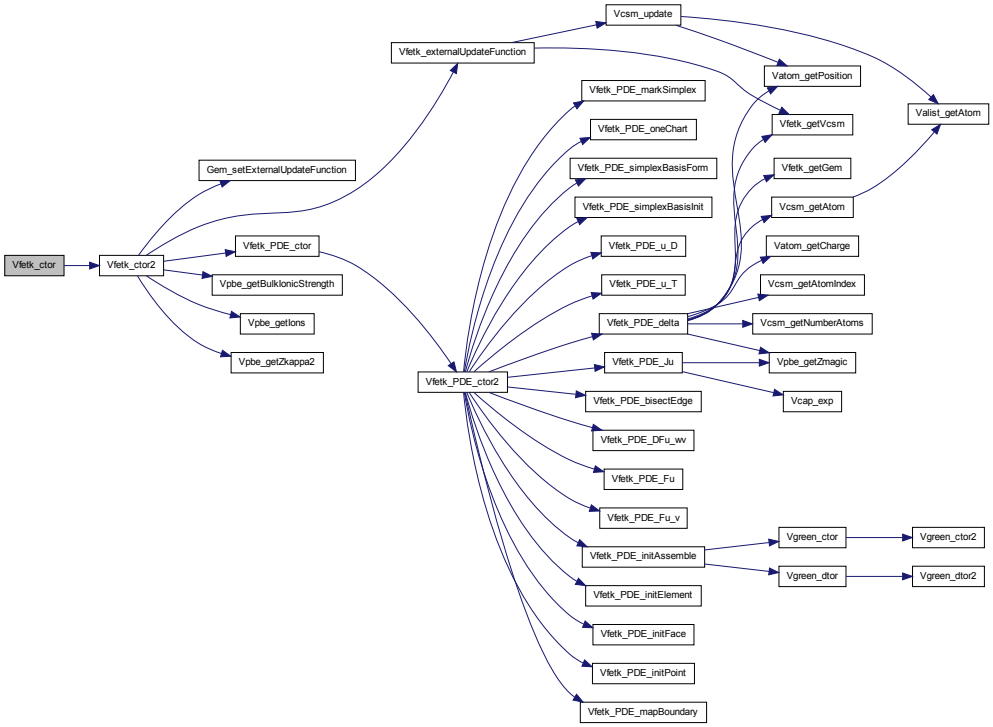
This sets up the Gem, AM, and Aprx FEtk objects but does not create a mesh. The easiest way to create a mesh is to then call Vfetk_genCube

Parameters

pbe	Vpbe (PBE manager object)
type	Version of PBE to solve

Definition at line 527 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

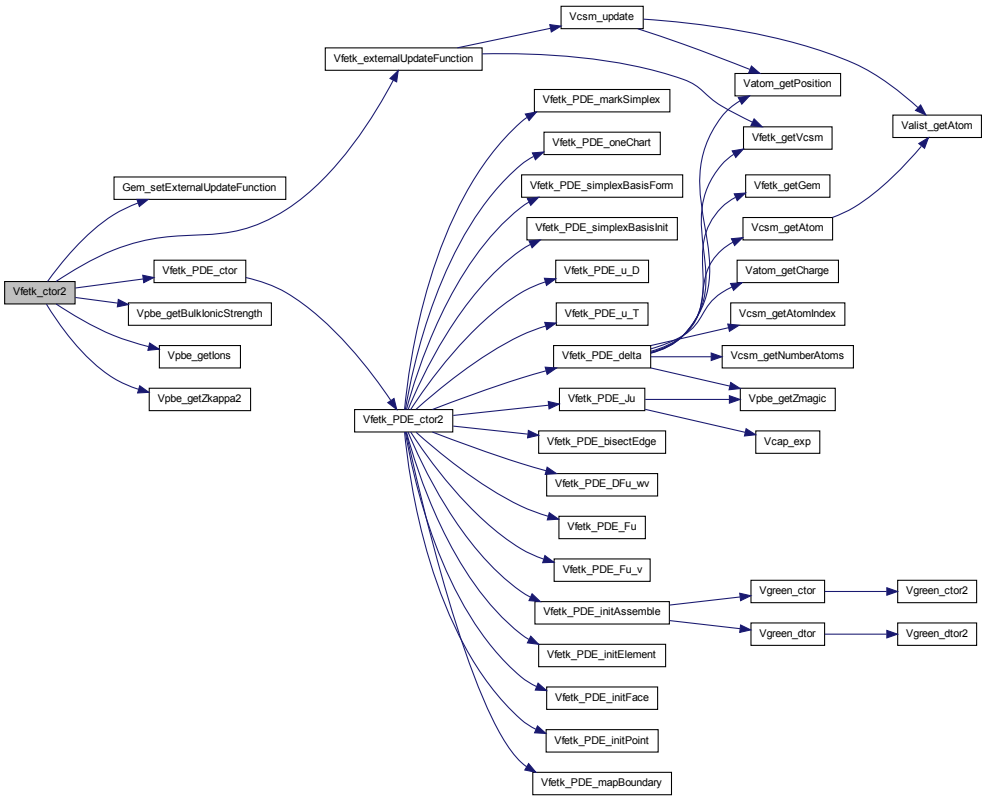
This sets up the Gem, AM, and Aprx FEtk objects but does not create a mesh. The easiest way to create a mesh is to then call Vfetk_genCube

Parameters

thee	Vfetk object memory
pbe	PBE manager object
type	Version of PBE to solve

Definition at line 538 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:

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

for the NPBE and

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

for the LPBE. Here i denotes the counterion species, I_s is the bulk ionic strength, $\overline{\kappa}^2(x)$ 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

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

Returns

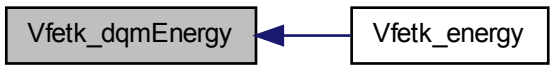
The "mobile charge" and "polarization" contributions to the electrostatic energy in units of $k_B T$.

Parameters

thee	The Vfetk object
color	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 794 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

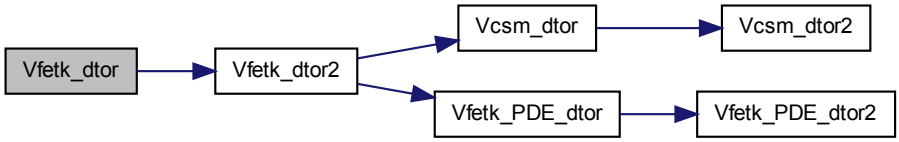
Nathan Baker

Parameters

thee	Pointer to memory location of Vfetk object

Definition at line 613 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

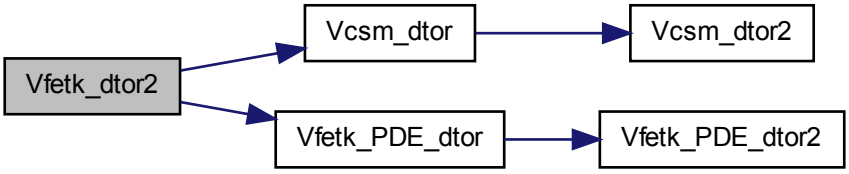
Nathan Baker

Parameters

thee	Pointer to Vfetk object to be destroyed

Definition at line 621 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 2170 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_BT. 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

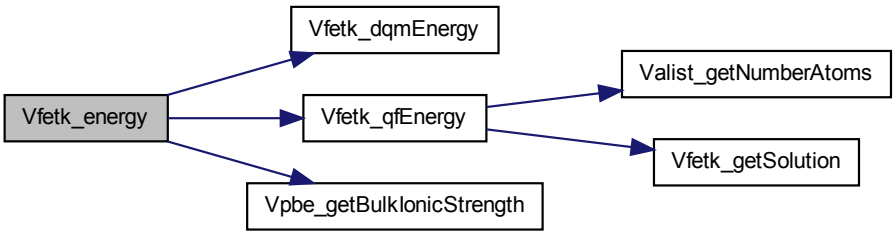
Total electrostatic energy in units of k_BT.

Parameters

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

Definition at line 660 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

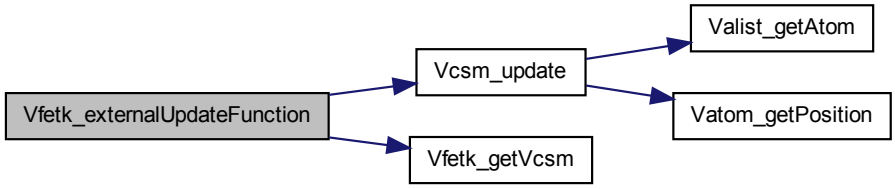
This function is not thread-safe.

Parameters

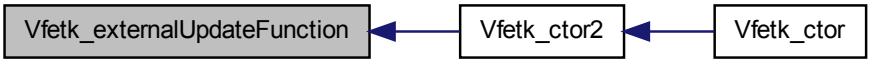
simps	List of parent (simps[0]) and children (remainder) simplices
num	Number of simplices in list

Definition at line 1993 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

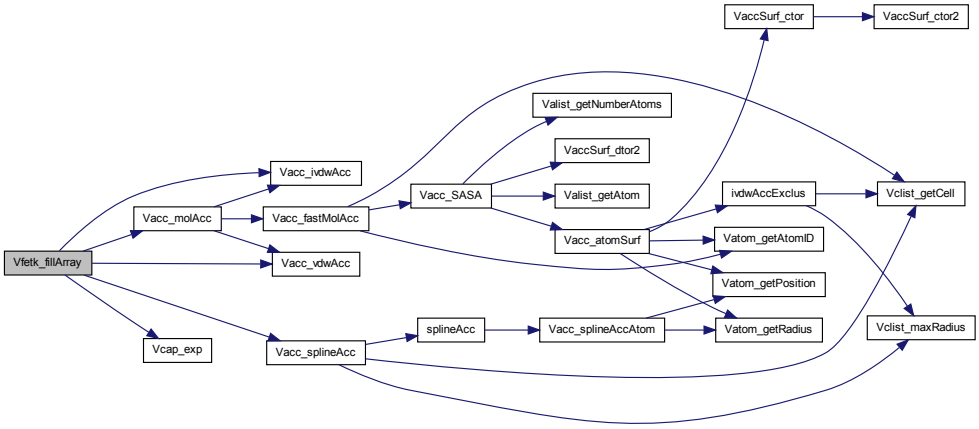
1 if successful, 0 otherwise

Parameters

thee	The Vfetk object with the data
vec	The vector to hold the data
type	THe type of data to write

Definition at line 2214 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

thee	Vfetk object
center	Center for mesh
length	Mesh lengths
meshType	Mesh boundary conditions

Definition at line 830 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

thee	The Vfetk object

Definition at line 494 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

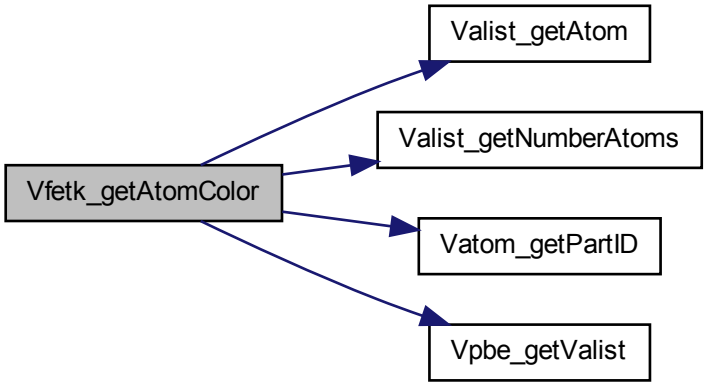
Partition ID

Parameters

thee	The Vfetk object
iatom	Valist atom index

Definition at line 514 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

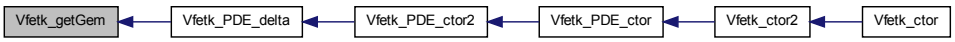
Pointer to the Gem (grid manager) object

Parameters

thee	Vfetk object

Definition at line 487 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

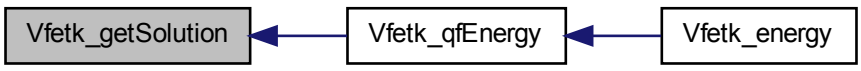
Newly created array of length "length" (see above); the user is responsible for destruction

Parameters

thee	Vfetk object with solution
length	Ste to length of the newly created solution array

Definition at line 629 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

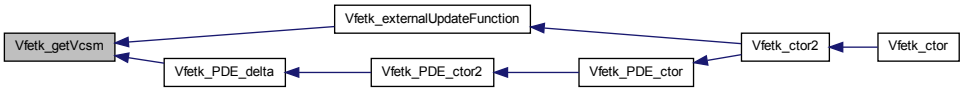
Pointer to the Vcsm (charge-simplex map) object

Parameters

thee	The Vfetk object

Definition at line 507 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

thee	The Vfetk object

Definition at line 500 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

thee	Destination
gm	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

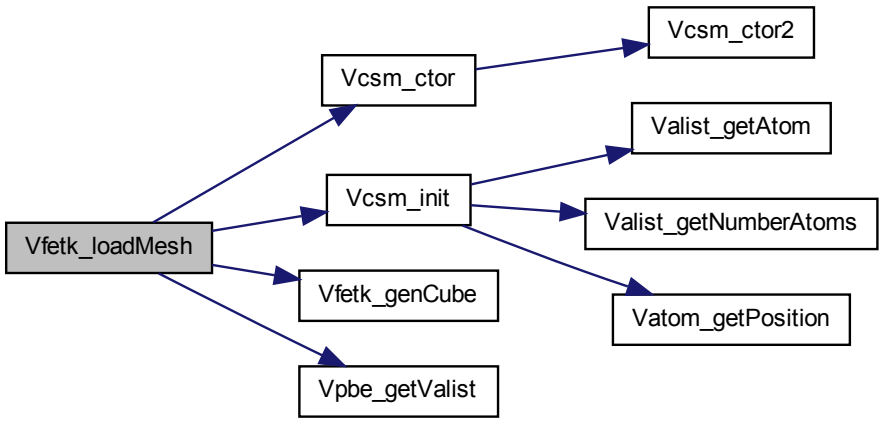
Nathan Baker

Parameters

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

Definition at line 911 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

thee	THe Vfetk object

Definition at line 818 of file vfetk.c.

Here is the call graph for this function:

8.2.3.21 VEXTERNC void Vfetk_PDE_bisectEdge (int *dim,* int *dimII,* int *edgeType,* int *chart[ ],* double *vx[ ][VAPBS_DIM]*)

Define the way manifold edges are bisected.

Author

Nathan Baker and Mike Holst

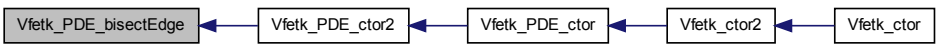
Note

This function is thread-safe.

Parameters

dim	Intrinsic dimension of manifold
dimII	Embedding dimension of manifold
edgeType	Type of edge being refined
chart	Chart for edge vertices, used here as accessibility bitfields
vx	Edge vertex coordindates

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

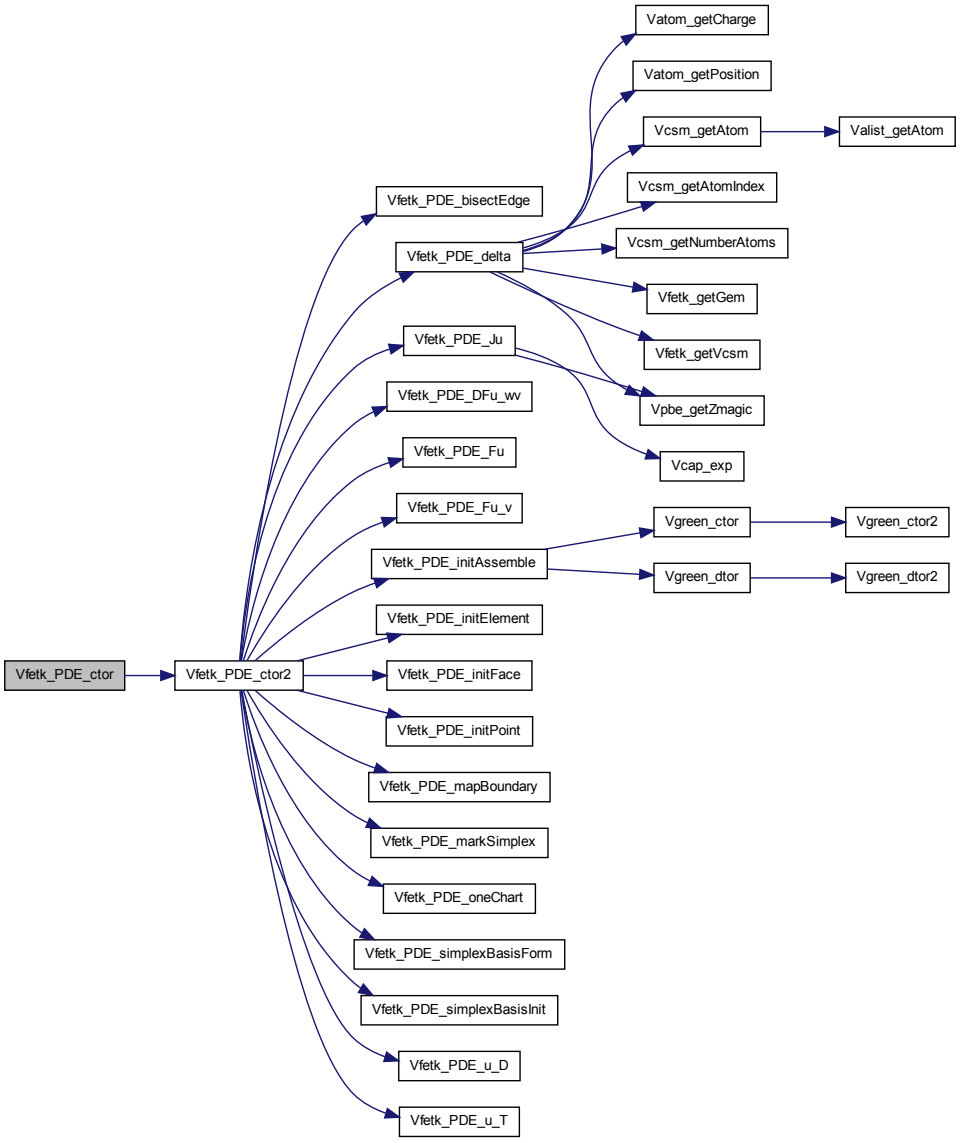
Not thread-safe

Parameters

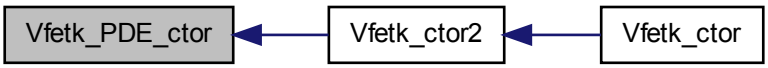
fetk	The Vfetk object

Definition at line 1106 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*)

Intializes the FEtk PDE object.

Author

Nathan Baker (with code by Mike Holst)

Returns

1 if successful, 0 otherwise

Bug

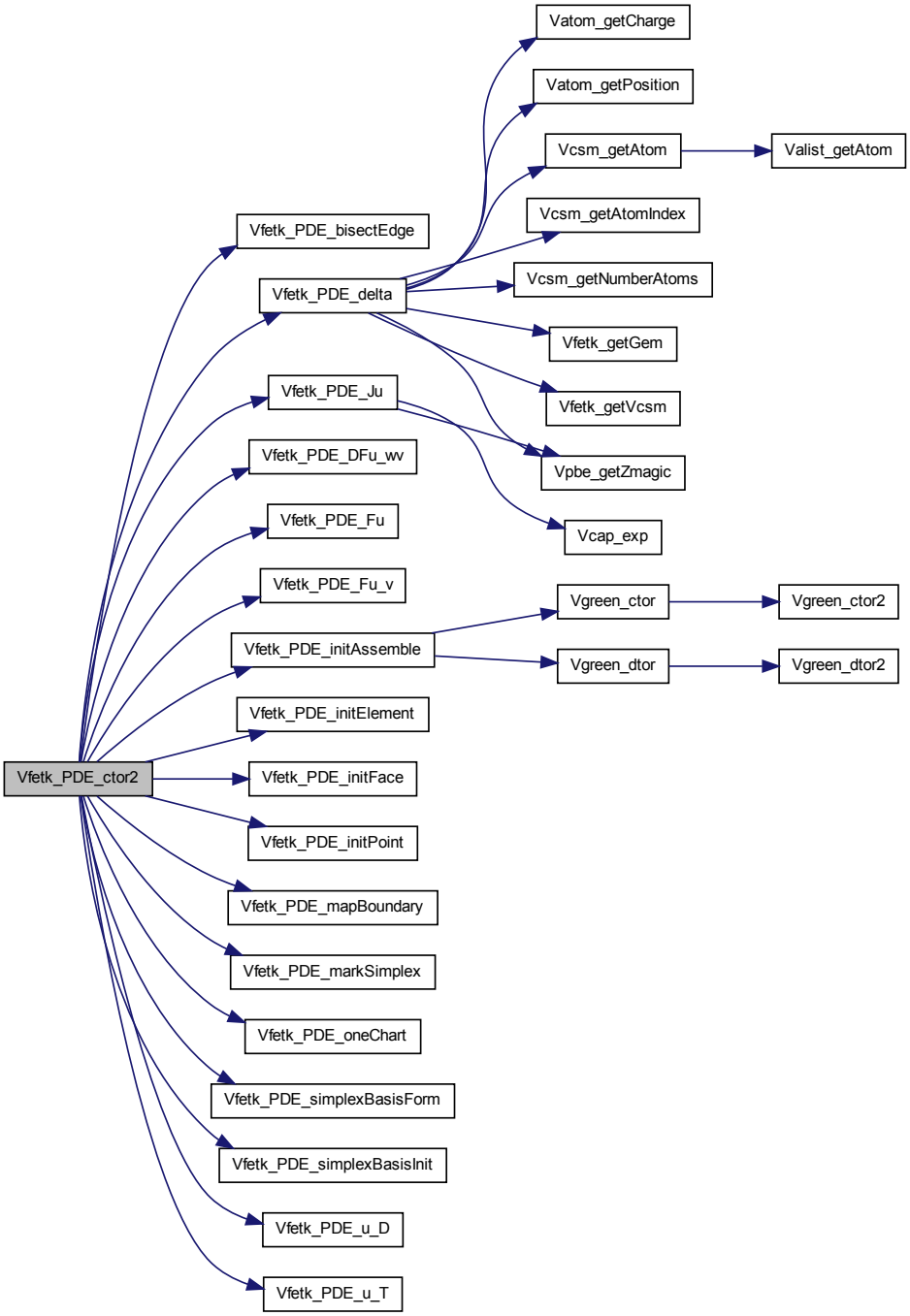
Not thread-safe

Parameters

thee	The newly-allocated PDE object
fetk	The parent Vfetk object

Definition at line 1117 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

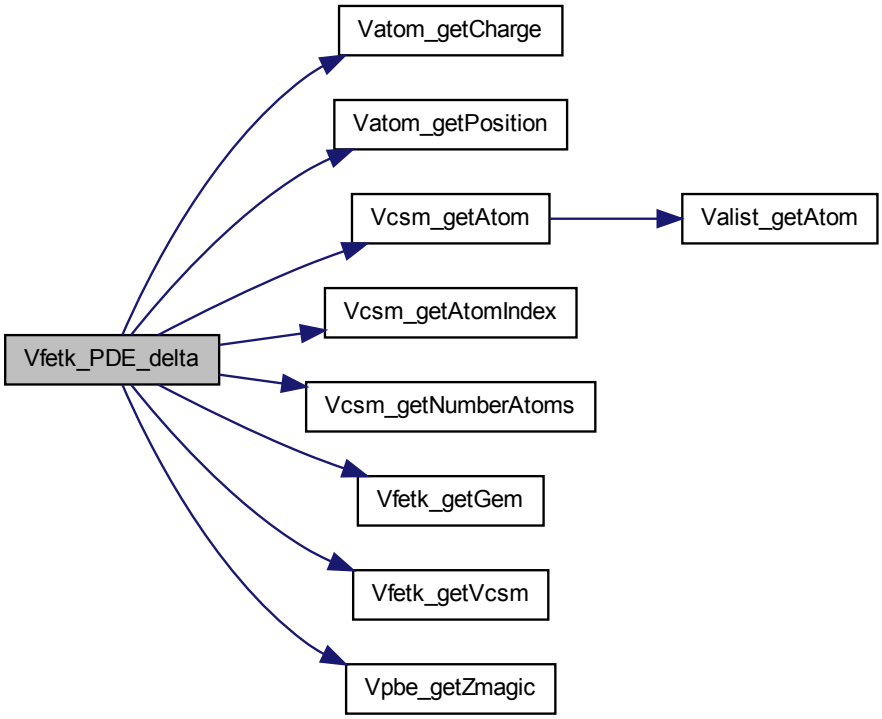
This function is not thread-safe

Parameters

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

Definition at line 1708 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:

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

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

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

thee	Pointer to PDE object memory

Definition at line 1159 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

thee	PDE object memory

Definition at line 1174 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:

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

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:

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

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

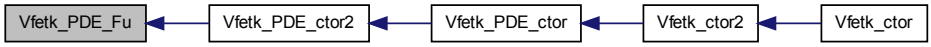
This function is not thread-safe
This function is not implemented (sets error to zero)

Parameters

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

Definition at line 1623 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

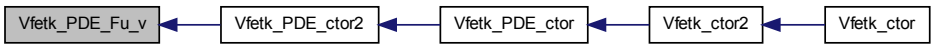
This function is not thread-safe

Parameters

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

Definition at line 1631 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

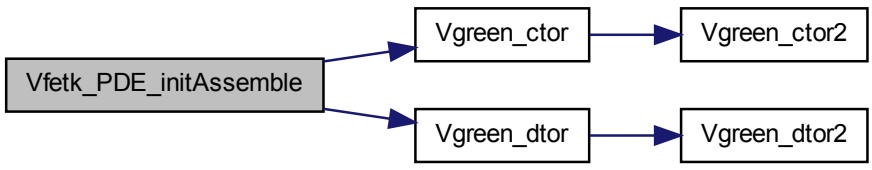
Thread-safe

Parameters

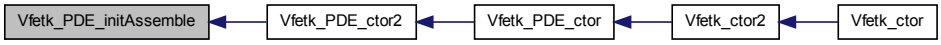
thee	PDE object
ip	Integer parameter array (not used)
rp	Double parameter array (not used)

Definition at line 1436 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

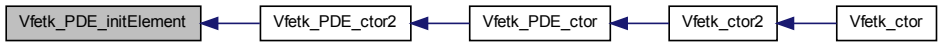
Bug

This function is not thread-safe

Parameters

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

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

Definition at line 1487 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 boudnary definitions for the molecular surface.

Parameters

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

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

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

Author

Nathan Baker

Returns

Energy value (in kT)

Bug

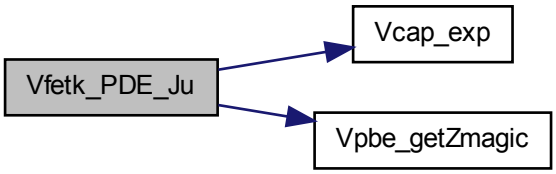
This function is not thread-safe.

Parameters

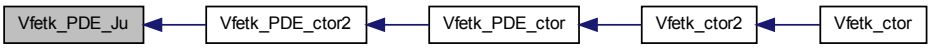
thee	The PDE object
key	What to evluate: interior (0) or boundary (1)?

Definition at line 1918 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 *dimII,* 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

dim	Intrinsic dimension of manifold
dimII	Embedding dimension of manifold
vertexType	Type of vertex
chart	Chart for vertex coordinates
vx	Vertex coordinates

Here is the caller graph for this function:

8.2.3.36 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.

Author

Nathan Baker

Returns

1 if mark simplex for refinement, 0 otherwise

Bug

This function is not thread-safe

Parameters

dim	Intrinsic manifold dimension
dimII	Embedding manifold dimension
simplexType	Type of simplex being refined
faceType	Types of faces in simplex
vertexType	Types of vertices in simplex
chart	Charts for vertex coordinates
vx	Vertex coordinates
simplex	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

dim	Intrinsic manifold dimension
dimII	Embedding manifold dimension
objType	???
chart	Charts of vertices' coordinates
vx	Vertices' coordinates
dimV	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

key	Basis type to evaluate (0 = trial, 1 = test, 2 = trialB, 3 = testB)
dim	Spatial dimension
comp	Which component of elliptic system to produce basis for
pdkey	Basis partial differential equation evaluation key: • 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...
xq	Set to quad pt coordinate
basis	Set to all basis functions evaluated at all quadrature pts

Definition at line 2118 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, eletromagnetics, 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

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

Definition at line 2055 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

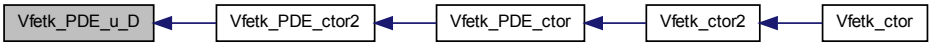
This function is hard-coded to call only multiple-sphere Debye-Hü functions.
This function is not thread-safe.

Parameters

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

Definition at line 1795 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

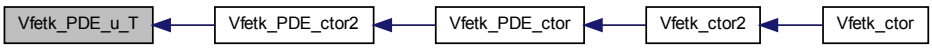
This function is not thread-safe.

Parameters

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

Definition at line 1808 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

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

Returns

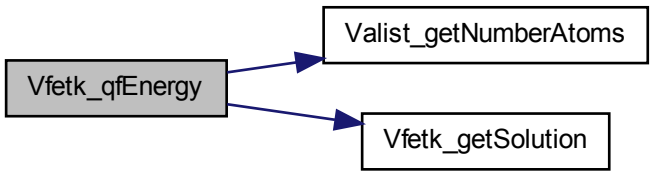
The fixed charge electrostatic energy in units of $k_B T$.

Parameters

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

Definition at line 692 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

thee	THe Vfetk object
skey	The sock format key (0 = MCSF simplex format)
sock	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

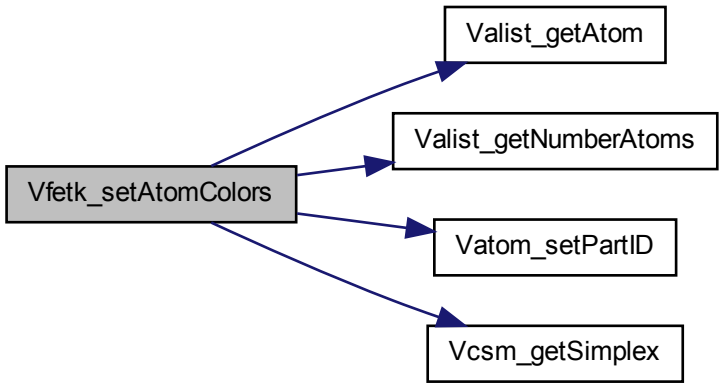
This is a friend function of Vcsm

Parameters

thee	THe Vfetk object

Definition at line 800 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

thee	The Vfetk object
pbeparm	Parameters for solution of the PBE
feparm	FEM-speecific solution parameters

Definition at line 605 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

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

thee	The Vfetk object
iodev	Output device type (FILE = file, BUFF = buffer, UNIX = unix pipe, INET = network socket)
iofmt	Output device format (ASCII = ascii/plaintext, XDR = xdr)

thost	Output hostname for sockets
fname	Output filename for other
vec	Data vector
format	Data format

Definition at line 2379 of file vfetk.c.

8.3 Vpee class

This class provides some functionality for error esimation 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 esimation in parallel. This class provides some functionality for error esimation 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

gm	FEtk geometry manager object
localPartID	ID of the local partition (focus of refinement)
killFlag	A flag to indicate how error estimates are to be attenuated outside the local partition: • 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 (killParam∗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
killParam	

See also

killFlag for usage

Definition at line 76 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

thee	The Vpee object
gm	FEtk geometry manager object
localPartID	ID of the local partition (focus of refinement)

killFlag	A flag to indicate how error estimates are to be attenuated outside the local partition: • 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 (killParam∗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
killParam	

See also

killFlag for usage

Definition at line 94 of file vpee.c.

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

Object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of the Vpee object

Definition at line 197 of file vpee.c.

8.3.2.4 VEXTERNC void Vpee_dtor2 (Vpee ∗ *thee*)

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to object to be destroyed

Definition at line 212 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

thee	The Vpee object
am	The FEtk algebra manager currently used to solve the PB
level	The current level of the multigrid hierarchy
akey	The marking method: • -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.
rcol	The ID of the main parition on which to mark (or -1 if all partitions should be marked). NOte that we shouldhave (rcol == thee->localPartID) for (thee->killFlag == 2 or 3)
etol	The error tolerance criterion for marking

bkey	How the error tolerance is interpreted: • 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 220 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

thee	The Vpee object

Definition at line 438 of file vpee.c.

8.4 APOLparm class

Parameter structure for APOL-specific variables from input files.

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_COMPS Calculate per-atom energy components

Definition at line 68 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_COMPS Calculate per-atom force components

Definition at line 84 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 100 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

thee	APOLparm object

Returns

Success enumeration

Definition at line 173 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

thee	Destination object
source	Source object

Definition at line 102 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 59 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

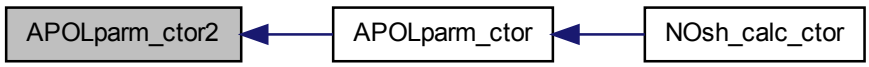
thee	Pointer to allocated APOLparm object

Returns

Success enumeration

Definition at line 70 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

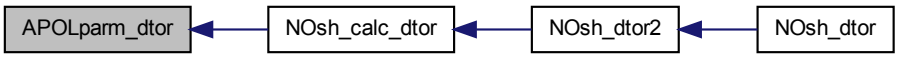
Parameters

thee	Pointer to memory location of APOLparm object

Definition at line 161 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

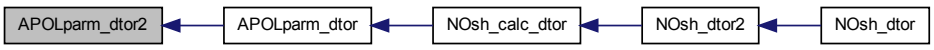
David Gohara

Parameters

thee	Pointer to APOLparm object

Definition at line 171 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.

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 FEparm_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 refinment error estimate tolerance key.
- enum eFEMparm_EstType {
  FRT_UNIF = 0, FRT_GEOM = 1, FRT_RESI = 2, FRT_DUAL = 3,
  FRT_LOCA = 4 }
  Adaptive refinment 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 FEparm_EtolType type.

Author

Nathan Baker

Definition at line 79 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 106 of file femparm.h.

8.5.3.2 enum eFEMparm_EstType

Adaptive refinment 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 87 of file femparm.h.

8.5.3.3 enum eFEMparm_EtolType

Adaptive refinment 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 68 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

thee	FEMparm object

Returns

1 if OK, 0 otherwise

Definition at line 135 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

thee	Destination object
source	Source object

Definition at line 92 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 59 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

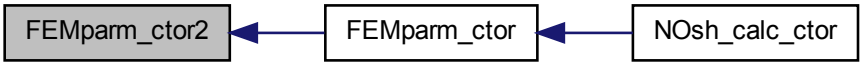
thee	Pointer to allocated FEMparm object
type	FEM calculation type

Returns

1 if successful, 0 otherwise

Definition at line 70 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

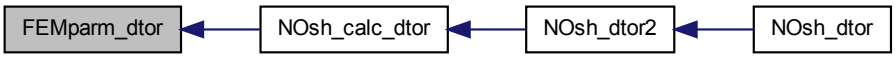
Parameters

thee	Pointer to memory location of FEMparm object

Definition at line 125 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

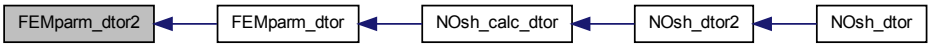
Nathan Baker

Parameters

thee	Pointer to FEMparm object

Definition at line 133 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 66 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 84 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

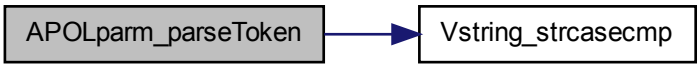
thee	MGparm object
tok	Token to parse
sock	Stream for more tokens

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 571 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

thee	MGparm object
tok	Token to parse
sock	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 417 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

thee	MGparm object

Returns

Success enumeration

Definition at line 181 of file mgparm.c.

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

Copy MGparm object into thee.

Author

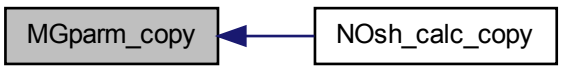
Nathan Baker and Todd Dolinsky

Parameters

thee	MGparm object (target for copy)
parm	MGparm object (source for copy)

Definition at line 337 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

type	Type of MG calculation

Returns

Newly allocated and initialized MGparm object

Definition at line 110 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

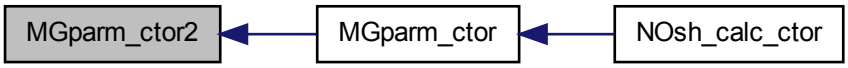
thee	Space for MGparm object
type	Type of MG calculation

Returns

Success enumeration

Definition at line 121 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

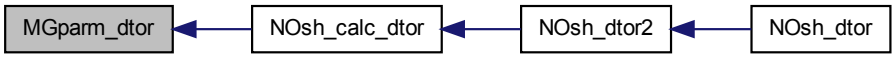
Parameters

thee	Pointer to memory location of MGparm object

Definition at line 171 of file mgparm.c.

Here is the call graph for this function:

Here is the caller graph for this function:



8.6.3.8 VEXTERNC void MGparm_dtor2 (MGparm * *thee*)

FORTRAN stub for object destructor.

Author

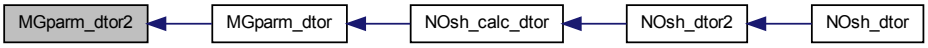
Nathan Baker

Parameters

thee	Pointer to MGparm object

Definition at line 179 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

thee	MGparm object

Returns

x-coordinate

Definition at line 73 of file mgparm.c.

8.6.3.10 VEXTERNC double MGparm_getCenterY (MGparm * *thee*)

Get center y-coordinate.

Author

Nathan Baker

Parameters

thee	MGparm object

Returns

y-coordinate

Definition at line 77 of file mgparm.c.

8.6.3.11 VEXTERNC double MGparm_getCenterZ (MGparm * *thee*)

Get center z-coordinate.

Author

Nathan Baker

Parameters

thee	MGparm object

Returns

z-coordinate

Definition at line 81 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

thee	MGparm object

Returns

Grid spacing in the x direction

Definition at line 97 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

thee	MGparm object

Returns

Grid spacing in the y direction

Definition at line 101 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

thee	MGparm object

Returns

Grid spacing in the z direction

Definition at line 105 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

thee	MGparm object

Returns

Number of grid points in the x direction

Definition at line 85 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

thee	MGparm object

Returns

Number of grid points in the y direction

Definition at line 89 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

thee	MGparm object

Returns

Number of grid points in the z direction

Definition at line 93 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

thee	MGparm object
tok	Token to parse
sock	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 915 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

thee	MGparm object
x	x-coordinate

Definition at line 61 of file mgparm.c.

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

Set center y-coordinate.

Author

Nathan Baker

Parameters

thee	MGparm object
y	y-coordinate

Definition at line 65 of file mgparm.c.

8.6.3.21 VEXTERNC void MGparm_setCenterZ (MGparm * *thee*, double *z*)

Set center z-coordinate.

Author

Nathan Baker

Parameters

thee	MGparm object
z	z-coordinate

Definition at line 69 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 104 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 88 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 120 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 135 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

thee	NOsh object to use
icalc	ID of CALC statement

Returns

The name (if present) of an APOL statement

Definition at line 214 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

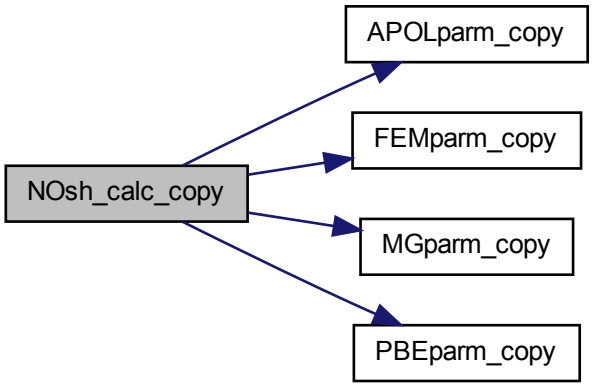
Nathan Baker

Parameters

thee	Target object
source	Source object

Definition at line 368 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

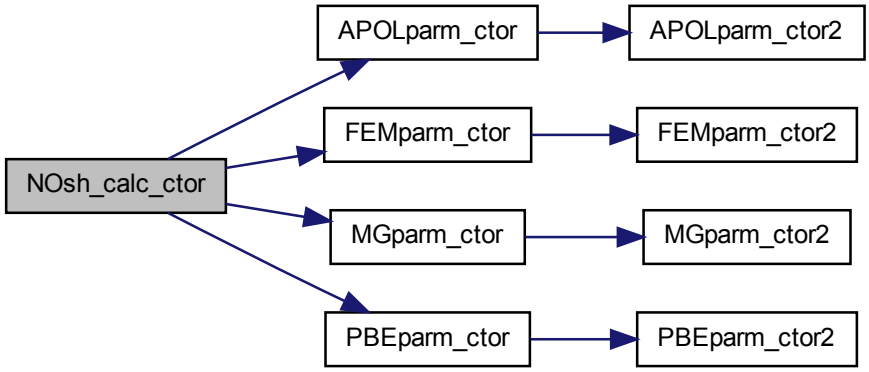
calcType	Calculation type

Returns

Newly allocated and initialized NOsh object

Definition at line 306 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

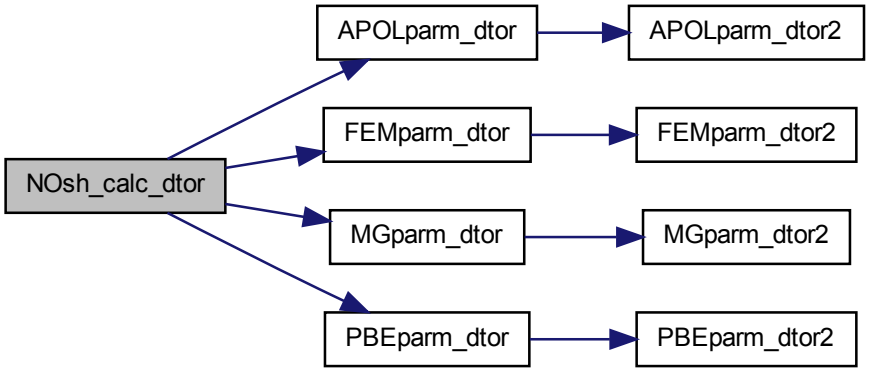
Nathan Baker

Parameters

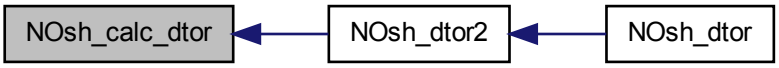
thee	Pointer to memory location of NOsh_calc object

Definition at line 338 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

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

Returns

Newly allocated and initialized NOsh object

Definition at line 240 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

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

Returns

1 if successful, 0 otherwise

Definition at line 251 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

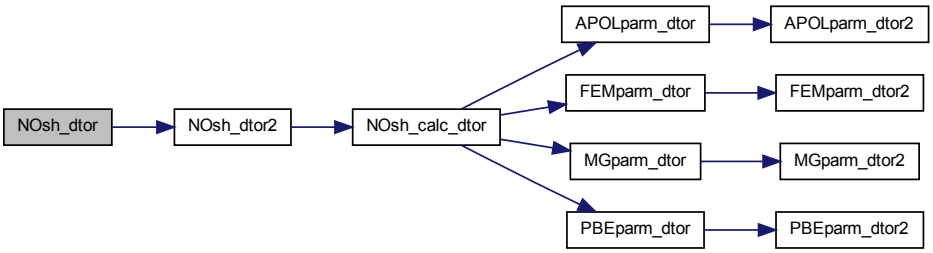
Nathan Baker

Parameters

thee	Pointer to memory location of NOsh object

Definition at line 286 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

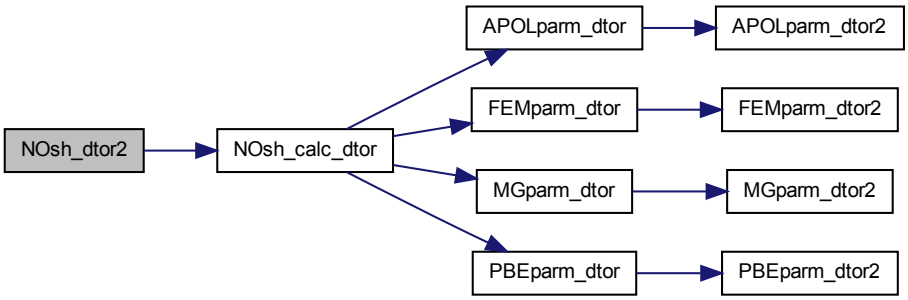
Nathan Baker

Parameters

thee	Pointer to NOsh object

Definition at line 294 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

thee	NOsh object to use
icalc	ID of CALC statement

Returns

The name (if present) of an ELEC statement

Definition at line 208 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

thee	NOsh object to use
ielec	ID of ELEC statement

Returns

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

See also

elec2calc)

Definition at line 220 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

thee	Pointer to NOsh object
icalc	Calculation ID of interest

Returns

Pointer to specified calculation object

Definition at line 167 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

thee	Pointer to NOsh object
imap	Calculation ID of interest

Returns

Format of charge map

Definition at line 187 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

thee	Pointer to NOsh object
imap	Map ID of interest

Returns

Path string

Definition at line 162 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

thee	Pointer to NOsh object
imap	Calculation ID of interest

Returns

Format of dielectric map

Definition at line 172 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

thee	Pointer to NOsh object
imap	Map ID of interest

Returns

Path string

Definition at line 137 of file nosh.c.

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

Returns path to specified y-shifted dielectric map.

Author

Nathan Baker

Parameters

thee	Pointer to NOsh object
imap	Map ID of interest

Returns

Path string

Definition at line 142 of file nosh.c.

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

Returns path to specified z-shifted dielectric map.

Author

Nathan Baker

Parameters

thee	Pointer to NOsh object
imap	Map ID of interest

Returns

Path string

Definition at line 147 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

thee	Pointer to NOsh object
imap	Calculation ID of interest

Returns

Format of kappa map

Definition at line 177 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

thee	Pointer to NOsh object
imap	Map ID of interest

Returns

Path string

Definition at line 152 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

thee	Pointer to NOsh object
imol	Molecule ID of interest

Returns

Path string

Definition at line 132 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

thee	Pointer to NOsh object
imap	Calculation ID of interest

Returns

Format of potential map

Definition at line 182 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

thee	Pointer to NOsh object
imap	Map ID of interest

Returns

Path string

Definition at line 157 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

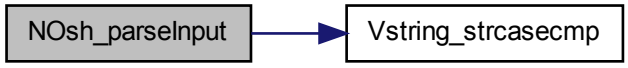
thee	Pointer to NOsh object
sock	Stream of tokens to parse

Returns

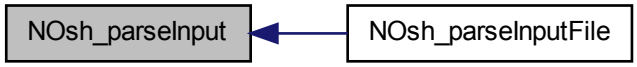
1 if successful, 0 otherwise

Definition at line 404 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

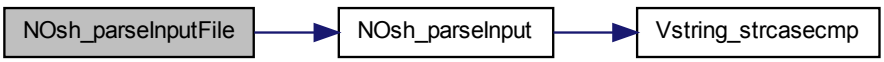
thee	Pointer to NOsh object
filename	Name/path of readable file

Returns

1 if successful, 0 otherwise

Definition at line 389 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

thee	NOsh object to use
iprint	ID of PRINT statement
iarg	ID of operation in PRINT statement

Returns

Calculation ID for specified PRINT statement (

See also

printcalc)

Definition at line 233 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

thee	NOsh object to use
iprint	ID of PRINT statement

Returns

Number of arguments to PRINT statement (

See also

printnarg)

Definition at line 202 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

thee	NOsh object to use
iprint	ID of PRINT statement
iarg	ID of operation in PRINT statement

Returns

Integer ID for specified operation (

See also

printop)

Definition at line 226 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

thee	NOsh object to use
iprint	ID of PRINT statement

Returns

An integer ID of the observable to print (

See also

printwhat)

Definition at line 196 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

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

Returns

1 if successful, 0 otherwise

Parameters

thee	NOsh object
alist	Atom list for calculation

Definition at line 1288 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

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

Returns

1 if successful, 0 otherwise

Parameters

thee	NOsh object
alist	Atom list for calculation

Definition at line 1205 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 72 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_COMPS Calculate per-atom force components

Definition at line 88 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

thee	Object to be checked

Definition at line 177 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

thee	Target for copy
parm	Source for copy

Definition at line 277 of file pbeparm.c.

Here is the caller graph for this function:

8.8.3.3 VEXTERNC PBEparm* PBEparm_ctor ( )

Construct PBEparm object.

Author

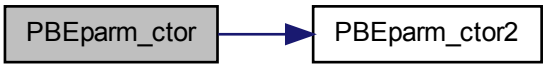
Nathan Baker

Returns

Newly allocated and initialized PBEparm object

Definition at line 98 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

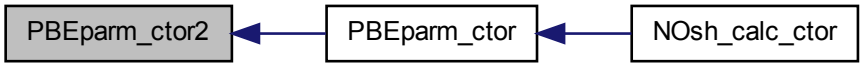
1 if succesful, 0 otherwise

Parameters

thee	Memory location for object

Definition at line 109 of file pbeparm.c.

Here is the caller graph for this function:



8.8.3.5 VEXTERNC void PBEparm_dtor (PBEparm ** *thee*)

Object destructor.

Author

Nathan Baker

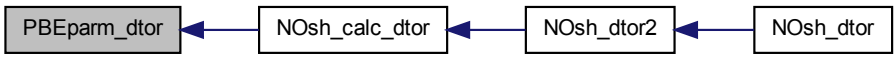
Parameters

thee	Pointer to memory location of object

Definition at line 167 of file pbeparm.c.

Here is the call graph for this function:

Here is the caller graph for this function:



8.8.3.6 VEXTERNC void PBEparm_dtor2 (PBEparm * *thee*)

FORTRAN stub for object destructor.

Author

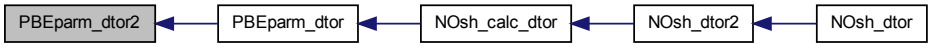
Nathan Baker

Parameters

thee	Pointer to object to be destroyed

Definition at line 175 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

thee	PBEparm object
iion	Ion species ID/index

Definition at line 59 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

thee	PBEparm object
iion	Ion species ID/index

Definition at line 65 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

thee	PBEparm object
iion	Ion species ID/index

Definition at line 71 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

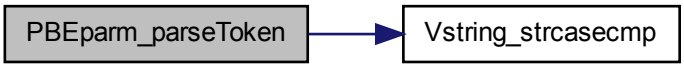
1 if matched and assigned; -1 if matched, but there's some sort of error (i.e., too few args); 0 if not matched

Parameters

thee	Parsing object
tok	Token to parse
sock	Socket for additional tokens

Definition at line 1200 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 derivatve of solvent accessible volume.

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

  Get the derivatve 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 derivatve of solvent accessible area.

Author

Jason Wagoner, David Gohara, Nathan Baker

Parameters

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

Definition at line 1252 of file vacc.c.

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

Get the derivatve of solvent accessible volume.

Author

Jason Wagoner, Nathan Baker

Parameters

thee	Acessibility object
radius	Probe radius (Å)
atom	Atom of interest
dSA	Array holding answers of calc

Definition at line 1141 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

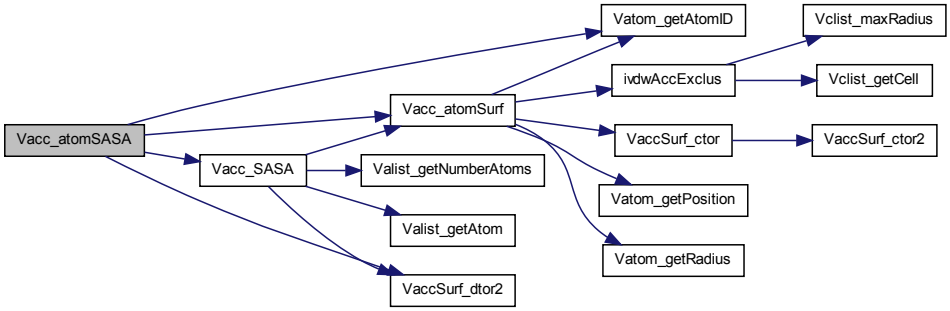
Atomic solvent accessible area (A^2)

Parameters

thee	Accessibility object
radius	Probe molecule radius (Å)
atom	Atom of interest

Definition at line 708 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

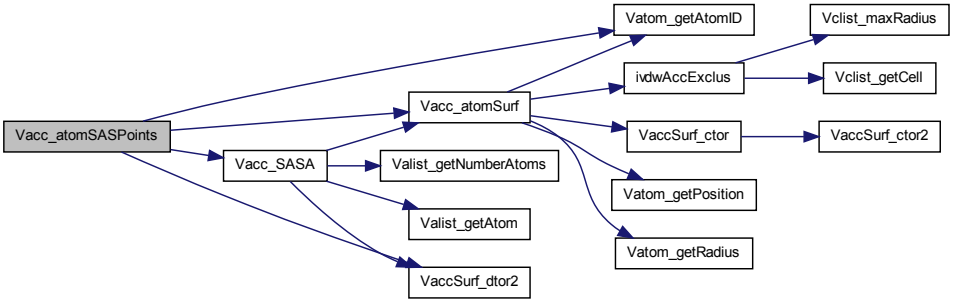
Pointer to VaccSurf object for this atom

Parameters

thee	Accessibility object
radius	Probe molecule radius (Å)
atom	Atom of interest

Definition at line 923 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

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

See also

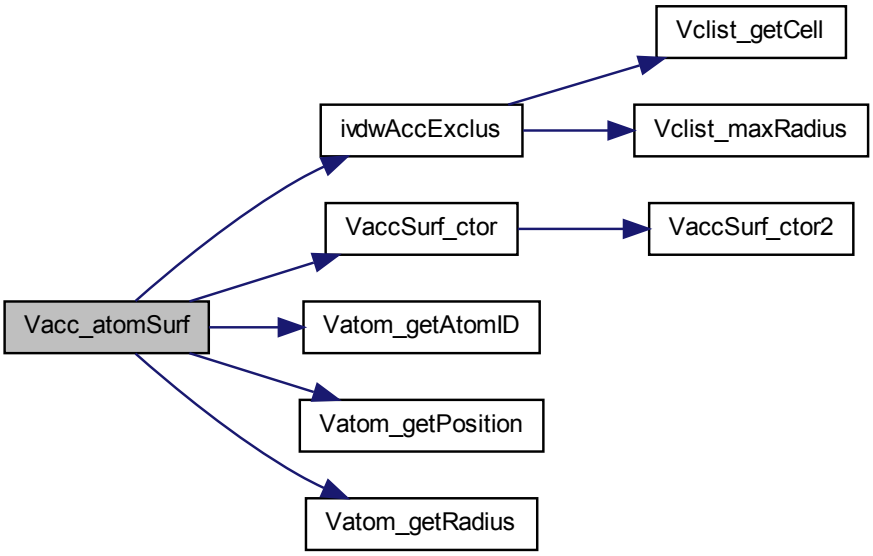
VaccSurf_refSphere

Parameters

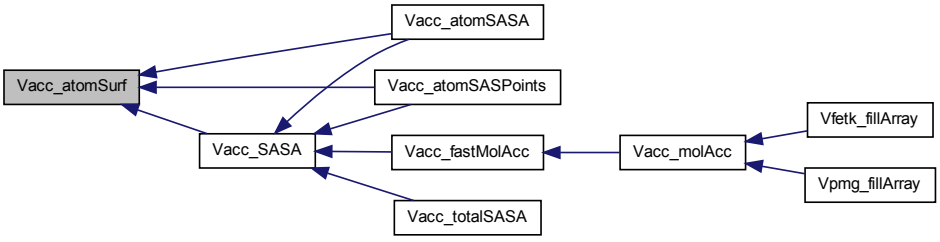
probe_radius	Probe radius (in A)

Definition at line 811 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

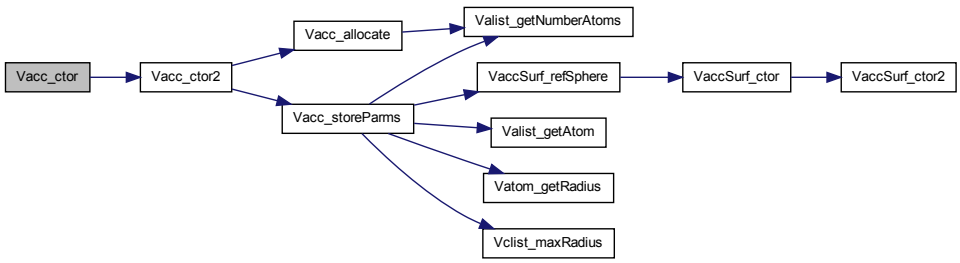
Newly allocated Vacc object

Parameters

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

Definition at line 124 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

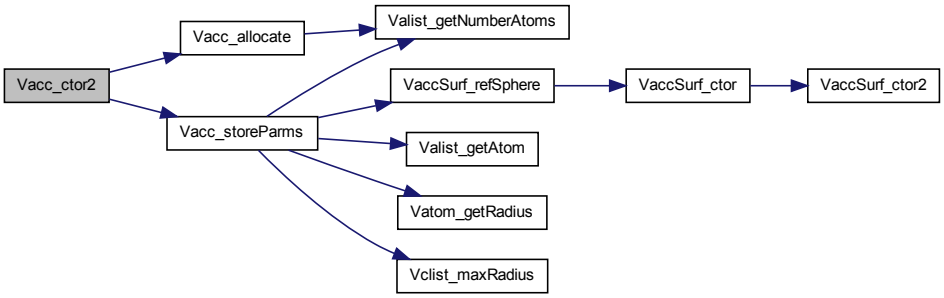
1 if successful, 0 otherwise

Parameters

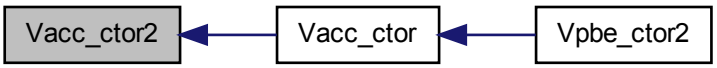
thee	Memory for Vacc objet
alist	Molecule for accessibility queries
clist	Pre-constructed cell list for looking up atoms near specific positions
surf_density	Minimum per-atom solvent accessible surface point density (in pts/A^2)

Definition at line 195 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

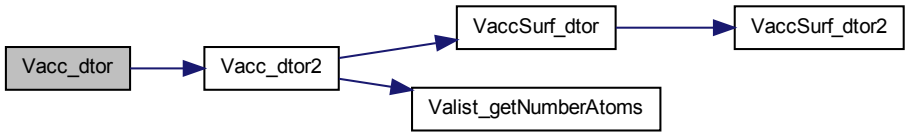
Nathan Baker

Parameters

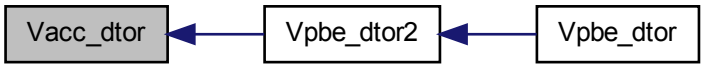
thee	Pointer to memory location of object

Definition at line 224 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

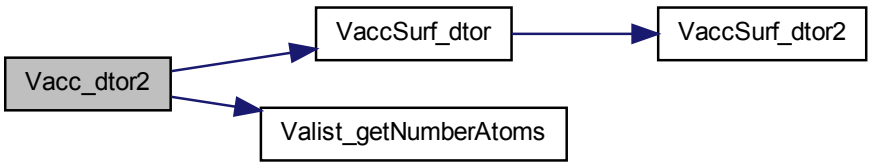
Nathan Baker

Parameters

thee	Pointer to object

Definition at line 234 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

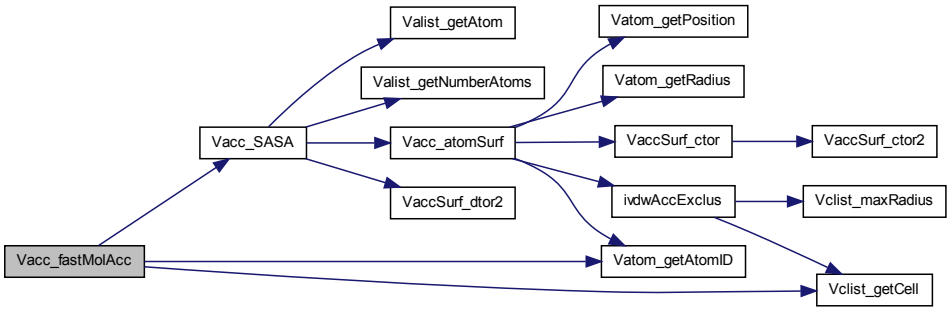
This routine has a slight bug which can generate very small internal regions of high dielectric (thanks to John Mongan and Jess Swanson for finding this)

Parameters

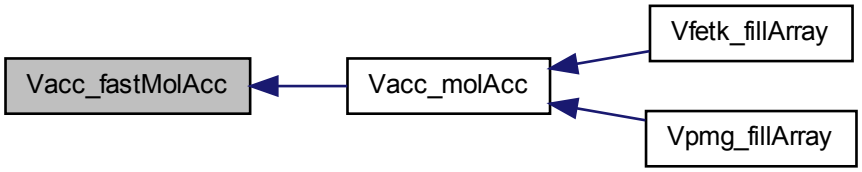
thee	Accessibility object
center	Probe center coordinates
radius	Probe radius (in Å)

Definition at line 573 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

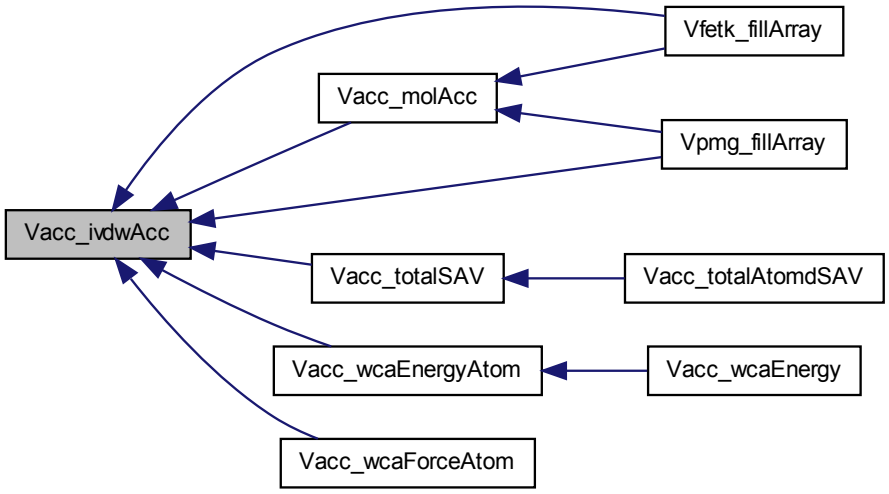
Returns

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

Parameters

thee	Accessibility object
center	Probe center coordinates
radius	Probe radius (Å)

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

thee	Object for memory check

Definition at line 61 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

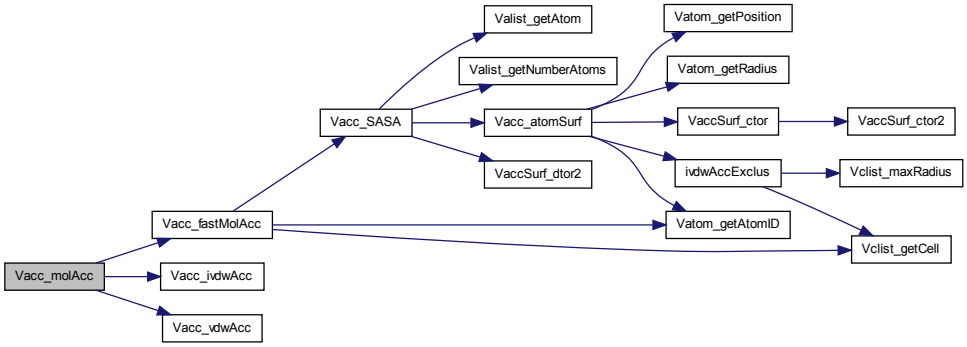
This routine has a slight bug which can generate very small internal regions of high dielectric (thanks to John Mongan and Jess Swanson for finding this)

Parameters

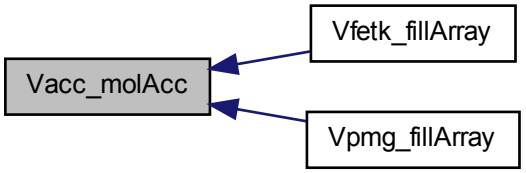
thee	Accessibility object
center	Probe center coordinates
radius	Probe radius (in Å)

Definition at line 544 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

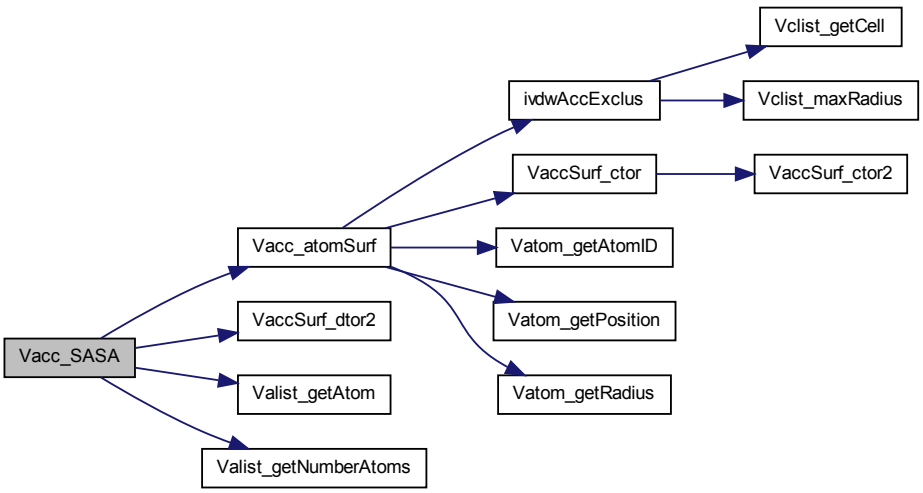
Total solvent accessible area (A^2)

Parameters

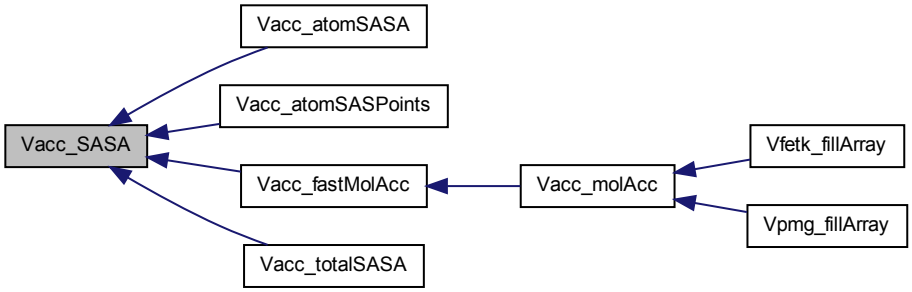
thee	Accessibility object
radius	Probe molecule radius (Å)

Definition at line 649 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 *infrad*)

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 evalation; basically a cubic spline.

Author

Nathan Baker

Returns

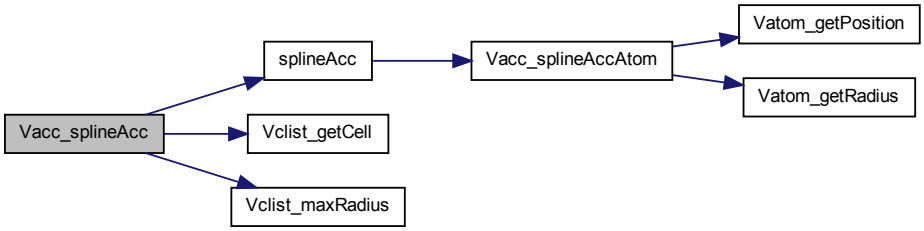
Characteristic function value between 1.0 (accessible) and 0.0 (inaccessible)

Parameters

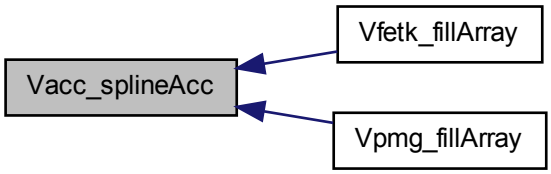
thee	Accessibility object
center	Probe center coordinates
win	Spline window (Å)
infrad	Inflation radius (Å) for ion access.

Definition at line 464 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

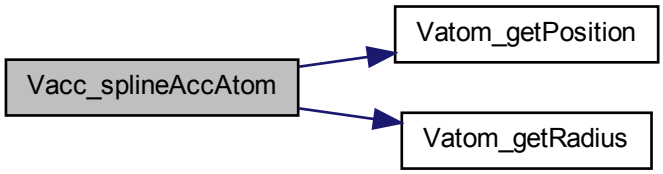
Characteristic function value between 1.0 (accessible) and 0.0 (inaccessible)

Parameters

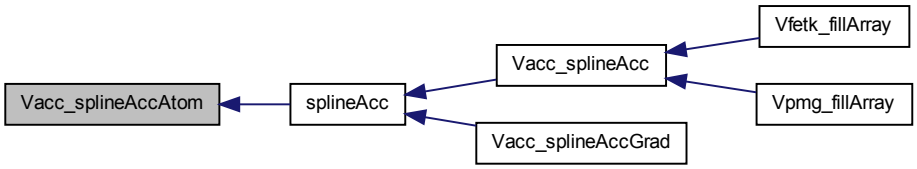
thee	Accessibility object
center	Probe center coordinates
win	Spline window (Å)
infrad	Inflation radius (Å) for ion access.
atom	Atom

Definition at line 387 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 *infrad,* double * *grad*)

Report gradient of spline-based accessibility.

Author

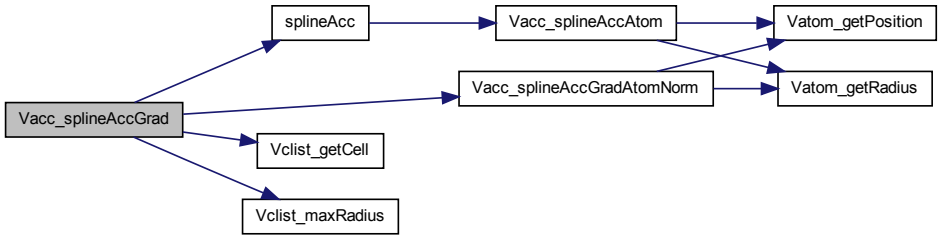
Nathan Baker

Parameters

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

Definition at line 497 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 *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)

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

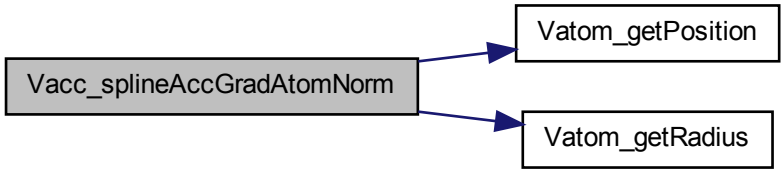
Nathan Baker

Parameters

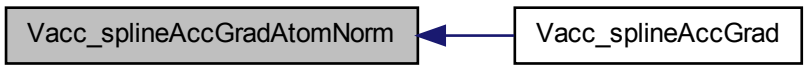
thee	Accessibility object
center	Probe center coordinates
win	Spline window (Å)
infrad	Inflation radius (Å) for ion access.
atom	Atom
force	VAPBS_DIM-vector set to gradient of accessibility

Definition at line 289 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

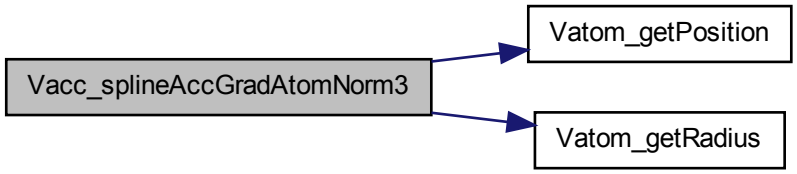
Michael Schnieders

Parameters

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

Definition at line 1040 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

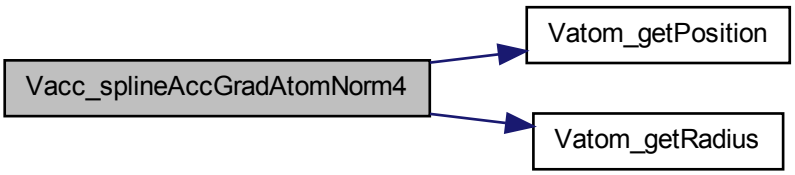
Michael Schnieders

Parameters

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

Definition at line 947 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 evalation; basically a cubic spline.

Author

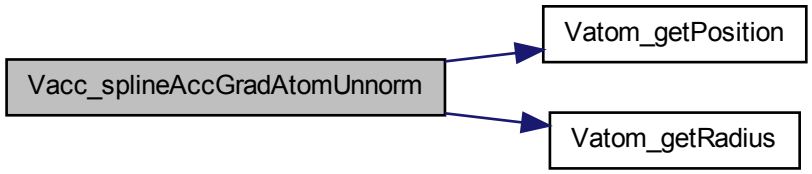
Nathan Baker

Parameters

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

Definition at line 338 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

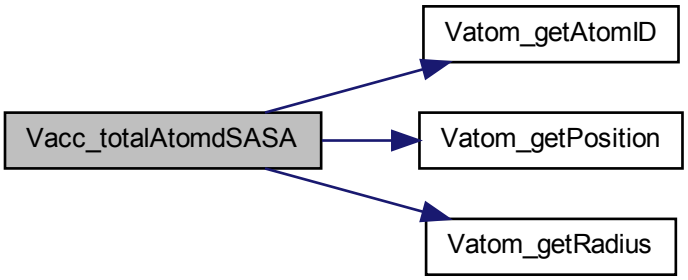
David Gohara, Nathan Baker

Parameters

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

Definition at line 1314 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

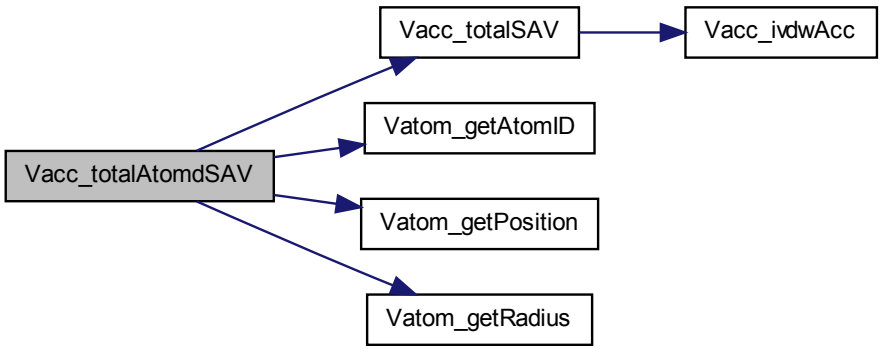
David Gohara, Nathan Baker

Parameters

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

Definition at line 1373 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

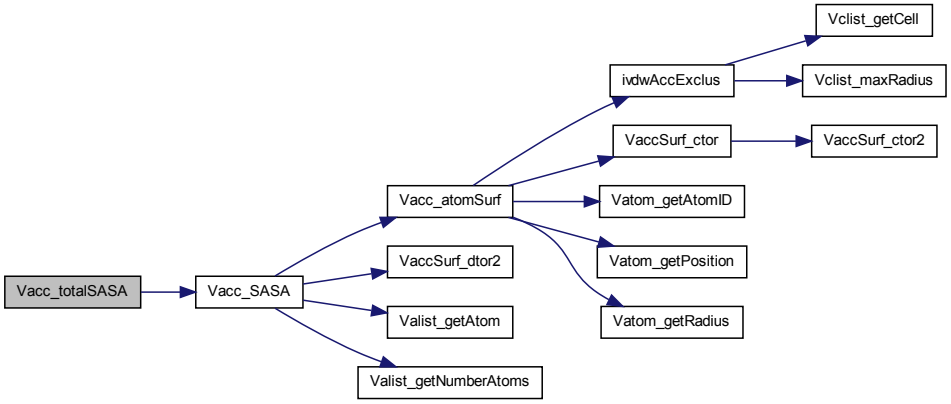
Total solvent accessible area (A^2)

Parameters

thee	Accessibility object
radius	Probe molecule radius (Å)

Definition at line 702 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 (A^3)

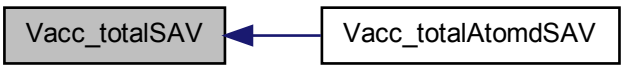
Parameters

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

Definition at line 1428 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

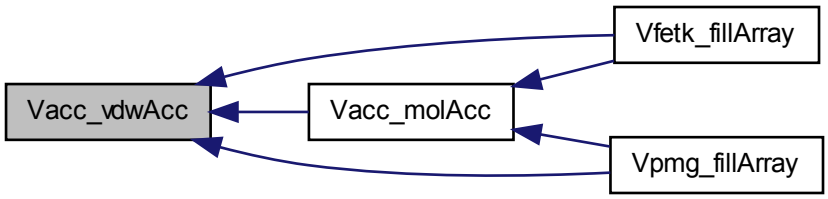
Returns

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

Parameters

thee	Accessibility object
center	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

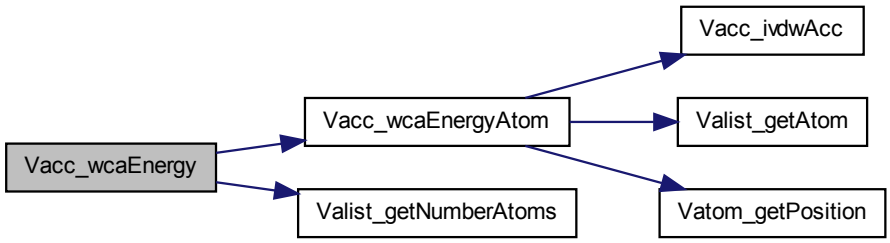
Success flag

Parameters

thee	Accessibility object
apolparm	Apolar calculation parameters
alist	Alist for acc object
clist	Clist for acc object

Definition at line 1646 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

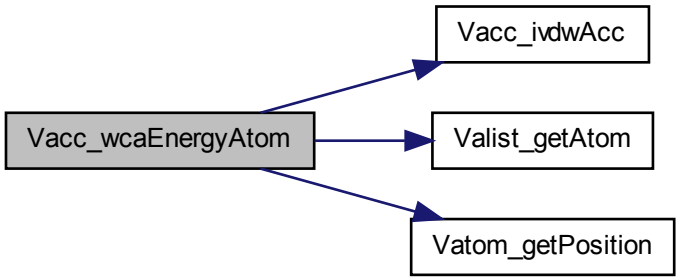
Success flag

Parameters

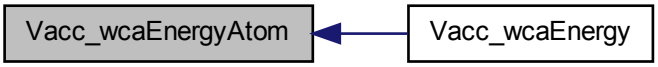
thee	Accessibility object
apolparm	Apolar calculation parameters
alist	Atom list
clist	Cell list associated with Vacc object
iatom	Index for atom of interest
value	Set to energy value

Definition at line 1505 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

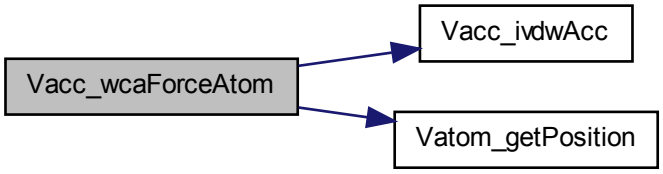
WCA energy (kJ/mol/A)

Parameters

thee	Accessibility object
apolparm	Apolar calculation parameters
clist	Clist for acc object
atom	Current atom
force	Force for atom

Definition at line 1681 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

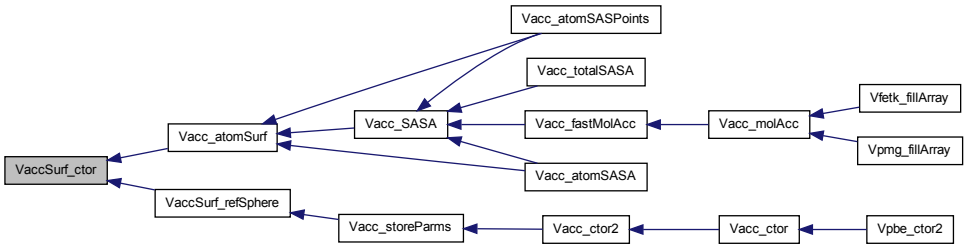
Parameters

mem	Memory manager (can be VNULL)
probe_radius	Probe radius (in A) for this surface
nsphere	Number of points in sphere

Definition at line 731 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

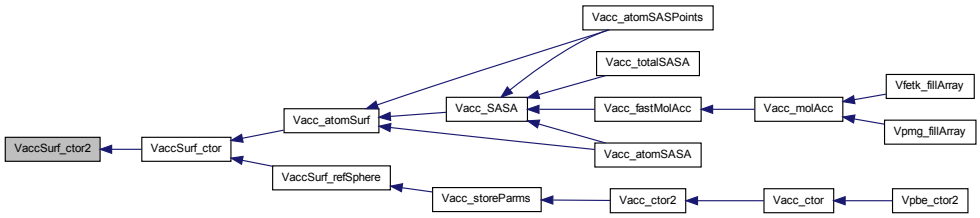
1 if successful, 0 otherwise

Parameters

thee	Allocated memory
mem	Memory manager (can be VNULL)
probe_-radius	Probe radius (in A) for this surface
nsphere	Number of points in sphere

Definition at line 746 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

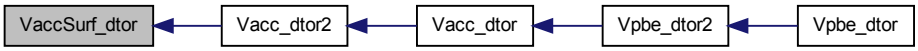
Parameters

thee	Object to be destroyed

Definition at line 777 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

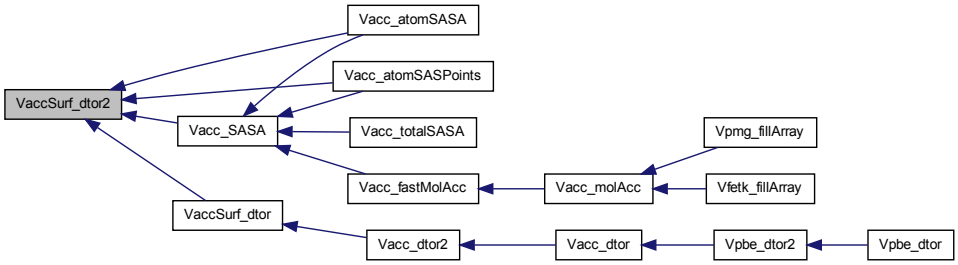
Nathan Baker

Parameters

thee	Object to be destroyed

Definition at line 791 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

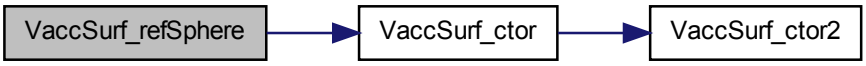
Reference sphere surface object

Parameters

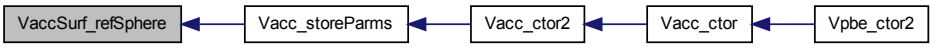
mem	Memory object
npts	Requested number of points on sphere

Definition at line 867 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 131 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

thee	Storage for new atom list

Definition at line 148 of file valist.c.

Here is the caller graph for this function:

8.10.2.3 VEXTERNC void Valist_dtor (Valist ** *thee*)

Destroys atom list object.

Author

Nathan Baker

Parameters

thee	Pointer to storage for atom list

Definition at line 160 of file valist.c.

Here is the call graph for this function:

8.10.2.4 VEXTERNC void Valist_dtor2 (Valist * *thee*)

FORTRAN stub to destroy atom list object.

Author

Nathan Baker

Parameters

thee	Pointer to atom list object

Definition at line 169 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

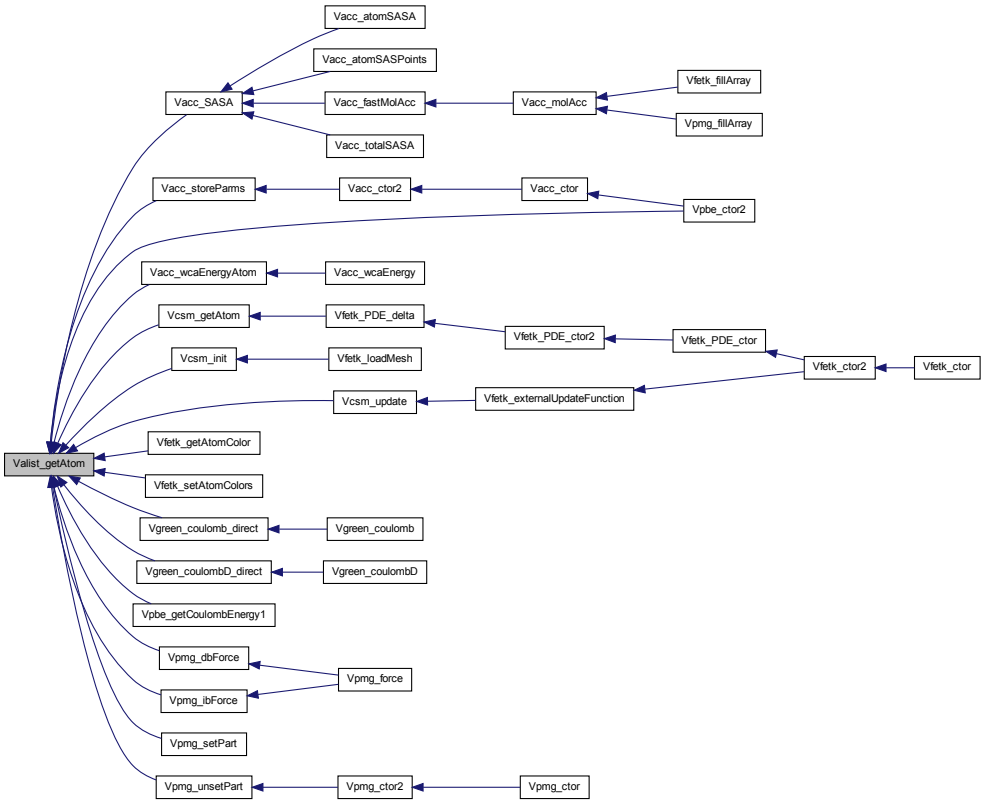
Pointer to atom object i

Parameters

thee	Atom list object
i	Index of atom in list

Definition at line 108 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

thee	Atom list object

Definition at line 88 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

thee	Atom list object

Definition at line 59 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

thee	Atom list object

Definition at line 69 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

thee	Atom list object

Definition at line 78 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

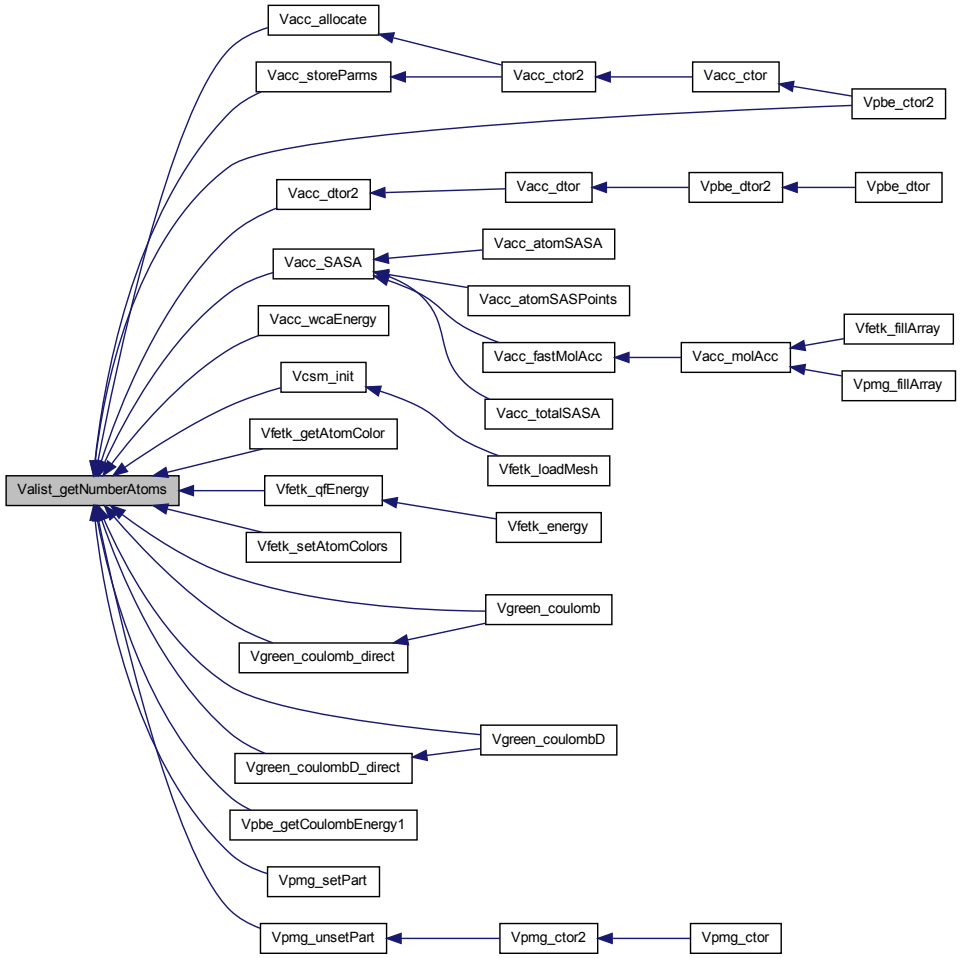
Number of atoms in list

Parameters

thee	Atom list object

Definition at line 98 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

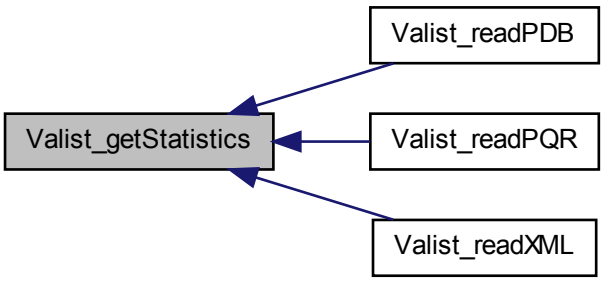
Nathan Baker, Yong Huang

Returns

Success enumeration

Definition at line 851 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

thee	Atom list object

Definition at line 122 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

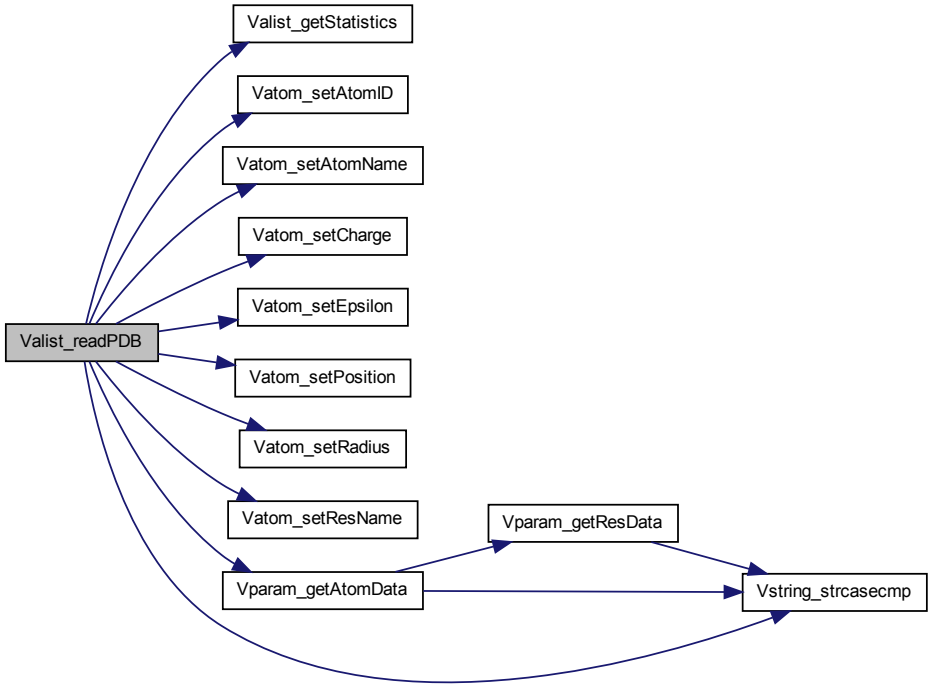
We don't actually respect PDB format; instead recognize whitespace- or tab-delimited fields which allows us to deal with structures with coordinates $>$ 999 or $<$ -999.

Parameters

thee	Atom list object
param	A pre-initialized parameter object
sock	Socket read for reading PDB file

Definition at line 508 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

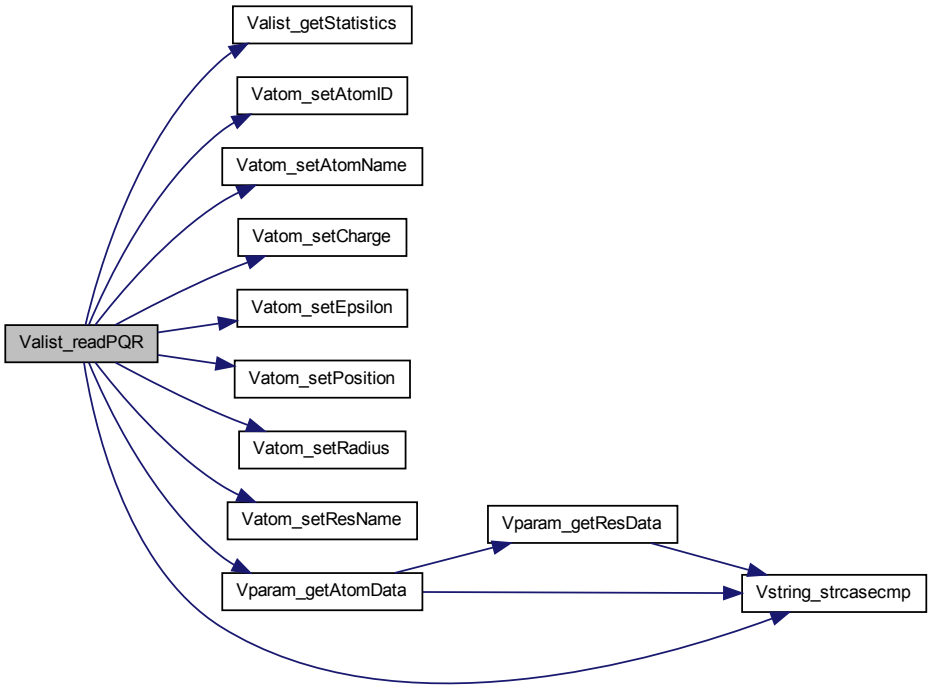
- We don't actually respect PDB format; instead recognize whitespace- or tab-delimited fields which allows us to deal with structures with coordinates $>$ 999 or $<$ -999.

Parameters

thee	Atom list object
param	A pre-initialized parameter object
sock	Socket reading for reading PQR file

Definition at line 599 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

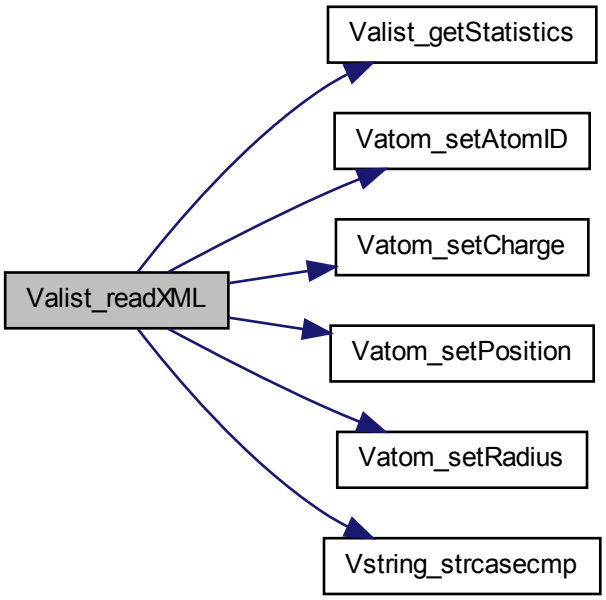
- The XML file must adhere to some guidelines, notably the presence of an <atom> tag with all other useful information (x, y, z, charge, and radius) as nested elements.

Parameters

thee	Atom list object
param	A pre-initialized parameter object
sock	Socket reading for reading PQR file

Definition at line 707 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 66 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

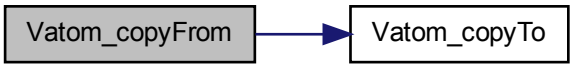
Nathan Baker

Parameters

thee	Destination for atom information
src	Source for atom information

Definition at line 167 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

thee	Source for atom information
dest	Destination for atom information

Definition at line 158 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 123 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

thee	Pointer to Vatom allocated memory location

Returns

1 if succesful, 0 otherwise

Definition at line 134 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

thee	Pointer to memory location of object to be destroyed

Definition at line 139 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

thee	Pointer to object to be destroyed

Definition at line 147 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

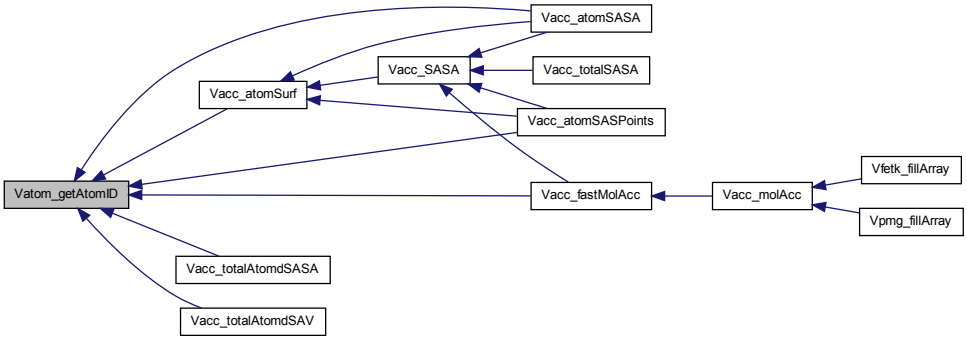
thee	Vatom object

Returns

Unique non-negative number

Definition at line 77 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

thee	Vatom object
atomName	Atom name

Definition at line 195 of file vatom.c.

8.11.3.9 VEXTERNC double Vatom_getCharge (Vatom * *thee*)

Get atomic charge.

Author

Nathan Baker

Parameters

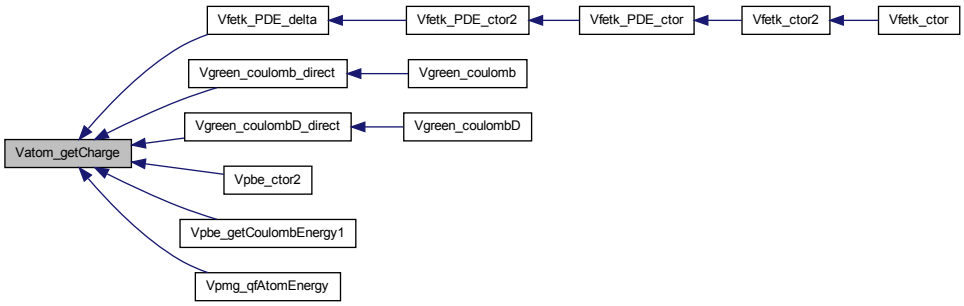
thee	Vatom object

Returns

Atom partial charge (in e)

Definition at line 112 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

thee	Vatom object

Returns

Atomic epsilon (in Å)

8.11.3.11 VEXTERNC double Vatom_getPartID (Vatom * *thee*)

Get partition ID.

Author

Nathan Baker

Parameters

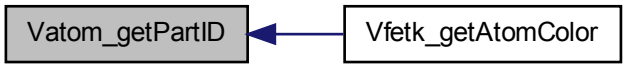
thee	Vatom object

Returns

Partition ID; a negative value means this atom is not assigned to any partition

Definition at line 63 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

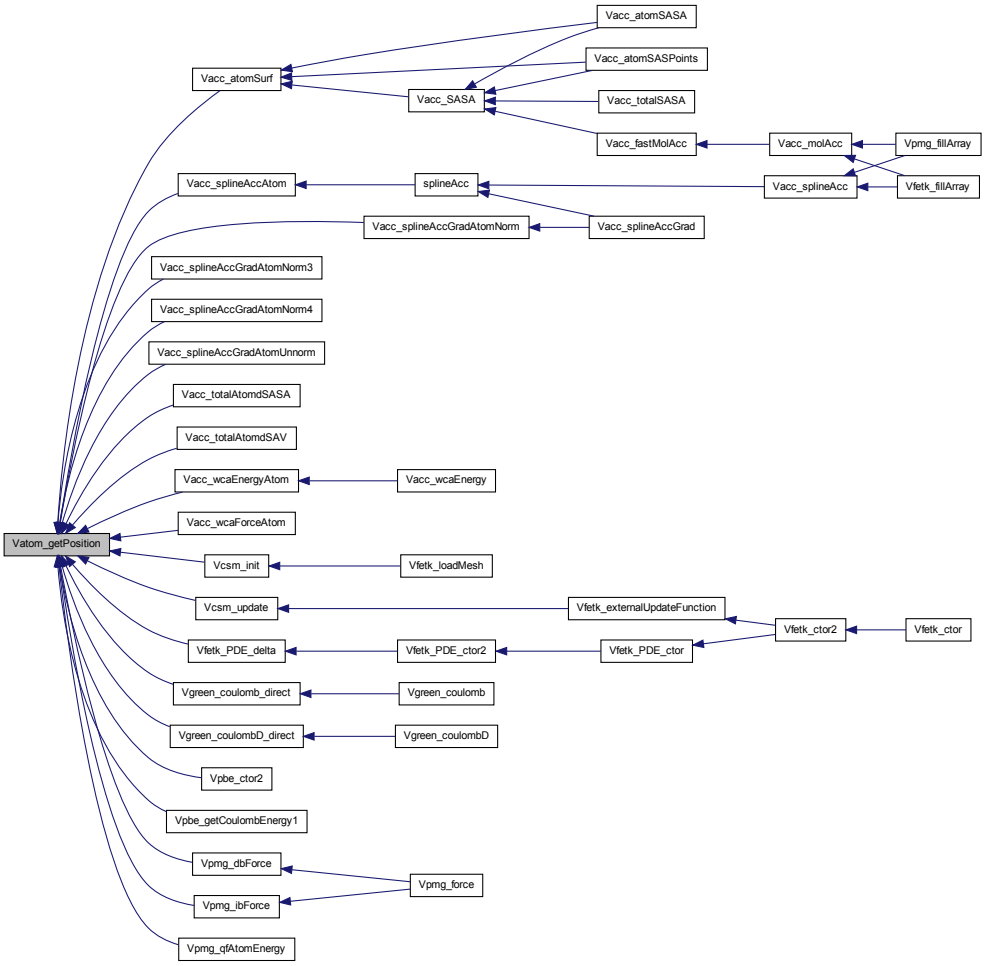
thee	Vatom object

Returns

Pointer to 3*double array of atomic coordinates (in Å)

Definition at line 56 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

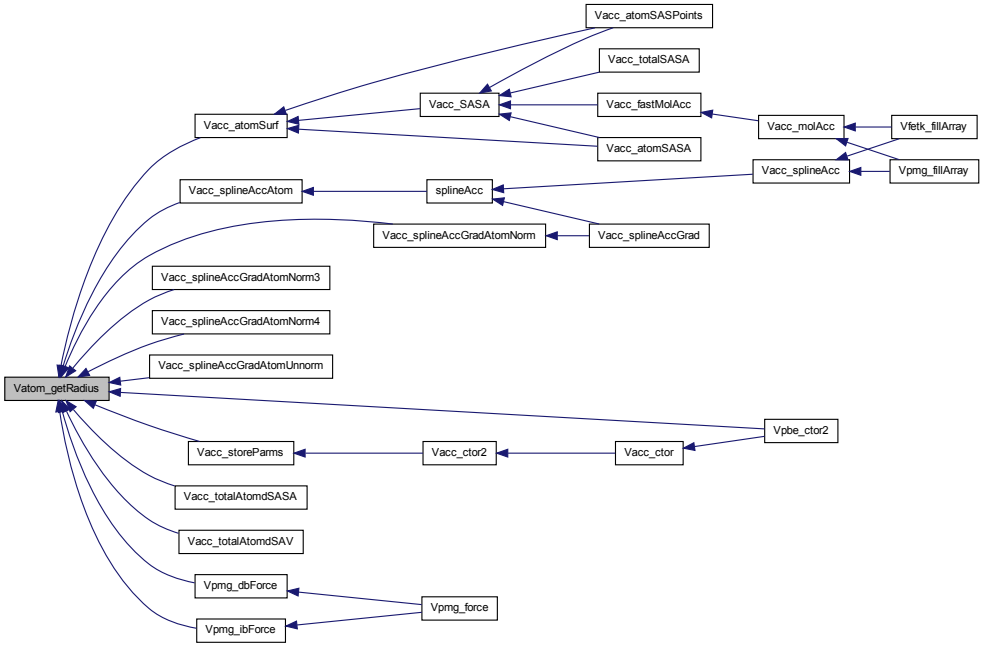
thee	Vatom object

Returns

Atomic radius (in Å)

Definition at line 98 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

thee	Vatom object
resName	Residue Name

Definition at line 180 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

thee	Vpmg object

Returns

The memory used by this structure and its contents in bytes

Definition at line 119 of file vatom.c.

8.11.3.16 VEXTERNC void Vatom_setAtomID (Vatom * *thee,* int *id*)

Set atom ID.

Author

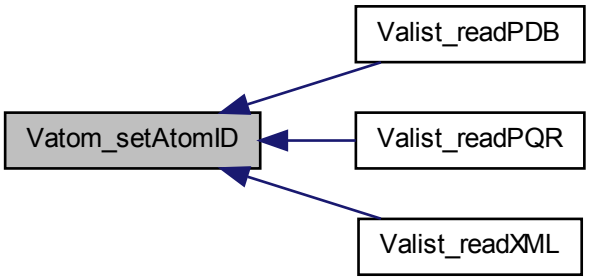
Nathan Baker

Parameters

thee	Vatom object
id	Unique non-negative number

Definition at line 84 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

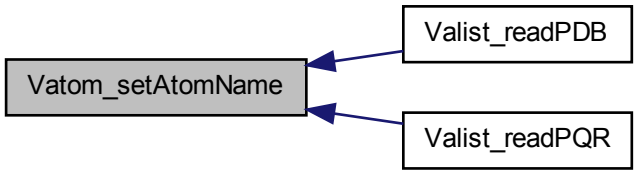
Jason Wagoner

Parameters

thee	Vatom object
atomName	Atom name

Definition at line 188 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

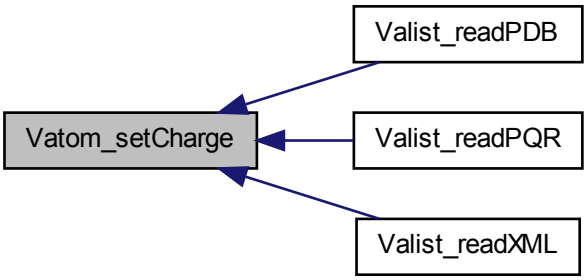
Nathan Baker

Parameters

thee	Vatom object
charge	Atom partial charge (in e)

Definition at line 105 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.

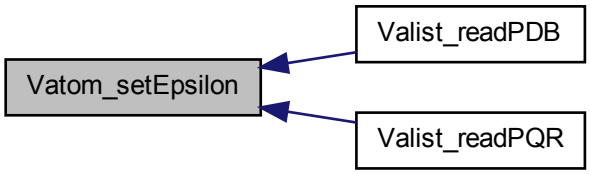
Author

David Gohara

Parameters

thee	Vatom object
epsilon	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

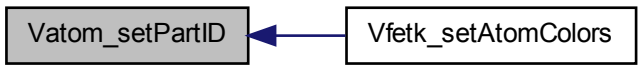
Nathan Baker

Parameters

thee	Vatom object
partID	Partition ID; a negative value means this atom is not assigned to any partition

Definition at line 70 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

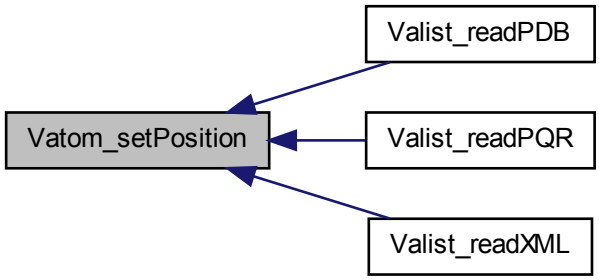
Nathan Baker

Parameters

thee	Vatom object to be modified
position	Coordinates (in Å)

Definition at line 149 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

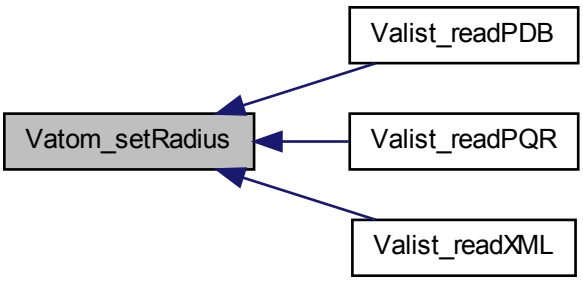
Nathan Baker

Parameters

thee	Vatom object
radius	Atomic radius (in Å)

Definition at line 91 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

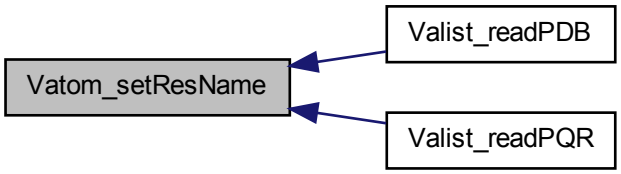
Jason Wagoner

Parameters

thee	Vatom object
resName	Residue Name

Definition at line 173 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 84 of file vcap.c.

8.12.2.2 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

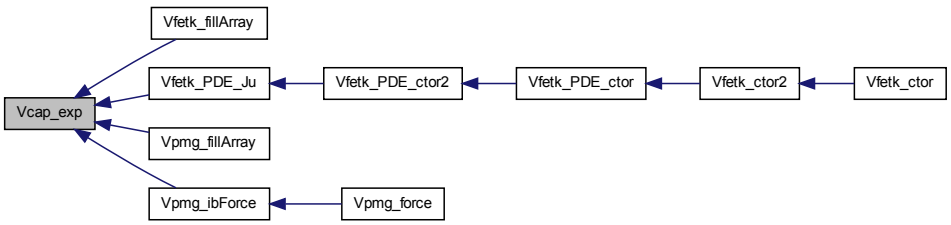
exp(x) or capped equivalent

Parameters

x	Argument to exp()
ichop	Set to 1 if function capped, 0 otherwise

Definition at line 52 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 68 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 71 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

alist	Molecule for cell list queries
max_radius	Max probe radius (Å) to be queried
npts	Number of in hash table points in each direction
mode	Mode to construct table
lower_corner	Hash table lower corner for manual construction (see mode variable); ignored otherwise
upper_corner	Hash table upper corner for manual construction (see mode variable); ignored otherwise

Definition at line 72 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

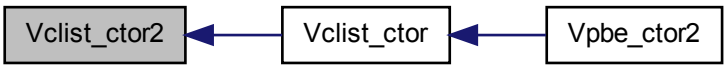
Parameters

thee	Memory for Vclist objet
alist	Molecule for cell list queries
max_radius	Max probe radius (Å) to be queried
npts	Number of in hash table points in each direction

mode	Mode to construct table
lower_corner	Hash table lower corner for manual construction (see mode variable); ignored otherwise
upper_corner	Hash table upper corner for manual construction (see mode variable); ignored otherwise

Definition at line 340 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

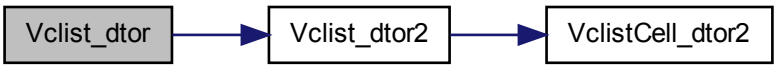
Nathan Baker

Parameters

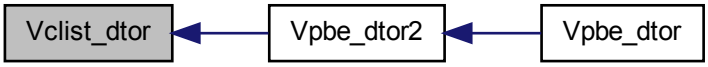
thee	Pointer to memory location of object

Definition at line 394 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

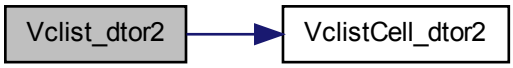
Nathan Baker

Parameters

thee	Pointer to object

Definition at line 405 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

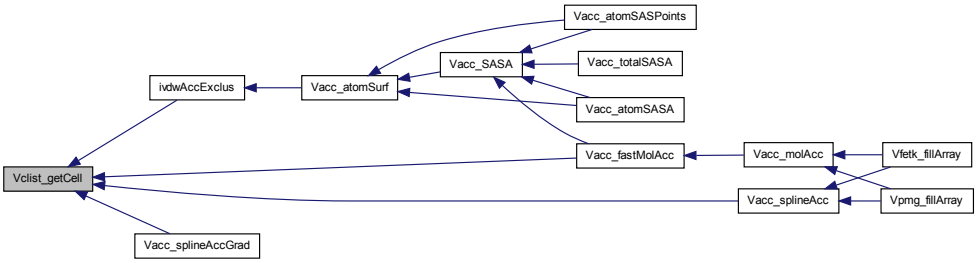
Pointer to VclistCell object or VNULL if no cell available (away from molecule).

Parameters

thee	Pointer to Vclist cell list
position	Position to evaluate

Definition at line 420 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

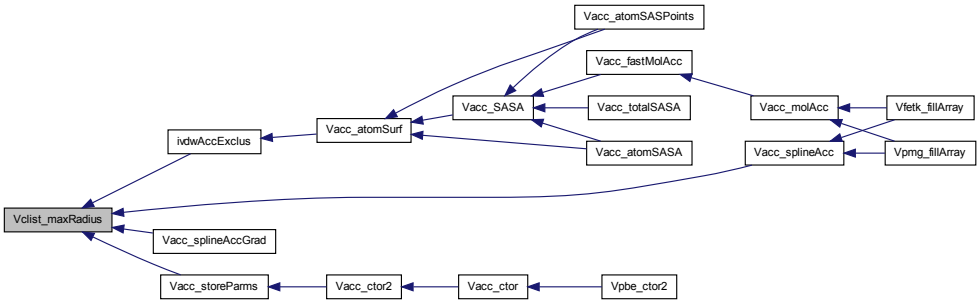
Max probe radius (in A)

Parameters

thee	Cell list object

Definition at line 65 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

thee	Object for memory check

Definition at line 60 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

natoms	Number of atoms associated with this cell

Definition at line 446 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

thee	Memory location for object
natoms	Number of atoms associated with this cell

Definition at line 458 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

thee	Pointer to memory location of object

Definition at line 480 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

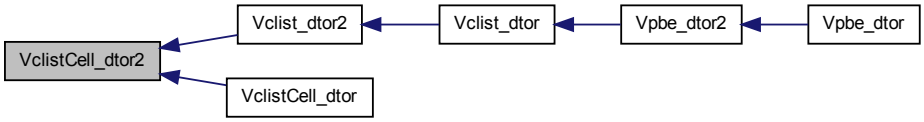
Nathan Baker

Parameters

thee	Pointer to object

Definition at line 491 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@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
* 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 Washington University in St. Louis 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

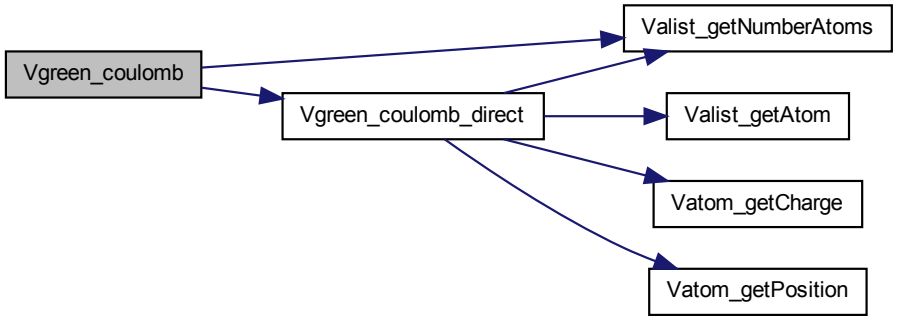
thee	Vgreen object
npos	The number of positions to evaluate
x	The npos x-coordinates
y	The npos y-coordinates
z	The npos z-coordinates
val	The npos values

Returns

1 if successful, 0 otherwise

Definition at line 251 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

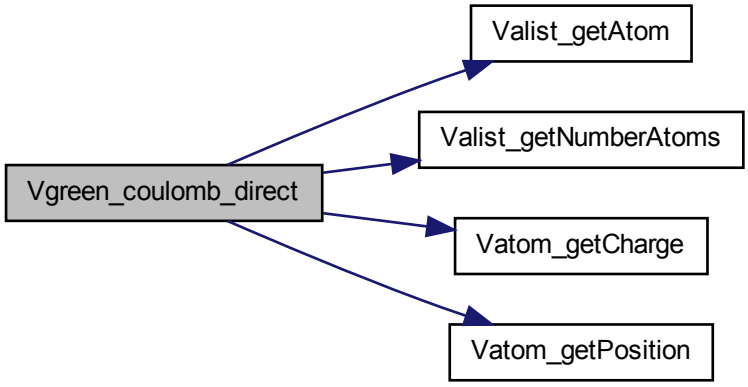
thee	Vgreen object
npos	The number of positions to evaluate
x	The npos x-coordinates
y	The npos y-coordinates
z	The npos z-coordinates
val	The npos values

Returns

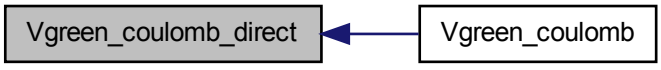
1 if successful, 0 otherwise

Definition at line 217 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. Johnston/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

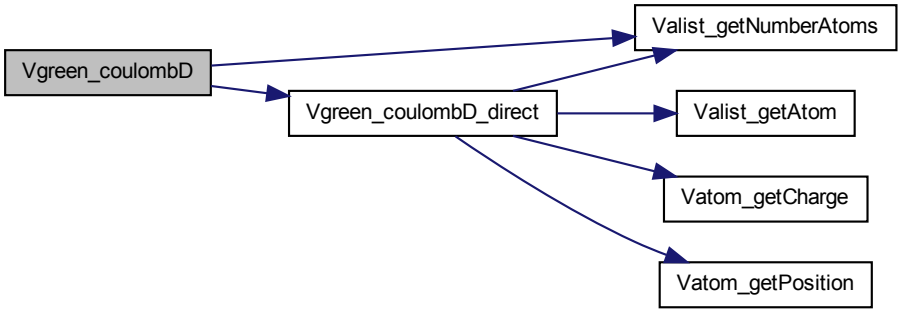
thee	Vgreen object
npos	The number of positions to evaluate
x	The npos x-coordinates
y	The npos y-coordinates
z	The npos z-coordinates
pot	The npos potential values
gradx	The npos gradient x-components
grady	The npos gradient y-components
gradz	The npos gradient z-components

Returns

1 if successful, 0 otherwise

Definition at line 355 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

thee	Vgreen object
npos	The number of positions to evaluate
x	The npos x-coordinates
y	The npos y-coordinates
z	The npos z-coordinates
pot	The npos potential values

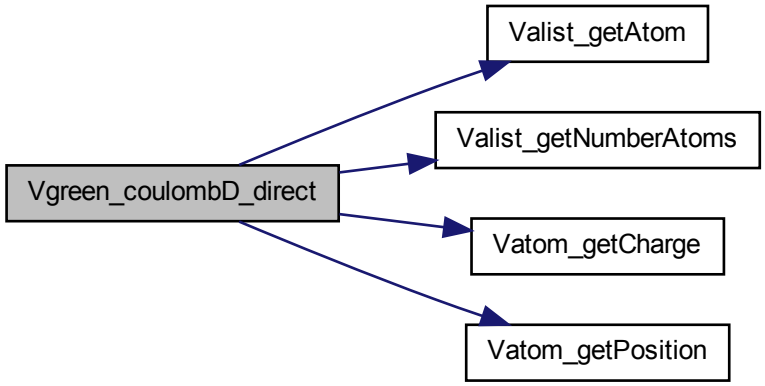
gradx	The npos gradient x-components
grady	The npos gradient y-components
gradz	The npos gradient z-components

Returns

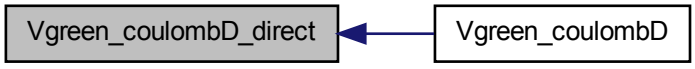
1 if successful, 0 otherwise

Definition at line 303 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

alist	Atom (charge) list associated with object

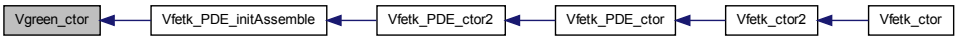
Returns

Pointer to newly allocated Green's function oracle

Definition at line 149 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

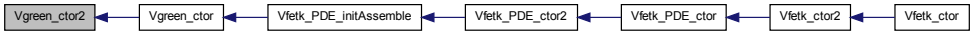
thee	Pointer to memory allocated for object
alist	Atom (charge) list associated with object

Returns

1 if successful, 0 otherwise

Definition at line 160 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

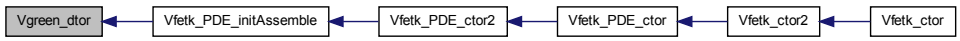
Parameters

thee	Pointer to memory location for object

Definition at line 185 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

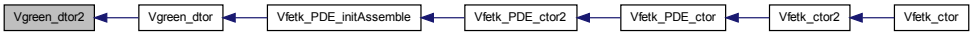
Nathan Baker

Parameters

thee	Pointer to object

Definition at line 193 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

thee	Vgreen object

Returns

Pointer to Valist object associated with this Green's function object

Definition at line 135 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 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

thee	Vgreen object
npos	Number of positions to evaluate
x	The npos x-coordinates
y	The npos y-coordinates
z	The npos z-coordinates
val	The npos values
kappa	The value of κ (see above)

Returns

1 if successful, 0 otherwise

Definition at line 202 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

thee	Vgreen object
npos	The number of positions to evaluate
x	The npos x-coordinates
y	The npos y-coordinates
z	The npos z-coordinates
gradx	The npos gradient x-components
grady	The npos gradient y-components
gradz	The npos gradient z-components
kappa	The value of κ (see above)

Returns

int 1 if sucessful, 0 otherwise

Definition at line 209 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

thee	Vgreen object

Returns

The memory used by this structure and its contents in bytes

Definition at line 142 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

  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

  Maximum number of points on a sphere.

- #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-zer0 (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**,

  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_TRIL = 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_AVS = 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 412 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 436 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 442 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 418 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 430 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 406 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 424 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 561 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 552 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 206 of file vhal.h.

8.15.3.2 enum eVchrg_Meth

Types of charge discretization methods.

Author

Nathan Baker

Enumerator:

VCM_TRIL 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 229 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 250 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_AVS AVS UCD format

VDF_MCSF FEtk MC Simplex Format (MCSF)

VDF_GZ Binary file (GZip)

VDF_FLAT Write flat file

Definition at line 308 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 $\varepsilon(\nabla u)^2$, 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 y-shifted map as calculated with the currently specified scheme (dimensionless)

VDT_KAPPA Kappa map as calculated with the currently specified scheme ($^{-3}$)

Definition at line 268 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 156 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 138 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 190 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 65 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 80 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 101 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\r<>"

  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

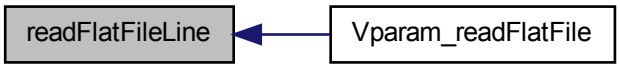
sock	Socket ready for reading
atom	Atom to hold parsed data

Returns

1 if successful, 0 otherwise

Definition at line 688 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

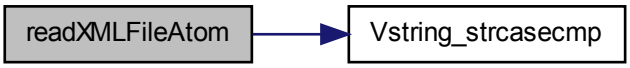
sock	Socket ready for reading
atom	Atom to hold parsed data

Returns

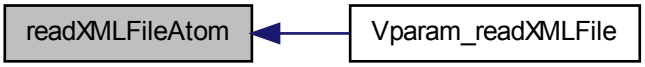
1 if successful, 0 otherwise

Definition at line 607 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

thee	Pointer to destination object
src	Pointer to source object

Definition at line 604 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

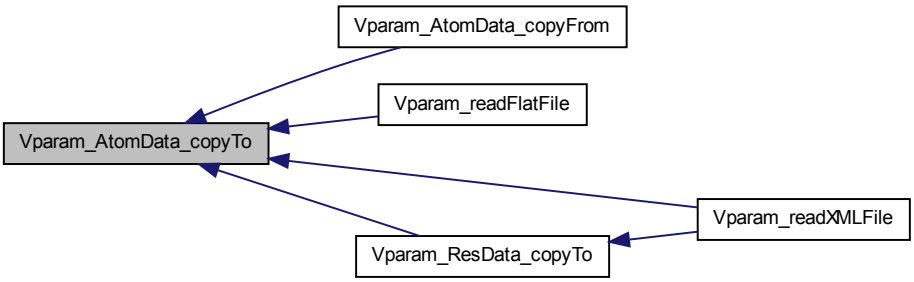
Nathan Baker

Parameters

thee	Pointer to source object
dest	Pointer to destination object

Definition at line 568 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 106 of file vparam.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

thee	Allocated memory

Returns

1 if successful, 0 otherwise

Definition at line 118 of file vparam.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

thee	Pointer to memory location of object

Definition at line 120 of file vparam.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

thee	Pointer to object

Definition at line 130 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 178 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

thee	Allocated Vparam memory

Returns

1 if successful, 0 otherwise

Definition at line 190 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

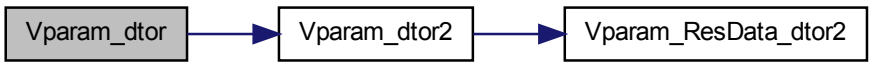
Nathan Baker

Parameters

thee	Pointer to memory location of object

Definition at line 210 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

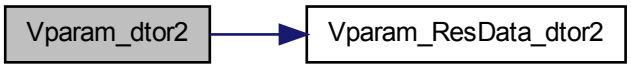
Nathan Baker

Parameters

thee	Pointer to object

Definition at line 220 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

thee	Vparam object
resName	Residue name
atomName	Atom name

Returns

Pointer to the desired atom object or VNULL if residue not found

Note

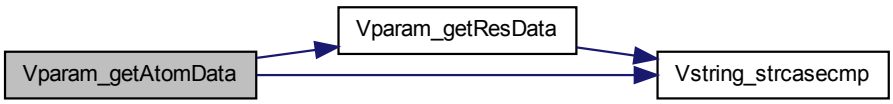
Some method to initialize the database must be called before this method (e.g.,

See also

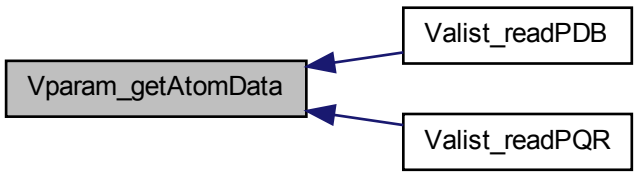
Vparam_readFlatFile)

Definition at line 264 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

thee	Vparam object
resName	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.,

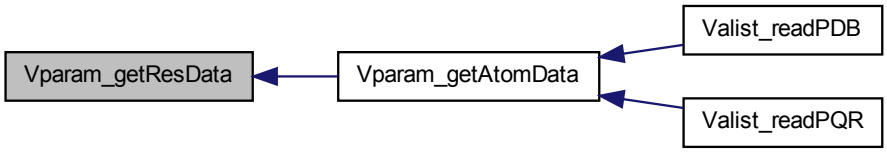
See also

Vparam_readFlatFile)

Definition at line 238 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

thee	Vparam object

Returns

Number of bytes allocated for object

Definition at line 99 of file vparam.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

thee	Vparam object
iodev	Input device type (FILE/BUFF/UNIX/INET)

iofmt	Input device format (ASCII/XDR)
thost	Input hostname (for sockets)
fname	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.

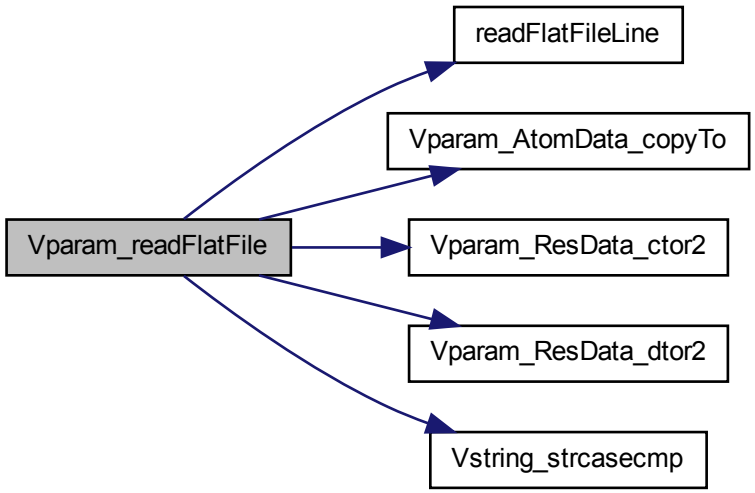
ASCII-format flat files are provided with the APBS source code:

tools/conversion/vparam-amber-parm94.dat AMBER parm94 parameters

tools/conversion/vparam-charmm-par_all27.dat CHARMM par_all27_prot_na parameters

Definition at line 442 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

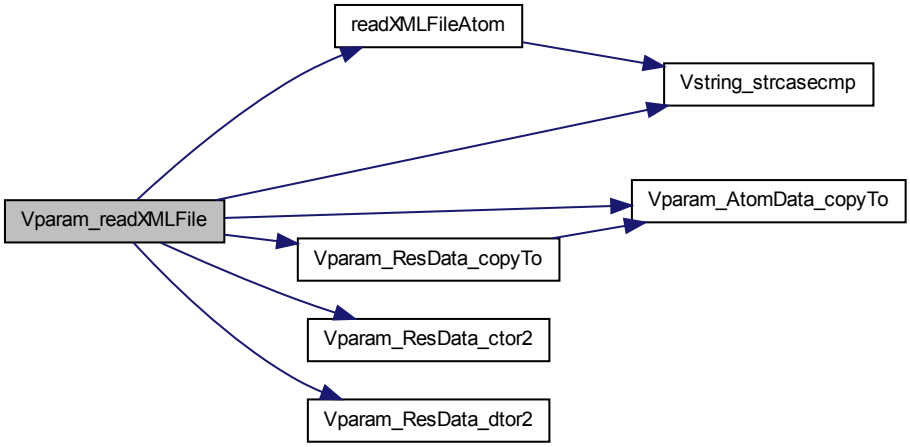
thee	Vparam object
iodev	Input device type (FILE/BUFF/UNIX/INET)
iofmt	Input device format (ASCII/XDR)
thost	Input hostname (for sockets)
fname	Input FILE/BUFF/UNIX/INET name

Returns

1 if successful, 0 otherwise

Definition at line 303 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

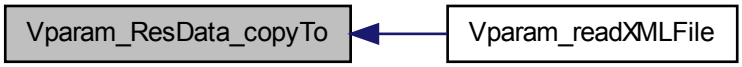
Parameters

thee	Pointer to source object
dest	Pointer to destination object

Definition at line 582 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

mem	Memory object of Vparam master class

Returns

Newly allocated object

Definition at line 132 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

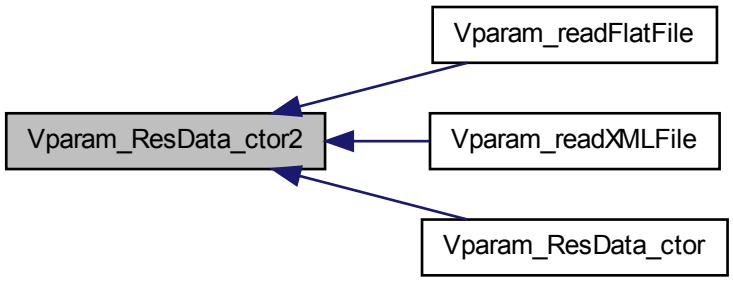
thee	Allocated memory
mem	Memory object of Vparam master class

Returns

1 if successful, 0 otherwise

Definition at line 144 of file vparam.c.

Here is the caller graph for this function:



8.16.2.21 VEXTERNC void Vparam_ResData_dtor (Vparam_ResData ** *thee*)

Destroy object.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of object

Definition at line 157 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

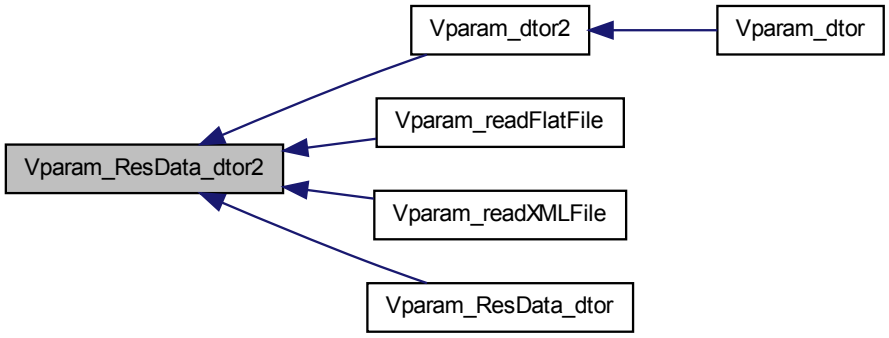
Nathan Baker

Parameters

thee	Pointer to object

Definition at line 167 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)

  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)
  aauthor 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:

$$\kappa^2 = \frac{8\pi N_A e_c^2 I_s}{1000\varepsilon_w k_B T}$$

where the units are esu*esu/erg/mol. To obtain $^{-2}$, we multiply by 10^{-16}. Thus, in $^{-2}$, where k_B and e_c are in gaussian rather than mks units, the proper value for kappa is:

$$\kappa^2 = \frac{8\pi N_A e_c^2 I_s}{1000\varepsilon_w k_b T} \times 10^{-16}$$

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

alist	Atom list
ionNum	Number of counterion species
ionConc	Array containing counterion concentrations (M)
ionRadii	Array containing counterion radii (A)
ionQ	Array containing counterion charges (e)
T	Temperature for Boltzmann distribution (K)
soluteDiel	Solute internal dielectric constant
solventDiel	Solvent dielectric constant
solventRadius	Solvent probe radius for surfaces that use it (A)

focusFlag	1 if focusing operation, 0 otherwise
sdens	Vacc sphere density
z_mem	Membrane location (A)
L	Membrane thickness (A)
membraneDiel	Membrane dielectric constant
V	Transmembrane potential (V)

Definition at line 239 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:

$$\kappa^2 = \frac{8\pi N_A e_c^2 I_s}{1000 eps_w k_B T}$$

where the units are esu*esu/erg/mol. To obtain $^{-2}$, we multiply by 10^{-16}. Thus, in $^{-2}$, where k_B and e_c are in gaussian rather than mks units, the proper value for kappa is:

$$\kappa^2 = \frac{8 pi N_A e_c^2 I_s}{1000 eps_w k_b T} \times 10^{-16}$$

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

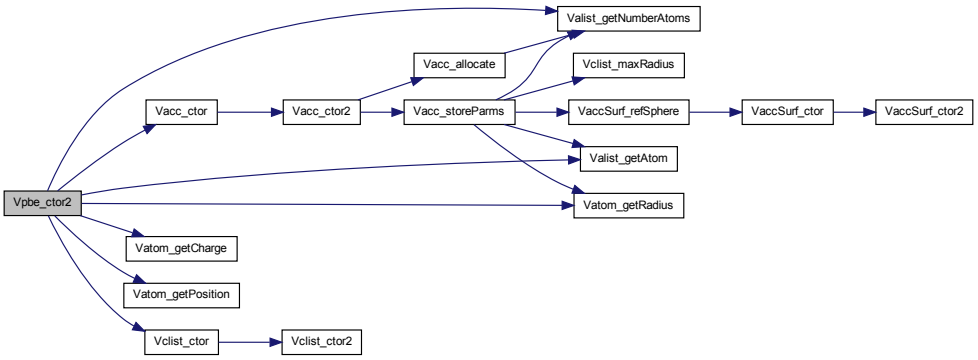
1 if successful, 0 otherwise

Parameters

thee	Pointer to memory allocated for Vpbe object
alist	Atom list
ionNum	Number of counterion species
ionConc	Array containing counterion concentrations (M)
ionRadii	Array containing counterion radii (A)
ionQ	Array containing counterion charges (e)
T	Temperature for Boltzmann distribution (K)
soluteDiel	Solute internal dielectric constant
solventDiel	Solvent dielectric constant
solventRadius	Solvent probe radius for surfaces that use it (A)
focusFlag	1 if focusing operation, 0 otherwise
sdens	Vacc sphere density
z_mem	Membrane location (A)
L	Membrane thickness (A)
membraneDiel	Membrane dielectric constant
V	Transmembrane potential (V)

Definition at line 257 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

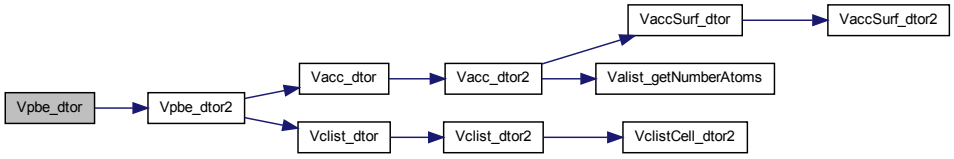
Nathan Baker

Parameters

thee	Pointer to memory location of object to be destroyed

Definition at line 460 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

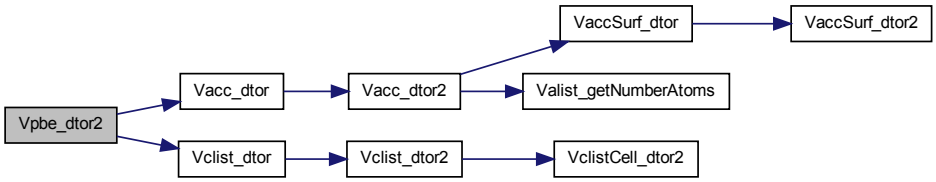
Nathan Baker

Parameters

thee	Pointer to object to be destroyed

Definition at line 468 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

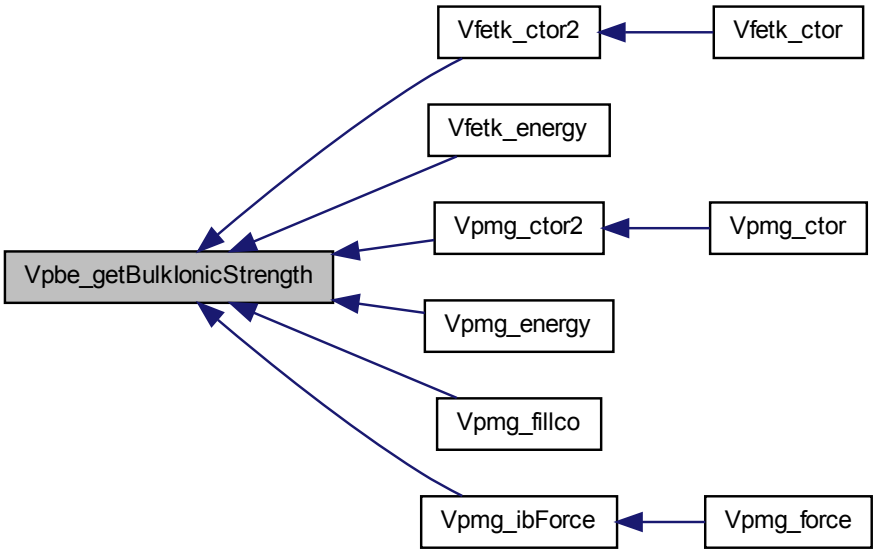
thee	Vpbe object

Returns

Bulk ionic strength (M)

Definition at line 77 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

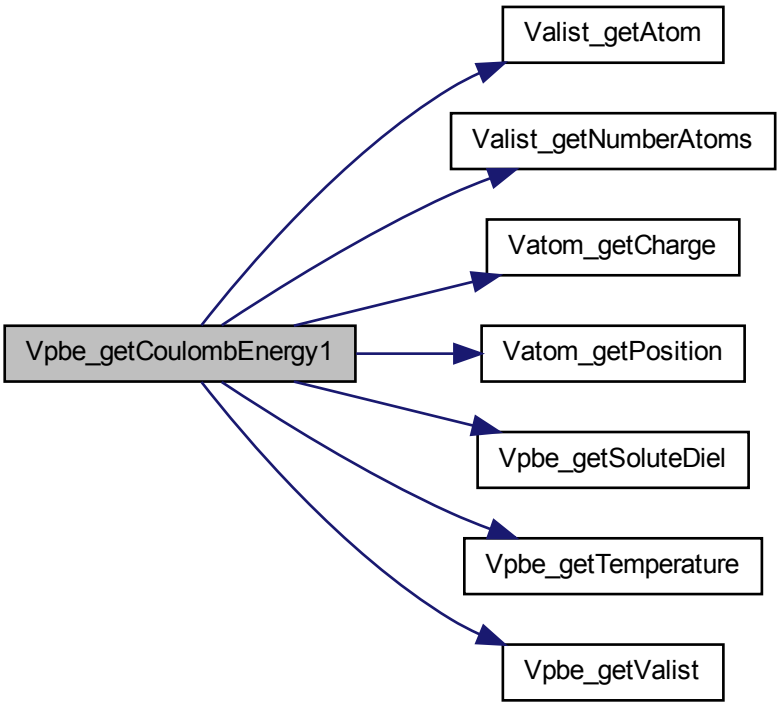
thee	Vpbe object

Returns

Coulombic energy in units of $k_B T$.

Definition at line 474 of file vpbe.c.

Here is the call graph for this function:



8.17.2.7 VEXTERNC double Vpbe_getDeblen (Vpbe * *thee*)

Get Debye-Huckel screening length.

Author

Nathan Baker

Parameters

thee	Vpbe object

Returns

Debye-Huckel screening length (Å)

Definition at line 134 of file vpbe.c.

8.17.2.8 VEXTERNC double Vpbe_getGamma (Vpbe * *thee*)

Get apolar coefficient.

Author

Nathan Baker

Parameters

thee	Vpbe object

Returns

Apolar coefficent (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

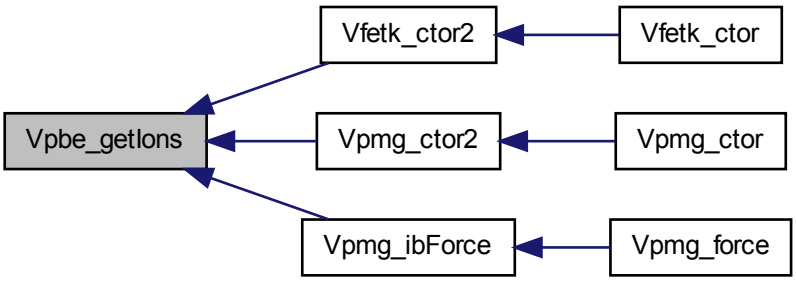
thee	Pointer to Vpbe object
nion	Set to the number of counterion species
ionConc	Array to store counterion species' concentrations (M)
ionRadii	Array to store counterion species' radii (A)
ionQ	Array to store counterion species' charges (e)

Returns

Number of ions

Definition at line 528 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

thee	Vpbe object

Returns

Length of the membrane (A)

Definition at line 202 of file vpbe.c.

8.17.2.11 VEXTERNC double Vpbe_getMaxIonRadius (Vpbe * *thee*)

Get maximum radius of ion species.

Author

Nathan Baker

Parameters

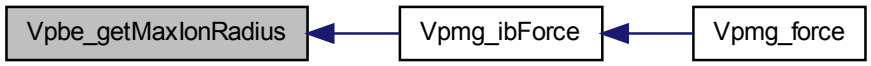
thee	Vpbe object

Returns

Maximum radius (A)

Definition at line 120 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

thee	Vpbe object

Returns

Membrane dielectric constant

Definition at line 214 of file vpbe.c.

8.17.2.13 VEXTERNC double Vpbe_getmemv (Vpbe * *thee*)

Get membrane potential (kT)

Author

Michael Grabe

Parameters

thee	Vpbe object

Definition at line 226 of file vpbe.c.

8.17.2.14 VEXTERNC double* Vpbe_getSoluteCenter (Vpbe * *thee*)

Get coordinates of solute center.

Author

Nathan Baker

Parameters

thee	Vpbe object

Returns

Pointer to 3*double array with solute center coordinates (A)

Definition at line 100 of file vpbe.c.

8.17.2.15 VEXTERNC double Vpbe_getSoluteCharge (Vpbe * *thee*)

Get total solute charge.

Author

Nathan Baker

Parameters

thee	Vpbe object

Returns

Total solute charge (e)

Definition at line 179 of file vpbe.c.

8.17.2.16 VEXTERNC double Vpbe_getSoluteDiel (Vpbe * *thee*)

Get solute dielectric constant.

Author

Nathan Baker

Parameters

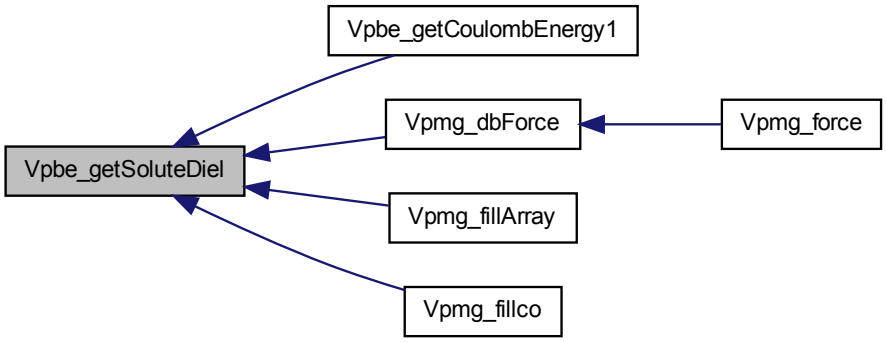
thee	Vpbe object

Returns

Solute dielectric constant

Definition at line 92 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

thee	Vpbe object

Returns

Sphere radius which bounds biomolecule (A)

Definition at line 155 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

thee	Vpbe object

Returns

Length of solute in x dimension (A)

Definition at line 161 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

thee	Vpbe object

Returns

Length of solute in y dimension (A)

Definition at line 167 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

thee	Vpbe object

Returns

Length of solute in z dimension (A)

Definition at line 173 of file vpbe.c.

8.17.2.21 VEXTERNC double Vpbe_getSolventDiel (Vpbe * *thee*)

Get solvent dielectric constant.

Author

Nathan Baker

Parameters

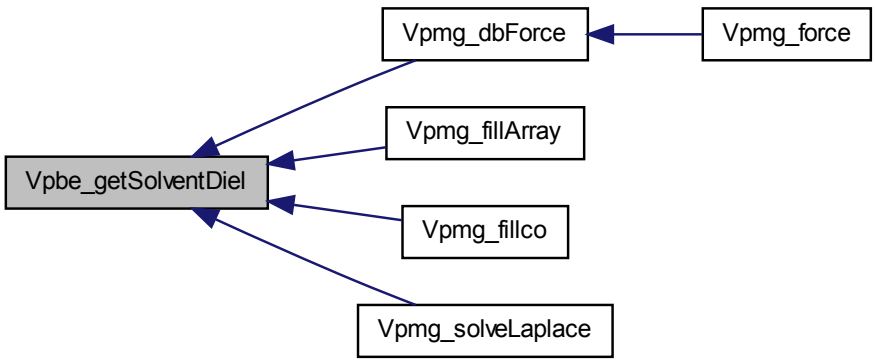
thee	Vpbe object

Returns

Solvent dielectric constant

Definition at line 106 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

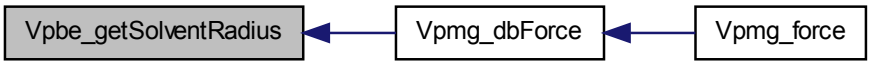
thee	Vpbe object

Returns

Solvent molecule radius (A)

Definition at line 113 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

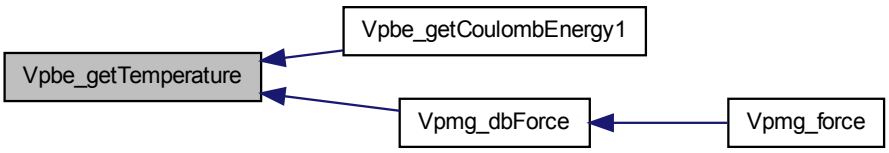
thee	Vpbe object

Returns

Temperature (K)

Definition at line 84 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

thee	Vpbe object

Returns

Pointer to internal Vacc object

Definition at line 69 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

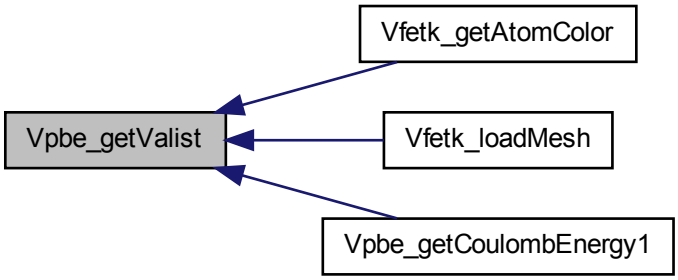
thee	Vpbe object

Returns

Pointer to internal Valist object

Definition at line 62 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

thee	Vpbe object

Returns

Bulk Debye-Huckel parameter (Å)

Definition at line 127 of file vpbe.c.

8.17.2.27 VEXTERNC double Vpbe_getZkappa2 (Vpbe * *thee*)

Get modified squared Debye-Huckel parameter.

Author

Nathan Baker

Parameters

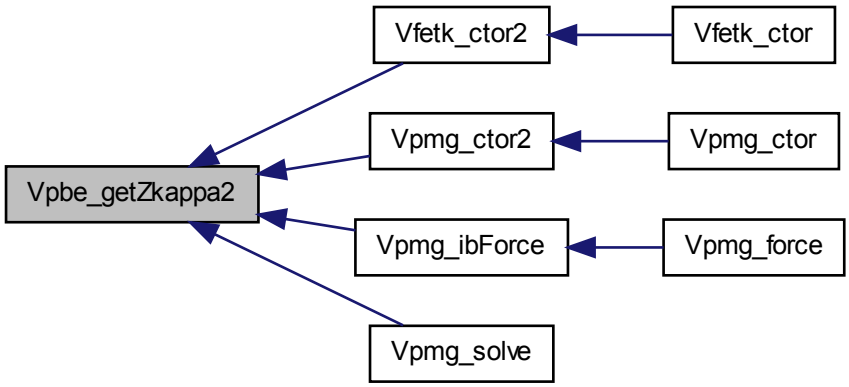
thee	Vpbe object

Returns

Modified squared Debye-Huckel parameter ($^{-2}$)

Definition at line 141 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

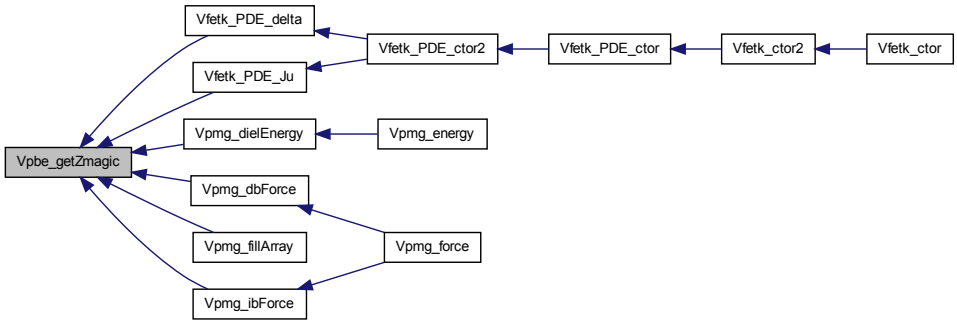
thee	Vpbe object

Returns

Get factor for scaling charges (in e) to internal units

Definition at line 148 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

thee	Vpbe object

Returns

z value of membrane (A)

Definition at line 190 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

thee	Vpbe object

Returns

The memory used by this structure and its contents in bytes

Definition at line 516 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.

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

tok	The string to examine

Returns

1 if the entire string is an integer, 0 if otherwise.

Definition at line 76 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

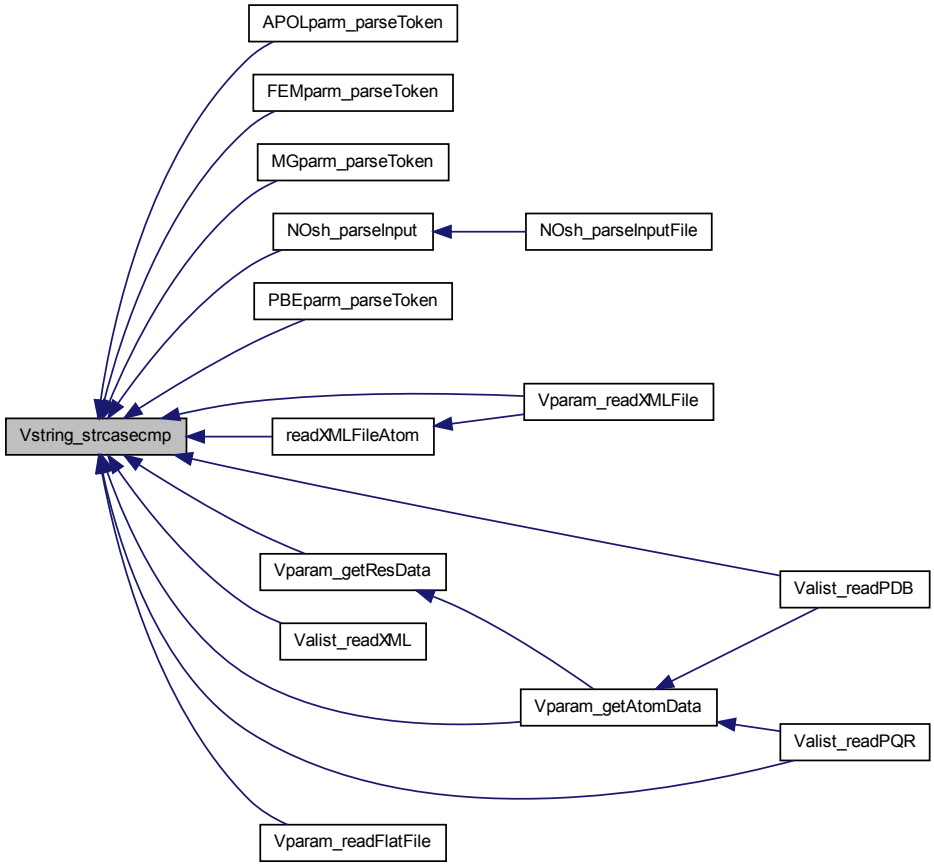
s1	First string for comparison
s2	Second string for comparison

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 12 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 / *in ESU units => kcal/mol*
- #define Vunit_esu_kb 1.9871913e-03
  k_b *in ESU units => 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

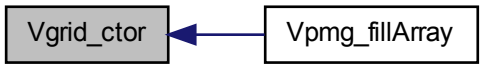
nx	Number grid points in x direction
ny	Number grid points in y direction
nz	Number grid points in z direction
hx	Grid spacing in x direction
hy	Grid spacing in y direction
hzed	Grid spacing in z direction
xmin	x coordinate of lower grid corner
ymin	y coordinate of lower grid corner
zmin	z coordinate of lower grid corner
data	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 70 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

thee	Pointer to newly allocated Vgrid object
nx	Number grid points in x direction
ny	Number grid points in y direction
nz	Number grid points in z direction
hx	Grid spacing in x direction
hy	Grid spacing in y direction
hzed	Grid spacing in z direction
xmin	x coordinate of lower grid corner
ymin	y coordinate of lower grid corner
zmin	z coordinate of lower grid corner
data	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 89 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

thee	Pointer to Vgrid object
pt	Location to evaluate second derivative
cflag	• 0: Reduced Maximal Curvature • 1: Mean Curvature (Laplace) • 2: Gauss Curvature • 3: True Maximal Curvature
curv	Specified curvature value

Returns

1 if successful, 0 if off grid

Definition at line 274 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

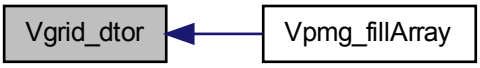
Nathan Baker

Parameters

thee	Pointer to memory location of object to be destroyed

Definition at line 129 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

thee	Pointer to object to be destroyed

Definition at line 142 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

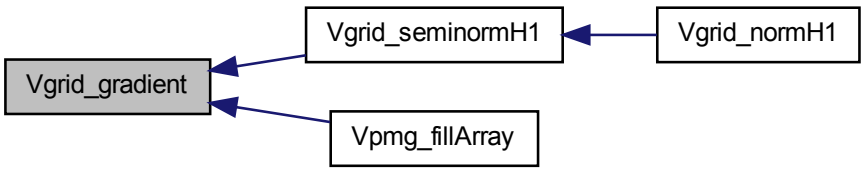
thee	Pointer to Vgrid object
pt	Location to evaluate gradient
grad	Gradient

Returns

1 if successful, 0 if off grid

Definition at line 354 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

thee	Vgrid object

Returns

Integral of data

Definition at line 1320 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

thee	Vgrid object

Returns

The memory used by this structure and its contents in bytes

Definition at line 55 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

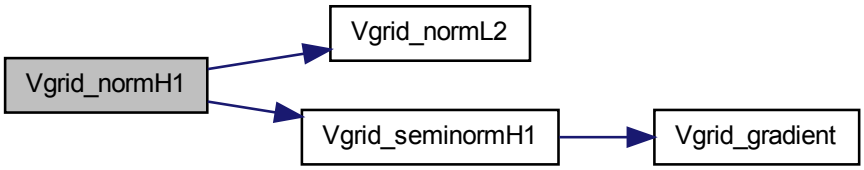
thee	Vgrid object

Returns

Integral of data

Definition at line 1457 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

thee	Vgrid object

Returns

L_1 norm of data

Definition at line 1357 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

thee	Vgrid object

Returns

L_2 norm of data

Definition at line 1386 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

thee	Vgrid object

Returns

$L\infty$ norm of data

Definition at line 1472 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

thee	Vgrid object
iodev	Input device type (FILE/BUFF/UNIX/INET)
iofmt	Input device format (ASCII/XDR)
thost	Input hostname (for sockets)
fname	Input FILE/BUFF/UNIX/INET name

Returns

1 if sucessful, 0 otherwise

Definition at line 555 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

thee	Object with grid data to write
fname	Path to write to

Definition at line 437 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

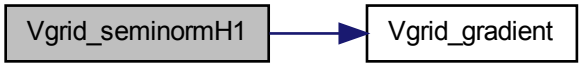
thee	Vgrid object

Returns

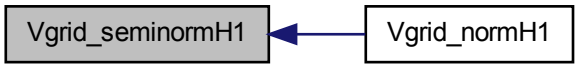
Integral of data

Definition at line 1415 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

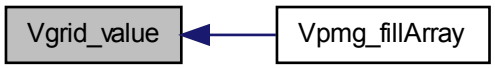
thee	Vgrid obejct
x	Point at which to evaluate potential
value	Value of data at point x

Returns

1 if successful, 0 if off grid

Definition at line 156 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 * *iofmt*, const char * *thost*, const char * *fname*, char * *title*, double * *pvec*)

Write out the data in OpenDX grid format.

Author

Nathan Baker

Parameters

thee	Grid object
iodev	Output device type (FILE/BUFF/UNIX/INET)
iofmt	Output device format (ASCII/XDR)
thost	Output hostname (for sockets)
fname	Output FILE/BUFF/UNIX/INET name
title	Title to be inserted in grid file
pvec	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 973 of file vgrid.c.

8.20.2.18 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.

Note

- The mesh spacing should be uniform
- Format changed from 12.6E to 12.5E

Author

Nathan Baker

Parameters

thee	Grid object
iodev	Output device type (FILE/BUFF/UNIX/INET)
iofmt	Output device format (ASCII/XDR)
thost	Output hostname (for sockets)
fname	Output FILE/BUFF/UNIX/INET name
title	Title to be inserted in grid file
pvec	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 1223 of file vgrid.c.

8.21 Vmgrid class

Oracle for Cartesian mesh data.

Data Structures

- struct sVmgrid

  Multiresoltion 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 hiearchy.

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 hiearchy.
- VEXTERNC Vgrid * Vmgrid_getGridByPoint (Vmgrid *thee, double pt[3])
  Get grid in hiearchy 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

thee	Pointer to object to be destroyed
grid	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 196 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 58 of file vmgrid.c.

8.21.2.3 VEXTERNC int Vmgrid_ctor2 (Vmgrid * *thee*)

Initialize Vmgrid object.

Author

Nathan Baker

Parameters

thee	Newly allocated Vmgrid object

Returns

Newly allocated and initialized Vmgrid object

Definition at line 73 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

thee	Pointer to Vmgrid object
pt	Location to evaluate second derivative
cflag	• 0: Reduced Maximal Curvature • 1: Mean Curvature (Laplace) • 2: Gauss Curvature • 3: True Maximal Curvature
curv	Specified curvature value

Returns

1 if successful, 0 if off grid

Definition at line 139 of file vmgrid.c.

8.21.2.5 VEXTERNC void Vmgrid_dtor (Vmgrid ** *thee*)

Object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of object to be destroyed

Definition at line 89 of file vmgrid.c.

8.21.2.6 VEXTERNC void Vmgrid_dtor2 (Vmgrid * *thee*)

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to object to be destroyed

Definition at line 102 of file vmgrid.c.

8.21.2.7 VEXTERNC Vgrid* Vmgrid_getGridByNum (Vmgrid * *thee*, int *num*)

Get specific grid in hiearchy.

Author

Nathan Baker

Parameters

thee	Pointer to Vmgrid object
num	Number of grid in hiearchy

Returns

Pointer to specified grid

8.21.2.8 VEXTERNC Vgrid* Vmgrid_getGridByPoint (Vmgrid * *thee*, double *pt[3]*)

Get grid in hiearchy which contains specified point or VNULL.

Author

Nathan Baker

Parameters

thee	Pointer to Vmgrid object
pt	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

thee	Pointer to Vmgrid object
pt	Location to evaluate gradient
grad	Gradient

Returns

1 if successful, 0 if off grid

Definition at line 168 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

thee	Vmgrid obejct
x	Point at which to evaluate potential
value	Value of data at point x

Returns

1 if successful, 0 if off grid

Definition at line 108 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

mgrid	Multiple grid object containing potential data (in units kT/e)
pbe	Pointer to Vpbe object for parameters
bcfl	Boundary condition to use for potential values off the grid

Returns

Newly allocated and initialized Vopot object

Definition at line 59 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

thee	Pointer to newly allocated Vopot object
mgrid	Multiple grid object containing potential data (in units kT/e)
pbe	Pointer to Vpbe object for parameters
bcfl	Boundary condition to use for potential values off the grid

Returns

1 if successful, 0 otherwise

Definition at line 74 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

thee	Pointer to Vopot object
pt	Location to evaluate second derivative
cflag	• 0: Reduced Maximal Curvature • 1: Mean Curvature (Laplace) • 2: Gauss Curvature • 3: True Maximal Curvature
curv	Set to specified curvature value

Returns

1 if successful, 0 otherwise

Definition at line 208 of file vopot.c.

8.22.2.4 VEXTERNC void Vopot_dtor (Vopot ** *thee*)

Object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of object to be destroyed

Definition at line 88 of file vopot.c.

8.22.2.5 VEXTERNC void Vopot_dtor2 (Vopot * *thee*)

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to object to be destroyed

Definition at line 101 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

thee	Pointer to Vopot object
pt	Location to evaluate gradient
grad	Gradient

Returns

1 if successful, 0 otherwise

Definition at line 294 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

thee	Vopot obejct
x	Point at which to evaluate potential
pot	Set to dimensionless potential (units kT/e) at point x

Returns

1 if successful, 0 otherwise

Definition at line 108 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 *nlInduced, 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 dipooles 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 *nlInduced, 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 dipooles 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 *nlInduced, 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.

- VEXTERNC void Vpmg_qfMutualPolForce (Vpmg *thee, Vgrid *induced, Vgrid *nlInduced, 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 *nlInduced, 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 *nlInduced, 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

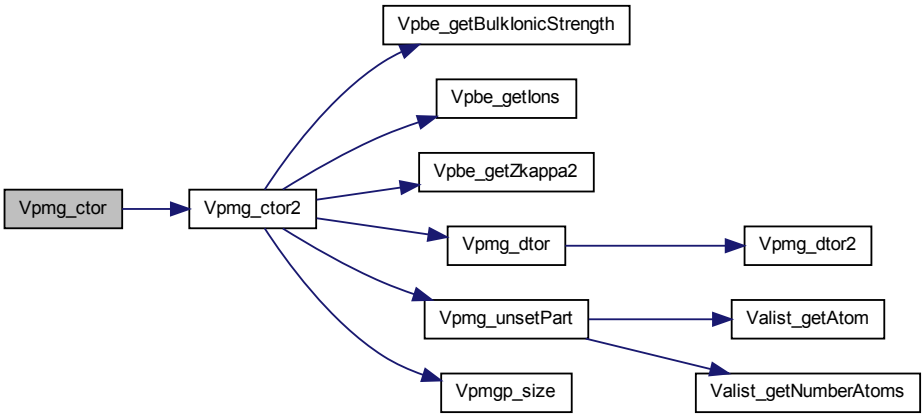
Pointer to newly allocated Vpmg object

Parameters

parms	PMG parameter object
pbe	PBE-specific variables
focusFlag	1 for focusing, 0 otherwise
pmgOLD	Old Vpmg object to use for boundary conditions
mgparm	MGparm parameter object for boundary conditions
energyFlag	What types of energies to calculate

Definition at line 119 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_calcEnergy *energyFlag*)

FORTRAN stub constructor for the Vpmg class (uses previously-allocated memory)

Author

Nathan Baker

Returns

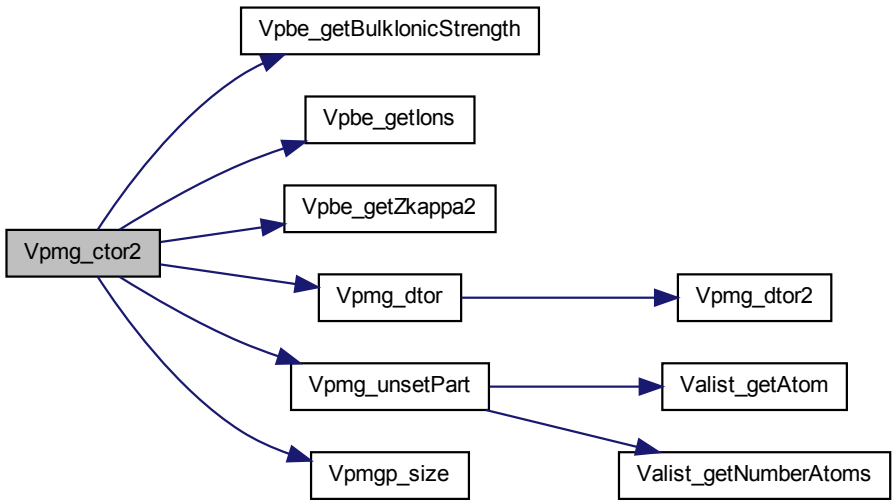
1 if successful, 0 otherwise

Parameters

thee	Memory location for object
parms	PMG parameter object
pbe	PBE-specific variables
focusFlag	1 for focusing, 0 otherwise
pmgOLD	Old Vpmg object to use for boundary conditions (can be VNULL if focusFlag = 0)
mgparm	MGparm parameter object for boundary conditions (can be VNULL if focusFlag = 0)
energyFlag	What types of energies to calculate (ignored if focusFlag = 0)

Definition at line 131 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

thee	Vpmg object
perm	Permanent multipole potential
induced	Induced dipole potential
atomID	Atom index
force	(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

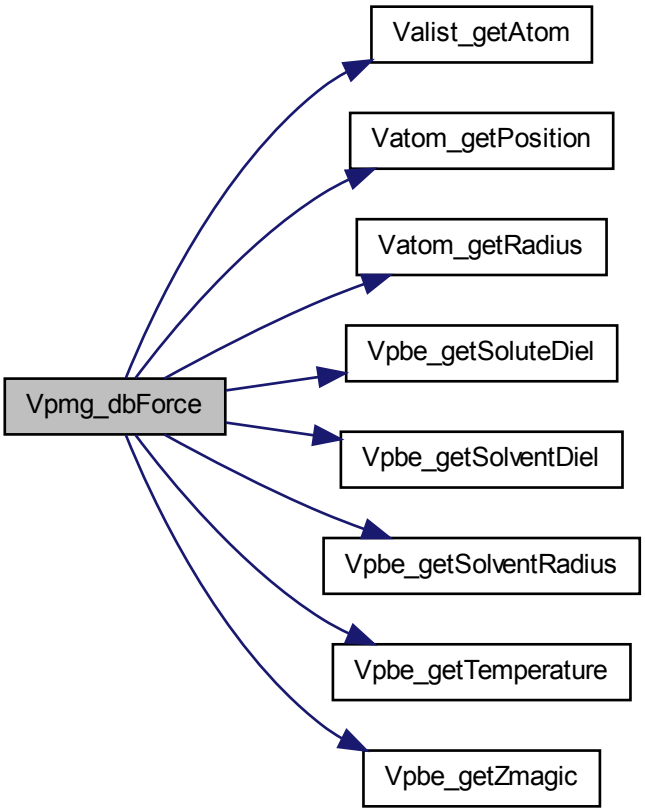
1 if successful, 0 otherwise

Parameters

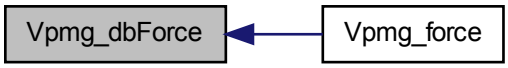
thee	Vpmg object
dbForce	3∗sizeof(double) space to hold the dielectric boundary force in units of k_B T/AA
atomID	Valist ID of desired atom
srfm	Surface discretization method

Definition at line 5744 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 * *nlInduced,* 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

thee	Vpmg object
induced	Induced dipole potential
nlInduced	Non-local induced dipole potential
atomID	Atom index
force	(returned) force

8.23.2.6 VEXTERNC void Vpmg_dbNLDirectPolForce (Vpmg * *thee,* Vgrid * *perm,* Vgrid * *nlInduced,* 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

thee	Vpmg object
perm	Permanent multipole potential
nlInduced	Non-local induced dipole potential
atomID	Atom index
force	(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

thee	Vpmg object
atomID	Atom index
force	(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:

$$G = \frac{1}{2}\int \varepsilon(\nabla u)^2 dx$$

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.

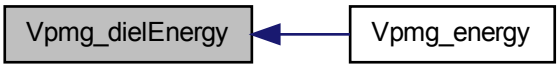
Parameters

thee	Vpmg object
extFlag	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 1182 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):

$$\int \|\nabla \varepsilon\| dx$$

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

thee	Vpmg object

Definition at line 1229 of file vpmg.c.

8.23.2.10 VEXTERNC void Vpmg_dtor (Vpmg ** *thee*)

Object destructor.

Author

Nathan Baker

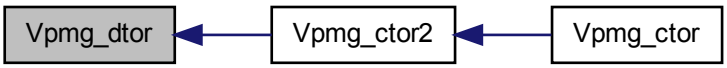
Parameters

thee	Pointer to memory location of object to be destroyed

Definition at line 487 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

thee	Pointer to object to be destroyed

Definition at line 497 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

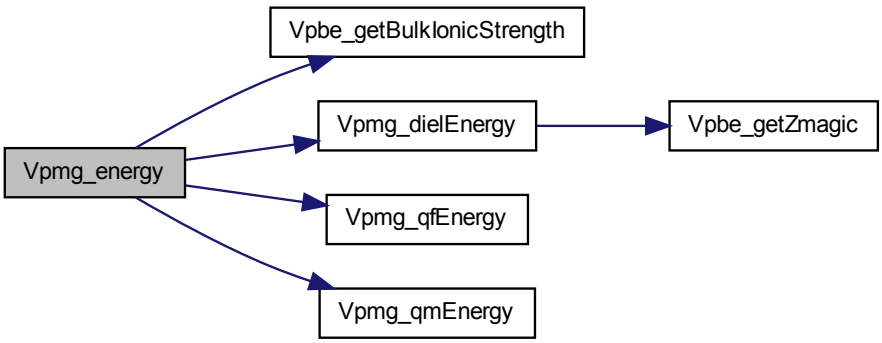
The electrostatic energy in units of k_B T.

Parameters

thee	Vpmg object
extFlag	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 1153 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

thee	Vpmg object
atomID	Atom index
field	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

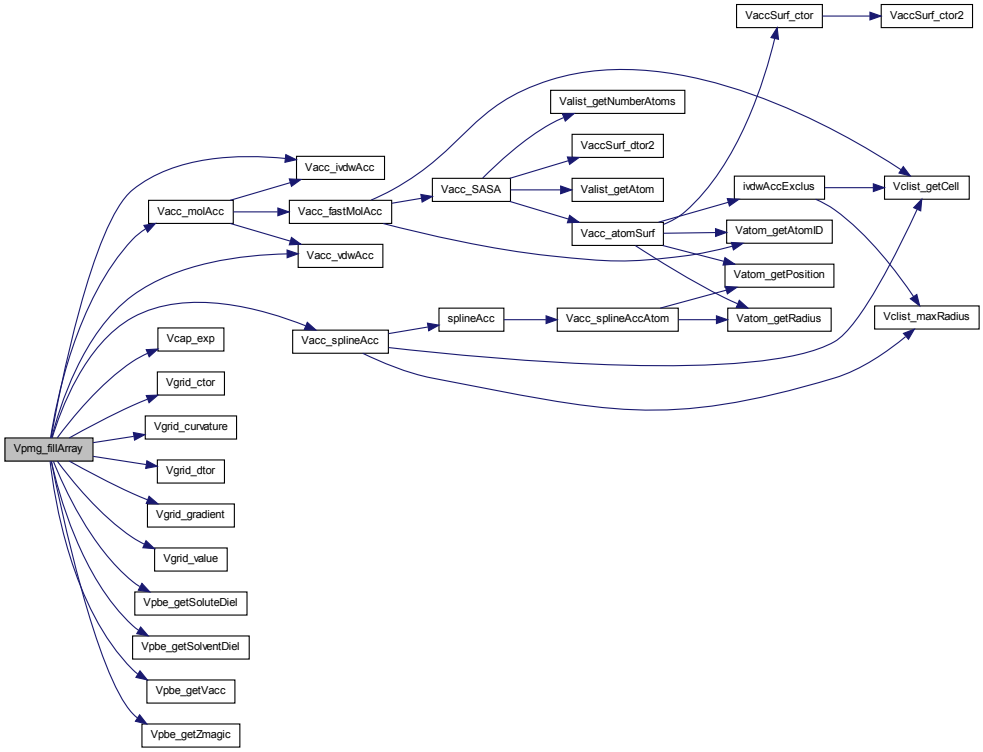
1 if successful, 0 otherwise

Parameters

thee	Vpmg object
vec	A nx*ny*nz*sizeof(double) array to contain the values to be written
type	What to write
parm	Parameter for data type definition (if needed)
pbetype	Parameter for PBE type (if needed)
pbeparm	Pass in the PBE parameters (if needed)

Definition at line 818 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

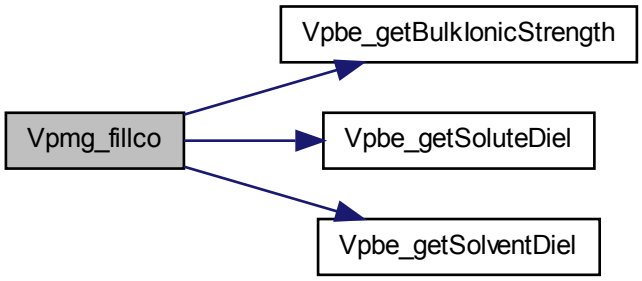
1 if successful, 0 otherwise

Parameters

thee	Vpmg object
surfMeth	Surface discretization method
splineWin	Spline window (in A) for surfMeth = VSM_SPLINE
chargeMeth	Charge discretization method
useDielXMap	Boolean to use dielectric map argument
dielXMap	External dielectric map
useDielYMap	Boolean to use dielectric map argument
dielYMap	External dielectric map
useDielZMap	Boolean to use dielectric map argument
dielZMap	External dielectric map
useKappaMap	Boolean to use kappa map argument
kappaMap	External kappa map
usePotMap	Boolean to use potential map argument
potMap	External potential map
useChargeMa	Boolean to use charge map argument
chargeMap	External charge map

Definition at line 5414 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

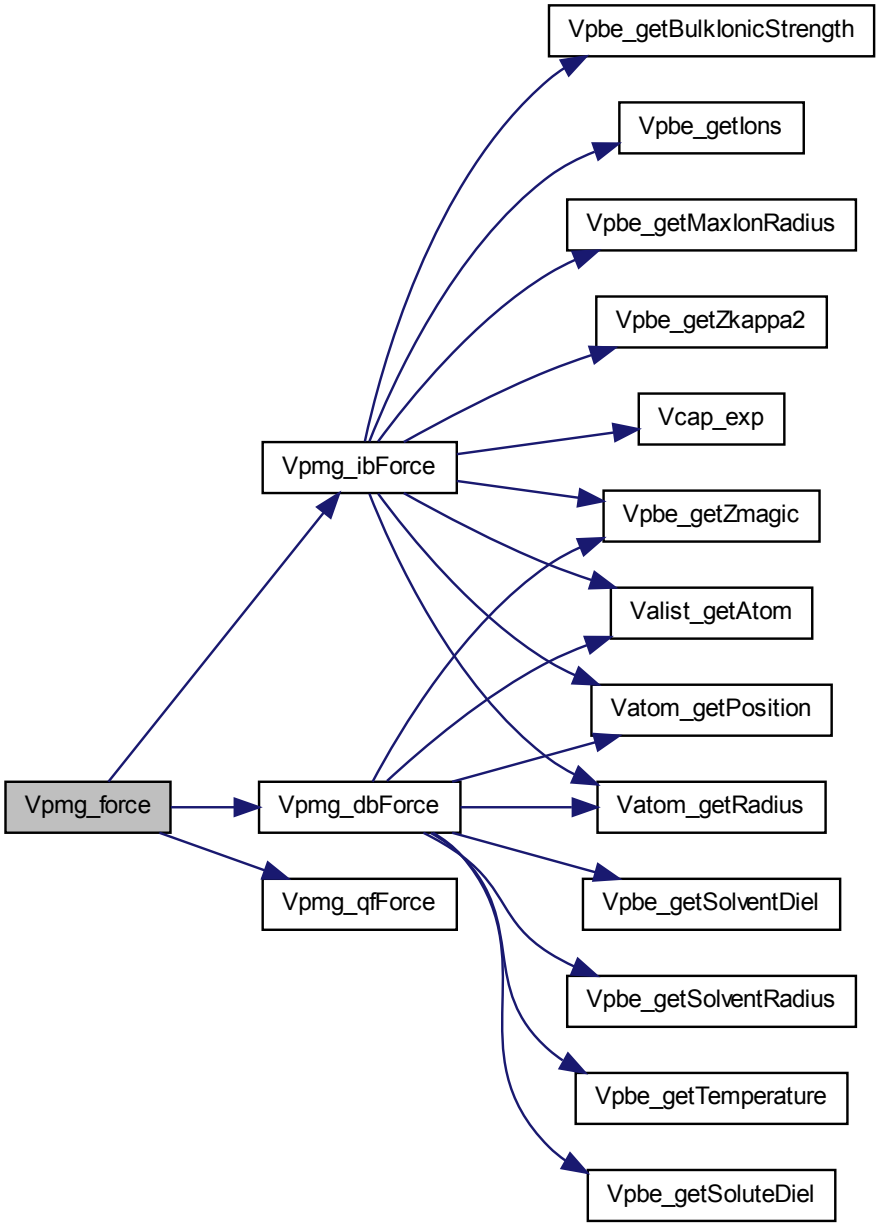
1 if successful, 0 otherwise

Parameters

thee	Vpmg object
force	3*sizeof(double) space to hold the force in units of k_B T/AA
atomID	Valist ID of desired atom
srfm	Surface discretization method
chgm	Charge discretization method

Definition at line 5556 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

thee	Vpmg object
perm	Permanent multipole potential
induced	Induced dipole potential
atomID	Atom index
force	(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

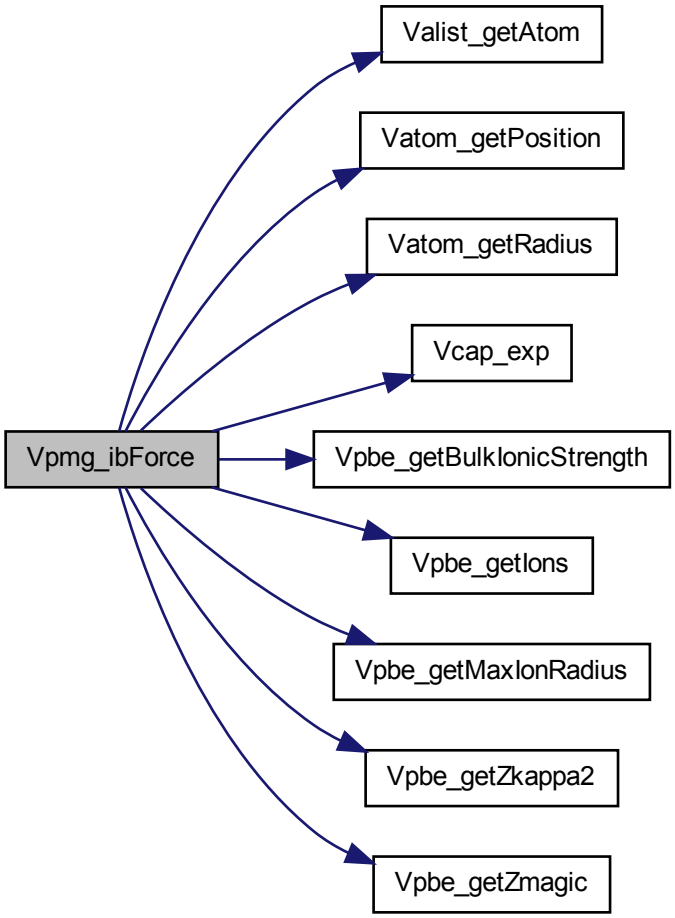
1 if successful, 0 otherwise

Parameters

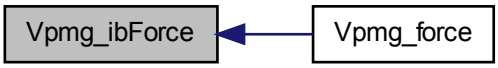
thee	Vpmg object
force	3*sizeof(double) space to hold the boundary force in units of k_B T/AA
atomID	Valist ID of desired atom
srfm	Surface discretization method

Definition at line 5579 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 * *nlInduced,* 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

thee	Vpmg object
induced	Induced dipole potential
nlInduced	Non-local induced dipole potential
atomID	Atom index
force	(returned) force

8.23.2.20 VEXTERNC void Vpmg_ibNLDirectPolForce (Vpmg * *thee,* Vgrid * *perm,* Vgrid * *nlInduced,* 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 dipooles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

Author

Michael Schnieders

Parameters

thee	Vpmg object
perm	Permanent multipole potential
nlInduced	Induced dipole potential
atomID	Atom index
force	(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

thee	Vpmg object
atomID	Atom index
force	(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

thee	Object for memory check

Definition at line 58 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

thee	Vpmg object
path	The file to which the matrix is to be written
title	The title of the matrix
mxtype	The type of REAL-valued matrix, a 3-character string of the form "R_A" where the '_' can be one of: • S: symmetric matrix • U: unsymmetric matrix • H: Hermitian matrix • Z: skew-symmetric matrix • R: rectangular matrix
flag	The operator to compress: • 0: Poisson operator • 1: Linearization of the full Poisson-Boltzmann operator around the current solution

Definition at line 66 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

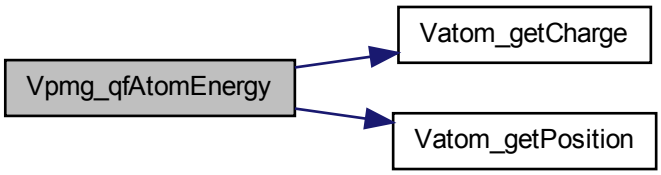
The fixed charge electrostatic energy in units of k_B T.

Parameters

thee	The Vpmg object
atom	The atom for energy calculations

Definition at line 1612 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

thee	Vpmg object

perm	Permanent multipole potential
induced	Induced dipole potential
atomID	Atom index
force	(returned) force
torque	(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

thee	Vpmg object
extFlag	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 1512 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

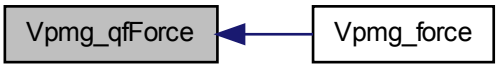
1 if sucessful, 0 otherwise

Parameters

thee	Vpmg object
force	3*sizeof(double) space to hold the force in units of k_B T/A
atomID	Valist ID of desired atom
chgm	Charge discretization method

Definition at line 6001 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 * *nlInduced,* 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

thee	Vpmg object
induced	Induced dipole potential
nlInduced	Non-local induced dipole potential
atomID	Atom index
force	(returned) force

8.23.2.29 VEXTERNC void Vpmg_qfNLDirectPolForce (Vpmg * *thee,* Vgrid * *perm,* Vgrid * *nlInduced,* 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 dipooles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

Author

Michael Schnieders

Parameters

thee	Vpmg object
perm	Permanent multipole potential
nlInduced	Non-local induced dipole potential
atomID	Atom index
force	(returned) force
torque	(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

thee	Vpmg object
atomID	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

thee	Vpmg object
atomID	Atom index
force	(returned) force
torque	(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

$$G = \frac{1}{2} \int \overline{\kappa}^2(x) u^2(x) dx$$

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

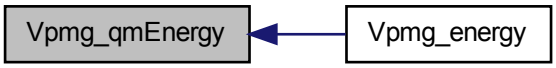
The mobile charge electrostatic energy in units of k_B T.

Parameters

thee	Vpmg object
extFlag	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 1273 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

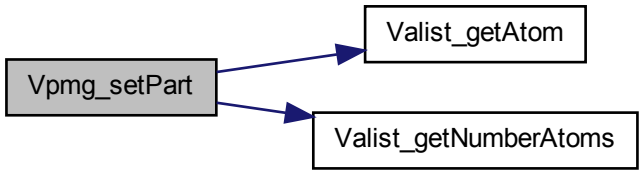
Nathan Baker

Parameters

thee	Vpmg object
lowerCorner	Partition lower corner
upperCorner	Partition upper corner
bflags	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 553 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

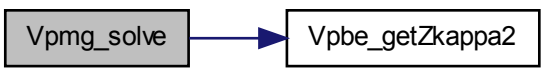
1 if successful, 0 otherwise

Parameters

thee	Vpmg object

Definition at line 351 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

thee	Vpmg object

Definition at line 6776 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

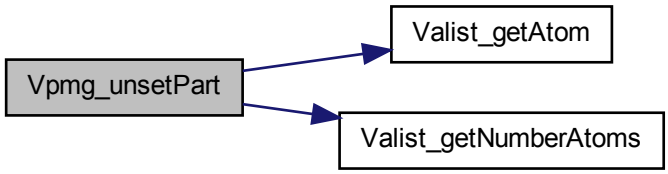
Nathan Baker

Parameters

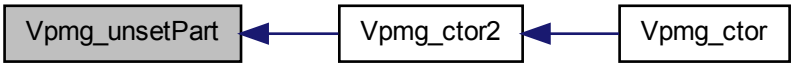
thee	Vpmg object

Definition at line 798 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

mgparm	MGParm object containing parameters to be used in setup

Returns

Newly allocated and initialized Vpmgp object

Definition at line 70 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

thee	Newly allocated PMG object
mgparm	MGParm object containing parameters to be used in setup

Returns

1 if successful, 0 otherwise

Definition at line 87 of file vpmgp.c.

8.24.2.3 VEXTERNC void Vpmgp_dtor (Vpmgp ** *thee*)

Object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to memory location for Vpmgp object

Definition at line 172 of file vpmgp.c.

8.24.2.4 VEXTERNC void Vpmgp_dtor2 (Vpmgp * *thee*)

FORTRAN stub for object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to Vpmgp object

Definition at line 187 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

numLevel	Number of levels to coarsen
nxOld	Number of old grid points in this direction
nyOld	Number of old grid points in this direction
nzOld	Number of old grid points in this direction
nxNew	Number of new grid points in this direction
nyNew	Number of new grid points in this direction
nzNew	Number of new grid points in this direction

Definition at line 306 of file vpmgp.c.

8.24.2.6 VEXTERNC void Vpmgp_size (Vpmgp * *thee*)

Determine array sizes and parameters for multigrid solver.

Author

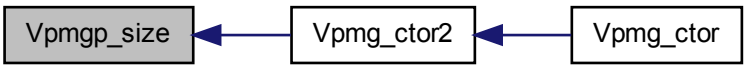
Mike Holst and Nathan Baker

Parameters

thee	Object to be sized

Definition at line 190 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/Desktop/Sync/Software/apbs/trunk/src/generic/apbs/apolpar`

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 118 of file apolparm.h.

9.1.2 Field Documentation

9.1.2.1 double bconc

Vacc sphere density

Definition at line 128 of file apolparm.h.

9.1.2.2 APOLparm_calcEnergy calcenergy

Energy calculation flag

Definition at line 156 of file apolparm.h.

9.1.2.3 APOLparm_calcForce calcforce

Atomic forces calculation

Definition at line 159 of file apolparm.h.

9.1.2.4 double dpos

Atom position offset

Definition at line 134 of file apolparm.h.

9.1.2.5 double gamma

Surface tension for apolar energies/forces (in kJ/mol/A^2)

Definition at line 152 of file apolparm.h.

9.1.2.6 double grid[3]

Grid spacing

Definition at line 122 of file apolparm.h.

9.1.2.7 int molid

Molecule ID to perform calculation on

Definition at line 125 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 120 of file apolparm.h.

9.1.2.9 double press

Solvent pressure

Definition at line 137 of file apolparm.h.

9.1.2.10 double sasa

Solvent accessible surface area for this calculation

Definition at line 164 of file apolparm.h.

9.1.2.11 double sav

Solvent accessible volume for this calculation

Definition at line 165 of file apolparm.h.

9.1.2.12 double sdens

Vacc sphere density

Definition at line 131 of file apolparm.h.

9.1.2.13 int setbconc

Flag,

See also

bconc

Definition at line 129 of file apolparm.h.

9.1.2.14 int setcalcenergy

Flag,

See also

calcenergy

Definition at line 157 of file apolparm.h.

9.1.2.15 int setcalcforce

Flag,

See also

calcforce

Definition at line 160 of file apolparm.h.

9.1.2.16 int setdpos

Flag,

See also

dpos

Definition at line 135 of file apolparm.h.

9.1.2.17 int setgamma

Flag,

See also

gamma

Definition at line 154 of file apolparm.h.

9.1.2.18 int setgrid

Flag,

See also

grid

Definition at line 123 of file apolparm.h.

9.1.2.19 int setmolid

Flag,

See also

molid

Definition at line 126 of file apolparm.h.

9.1.2.20 int setpress

Flag,

See also

press

Definition at line 138 of file apolparm.h.

9.1.2.21 int setsdens

Flag,

See also

sdens

Definition at line 132 of file apolparm.h.

9.1.2.22 int setsrad

Flag,

See also

srad

Definition at line 144 of file apolparm.h.

9.1.2.23 int setsrfm

Flag,

See also

srfm

Definition at line 141 of file apolparm.h.

9.1.2.24 int setswin

Flag,

See also

swin

Definition at line 147 of file apolparm.h.

9.1.2.25 int settemp

Flag,

See also

temp

Definition at line 150 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 169 of file apolparm.h.

9.1.2.27 double srad

Solvent radius

Definition at line 143 of file apolparm.h.

9.1.2.28 Vsurf_Meth srfm

Surface calculation method

Definition at line 140 of file apolparm.h.

9.1.2.29 double swin

Cubic spline window

Definition at line 146 of file apolparm.h.

9.1.2.30 double temp

Temperature (in K)

Definition at line 149 of file apolparm.h.

9.1.2.31 double totForce[3]

Total forces on x, y, z

Definition at line 167 of file apolparm.h.

9.1.2.32 double watepsilon

Water oxygen Lennard-Jones well depth (kJ/mol)

Definition at line 163 of file apolparm.h.

9.1.2.33 double watsigma

Water oxygen Lennard-Jones radius (A)

Definition at line 162 of file apolparm.h.

9.1.2.34 double wcaEnergy

wcaEnergy

Definition at line 166 of file apolparm.h.

The documentation for this struct was generated from the following file:

- src/generic/apbs/apolparm.h

9.2 sFEMparm Struct Reference

Parameter structure for FEM-specific variables from input files.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/src/generic/apb`

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 122 of file femparm.h.

9.2.2 Field Documentation

9.2.2.1 FEMparm_EstType akeyPRE

Adaptive refinment error estimator method for pre-solution refine. Note, this should either be FRT_UNIF or FRT_GEOM.

Definition at line 137 of file femparm.h.

9.2.2.2 FEMparm_EstType akeySOLVE

Adaptive refinment error estimator method for a posteriori solution-based refinement.

Definition at line 141 of file femparm.h.

9.2.2.3 FEMparm_EtolType ekey

Adaptive refinment interpretation of error tolerance

Definition at line 134 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 131 of file femparm.h.

9.2.2.5 double glen[3]

Domain side lengths (in Å)

Definition at line 129 of file femparm.h.

9.2.2.6 int maxsolve

Maximum number of solve-estimate-refine cycles

Definition at line 154 of file femparm.h.

9.2.2.7 int maxvert

Maximum number of vertices in mesh (ignored if less than zero)

Definition at line 156 of file femparm.h.

9.2.2.8 int meshID

External finite element mesh ID (if used)

Definition at line 163 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 124 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 159 of file femparm.h.

9.2.2.11 int setakeyPRE

Boolean

Definition at line 140 of file femparm.h.

9.2.2.12 int setakeySOLVE

Boolean

Definition at line 143 of file femparm.h.

9.2.2.13 int setekey

Boolean

Definition at line 136 of file femparm.h.

9.2.2.14 int setetol

Boolean

Definition at line 133 of file femparm.h.

9.2.2.15 int setglen

Boolean

Definition at line 130 of file femparm.h.

9.2.2.16 int setmaxsolve

Boolean

Definition at line 155 of file femparm.h.

9.2.2.17 int setmaxvert

Boolean

Definition at line 158 of file femparm.h.

9.2.2.18 int settargetNum

Boolean

Definition at line 148 of file femparm.h.

9.2.2.19 int settargetRes

Boolean

Definition at line 153 of file femparm.h.

9.2.2.20 int settype

Boolean

Definition at line 128 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 144 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 149 of file femparm.h.

9.2.2.23 FEMparm_CalcType type

Calculation type

Definition at line 127 of file femparm.h.

9.2.2.24 int useMesh

Indicates whether we use external finite element mesh

Definition at line 162 of file femparm.h.

The documentation for this struct was generated from the following file:

- src/generic/apbs/femparm.h

9.3 sMGparm Struct Reference

Parameter structure for MG-specific variables from input files.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/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 setgcent
- double cglen [3]
- int setcglen
- double fglen [3]
- int setfglen
- MGparm_CentMeth ccmeth
- double ccenter [3]
- int ccentmol
- int setcgcent
- 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 setsize
- 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 103 of file mgparm.h.

9.3.2 Field Documentation

9.3.2.1 int async

Processor ID for asynchronous calculation

Definition at line 175 of file mgparm.h.

9.3.2.2 double ccenter[3]

Coarse grid center.

Definition at line 146 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 147 of file mgparm.h.

9.3.2.4 MGparm_CentMeth ccmeth

Coarse grid centering method

Definition at line 145 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 127 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 135 of file mgparm.h.

9.3.2.7 double cglen[3]

Coarse grid side lengths

Definition at line 141 of file mgparm.h.

9.3.2.8 Vchrg_Meth chgm

Charge discretization method

Definition at line 111 of file mgparm.h.

9.3.2.9 Vchrg_Src chgs

Charge source (Charge, Multipole, Induced Dipole, NL Induced

Definition at line 113 of file mgparm.h.

9.3.2.10 MGparm_CentMeth cmeth

Centering method

Definition at line 126 of file mgparm.h.

9.3.2.11 int dime[3]

Grid dimensions

Definition at line 109 of file mgparm.h.

9.3.2.12 double etol

User-defined error tolerance

Definition at line 120 of file mgparm.h.

9.3.2.13 double fcenter[3]

Fine grid center.

Definition at line 152 of file mgparm.h.

9.3.2.14 int fcentmol

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 153 of file mgparm.h.

9.3.2.15 MGparm_CentMeth fcmeth

Fine grid centering method

Definition at line 151 of file mgparm.h.

9.3.2.16 double fglen[3]

Fine grid side lengths

Definition at line 143 of file mgparm.h.

9.3.2.17 double glen[3]

Grid side lengths.

Definition at line 124 of file mgparm.h.

9.3.2.18 double grid[3]

Grid spacings

Definition at line 122 of file mgparm.h.

9.3.2.19 int method

Solver Method

Definition at line 181 of file mgparm.h.

9.3.2.20 int nlev

Levels in multigrid hierarchy

Deprecated

Just ignored now

Definition at line 117 of file mgparm.h.

9.3.2.21 int nonlintype

Linearity Type Method to be used

Definition at line 178 of file mgparm.h.

9.3.2.22 double ofrac

Overlap fraction between procs

Definition at line 173 of file mgparm.h.

9.3.2.23 int parsed

Has this structure been filled? (0 = no, 1 = yes)

Definition at line 106 of file mgparm.h.

9.3.2.24 double partDisjCenter[3]

This gives the center of the disjoint partitions

Definition at line 160 of file mgparm.h.

9.3.2.25 double partDisjLength[3]

This gives the lengths of the disjoint partitions

Definition at line 162 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 164 of file mgparm.h.

9.3.2.27 int pdime[3]

Grid of processors to be used in calculation

Definition at line 167 of file mgparm.h.

9.3.2.28 int proc_rank

Rank of this processor

Definition at line 169 of file mgparm.h.

9.3.2.29 int proc_size

Total number of processors

Definition at line 171 of file mgparm.h.

9.3.2.30 int setasync

Flag,

See also

asynch

Definition at line 176 of file mgparm.h.

9.3.2.31 int setcgcent

Flag,

See also

ccmeth

Definition at line 150 of file mgparm.h.

9.3.2.32 int setcglen

Flag,

See also

cglen

Definition at line 142 of file mgparm.h.

9.3.2.33 int setchgm

Flag,

See also

chgm

Definition at line 112 of file mgparm.h.

9.3.2.34 int setdime

Flag,

See also

dime

Definition at line 110 of file mgparm.h.

9.3.2.35 int setetol

Flag,

See also

etol

Definition at line 121 of file mgparm.h.

9.3.2.36 int setfgcent

Flag,

See also

fcmeth

Definition at line 156 of file mgparm.h.

9.3.2.37 int setfglen

Flag,

See also

fglen

Definition at line 144 of file mgparm.h.

9.3.2.38 int setgcent

Flag,

See also

cmeth

Definition at line 138 of file mgparm.h.

9.3.2.39 int setglen

Flag,

See also

glen

Definition at line 125 of file mgparm.h.

9.3.2.40 int setgrid

Flag,

See also

grid

Definition at line 123 of file mgparm.h.

9.3.2.41 int setmethod

Flag,

See also

method

Definition at line 182 of file mgparm.h.

9.3.2.42 int setnlev

Flag,

See also

nlev

Definition at line 119 of file mgparm.h.

9.3.2.43 int setnonlintype

Flag,

See also

nonlintype

Definition at line 179 of file mgparm.h.

9.3.2.44 int setofrac

Flag,

See also

ofrac

Definition at line 174 of file mgparm.h.

9.3.2.45 int setpdime

Flag,

See also

pdime

Definition at line 168 of file mgparm.h.

9.3.2.46 int setrank

Flag,

See also

proc_rank

Definition at line 170 of file mgparm.h.

9.3.2.47 int setsize

Flag,

See also

proc_size

Definition at line 172 of file mgparm.h.

9.3.2.48 int setUseAqua

Flag,

See also

useAqua

Definition at line 185 of file mgparm.h.

9.3.2.49 MGparm_CalcType type

What type of MG calculation?

Definition at line 105 of file mgparm.h.

9.3.2.50 int useAqua

Enable use of lpbe/aqua

Definition at line 184 of file mgparm.h.

The documentation for this struct was generated from the following file:

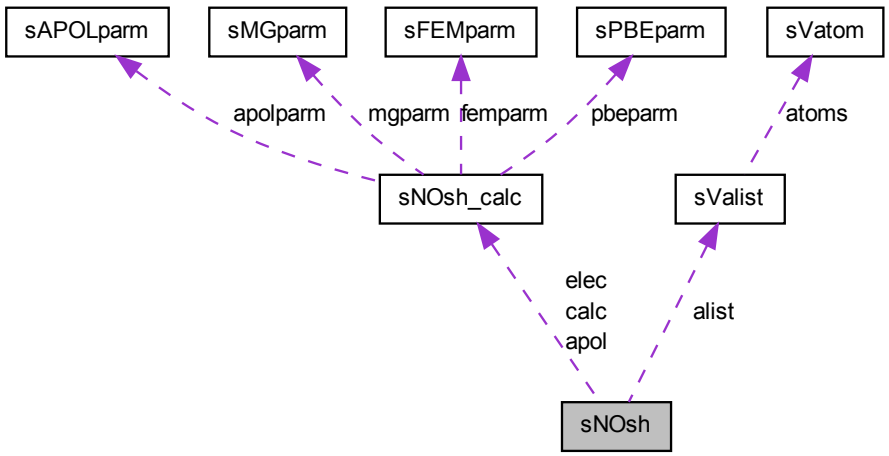
- src/generic/apbs/mgparm.h

9.4 sNOsh Struct Reference

Class for parsing fixed format input files.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/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 174 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 213 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 187 of file nosh.h.

9.4.2.3 int apol2calc[NOSH_MAXCALC]

(see elec2calc)

Definition at line 208 of file nosh.h.

9.4.2.4 char apolname[NOSH_MAXCALC][VMAX_ARGLEN]

Optional user-specified name for APOLAR statement

Definition at line 248 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 196 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 176 of file nosh.h.

9.4.2.7 Vdata_Format chargefmt[NOSH_MAXMOL]

Charge maps fileformats

Definition at line 234 of file nosh.h.

9.4.2.8 char chargepath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to charge map files

Definition at line 233 of file nosh.h.

9.4.2.9 Vdata_Format dielfmt[NOSH_MAXMOL]

Dielectric maps file formats

Definition at line 225 of file nosh.h.

9.4.2.10 char dielXpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to x-shifted dielectric map files

Definition at line 219 of file nosh.h.

9.4.2.11 char dielYpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to y-shifted dielectric map files

Definition at line 221 of file nosh.h.

9.4.2.12 char dielZpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to z-shifted dielectric map files

Definition at line 223 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 181 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 calcualtions. 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 200 of file nosh.h.

9.4.2.15 char elecname[NOSH_MAXCALC][VMAX_ARGLEN]

Optional user-specified name for ELEC statement

Definition at line 246 of file nosh.h.

9.4.2.16 int gotparm

Either have (1) or don't have (0) parm

Definition at line 215 of file nosh.h.

9.4.2.17 int ispara

1 => is a parallel calculation, 0 => is not

Definition at line 193 of file nosh.h.

9.4.2.18 Vdata_Format kappafmt[NOSH_MAXMOL]

Kappa maps file formats

Definition at line 228 of file nosh.h.

9.4.2.19 char kappapath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to kappa map files

Definition at line 227 of file nosh.h.

9.4.2.20 Vdata_Format meshfmt[NOSH_MAXMOL]

Mesh fileformats

Definition at line 237 of file nosh.h.

9.4.2.21 char meshpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to mesh files

Definition at line 236 of file nosh.h.

9.4.2.22 NOsh_MolFormat molfmt[NOSH_MAXMOL]

Mol files formats

Definition at line 212 of file nosh.h.

9.4.2.23 char molpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to mol files

Definition at line 211 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 190 of file nosh.h.

9.4.2.25 int ncalc

The number of calculations in the calc array

Definition at line 179 of file nosh.h.

9.4.2.26 int ncharge

Number of charge maps

Definition at line 232 of file nosh.h.

9.4.2.27 int ndiel

Number of dielectric maps

Definition at line 218 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 184 of file nosh.h.

9.4.2.29 int nkappa

Number of kappa maps

Definition at line 226 of file nosh.h.

9.4.2.30 int nmesh

Number of meshes

Definition at line 235 of file nosh.h.

9.4.2.31 int nmol

Number of molecules

Definition at line 210 of file nosh.h.

9.4.2.32 int npot

Number of potential maps

Definition at line 229 of file nosh.h.

9.4.2.33 int nprint

How many print sections?

Definition at line 238 of file nosh.h.

9.4.2.34 NOsh_ParmFormat parmfmt

Parm file format

Definition at line 217 of file nosh.h.

9.4.2.35 char parmpath[VMAX_ARGLEN]

Paths to parm file

Definition at line 216 of file nosh.h.

9.4.2.36 int parsed

Have we parsed an input file yet?

Definition at line 245 of file nosh.h.

9.4.2.37 Vdata_Format potfmt[NOSH_MAXMOL]

Potential maps file formats

Definition at line 231 of file nosh.h.

9.4.2.38 char potpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to potential map files

Definition at line 230 of file nosh.h.

9.4.2.39 int printcalc[NOSH_MAXPRINT][NOSH_MAXPOP]

ELEC id (see elec2calc)

Definition at line 242 of file nosh.h.

9.4.2.40 int printnarg[NOSH_MAXPRINT]

How many arguments in energy list

Definition at line 241 of file nosh.h.

9.4.2.41 int printop[NOSH_MAXPRINT][NOSH_MAXPOP]

Operation id (0 = add, 1 = subtract)

Definition at line 243 of file nosh.h.

9.4.2.42 NOsh_PrintType printwhat[NOSH_MAXPRINT]

What do we print:

- 0 = energy,
- 1 = force

Definition at line 239 of file nosh.h.

9.4.2.43 int proc_rank

Processor rank in parallel calculation

Definition at line 194 of file nosh.h.

9.4.2.44 int proc_size

Number of processors in parallel calculation

Definition at line 195 of file nosh.h.

The documentation for this struct was generated from the following file:

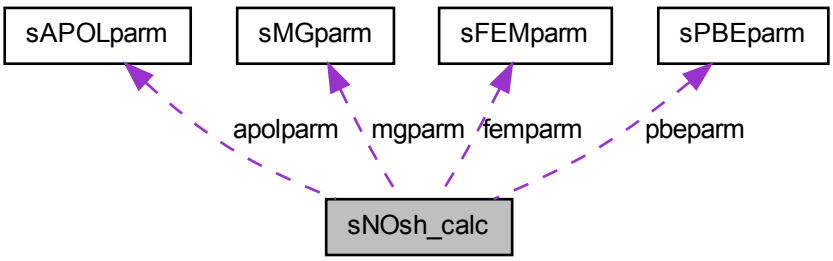
- src/generic/apbs/nosh.h

9.5 sNOsh_calc Struct Reference

Calculation class for use when parsing fixed format input files.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/src/generic/apb`

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 155 of file nosh.h.

9.5.2 Field Documentation

9.5.2.1 APOLparm* apolparm

Non-polar parameters

Definition at line 159 of file nosh.h.

9.5.2.2 NOsh_CalcType calctype

Calculation type

Definition at line 160 of file nosh.h.

9.5.2.3 FEMparm* femparm

Finite element parameters

Definition at line 157 of file nosh.h.

9.5.2.4 MGparm* mgparm

Multigrid parameters

Definition at line 156 of file nosh.h.

9.5.2.5 PBEparm* pbeparm

Generic PBE parameters

Definition at line 158 of file nosh.h.

The documentation for this struct was generated from the following file:

- src/generic/apbs/nosh.h

9.6 sPBEparm Struct Reference

Parameter structure for PBE variables from input files.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/src/generic/apbs`

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 writmatstem [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 108 of file pbeparm.h.

9.6.2 Field Documentation

9.6.2.1 Vbcfl bcfl

Boundary condition method

Definition at line 127 of file pbeparm.h.

9.6.2.2 PBEparm_calcEnergy calcenergy

Energy calculation flag

Definition at line 156 of file pbeparm.h.

9.6.2.3 PBEparm_calcForce calcforce

Atomic forces calculation

Definition at line 158 of file pbeparm.h.

9.6.2.4 int chargeMapID

Charge distribution map ID (if used)

Definition at line 124 of file pbeparm.h.

9.6.2.5 int dielMapID

Dielectric map ID (if used)

Definition at line 114 of file pbeparm.h.

9.6.2.6 double ionc[MAXION]

Counterion concentrations (in M)

Definition at line 132 of file pbeparm.h.

9.6.2.7 double ionq[MAXION]

Counterion charges (in e)

Definition at line 131 of file pbeparm.h.

9.6.2.8 double ionr[MAXION]

Counterion radii (in A)

Definition at line 133 of file pbeparm.h.

9.6.2.9 int kappaMapID

Kappa map ID (if used)

Definition at line 117 of file pbeparm.h.

9.6.2.10 double Lmem

membrane width

Definition at line 167 of file pbeparm.h.

9.6.2.11 double mdie

membrane dielectric constant

Definition at line 169 of file pbeparm.h.

9.6.2.12 double memv

Membrane potential

Definition at line 171 of file pbeparm.h.

9.6.2.13 int molid

Molecule ID to perform calculation on

Definition at line 110 of file pbeparm.h.

9.6.2.14 int nion

Number of counterion species

Definition at line 129 of file pbeparm.h.

9.6.2.15 int numwrite

Number of write statements encountered

Definition at line 176 of file pbeparm.h.

9.6.2.16 int parsed

Has this been filled with anything other than the default values?

Definition at line 192 of file pbeparm.h.

9.6.2.17 Vhal_PBEType pbetype

Which version of the PBE are we solving?

Definition at line 125 of file pbeparm.h.

9.6.2.18 double pdie

Solute dielectric

Definition at line 135 of file pbeparm.h.

9.6.2.19 int potMapID

Kappa map ID (if used)

Definition at line 120 of file pbeparm.h.

9.6.2.20 double sdens

Vacc sphere density

Definition at line 137 of file pbeparm.h.

9.6.2.21 double sdie

Solvent dielectric

Definition at line 139 of file pbeparm.h.

9.6.2.22 int setbcfl

Flag,

See also

bcfl

Definition at line 128 of file pbeparm.h.

9.6.2.23 int setcalcenergy

Flag,

See also

calcenergy

Definition at line 157 of file pbeparm.h.

9.6.2.24 int setcalcforce

Flag,

See also

calcforce

Definition at line 159 of file pbeparm.h.

9.6.2.25 int setion[MAXION]

Flag,

See also

ionq

Definition at line 134 of file pbeparm.h.

9.6.2.26 int setLmem

Flag

Definition at line 168 of file pbeparm.h.

9.6.2.27 int setmdie

Flag

Definition at line 170 of file pbeparm.h.

9.6.2.28 int setmemv

Flag

Definition at line 172 of file pbeparm.h.

9.6.2.29 int setmolid

Flag,

See also

molid

Definition at line 111 of file pbeparm.h.

9.6.2.30 int setnion

Flag,

See also

nion

Definition at line 130 of file pbeparm.h.

9.6.2.31 int setpbetype

Flag,

See also

pbetype

Definition at line 126 of file pbeparm.h.

9.6.2.32 int setpdie

Flag,

See also

pdie

Definition at line 136 of file pbeparm.h.

9.6.2.33 int setsdens

Flag,

See also

sdens

Definition at line 138 of file pbeparm.h.

9.6.2.34 int setsdie

Flag,

See also

sdie

Definition at line 140 of file pbeparm.h.

9.6.2.35 int setsmsize

Flag,

See also

temp

Definition at line 151 of file pbeparm.h.

9.6.2.36 int setsmvolume

Flag,

See also

temp

Definition at line 154 of file pbeparm.h.

9.6.2.37 int setsrad

Flag,

See also

srad

Definition at line 144 of file pbeparm.h.

9.6.2.38 int setsrfm

Flag,

See also

srfm

Definition at line 142 of file pbeparm.h.

9.6.2.39 int setswin

Flag,

See also

swin

Definition at line 146 of file pbeparm.h.

9.6.2.40 int settemp

Flag,

See also

temp

Definition at line 148 of file pbeparm.h.

9.6.2.41 int setwritemat

Flag,

See also

writemat

Definition at line 185 of file pbeparm.h.

9.6.2.42 int setzmem

Flag

Definition at line 166 of file pbeparm.h.

9.6.2.43 double smsize

SMPBE size

Definition at line 150 of file pbeparm.h.

9.6.2.44 double smvolume

SMPBE size

Definition at line 153 of file pbeparm.h.

9.6.2.45 double srad

Solvent radius

Definition at line 143 of file pbeparm.h.

9.6.2.46 Vsurf_Meth srfm

Surface calculation method

Definition at line 141 of file pbeparm.h.

9.6.2.47 double swin

Cubic spline window

Definition at line 145 of file pbeparm.h.

9.6.2.48 double temp

Temperature (in K)

Definition at line 147 of file pbeparm.h.

9.6.2.49 int useChargeMap

Indicates whether we use an external charge distribution map

Definition at line 122 of file pbeparm.h.

9.6.2.50 int useDielMap

Indicates whether we use external dielectric maps (note plural)

Definition at line 112 of file pbeparm.h.

9.6.2.51 int useKappaMap

Indicates whether we use an external kappa map

Definition at line 115 of file pbeparm.h.

9.6.2.52 int usePotMap

Indicates whether we use an external kappa map

Definition at line 118 of file pbeparm.h.

9.6.2.53 Vdata_Format writefmt[PBEPARM_MAXWRITE]

File format to write data in

Definition at line 180 of file pbeparm.h.

9.6.2.54 int writemat

Write out the operator matrix?

- 0 => no
- 1 => yes

Definition at line 182 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 187 of file pbeparm.h.

9.6.2.56 char writematstem[VMAX_ARGLEN]

File stem to write mat

Definition at line 186 of file pbeparm.h.

9.6.2.57 char writestem[PBEPARM_MAXWRITE][VMAX_ARGLEN]

File stem to write data to

Definition at line 177 of file pbeparm.h.

9.6.2.58 Vdata_Type writetype[PBEPARM_MAXWRITE]

What data to write

Definition at line 179 of file pbeparm.h.

9.6.2.59 double zmem

z value of membrane bottom

Definition at line 165 of file pbeparm.h.

The documentation for this struct was generated from the following file:

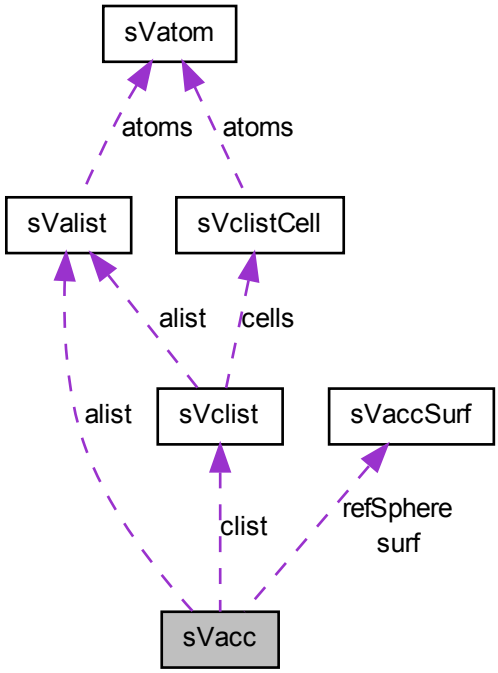
- src/generic/apbs/pbeparm.h

9.7 sVacc Struct Reference

Oracle for solvent- and ion-accessibility around a biomolecule.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/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 97 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 109 of file vacc.h.

9.7.2.2 Valist* alist

Valist structure for list of atoms

Definition at line 100 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 102 of file vacc.h.

9.7.2.4 Vclist* clist

Vclist structure for atom cell list

Definition at line 101 of file vacc.h.

9.7.2.5 Vmem* mem

Memory management object for this class

Definition at line 99 of file vacc.h.

9.7.2.6 VaccSurf* refSphere

Reference sphere for SASA calculations

Definition at line 105 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 106 of file vacc.h.

9.7.2.8 double surf_density

Minimum solvent accessible surface point density (in pts/A^2)

Definition at line 111 of file vacc.h.

The documentation for this struct was generated from the following file:

- src/generic/apbs/vacc.h

9.8 sVaccSurf Struct Reference

Surface object list of per-atom surface points.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/src/generic/apbs`

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 73 of file vacc.h.

9.8.2 Field Documentation

9.8.2.1 double area

Area spanned by these points

Definition at line 80 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 78 of file vacc.h.

9.8.2.3 Vmem* mem

Memory object

Definition at line 74 of file vacc.h.

9.8.2.4 int npts

Length of thee->xpts, ypts, zpts arrays

Definition at line 81 of file vacc.h.

9.8.2.5 double probe_radius

Probe radius (A) with which this surface was constructed

Definition at line 82 of file vacc.h.

9.8.2.6 double* xpts

Array of point x-locations

Definition at line 75 of file vacc.h.

9.8.2.7 double* ypts

Array of point y-locations

Definition at line 76 of file vacc.h.

9.8.2.8 double* zpts

Array of point z-locations

Definition at line 77 of file vacc.h.

The documentation for this struct was generated from the following file:

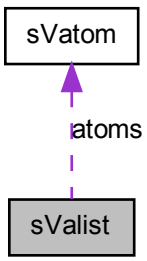
- src/generic/apbs/vacc.h

9.9 sValist Struct Reference

Container class for list of atom objects.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/src/generic/apb`

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 70 of file valist.h.

9.9.2 Field Documentation

9.9.2.1 Vatom* atoms

Atom list

Definition at line 78 of file valist.h.

9.9.2.2 double center[3]

Molecule center (xmin - xmax)/2, etc.

Definition at line 73 of file valist.h.

9.9.2.3 double charge

Net charge

Definition at line 77 of file valist.h.

9.9.2.4 double maxcrd[3]

Maximum coordinates

Definition at line 75 of file valist.h.

9.9.2.5 double maxrad

Maximum radius

Definition at line 76 of file valist.h.

9.9.2.6 double mincrd[3]

Minimum coordinates

Definition at line 74 of file valist.h.

9.9.2.7 int number

Number of atoms in list

Definition at line 72 of file valist.h.

9.9.2.8 Vmem* vmem

Memory management object

Definition at line 79 of file valist.h.

The documentation for this struct was generated from the following file:

- src/generic/apbs/valist.h

9.10 sVatom Struct Reference

Contains public data members for Vatom class/module.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/src/generic/apb`

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 73 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 87 of file vatom.h.

9.10.2.2 double charge

Atomic charge

Definition at line 77 of file vatom.h.

9.10.2.3 double epsilon

Epsilon value for WCA calculations

Definition at line 80 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 82 of file vatom.h.

9.10.2.5 double partID

Partition value for assigning atoms to particular processors and/or partitions

Definition at line 78 of file vatom.h.

9.10.2.6 double position[3]

Atomic position

Definition at line 75 of file vatom.h.

9.10.2.7 double radius

Atomic radius

Definition at line 76 of file vatom.h.

9.10.2.8 char resName[VMAX_RECLEN]

Residue name from PDB/PQR file

Definition at line 86 of file vatom.h.

The documentation for this struct was generated from the following file:

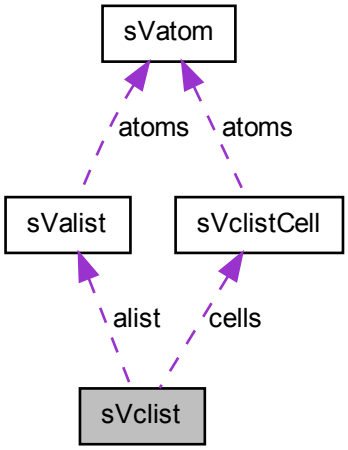
- src/generic/apbs/vatom.h

9.11 sVclist Struct Reference

Atom cell list.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/src/generic/apbs`

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 106 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 109 of file vclist.h.

9.11.2.2 VclistCell* cells

Cell array of length thee->n

Definition at line 114 of file vclist.h.

9.11.2.3 double lower_corner[VAPBS_DIM]

Hash table grid corner

Definition at line 115 of file vclist.h.

9.11.2.4 double max_radius

Maximum probe radius

Definition at line 113 of file vclist.h.

9.11.2.5 Vclist_DomainMode mode

How the cell list was constructed

Definition at line 110 of file vclist.h.

9.11.2.6 int n

n = nx*nz*ny

Definition at line 112 of file vclist.h.

9.11.2.7 int npts[VAPBS_DIM]

Hash table grid dimensions

Definition at line 111 of file vclist.h.

9.11.2.8 double spacs[VAPBS_DIM]

Hash table grid spacings

Definition at line 117 of file vclist.h.

9.11.2.9 double upper_corner[VAPBS_DIM]

Hash table grid corner

Definition at line 116 of file vclist.h.

9.11.2.10 Vmem* vmem

Memory management object for this class

Definition at line 108 of file vclist.h.

The documentation for this struct was generated from the following file:

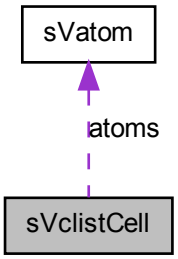
- src/generic/apbs/vclist.h

9.12 sVclistCell Struct Reference

Atom cell list cell.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/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 90 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 91 of file vclist.h.

9.12.2.2 int natoms

Length of thee->atoms array

Definition at line 92 of file vclist.h.

The documentation for this struct was generated from the following file:

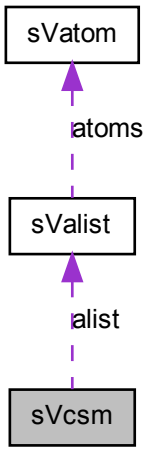
- src/generic/apbs/vclist.h

9.13 sVcsm Struct Reference

Charge-simplex map class.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/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 81 of file vcsm.h.

9.13.2 Field Documentation

9.13.2.1 Valist* alist

Atom (charge) list

Definition at line 83 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 86 of file vcsm.h.

9.13.2.3 int initFlag

Indicates whether the maps have been initialized yet

Definition at line 104 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 99 of file vcsm.h.

9.13.2.5 int natom

Size of thee->alist; redundant, but useful for convenience

Definition at line 84 of file vcsm.h.

9.13.2.6 int* nqsm

The length of the simplex lists in thee->qsm

Definition at line 103 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 97 of file vcsm.h.

9.13.2.8 int* nsqm

The length of the charge lists in thee->sqm

Definition at line 96 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 101 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->nsqm

Definition at line 89 of file vcsm.h.

9.13.2.11 Vmem* vmem

Memory management object

Definition at line 106 of file vcsm.h.

The documentation for this struct was generated from the following file:

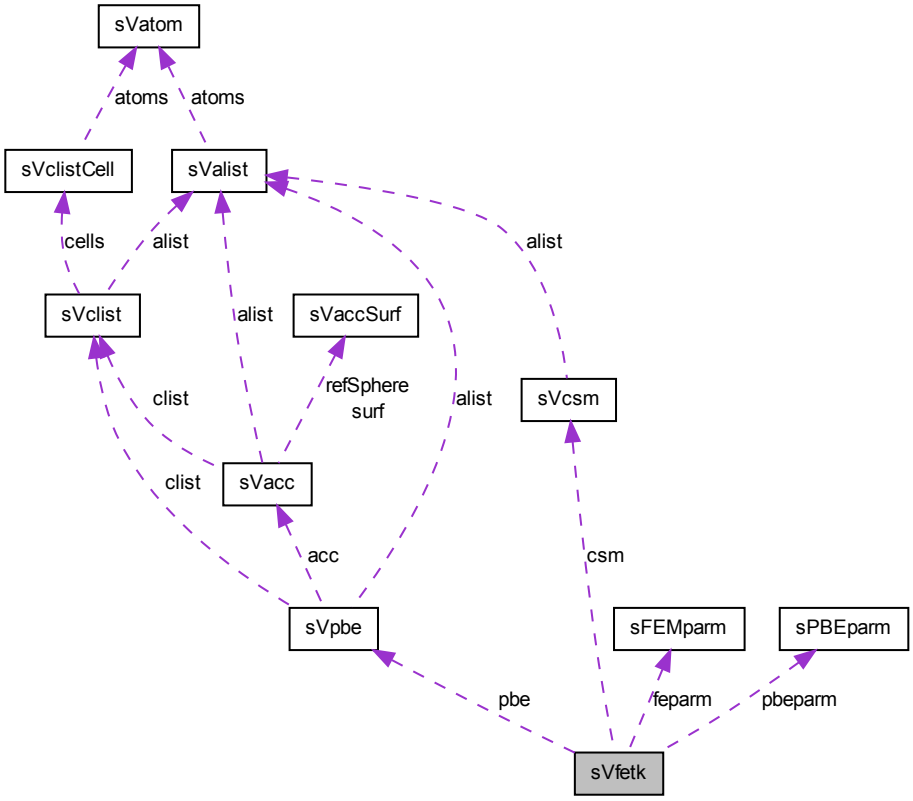
- src/fem/apbs/vcsm.h

9.14 sVfetk Struct Reference

Contains public data members for Vfetk class/module.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/src/fem/apbs/vf`

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 165 of file vfetk.h.

9.14.2 Field Documentation

9.14.2.1 AM* am

Multilevel algebra manager.

Definition at line 171 of file vfetk.h.

9.14.2.2 Aprx* aprx

Approximation manager.

Definition at line 172 of file vfetk.h.

9.14.2.3 Vcsm* csm

Charge-simplex map

Definition at line 175 of file vfetk.h.

9.14.2.4 FEMparm* feparm

FEM-specific parameters

Definition at line 187 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 168 of file vfetk.h.

9.14.2.6 Vfetk_GuessType gues

Initial guess method

Definition at line 182 of file vfetk.h.

9.14.2.7 int level

Refinement level (starts at 0)

Definition at line 189 of file vfetk.h.

9.14.2.8 Vfetk_LsolvType lkey

Linear solver method

Definition at line 176 of file vfetk.h.

9.14.2.9 int lmax

Maximum number of linear solver iterations

Definition at line 177 of file vfetk.h.

9.14.2.10 Vfetk_PrecType lprec

Linear preconditioner

Definition at line 183 of file vfetk.h.

9.14.2.11 double ltol

Residual tolerance for linear solver

Definition at line 178 of file vfetk.h.

9.14.2.12 Vfetk_NsolvType nkey

Nonlinear solver method

Definition at line 179 of file vfetk.h.

9.14.2.13 int nmax

Maximum number of nonlinear solver iterations

Definition at line 180 of file vfetk.h.

9.14.2.14 double ntol

Residual tolerance for nonlinear solver

Definition at line 181 of file vfetk.h.

9.14.2.15 Vpbe* pbe

Poisson-Boltzmann object

Definition at line 174 of file vfetk.h.

9.14.2.16 PBEparm* pbeparm

Generic PB parameters

Definition at line 186 of file vfetk.h.

9.14.2.17 PDE* pde

FEtk PDE object

Definition at line 173 of file vfetk.h.

9.14.2.18 int pjac

Flag to print the jacobians (usually set this to -1, please)

Definition at line 184 of file vfetk.h.

9.14.2.19 Vhal_PBEType type

Version of PBE to solve

Definition at line 188 of file vfetk.h.

9.14.2.20 Vmem* vmem

Memory management object

Definition at line 167 of file vfetk.h.

The documentation for this struct was generated from the following file:

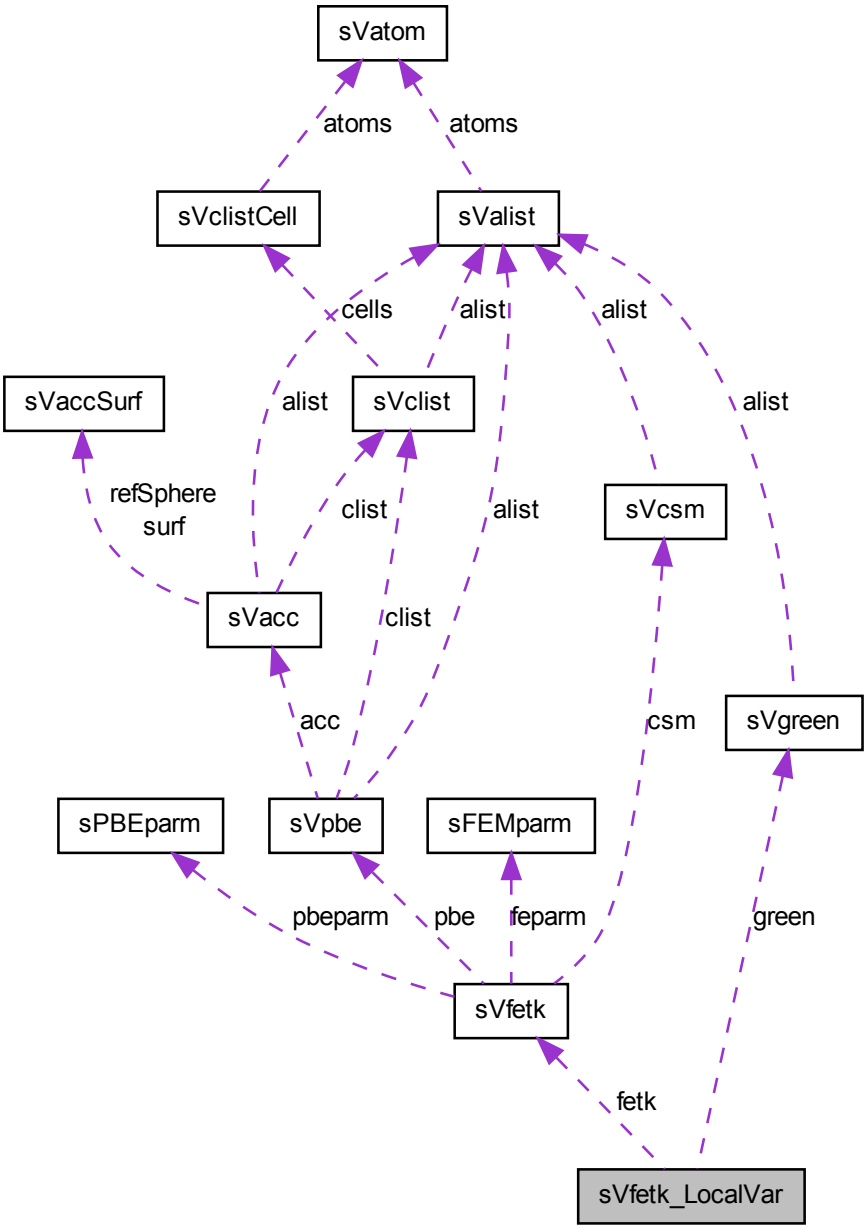
- src/fem/apbs/vfetk.h

9.15 sVfetk_LocalVar Struct Reference

Vfetk LocalVar subclass.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/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 204 of file vfetk.h.

9.15.2 Field Documentation

9.15.2.1 double A

Second-order differential term

Definition at line 217 of file vfetk.h.

9.15.2.2 double B

Entire ionic strength term

Definition at line 219 of file vfetk.h.

9.15.2.3 double d2W

Coulomb regularization term Laplacia

Definition at line 212 of file vfetk.h.

9.15.2.4 double DB

Entire ionic strength term derivative

Definition at line 220 of file vfetk.h.

9.15.2.5 double delta

Store delta value

Definition at line 239 of file vfetk.h.

9.15.2.6 double DFu_wv

Store DFu_wv value

Definition at line 238 of file vfetk.h.

9.15.2.7 double diel

Dielectric value

Definition at line 215 of file vfetk.h.

9.15.2.8 double dU[MAXV][VAPBS_DIM]

Solution gradient

Definition at line 209 of file vfetk.h.

9.15.2.9 double dW[VAPBS_DIM]

Coulomb regularization term gradient

Definition at line 211 of file vfetk.h.

9.15.2.10 double F

RHS characteristic function value

Definition at line 218 of file vfetk.h.

9.15.2.11 Vfetk* fetk

Pointer to the VFETK object

Definition at line 222 of file vfetk.h.

9.15.2.12 int fType

Face type

Definition at line 214 of file vfetk.h.

9.15.2.13 double Fu_v

Store Fu_v value

Definition at line 237 of file vfetk.h.

9.15.2.14 Vgreen* green

Pointer to a Green's function object

Definition at line 223 of file vfetk.h.

9.15.2.15 int initGreen

Boolean to designate whether Green's function has been initialized

Definition at line 224 of file vfetk.h.

9.15.2.16 double ionacc

Ion accessibility value

Definition at line 216 of file vfetk.h.

9.15.2.17 double ionConc[MAXION]

Counterion species' concentrations

Definition at line 230 of file vfetk.h.

9.15.2.18 double ionQ[MAXION]

Counterion species' valencies

Definition at line 231 of file vfetk.h.

9.15.2.19 double ionRadii[MAXION]

Counterion species' radii

Definition at line 232 of file vfetk.h.

9.15.2.20 double ionstr

Ionic strength parameters (M)

Definition at line 235 of file vfetk.h.

9.15.2.21 double jumpDiel

Dielectric value on one side of a simplex face

Definition at line 221 of file vfetk.h.

9.15.2.22 int nion

Number of ion species

Definition at line 236 of file vfetk.h.

9.15.2.23 double nvec[VAPBS_DIM]

Normal vector for a simplex face

Definition at line 205 of file vfetk.h.

9.15.2.24 int nverts

number of vertices in the simplex

Definition at line 229 of file vfetk.h.

9.15.2.25 SS* simp

Pointer to the latest simplex object; set in initElement() and delta()

Definition at line 226 of file vfetk.h.

9.15.2.26 int sType

Simplex type

Definition at line 213 of file vfetk.h.

9.15.2.27 double U[MAXV]

Solution value

Definition at line 208 of file vfetk.h.

9.15.2.28 double u_D

Store Dirichlet value

Definition at line 240 of file vfetk.h.

9.15.2.29 double u_T

Store true value

Definition at line 241 of file vfetk.h.

9.15.2.30 VV∗ verts[4]

Pointer to the latest vertices; set in initElement

Definition at line 228 of file vfetk.h.

9.15.2.31 double vx[4][VAPBS_DIM]

Vertex coordinates

Definition at line 206 of file vfetk.h.

9.15.2.32 double W

Coulomb regularization term scalar value

Definition at line 210 of file vfetk.h.

9.15.2.33 double xq[VAPBS_DIM]

Quadrature pt

Definition at line 207 of file vfetk.h.

9.15.2.34 double zkappa2

Ionic strength parameters

Definition at line 233 of file vfetk.h.

9.15.2.35 double zks2

Ionic strength parameters

Definition at line 234 of file vfetk.h.

The documentation for this struct was generated from the following file:

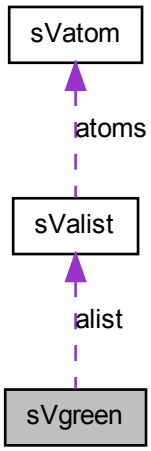
- src/fem/apbs/vfetk.h

9.16 sVgreen Struct Reference

Contains public data members for Vgreen class/module.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/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 75 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 77 of file vgreen.h.

9.16.2.2 int np

Set to size of above arrays

Definition at line 87 of file vgreen.h.

9.16.2.3 double* qp

Array of particle charges for use with treecode routines

Definition at line 85 of file vgreen.h.

9.16.2.4 Vmem* vmem

Memory management object

Definition at line 78 of file vgreen.h.

9.16.2.5 double* xp

Array of particle x-coordinates for use with treecode routines

Definition at line 79 of file vgreen.h.

9.16.2.6 double* yp

Array of particle y-coordinates for use with treecode routines

Definition at line 81 of file vgreen.h.

9.16.2.7 double* zp

Array of particle z-coordinates for use with treecode routines

Definition at line 83 of file vgreen.h.

The documentation for this struct was generated from the following file:

- src/generic/apbs/vgreen.h

9.17 sVgrid Struct Reference

Electrostatic potential oracle for Cartesian mesh data.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/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 72 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 88 of file vgrid.h.

9.17.2.2 double* data

nx*ny*nz array of data

Definition at line 86 of file vgrid.h.

9.17.2.3 double hx

Grid spacing in x direction

Definition at line 77 of file vgrid.h.

9.17.2.4 double hy

Grid spacing in y direction

Definition at line 78 of file vgrid.h.

9.17.2.5 double hzed

Grid spacing in z direction

Definition at line 79 of file vgrid.h.

9.17.2.6 Vmem* mem

Memory manager object

Definition at line 90 of file vgrid.h.

9.17.2.7 int nx

Number grid points in x direction

Definition at line 74 of file vgrid.h.

9.17.2.8 int ny

Number grid points in y direction

Definition at line 75 of file vgrid.h.

9.17.2.9 int nz

Number grid points in z direction

Definition at line 76 of file vgrid.h.

9.17.2.10 int readdata

flag indicating whether data was read from file

Definition at line 87 of file vgrid.h.

9.17.2.11 double xmax

x coordinate of upper grid corner

Definition at line 83 of file vgrid.h.

9.17.2.12 double xmin

x coordinate of lower grid corner

Definition at line 80 of file vgrid.h.

9.17.2.13 double ymax

y coordinate of upper grid corner

Definition at line 84 of file vgrid.h.

9.17.2.14 double ymin

y coordinate of lower grid corner

Definition at line 81 of file vgrid.h.

9.17.2.15 double zmax

z coordinate of upper grid corner

Definition at line 85 of file vgrid.h.

9.17.2.16 double zmin

z coordinate of lower grid corner

Definition at line 82 of file vgrid.h.

The documentation for this struct was generated from the following file:

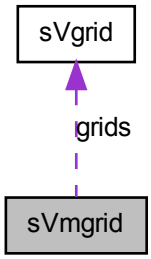
- src/mg/apbs/vgrid.h

9.18 sVmgrid Struct Reference

Multiresoltion oracle for Cartesian mesh data.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/src/mg/apbs/vmg`

Collaboration diagram for sVmgrid:



Data Fields

- int ngrids
- Vgrid ∗ grids [VMGRIDMAX]

9.18.1 Detailed Description

Multiresoltion oracle for Cartesian mesh data.

Author

Nathan Baker

Definition at line 76 of file vmgrid.h.

9.18.2 Field Documentation

9.18.2.1 Vgrid∗ grids[VMGRIDMAX]

Grids in hiearchy. 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 79 of file vmgrid.h.

9.18.2.2 int ngrids

Number of grids in hiearchy

Definition at line 78 of file vmgrid.h.

The documentation for this struct was generated from the following file:

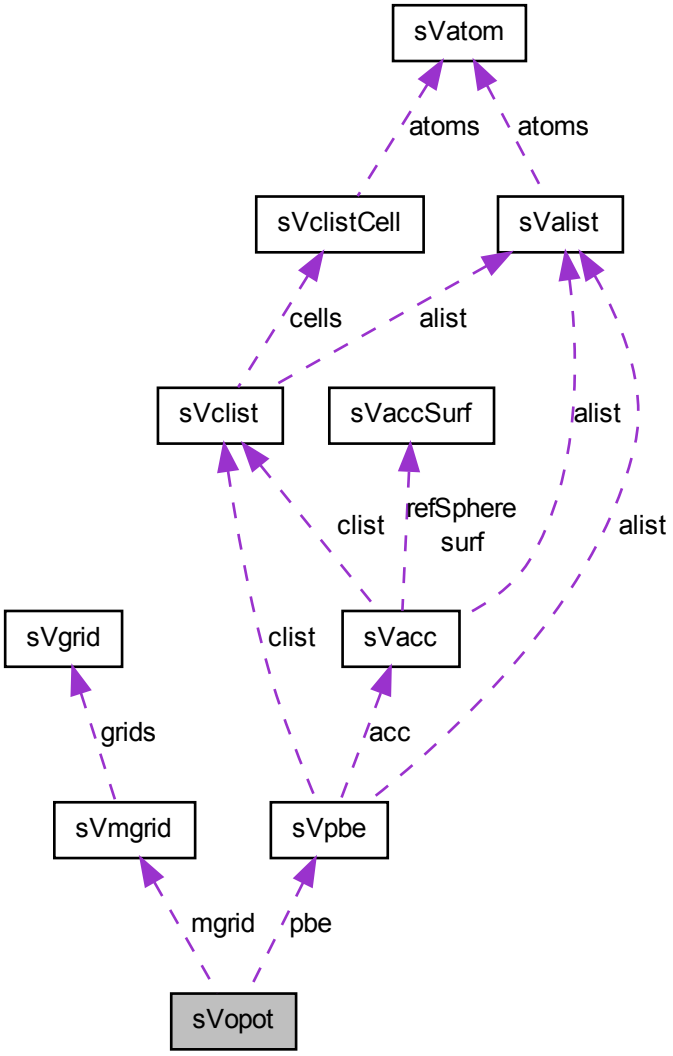
- src/mg/apbs/vmgrid.h

9.19 sVopot Struct Reference

Electrostatic potential oracle for Cartesian mesh data.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/src/mg/apbs/vop`

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 74 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 79 of file vopot.h.

9.19.2.2 Vmgrid* mgrid

Multiple grid object containing potential data (in units kT/e)

Definition at line 76 of file vopot.h.

9.19.2.3 Vpbe* pbe

Pointer to PBE object

Definition at line 78 of file vopot.h.

The documentation for this struct was generated from the following file:

- src/mg/apbs/vopot.h

9.20 sVparam_AtomData Struct Reference

AtomData sub-class; stores atom data.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/src/generic/apbs`

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:

$$V_{ij}(r_{ij}) = \varepsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - 2\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right]$$

where $\varepsilon_{ij} = \sqrt{\varepsilon_i \varepsilon_j}$ 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 79 of file vparam.h.

9.20.2 Field Documentation

9.20.2.1 char atomName[VMAX_ARGLEN]

Atom name

Definition at line 80 of file vparam.h.

9.20.2.2 double charge

Atom charge (in e)

Definition at line 82 of file vparam.h.

9.20.2.3 double epsilon

Atom VdW well depth (ε_i above; in kJ/mol)

Definition at line 84 of file vparam.h.

9.20.2.4 double radius

Atom VdW radius (σ_i above; in Å)

Definition at line 83 of file vparam.h.

9.20.2.5 char resName[VMAX_ARGLEN]

Residue name

Definition at line 81 of file vparam.h.

The documentation for this struct was generated from the following file:

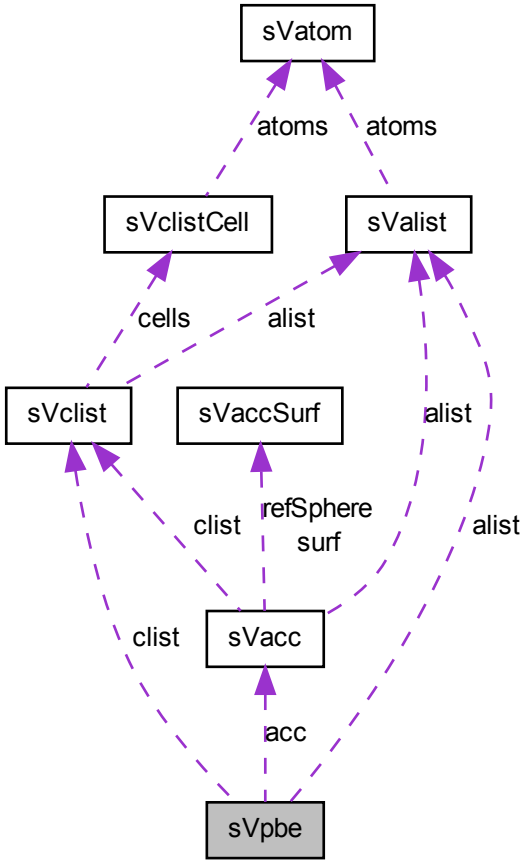
- src/generic/apbs/vparam.h

9.21 sVpbe Struct Reference

Contains public data members for Vpbe class/module.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/src/generic/apbs`

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 76 of file vpbe.h.

9.21.2 Field Documentation

9.21.2.1 Vacc* acc

Accessibility object

Definition at line 82 of file vpbe.h.

9.21.2.2 Valist* alist

Atom (charge) list

Definition at line 80 of file vpbe.h.

9.21.2.3 double bulkIonicStrength

Bulk ionic strength (M)

Definition at line 91 of file vpbe.h.

9.21.2.4 Vclist* clist

Atom location cell list

Definition at line 81 of file vpbe.h.

9.21.2.5 double deblen

Debye length (bulk)

Definition at line 101 of file vpbe.h.

9.21.2.6 double ionConc[MAXION]

Concentration (M) of each species

Definition at line 96 of file vpbe.h.

9.21.2.7 double ionQ[MAXION]

Charge (e) of each species

Definition at line 98 of file vpbe.h.

9.21.2.8 double ionRadii[MAXION]

Ionic radius (A) of each species

Definition at line 97 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 114 of file vpbe.h.

9.21.2.10 double L

Length of the membrane (A)

Definition at line 124 of file vpbe.h.

9.21.2.11 double maxIonRadius

Max ion radius (A; used for calculating accessiblity and defining volumes for ionic strength coeffcients)

Definition at line 92 of file vpbe.h.

9.21.2.12 double membraneDiel

Membrane dielectric constant

Definition at line 125 of file vpbe.h.

9.21.2.13 int numIon

Total number of ion species

Definition at line 95 of file vpbe.h.

9.21.2.14 int param2Flag

Check to see if bcfl=3 parms have been set

Definition at line 127 of file vpbe.h.

9.21.2.15 int paramFlag

Check to see if the parameters have been set

Definition at line 117 of file vpbe.h.

9.21.2.16 double smsize

Size-Modified PBE size

Definition at line 113 of file vpbe.h.

9.21.2.17 double smvolume

Size-Modified PBE relative volume

Definition at line 112 of file vpbe.h.

9.21.2.18 double soluteCenter[3]

Center of solute molecule (A)

Definition at line 105 of file vpbe.h.

9.21.2.19 double soluteCharge

Charge of solute molecule (e)

Definition at line 110 of file vpbe.h.

9.21.2.20 double soluteDiel

Solute dielectric constant (unitless)

Definition at line 85 of file vpbe.h.

9.21.2.21 double soluteRadius

Radius of solute molecule (A)

Definition at line 106 of file vpbe.h.

9.21.2.22 double soluteXlen

Solute length in x-direction

Definition at line 107 of file vpbe.h.

9.21.2.23 double soluteYlen

Solute length in y-direction

Definition at line 108 of file vpbe.h.

9.21.2.24 double soluteZlen

Solute length in z-direction

Definition at line 109 of file vpbe.h.

9.21.2.25 double solventDiel

Solvent dielectric constant (unitless)

Definition at line 86 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 88 of file vpbe.h.

9.21.2.27 double T

Temperature (K)

Definition at line 84 of file vpbe.h.

9.21.2.28 double V

Membrane potential

Definition at line 126 of file vpbe.h.

9.21.2.29 Vmem* vmem

Memory management object

Definition at line 78 of file vpbe.h.

9.21.2.30 double xkappa

Debye-Huckel parameter (bulk)

Definition at line 100 of file vpbe.h.

9.21.2.31 double z_mem

Z value of the botton of the membrane (A)

Definition at line 123 of file vpbe.h.

9.21.2.32 double zkappa2

Square of modified Debye-Huckel parameter (bulk)

Definition at line 102 of file vpbe.h.

9.21.2.33 double zmagic

Delta function scaling parameter

Definition at line 103 of file vpbe.h.

The documentation for this struct was generated from the following file:

- src/generic/apbs/vpbe.h

9.22 sVpee Struct Reference

Contains public data members for Vpee class/module.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/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 80 of file vpee.h.

9.22.2 Field Documentation

9.22.2.1 Gem* gm

Grid manager

Definition at line 82 of file vpee.h.

9.22.2.2 int killFlag

A flag indicating the method we're using to artificially decrease the error esimate outside the local partition

Definition at line 90 of file vpee.h.

9.22.2.3 double killParam

A parameter for the error estimate attenuation method

Definition at line 93 of file vpee.h.

9.22.2.4 double localPartCenter[3]

The coordinates of the center of the local partition

Definition at line 86 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 83 of file vpee.h.

9.22.2.6 double localPartRadius

The radius of the circle/sphere which circumscribes the local partition

Definition at line 88 of file vpee.h.

9.22.2.7 Vmem* mem

Memory manager

Definition at line 95 of file vpee.h.

The documentation for this struct was generated from the following file:

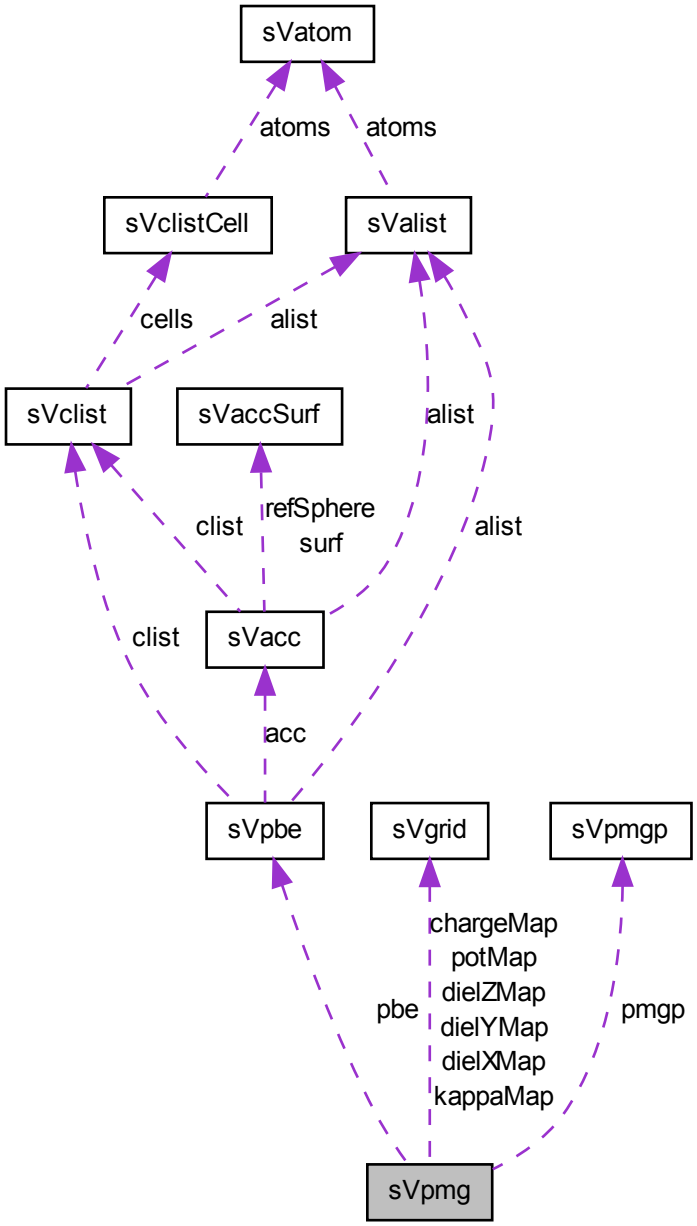
- src/fem/apbs/vpee.h

9.23 sVpmg Struct Reference

Contains public data members for Vpmg class/module.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/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 * dielZMap
- 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 88 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 105 of file vpmg.h.

9.23.2.2 double* a2cf

Operator coefficient values (a22) -- this array can be overwritten

Definition at line 107 of file vpmg.h.

9.23.2.3 double* a3cf

Operator coefficient values (a33) -- this array can be overwritten

Definition at line 109 of file vpmg.h.

9.23.2.4 double* ccf

Helmholtz term -- this array can be overwritten

Definition at line 111 of file vpmg.h.

9.23.2.5 double* charge

Charge map

Definition at line 99 of file vpmg.h.

9.23.2.6 Vgrid* chargeMap

External charge distribution map

Definition at line 155 of file vpmg.h.

9.23.2.7 Vchrg_Meth chargeMeth

Charge discretization method

Definition at line 132 of file vpmg.h.

9.23.2.8 Vchrg_Src chargeSrc

Charge source

Definition at line 133 of file vpmg.h.

9.23.2.9 Vgrid* dielXMap

External x-shifted dielectric map

Definition at line 139 of file vpmg.h.

9.23.2.10 Vgrid* dielYMap

External y-shifted dielectric map

Definition at line 142 of file vpmg.h.

9.23.2.11 Vgrid* dielZMap

External z-shifted dielectric map

Definition at line 145 of file vpmg.h.

9.23.2.12 double* epsx

X-shifted dielectric map

Definition at line 94 of file vpmg.h.

9.23.2.13 double* epsy

Y-shifted dielectric map

Definition at line 95 of file vpmg.h.

9.23.2.14 double* epsz

Y-shifted dielectric map

Definition at line 96 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 122 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 128 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 126 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 124 of file vpmg.h.

9.23.2.19 double* fcf

Right-hand side -- this array can be overwritten

Definition at line 112 of file vpmg.h.

9.23.2.20 int filled

Indicates whether Vpmg_fillco has been called

Definition at line 135 of file vpmg.h.

9.23.2.21 double* gxcf

Boundary conditions for x faces

Definition at line 118 of file vpmg.h.

9.23.2.22 double* gycf

Boundary conditions for y faces

Definition at line 119 of file vpmg.h.

9.23.2.23 double* gzcf

Boundary conditions for z faces

Definition at line 120 of file vpmg.h.

9.23.2.24 int* iparm

Passing int parameters to FORTRAN

Definition at line 101 of file vpmg.h.

9.23.2.25 int* iwork

Work array

Definition at line 103 of file vpmg.h.

9.23.2.26 double* kappa

Ion accessibility map (0 <= kappa(x) <= 1)

Definition at line 97 of file vpmg.h.

9.23.2.27 Vgrid* kappaMap

External kappa map

Definition at line 148 of file vpmg.h.

9.23.2.28 Vpbe* pbe

Information about the PBE system

Definition at line 92 of file vpmg.h.

9.23.2.29 Vpmgp* pmgp

Parameters

Definition at line 91 of file vpmg.h.

9.23.2.30 double* pot

Potential map

Definition at line 98 of file vpmg.h.

9.23.2.31 Vgrid* potMap

External potential map

Definition at line 151 of file vpmg.h.

9.23.2.32 double* pvec

Partition mask array

Definition at line 121 of file vpmg.h.

9.23.2.33 double* rparm

Passing real parameters to FORTRAN

Definition at line 102 of file vpmg.h.

9.23.2.34 double* rwork

Work array

Definition at line 104 of file vpmg.h.

9.23.2.35 double splineWin

Spline window parm for surf defs

Definition at line 131 of file vpmg.h.

9.23.2.36 Vsurf_Meth surfMeth

Surface definition method

Definition at line 130 of file vpmg.h.

9.23.2.37 double* tcf

True solution

Definition at line 113 of file vpmg.h.

9.23.2.38 double* u

Solution

Definition at line 114 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 153 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 137 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 140 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 143 of file vpmg.h.

9.23.2.43 int useKappaMap

Indicates whether Vpmg_fillco was called with an external kappa map

Definition at line 146 of file vpmg.h.

9.23.2.44 int usePotMap

Indicates whether Vpmg_fillco was called with an external potential map

Definition at line 149 of file vpmg.h.

9.23.2.45 Vmem* vmem

Memory management object for this class

Definition at line 90 of file vpmg.h.

9.23.2.46 double* xf

Mesh point x coordinates

Definition at line 115 of file vpmg.h.

9.23.2.47 double* yf

Mesh point y coordinates

Definition at line 116 of file vpmg.h.

9.23.2.48 double* zf

Mesh point z coordinates

Definition at line 117 of file vpmg.h.

The documentation for this struct was generated from the following file:

- src/mg/apbs/vpmg.h

9.24 sVpmgp Struct Reference

Contains public data members for Vpmgp class/module.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/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 70 of file vpmgp.h.

9.24.2 Field Documentation

9.24.2.1 Vbcfl bcfl

Boundary condition method [default = BCFL_SDH]

Definition at line 125 of file vpmgp.h.

9.24.2.2 double errtol

Desired error tolerance [default = 1e-9]

Definition at line 111 of file vpmgp.h.

9.24.2.3 double hx

Grid x spacings [no default]

Definition at line 77 of file vpmgp.h.

9.24.2.4 double hy

Grid y spacings [no default]

Definition at line 78 of file vpmgp.h.

9.24.2.5 double hzed

Grid z spacings [no default]

Definition at line 79 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 120 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 173 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 129 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 99 of file vpmgp.h.

9.24.2.10 int irite

FORTRAN output unit [default = 8]

Definition at line 172 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 113 of file vpmgp.h.

9.24.2.12 int itmax

Maximum number of iters [default = 100]

Definition at line 112 of file vpmgp.h.

9.24.2.13 int key

Print solution to file [default = 0]

- 0: no
- 1: yes

Definition at line 126 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: sucessive overrelaxation
- 5: red-black gauss-seidel
- 6: weighted jacobi
- 7: richardson
- 8: conjugate gradient multigrid aqua
- 9: newton aqua

Definition at line 134 of file vpmgp.h.

9.24.2.15 int mgcoar

Coarsening method [default = 2]

- 0: standard
- 1: harmonic
- 2: galerkin

Definition at line 160 of file vpmgp.h.

9.24.2.16 int mgdisc

Discretization method [default = 0]

- 0: finite volume
- 1: finite element

Definition at line 167 of file vpmgp.h.

9.24.2.17 int mgkey

Multigrid method [default = 0]

- 0: variable v-cycle
- 1: nested iteration

Definition at line 145 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 156 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: cghs

Definition at line 150 of file vpmgp.h.

9.24.2.20 int mgsolv

Coarse equation solve method [default = 1]

- 0: cghs
- 1: banded linpack

Definition at line 164 of file vpmgp.h.

9.24.2.21 int n_ipc

Integer info work array required storage

Definition at line 94 of file vpmgp.h.

9.24.2.22 int n_iz

Integer storage parameter (index max)

Definition at line 93 of file vpmgp.h.

9.24.2.23 int n_rpc

Real info work array required storage

Definition at line 92 of file vpmgp.h.

9.24.2.24 int narr

Array work storage

Definition at line 98 of file vpmgp.h.

9.24.2.25 int narrc

Size of vector on coarse level

Definition at line 91 of file vpmgp.h.

9.24.2.26 int nc

Number of coarse grid unknowns

Definition at line 90 of file vpmgp.h.

9.24.2.27 int nf

Number of fine grid unknowns

Definition at line 89 of file vpmgp.h.

9.24.2.28 int niwk

Integer work storage

Definition at line 97 of file vpmgp.h.

9.24.2.29 int nlev

Number of mesh levels [no default]

Definition at line 76 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 80 of file vpmgp.h.

9.24.2.31 int nrwk

Real work storage

Definition at line 96 of file vpmgp.h.

9.24.2.32 int nu1

Number of pre-smoothings [default = 2]

Definition at line 148 of file vpmgp.h.

9.24.2.33 int nu2

Number of post-smoothings [default = 2]

Definition at line 149 of file vpmgp.h.

9.24.2.34 int nx

Grid x dimensions [no default]

Definition at line 73 of file vpmgp.h.

9.24.2.35 int nxc

Coarse level grid x dimensions

Definition at line 86 of file vpmgp.h.

9.24.2.36 int ny

Grid y dimensions [no default]

Definition at line 74 of file vpmgp.h.

9.24.2.37 int nyc

Coarse level grid y dimensions

Definition at line 87 of file vpmgp.h.

9.24.2.38 int nz

Grid z dimensions [no default]

Definition at line 75 of file vpmgp.h.

9.24.2.39 int nzc

Coarse level grid z dimensions

Definition at line 88 of file vpmgp.h.

9.24.2.40 double omegal

Linear relax parameter [default = 8e-1]

Definition at line 170 of file vpmgp.h.

9.24.2.41 double omegan

Nonlin relax parameter [default = 9e-1]

Definition at line 171 of file vpmgp.h.

9.24.2.42 double xcent

Grid x center [0]

Definition at line 108 of file vpmgp.h.

9.24.2.43 double xlen

Domain x length

Definition at line 179 of file vpmgp.h.

9.24.2.44 double xmax

Domain upper x corner

Definition at line 185 of file vpmgp.h.

9.24.2.45 double xmin

Domain lower x corner

Definition at line 182 of file vpmgp.h.

9.24.2.46 double ycent

Grid y center [0]

Definition at line 109 of file vpmgp.h.

9.24.2.47 double ylen

Domain y length

Definition at line 180 of file vpmgp.h.

9.24.2.48 double ymax

Domain upper y corner

Definition at line 186 of file vpmgp.h.

9.24.2.49 double ymin

Domain lower y corner

Definition at line 183 of file vpmgp.h.

9.24.2.50 double zcent

Grid z center [0]

Definition at line 110 of file vpmgp.h.

9.24.2.51 double zlen

Domain z length

Definition at line 181 of file vpmgp.h.

9.24.2.52 double zmax

Domain upper z corner

Definition at line 187 of file vpmgp.h.

9.24.2.53 double zmin

Domain lower z corner

Definition at line 184 of file vpmgp.h.

The documentation for this struct was generated from the following file:

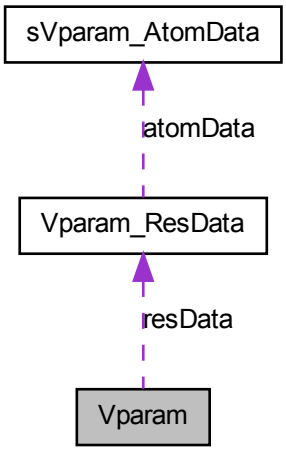
- src/mg/apbs/vpmgp.h

9.25 Vparam Struct Reference

Reads and assigns charge/radii parameters.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/src/generic/apbs`

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 122 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 125 of file vparam.h.

9.25.2.2 Vparam_ResData* resData

Array of nResData Vparam_ResData objects

Definition at line 127 of file vparam.h.

9.25.2.3 Vmem* vmem

Memory management object for this class

Definition at line 124 of file vparam.h.

The documentation for this struct was generated from the following file:

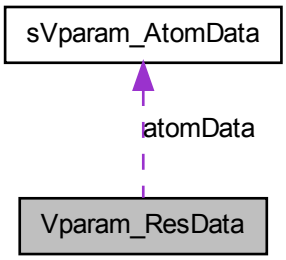
- src/generic/apbs/vparam.h

9.26 Vparam_ResData Struct Reference

ResData sub-class; stores residue data.

`#include <C:/Users/bake113/Desktop/Sync/Software/apbs/trunk/src/generic/apb`

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 101 of file vparam.h.

9.26.2 Field Documentation

9.26.2.1 Vparam_AtomData* atomData

Array of Vparam_AtomData natom objects

Definition at line 106 of file vparam.h.

9.26.2.2 char name[VMAX_ARGLEN]

Residue name

Definition at line 103 of file vparam.h.

9.26.2.3 int nAtomData

Number of Vparam_AtomData objects associated with this object

Definition at line 104 of file vparam.h.

9.26.2.4 Vmem* vmem

Pointer to memory manager from Vparam master class

Definition at line 102 of file vparam.h.

The documentation for this struct was generated from the following file:

- src/generic/apbs/vparam.h

Chapter 10

File Documentation

10.1 doc/license/LICENSE.h File Reference

APBS license.

10.1.1 Detailed Description

APBS license.

Author

Nathan Baker

Version

Id:

LICENSE.h 1552 2010-02-10 17:46:27Z yhuang01

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010.  Pacific Northwest National Laboratory.
* Portions Copyright (c) 2002-2010.  Washington University in St. Louis.
```

Definition in file LICENSE.h.

10.2 doc/license/LICENSE.h

```
00001
```

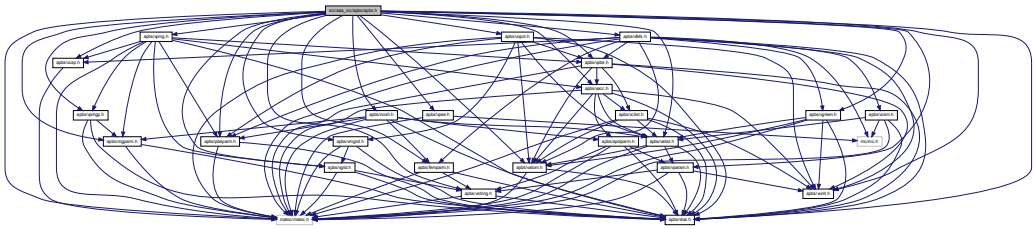
10.3 src/aaa_inc/apbs/apbs.h File Reference

Top-level header for APBS.

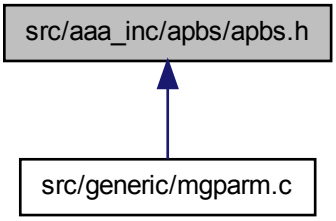
```
#include "maloc/maloc.h"
#include "apbs/femparm.h"
#include "apbs/mgparm.h"
#include "apbs/nosh.h"
```

```
#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 1552 2010-02-10 17:46:27Z yhuang01

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
* 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 Washington University in St. Louis 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 src/aaa_inc/apbs/apbs.h

```
00001
00049 #ifndef _APBS_H_
00050 #define _APBS_H_
00051
00052 /* MALOC headers */
00053 #include "maloc/maloc.h"
00054
00055 /* Generic headers */
00056 #include "apbs/femparm.h"
00057 #include "apbs/mgparm.h"
00058 #include "apbs/nosh.h"
00059 #include "apbs/pbeparm.h"
00060 #include "apbs/vacc.h"
00061 #include "apbs/valist.h"
00062 #include "apbs/vatom.h"
00063 #include "apbs/vcap.h"
00064 #include "apbs/vgreen.h"
00065 #include "apbs/vhal.h"
00066 #include "apbs/vpbe.h"
00067 #include "apbs/vstring.h"
00068 #include "apbs/vunit.h"
00069 #include "apbs/vparam.h"
00070
00071 /* MG headers */
00072 #include "apbs/vgrid.h"
00073 #include "apbs/vmgrid.h"
```

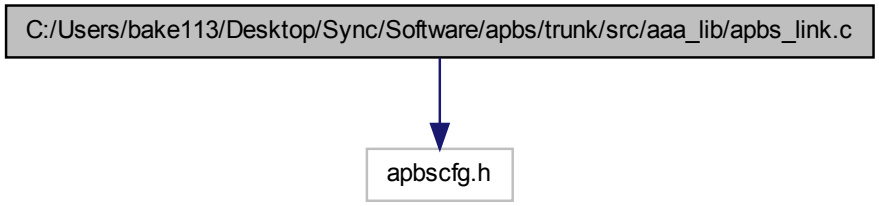
```
00074 #include "apbs/vopot.h"
00075 #include "apbs/vpmg.h"
00076 #include "apbs/vpmgp.h"
00077
00078 /* FEM headers */
00079 #if defined(HAVE_MC_H)
00080 #   include "apbs/vfetk.h"
00081 #   include "apbs/vpee.h"
00082 #endif
00083
00084 #endif /* ifndef _APBS_H_ */
```

10.5 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 1552 2010-02-10 17:46:27Z yhuang01

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
* 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 Washington University in St. Louis 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 src/aaa_lib/apbs_link.c

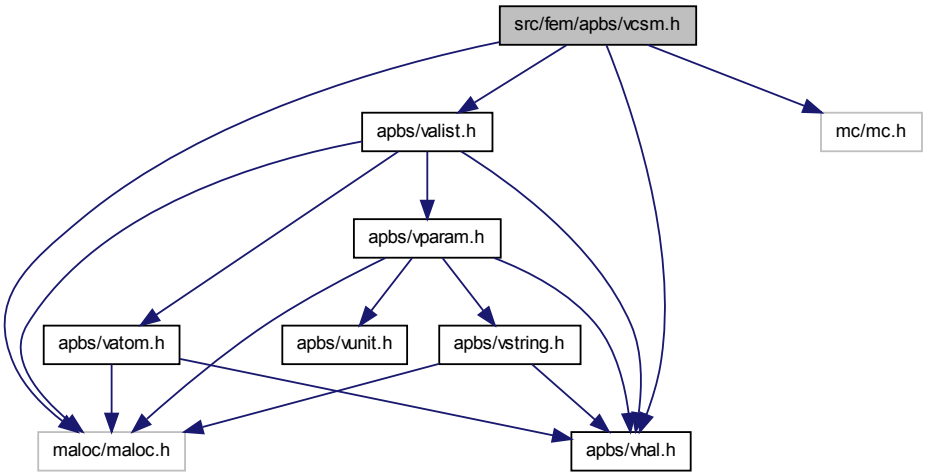
```
00001 #include "apbscfg.h"
00002
00050 #if defined(HAVE_MC_H)
00051     void apbs_needs_mc(void) { }
00052 #endif
00053 #if !defined(USE_PMG_BLAS)
00054     void apbs_needs_blas(void) { }
00055 #endif
00056
00057 void apbs_link(void)
00058 {
00059 }
00060
```

10.7 src/fem/apbs/vcsm.h File Reference

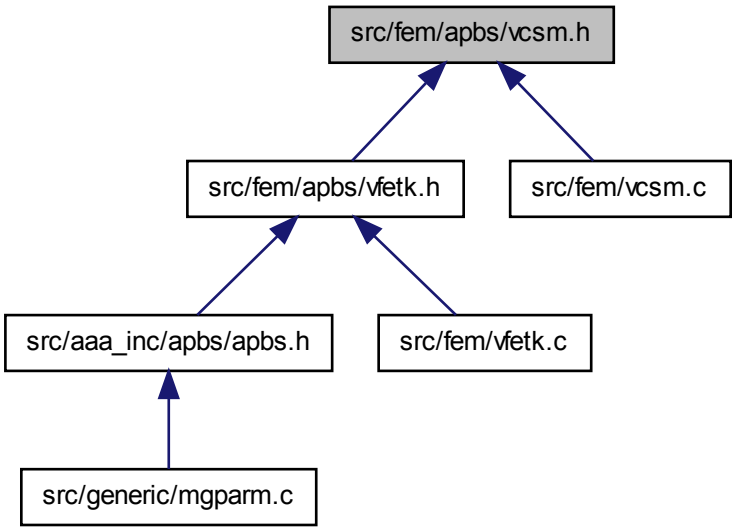
Contains declarations for the Vcsm class.

```
#include "maloc/maloc.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 **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.

10.7.1 Detailed Description

Contains declarations for the Vcsm class.

Version

Id:

vcsm.h 1565 2010-03-07 16:06:27Z sobolevnrm

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
* 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 Washington University in St. Louis 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 src/fem/apbs/vcsm.h

```
00001
00055 #ifndef _VCSM_H_
00056 #define _VCSM_H_
00057
00058 /* Generic headers */
00059 #include "maloc/maloc.h"
00060 #include "apbs/vhal.h"
00061 #include "apbs/valist.h"
00062
00063 /* Specific headers */
00064 #include "mc/mc.h"
00065
00070 VEXTERNC void Gem_setExternalUpdateFunction(
00071         Gem *thee,
00072         void (*externalUpdate)(SS **simps, int num)
00075         );
00076
00081 struct sVcsm {
00082
00083   Valist *alist;
00084   int natom;
00086   Gem *gm;
00089   int **sqm;
00096   int *nsqm;
00097   int nsimp;
00099   int msimp;
00101   int **qsm;
00103   int *nqsm;
00104   int initFlag;
00106   Vmem *vmem;
00108 };
00109
00114 typedef struct sVcsm Vcsm;
00115
00116 /* ////////////////////////////////////////////////////////////////////////
00117 // Class Vcsm: Inlineable methods (vcsm.c)
00119
00120 #if !defined(VINLINE_VCSM)
00121
00127     VEXTERNC Valist* Vcsm_getValist(
00128             Vcsm *thee
00129             );
00130
00136     VEXTERNC int Vcsm_getNumberAtoms(
00137             Vcsm *thee,
00138             int isimp
00139             );
00140
00146     VEXTERNC Vatom* Vcsm_getAtom(
00147             Vcsm *thee,
```

```
00148             int iatom,
00149             int isimp
00150             );
00151
00157     VEXTERNC int Vcsm_getAtomIndex(
00158             Vcsm *thee,
00159             int iatom,
00160             int isimp
00161             );
00162
00168     VEXTERNC int Vcsm_getNumberSimplices(
00169             Vcsm *thee,
00170             int iatom
00171             );
00172
00178     VEXTERNC SS* Vcsm_getSimplex(
00179             Vcsm *thee,
00180             int isimp,
00181             int iatom
00182             );
00183
00189     VEXTERNC int Vcsm_getSimplexIndex(
00190             Vcsm *thee,
00191             int isimp,
00192             int iatom
00193             );
00194
00201     VEXTERNC unsigned long int Vcsm_memChk(
00202             Vcsm *thee
00203             );
00204
00205 #else /* if defined(VINLINE_VCSM) */
00206 #   define Vcsm_getValist(thee) ((thee)->alist)
00207 #   define Vcsm_getNumberAtoms(thee, isimp) ((thee)->nsqm[isimp])
00208 #   define Vcsm_getAtom(thee, iatom, isimp) (Valist_getAtom((thee)->alist, ((thee
      )->sqm)[isimp][iatom]))
00209 #   define Vcsm_getAtomIndex(thee, iatom, isimp) (((thee)->sqm)[isimp][iatom])
00210 #   define Vcsm_getNumberSimplices(thee, iatom) (((thee)->nqsm)[iatom])
00211 #   define Vcsm_getSimplex(thee, isimp, iatom) (Gem_SS((thee)->gm, ((thee)->qsm)[
      iatom][isimp]))
00212 #   define Vcsm_getSimplexIndex(thee, isimp, iatom) (((thee)->qsm)[iatom][isimp])

00213 #   define Vcsm_memChk(thee) (Vmem_bytes((thee)->vmem))
00214 #endif /* if !defined(VINLINE_VCSM) */
00215
00216 /* ///////////////////////////////////////////////////////////////////////////
00217 // Class Vcsm: Non-Inlineable methods (vcsm.c)
00219
00228 VEXTERNC Vcsm* Vcsm_ctor(
00229         Valist *alist,
00230         Gem *gm
00231         );
00232
00241 VEXTERNC int Vcsm_ctor2(
00242         Vcsm *thee,
00243         Valist *alist,
00244         Gem *gm
```

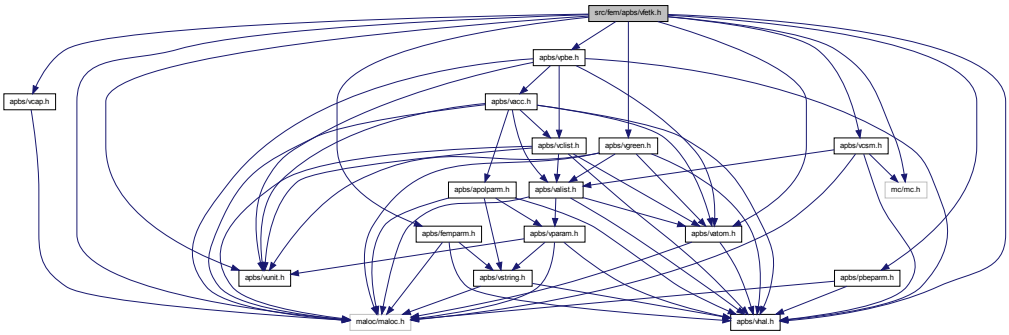
```
00245         );
00246
00251 VEXTERNC void Vcsm_dtor(
00252         Vcsm **thee
00253         );
00254
00259 VEXTERNC void Vcsm_dtor2(
00260         Vcsm *thee
00261         );
00262
00269 VEXTERNC void Vcsm_init(
00270         Vcsm *thee
00271         );
00272
00279 VEXTERNC int Vcsm_update(
00280         Vcsm *thee,
00281         SS **simps,
00286         int num
00287         );
00288
00289 #endif /* ifndef _VCSM_H_ */
```

10.9 src/fem/apbs/vfetk.h File Reference

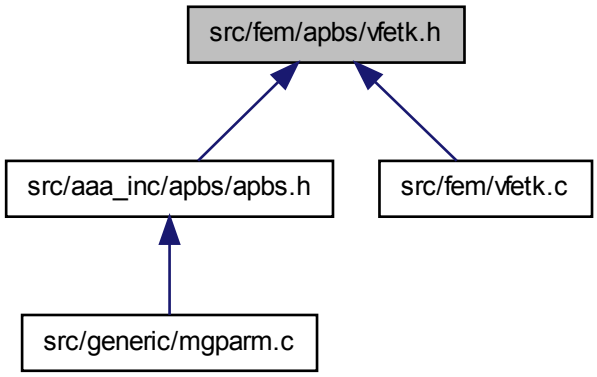
Contains declarations for class Vfetk.

```
#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 "apbs/femparm.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_BT/e) at the finest mesh level.

- VEXTERNC void Vfetk_setParameters (Vfetk *thee, PBEparm *pbeparm, FEMparm *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 frmo 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)

  Intializes 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 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:

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

  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:

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

  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:

  $$\int_\Omega \left[\varepsilon \nabla w \cdot \nabla v + b'(u)wv - fv\right] dx$$

  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:

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

  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 1565 2010-03-07 16:06:27Z sobolevnrm

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
```

```
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
* 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 Washington University in St. Louis 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 src/fem/apbs/vfetk.h

```
00001
00054 #ifndef _VFETK_H_
00055 #define _VFETK_H_
00056
00057 #include "maloc/maloc.h"
00058 #include "mc/mc.h"
00059 #include "apbs/vhal.h"
00060 #include "apbs/vatom.h"
00061 /* #include "apbs/valist.h" */
00062 #include "apbs/vcsm.h"
00063 #include "apbs/vpbe.h"
00064 #include "apbs/vunit.h"
00065 #include "apbs/vgreen.h"
00066 #include "apbs/vcap.h"
00067 #include "apbs/pbeparm.h"
00068 #include "apbs/femparm.h"
00069
00075 enum eVfetk_LsolvType {
```

```
00076     VLT_SLU=0,
00077     VLT_MG=1,
00078     VLT_CG=2,
00079     VLT_BCG=3
00080 };
00081
00086 typedef enum eVfetk_LsolvType Vfetk_LsolvType;
00087
00088
00093 enum eVfetk_MeshLoad {
00094     VML_DIRICUBE,
00095     VML_NEUMCUBE,
00096     VML_EXTERNAL
00097 };
00098
00103 typedef enum eVfetk_MeshLoad Vfetk_MeshLoad;
00104
00110 enum eVfetk_NsolvType {
00111     VNT_NEW=0,
00112     VNT_INC=1,
00113     VNT_ARC=2
00114 };
00115
00120 typedef enum eVfetk_NsolvType Vfetk_NsolvType;
00121
00127 enum eVfetk_GuessType {
00128     VGT_ZERO=0,
00129     VGT_DIRI=1,
00130     VGT_PREV=2
00131 };
00132
00137 typedef enum eVfetk_GuessType Vfetk_GuessType;
00138
00144 enum eVfetk_PrecType {
00145     VPT_IDEN=0,
00146     VPT_DIAG=1,
00147     VPT_MG=2
00148 };
00149
00154 typedef enum eVfetk_PrecType Vfetk_PrecType;
00155
00165 struct sVfetk {
00166
00167   Vmem *vmem;
00168   Gem *gm;
00171   AM *am;
00172   Aprx *aprx;
00173   PDE *pde;
00174   Vpbe *pbe;
00175   Vcsm *csm;
00176   Vfetk_LsolvType lkey;
00177   int lmax;
00178   double ltol;
00179   Vfetk_NsolvType nkey;
00180   int nmax;
00181   double ntol;
00182   Vfetk_GuessType gues;
```

```
00183   Vfetk_PrecType lprec;
00184   int pjac;
00186   PBEparm *pbeparm;
00187   FEMparm *feparm;
00188   Vhal_PBEType type;
00189   int level;
00191 };
00192
00196 typedef struct sVfetk Vfetk;
00197
00204 struct sVfetk_LocalVar {
00205     double nvec[VAPBS_DIM];
00206     double vx[4][VAPBS_DIM];
00207     double xq[VAPBS_DIM];
00208     double U[MAXV];
00209     double dU[MAXV][VAPBS_DIM];
00210     double W;
00211     double dW[VAPBS_DIM];
00212     double d2W;
00213     int sType;
00214     int fType;
00215     double diel;
00216     double ionacc;
00217     double A;
00218     double F;
00219     double B;
00220     double DB;
00221     double jumpDiel;
00222     Vfetk *fetk;
00223     Vgreen *green;
00224     int initGreen;
00226     SS *simp;
00228     VV *verts[4];
00229     int nverts;
00230     double ionConc[MAXION];
00231     double ionQ[MAXION];
00232     double ionRadii[MAXION];
00233     double zkappa2;
00234     double zks2;
00235     double ionstr;
00236     int nion;
00237     double Fu_v;
00238     double DFu_wv;
00239     double delta;
00240     double u_D;
00241     double u_T;
00242 };
00243
00248 typedef struct sVfetk_LocalVar Vfetk_LocalVar;
00249
00250 #if !defined(VINLINE_VFETK)
00251
00257     VEXTERNC Gem* Vfetk_getGem(
00258             Vfetk *thee
00259             );
00260
00266     VEXTERNC AM* Vfetk_getAM(
```

```
00267             Vfetk *thee
00268             );
00269
00275     VEXTERNC Vpbe* Vfetk_getVpbe(
00276             Vfetk *thee
00277             );
00278
00284     VEXTERNC Vcsm* Vfetk_getVcsm(
00285             Vfetk *thee
00286             );
00287
00294     VEXTERNC int Vfetk_getAtomColor(
00295             Vfetk *thee,
00296             int iatom
00297             );
00298
00299 #else /* if defined(VINLINE_VFETK) */
00300 #   define Vfetk_getGem(thee) ((thee)->gm)
00301 #   define Vfetk_getAM(thee) ((thee)->am)
00302 #   define Vfetk_getVpbe(thee) ((thee)->pbe)
00303 #   define Vfetk_getVcsm(thee) ((thee)->csm)
00304 #   define Vfetk_getAtomColor(thee, iatom) (Vatom_getPartID(Valist_getAtom(Vpbe_g
    etValist(thee->pbe), iatom)))
00305 #endif /* if !defined(VINLINE_VFETK) */
00306
00307 /* ////////////////////////////////////////////////////////////////////////////
00308 // Class Vfetk: Non-Inlineable methods (vfetk.c)
00310
00320 VEXTERNC Vfetk* Vfetk_ctor(
00321         Vpbe *pbe,
00322         Vhal_PBEType type
00323         );
00324
00334 VEXTERNC int Vfetk_ctor2(
00335         Vfetk *thee,
00336         Vpbe *pbe,
00337         Vhal_PBEType type
00338         );
00339
00345 VEXTERNC void Vfetk_dtor(
00346         Vfetk **thee
00347         );
00348
00354 VEXTERNC void Vfetk_dtor2(
00355         Vfetk *thee
00356         );
00357
00367 VEXTERNC double* Vfetk_getSolution(
00368         Vfetk *thee,
00369         int *length
00370         );
00371
00377 VEXTERNC void Vfetk_setParameters(
00378         Vfetk *thee,
00379         PBEparm *pbeparm,
00380         FEMparm *feparm
00381         );
```

```
00382
00401 VEXTERNC double  Vfetk_energy(
00402         Vfetk *thee,
00403         int color,
00407         int nonlin
00409         );
00410
00440 VEXTERNC double  Vfetk_dqmEnergy(
00441         Vfetk *thee,
00442         int color
00446         );
00447
00465 VEXTERNC double Vfetk_qfEnergy(
00466         Vfetk *thee,
00467         int color
00469         );
00470
00478 VEXTERNC unsigned long int Vfetk_memChk(
00479         Vfetk *thee
00480         );
00481
00497 VEXTERNC void Vfetk_setAtomColors(
00498         Vfetk *thee
00499         );
00500
00509 VEXTERNC void Bmat_printHB(
00510         Bmat *thee,
00511         char *fname
00512         );
00513
00519 VEXTERNC Vrc_Codes Vfetk_genCube(
00520          Vfetk *thee,
00521          double center[3],
00522          double length[3],
00523          Vfetk_MeshLoad meshType
00524          );
00525
00531 VEXTERNC Vrc_Codes Vfetk_loadMesh(
00532           Vfetk *thee,
00533           double center[3],
00534           double length[3],
00535           Vfetk_MeshLoad meshType,
00536           Vio *sock
00537           );
00538
00545 VEXTERNC PDE* Vfetk_PDE_ctor(
00546         Vfetk *fetk
00547         );
00548
00555 VEXTERNC int Vfetk_PDE_ctor2(
00556         PDE *thee,
00557         Vfetk *fetk
00558         );
00559
00566 VEXTERNC void Vfetk_PDE_dtor(
00567         PDE **thee
00568         );
```

```
00569
00576 VEXTERNC void Vfetk_PDE_dtor2(
00577         PDE *thee
00578         );
00579
00585 VEXTERNC void Vfetk_PDE_initAssemble(
00586         PDE *thee,
00587         int ip[],
00588         double rp[]
00589         );
00590
00597 VEXTERNC void Vfetk_PDE_initElement(
00598         PDE *thee,
00599         int elementType,
00600         int chart,
00603         double tvx[][VAPBS_DIM],
00604         void *data
00605         );
00606
00612 VEXTERNC void Vfetk_PDE_initFace(
00613         PDE *thee,
00614         int faceType,
00616         int chart,
00618         double tnvec[]
00619         );
00620
00628 VEXTERNC void Vfetk_PDE_initPoint(
00629         PDE *thee,
00630         int pointType,
00631         int chart,
00633         double txq[],
00634         double tU[],
00635         double tdU[][VAPBS_DIM]
00636         );
00637
00655 VEXTERNC void Vfetk_PDE_Fu(
00656         PDE *thee,
00657         int key,
00659         double F[]
00660         );
00661
00672 VEXTERNC double Vfetk_PDE_Fu_v(
00673         PDE *thee,
00674         int key,
00676         double V[],
00677         double dV[][VAPBS_DIM]
00678         );
00679
00691 VEXTERNC double Vfetk_PDE_DFu_wv(
00692         PDE *thee,
00693         int key,
00695         double W[],
00696         double dW[][VAPBS_DIM],
00697         double V[],
00698         double dV[][VAPBS_DIM]
00699         );
00700
```

```
00707 VEXTERNC void Vfetk_PDE_delta(
00708         PDE *thee,
00709         int type,
00710         int chart,
00711         double txq[],
00712         void *user,
00713         double F[]
00714         );
00715
00723 VEXTERNC void Vfetk_PDE_u_D(
00724         PDE *thee,
00725         int type,
00726         int chart,
00727         double txq[],
00728         double F[]
00729         );
00730
00738 VEXTERNC void Vfetk_PDE_u_T(
00739         PDE *thee,
00740         int type,
00741         int chart,
00742         double txq[],
00743         double F[]
00744         );
00745
00751 VEXTERNC void Vfetk_PDE_bisectEdge(
00752         int dim,
00753         int dimII,
00754         int edgeType,
00755         int chart[],
00757         double vx[][VAPBS_DIM]
00758         );
00759
00765 VEXTERNC void Vfetk_PDE_mapBoundary(
00766         int dim,
00767         int dimII,
00768         int vertexType,
00769         int chart,
00770         double vx[VAPBS_DIM]
00771         );
00772
00781 VEXTERNC int Vfetk_PDE_markSimplex(
00782         int dim,
00783         int dimII,
00784         int simplexType,
00785         int faceType[VAPBS_NVS],
00786         int vertexType[VAPBS_NVS],
00787         int chart[],
00788         double vx[][VAPBS_DIM],
00789         void *simplex
00790         );
00791
00797 VEXTERNC void Vfetk_PDE_oneChart(
00798         int dim,
00799         int dimII,
00800         int objType,
00801         int chart[],
```

```
00802         double vx[][VAPBS_DIM],
00803         int dimV
00804         );
00805
00815 VEXTERNC double Vfetk_PDE_Ju(
00816         PDE *thee,
00817         int key
00818         );
00819
00827 VEXTERNC void Vfetk_externalUpdateFunction(
00828         SS **simps,
00830         int num
00831         );
00832
00833
00896 VEXTERNC int Vfetk_PDE_simplexBasisInit(
00897         int key,
00899         int dim,
00900         int comp,
00902         int *ndof,
00903         int dof[]
00904         );
00905
00913 VEXTERNC void Vfetk_PDE_simplexBasisForm(
00914         int key,
00916         int dim,
00917         int comp ,
00918         int pdkey,
00927         double xq[],
00928         double basis[]
00930         );
00931
00937 VEXTERNC void Vfetk_readMesh(
00938         Vfetk *thee,
00939         int skey,
00940         Vio *sock
00941         );
00942
00948 VEXTERNC void Vfetk_dumpLocalVar();
00949
00957 VEXTERNC int Vfetk_fillArray(
00958         Vfetk *thee,
00959         Bvec *vec,
00960         Vdata_Type type
00961         );
00962
00977 VEXTERNC int Vfetk_write(
00978         Vfetk *thee,
00979         const char *iodev,
00981         const char *iofmt,
00983         const char *thost,
00984         const char *fname,
00985         Bvec *vec,
00986         Vdata_Format format
00987         );
00988
00994 VEXTERNC Vrc_Codes Vfetk_loadGem(
```

```
00995           Vfetk *thee,
00996           Gem *gm
00997           );
00998
00999
01000 #endif /* ifndef _VFETK_H_ */
```

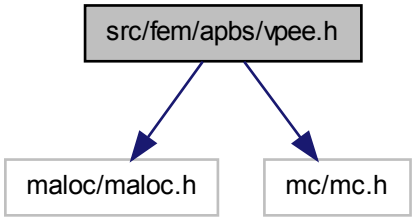
10.11 src/fem/apbs/vpee.h File Reference

Contains declarations for class Vpee.

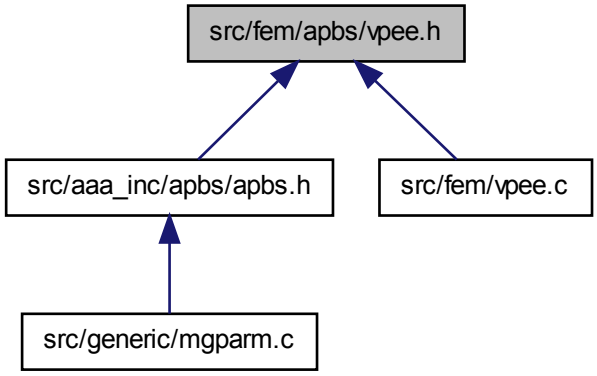
```
#include "maloc/maloc.h"
```

```
#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 1565 2010-03-07 16:06:27Z sobolevnrm

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
* 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 Washington University in St. Louis 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 src/fem/apbs/vpee.h

```
00001
00068 #ifndef _VPEE_H
00069 #define _VPEE_H
00070
00071 /* Generic headers */
00072 #include "maloc/maloc.h"
00073 #include "mc/mc.h"
00074
00080 struct sVpee {
00081
00082   Gem *gm;
00083   int localPartID;
00086   double localPartCenter[3];
00088   double localPartRadius;
00090   int killFlag;
00093   double killParam;
00095   Vmem *mem;
00097 };
00098
00103 typedef struct sVpee Vpee;
00104
00105 /* ////////////////////////////////////////////////////////////////////////
00106 // Class Vpee Inlineable methods
00108
00109 #if !defined(VINLINE_VPEE)
00110 #else /* if defined(VINLINE_VPEE) */
```

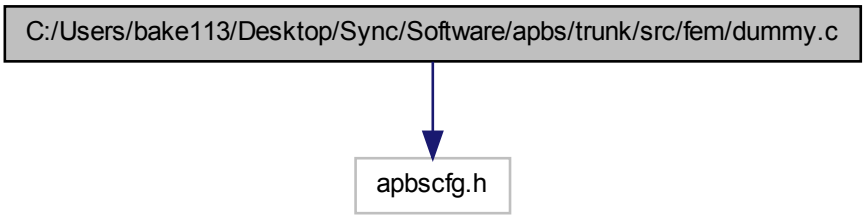
```
00111 #endif /* if !defined(VINLINE_VPEE) */
00112
00113 /* ////////////////////////////////////////////////////////////////////////
00114 // Class Vpee: Non-Inlineable methods (vpee.c)
00116
00123 VEXTERNC Vpee* Vpee_ctor(
00124         Gem *gm,
00125         int localPartID,
00126         int killFlag,
00137         double killParam
00138         );
00139
00146 VEXTERNC int Vpee_ctor2(
00147         Vpee *thee,
00148         Gem *gm,
00149         int localPartID,
00150         int killFlag,
00161         double killParam
00162         );
00163
00168 VEXTERNC void Vpee_dtor(
00169         Vpee **thee
00170         );
00171
00176 VEXTERNC void Vpee_dtor2(
00177         Vpee *thee
00178         );
00179
00195 VEXTERNC int Vpee_markRefine(
00196         Vpee *thee,
00197         AM *am,
00198         int level,
00199         int akey,
00207         int rcol,
00210         double etol,
00211         int bkey
00215         );
00216
00222 VEXTERNC int Vpee_numSS(
00223         Vpee *thee
00224         );
00225
00226 #endif    /* ifndef _VPEE_H_ */
```

10.13 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 1552 2010-02-10 17:46:27Z yhuang01

Attention

```
 *
 * APBS -- Adaptive Poisson-Boltzmann Solver
 *
 * Nathan A. Baker (nathan.baker@pnl.gov)
 * Pacific Northwest National Laboratory
 *
 * Additional contributing authors listed in the code documentation.
 *
 * Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
```

```
 * 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 Washington University in St. Louis 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 src/fem/dummy.c

```
00001
00048 #include "apbscfg.h"
00049
00050 int APBSFEM_dummy(int i) {
00051
00052     int j;
00053
00054     j = i;
00055
00056     return j;
00057 }
```

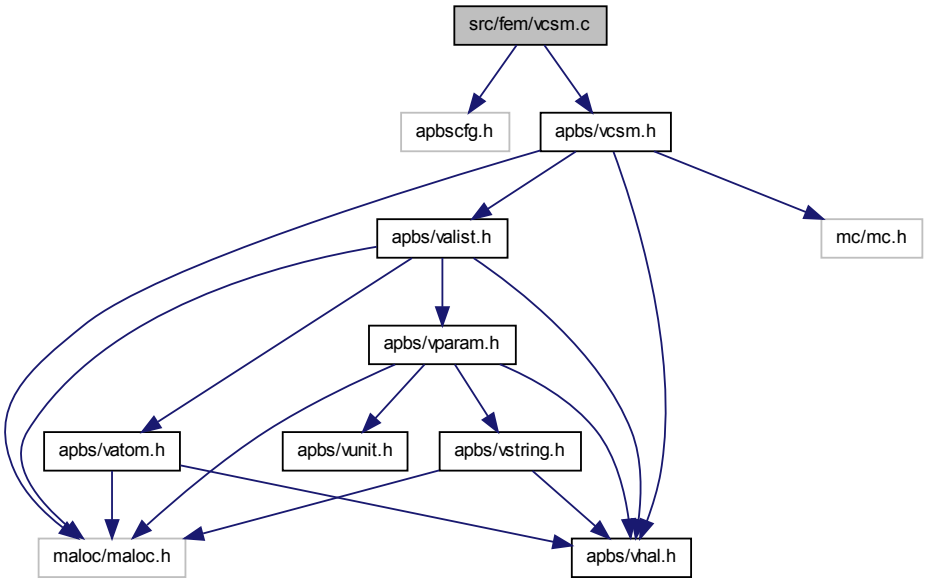
10.15 src/fem/vcsm.c File Reference

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 1552 2010-02-10 17:46:27Z yhuang01

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
* 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 Washington University in St. Louis 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 src/fem/vcsm.c

```
00001
00050 #include "apbscfg.h"
00051
00052 #if defined(HAVE_MC_H)
00053 #include "apbs/vcsm.h"
00054
00055 /* Inlineable methods */
00056 #if !defined(VINLINE_VCSM)
00057
```

```
00058 VPUBLIC Valist* Vcsm_getValist(Vcsm *thee) {
00059
00060    VASSERT(thee != VNULL);
00061    return thee->alist;
00062
00063 }
00064
00065 VPUBLIC int Vcsm_getNumberAtoms(Vcsm *thee, int isimp) {
00066
00067    VASSERT(thee != VNULL);
00068    VASSERT(thee->initFlag);
00069    return thee->nsqm[isimp];
00070
00071 }
00072
00073 VPUBLIC Vatom* Vcsm_getAtom(Vcsm *thee, int iatom, int isimp) {
00074
00075
00076    VASSERT(thee != VNULL);
00077    VASSERT(thee->initFlag);
00078
00079    VASSERT(iatom < (thee->nsqm)[isimp]);
00080    return Valist_getAtom(thee->alist, (thee->sqm)[isimp][iatom]);
00081
00082 }
00083
00084 VPUBLIC int Vcsm_getAtomIndex(Vcsm *thee, int iatom, int isimp) {
00085
00086
00087    VASSERT(thee != VNULL);
00088    VASSERT(thee->initFlag);
00089
00090    VASSERT(iatom < (thee->nsqm)[isimp]);
00091    return (thee->sqm)[isimp][iatom];
00092
00093 }
00094
00095 VPUBLIC int Vcsm_getNumberSimplices(Vcsm *thee, int iatom) {
00096
00097
00098    VASSERT(thee != VNULL);
00099    VASSERT(thee->initFlag);
00100
00101    return (thee->nqsm)[iatom];
00102
00103 }
00104
00105 VPUBLIC SS* Vcsm_getSimplex(Vcsm *thee, int isimp, int iatom) {
00106
00107
00108    VASSERT(thee != VNULL);
00109    VASSERT(thee->initFlag);
00110
00111    return Gem_SS(thee->gm, (thee->qsm)[iatom][isimp]);
00112
00113 }
00114
```

```
00115 VPUBLIC int Vcsm_getSimplexIndex(Vcsm *thee, int isimp, int iatom) {
00116
00117
00118    VASSERT(thee != VNULL);
00119    VASSERT(thee->initFlag);
00120
00121    return (thee->qsm)[iatom][isimp];
00122
00123 }
00124
00125 VPUBLIC unsigned long int Vcsm_memChk(Vcsm *thee) {
00126     if (thee == VNULL) return 0;
00127     return Vmem_bytes(thee->vmem);
00128 }
00129
00130 #endif /* if !defined(VINLINE_VCSM) */
00131
00132 VPUBLIC Vcsm* Vcsm_ctor(Valist *alist, Gem *gm) {
00133
00134     /* Set up the structure */
00135     Vcsm *thee = VNULL;
00136     thee = Vmem_malloc(VNULL, 1, sizeof(Vcsm) );
00137     VASSERT( thee != VNULL);
00138     VASSERT( Vcsm_ctor2(thee, alist, gm));
00139
00140     return thee;
00141 }
00142
00143 VPUBLIC int Vcsm_ctor2(Vcsm *thee, Valist *alist, Gem *gm) {
00144
00145     VASSERT( thee != VNULL );
00146
00147     /* Memory management object */
00148     thee->vmem = Vmem_ctor("APBS:VCSM");
00149
00150     /* Set up the atom list and grid manager */
00151     if( alist == VNULL) {
00152         Vnm_print(2,"Vcsm_ctor2: got null pointer to Valist object!\n");
00153         return 0;
00154     }
00155     thee->alist = alist;
00156     if( gm == VNULL) {
00157         Vnm_print(2,"Vcsm_ctor2: got a null pointer to the Gem object!\n");
00158         return 0;
00159     }
00160     thee->gm = gm;
00161
00162     thee->initFlag = 0;
00163     return 1;
00164 }
00165
00166 VPUBLIC void Vcsm_init(Vcsm *thee) {
00167
00168     /* Counters */
00169     int iatom, jatom, isimp, jsimp, gotSimp;
00170     /* Atomic information */
00171     Vatom *atom;
```

```
00172     double *position;
00173     /* Simplex/Vertex information */
00174     SS *simplex;
00175     /* Basis function values */
00176
00177     if (thee == VNULL) {
00178         Vnm_print(2, "Vcsm_init:  Error!  Got NULL thee!\n");
00179         VASSERT(0);
00180     }
00181     if (thee->gm == VNULL) {
00182     VASSERT(thee->gm != VNULL);
00183         Vnm_print(2, "Vcsm_init:  Error!  Got NULL thee->gm!\n");
00184         VASSERT(0);
00185     }
00186     thee->nsimp = Gem_numSS(thee->gm);
00187     if (thee->nsimp <= 0) {
00188         Vnm_print(2, "Vcsm_init:  Error!  Got %d simplices!\n", thee->nsimp);
00189         VASSERT(0);
00190     }
00191     thee->natom = Valist_getNumberAtoms(thee->alist);
00192
00193     /* Allocate and initialize space for the first dimensions of the
00194      * simplex-charge map, the simplex array, and the counters */
00195     thee->sqm = Vmem_malloc(thee->vmem, thee->nsimp, sizeof(int *));
00196     VASSERT(thee->sqm != VNULL);
00197     thee->nsqm = Vmem_malloc(thee->vmem, thee->nsimp, sizeof(int));
00198     VASSERT(thee->nsqm != VNULL);
00199     for (isimp=0; isimp<thee->nsimp; isimp++) (thee->nsqm)[isimp] = 0;
00200
00201     /* Count the number of charges per simplex. */
00202     for (iatom=0; iatom<thee->natom; iatom++) {
00203         atom = Valist_getAtom(thee->alist, iatom);
00204         position = Vatom_getPosition(atom);
00205         gotSimp = 0;
00206         for (isimp=0; isimp<thee->nsimp; isimp++) {
00207             simplex = Gem_SS(thee->gm, isimp);
00208             if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00209                 (thee->nsqm)[isimp]++;
00210                 gotSimp = 1;
00211              }
00212         }
00213     }
00214
00215     /* Allocate the space for the simplex-charge map */
00216     for (isimp=0; isimp<thee->nsimp; isimp++) {
00217         if ((thee->nsqm)[isimp] > 0) {
00218             thee->sqm[isimp] = Vmem_malloc(thee->vmem, (thee->nsqm)[isimp],
00219               sizeof(int));
00220             VASSERT(thee->sqm[isimp] != VNULL);
00221         }
00222     }
00223
00224     /* Finally, set up the map */
00225     for (isimp=0; isimp<thee->nsimp; isimp++) {
00226         jsimp = 0;
00227         simplex = Gem_SS(thee->gm, isimp);
00228         for (iatom=0; iatom<thee->natom; iatom++) {
```

```
00229             atom = Valist_getAtom(thee->alist, iatom);
00230             position = Vatom_getPosition(atom);
00231             /* Check to see if the atom's in this simplex */
00232             if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00233                 /* Assign the entries in the next vacant spot */
00234                 (thee->sqm)[isimp][jsimp] = iatom;
00235                 jsimp++;
00236             }
00237         }
00238     }
00239 
00240     thee->msimp = thee->nsimp;
00241 
00242     /* Allocate space for the charge-simplex map */
00243     thee->qsm = Vmem_malloc(thee->vmem, thee->natom, sizeof(int *));
00244     VASSERT(thee->qsm != VNULL);
00245     thee->nqsm = Vmem_malloc(thee->vmem, thee->natom, sizeof(int));
00246     VASSERT(thee->nqsm != VNULL);
00247     for (iatom=0; iatom<thee->natom; iatom++) (thee->nqsm)[iatom] = 0;
00248     /* Loop through the list of simplices and count the number of times
00249      * each atom appears */
00250     for (isimp=0; isimp<thee->nsimp; isimp++) {
00251         for (iatom=0; iatom<thee->nsqm[isimp]; iatom++) {
00252             jatom = thee->sqm[isimp][iatom];
00253             thee->nqsm[jatom]++;
00254         }
00255     }
00256     /* Do a TIME-CONSUMING SANITY CHECK to make sure that each atom was
00257      * placed in at simplex */
00258     for (iatom=0; iatom<thee->natom; iatom++) {
00259         if (thee->nqsm[iatom] == 0) {
00260             Vnm_print(2, "Vcsm_init: Atom %d not placed in simplex!\n", iatom);
00261             VASSERT(0);
00262         }
00263     }
00264     /* Allocate the appropriate amount of space for each entry in the
00265      * charge-simplex map and clear the counter for re-use in assignment */
00266     for (iatom=0; iatom<thee->natom; iatom++) {
00267         thee->qsm[iatom] = Vmem_malloc(thee->vmem, (thee->nqsm)[iatom],
00268           sizeof(int));
00269         VASSERT(thee->qsm[iatom] != VNULL);
00270         thee->nqsm[iatom] = 0;
00271     }
00272     /* Assign the simplices to atoms */
00273     for (isimp=0; isimp<thee->nsimp; isimp++) {
00274         for (iatom=0; iatom<thee->nsqm[isimp]; iatom++) {
00275             jatom = thee->sqm[isimp][iatom];
00276             thee->qsm[jatom][thee->nqsm[jatom]] = isimp;
00277             thee->nqsm[jatom]++;
00278         }
00279     }
00280 
00281     thee->initFlag = 1;
00282 }
00283 
00284 VPUBLIC void Vcsm_dtor(Vcsm **thee) {
00285     if ((*thee) != VNULL) {
```

```
00286         Vcsm_dtor2(*thee);
00287         Vmem_free(VNULL, 1, sizeof(Vcsm), (void **)thee);
00288         (*thee) = VNULL;
00289     }
00290 }
00291
00292 VPUBLIC void Vcsm_dtor2(Vcsm *thee) {
00293     int i;
00294
00295     if ((thee != VNULL) && thee->initFlag) {
00296
00297         for (i=0; i<thee->msimp; i++) {
00298             if (thee->nsqm[i] > 0) Vmem_free(thee->vmem, thee->nsqm[i],
00299               sizeof(int), (void **)&(thee->sqm[i]));
00300         }
00301         for (i=0; i<thee->natom; i++) {
00302             if (thee->nqsm[i] > 0) Vmem_free(thee->vmem, thee->nqsm[i],
00303               sizeof(int), (void **)&(thee->qsm[i]));
00304         }
00305         Vmem_free(thee->vmem, thee->msimp, sizeof(int *),
00306           (void **)&(thee->sqm));
00307         Vmem_free(thee->vmem, thee->msimp, sizeof(int),
00308           (void **)&(thee->nsqm));
00309         Vmem_free(thee->vmem, thee->natom, sizeof(int *),
00310           (void **)&(thee->qsm));
00311         Vmem_free(thee->vmem, thee->natom, sizeof(int),
00312           (void **)&(thee->nqsm));
00313
00314     }
00315     Vmem_dtor(&(thee->vmem));
00316 }
00317
00318 VPUBLIC int Vcsm_update(Vcsm *thee, SS **simps, int num) {
00319
00320     /* Counters */
00321     int isimp, jsimp, iatom, jatom, atomID, simpID;
00322     int nsimps, gotMem;
00323     /* Object info */
00324     Vatom *atom;
00325     SS *simplex;
00326     double *position;
00327     /* Lists */
00328     int *qParent, nqParent;
00329     int **sqmNew, *nsqmNew;
00330     int *affAtoms, nAffAtoms;
00331     int *dnqsm, *nqsmNew, **qsmNew;
00332
00333     VASSERT(thee != VNULL);
00334     VASSERT(thee->initFlag);
00335
00336     /* If we don't have enough memory to accommodate the new entries,
00337      * add more by doubling the existing amount */
00338     isimp = thee->nsimp + num - 1;
00339     gotMem = 0;
00340     while (!gotMem) {
00341         if (isimp > thee->msimp) {
00342             isimp = 2 * isimp;
```

```
00343             thee->nsqm = Vmem_realloc(thee->vmem, thee->msimp, sizeof(int),
00344               (void **)&(thee->nsqm), isimp);
00345             VASSERT(thee->nsqm != VNULL);
00346             thee->sqm = Vmem_realloc(thee->vmem, thee->msimp, sizeof(int *),
00347               (void **)&(thee->sqm), isimp);
00348             VASSERT(thee->sqm != VNULL);
00349             thee->msimp = isimp;
00350         } else gotMem = 1;
00351     }
00352     /* Initialize the nsqm entires we just allocated */
00353     for (isimp = thee->nsimp; isimp<thee->nsimp+num-1 ; isimp++) {
00354        thee->nsqm[isimp] = 0;
00355     }
00356
00357     thee->nsimp = thee->nsimp + num - 1;
00358
00359     /* There's a simple case to deal with:  if simps[0] didn't have a
00360      * charge in the first place */
00361     isimp = SS_id(simps[0]);
00362     if (thee->nsqm[isimp] == 0) {
00363         for (isimp=1; isimp<num; isimp++) {
00364             thee->nsqm[SS_id(simps[isimp])] = 0;
00365         }
00366         return 1;
00367     }
00368
00369     /* The more complicated case has occured; the parent simplex had one or
00370      * more charges.  First, generate the list of affected charges. */
00371     isimp = SS_id(simps[0]);
00372     nqParent = thee->nsqm[isimp];
00373     qParent = thee->sqm[isimp];
00374
00375     sqmNew = Vmem_malloc(thee->vmem, num, sizeof(int *));
00376     VASSERT(sqmNew != VNULL);
00377     nsqmNew = Vmem_malloc(thee->vmem, num, sizeof(int));
00378     VASSERT(nsqmNew != VNULL);
00379     for (isimp=0; isimp<num; isimp++) nsqmNew[isimp] = 0;
00380
00381     /* Loop throught the affected atoms to determine how many atoms each
00382      * simplex will get. */
00383     for (iatom=0; iatom<nqParent; iatom++) {
00384
00385         atomID = qParent[iatom];
00386         atom = Valist_getAtom(thee->alist, atomID);
00387         position = Vatom_getPosition(atom);
00388         nsimps = 0;
00389
00390         jsimp = 0;
00391
00392         for (isimp=0; isimp<num; isimp++) {
00393             simplex = simps[isimp];
00394             if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00395                 nsqmNew[isimp]++;
00396                 jsimp = 1;
00397             }
00398         }
00399
```

```
00400         VASSERT(jsimp != 0);
00401     }
00402
00403     /* Sanity check that we didn't lose any atoms... */
00404     iatom = 0;
00405     for (isimp=0; isimp<num; isimp++) iatom += nsqmNew[isimp];
00406     if (iatom < nqParent) {
00407         Vnm_print(2,"Vcsm_update: Lost %d (of %d) atoms!\n",
00408             nqParent - iatom, nqParent);
00409         VASSERT(0);
00410     }
00411
00412     /* Allocate the storage */
00413     for (isimp=0; isimp<num; isimp++) {
00414         if (nsqmNew[isimp] > 0) {
00415             sqmNew[isimp] = Vmem_malloc(thee->vmem, nsqmNew[isimp],
00416               sizeof(int));
00417             VASSERT(sqmNew[isimp] != VNULL);
00418         }
00419     }
00420
00421     /* Assign charges to simplices */
00422     for (isimp=0; isimp<num; isimp++) {
00423
00424         jsimp = 0;
00425         simplex = simps[isimp];
00426
00427         /* Loop over the atoms associated with the parent simplex */
00428         for (iatom=0; iatom<nqParent; iatom++) {
00429
00430             atomID = qParent[iatom];
00431             atom = Valist_getAtom(thee->alist, atomID);
00432             position = Vatom_getPosition(atom);
00433             if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00434                 sqmNew[isimp][jsimp] = atomID;
00435                 jsimp++;
00436             }
00437         }
00438     }
00439
00440     /* Update the QSM map using the old and new SQM lists */
00441     /* The affected atoms are those contained in the parent simplex; i.e.
00442      * thee->sqm[SS_id(simps[0])] */
00443     affAtoms = thee->sqm[SS_id(simps[0])];
00444     nAffAtoms = thee->nsqm[SS_id(simps[0])];
00445     /* Each of these atoms will go somewhere else; i.e., the entries in
00446      * thee->qsm are never destroyed and thee->nqsm never decreases.
00447      * However, it is possible that a subdivision could cause an atom to be
00448      * shared by two child simplices.  Here we record the change, if any,
00449      * in the number of simplices associated with each atom. */
00450     dnqsm = Vmem_malloc(thee->vmem, nAffAtoms, sizeof(int));
00451     VASSERT(dnqsm != VNULL);
00452     nqsmNew = Vmem_malloc(thee->vmem, nAffAtoms, sizeof(int));
00453     VASSERT(nqsmNew != VNULL);
00454     qsmNew = Vmem_malloc(thee->vmem, nAffAtoms, sizeof(int*));
00455     VASSERT(qsmNew != VNULL);
00456     for (iatom=0; iatom<nAffAtoms; iatom++) {
```

```
00457         dnqsm[iatom] = -1;
00458         atomID = affAtoms[iatom];
00459         for (isimp=0; isimp<num; isimp++) {
00460             for (jatom=0; jatom<nsqmNew[isimp]; jatom++) {
00461                 if (sqmNew[isimp][jatom] == atomID) dnqsm[iatom]++;
00462             }
00463         }
00464         VASSERT(dnqsm[iatom] > -1);
00465     }
00466     /* Setup the new entries in the array */
00467     for (iatom=0;iatom<nAffAtoms; iatom++) {
00468         atomID = affAtoms[iatom];
00469         qsmNew[iatom] = Vmem_malloc(thee->vmem,
00470                 (dnqsm[iatom] + thee->nqsm[atomID]),
00471                 sizeof(int));
00472         nqsmNew[iatom] = 0;
00473         VASSERT(qsmNew[iatom] != VNULL);
00474     }
00475     /* Fill the new entries in the array */
00476     /* First, do the modified entries */
00477     for (isimp=0; isimp<num; isimp++) {
00478         simpID = SS_id(simps[isimp]);
00479         for (iatom=0; iatom<nsqmNew[isimp]; iatom++) {
00480             atomID = sqmNew[isimp][iatom];
00481             for (jatom=0; jatom<nAffAtoms; jatom++) {
00482                 if (atomID == affAtoms[jatom]) break;
00483             }
00484             if (jatom < nAffAtoms) {
00485                 qsmNew[jatom][nqsmNew[jatom]] = simpID;
00486                 nqsmNew[jatom]++;
00487             }
00488         }
00489     }
00490     /* Now do the unmodified entries */
00491     for (iatom=0; iatom<nAffAtoms; iatom++) {
00492         atomID = affAtoms[iatom];
00493         for (isimp=0; isimp<thee->nqsm[atomID]; isimp++) {
00494             for (jsimp=0; jsimp<num; jsimp++) {
00495                 simpID = SS_id(simps[jsimp]);
00496                 if (thee->qsm[atomID][isimp] == simpID) break;
00497             }
00498             if (jsimp == num) {
00499                 qsmNew[iatom][nqsmNew[iatom]] = thee->qsm[atomID][isimp];
00500                 nqsmNew[iatom]++;
00501             }
00502         }
00503     }
00504
00505     /* Replace the existing entries in the table.  Do the QSM entires
00506      * first, since they require affAtoms = thee->sqm[simps[0]] */
00507     for (iatom=0; iatom<nAffAtoms; iatom++) {
00508         atomID = affAtoms[iatom];
00509         Vmem_free(thee->vmem, thee->nqsm[atomID], sizeof(int),
00510           (void **)&(thee->qsm[atomID]));
00511         thee->qsm[atomID] = qsmNew[iatom];
00512         thee->nqsm[atomID] = nqsmNew[iatom];
00513     }
```

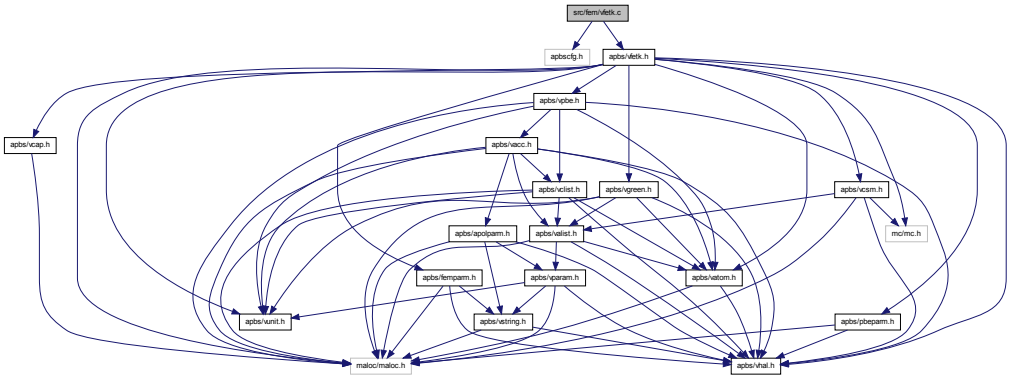
```
00514     for (isimp=0; isimp<num; isimp++) {
00515         simpID = SS_id(simps[isimp]);
00516         if (thee->nsqm[simpID] > 0) Vmem_free(thee->vmem, thee->nsqm[simpID],
00517           sizeof(int), (void **)&(thee->sqm[simpID]));
00518         thee->sqm[simpID] = sqmNew[isimp];
00519         thee->nsqm[simpID] = nsqmNew[isimp];
00520     }
00521
00522     Vmem_free(thee->vmem, num, sizeof(int *), (void **)&sqmNew);
00523     Vmem_free(thee->vmem, num, sizeof(int), (void **)&nsqmNew);
00524     Vmem_free(thee->vmem, nAffAtoms, sizeof(int *), (void **)&qsmNew);
00525     Vmem_free(thee->vmem, nAffAtoms, sizeof(int), (void **)&nqsmNew);
00526     Vmem_free(thee->vmem, nAffAtoms, sizeof(int), (void **)&dnqsm);
00527
00528
00529     return 1;
00530
00531
00532 }
00533
00534 #endif
```

10.17 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[][VMAXP], int icx[][VMAXP], int icy[][VMAXP], int icz[][VMAXP])
- VPRIVATE void **polyEval** (int numP, double p[], double c[][VMAXP], double xv[])
- 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, FEMparm *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_BT/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 frmo 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)

  Intializes 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:

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

  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:

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

  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:

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

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 dimII, int edgeType, int chart[], double vx[][3])
- VPUBLIC void **Vfetk_PDE_mapBoundary** (int dim, int dimII, int vertexType, int chart, double vx[3])
- VPUBLIC int **Vfetk_PDE_markSimplex** (int dim, int dimII, 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:

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

  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 **lgr_2DP1** [3][VMAXP]
- VPRIVATE int **lgr_2DP1x** [3][VMAXP]
- VPRIVATE int **lgr_2DP1y** [3][VMAXP]
- VPRIVATE int **lgr_2DP1z** [3][VMAXP]
- VPRIVATE int **dim_3DP1** = VAPBS_NVS
- VPRIVATE int **lgr_3DP1** [VAPBS_NVS][VMAXP]
- VPRIVATE int **lgr_3DP1x** [VAPBS_NVS][VMAXP]
- VPRIVATE int **lgr_3DP1y** [VAPBS_NVS][VMAXP]
- VPRIVATE int **lgr_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 1567 2010-03-12 22:10:39Z sdg0919

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
* 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 Washington University in St. Louis 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 90 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,  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 }
}
```

Definition at line 383 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 },
{  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 }
}
```

Definition at line 392 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 },
{  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
}
```

Definition at line 399 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 }
}
```

Definition at line 406 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,  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,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
}
```

Definition at line 435 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 },
{  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 }
}
```

Definition at line 443 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 },
{  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 }
}
```

Definition at line 451 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 },
{  2,  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.

10.17.2.10 VPRIVATE char* neumCubeString

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 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 127 of file vfetk.c.

10.18 src/fem/vfetk.c

```
00001
00050 #include "apbscfg.h"
00051
00052 #ifdef HAVE_MC_H
00053
00054 #include "apbs/vfetk.h"
00055
00056 /* Define the macro DONEUMANN to run with all-Neumann boundary conditions.
00057  * Set this macro at your own risk! */
00058 /* #define DONEUMANN 1 */
00059
00060 /*
00061  * @brief  Calculuate the contribution to the charge-potential energy from one
00062  * atom
00063  * @ingroup Vfetk
00064  * @author  Nathan Baker
00065  * @param  thee  current Vfetk object
00066  * @param  iatom  current atom index
00067  * @param  color  simplex subset (partition) under consideration
00068  * @param  sol  current solution
00069  * @returns Per-atom energy
00070  */
00071 VPRIVATE double Vfetk_qfEnergyAtom(
00072         Vfetk *thee,
00073         int iatom,
00074         int color,
00075         double *sol
00076         );
00077
00078 /*
00079  * @brief  Container for local variables
00080  * @ingroup  Vfetk
00081  * @bug  Not thread-safe
00082  */
00083 VPRIVATE Vfetk_LocalVar var;
00084
00085 /*
00086  * @brief  MCSF-format cube mesh (all Dirichlet)
00087  * @ingroup Vfetk
00088  * @author  Based on mesh by Mike Holst
00089  */
00090 VPRIVATE char *diriCubeString =
00091 "mcsf_begin=1;\n\
00092 \n\
00093 dim=3;\n\
00094 dimii=3;\n\
00095 vertices=8;\n\
00096 simplices=6;\n\
00097 \n\
00098 vert=[\n\
00099 0 0 -0.5 -0.5 -0.5\n\
00100 1 0  0.5 -0.5 -0.5\n\
00101 2 0 -0.5  0.5 -0.5\n\
00102 3 0  0.5  0.5 -0.5\n\
00103 4 0 -0.5 -0.5  0.5\n\
```

```
00104 5 0  0.5 -0.5  0.5\n\
00105 6 0 -0.5  0.5  0.5\n\
00106 7 0  0.5  0.5  0.5\n\
00107 ];\n\
00108 \n\
00109 simp=[\n\
00110 0 0 0 0 1 0 1 0 5 1 2\n\
00111 1 0 0 0 1 1 0 0 5 2 4\n\
00112 2 0 0 0 1 0 1 1 5 3 2\n\
00113 3 0 0 0 1 0 1 3 5 7 2\n\
00114 4 0 0 1 1 0 0 2 5 7 6\n\
00115 5 0 0 1 1 0 0 2 5 6 4\n\
00116 ];\n\
00117 \n\
00118 mcsf_end=1;\n\
00119 \n\
00120 ";
00121
00122 /*
00123  * @brief  MCSF-format cube mesh (all Neumann)
00124  * @ingroup Vfetk
00125  * @author  Based on mesh by Mike Holst
00126  */
00127 VPRIVATE char *neumCubeString =
00128 "mcsf_begin=1;\n\
00129 \n\
00130 dim=3;\n\
00131 dimii=3;\n\
00132 vertices=8;\n\
00133 simplices=6;\n\
00134 \n\
00135 vert=[\n\
00136 0 0 -0.5 -0.5 -0.5\n\
00137 1 0  0.5 -0.5 -0.5\n\
00138 2 0 -0.5  0.5 -0.5\n\
00139 3 0  0.5  0.5 -0.5\n\
00140 4 0 -0.5 -0.5  0.5\n\
00141 5 0  0.5 -0.5  0.5\n\
00142 6 0 -0.5  0.5  0.5\n\
00143 7 0  0.5  0.5  0.5\n\
00144 ];\n\
00145 \n\
00146 simp=[\n\
00147 0 0 0 0 2 0 2 0 5 1 2\n\
00148 1 0 0 0 2 2 0 0 5 2 4\n\
00149 2 0 0 0 2 0 2 1 5 3 2\n\
00150 3 0 0 0 2 0 2 3 5 7 2\n\
00151 4 0 0 2 2 0 0 2 5 7 6\n\
00152 5 0 0 2 2 0 0 2 5 6 4\n\
00153 ];\n\
00154 \n\
00155 mcsf_end=1;\n\
00156 \n\
00157 ";
00158
00159 /*
00160  * @brief  Return the smoothed value of the dielectric coefficient at the
```

```
00161  * current point using a fast, chart-based method
00162  * @ingroup  Vfetk
00163  * @author  Nathan Baker
00164  * @returns  Value of dielectric coefficient
00165  * @bug  Not thread-safe
00166  */
00167 VPRIVATE double diel();
00168
00169 /*
00170  * @brief  Return the smoothed value of the ion accessibility at the
00171  * current point using a fast, chart-based method
00172  * @ingroup  Vfetk
00173  * @author  Nathan Baker
00174  * @returns  Value of mobile ion coefficient
00175  * @bug  Not thread-safe
00176  */
00177 VPRIVATE double ionacc();
00178
00179 /*
00180  * @brief  Smooths a mesh-based coefficient with a simple harmonic function
00181  * @ingroup  Vfetk
00182  * @author  Nathan Baker
00183  * @param  meth  Method for smoothing
00184  *   \li  0 ==> arithmetic mean (gives bad results)
00185  *   \li  1 ==> geometric mean
00186  * @param  nverts  Number of vertices
00187  * @param  dist  distance from point to each vertex
00188  * @param  coeff  coefficient value at each vertex
00189  * @note  Thread-safe
00190  * @return smoothed value of coefficieent at point of interest */
00191 VPRIVATE double smooth(
00192         int nverts,
00193         double dist[VAPBS_NVS],
00194         double coeff[VAPBS_NVS],
00195         int meth
00196         );
00197
00198
00199 /*
00200  * @brief  Return the analytical multi-sphere Debye-Huckel approximation (in
00201  * kT/e) at the specified point
00202  * @ingroup  Vfetk
00203  * @author  Nathan Baker
00204  * @param  pbe  Vpbe object
00205  * @param  d  Dimension of x
00206  * @param  x  Coordinates of point of interest (in &Aring;)
00207  * @note  Thread-safe
00208  * @returns  Multi-sphere Debye-Huckel potential in kT/e
00209  */
00210 VPRIVATE double debye_U(
00211         Vpbe *pbe,
00212         int d,
00213         double x[]
00214         );
00215
00216 /*
00217  * @brief  Return the difference between the analytical multi-sphere
```

```
00218  * Debye-Huckel approximation and Coulomb's law (in kT/e) at the specified
00219  * point
00220  * @ingroup  Vfetk
00221  * @author  Nathan Baker
00222  * @param  pbe  Vpbe object
00223  * @param  d  Dimension of x
00224  * @param  x  Coordinates of point of interest (in &Aring;)
00225  * @note  Thread-safe
00226  * @returns  Multi-sphere Debye-Huckel potential in kT/e */
00227 VPRIVATE double debye_Udiff(
00228         Vpbe *pbe,
00229         int d,
00230         double x[]
00231         );
00232
00233 /*
00234  * @brief  Calculate the Coulomb's
00235  * Debye-Huckel approximation and Coulomb's law (in kT/e) at the specified
00236  * point
00237  * @ingroup  Vfetk
00238  * @author  Nathan Baker
00239  * @param  pbe  Vpbe object
00240  * @param  d  Dimension of x
00241  * @param  x  Coordinates of point of interest (in &Aring;)
00242  * @param  eps  Dielectric constant
00243  * @param  U  Set to potential (in kT/e)
00244  * @param  dU  Set to potential gradient (in kT/e/&Aring;)
00245  * @param  d2U  Set to Laplacian of potential (in \f$kT e^{-1} \AA^{-2}\f$)
00246  * @returns  Multi-sphere Debye-Huckel potential in kT/e */
00247 VPRIVATE void coulomb(
00248         Vpbe *pbe,
00249         int d,
00250         double x[],
00251         double eps,
00252         double *U,
00253         double dU[],
00254         double *d2U
00255         );
00256
00257 /*
00258  * @brief  2D linear master simplex information generator
00259  * @ingroup  Vfetk
00260  * @author  Mike Holst
00261  * @param dimIS  dunno
00262  * @param ndof  dunno
00263  * @param dof  dunno
00264  * @param c  dunno
00265  * @param cx  dunno
00266  * @note  Trust in Mike */
00267 VPRIVATE void init_2DP1(
00268         int dimIS[],
00269         int *ndof,
00270         int dof[],
00271         double c[][VMAXP],
00272         double cx[][VMAXP],
00273         double cy[][VMAXP],
00274         double cz[][VMAXP]
```

```
00275         );
00276
00277 /*
00278  * @brief  3D linear master simplex information generator
00279  * @ingroup  Vfetk
00280  * @author  Mike Holst
00281  * @param  dimIS  dunno
00282  * @param ndof dunno
00283  * @param dof dunno
00284  * @param c dunno
00285  * @param cx dunno
00286  * @param cy dunno
00287  * @param cz dunno
00288  * @note  Trust in Mike */
00289 VPRIVATE void init_3DP1(
00290         int dimIS[],
00291         int *ndof,
00292         int dof[],
00293         double c[][VMAXP],
00294         double cx[][VMAXP],
00295         double cy[][VMAXP],
00296         double cz[][VMAXP]
00297         );
00298
00299 /*
00300  * @brief  Setup coefficients of polynomials from integer table data
00301  * @ingroup  Vfetk
00302  * @author  Mike Holst
00303  * @param numP  dunno
00304  * @param c  dunno
00305  * @param cx  dunno
00306  * @param cy  dunno
00307  * @param cz  dunno
00308  * @param ic  dunno
00309  * @param icx  dunno
00310  * @param icy  dunno
00311  * @param icz  dunno
00312  * @note  Trust in Mike */
00313 VPRIVATE void setCoef(
00314         int numP,
00315         double c[][VMAXP],
00316         double cx[][VMAXP],
00317         double cy[][VMAXP],
00318         double cz[][VMAXP],
00319         int ic[][VMAXP],
00320         int icx[][VMAXP],
00321         int icy[][VMAXP],
00322         int icz[][VMAXP]
00323         );
00324
00325 /*
00326  * @brief  Evaluate a collection of at most cubic polynomials at a
00327  * specified point in at most R^3.
00328  * @ingroup  Vfetk
00329  * @author  Mike Holst
00330  * @param numP  the number of polynomials to evaluate
00331  * @param p  the results of the evaluation
```

```
00332  * @param c   the coefficients of each polynomial
00333  * @param xv  the point (x,y,z) to evaluate the polynomials.
00334  * @note  Mike says:
00335  * <pre>
00336  *  Note that "VMAXP" must be >= 19 for cubic polynomials.
00337  *  The polynomials are build from the coefficients c[][] as
00338  *  follows.  To build polynomial "k", fix k and set:
00339  *
00340  *  c0=c[k][0], c1=c[k][1], .... , cp=c[k][p]
00341  *
00342  *  Then evaluate as:
00343  *
00344  *  p3(x,y,z) = c0 + c1*x + c2*y + c3*z
00345  *            + c4*x*x + c5*y*y + c6*z*z + c7*x*y + c8*x*z + c9*y*z
00346  *            + c10*x*x*x + c11*y*y*y + c12*z*z*z
00347  *            + c13*x*x*y + c14*x*x*z + c15*x*y*y
00348  *            + c16*y*y*z + c17*x*z*z + c18*y*z*z
00349  * </pre>
00350  */
00351 VPRIVATE void polyEval(
00352         int numP,
00353         double p[],
00354         double c[][VMAXP],
00355         double xv[]
00356         );
00357
00358 /*
00359  * @brief  I have no clue what this variable does, but we need it to initialize
00360  * the simplices
00361  * @ingroup  Vfetk
00362  * @author  Mike Holst */
00363 VPRIVATE int dim_2DP1 = 3;
00364
00365 /*
00366  * @brief  I have no clue what these variable do, but we need it to initialize
00367  * the simplices
00368  * @ingroup  Vfetk
00369  * @author  Mike Holst
00370  * @note  Mike says:
00371  * <pre>
00372  *  2D-P1 Basis:
00373  *
00374  *  p1(x,y) = c0 + c1*x + c2*y
00375  *
00376  *  Lagrange Point    Lagrange Basis Function Definition
00377  *  --------------    ----------------------------------
00378  *  (0, 0)            p[0](x,y) = 1 - x - y
00379  *  (1, 0)            p[1](x,y) = x
00380  *  (0, 1)            p[2](x,y) = y
00381  *  </pre>
00382  */
00383 VPRIVATE int lgr_2DP1[3][VMAXP] = {
00384 /*c0  c1  c2  c3
00385 * ------------------------------------------------------------ */
00386 /* 1   x   y   z
00387 * ------------------------------------------------------------ */
00388 {  2, -2, -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
```

```
00389 {  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00390 {  0,  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
00391 };
00392 VPRIVATE int lgr_2DP1x[3][VMAXP] = {
00393 /*c0 ------------------------------------------------------------------ */
00394 /* 1 ------------------------------------------------------------------ */
00395 { -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00396 {  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00397 {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
00398 };
00399 VPRIVATE int lgr_2DP1y[3][VMAXP] = {
00400 /*c0 ------------------------------------------------------------------ */
00401 /* 1 ------------------------------------------------------------------ */
00402 { -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00403 {  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 }
00405 };
00406 VPRIVATE int lgr_2DP1z[3][VMAXP] = {
00407 /*c0 ------------------------------------------------------------------ */
00408 /* 1 ------------------------------------------------------------------ */
00409 {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00410 {  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 }
00412 };
00413
00414
00415 /*
00416  * @brief  I have no clue what these variable do, but we need it to initialize
00417  * the simplices
00418  * @ingroup  Vfetk
00419  * @author  Mike Holst
00420  * @note  Mike says:
00421  * <pre>
00422  * 3D-P1 Basis:
00423  *
00424  * p1(x,y,z) = c0 + c1*x + c2*y + c3*z
00425  *
00426  * Lagrange Point    Lagrange Basis Function Definition
00427  * --------------    ----------------------------------
00428  * (0, 0, 0)         p[0](x,y,z) = 1 - x - y - z
00429  * (1, 0, 0)         p[1](x,y,z) = x
00430  * (0, 1, 0)         p[2](x,y,z) = y
00431  * (0, 0, 1)         p[3](x,y,z) = z
00432  * </pre>
00433  */
00434 VPRIVATE int dim_3DP1 = VAPBS_NVS;
00435 VPRIVATE int lgr_3DP1[VAPBS_NVS][VMAXP] = {
00436 /*c0  c1  c2  c3 ------------------------------------------------------- */
00437 /* 1   x   y   z ------------------------------------------------------- */
00438 {  2, -2, -2, -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00439 {  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00440 {  0,  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00441 {  0,  0,  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
00442 };
00443 VPRIVATE int lgr_3DP1x[VAPBS_NVS][VMAXP] = {
00444 /*c0 ------------------------------------------------------------------ */
00445 /* 1 ------------------------------------------------------------------ */
```

```
00446 { -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00447 {  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00448 {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00449 {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
00450 };
00451 VPRIVATE int lgr_3DP1y[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 },
00455 {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00456 {  2,  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 }
00458 };
00459 VPRIVATE int lgr_3DP1z[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 },
00463 {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00464 {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00465 {  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00466 };
00467
00468 /*
00469  * @brief  Another Holst variable
00470  * @ingroup  Vfetk
00471  * @author  Mike Holst
00472  * @note  Mike says: 1 = linear, 2 = quadratic */
00473 VPRIVATE const int P_DEG=1;
00474
00475 /*
00476  * @brief  Another Holst variable
00477  * @ingroup  Vfetk
00478  * @author  Mike Holst */
00479 VPRIVATE int numP;
00480 VPRIVATE double c[VMAXP][VMAXP];
00481 VPRIVATE double cx[VMAXP][VMAXP];
00482 VPRIVATE double cy[VMAXP][VMAXP];
00483 VPRIVATE double cz[VMAXP][VMAXP];
00484
00485 #if !defined(VINLINE_VFETK)
00486
00487 VPUBLIC Gem* Vfetk_getGem(Vfetk *thee) {
00488
00489    VASSERT(thee != VNULL);
00490    return thee->gm;
00491
00492 }
00493
00494 VPUBLIC AM* Vfetk_getAM(Vfetk *thee) {
00495
00496    VASSERT(thee != VNULL);
00497    return thee->am;
00498 }
00499
00500 VPUBLIC Vpbe* Vfetk_getVpbe(Vfetk *thee) {
00501
00502    VASSERT(thee != VNULL);
```

```
00503     return thee->pbe;
00504
00505 }
00506
00507 VPUBLIC Vcsm* Vfetk_getVcsm(Vfetk *thee) {
00508
00509     VASSERT(thee != VNULL);
00510     return thee->csm;
00511
00512 }
00513
00514 VPUBLIC int Vfetk_getAtomColor(Vfetk *thee, int iatom) {
00515
00516     int natoms;
00517
00518     VASSERT(thee != VNULL);
00519
00520     natoms = Valist_getNumberAtoms(Vpbe_getValist(thee->pbe));
00521     VASSERT(iatom < natoms);
00522
00523     return Vatom_getPartID(Valist_getAtom(Vpbe_getValist(thee->pbe), iatom));
00524 }
00525 #endif /* if !defined(VINLINE_VFETK) */
00526
00527 VPUBLIC Vfetk* Vfetk_ctor(Vpbe *pbe, Vhal_PBEType type) {
00528
00529     /* Set up the structure */
00530     Vfetk *thee = VNULL;
00531     thee = Vmem_malloc(VNULL, 1, sizeof(Vfetk) );
00532     VASSERT(thee != VNULL);
00533     VASSERT(Vfetk_ctor2(thee, pbe, type));
00534
00535     return thee;
00536 }
00537
00538 VPUBLIC int Vfetk_ctor2(Vfetk *thee, Vpbe *pbe, Vhal_PBEType type) {
00539
00540     int i;
00541     double center[VAPBS_DIM];
00542
00543     /* Make sure things have been properly initialized & store them */
00544     VASSERT(pbe != VNULL);
00545     thee->pbe = pbe;
00546     VASSERT(pbe->alist != VNULL);
00547     VASSERT(pbe->acc != VNULL);
00548
00549     /* Store PBE type */
00550     thee->type = type;
00551
00552     /* Set up memory management object */
00553     thee->vmem = Vmem_ctor("APBS::VFETK");
00554
00555     /* Set up FEtk objects */
00556  Vnm_print(0, "Vfetk_ctor2:  Constructing PDE...\n");
00557     thee->pde = Vfetk_PDE_ctor(thee);
00558     Vnm_print(0, "Vfetk_ctor2:  Constructing Gem...\n");
00559     thee->gm = Gem_ctor(thee->vmem, thee->pde);
```

```
00560     Vnm_print(0, "Vfetk_ctor2:  Constructing Aprx...\n");
00561     thee->aprx = Aprx_ctor(thee->vmem, thee->gm, thee->pde);
00562     Vnm_print(0, "Vfetk_ctor2:  Constructing Aprx...\n");
00563     thee->am = AM_ctor(thee->vmem, thee->aprx);
00564
00565     /* Reset refinement level */
00566     thee->level = 0;
00567
00568     /* Set default solver variables */
00569     thee->lkey = VLT_MG;
00570     thee->lmax = 1000000;
00571     thee->ltol = 1e-5;
00572     thee->lprec = VPT_MG;
00573     thee->nkey = VNT_NEW;
00574     thee->nmax = 1000000;
00575     thee->ntol = 1e-5;
00576     thee->gues = VGT_ZERO;
00577     thee->pjac = -1;
00578
00579     /* Store local copy of myself */
00580     var.fetk = thee;
00581     var.initGreen = 0;
00582
00583  /* Set up the external Gem subdivision hook */
00584     Gem_setExternalUpdateFunction(thee->gm, Vfetk_externalUpdateFunction);
00585
00586     /* Set up ion-related variables */
00587     var.zkappa2 = Vpbe_getZkappa2(var.fetk->pbe);
00588     var.ionstr = Vpbe_getBulkIonicStrength(var.fetk->pbe);
00589     if (var.ionstr > 0.0) var.zks2 = 0.5*var.zkappa2/var.ionstr;
00590     else var.zks2 = 0.0;
00591     Vpbe_getIons(var.fetk->pbe, &(var.nion), var.ionConc, var.ionRadii,
00592       var.ionQ);
00593     for (i=0; i<var.nion; i++) {
00594         var.ionConc[i] = var.zks2 * var.ionConc[i] * var.ionQ[i];
00595     }
00596
00597     /* Set uninitialized objects to NULL */
00598     thee->pbeparm = VNULL;
00599     thee->feparm = VNULL;
00600     thee->csm = VNULL;
00601
00602     return 1;
00603 }
00604
00605 VPUBLIC void Vfetk_setParameters(Vfetk *thee, PBEparm *pbeparm,
00606   FEMparm *feparm) {
00607
00608     VASSERT(thee != VNULL);
00609     thee->feparm = feparm;
00610     thee->pbeparm = pbeparm;
00611 }
00612
00613 VPUBLIC void Vfetk_dtor(Vfetk **thee) {
00614     if ((*thee) != VNULL) {
00615         Vfetk_dtor2(*thee);
00616         //Vmem_free(VNULL, 1, sizeof(Vfetk), (void **)thee);
```

```
00617         (*thee) = VNULL;
00618     }
00619 }
00620
00621 VPUBLIC void Vfetk_dtor2(Vfetk *thee) {
00622     Vcsm_dtor(&(thee->csm));
00623     AM_dtor(&(thee->am));
00624     Aprx_dtor(&(thee->aprx));
00625     Vfetk_PDE_dtor(&(thee->pde));
00626     Vmem_dtor(&(thee->vmem));
00627 }
00628
00629 VPUBLIC double* Vfetk_getSolution(Vfetk *thee, int *length) {
00630
00631    int i;
00632    double *solution;
00633    double *theAnswer;
00634    AM *am;
00635
00636    VASSERT(thee != VNULL);
00637
00638    /* Get the AM object */
00639    am = thee->am;
00640    /* Copy the solution into the w0 vector */
00641    Bvec_copy(am->w0, am->u);
00642    /* Add the Dirichlet conditions */
00643    Bvec_axpy(am->w0, am->ud, 1.);
00644    /* Get the data from the Bvec */
00645    solution = Bvec_addr(am->w0);
00646    /* Get the length of the data from the Bvec */
00647    *length = Bvec_numRT(am->w0);
00648    /* Make sure that we got scalar data (only one block) for the solution
00649     * to the FETK */
00650    VASSERT(1 == Bvec_numB(am->w0));
00651    /* Allocate space for the returned vector and copy the solution into it */
00652    theAnswer = VNULL;
00653    theAnswer = Vmem_malloc(VNULL, *length, sizeof(double));
00654    VASSERT(theAnswer != VNULL);
00655    for (i=0; i<(*length); i++) theAnswer[i] = solution[i];
00656
00657    return theAnswer;
00658 }
00659
00660 VPUBLIC double Vfetk_energy(Vfetk *thee, int color, int nonlin) {
00661
00662     double totEnergy = 0.0;
00663     double qfEnergy = 0.0;
00664     double dqmEnergy = 0.0;
00665
00666     VASSERT(thee != VNULL);
00667
00668     if (nonlin && (Vpbe_getBulkIonicStrength(thee->pbe) > 0.)) {
00669         Vnm_print(0, "Vfetk_energy:  calculating full PBE energy\n");
00670         Vnm_print(0, "Vfetk_energy:  bulk ionic strength = %g M\n",
00671           Vpbe_getBulkIonicStrength(thee->pbe));
00672         dqmEnergy = Vfetk_dqmEnergy(thee, color);
00673         Vnm_print(0, "Vfetk_energy:  dqmEnergy = %g kT\n", dqmEnergy);
```

```
00674         qfEnergy = Vfetk_qfEnergy(thee, color);
00675         Vnm_print(0, "Vfetk_energy:  qfEnergy = %g kT\n", qfEnergy);
00676 
00677         totEnergy = qfEnergy - dqmEnergy;
00678     } else {
00679         Vnm_print(0, "Vfetk_energy:  calculating only q-phi energy\n");
00680         dqmEnergy = Vfetk_dqmEnergy(thee, color);
00681         Vnm_print(0, "Vfetk_energy:  dqmEnergy = %g kT (NOT USED)\n", dqmEnergy);

00682         qfEnergy = Vfetk_qfEnergy(thee, color);
00683         Vnm_print(0, "Vfetk_energy:  qfEnergy = %g kT\n", qfEnergy);
00684         totEnergy = 0.5*qfEnergy;
00685     }
00686 
00687     return totEnergy;
00688 
00689 }
00690 
00691 
00692 VPUBLIC double Vfetk_qfEnergy(Vfetk *thee, int color) {
00693 
00694     double *sol; int nsol;
00695     int iatom, natoms;
00696     double energy = 0.0;
00697 
00698     AM *am;
00699 
00700     VASSERT(thee != VNULL);
00701     am = thee->am;
00702 
00703     /* Get the finest level solution */
00704     sol= VNULL;
00705     sol = Vfetk_getSolution(thee, &nsol);
00706     VASSERT(sol != VNULL);
00707 
00708     /* Make sure the number of entries in the solution array matches the
00709      * number of vertices currently in the mesh */
00710     if (nsol != Gem_numVV(thee->gm)) {
00711        Vnm_print(2, "Vfetk_qfEnergy: Number of unknowns in solution does not matc
      h\n");
00712        Vnm_print(2, "Vfetk_qfEnergy: number of vertices in mesh!!!  Bailing out!\
      n");
00713        VASSERT(0);
00714     }
00715 
00716     /* Now we do the sum over atoms... */
00717     natoms = Valist_getNumberAtoms(thee->pbe->alist);
00718     for (iatom=0; iatom<natoms; iatom++) {
00719 
00720         energy = energy + Vfetk_qfEnergyAtom(thee, iatom, color, sol);
00721 
00722     } /* end for iatom */
00723 
00724     /* Destroy the finest level solution */
00725     Vmem_free(VNULL, nsol, sizeof(double), (void **)&sol);
00726 
00727     /* Return the energy */
```

```
00728     return energy;
00729 }
00730
00731 VPRIVATE double Vfetk_qfEnergyAtom(
00732         Vfetk *thee,
00733         int iatom,
00734         int color,
00735         double *sol) {
00736
00737     Vatom *atom;
00738     double charge;
00739     double phi[VAPBS_NVS], phix[VAPBS_NVS][3], *position;
00740     double uval;
00741     double energy = 0.0;
00742     int isimp, nsimps;
00743     SS *simp;
00744     int icolor, ivert, usingColor;
00745
00746
00747     /* Get atom information */
00748     atom = Valist_getAtom(thee->pbe->alist, iatom);
00749     icolor = Vfetk_getAtomColor(thee, iatom);
00750     charge = Vatom_getCharge(atom);
00751     position = Vatom_getPosition(atom);
00752
00753     /* Find out if we're using colors */
00754     usingColor = (color >= 0);
00755
00756     if (usingColor && (icolor<0)) {
00757         Vnm_print(2, "Vfetk_qfEnergy: Atom colors not set!\n");
00758         VASSERT(0);
00759     }
00760
00761     /* Check if this atom belongs to the specified partition */
00762     if ((icolor==color) || (!usingColor)) {
00763         /* Loop over the simps associated with this atom */
00764         nsimps =  Vcsm_getNumberSimplices(thee->csm, iatom);
00765
00766         /* Get the first simp of the correct color; we can use just one
00767          * simplex for energy evaluations, but not for force
00768          * evaluations */
00769         for (isimp=0; isimp<nsimps; isimp++) {
00770
00771             simp = Vcsm_getSimplex(thee->csm, isimp, iatom);
00772
00773             /* If we've asked for a particular partition AND if the atom
00774              * is our partition, then compute the energy */
00775             if ((SS_chart(simp)==color)||(color<0)) {
00776                 /* Get the value of each basis function evaluated at this
00777                  * point */
00778                 Gem_pointInSimplexVal(thee->gm, simp, position, phi, phix);
00779                 for (ivert=0; ivert<SS_dimVV(simp); ivert++) {
00780                     uval = sol[VV_id(SS_vertex(simp,ivert))];
00781                     energy += (charge*phi[ivert]*uval);
00782                 } /* end for ivert */
00783                 /* We only use one simplex of the appropriate color for
00784                  * energy calculations, so break here */
```

```
00785                 break;
00786             } /* endif (color) */
00787         } /* end for isimp */
00788     }
00789
00790    return energy;
00791 }
00792
00793
00794 VPUBLIC double Vfetk_dqmEnergy(Vfetk *thee, int color) {
00795
00796     return AM_evalJ(thee->am);
00797
00798 }
00799
00800 VPUBLIC void Vfetk_setAtomColors(Vfetk *thee) {
00801
00802 #define VMAXLOCALCOLORSDONTREUSETHISVARIABLE 1024
00803     SS *simp;
00804     Vatom *atom;
00805     int i, natoms;
00806
00807     VASSERT(thee != VNULL);
00808
00809     natoms = Valist_getNumberAtoms(thee->pbe->alist);
00810     for (i=0; i<natoms; i++) {
00811         atom = Valist_getAtom(thee->pbe->alist, i);
00812         simp = Vcsm_getSimplex(thee->csm, 0, i);
00813         Vatom_setPartID(atom, SS_chart(simp));
00814     }
00815
00816 }
00817
00818 VPUBLIC unsigned long int Vfetk_memChk(Vfetk *thee) {
00819
00820     int memUse = 0;
00821
00822     if (thee == VNULL) return 0;
00823
00824     memUse = memUse + sizeof(Vfetk);
00825     memUse = memUse + Vcsm_memChk(thee->csm);
00826
00827     return memUse;
00828 }
00829
00830 VPUBLIC Vrc_Codes Vfetk_genCube(
00831         Vfetk *thee,
00832         double center[3],
00833         double length[3],
00834         Vfetk_MeshLoad meshType) {
00835     AM *am = VNULL;
00836     Gem *gm = VNULL;
00837
00838     int skey = 0;  /* Simplex format */
00839     char *key = "r";  /* Read */
00840     char *iodev = "BUFF";  /* Buffer */
00841     char *iofmt = "ASC";  /* ASCII */
```

```
00842     char *iohost = "localhost";  /* localhost (dummy) */
00843     char *iofile = "0";  /*< socket 0 (dummy) */
00844     Vio *sock = VNULL;
00845     char buf[VMAX_BUFSIZE];
00846     int bufsize = 0;
00847     VV *vx = VNULL;
00848     int i, j;
00849     double x;
00850
00851     VASSERT(thee != VNULL);
00852     am = thee->am;
00853     VASSERT(am != VNULL);
00854     gm = thee->gm;
00855     VASSERT(gm != VNULL);
00856
00857     /* @note This code is based on Gem_makeCube by Mike Holst */
00858     /* Write mesh string to buffer and read back */
00859  switch (meshType) {
00860   case VML_DIRICUBE:
00861    bufsize = strlen(diriCubeString);
00862    VASSERT( bufsize <= VMAX_BUFSIZE );
00863    strncpy(buf, diriCubeString, VMAX_BUFSIZE);
00864    break;
00865   case VML_NEUMCUBE:
00866    bufsize = strlen(neumCubeString);
00867    Vnm_print(2, "Vfetk_genCube:  WARNING!  USING EXPERIMENTAL NEUMANN BOUNDARY CO
      NDITIONS!\n");
00868    VASSERT( bufsize <= VMAX_BUFSIZE );
00869    strncpy(buf, neumCubeString, VMAX_BUFSIZE);
00870    break;
00871   case VML_EXTERNAL:
00872    Vnm_print(2, "Vfetk_genCube:  Got request for external mesh!\n");
00873    Vnm_print(2, "Vfetk_genCube:  How did we get here?\n");
00874    return VRC_FAILURE;
00875    break;
00876   default:
00877    Vnm_print(2, "Vfetk_genCube:  Unknown mesh type (%d)\n", meshType);
00878    return VRC_FAILURE;
00879  }
00880     VASSERT( VNULL != (sock=Vio_socketOpen(key,iodev,iofmt,iohost,iofile)) );
00881     Vio_bufTake(sock, buf, bufsize);
00882     AM_read(am, skey, sock);
00883  Vio_connectFree(sock);
00884     Vio_bufGive(sock);
00885     Vio_dtor(&sock);
00886
00887     /* Scale (unit) cube */
00888     for (i=0; i<Gem_numVV(gm); i++) {
00889         vx = Gem_VV(gm, i);
00890         for (j=0; j<3; j++) {
00891             x = VV_coord(vx, j);
00892             x *= length[j];
00893             VV_setCoord(vx, j, x);
00894         }
00895     }
00896
00897     /* Add new center */
```

```
00898     for (i=0; i<Gem_numVV(gm); i++) {
00899         vx = Gem_VV(gm, i);
00900         for (j=0; j<3; j++) {
00901             x = VV_coord(vx, j);
00902             x += center[j];
00903             VV_setCoord(vx, j, x);
00904         }
00905     }
00906 
00907 
00908     return VRC_SUCCESS;
00909 }
00910 
00911 VPUBLIC Vrc_Codes Vfetk_loadMesh(
00912          Vfetk *thee,
00913          double center[3],
00914          double length[3],
00915          Vfetk_MeshLoad meshType,
00916          Vio *sock) {
00917 
00918  Vrc_Codes vrc;
00919  int skey = 0;  /* Simplex format */
00920 
00921 
00922  switch (meshType) {
00923   case VML_EXTERNAL:
00924    if (sock == VNULL) {
00925     Vnm_print(2, "Vfetk_loadMesh:  Got NULL socket!\n");
00926     return VRC_FAILURE;
00927    }
00928    AM_read(thee->am, skey, sock);
00929    Vio_connectFree(sock);
00930    Vio_bufGive(sock);
00931    Vio_dtor(&sock);
00932    break;
00933   case VML_DIRICUBE:
00934    vrc = Vfetk_genCube(thee, center, length, meshType);
00935    if (vrc == VRC_FAILURE) return VRC_FAILURE;
00936    break;
00937   case VML_NEUMCUBE:
00938    vrc = Vfetk_genCube(thee, center, length, meshType);
00939    if (vrc == VRC_FAILURE) return VRC_FAILURE;
00940    break;
00941   default:
00942    Vnm_print(2, "Vfetk_loadMesh:  unrecognized mesh type (%d)!\n",
00943        meshType);
00944    return VRC_FAILURE;
00945  };
00946 
00947  /* Setup charge-simplex map */
00948     Vnm_print(0, "Vfetk_ctor2:  Constructing Vcsm...\n");
00949     thee->csm = VNULL;
00950     thee->csm = Vcsm_ctor(Vpbe_getValist(thee->pbe), thee->gm);
00951     VASSERT(thee->csm != VNULL);
00952     Vcsm_init(thee->csm);
00953 
00954  return VRC_SUCCESS;
```

```
00955 }
00956
00957
00958 VPUBLIC void Bmat_printHB( Bmat *thee, char *fname ) {
00959
00960     Mat *Ablock;
00961     MATsym pqsym;
00962     int i, j, jj;
00963     int *IA, *JA;
00964     double *D, *L, *U;
00965     FILE *fp;
00966
00967     char mmtitle[72];
00968     char mmkey[] = {"8charkey"};
00969     int totc = 0, ptrc = 0, indc = 0, valc = 0;
00970     char mxtyp[] = {"RUA"}; /* Real Unsymmetric Assembled */
00971     int nrow = 0, ncol = 0, numZ = 0;
00972     int numZdigits = 0, nrowdigits = 0;
00973     int nptrline = 8, nindline = 8, nvalline = 5;
00974     char ptrfmt[] = {"(8I10)          "}, ptrfmtstr[] = {"%10d"};
00975     char indfmt[] = {"(8I10)          "}, indfmtstr[] = {"%10d"};
00976     char valfmt[] = {"(5E16.8)            "}, valfmtstr[] = {"%16.8E"};
00977
00978     VASSERT( thee->numB == 1 );             /* HARDWIRE FOR NOW */
00979     Ablock = thee->AD[0][0];
00980
00981     VASSERT( Mat_format( Ablock ) == DRC_FORMAT );  /* HARDWIRE FOR NOW */
00982
00983     pqsym = Mat_sym( Ablock );
00984
00985     if ( pqsym == IS_SYM ) {
00986         mxtyp[1] = 'S';
00987     } else if ( pqsym == ISNOT_SYM ) {
00988         mxtyp[1] = 'U';
00989     } else {
00990         VASSERT( 0 ); /* NOT VALID */
00991     }
00992
00993     nrow = Bmat_numRT( thee ); /* Number of rows */
00994     ncol = Bmat_numCT( thee ); /* Number of cols */
00995     numZ = Bmat_numZT( thee ); /* Number of entries */
00996
00997     nrowdigits = (int) (log( nrow )/log( 10 )) + 1;
00998     numZdigits = (int) (log( numZ )/log( 10 )) + 1;
00999
01000     nptrline = (int) ( 80 / (numZdigits + 1) );
01001     nindline = (int) ( 80 / (nrowdigits + 1) );
01002
01003     sprintf(ptrfmt,"(%dI%d)",nptrline,numZdigits+1);
01004     sprintf(ptrfmtstr,"%%%dd",numZdigits+1);
01005     sprintf(indfmt,"(%dI%d)",nindline,nrowdigits+1);
01006     sprintf(indfmtstr,"%%%dd",nrowdigits+1);
01007
01008     ptrc = (int) ( ( (ncol + 1) - 1 ) / nptrline ) + 1;
01009     indc = (int) ( (numZ - 1) / nindline ) + 1;
01010     valc = (int) ( (numZ - 1) / nvalline ) + 1;
01011
```

```
01012     totc = ptrc + indc + valc;
01013
01014     sprintf( mmtitle, "Sparse '%s' Matrix - Harwell-Boeing Format - '%s'",
01015              thee->name, fname );
01016
01017    /* Step 0:  Open the file for writing */
01018
01019     fp = fopen( fname, "w" );
01020     if (fp == VNULL) {
01021         Vnm_print(2,"Bmat_printHB:  Ouch couldn't open file <%s>\n",fname);
01022         return;
01023     }
01024
01025     /* Step 1:  Print the header information */
01026
01027     fprintf( fp, "%-72s%-8s\n", mmtitle, mmkey );
01028     fprintf( fp, "%14d%14d%14d%14d%14d\n", totc, ptrc, indc, valc, 0 );
01029     fprintf( fp, "%3s%11s%14d%14d%14d\n", mxtyp, " ", nrow, ncol, numZ );
01030     fprintf( fp, "%-16s%-16s%-20s%-20s\n", ptrfmt, indfmt, valfmt, "6E13.5" );
01031
01032     IA = Ablock->IA;
01033     JA = Ablock->JA;
01034     D = Ablock->diag;
01035     L = Ablock->offL;
01036     U = Ablock->offU;
01037
01038     if ( pqsym == IS_SYM ) {
01039
01040         /* Step 2:  Print the pointer information */
01041
01042         for (i=0; i<(ncol+1); i++) {
01043             fprintf( fp, ptrfmtstr, Ablock->IA[i] + (i+1) );
01044             if ( ( (i+1) % nptrline ) == 0 ) {
01045                 fprintf( fp, "\n" );
01046             }
01047         }
01048
01049         if ( ( (ncol+1) % nptrline ) != 0 ) {
01050             fprintf( fp, "\n" );
01051         }
01052
01053         /* Step 3:  Print the index information */
01054
01055         j = 0;
01056         for (i=0; i<ncol; i++) {
01057             fprintf( fp, indfmtstr, i+1); /* diagonal */
01058             if ( ( (j+1) % nindline ) == 0 ) {
01059                 fprintf( fp, "\n" );
01060             }
01061             j++;
01062             for (jj=IA[i]; jj<IA[i+1]; jj++) {
01063                 fprintf( fp, indfmtstr, JA[jj] + 1 ); /* lower triangle */
01064                 if ( ( (j+1) % nindline ) == 0 ) {
01065                     fprintf( fp, "\n" );
01066                 }
01067                 j++;
01068             }
```

```
01069         }
01070
01071         if ( ( j % nindline ) != 0 ) {
01072             fprintf( fp, "\n" );
01073         }
01074
01075         /* Step 4:  Print the value information */
01076
01077         j = 0;
01078         for (i=0; i<ncol; i++) {
01079             fprintf( fp, valfmtstr, D[i] );
01080             if ( ( (j+1) % nvalline ) == 0 ) {
01081                 fprintf( fp, "\n" );
01082             }
01083             j++;
01084             for (jj=IA[i]; jj<IA[i+1]; jj++) {
01085                 fprintf( fp, valfmtstr, L[jj] );
01086                 if ( ( (j+1) % nvalline ) == 0 ) {
01087                     fprintf( fp, "\n" );
01088                 }
01089                 j++;
01090             }
01091         }
01092
01093         if ( ( j % nvalline ) != 0 ) {
01094             fprintf( fp, "\n" );
01095         }
01096
01097     } else { /* ISNOT_SYM */
01098
01099         VASSERT( 0 ); /* NOT CODED YET */
01100     }
01101
01102     /* Step 5:  Close the file */
01103     fclose( fp );
01104 }
01105
01106 VPUBLIC PDE* Vfetk_PDE_ctor(Vfetk *fetk) {
01107
01108     PDE *thee = VNULL;
01109
01110     thee = Vmem_malloc(fetk->vmem, 1, sizeof(PDE));
01111     VASSERT(thee != VNULL);
01112     VASSERT(Vfetk_PDE_ctor2(thee, fetk));
01113
01114     return thee;
01115 }
01116
01117 VPUBLIC int Vfetk_PDE_ctor2(PDE *thee, Vfetk *fetk) {
01118
01119     int i;
01120
01121     if (thee == VNULL) {
01122         Vnm_print(2, "Vfetk_PDE_ctor2:  Got NULL thee!\n");
01123         return 0;
01124     }
01125
```

```
01126     /* Store a local copy of the Vfetk class */
01127     var.fetk = fetk;
01128
01129     /* PDE-specific parameters and function pointers */
01130     thee->initAssemble = Vfetk_PDE_initAssemble;
01131     thee->initElement  = Vfetk_PDE_initElement;
01132     thee->initFace     = Vfetk_PDE_initFace;
01133     thee->initPoint    = Vfetk_PDE_initPoint;
01134     thee->Fu           = Vfetk_PDE_Fu;
01135     thee->Fu_v         = Vfetk_PDE_Fu_v;
01136     thee->DFu_wv       = Vfetk_PDE_DFu_wv;
01137     thee->delta        = Vfetk_PDE_delta;
01138     thee->u_D          = Vfetk_PDE_u_D;
01139     thee->u_T          = Vfetk_PDE_u_T;
01140     thee->Ju           = Vfetk_PDE_Ju;
01141     thee->vec          = 1; /* FIX! */
01142     thee->sym[0][0]    = 1;
01143     thee->est[0]       = 1.0;
01144     for (i=0; i<VMAX_BDTYPE; i++) thee->bmap[0][i] = i;
01145
01146     /* Manifold-specific function pointers */
01147     thee->bisectEdge  = Vfetk_PDE_bisectEdge;
01148     thee->mapBoundary = Vfetk_PDE_mapBoundary;
01149     thee->markSimplex = Vfetk_PDE_markSimplex;
01150     thee->oneChart    = Vfetk_PDE_oneChart;
01151
01152     /* Element-specific function pointers */
01153     thee->simplexBasisInit = Vfetk_PDE_simplexBasisInit;
01154     thee->simplexBasisForm = Vfetk_PDE_simplexBasisForm;
01155
01156     return 1;
01157 }
01158
01159 VPUBLIC void Vfetk_PDE_dtor(PDE **thee) {
01160
01161     if ((*thee) != VNULL) {
01162         Vfetk_PDE_dtor2(*thee);
01163   /* TODO: The following line is commented out because at the moment,
01164    there is a seg fault when deallocating at the end of a run. Since
01165    this routine is called only once at the very end, we'll leave it
01166    commented out. However, this could be a memory leak.
01167    */
01168         /* Vmem_free(var.fetk->vmem, 1, sizeof(PDE), (void **)thee); */
01169         (*thee) = VNULL;
01170     }
01171
01172 }
01173
01174 VPUBLIC void Vfetk_PDE_dtor2(PDE *thee) {
01175     var.fetk = VNULL;
01176 }
01177
01178 VPRIVATE double smooth(int nverts, double dist[VAPBS_NVS], double coeff[VAPBS_NVS
      ], int meth) {
01179
01180     int i;
01181     double weight;
```

```
01182     double num = 0.0;
01183     double den = 0.0;
01184
01185     for (i=0; i<nverts; i++) {
01186         if (dist[i] < VSMALL) return coeff[i];
01187         weight = 1.0/dist[i];
01188         if (meth == 0) {
01189             num += (weight * coeff[i]);
01190             den += weight;
01191         } else if (meth == 1) {
01192             /* Small coefficients reset the average to 0; we need to break out
01193              * of the loop */
01194             if (coeff[i] < VSMALL) {
01195                 num = 0.0;
01196                 break;
01197             } else {
01198                 num += weight; den += (weight/coeff[i]);
01199             }
01200         } else VASSERT(0);
01201     }
01202
01203     return (num/den);
01204
01205 }
01206
01207 VPRIVATE double diel() {
01208
01209     int i, j;
01210     double eps, epsp, epsw, dist[5], coeff[5], srad, swin, *vx;
01211     Vsurf_Meth srfm;
01212     Vacc *acc;
01213     PBEparm *pbeparm;
01214
01215     epsp = Vpbe_getSoluteDiel(var.fetk->pbe);
01216     epsw = Vpbe_getSolventDiel(var.fetk->pbe);
01217     VASSERT(var.fetk->pbeparm != VNULL);
01218     pbeparm = var.fetk->pbeparm;
01219     srfm = pbeparm->srfm;
01220     srad = pbeparm->srad;
01221     swin = pbeparm->swin;
01222     acc = var.fetk->pbe->acc;
01223
01224     eps = 0;
01225
01226     if (VABS(epsp - epsw) < VSMALL) return epsp;
01227     switch (srfm) {
01228         case VSM_MOL:
01229             eps = ((epsw-epsp)*Vacc_molAcc(acc, var.xq, srad) + epsp);
01230             break;
01231         case VSM_MOLSMOOTH:
01232             for (i=0; i<var.nverts; i++) {
01233                 dist[i] = 0;
01234                 vx = var.vx[i];
01235                 for (j=0; j<3; j++) {
01236                     dist[i] += VSQR(var.xq[j] - vx[j]);
01237                 }
01238                 dist[i] = VSQRT(dist[i]);
```

```
01239                 coeff[i] = (epsw-epsp)*Vacc_molAcc(acc, var.xq, srad) + epsp;
01240             }
01241             eps = smooth(var.nverts, dist, coeff, 1);
01242             break;
01243         case VSM_SPLINE:
01244             eps = ((epsw-epsp)*Vacc_splineAcc(acc, var.xq, swin, 0.0) + epsp);
01245             break;
01246         default:
01247             Vnm_print(2, "Undefined surface method (%d)!\n", srfm);
01248             VASSERT(0);
01249     }
01250 
01251     return eps;
01252 }
01253 
01254 VPRIVATE double ionacc() {
01255 
01256     int i, j;
01257     double dist[5], coeff[5], irad, swin, *vx, accval;
01258     Vsurf_Meth srfm;
01259     Vacc *acc = VNULL;
01260     PBEparm *pbeparm = VNULL;
01261 
01262     VASSERT(var.fetk->pbeparm != VNULL);
01263     pbeparm = var.fetk->pbeparm;
01264     srfm = pbeparm->srfm;
01265     irad = Vpbe_getMaxIonRadius(var.fetk->pbe);
01266     swin = pbeparm->swin;
01267     acc = var.fetk->pbe->acc;
01268 
01269     if (var.zks2 < VSMALL) return 0.0;
01270     switch (srfm) {
01271         case VSM_MOL:
01272             accval = Vacc_ivdwAcc(acc, var.xq, irad);
01273             break;
01274         case VSM_MOLSMOOTH:
01275             for (i=0; i<var.nverts; i++) {
01276                 dist[i] = 0;
01277                 vx = var.vx[i];
01278                 for (j=0; j<3; j++) {
01279                     dist[i] += VSQR(var.xq[j] - vx[j]);
01280                 }
01281                 dist[i] = VSQRT(dist[i]);
01282                 coeff[i] = Vacc_ivdwAcc(acc, var.xq, irad);
01283             }
01284             accval = smooth(var.nverts, dist, coeff, 1);
01285             break;
01286         case VSM_SPLINE:
01287             accval = Vacc_splineAcc(acc, var.xq, swin, irad);
01288             break;
01289         default:
01290             Vnm_print(2, "Undefined surface method (%d)!\n", srfm);
01291             VASSERT(0);
01292     }
01293 
01294     return accval;
01295 }
```

```
01296
01297 VPRIVATE double debye_U(Vpbe *pbe, int d, double x[]) {
01298
01299     double size, *position, charge, xkappa, eps_w, dist, T, pot, val;
01300     int iatom, i;
01301     Valist *alist;
01302     Vatom *atom;
01303
01304     eps_w = Vpbe_getSolventDiel(pbe);
01305     xkappa = (1.0e10)*Vpbe_getXkappa(pbe);
01306     T = Vpbe_getTemperature(pbe);
01307     alist = Vpbe_getValist(pbe);
01308     val = 0;
01309     pot = 0;
01310
01311     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
01312         atom = Valist_getAtom(alist, iatom);
01313         position = Vatom_getPosition(atom);
01314         charge = Vunit_ec*Vatom_getCharge(atom);
01315         size = (1e-10)*Vatom_getRadius(atom);
01316         dist = 0;
01317         for (i=0; i<d; i++) {
01318             dist += VSQR(position[i] - x[i]);
01319         }
01320         dist = (1.0e-10)*VSQRT(dist);
01321         val = (charge)/(4*VPI*Vunit_eps0*eps_w*dist);
01322         if (xkappa != 0.0) {
01323             val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
01324         }
01325         pot = pot + val;
01326     }
01327     pot = pot*Vunit_ec/(Vunit_kb*T);
01328
01329     return pot;
01330 }
01331
01332 VPRIVATE double debye_Udiff(Vpbe *pbe, int d, double x[]) {
01333
01334     double size, *position, charge, eps_p, dist, T, pot, val;
01335     double Ufull;
01336     int iatom, i;
01337     Valist *alist;
01338     Vatom *atom;
01339
01340     Ufull = debye_U(pbe, d, x);
01341
01342     eps_p = Vpbe_getSoluteDiel(pbe);
01343     T = Vpbe_getTemperature(pbe);
01344     alist = Vpbe_getValist(pbe);
01345     val = 0;
01346     pot = 0;
01347
01348     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
01349         atom = Valist_getAtom(alist, iatom);
01350         position = Vatom_getPosition(atom);
01351         charge = Vunit_ec*Vatom_getCharge(atom);
01352         size = (1e-10)*Vatom_getRadius(atom);
```

```
01353         dist = 0;
01354         for (i=0; i<d; i++) {
01355             dist += VSQR(position[i] - x[i]);
01356         }
01357         dist = (1.0e-10)*VSQRT(dist);
01358         val = (charge)/(4*VPI*Vunit_eps0*eps_p*dist);
01359         pot = pot + val;
01360     }
01361     pot = pot*Vunit_ec/(Vunit_kb*T);
01362 
01363     pot = Ufull - pot;
01364 
01365     return pot;
01366 }
01367 
01368 VPRIVATE void coulomb(Vpbe *pbe, int d, double pt[], double eps, double *U,
01369   double dU[], double *d2U) {
01370 
01371     int iatom, i;
01372     double T, pot, fx, fy, fz, x, y, z, scale;
01373     double *position, charge, dist, dist2, val, vec[3], dUold[3], Uold;
01374     Valist *alist;
01375     Vatom *atom;
01376 
01377     /* Initialize variables */
01378     T = Vpbe_getTemperature(pbe);
01379     alist = Vpbe_getValist(pbe);
01380     pot = 0;  fx = 0; fy = 0; fz = 0;
01381     x = pt[0]; y = pt[1]; z = pt[2];
01382 
01383     /* Calculate */
01384     if (!Vgreen_coulombD(var.green, 1, &x, &y, &z, &pot, &fx, &fy, &fz)) {
01385         Vnm_print(2, "Error calculating Green's function!\n");
01386         VASSERT(0);
01387     }
01388 
01389 
01390     /* Scale the results */
01391     scale = Vunit_ec/(eps*Vunit_kb*T);
01392     *U = pot*scale;
01393     *d2U = 0.0;
01394     dU[0] = -fx*scale;
01395     dU[1] = -fy*scale;
01396     dU[2] = -fz*scale;
01397 
01398 #if 0
01399     /* Compare with old results */
01400     val = 0.0;
01401     Uold = 0.0; dUold[0] = 0.0; dUold[1] = 0.0; dUold[2] = 0.0;
01402     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
01403        atom = Valist_getAtom(alist, iatom);
01404        position = Vatom_getPosition(atom);
01405        charge = Vatom_getCharge(atom);
01406        dist2 = 0;
01407        for (i=0; i<d; i++) {
01408            vec[i] = (position[i] - pt[i]);
01409            dist2 += VSQR(vec[i]);
```

```
01410         }
01411         dist = VSQRT(dist2);
01412 
01413         /* POTENTIAL */
01414         Uold = Uold + charge/dist;
01415 
01416         /* GRADIENT */
01417         for (i=0; i<d; i++) dUold[i] = dUold[i] + vec[i]*charge/(dist2*dist);
01418 
01419     }
01420     Uold = Uold*VSQR(Vunit_ec)*(1.0e10)/(4*VPI*Vunit_eps0*eps*Vunit_kb*T);
01421     for (i=0; i<d; i++) {
01422         dUold[i] = dUold[i]*VSQR(Vunit_ec)*(1.0e10)/(4*VPI*Vunit_eps0*eps*
      Vunit_kb*T);
01423     }
01424 
01425     printf("Unew - Uold = %g - %g = %g\n", *U, Uold, (*U - Uold));
01426     printf("||dUnew - dUold||^2 = %g\n", (VSQR(dU[0] - dUold[0])
01427                 + VSQR(dU[1] - dUold[1]) + VSQR(dU[2] - dUold[2])));
01428     printf("dUnew[0] = %g, dUold[0] = %g\n", dU[0], dUold[0]);
01429     printf("dUnew[1] = %g, dUold[1] = %g\n", dU[1], dUold[1]);
01430     printf("dUnew[2] = %g, dUold[2] = %g\n", dU[2], dUold[2]);
01431 
01432 #endif
01433 
01434 }
01435 
01436 VPUBLIC void Vfetk_PDE_initAssemble(PDE *thee, int ip[], double rp[]) {
01437 
01438 #if 1
01439     /* Re-initialize the Green's function oracle in case the atom list has
01440      * changed */
01441     if (var.initGreen) {
01442         Vgreen_dtor(&(var.green));
01443         var.initGreen = 0;
01444     }
01445     var.green = Vgreen_ctor(var.fetk->pbe->alist);
01446     var.initGreen = 1;
01447 #else
01448     if (!var.initGreen) {
01449         var.green = Vgreen_ctor(var.fetk->pbe->alist);
01450         var.initGreen = 1;
01451     }
01452 #endif
01453 
01454 }
01455 
01456 VPUBLIC void Vfetk_PDE_initElement(PDE *thee, int elementType, int chart,
01457   double tvx[][3], void *data) {
01458 
01459     int i, j;
01460     double epsp, epsw;
01461 
01462     /* We assume that the simplex has been passed in as the void *data * *
01463      * argument.  Store it */
01464     VASSERT(data != NULL);
01465     var.simp = (SS *)data;
```

```
01466
01467     /* save the element type */
01468     var.sType = elementType;
01469
01470     /* Grab the vertices from this simplex */
01471     var.nverts = thee->dim+1;
01472     for (i=0; i<thee->dim+1; i++) var.verts[i] = SS_vertex(var.simp, i);
01473
01474     /* Vertex locations of this simplex */
01475     for (i=0; i<thee->dim+1; i++) {
01476         for (j=0; j<thee->dim; j++) {
01477             var.vx[i][j] = tvx[i][j];
01478         }
01479     }
01480
01481     /* Set the dielectric constant for this element for use in the jump term *
01482      * of the residual-based error estimator.  The value is set to the average
01483      * * value of the vertices */
01484     var.jumpDiel = 0;  /* NOT IMPLEMENTED YET! */
01485 }
01486
01487 VPUBLIC void Vfetk_PDE_initFace(PDE *thee, int faceType, int chart,
01488   double tnvec[]) {
01489
01490     int i;
01491
01492     /* unit normal vector of this face */
01493     for (i=0; i<thee->dim; i++) var.nvec[i] = tnvec[i];
01494
01495     /* save the face type */
01496     var.fType = faceType;
01497 }
01498
01499 VPUBLIC void Vfetk_PDE_initPoint(PDE *thee, int pointType, int chart,
01500   double txq[], double tU[], double tdU[][3]) {
01501
01502     int i, j, ichop;
01503     double u2, coef2, eps_p;
01504     Vhal_PBEType pdetype;
01505     Vpbe *pbe = VNULL;
01506
01507     eps_p = Vpbe_getSoluteDiel(var.fetk->pbe);
01508     pdetype = var.fetk->type;
01509     pbe = var.fetk->pbe;
01510
01511     /* the point, the solution value and gradient, and the Coulomb value and *
01512      * gradient at the point */
01513     if ((pdetype == PBE_LRPBE) || (pdetype == PBE_NRPBE)) {
01514         coulomb(pbe, thee->dim, txq, eps_p, &(var.W), var.dW, &(var.d2W));
01515     }
01516     for (i=0; i<thee->vec; i++) {
01517         var.U[i] = tU[i];
01518         for (j=0; j<thee->dim; j++) {
01519             var.xq[j] = txq[j];
01520             var.dU[i][j] = tdU[i][j];
01521         }
01522     }
```

```
01523
01524     /* interior form case */
01525     if (pointType == 0) {
01526
01527         /* Get the dielectric values */
01528         var.diel  = diel();
01529         var.ionacc  = ionacc();
01530         var.A = var.diel;
01531         var.F = (var.diel - eps_p);
01532
01533         switch (pdetype) {
01534
01535             case PBE_LPBE:
01536                 var.DB = var.ionacc*var.zkappa2*var.ionstr;
01537                 var.B = var.DB*var.U[0];
01538                 break;
01539
01540             case PBE_NPBE:
01541
01542                 var.B  = 0;
01543                 var.DB  = 0;
01544                 if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01545                     for (i=0; i<var.nion; i++) {
01546                         u2 = -1.0 * var.U[0] * var.ionQ[i];
01547
01548                         /* NONLINEAR TERM */
01549                         coef2 = -1.0 * var.ionacc * var.zks2 * var.ionConc[i];
01550                         var.B += (coef2 * Vcap_exp(u2, &ichop));
01551                         /* LINEARIZED TERM */
01552                         coef2 = -1.0 * var.ionQ[i] * coef2;
01553                         var.DB += (coef2 * Vcap_exp(u2, &ichop));
01554                     }
01555                 }
01556                 break;
01557
01558             case PBE_LRPBE:
01559                 var.DB = var.ionacc*var.zkappa2*var.ionstr;
01560                 var.B = var.DB*(var.U[0]+var.W);
01561                 break;
01562
01563             case PBE_NRPBE:
01564
01565                 var.B  = 0;
01566                 var.DB  = 0;
01567                 if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01568                     for (i=0; i<var.nion; i++) {
01569                         u2 = -1.0 * (var.U[0] + var.W) * var.ionQ[i];
01570
01571                         /* NONLINEAR TERM */
01572                         coef2 = -1.0 * var.ionacc * var.zks2 * var.ionConc[i];
01573                         var.B += (coef2 * Vcap_exp(u2, &ichop));
01574
01575                         /* LINEARIZED TERM */
01576                         coef2 = -1.0 * var.ionQ[i] * coef2;
01577                         var.DB += (coef2 * Vcap_exp(u2, &ichop));
01578                     }
01579                 }
```

```
01580                 break;
01581
01582     case PBE_SMPBE: /* SMPBE Temp */
01583
01584                 var.B  = 0;
01585                 var.DB  = 0;
01586                 if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01587                     for (i=0; i<var.nion; i++) {
01588                         u2 = -1.0 * var.U[0] * var.ionQ[i];
01589
01590                         /* NONLINEAR TERM */
01591                         coef2 = -1.0 * var.ionacc * var.zks2 * var.ionConc[i];
01592                         var.B += (coef2 * Vcap_exp(u2, &ichop));
01593                         /* LINEARIZED TERM */
01594                         coef2 = -1.0 * var.ionQ[i] * coef2;
01595                         var.DB += (coef2 * Vcap_exp(u2, &ichop));
01596                     }
01597                 }
01598      break;
01599             default:
01600                 Vnm_print(2, "Vfetk_PDE_initPoint:  Unknown PBE type (%d)!\n",
01601                   pdetype);
01602                 VASSERT(0);
01603                 break;
01604         }
01605
01606
01607     /* boundary form case */
01608     } else {
01609 #ifdef DONEUMANN
01610         ;
01611 #else
01612         Vnm_print(2, "Vfetk:  Whoa!  I just got a boundary point to evaluate (%d)
      !\n", pointType);
01613         Vnm_print(2, "Vfetk:  Did you do that on purpose?\n");
01614 #endif
01615     }
01616
01617 #if 0 /* THIS IS VERY NOISY! */
01618     Vfetk_dumpLocalVar();
01619 #endif
01620
01621 }
01622
01623 VPUBLIC void Vfetk_PDE_Fu(PDE *thee, int key, double F[]) {
01624
01625     //Vnm_print(2, "Vfetk_PDE_Fu:  Setting error to zero!\n");
01626
01627     F[0] = 0.;
01628
01629 }
01630
01631 VPUBLIC double Vfetk_PDE_Fu_v(
01632         PDE *thee,
01633         int key,
01634         double V[],
01635         double dV[][VAPBS_DIM]
```

```
01636         ) {
01637 
01638     Vhal_PBEType type;
01639     int i;
01640     double value = 0.;
01641 
01642     type = var.fetk->type;
01643 
01644     /* interior form case */
01645     if (key == 0) {
01646 
01647         for (i=0; i<thee->dim; i++) value += ( var.A * var.dU[0][i] * dV[0][i] );

01648         value += var.B * V[0];
01649 
01650         if ((type == PBE_LRPBE) || (type == PBE_NRPBE)) {
01651             for (i=0; i<thee->dim; i++) {
01652                 if (var.F > VSMALL) value += (var.F * var.dW[i] * dV[0][i]);
01653             }
01654         }
01655 
01656     /* boundary form case */
01657     } else {
01658 #ifdef DONEUMANN
01659         value = 0.0;
01660 #else
01661         Vnm_print(2, "Vfetk:  Whoa! I was just asked to evaluate a boundary weak
      form for point type %d!\n", key);
01662 #endif
01663     }
01664 
01665     var.Fu_v = value;
01666     return value;
01667 }
01668 
01669 VPUBLIC double Vfetk_PDE_DFu_wv(
01670         PDE *thee,
01671         int key,
01672         double W[],
01673         double dW[][VAPBS_DIM],
01674         double V[],
01675         double dV[][3]
01676         ) {
01677 
01678     Vhal_PBEType type;
01679     int i;
01680     double value = 0.;
01681 
01682     type = var.fetk->type;
01683 
01684     /* Interior form */
01685     if (key == 0) {
01686             value += var.DB * W[0] * V[0];
01687             for (i=0; i<thee->dim; i++) value += ( var.A * dW[0][i] * dV[0][i] );

01688 
01689     /* boundary form case */
```

```
01690     } else {
01691 #ifdef DONEUMANN
01692         value = 0.0;
01693 #else
01694         Vnm_print(2, "Vfetk:  Whoa! I was just asked to evaluate a boundary weak
      form for point type %d!\n", key);
01695 #endif
01696     }
01697 
01698     var.DFu_wv = value;
01699     return value;
01700 }
01701 
01704 #define VRINGMAX 1000
01705 
01707 #define VATOMMAX 1000000
01708 VPUBLIC void Vfetk_PDE_delta(PDE *thee, int type, int chart, double txq[],
01709   void *user, double F[]) {
01710 
01711     int iatom, jatom, natoms, atomIndex, atomList[VATOMMAX], nAtomList;
01712     int gotAtom, numSring, isimp, ivert, sid;
01713     double *position, charge, phi[VAPBS_NVS], phix[VAPBS_NVS][3], value;
01714     Vatom *atom;
01715     Vhal_PBEType pdetype;
01716     SS *sring[VRINGMAX];
01717     VV *vertex = (VV *)user;
01718 
01719     pdetype = var.fetk->type;
01720 
01721     F[0] = 0.0;
01722 
01723     if ((pdetype == PBE_LPBE) || (pdetype == PBE_NPBE) || (pdetype == PBE_SMPBE)
      /* SMPBE Added */) {
01724         VASSERT( vertex != VNULL);
01725         numSring = 0;
01726         sring[numSring] = VV_firstSS(vertex);
01727         while (sring[numSring] != VNULL) {
01728             numSring++;
01729             sring[numSring] = SS_link(sring[numSring-1], vertex);
01730         }
01731         VASSERT( numSring > 0 );
01732         VASSERT( numSring <= VRINGMAX );
01733 
01734         /* Move around the simplex ring and determine the charge locations */
01735         F[0] = 0.;
01736         charge = 0.;
01737         nAtomList = 0;
01738         for (isimp=0; isimp<numSring; isimp++) {
01739             sid = SS_id(sring[isimp]);
01740             natoms = Vcsm_getNumberAtoms(Vfetk_getVcsm(var.fetk), sid);
01741             for (iatom=0; iatom<natoms; iatom++) {
01742                 /* Get the delta function information * */
01743                 atomIndex = Vcsm_getAtomIndex(Vfetk_getVcsm(var.fetk),
01744                   iatom, sid);
01745                 gotAtom = 0;
01746                 for (jatom=0; jatom<nAtomList; jatom++) {
01747                     if (atomList[jatom] == atomIndex) {
```

```
01748                         gotAtom = 1;
01749                         break;
01750                     }
01751                 }
01752                 if (!gotAtom) {
01753                     VASSERT(nAtomList < VATOMMAX);
01754                     atomList[nAtomList] = atomIndex;
01755                     nAtomList++;
01756 
01757                     atom = Vcsm_getAtom(Vfetk_getVcsm(var.fetk), iatom, sid);
01758                     charge = Vatom_getCharge(atom);
01759                     position = Vatom_getPosition(atom);
01760 
01761                     /* Get the test function value at the delta function I
01762                      * used to do a VASSERT to make sure the point was in the
01763                      * simplex (i.e., make sure round-off error isn't an
01764                      * issue), but round off errors became an issue */
01765                     if (!Gem_pointInSimplexVal(Vfetk_getGem(var.fetk),
01766                       sring[isimp], position, phi, phix)) {
01767                         if (!Gem_pointInSimplex(Vfetk_getGem(var.fetk),
01768                           sring[isimp], position)) {
01769                             Vnm_print(2, "delta: Both Gem_pointInSimplexVal \
01770 and Gem_pointInSimplex detected misplaced point charge!\n");
01771                             Vnm_print(2, "delta: I think you have problems: \
01772 phi = {");
01773                             for (ivert=0; ivert<Gem_dimVV(Vfetk_getGem(var.fetk))
      ; ivert++) Vnm_print(2, "%e ", phi[ivert]);
01774 
                              Vnm_print(2, "}\n");
01775                         }
01776                     }
01777                     value = 0;
01778                     for (ivert=0; ivert<Gem_dimVV(Vfetk_getGem(var.fetk)); ivert+
      +) {
01779                         if (VV_id(SS_vertex(sring[isimp], ivert)) == VV_id(vertex
      )) value += phi[ivert];
01780                     }
01781 
01782                     F[0] += (value * Vpbe_getZmagic(var.fetk->pbe) * charge);
01783                 } /* if !gotAtom */
01784             } /* for iatom */
01785         } /* for isimp */
01786 
01787     } else if ((pdetype == PBE_LRPBE) || (pdetype == PBE_NRPBE)) {
01788         F[0] = 0.0;
01789     } else { VASSERT(0); }
01790 
01791     var.delta = F[0];
01792 
01793 }
01794 
01795 VPUBLIC void Vfetk_PDE_u_D(PDE *thee, int type, int chart, double txq[],
01796   double F[]) {
01797 
01798     if ((var.fetk->type == PBE_LPBE) || (var.fetk->type == PBE_NPBE) || (var.
      fetk->type == PBE_SMPBE) /* SMPBE Added */) {
01799         F[0] = debye_U(var.fetk->pbe, thee->dim, txq);
```

```
01800     } else if ((var.fetk->type == PBE_LRPBE) || (var.fetk->type == PBE_NRPBE)) {
01801         F[0] = debye_Udiff(var.fetk->pbe, thee->dim, txq);
01802     } else VASSERT(0);
01803 
01804     var.u_D = F[0];
01805 
01806 }
01807 
01808 VPUBLIC void Vfetk_PDE_u_T(PDE *thee, int type, int chart, double txq[],
01809   double F[]) {
01810 
01811     F[0] = 0.0;
01812     var.u_T = F[0];
01813 
01814 }
01815 
01816 
01817 VPUBLIC void Vfetk_PDE_bisectEdge(int dim, int dimII, int edgeType,
01818   int chart[], double vx[][3]) {
01819 
01820     int i;
01821 
01822     for (i=0; i<dimII; i++) vx[2][i] = .5 * (vx[0][i] + vx[1][i]);
01823     chart[2] = chart[0];
01824 
01825 }
01826 
01827 VPUBLIC void Vfetk_PDE_mapBoundary(int dim, int dimII, int vertexType,
01828   int chart, double vx[3]) {
01829 
01830 }
01831 
01832 VPUBLIC int Vfetk_PDE_markSimplex(int dim, int dimII, int simplexType,
01833   int faceType[VAPBS_NVS], int vertexType[VAPBS_NVS], int chart[], double vx[][3]
      ,
01834   void *simplex) {
01835 
01836     double targetRes, edgeLength, srad, swin, myAcc, refAcc;
01837     int i, natoms;
01838     Vsurf_Meth srfm;
01839     Vhal_PBEType type;
01840     FEMparm *feparm = VNULL;
01841     PBEparm *pbeparm = VNULL;
01842     Vpbe *pbe = VNULL;
01843     Vacc *acc = VNULL;
01844     Vcsm *csm = VNULL;
01845     SS *simp = VNULL;
01846 
01847     VASSERT(var.fetk->feparm != VNULL);
01848     feparm = var.fetk->feparm;
01849     VASSERT(var.fetk->pbeparm != VNULL);;
01850     pbeparm = var.fetk->pbeparm;
01851     pbe = var.fetk->pbe;
01852     csm = Vfetk_getVcsm(var.fetk);
01853     acc = pbe->acc;
01854     targetRes = feparm->targetRes;
01855     srfm = pbeparm->srfm;
```

```
01856     srad = pbeparm->srad;
01857     swin = pbeparm->swin;
01858     simp = (SS *)simplex;
01859     type = var.fetk->type;
01860 
01861     /* Check to see if this simplex is smaller than the target size */
01862     /* 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. */
01863     Gem_longestEdge(var.fetk->gm, simp, -1, &edgeLength);
01864     if (edgeLength < targetRes) return 0;
01865 
01866     /* For non-regularized PBE, check charge-simplex map */
01867     if ((type == PBE_LPBE) || (type == PBE_NPBE) || (type == PBE_SMPBE)  /* SMPBE
       Added */) {
01868         natoms = Vcsm_getNumberAtoms(csm, SS_id(simp));
01869         if (natoms > 0) {
01870             return 1;
01871         }
01872     }
01873 
01874     /* We would like to resolve the mesh between the van der Waals surface the
01875      * max distance from this surface where there could be coefficient
01876      * changes */
01877     switch(srfm) {
01878         case VSM_MOL:
01879             refAcc = Vacc_molAcc(acc, vx[0], srad);
01880             for (i=1; i<(dim+1); i++) {
01881                 myAcc = Vacc_molAcc(acc, vx[i], srad);
01882                 if (myAcc != refAcc) {
01883                     return 1;
01884                 }
01885             }
01886             break;
01887         case VSM_MOLSMOOTH:
01888             refAcc = Vacc_molAcc(acc, vx[0], srad);
01889             for (i=1; i<(dim+1); i++) {
01890                 myAcc = Vacc_molAcc(acc, vx[i], srad);
01891                 if (myAcc != refAcc) {
01892                     return 1;
01893                 }
01894             }
01895             break;
01896         case VSM_SPLINE:
01897             refAcc = Vacc_splineAcc(acc, vx[0], swin, 0.0);
01898             for (i=1; i<(dim+1); i++) {
01899                 myAcc = Vacc_splineAcc(acc, vx[i], swin, 0.0);
01900                 if (myAcc != refAcc) {
01901                     return 1;
01902                 }
01903             }
01904             break;
01905         default:
01906             VASSERT(0);
01907             break;
01908     }
01909 
01910     return 0;
```

```
01911 }
01912
01913 VPUBLIC void Vfetk_PDE_oneChart(int dim, int dimII, int objType, int chart[],
01914   double vx[][3], int dimV) {
01915
01916 }
01917
01918 VPUBLIC double Vfetk_PDE_Ju(PDE *thee, int key) {
01919
01920     int i, ichop;
01921     double dielE, qmE, coef2, u2;
01922     double value = 0.;
01923     Vhal_PBEType type;
01924
01925     type = var.fetk->type;
01926
01927     /* interior form case */
01928     if (key == 0) {
01929         dielE = 0;
01930         for (i=0; i<3; i++) dielE += VSQR(var.dU[0][i]);
01931         dielE = dielE*var.diel;
01932
01933         switch (type) {
01934             case PBE_LPBE:
01935                 if ((var.ionacc > VSMALL) && (var.zkappa2 > VSMALL)) {
01936                     qmE = var.ionacc*var.zkappa2*VSQR(var.U[0]);
01937                 } else qmE = 0;
01938                 break;
01939             case PBE_NPBE:
01940                 if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01941                     qmE = 0.;
01942                     for (i=0; i<var.nion; i++) {
01943                         coef2 = var.ionacc * var.zks2 * var.ionConc[i] * var.
      ionQ[i];
01944                         u2 = -1.0 * (var.U[0]) * var.ionQ[i];
01945                         qmE += (coef2 * (Vcap_exp(u2, &ichop) - 1.0));
01946                     }
01947                 } else qmE = 0;
01948                 break;
01949             case PBE_LRPBE:
01950                 if ((var.ionacc > VSMALL) && (var.zkappa2 > VSMALL)) {
01951                     qmE = var.ionacc*var.zkappa2*VSQR((var.U[0] + var.W));
01952                 } else qmE = 0;
01953                 break;
01954             case PBE_NRPBE:
01955                 if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01956                     qmE = 0.;
01957                     for (i=0; i<var.nion; i++) {
01958                         coef2 = var.ionacc * var.zks2 * var.ionConc[i] * var.
      ionQ[i];
01959                         u2 = -1.0 * (var.U[0] + var.W) * var.ionQ[i];
01960                         qmE += (coef2 * (Vcap_exp(u2, &ichop) - 1.0));
01961                     }
01962                 } else qmE = 0;
01963                 break;
01964    case PBE_SMPBE: /* SMPBE Temp */
01965                 if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
```

```
01966                     qmE = 0.;
01967                     for (i=0; i<var.nion; i++) {
01968                         coef2 = var.ionacc * var.zks2 * var.ionConc[i] * var.
      ionQ[i];
01969                         u2 = -1.0 * (var.U[0]) * var.ionQ[i];
01970                         qmE += (coef2 * (Vcap_exp(u2, &ichop) - 1.0));
01971                     }
01972                 } else qmE = 0;
01973                 break;
01974             default:
01975                 Vnm_print(2, "Vfetk_PDE_Ju:  Invalid PBE type (%d)!\n", type);
01976                 VASSERT(0);
01977                 break;
01978         }
01979
01980         value = 0.5*(dielE + qmE)/Vpbe_getZmagic(var.fetk->pbe);
01981
01982     /* boundary form case */
01983     } else if (key == 1) {
01984         value = 0.0;
01985
01986     /* how did we get here? */
01987     } else VASSERT(0);
01988
01989     return value;
01990
01991 }
01992
01993 VPUBLIC void Vfetk_externalUpdateFunction(SS **simps, int num) {
01994
01995     Vcsm *csm = VNULL;
01996     int rc;
01997
01998     VASSERT(var.fetk != VNULL);
01999     csm = Vfetk_getVcsm(var.fetk);
02000     VASSERT(csm != VNULL);
02001
02002     rc = Vcsm_update(csm, simps, num);
02003
02004     if (!rc) {
02005         Vnm_print(2, "Error while updating charge-simplex map!\n");
02006         VASSERT(0);
02007     }
02008 }
02009
02010 VPRIVATE void polyEval(int numP, double p[], double c[][VMAXP], double xv[]) {
02011     int i;
02012     double x, y, z;
02013
02014     x = xv[0];
02015     y = xv[1];
02016     z = xv[2];
02017     for (i=0; i<numP; i++) {
02018         p[i] = c[i][0]
02019              + c[i][1]  * x
02020              + c[i][2]  * y
02021              + c[i][3]  * z
```

```
02022               + c[i][4]  * x*x
02023               + c[i][5]  * y*y
02024               + c[i][6]  * z*z
02025               + c[i][7]  * x*y
02026               + c[i][8]  * x*z
02027               + c[i][9]  * y*z
02028               + c[i][10] * x*x*x
02029               + c[i][11] * y*y*y
02030               + c[i][12] * z*z*z
02031               + c[i][13] * x*x*y
02032               + c[i][14] * x*x*z
02033               + c[i][15] * x*y*y
02034               + c[i][16] * y*y*z
02035               + c[i][17] * x*z*z
02036               + c[i][18] * y*z*z;
02037     }
02038 }
02039
02040 VPRIVATE void setCoef(int numP, double c[][VMAXP], double cx[][VMAXP],
02041   double cy[][VMAXP], double cz[][VMAXP], int ic[][VMAXP], int icx[][VMAXP],
02042   int icy[][VMAXP], int icz[][VMAXP]) {
02043
02044     int i, j;
02045     for (i=0; i<numP; i++) {
02046         for (j=0; j<VMAXP; j++) {
02047             c[i][j]  = 0.5 * (double)ic[i][j];
02048             cx[i][j] = 0.5 * (double)icx[i][j];
02049             cy[i][j] = 0.5 * (double)icy[i][j];
02050             cz[i][j] = 0.5 * (double)icz[i][j];
02051         }
02052     }
02053 }
02054
02055 VPUBLIC int Vfetk_PDE_simplexBasisInit(int key, int dim, int comp, int *ndof,
02056   int dof[]) {
02057
02058     int qorder, bump, dimIS[VAPBS_NVS];
02059
02060     /* necessary quadrature order to return at the end */
02061     qorder = P_DEG;
02062
02063     /* deal with bump function requests */
02064     if ((key == 0) || (key == 1)) {
02065         bump = 0;
02066     } else if ((key == 2) || (key == 3)) {
02067         bump = 1;
02068     } else { VASSERT(0); }
02069
02070     /* for now use same element for all components, both trial and test */
02071     if (dim==2) {
02072         /* 2D simplex dimensions */
02073         dimIS[0] = 3;  /* number of vertices             */
02074         dimIS[1] = 3;  /* number of edges                */
02075         dimIS[2] = 0;  /* number of faces (3D only)      */
02076         dimIS[3] = 1;  /* number of simplices (always=1) */
02077         if (bump==0) {
02078             if (P_DEG==1) {
```

```
02079                 init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
02080             } else if (P_DEG==2) {
02081                 init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
02082             } else if (P_DEG==3) {
02083                 init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
02084             } else Vnm_print(2, "..bad order..");
02085         } else if (bump==1) {
02086             if (P_DEG==1) {
02087                 init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
02088             } else Vnm_print(2, "..bad order..");
02089         } else Vnm_print(2, "..bad bump..");
02090     } else if (dim==3) {
02091         /* 3D simplex dimensions */
02092         dimIS[0] = 4;  /* number of vertices             */
02093         dimIS[1] = 6;  /* number of edges                */
02094         dimIS[2] = 4;  /* number of faces (3D only)      */
02095         dimIS[3] = 1;  /* number of simplices (always=1) */
02096         if (bump==0) {
02097             if (P_DEG==1) {
02098                 init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
02099             } else if (P_DEG==2) {
02100                 init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
02101             } else if (P_DEG==3) {
02102                 init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
02103             } else Vnm_print(2, "..bad order..");
02104         } else if (bump==1) {
02105             if (P_DEG==1) {
02106                 init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
02107             } else Vnm_print(2, "..bad order..");
02108         } else Vnm_print(2, "..bad bump..");
02109     } else Vnm_print(2, "..bad dimension..");
02110
02111     /* save number of DF */
02112     numP = *ndof;
02113
02114     /* return the required quarature order */
02115     return qorder;
02116 }
02117
02118 VPUBLIC void Vfetk_PDE_simplexBasisForm(int key, int dim, int comp, int pdkey,
02119   double xq[], double basis[]) {
02120
02121     if (pdkey == 0) {
02122         polyEval(numP, basis, c, xq);
02123     } else if (pdkey == 1) {
02124         polyEval(numP, basis, cx, xq);
02125     } else if (pdkey == 2) {
02126         polyEval(numP, basis, cy, xq);
02127     } else if (pdkey == 3) {
02128         polyEval(numP, basis, cz, xq);
02129     } else { VASSERT(0); }
02130 }
02131
02132 VPRIVATE void init_2DP1(int dimIS[], int *ndof, int dof[], double c[][VMAXP],
02133   double cx[][VMAXP], double cy[][VMAXP], double cz[][VMAXP]) {
02134
02135     int i;
```

```
02136
02137     /* dof number and locations */
02138     dof[0] = 1;
02139     dof[1] = 0;
02140     dof[2] = 0;
02141     dof[3] = 0;
02142     *ndof  = 0;
02143     for (i=0; i<VAPBS_NVS; i++) *ndof += dimIS[i] * dof[i];
02144     VASSERT( *ndof == dim_2DP1 );
02145     VASSERT( *ndof <= VMAXP );
02146
02147     /* coefficients of the polynomials */
02148     setCoef( *ndof, c, cx, cy, cz, lgr_2DP1, lgr_2DP1x, lgr_2DP1y, lgr_2DP1z );
02149 }
02150
02151 VPRIVATE void init_3DP1(int dimIS[], int *ndof, int dof[], double c[][VMAXP],
02152   double cx[][VMAXP], double cy[][VMAXP], double cz[][VMAXP]) {
02153
02154     int i;
02155
02156     /* dof number and locations */
02157     dof[0] = 1;
02158     dof[1] = 0;
02159     dof[2] = 0;
02160     dof[3] = 0;
02161     *ndof  = 0;
02162     for (i=0; i<VAPBS_NVS; i++) *ndof += dimIS[i] * dof[i];
02163     VASSERT( *ndof == dim_3DP1 );
02164     VASSERT( *ndof <= VMAXP );
02165
02166     /* coefficients of the polynomials */
02167     setCoef( *ndof, c, cx, cy, cz, lgr_3DP1, lgr_3DP1x, lgr_3DP1y, lgr_3DP1z );
02168 }
02169
02170 VPUBLIC void Vfetk_dumpLocalVar() {
02171
02172     int i;
02173
02174     Vnm_print(1, "DEBUG: nvec = (%g, %g, %g)\n", var.nvec[0], var.nvec[1],
02175       var.nvec[2]);
02176     Vnm_print(1, "DEBUG: nverts = %d\n", var.nverts);
02177     for (i=0; i<var.nverts; i++) {
02178         Vnm_print(1, "DEBUG: verts[%d] ID = %d\n", i, VV_id(var.verts[i]));
02179         Vnm_print(1, "DEBUG: vx[%d] = (%g, %g, %g)\n", i, var.vx[i][0],
02180           var.vx[i][1], var.vx[i][2]);
02181     }
02182     Vnm_print(1, "DEBUG: simp ID = %d\n", SS_id(var.simp));
02183     Vnm_print(1, "DEBUG: sType = %d\n", var.sType);
02184     Vnm_print(1, "DEBUG: fType = %d\n", var.fType);
02185     Vnm_print(1, "DEBUG: xq = (%g, %g, %g)\n", var.xq[0], var.xq[1], var.xq[2]);
02186     Vnm_print(1, "DEBUG: U[0] = %g\n", var.U[0]);
02187     Vnm_print(1, "DEBUG: dU[0] = (%g, %g, %g)\n", var.dU[0][0], var.dU[0][1],
02188       var.dU[0][2]);
02189     Vnm_print(1, "DEBUG: W = %g\n", var.W);
02190     Vnm_print(1, "DEBUG: d2W = %g\n", var.d2W);
02191     Vnm_print(1, "DEBUG: dW = (%g, %g, %g)\n", var.dW[0], var.dW[1], var.dW[2]);
02192     Vnm_print(1, "DEBUG: diel = %g\n", var.diel);
```

```
02193     Vnm_print(1, "DEBUG: ionacc = %g\n", var.ionacc);
02194     Vnm_print(1, "DEBUG: A = %g\n", var.A);
02195     Vnm_print(1, "DEBUG: F = %g\n", var.F);
02196     Vnm_print(1, "DEBUG: B = %g\n", var.B);
02197     Vnm_print(1, "DEBUG: DB = %g\n", var.DB);
02198     Vnm_print(1, "DEBUG: nion = %d\n", var.nion);
02199     for (i=0; i<var.nion; i++) {
02200         Vnm_print(1, "DEBUG: ionConc[%d] = %g\n", i, var.ionConc[i]);
02201         Vnm_print(1, "DEBUG: ionQ[%d] = %g\n", i, var.ionQ[i]);
02202         Vnm_print(1, "DEBUG: ionRadii[%d] = %g\n", i, var.ionRadii[i]);
02203     }
02204     Vnm_print(1, "DEBUG: zkappa2 = %g\n", var.zkappa2);
02205     Vnm_print(1, "DEBUG: zks2 = %g\n", var.zks2);
02206     Vnm_print(1, "DEBUG: Fu_v = %g\n", var.Fu_v);
02207     Vnm_print(1, "DEBUG: DFu_wv = %g\n", var.DFu_wv);
02208     Vnm_print(1, "DEBUG: delta = %g\n", var.delta);
02209     Vnm_print(1, "DEBUG: u_D = %g\n", var.u_D);
02210     Vnm_print(1, "DEBUG: u_T = %g\n", var.u_T);
02211
02212 };
02213
02214 VPUBLIC int Vfetk_fillArray(Vfetk *thee, Bvec *vec, Vdata_Type type) {
02215
02216     int i, j, ichop;
02217     double coord[3], chi, q, conc, val;
02218     VV *vert;
02219     Bvec *u, *u_d;
02220     AM *am;
02221     Gem *gm;
02222     PBEparm *pbeparm;
02223     Vacc *acc;
02224     Vpbe *pbe;
02225
02226     gm = thee->gm;
02227     am = thee->am;
02228     pbe = thee->pbe;
02229     pbeparm = thee->pbeparm;
02230     acc = pbe->acc;
02231
02232     /* Make sure vec has enough rows to accomodate the vertex data */
02233     if (Bvec_numRB(vec, 0) != Gem_numVV(gm)) {
02234         Vnm_print(2, "Vfetk_fillArray:  insufficient space in Bvec!\n");
02235         Vnm_print(2, "Vfetk_fillArray:  Have %d, need %d!\n", Bvec_numRB(vec, 0),

02236           Gem_numVV(gm));
02237         return 0;
02238     }
02239
02240     switch (type) {
02241
02242         case VDT_CHARGE:
02243             Vnm_print(2, "Vfetk_fillArray:  can't write out charge distribution!\
      n");
02244             return 0;
02245             break;
02246
02247         case VDT_POT:
```

```
02248             u = am->u;
02249             u_d = am->ud;
02250             /* Copy in solution */
02251             Bvec_copy(vec, u);
02252             /* Add dirichlet condition */
02253             Bvec_axpy(vec, u_d, 1.0);
02254             break;
02255 
02256         case VDT_SMOL:
02257             for (i=0; i<Gem_numVV(gm); i++) {
02258                 vert = Gem_VV(gm, i);
02259                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02260                 chi = Vacc_molAcc(acc, coord, pbe->solventRadius);
02261                 Bvec_set(vec, i, chi);
02262             }
02263             break;
02264 
02265         case VDT_SSPL:
02266             for (i=0; i<Gem_numVV(gm); i++) {
02267                 vert = Gem_VV(gm, i);
02268                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02269                 chi = Vacc_splineAcc(acc, coord, pbeparm->swin, 0.0);
02270                 Bvec_set(vec, i, chi);
02271             }
02272             break;
02273 
02274         case VDT_VDW:
02275             for (i=0; i<Gem_numVV(gm); i++) {
02276                 vert = Gem_VV(gm, i);
02277                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02278                 chi = Vacc_vdwAcc(acc, coord);
02279                 Bvec_set(vec, i, chi);
02280             }
02281             break;
02282 
02283         case VDT_IVDW:
02284             for (i=0; i<Gem_numVV(gm); i++) {
02285                 vert = Gem_VV(gm, i);
02286                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02287                 chi = Vacc_ivdwAcc(acc, coord, pbe->maxIonRadius);
02288                 Bvec_set(vec, i, chi);
02289             }
02290             break;
02291 
02292         case VDT_LAP:
02293             Vnm_print(2, "Vfetk_fillArray:  can't write out Laplacian!\n");
02294             return 0;
02295             break;
02296 
02297         case VDT_EDENS:
02298             Vnm_print(2, "Vfetk_fillArray:  can't write out energy density!\n");
02299             return 0;
02300             break;
02301 
02302         case VDT_NDENS:
02303             u = am->u;
02304             u_d = am->ud;
```

```
02305             /* Copy in solution */
02306             Bvec_copy(vec, u);
02307             /* Add dirichlet condition */
02308             Bvec_axpy(vec, u_d, 1.0);
02309             /* Load up ions */
02310             ichop = 0;
02311             for (i=0; i<Gem_numVV(gm); i++) {
02312                 val = 0;
02313                 for (j=0; j<pbe->numIon; j++) {
02314                     q = pbe->ionQ[j];
02315                     conc = pbe->ionConc[j];
02316                     if (thee->type == PBE_NPBE || thee->type == PBE_SMPBE /* SMPB
      E Added */) {
02317                         val += (conc*Vcap_exp(-q*Bvec_val(vec, i), &ichop));
02318                     } else if (thee->type == PBE_LPBE) {
02319                         val += (conc * ( 1 - q*Bvec_val(vec, i)));
02320                     }
02321                 }
02322                 Bvec_set(vec, i, val);
02323             }
02324             break;
02325 
02326         case VDT_QDENS:
02327             u = am->u;
02328             u_d = am->ud;
02329             /* Copy in solution */
02330             Bvec_copy(vec, u);
02331             /* Add dirichlet condition */
02332             Bvec_axpy(vec, u_d, 1.0);
02333             /* Load up ions */
02334             ichop = 0;
02335             for (i=0; i<Gem_numVV(gm); i++) {
02336                 val = 0;
02337                 for (j=0; j<pbe->numIon; j++) {
02338                     q = pbe->ionQ[j];
02339                     conc = pbe->ionConc[j];
02340                     if (thee->type == PBE_NPBE || thee->type == PBE_SMPBE /* SMPB
      E Added */) {
02341                         val += (q*conc*Vcap_exp(-q*Bvec_val(vec, i), &ichop));
02342                     } else if (thee->type == PBE_LPBE) {
02343                         val += (q*conc*(1 - q*Bvec_val(vec, i)));
02344                     }
02345                 }
02346                 Bvec_set(vec, i, val);
02347             }
02348             break;
02349 
02350         case VDT_DIELX:
02351             Vnm_print(2, "Vfetk_fillArray:  can't write out x-shifted diel!\n");
02352             return 0;
02353             break;
02354 
02355         case VDT_DIELY:
02356             Vnm_print(2, "Vfetk_fillArray:  can't write out y-shifted diel!\n");
02357             return 0;
02358             break;
02359 
```

```
02360         case VDT_DIELZ:
02361             Vnm_print(2, "Vfetk_fillArray:  can't write out z-shifted diel!\n");
02362             return 0;
02363             break;
02364 
02365         case VDT_KAPPA:
02366             Vnm_print(2, "Vfetk_fillArray:  can't write out kappa!\n");
02367             return 0;
02368             break;
02369 
02370         default:
02371             Vnm_print(2, "Vfetk_fillArray:  invalid data type (%d)!\n", type);
02372             return 0;
02373             break;
02374     }
02375 
02376     return 1;
02377 }
02378 
02379 VPUBLIC int Vfetk_write(Vfetk *thee,  const char *iodev, const char *iofmt,
02380   const char *thost, const char *fname, Bvec *vec, Vdata_Format format) {
02381 
02382     int i, j, ichop;
02383     Aprx *aprx;
02384     Gem *gm;
02385     Vio *sock;
02386 
02387     VASSERT(thee != VNULL);
02388     aprx = thee->aprx;
02389     gm = thee->gm;
02390 
02391     sock = Vio_ctor(iodev,iofmt,thost,fname,"w");
02392     if (sock == VNULL) {
02393         Vnm_print(2, "Vfetk_write: Problem opening virtual socket %s\n",
02394           fname);
02395         return 0;
02396     }
02397     if (Vio_connect(sock, 0) < 0) {
02398         Vnm_print(2, "Vfetk_write: Problem connecting to virtual socket %s\n",
02399           fname);
02400         return 0;
02401     }
02402 
02403     /* Make sure vec has enough rows to accomodate the vertex data */
02404     if (Bvec_numRB(vec, 0) != Gem_numVV(gm)) {
02405         Vnm_print(2, "Vfetk_fillArray:  insufficient space in Bvec!\n");
02406         Vnm_print(2, "Vfetk_fillArray:  Have %d, need %d!\n", Bvec_numRB(vec, 0),

02407           Gem_numVV(gm));
02408         return 0;
02409     }
02410 
02411     switch (format) {
02412 
02413         case VDF_DX:
02414             Aprx_writeSOL(aprx, sock, vec, "DX");
02415             break;
```

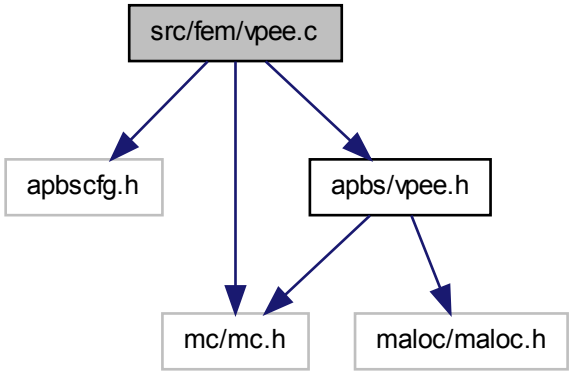
```
02416         case VDF_AVS:
02417             Aprx_writeSOL(aprx, sock, vec, "UCD");
02418             break;
02419         case VDF_UHBD:
02420             Vnm_print(2, "Vfetk_write:  UHBD format not supported!\n");
02421             return 0;
02422         default:
02423             Vnm_print(2, "Vfetk_write:  Invalid data format (%d)!\n", format);
02424             return 0;
02425     }
02426 
02427 
02428     Vio_connectFree(sock);
02429     Vio_dtor(&sock);
02430 
02431     return 1;
02432 }
02433 
02434 #endif
```

10.19 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 1552 2010-02-10 17:46:27Z yhuang01

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
* 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 Washington University in St. Louis 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 src/fem/vpee.c

```
00001
00050 #include "apbscfg.h"
00051
00052 #if defined(HAVE_MC_H)
00053 #if defined(HAVE_MCX_H)
00054
00055 #include "mc/mc.h"
00056 #include "apbs/vpee.h"
00057
00058 VPRIVATE int Vpee_userDefined(Vpee *thee, SS *sm);
00059 VPRIVATE int Vpee_ourSimp(Vpee *thee, SS *sm, int rcol);
00060 VEXTERNC double Aprx_estNonlinResid(Aprx *thee, SS *sm,
00061     Bvec *u, Bvec *ud, Bvec *f);
00062 VEXTERNC double Aprx_estLocalProblem(Aprx *thee, SS *sm,
00063     Bvec *u, Bvec *ud, Bvec *f);
00064 VEXTERNC double Aprx_estDualProblem(Aprx *thee, SS *sm,
00065     Bvec *u, Bvec *ud, Bvec *f);
00066
00067 /* ///////////////////////////////////////////////////////////////////////
00068 // Class Vpee: Non-inlineable methods
00070
00071 /* ///////////////////////////////////////////////////////////////////////
00072 // Routine:  Vpee_ctor
00073 //
00074 // Author:   Nathan Baker
00076 VPUBLIC Vpee* Vpee_ctor(Gem *gm, int localPartID, int killFlag, double
00077   killParam) {
00078
00079     Vpee *thee = VNULL;
00080
00081     /* Set up the structure */
00082     thee = Vmem_malloc(VNULL, 1, sizeof(Vpee) );
00083     VASSERT( thee != VNULL);
00084     VASSERT( Vpee_ctor2(thee, gm, localPartID, killFlag, killParam));
```

```
00085
00086     return thee;
00087 }
00088
00089 /* ///////////////////////////////////////////////////////////////////////////
00090 // Routine:  Vpee_ctor2
00091 //
00092 // Author:   Nathan Baker
00094 VPUBLIC int Vpee_ctor2(Vpee *thee, Gem *gm, int localPartID, int killFlag,
00095   double killParam) {
00096
00097     int ivert, nLocalVerts;
00098     SS *simp;
00099     VV *vert;
00100     double radius, dx, dy, dz;
00101
00102     VASSERT(thee != VNULL);
00103
00104     /* Sanity check on input values */
00105     if (killFlag == 0) {
00106         Vnm_print(0, "Vpee_ctor2: No error attenuation outside partition.\n");
00107     } else if (killFlag == 1) {
00108         Vnm_print(0, "Vpee_ctor2: Error outside local partition ignored.\n");
00109     } else if (killFlag == 2) {
00110         Vnm_print(0, "Vpee_ctor2: Error ignored outside sphere with radius %4.3f
      times the radius of the circumscribing sphere\n", killParam);
00111         if (killParam < 1.0) {
00112           Vnm_print(2, "Vpee_ctor2: Warning! Parameter killParam = %4.3 < 1.0!\n"
      ,
00113             killParam);
00114           Vnm_print(2, "Vpee_ctor2: This may result in non-optimal marking and re
      finement!\n");
00115         }
00116     } else if (killFlag == 3) {
00117         Vnm_print(0, "Vpee_ctor2: Error outside local partition and immediate nei
      ghbors ignored [NOT IMPLEMENTED].\n");
00118     } else {
00119         Vnm_print(2, "Vpee_ctor2: UNRECOGNIZED killFlag PARAMETER! BAILING!.\n");

00120         VASSERT(0);
00121     }
00122
00123     thee->gm = gm;
00124     thee->localPartID = localPartID;
00125     thee->killFlag = killFlag;
00126     thee->killParam = killParam;
00127     thee->mem = Vmem_ctor("APBS::VPEE");
00128
00129     /* Now, figure out the center of geometry for the local partition.  The
00130      * general plan is to loop through the vertices, loop through the
00131      * vertices' simplex lists and find the vertices with simplices containing
00132      * chart values we're interested in. */
00133     thee->localPartCenter[0] = 0.0;
00134     thee->localPartCenter[1] = 0.0;
00135     thee->localPartCenter[2] = 0.0;
00136     nLocalVerts = 0;
00137     for (ivert=0; ivert<Gem_numVV(thee->gm); ivert++) {
```

```
00138         vert = Gem_VV(thee->gm, ivert);
00139         simp = VV_firstSS(vert);
00140         VASSERT(simp != VNULL);
00141         while (simp != VNULL) {
00142             if (SS_chart(simp) == thee->localPartID) {
00143                 thee->localPartCenter[0] += VV_coord(vert, 0);
00144                 thee->localPartCenter[1] += VV_coord(vert, 1);
00145                 thee->localPartCenter[2] += VV_coord(vert, 2);
00146                 nLocalVerts++;
00147                 break;
00148             }
00149             simp = SS_link(simp, vert);
00150         }
00151     }
00152     VASSERT(nLocalVerts > 0);
00153     thee->localPartCenter[0] =
00154       thee->localPartCenter[0]/((double)(nLocalVerts));
00155     thee->localPartCenter[1] =
00156       thee->localPartCenter[1]/((double)(nLocalVerts));
00157     thee->localPartCenter[2] =
00158       thee->localPartCenter[2]/((double)(nLocalVerts));
00159     Vnm_print(0, "Vpee_ctor2: Part %d centered at (%4.3f, %4.3f, %4.3f)\n",
00160       thee->localPartID, thee->localPartCenter[0], thee->localPartCenter[1],
00161       thee->localPartCenter[2]);
00162
00163
00164     /* Now, figure out the radius of the sphere circumscribing the local
00165      * partition.  We need to keep track of vertices so we don't double count
00166      * them. */
00167     thee->localPartRadius = 0.0;
00168     for (ivert=0; ivert<Gem_numVV(thee->gm); ivert++) {
00169         vert = Gem_VV(thee->gm, ivert);
00170         simp = VV_firstSS(vert);
00171         VASSERT(simp != VNULL);
00172         while (simp != VNULL) {
00173             if (SS_chart(simp) == thee->localPartID) {
00174                 dx = thee->localPartCenter[0] - VV_coord(vert, 0);
00175                 dy = thee->localPartCenter[1] - VV_coord(vert, 1);
00176                 dz = thee->localPartCenter[2] - VV_coord(vert, 2);
00177                 radius = dx*dx + dy*dy + dz*dz;
00178                 if (radius > thee->localPartRadius) thee->localPartRadius =
00179                   radius;
00180                 break;
00181             }
00182             simp = SS_link(simp, vert);
00183         }
00184     }
00185     thee->localPartRadius = VSQRT(thee->localPartRadius);
00186     Vnm_print(0, "Vpee_ctor2: Part %d has circumscribing sphere of radius %4.3f\n
      ",
00187       thee->localPartID, thee->localPartRadius);
00188
00189     return 1;
00190 }
00191
00192 /* ///////////////////////////////////////////////////////////////////////////
00193 // Routine:  Vpee_dtor
```

```
00194 //
00195 // Author:   Nathan Baker
00197 VPUBLIC void Vpee_dtor(Vpee **thee) {
00198
00199     if ((*thee) != VNULL) {
00200         Vpee_dtor2(*thee);
00201         Vmem_free(VNULL, 1, sizeof(Vpee), (void **)thee);
00202         (*thee) = VNULL;
00203     }
00204
00205 }
00206
00207 /* ///////////////////////////////////////////////////////////////////////////
00208 // Routine:  Vpee_dtor2
00209 //
00210 // Author:   Nathan Baker
00212 VPUBLIC void Vpee_dtor2(Vpee *thee) { Vmem_dtor(&(thee->mem)); }
00213
00214 /* ///////////////////////////////////////////////////////////////////////////
00215 // Routine:  Vpee_markRefine
00216 //
00217 // Author:   Nathan Baker (and Michael Holst: the author of AM_markRefine, on
00218 //           which this is based)
00220 VPUBLIC int Vpee_markRefine(Vpee *thee, AM *am, int level, int akey, int rcol,
00221   double etol, int bkey) {
00222
00223     Aprx *aprx;
00224     int marked = 0;
00225     int markMe, i, smid, count, currentQ;
00226     double minError = 0.0;
00227     double maxError = 0.0;
00228     double errEst = 0.0;
00229     double mlevel, barrier;
00230     SS *sm;
00231
00232
00233     VASSERT(thee != VNULL);
00234
00235     /* Get the Aprx object from AM */
00236     aprx = am->aprx;
00237
00238     /* input check and some i/o */
00239     if ( ! ((-1 <= akey) && (akey <= 4)) ) {
00240         Vnm_print(0,"Vpee_markRefine: bad refine key; simplices marked = %d\n",
00241             marked);
00242         return marked;
00243     }
00244
00245     /* For uniform markings, we have no effect */
00246     if ((-1 <= akey) && (akey <= 0)) {
00247         marked = Gem_markRefine(thee->gm, akey, rcol);
00248         return marked;
00249     }
00250
00251     /* Informative I/O */
00252     if (akey == 2) {
00253         Vnm_print(0,"Vpee_estRefine: using Aprx_estNonlinResid().\n");
```

```
00254     } else if (akey == 3) {
00255         Vnm_print(0,"Vpee_estRefine: using Aprx_estLocalProblem().\n");
00256     } else if (akey == 4) {
00257         Vnm_print(0,"Vpee_estRefine: using Aprx_estDualProblem().\n");
00258     } else {
00259         Vnm_print(0,"Vpee_estRefine: bad key given; simplices marked = %d\n",
00260             marked);
00261         return marked;
00262     }
00263     if (thee->killFlag == 0) {
00264         Vnm_print(0, "Vpee_markRefine: No error attenuation -- simplices in all p
      artitions will be marked.\n");
00265     } else if (thee->killFlag == 1) {
00266         Vnm_print(0, "Vpee_markRefine: Maximum error attenuation -- only simplice
      s in local partition will be marked.\n");
00267     } else if (thee->killFlag == 2) {
00268         Vnm_print(0, "Vpee_markRefine: Spherical error attenutation -- simplices
      within a sphere of %4.3f times the size of the partition will be marked\n",
00269           thee->killParam);
00270     } else if (thee->killFlag == 2) {
00271         Vnm_print(0, "Vpee_markRefine: Neighbor-based error attenuation -- simpli
      ces in the local and neighboring partitions will be marked [NOT IMPLEMENTED]!\n")
      ;
00272         VASSERT(0);
00273     } else {
00274         Vnm_print(2,"Vpee_markRefine: bogus killFlag given; simplices marked = %d
      \n",
00275             marked);
00276         return marked;
00277     }
00278
00279     /* set the barrier type */
00280     mlevel = (etol*etol) / Gem_numSS(thee->gm);
00281     if (bkey == 0) {
00282         barrier = (etol*etol);
00283         Vnm_print(0,"Vpee_estRefine: forcing [err per S] < [TOL] = %g\n",
00284             barrier);
00285     } else if (bkey == 1) {
00286         barrier = mlevel;
00287         Vnm_print(0,"Vpee_estRefine: forcing [err per S] < [(TOL^2/numS)^{1/2}] =
       %g\n",
00288             VSQRT(barrier));
00289     } else {
00290         Vnm_print(0,"Vpee_estRefine: bad bkey given; simplices marked = %d\n",
00291             marked);
00292         return marked;
00293     }
00294
00295     /* timer */
00296     Vnm_tstart(30, "error estimation");
00297
00298     /* count = num generations to produce from marked simplices (minimally) */
00299     count = 1; /* must be >= 1 */
00300
00301     /* check the refinement Q for emptyness */
00302     currentQ = 0;
00303     if (Gem_numSQ(thee->gm,currentQ) > 0) {
```

```
00304         Vnm_print(0,"Vpee_markRefine: non-empty refinement Q%d....clearing..",
00305             currentQ);
00306         Gem_resetSQ(thee->gm,currentQ);
00307         Vnm_print(0,"..done.\n");
00308     }
00309     if (Gem_numSQ(thee->gm,!currentQ) > 0) {
00310         Vnm_print(0,"Vpee_markRefine: non-empty refinement Q%d....clearing..",
00311             !currentQ);
00312         Gem_resetSQ(thee->gm,!currentQ);
00313         Vnm_print(0,"..done.\n");
00314     }
00315     VASSERT( Gem_numSQ(thee->gm,currentQ)  == 0 );
00316     VASSERT( Gem_numSQ(thee->gm,!currentQ) == 0 );
00317
00318     /* clear everyone's refinement flags */
00319     Vnm_print(0,"Vpee_markRefine: clearing all simplex refinement flags..");
00320     for (i=0; i<Gem_numSS(thee->gm); i++) {
00321         if ( (i>0) && (i % VPRTKEY) == 0 ) Vnm_print(0,"[MS:%d]",i);
00322         sm = Gem_SS(thee->gm,i);
00323         SS_setRefineKey(sm,currentQ,0);
00324         SS_setRefineKey(sm,!currentQ,0);
00325         SS_setRefinementCount(sm,0);
00326     }
00327     Vnm_print(0,"..done.\n");
00328
00329     /* NON-ERROR-BASED METHODS */
00330     /* Simplex flag clearing */
00331     if (akey == -1) return marked;
00332     /* Uniform & user-defined refinement*/
00333     if ((akey == 0) || (akey == 1)) {
00334         smid = 0;
00335         while ( smid < Gem_numSS(thee->gm)) {
00336             /* Get the simplex and find out if it's markable */
00337             sm = Gem_SS(thee->gm,smid);
00338             markMe = Vpee_ourSimp(thee, sm, rcol);
00339             if (markMe) {
00340                 if (akey == 0) {
00341                     marked++;
00342                     Gem_appendSQ(thee->gm,currentQ, sm);
00343                     SS_setRefineKey(sm,currentQ,1);
00344                     SS_setRefinementCount(sm,count);
00345                 } else if (Vpee_userDefined(thee, sm)) {
00346                     marked++;
00347                     Gem_appendSQ(thee->gm,currentQ, sm);
00348                     SS_setRefineKey(sm,currentQ,1);
00349                     SS_setRefinementCount(sm,count);
00350                 }
00351             }
00352             smid++;
00353         }
00354     }
00355
00356     /* ERROR-BASED METHODS */
00357     /* gerror = global error accumulation */
00358     aprx->gerror = 0.;
00359
00360     /* traverse the simplices and process the error estimates */
```

```
00361     Vnm_print(0,"Vpee_markRefine: estimating error..");
00362     smid = 0;
00363     while ( smid < Gem_numSS(thee->gm)) {
00364
00365         /* Get the simplex and find out if it's markable */
00366         sm = Gem_SS(thee->gm,smid);
00367         markMe = Vpee_ourSimp(thee, sm, rcol);
00368
00369         if ( (smid>0) && (smid % VPRTKEY) == 0 ) Vnm_print(0,"[MS:%d]",smid);
00370
00371         /* Produce an error estimate for this element if it is in the set */
00372         if (markMe) {
00373             if (akey == 2) {
00374                 errEst = Aprx_estNonlinResid(aprx, sm, am->u,am->ud,am->f);
00375             } else if (akey == 3) {
00376                 errEst = Aprx_estLocalProblem(aprx, sm, am->u,am->ud,am->f);
00377             } else if (akey == 4) {
00378                 errEst = Aprx_estDualProblem(aprx, sm, am->u,am->ud,am->f);
00379             }
00380             VASSERT( errEst >= 0. );
00381
00382             /* if error estimate above tol, mark element for refinement */
00383             if ( errEst > barrier ) {
00384                 marked++;
00385                 Gem_appendSQ(thee->gm,currentQ, sm); /*add to refinement Q*/
00386                 SS_setRefineKey(sm,currentQ,1);      /* note now on refine Q */
00387                 SS_setRefinementCount(sm,count);     /* refine X many times? */
00388             }
00389
00390             /* keep track of min/max errors over the mesh */
00391             minError = VMIN2( VSQRT(VABS(errEst)), minError );
00392             maxError = VMAX2( VSQRT(VABS(errEst)), maxError );
00393
00394             /* store the estimate */
00395             Bvec_set( aprx->wev, smid, errEst );
00396
00397             /* accumlate into global error (errEst is SQUAREd already) */
00398             aprx->gerror += errEst;
00399
00400         /* otherwise store a zero for the estimate */
00401         } else {
00402             Bvec_set( aprx->wev, smid, 0. );
00403         }
00404
00405         smid++;
00406     }
00407
00408     /* do some i/o */
00409     Vnm_print(0,"..done.  [marked=<%d/%d>]\n",marked,Gem_numSS(thee->gm));
00410     Vnm_print(0,"Vpee_estRefine: TOL=<%g>  Global_Error=<%g>\n",
00411         etol, aprx->gerror);
00412     Vnm_print(0,"Vpee_estRefine: (TOL^2/numS)^{1/2}=<%g>  Max_Ele_Error=<%g>\n",
00413         VSQRT(mlevel),maxError);
00414     Vnm_tstop(30, "error estimation");
00415
00416     /* check for making the error tolerance */
00417     if ((bkey == 1) && (aprx->gerror <= etol)) {
```

```
00418         Vnm_print(0,
00419             "Vpee_estRefine: ****************************************\n");
00420         Vnm_print(0,
00421             "Vpee_estRefine: Global Error criterion met; setting marked=0.\n");
00422         Vnm_print(0,
00423             "Vpee_estRefine: ****************************************\n");
00424         marked = 0;
00425     }
00426 
00427 
00428     /* return */
00429     return marked;
00430 
00431 }
00432 
00433 /* ///////////////////////////////////////////////////////////////////////////
00434 // Routine:  Vpee_numSS
00435 //
00436 // Author:   Nathan Baker
00438 VPUBLIC int Vpee_numSS(Vpee *thee) {
00439     int num = 0;
00440     int isimp;
00441 
00442     for (isimp=0; isimp<Gem_numSS(thee->gm); isimp++) {
00443         if (SS_chart(Gem_SS(thee->gm, isimp)) == thee->localPartID) num++;
00444     }
00445 
00446     return num;
00447 }
00448 
00449 /* ///////////////////////////////////////////////////////////////////////////
00450 // Routine:  Vpee_userDefined
00451 //
00452 // Purpose:  Reduce code bloat by wrapping up the common steps for getting the
00453 //           user-defined error estimate
00454 //
00455 // Author:   Nathan Baker
00457 VPRIVATE int Vpee_userDefined(Vpee *thee, SS *sm) {
00458 
00459     int ivert, icoord, chart[4], fType[4], vType[4];
00460     double vx[4][3];
00461 
00462     for (ivert=0; ivert<Gem_dimVV(thee->gm); ivert++) {
00463         fType[ivert] = SS_faceType(sm,ivert);
00464         vType[ivert] = VV_type(SS_vertex(sm,ivert) );
00465         chart[ivert] = VV_chart(SS_vertex(sm,ivert) );
00466         for (icoord=0; icoord<Gem_dimII(thee->gm); icoord++) {
00467             vx[ivert][icoord] = VV_coord(SS_vertex(sm,ivert), icoord );
00468         }
00469     }
00470     return thee->gm->pde->markSimplex(Gem_dim(thee->gm), Gem_dimII(thee->gm),
00471              SS_type(sm), fType, vType, chart, vx, sm);
00472 }
00473 
00474 /* ///////////////////////////////////////////////////////////////////////////
00475 // Routine:  Vpee_ourSimp
00476 //
```

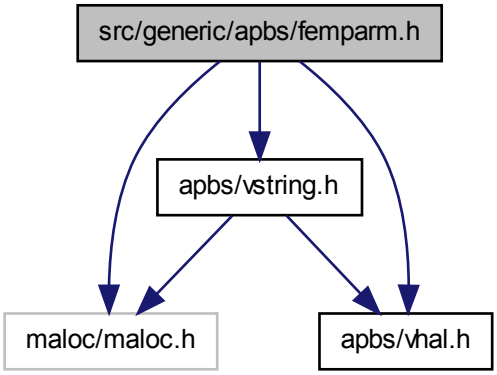
```
00477 // Purpose:  Reduce code bloat by wrapping up the common steps for determining
00478 //           whether the given simplex can be marked (i.e., belongs to our
00479 //           partition or overlap region)
00480 //
00481 // Returns:  1 if could be marked, 0 otherwise
00482 //
00483 // Author:   Nathan Baker
00485 VPRIVATE int Vpee_ourSimp(Vpee *thee, SS *sm, int rcol) {
00486
00487     int ivert;
00488     double dist, dx, dy, dz;
00489
00490     if (thee->killFlag == 0) return 1;
00491     else if (thee->killFlag == 1) {
00492         if ((SS_chart(sm) == rcol) || (rcol < 0)) return 1;
00493     } else if (thee->killFlag == 2) {
00494         if (rcol < 0) return 1;
00495         else {
00496             /* We can only do distance-based searches on the local partition */
00497             VASSERT(rcol == thee->localPartID);
00498             /* Find the closest distance between this simplex and the
00499              * center of the local partition and check it against
00500              * (thee->localPartRadius*thee->killParam) */
00501             dist = 0;
00502             for (ivert=0; ivert<SS_dimVV(sm); ivert++) {
00503                 dx = VV_coord(SS_vertex(sm, ivert), 0) -
00504                   thee->localPartCenter[0];
00505                 dy = VV_coord(SS_vertex(sm, ivert), 1) -
00506                   thee->localPartCenter[1];
00507                 dz = VV_coord(SS_vertex(sm, ivert), 2) -
00508                   thee->localPartCenter[2];
00509                 dist = VSQRT((dx*dx + dy*dy + dz*dz));
00510             }
00511             if (dist < thee->localPartRadius*thee->killParam) return 1;
00512         }
00513     } else if (thee->killFlag == 3) VASSERT(0);
00514     else VASSERT(0);
00515
00516     return 0;
00517
00518 }
00519
00520 #endif
00521 #endif
```

10.21 src/generic/apbs/femparm.h File Reference

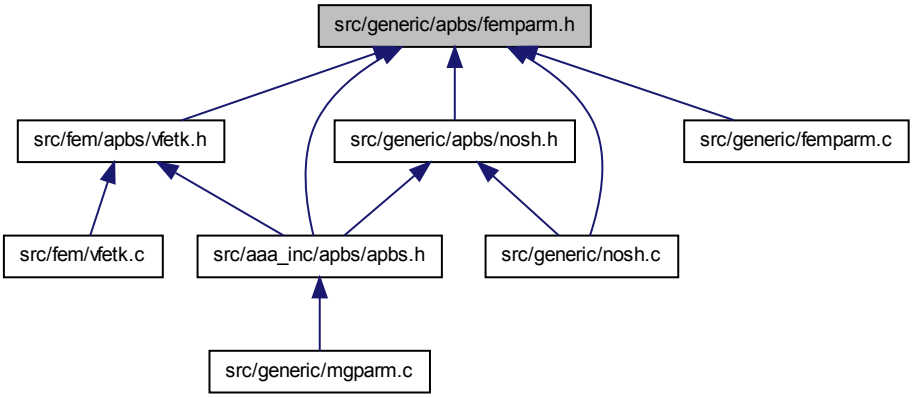
Contains declarations for class APOLparm.

```
#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 FEparm_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 refinment error estimate tolerance key.

- enum eFEMparm_EstType {
  FRT_UNIF = 0, FRT_GEOM = 1, FRT_RESI = 2, FRT_DUAL = 3,
  FRT_LOCA = 4 }

  Adaptive refinment 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 1564 2010-03-07 14:04:14Z sobolevnrm

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
```

Version

Id:

femparm.h 1552 2010-02-10 17:46:27Z yhuang01

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
* 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 Washington University in St. Louis 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 src/generic/apbs/femparm.h

```
00001
00055 #ifndef _FEMPARM_H_
00056 #define _FEMPARM_H_
00057
00058 /* Generic header files */
00059 #include "maloc/maloc.h"
00060 #include "apbs/vhal.h"
00061 #include "apbs/vstring.h"
00062
00068 enum eFEMparm_EtolType {
00069     FET_SIMP=0,
00070     FET_GLOB=1,
00071     FET_FRAC=2
00072 };
00073
00079 typedef enum eFEMparm_EtolType FEMparm_EtolType;
00080
00087 enum eFEMparm_EstType {
00088     FRT_UNIF=0,
00089     FRT_GEOM=1,
00090     FRT_RESI=2,
00091     FRT_DUAL=3,
00093     FRT_LOCA=4
```

```
00094 };
00095
00100 typedef enum eFEMparm_EstType FEMparm_EstType;
00101
00106 enum eFEMparm_CalcType {
00107     FCT_MANUAL,
00108  FCT_NONE
00109 };
00110
00115 typedef enum eFEMparm_CalcType FEMparm_CalcType;
00116
00122 struct sFEMparm {
00123
00124     int parsed;
00127     FEMparm_CalcType type;
00128     int settype;
00129     double glen[3];
00130     int setglen;
00131     double etol;
00133     int setetol;
00134     FEMparm_EtolType ekey;
00136     int setekey;
00137     FEMparm_EstType akeyPRE;
00140     int setakeyPRE;
00141     FEMparm_EstType akeySOLVE;
00143     int setakeySOLVE;
00144     int targetNum;
00148     int settargetNum;
00149     double targetRes;
00153     int settargetRes;
00154     int maxsolve;
00155     int setmaxsolve;
00156     int maxvert;
00158     int setmaxvert;
00159  int pkey;
00162  int useMesh;
00163  int meshID;
00165 };
00166
00171 typedef struct sFEMparm FEMparm;
00172
00173 /* ////////////////////////////////////////////////////////////////////////////
00174 // Class NOsh: Non-inlineable methods (nosh.c)
00176
00183 VEXTERNC FEMparm* FEMparm_ctor(FEMparm_CalcType type);
00184
00192 VEXTERNC int FEMparm_ctor2(FEMparm *thee, FEMparm_CalcType type);
00193
00199 VEXTERNC void FEMparm_dtor(FEMparm **thee);
00200
00206 VEXTERNC void FEMparm_dtor2(FEMparm *thee);
00207
00215 VEXTERNC int FEMparm_check(FEMparm *thee);
00216
00223 VEXTERNC void FEMparm_copy(FEMparm *thee, FEMparm *source);
00224
00235 VEXTERNC Vrc_Codes FEMparm_parseToken(FEMparm *thee, char tok[VMAX_BUFSIZE],
```

```
00236   Vio *sock);
00237
00238 #endif
00239
```

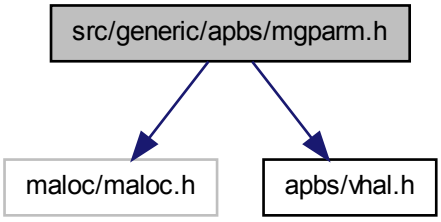
10.23 src/generic/apbs/mgparm.h File Reference

Contains declarations for class MGparm.

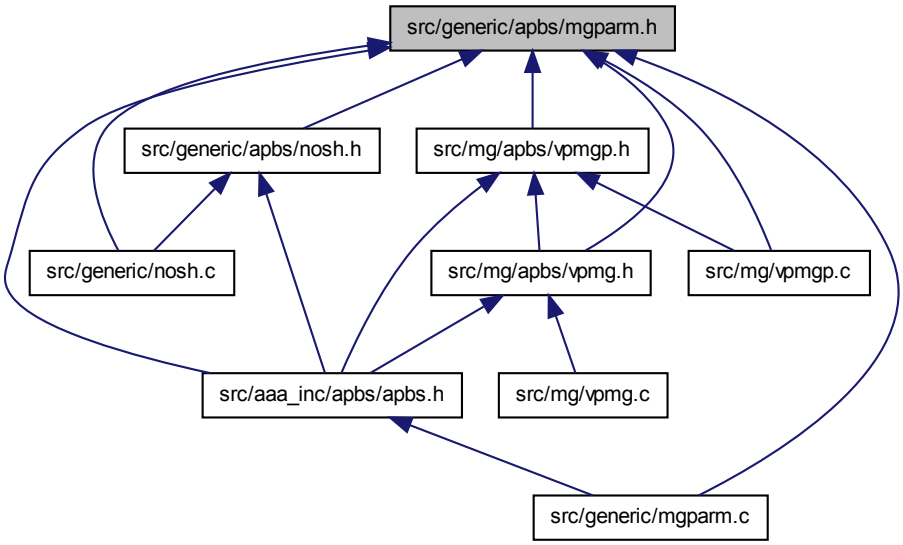
```
#include "maloc/maloc.h"
```

```
#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 1552 2010-02-10 17:46:27Z yhuang01

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
* 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 Washington University in St. Louis 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 src/generic/apbs/mgparm.h

```
00001
00056 #ifndef _MGPARM_H_
00057 #define _MGPARM_H_
00058
00059 #include "maloc/maloc.h"
```

```
00060 #include "apbs/vhal.h"
00061
00066 enum eMGparm_CalcType {
00067     MCT_MANUAL=0,
00068     MCT_AUTO=1,
00069     MCT_PARALLEL=2,
00070     MCT_DUMMY=3,
00071  MCT_NONE=4
00072 };
00073
00078 typedef enum eMGparm_CalcType MGparm_CalcType;
00079
00084 enum eMGparm_CentMeth {
00085     MCM_POINT=0,
00086     MCM_MOLECULE=1,
00087  MCM_FOCUS=2
00088 };
00089
00094 typedef enum eMGparm_CentMeth MGparm_CentMeth;
00103 struct sMGparm {
00104
00105     MGparm_CalcType type;
00106     int parsed;
00108     /* *** GENERIC PARAMETERS *** */
00109     int dime[3];
00110     int setdime;
00111     Vchrg_Meth chgm;
00112     int setchgm;
00113     Vchrg_Src  chgs;
00116     /* *** TYPE 0 PARAMETERS (SEQUENTIAL MANUAL) *** */
00117     int nlev;
00119     int setnlev;
00120     double etol;
00121     int setetol;
00122     double grid[3];
00123     int setgrid;
00124     double glen[3];
00125     int setglen;
00126     MGparm_CentMeth cmeth;
00127     double center[3];
00135     int centmol;
00138     int setgcent;
00140     /* ******** TYPE 1 & 2 PARAMETERS (SEQUENTIAL & PARALLEL AUTO-FOCUS) *** */
00141     double cglen[3];
00142     int setcglen;
00143     double fglen[3];
00144     int setfglen;
00145     MGparm_CentMeth ccmeth;
00146     double ccenter[3];
00147     int ccentmol;
00150     int setcgcent;
00151     MGparm_CentMeth fcmeth;
00152     double fcenter[3];
00153     int fcentmol;
00156     int setfgcent;
00159     /* ********* TYPE 2 PARAMETERS (PARALLEL AUTO-FOCUS) ******** */
00160     double partDisjCenter[3];
```

```
00162     double partDisjLength[3];
00164     int partDisjOwnSide[6];
00167     int pdime[3];
00168     int setpdime;
00169     int proc_rank;
00170     int setrank;
00171     int proc_size;
00172     int setsize;
00173     double ofrac;
00174     int setofrac;
00175     int async;
00176     int setasync;
00178  int nonlintype;
00179  int setnonlintype;
00181  int method;
00182  int setmethod;
00184  int useAqua;
00185  int setUseAqua;
00186 };
00187
00192 typedef struct sMGparm MGparm;
00193
00200 VEXTERNC int MGparm_getNx(MGparm *thee);
00201
00208 VEXTERNC int MGparm_getNy(MGparm *thee);
00209
00216 VEXTERNC int MGparm_getNz(MGparm *thee);
00217
00224 VEXTERNC double MGparm_getHx(MGparm *thee);
00225
00232 VEXTERNC double MGparm_getHy(MGparm *thee);
00233
00240 VEXTERNC double MGparm_getHz(MGparm *thee);
00241
00248 VEXTERNC void MGparm_setCenterX(MGparm *thee, double x);
00249
00256 VEXTERNC void MGparm_setCenterY(MGparm *thee, double y);
00257
00264 VEXTERNC void MGparm_setCenterZ(MGparm *thee, double z);
00265
00272 VEXTERNC double MGparm_getCenterX(MGparm *thee);
00273
00280 VEXTERNC double MGparm_getCenterY(MGparm *thee);
00281
00288 VEXTERNC double MGparm_getCenterZ(MGparm *thee);
00289
00296 VEXTERNC MGparm*  MGparm_ctor(MGparm_CalcType type);
00297
00305 VEXTERNC Vrc_Codes      MGparm_ctor2(MGparm *thee, MGparm_CalcType type);
00306
00312 VEXTERNC void     MGparm_dtor(MGparm **thee);
00313
00319 VEXTERNC void     MGparm_dtor2(MGparm *thee);
00320
00327 VEXTERNC Vrc_Codes      MGparm_check(MGparm *thee);
00328
00335 VEXTERNC void     MGparm_copy(MGparm *thee, MGparm *parm);
```

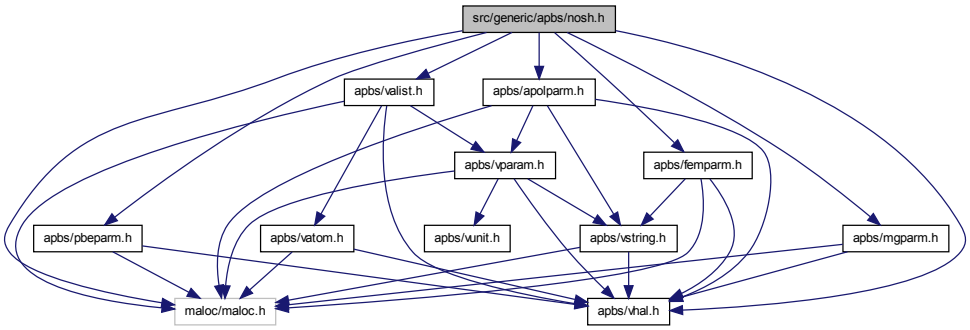
```
00336
00346 VEXTERNC Vrc_Codes      MGparm_parseToken(MGparm *thee, char tok[VMAX_BUFSIZE],
00347                     Vio *sock);
00348
00349 #endif
00350
```

10.25 src/generic/apbs/nosh.h File Reference

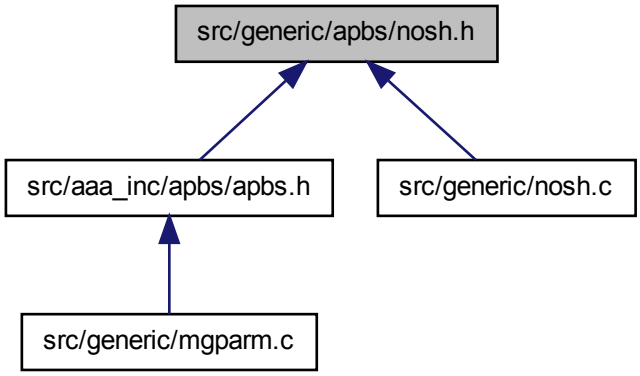
Contains declarations for class NOsh.

```
#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 1573 2010-03-18 17:12:50Z sdg0919

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
* 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 Washington University in St. Louis 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 src/generic/apbs/nosh.h

```
00001
00054 #ifndef _NOSH_H_
00055 #define _NOSH_H_
00056
00057 /* Generic headers */
00058 #include "maloc/maloc.h"
00059 #include "apbs/vhal.h"
00060
00061 /* Headers specific to this file */
00062 #include "apbs/pbeparm.h"
00063 #include "apbs/mgparm.h"
00064 #include "apbs/femparm.h"
00065 #include "apbs/apolparm.h"
00066 #include "apbs/valist.h"
00067
00070 #define NOSH_MAXMOL 20
00071
00074 #define NOSH_MAXCALC 20
00075
00078 #define NOSH_MAXPRINT 20
00079
00082 #define NOSH_MAXPOP 20
00083
00088 enum eNOsh_MolFormat {
00089     NMF_PQR=0,
00090     NMF_PDB=1,
00091     NMF_XML=2
00092 };
00093
00098 typedef enum eNOsh_MolFormat NOsh_MolFormat;
00099
00104 enum eNOsh_CalcType {
00105     NCT_MG=0,
00106     NCT_FEM=1,
```

```
00107  NCT_APOL=2
00108 };
00109
00114 typedef enum eNOsh_CalcType NOsh_CalcType;
00115
00120 enum eNOsh_ParmFormat {
00121     NPF_FLAT=0,
00122     NPF_XML=1
00123 };
00124
00129 typedef enum eNOsh_ParmFormat NOsh_ParmFormat;
00130
00135 enum eNOsh_PrintType {
00136     NPT_ENERGY=0,
00137     NPT_FORCE=1,
00138  NPT_ELECENERGY,
00139     NPT_ELECFORCE,
00140  NPT_APOLENERGY,
00141     NPT_APOLFORCE
00142 };
00143
00148 typedef enum eNOsh_PrintType NOsh_PrintType;
00149
00155 struct sNOsh_calc {
00156  MGparm *mgparm;
00157  FEMparm *femparm;
00158  PBEparm *pbeparm;
00159  APOLparm *apolparm;
00160  NOsh_CalcType calctype;
00161 };
00162
00167 typedef struct sNOsh_calc NOsh_calc;
00168
00174 struct sNOsh {
00175
00176  NOsh_calc *calc[NOSH_MAXCALC];
00179     int ncalc;
00181  NOsh_calc *elec[NOSH_MAXCALC];
00184     int nelec;
00187  NOsh_calc *apol[NOSH_MAXCALC];
00190     int napol;
00193     int ispara;
00194     int proc_rank;
00195     int proc_size;
00196     int bogus;
00200     int elec2calc[NOSH_MAXCALC];
00208  int apol2calc[NOSH_MAXCALC];
00210     int nmol;
00211     char molpath[NOSH_MAXMOL][VMAX_ARGLEN];
00212     NOsh_MolFormat molfmt[NOSH_MAXMOL];
00213  Valist *alist[NOSH_MAXMOL];
00215     int gotparm;
00216     char parmpath[VMAX_ARGLEN];
00217     NOsh_ParmFormat parmfmt;
00218     int ndiel;
00219     char dielXpath[NOSH_MAXMOL][VMAX_ARGLEN];
00221     char dielYpath[NOSH_MAXMOL][VMAX_ARGLEN];
```

```
00223     char dielZpath[NOSH_MAXMOL][VMAX_ARGLEN];
00225     Vdata_Format dielfmt[NOSH_MAXMOL];
00226     int nkappa;
00227     char kappapath[NOSH_MAXMOL][VMAX_ARGLEN];
00228     Vdata_Format kappafmt[NOSH_MAXMOL];
00229  int npot;
00230     char potpath[NOSH_MAXMOL][VMAX_ARGLEN];
00231     Vdata_Format potfmt[NOSH_MAXMOL];
00232     int ncharge;
00233     char chargepath[NOSH_MAXMOL][VMAX_ARGLEN];
00234     Vdata_Format chargefmt[NOSH_MAXMOL];
00235  int nmesh;
00236  char meshpath[NOSH_MAXMOL][VMAX_ARGLEN];
00237     Vdata_Format meshfmt[NOSH_MAXMOL];
00238     int nprint;
00239     NOsh_PrintType printwhat[NOSH_MAXPRINT];
00241     int printnarg[NOSH_MAXPRINT];
00242     int printcalc[NOSH_MAXPRINT][NOSH_MAXPOP];
00243     int printop[NOSH_MAXPRINT][NOSH_MAXPOP];
00245     int parsed;
00246     char elecname[NOSH_MAXCALC][VMAX_ARGLEN];
00248  char apolname[NOSH_MAXCALC][VMAX_ARGLEN];
00250 };
00251
00256 typedef struct sNOsh NOsh;
00257
00258 /* ///////////////////////////////////////////////////////////////////////////
00259    // Class NOsh: Inlineable methods (mcsh.c)
00261 #if !defined(VINLINE_NOSH)
00269 VEXTERNC char* NOsh_getMolpath(NOsh *thee, int imol);
00270
00278 VEXTERNC char* NOsh_getDielXpath(NOsh *thee, int imap);
00279
00287 VEXTERNC char* NOsh_getDielYpath(NOsh *thee, int imap);
00288
00296 VEXTERNC char* NOsh_getDielZpath(NOsh *thee, int imap);
00297
00305 VEXTERNC char* NOsh_getKappapath(NOsh *thee, int imap);
00306
00314 VEXTERNC char* NOsh_getPotpath(NOsh *thee, int imap);
00315
00323 VEXTERNC char* NOsh_getChargepath(NOsh *thee, int imap);
00324
00332 VEXTERNC NOsh_calc* NOsh_getCalc(NOsh *thee, int icalc);
00333
00341 VEXTERNC int NOsh_getDielfmt(NOsh *thee, int imap);
00342
00350 VEXTERNC int NOsh_getKappafmt(NOsh *thee, int imap);
00351
00359 VEXTERNC int NOsh_getPotfmt(NOsh *thee, int imap);
00360
00368 VEXTERNC int NOsh_getChargefmt(NOsh *thee, int imap);
00369
00370 #else
00371
00372 #   define NOsh_getMolpath(thee, imol) ((thee)->molpath[(imol)])
00373 #   define NOsh_getDielXpath(thee, imol) ((thee)->dielXpath[(imol)])
```

```
00374 #   define NOsh_getDielYpath(thee, imol) ((thee)->dielYpath[(imol)])
00375 #   define NOsh_getDielZpath(thee, imol) ((thee)->dielZpath[(imol)])
00376 #   define NOsh_getKappapath(thee, imol) ((thee)->kappapath[(imol)])
00377 #   define NOsh_getPotpath(thee, imol) ((thee)->potpath[(imol)])
00378 #   define NOsh_getChargepath(thee, imol) ((thee)->chargepath[(imol)])
00379 #   define NOsh_getCalc(thee, icalc) ((thee)->calc[(icalc)])
00380 #   define NOsh_getDielfmt(thee, imap) ((thee)->dielfmt[(imap)])
00381 #   define NOsh_getKappafmt(thee, imap) ((thee)->kappafmt[(imap)])
00382 #   define NOsh_getPotfmt(thee, imap) ((thee)->potfmt[(imap)])
00383 #   define NOsh_getChargefmt(thee, imap) ((thee)->chargefmt[(imap)])
00384
00385 #endif
00386
00387
00388 /* ///////////////////////////////////////////////////////////////////////
00389    // Class NOsh: Non-inlineable methods (mcsh.c)
00391
00399 VEXTERNC NOsh_PrintType NOsh_printWhat(NOsh *thee, int iprint);
00400
00410 VEXTERNC char* NOsh_elecname(NOsh *thee, int ielec);
00411
00419 VEXTERNC int NOsh_elec2calc(NOsh *thee, int icalc);
00420
00428 VEXTERNC int NOsh_apol2calc(NOsh *thee, int icalc);
00429
00437 VEXTERNC int NOsh_printNarg(NOsh *thee, int iprint);
00438
00447 VEXTERNC int NOsh_printOp(NOsh *thee, int iprint, int iarg);
00448
00459 VEXTERNC int NOsh_printCalc(NOsh *thee, int iprint, int iarg);
00460
00470 VEXTERNC NOsh* NOsh_ctor(int rank, int size);
00471
00478 VEXTERNC NOsh_calc* NOsh_calc_ctor(
00479          NOsh_CalcType calcType
00480          );
00481
00488 VEXTERNC int NOsh_calc_copy(
00489        NOsh_calc *thee,
00490        NOsh_calc *source
00491        );
00492
00498 VEXTERNC void  NOsh_calc_dtor(NOsh_calc **thee);
00499
00510 VEXTERNC int   NOsh_ctor2(NOsh *thee, int rank, int size);
00511
00517 VEXTERNC void  NOsh_dtor(NOsh **thee);
00518
00524 VEXTERNC void  NOsh_dtor2(NOsh *thee);
00525
00534 VEXTERNC int   NOsh_parseInput(NOsh *thee, Vio *sock);
00535
00545 VEXTERNC int   NOsh_parseInputFile(NOsh *thee, char *filename);
00546
00556 VEXTERNC int NOsh_setupElecCalc(
00557        NOsh *thee,
00558        Valist *alist[NOSH_MAXMOL]
```

```
00559         );
00560
00570 VEXTERNC int NOsh_setupApolCalc(
00571          NOsh *thee,
00572          Valist *alist[NOSH_MAXMOL]
00573          );
00574
00575 #endif
00576
```

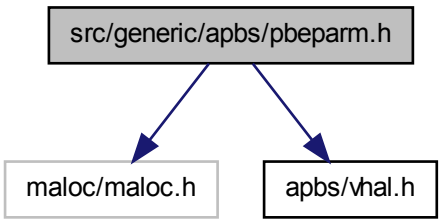
10.27 src/generic/apbs/pbeparm.h File Reference

Contains declarations for class PBEparm.

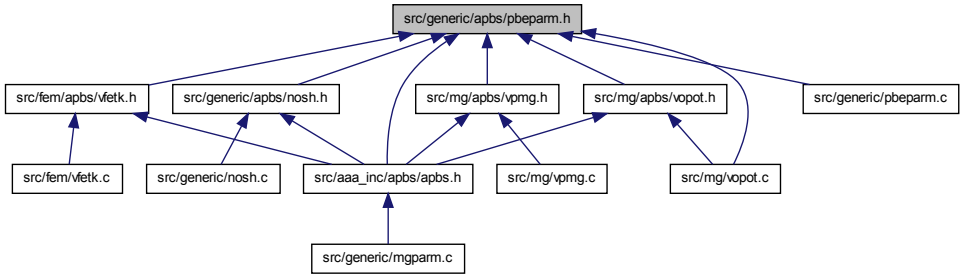
```
#include "maloc/maloc.h"
```

```
#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 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.

10.27.1 Detailed Description

Contains declarations for class PBEparm.

Version

Id:

pbeparm.h 1573 2010-03-18 17:12:50Z sdg0919

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
* 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 Washington University in St. Louis 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 src/generic/apbs/pbeparm.h

```
00001
00054 #ifndef _PBEPARM_H_
00055 #define _PBEPARM_H_
00056
00057 /* Generic headers */
00058 #include "maloc/maloc.h"
00059
00060 /* Headers specific to this file */
00061 #include "apbs/vhal.h"
00062
00066 #define PBEPARM_MAXWRITE 20
00067
00072 enum ePBEparm_calcEnergy {
00073     PCE_NO=0,
00074     PCE_TOTAL=1,
00075     PCE_COMPS=2
00076 };
00077
00082 typedef enum ePBEparm_calcEnergy PBEparm_calcEnergy;
00083
00088 enum ePBEparm_calcForce {
00089     PCF_NO=0,
00090     PCF_TOTAL=1,
00091     PCF_COMPS=2
00092 };
00093
00098 typedef enum ePBEparm_calcForce PBEparm_calcForce;
00099
00108 struct sPBEparm {
00109
00110     int molid;
00111     int setmolid;
00112     int useDielMap;
00114     int dielMapID;
00115     int useKappaMap;
00117     int kappaMapID;
00118     int usePotMap;
00120     int potMapID;
00122     int useChargeMap;
00124     int chargeMapID;
00125     Vhal_PBEType pbetype;
00126     int setpbetype;
00127     Vbcfl bcfl;
00128     int setbcfl;
00129     int nion;
00130     int setnion;
00131     double ionq[MAXION];
00132     double ionc[MAXION];
00133     double ionr[MAXION];
00134     int setion[MAXION];
```

```
00135     double pdie;
00136     int setpdie;
00137     double sdens;
00138     int setsdens;
00139     double sdie;
00140     int setsdie;
00141     Vsurf_Meth srfm;
00142     int setsrfm;
00143     double srad;
00144     int setsrad;
00145     double swin;
00146     int setswin;
00147     double temp;
00148     int settemp;
00150  double smsize;
00151  int setsmsize;
00153  double smvolume;
00154  int setsmvolume;
00156     PBEparm_calcEnergy calcenergy;
00157     int setcalcenergy;
00158     PBEparm_calcForce calcforce;
00159     int setcalcforce;
00161  /*-----------------------------------------------------------------------*/
00162  /* Added by Michael Grabe                                                */
00163  /*-----------------------------------------------------------------------*/
00164
00165     double zmem;
00166     int setzmem;
00167     double Lmem;
00168     int setLmem;
00169     double mdie;
00170     int setmdie;
00171     double memv;
00172     int setmemv;
00174  /*-----------------------------------------------------------------------*/
00175
00176     int numwrite;
00177     char writestem[PBEPARM_MAXWRITE][VMAX_ARGLEN];
00179     Vdata_Type writetype[PBEPARM_MAXWRITE];
00180     Vdata_Format writefmt[PBEPARM_MAXWRITE];
00182     int writemat;
00185     int setwritemat;
00186     char writematstem[VMAX_ARGLEN];
00187     int writematflag;
00192     int parsed;
00194 };
00195
00200 typedef struct sPBEparm PBEparm;
00201
00202 /* ///////////////////////////////////////////////////////////////////////////
00203 // Class NOsh: Non-inlineable methods (mcsh.c)
00205
00211 VEXTERNC double PBEparm_getIonCharge(
00212         PBEparm *thee,
00213         int iion
00214         );
00215
```

```
00221 VEXTERNC double PBEparm_getIonConc(
00222         PBEparm *thee,
00223         int iion
00224         );
00225
00231 VEXTERNC double PBEparm_getIonRadius(
00232         PBEparm *thee,
00233         int iion
00234         );
00235
00236
00242 VEXTERNC PBEparm* PBEparm_ctor();
00243
00249 VEXTERNC int PBEparm_ctor2(
00250         PBEparm *thee
00251         );
00252
00257 VEXTERNC void PBEparm_dtor(
00258         PBEparm **thee
00259         );
00260
00265 VEXTERNC void PBEparm_dtor2(
00266         PBEparm *thee
00267         );
00268
00274 VEXTERNC int PBEparm_check(
00275         PBEparm *thee
00276         );
00277
00282 VEXTERNC void PBEparm_copy(
00283         PBEparm *thee,
00284         PBEparm *parm
00285         );
00286
00293 VEXTERNC int PBEparm_parseToken(
00294         PBEparm *thee,
00295         char tok[VMAX_BUFSIZE],
00296         Vio *sock
00297         );
00298
00299
00300 #endif
00301
```

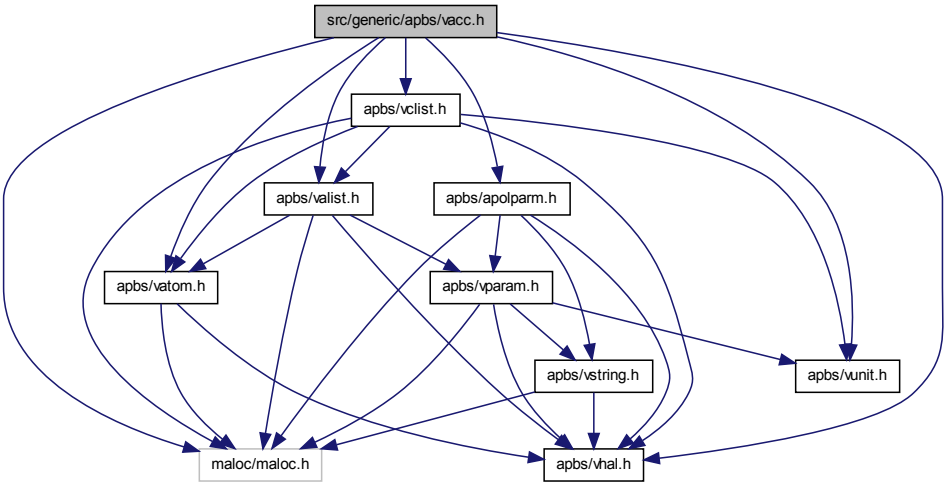
10.29 src/generic/apbs/vacc.h File Reference

Contains declarations for class Vacc.

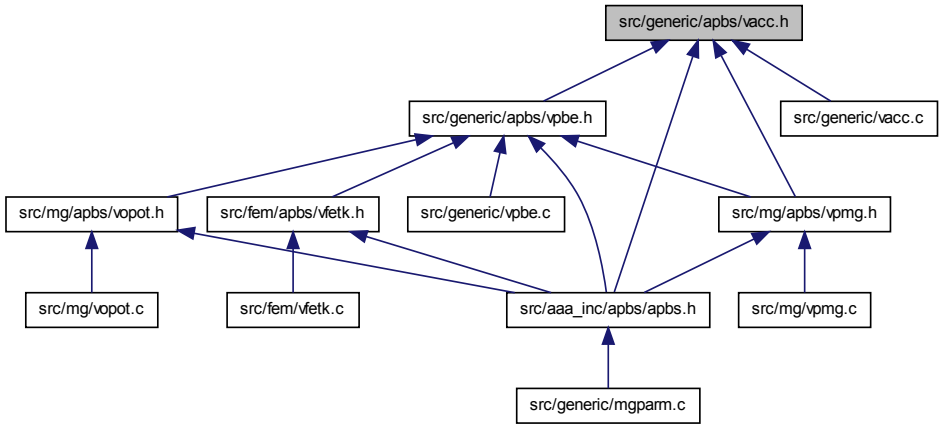
```
#include "maloc/maloc.h"

#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 derivatve of solvent accessible volume.

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

  Get the derivatve 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 1605 2010-09-13 15:12:09Z yhuang01

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
* 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 Washington University in St. Louis 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 src/generic/apbs/vacc.h

```
00001
00054 #ifndef _VACC_H_
00055 #define _VACC_H_
00056
00057 /* Generic headers */
00058 #include "maloc/maloc.h"
00059 #include "apbs/vhal.h"
00060
00061 /* Headers specific to this file */
00062 #include "apbs/valist.h"
00063 #include "apbs/vclist.h"
00064 #include "apbs/vatom.h"
00065 #include "apbs/vunit.h"
00066 #include "apbs/apolparm.h"
00067
00073 struct sVaccSurf {
00074     Vmem *mem;
00075     double *xpts;
00076     double *ypts;
00077     double *zpts;
00078     char *bpts;
00080     double area;
00081     int npts;
00082     double probe_radius;
00084 };
00085
00090 typedef struct sVaccSurf VaccSurf;
00091
00097 struct sVacc {
00098
00099   Vmem *mem;
00100   Valist *alist;
00101   Vclist *clist;
00102   int *atomFlags;
00105   VaccSurf *refSphere;
00106   VaccSurf **surf;
00109   Vset acc;
00111   double surf_density;
00114 };
00115
00120 typedef struct sVacc Vacc;
00121
00122 #if !defined(VINLINE_VACC)
00123
```

```
00129     VEXTERNC unsigned long int Vacc_memChk(
00130             Vacc *thee
00131             );
00132
00133 #else /* if defined(VINLINE_VACC) */
00134
00135 #   define Vacc_memChk(thee) (Vmem_bytes((thee)->mem))
00136
00137 #endif /* if !defined(VINLINE_VACC) */
00138
00146 VEXTERNC VaccSurf* VaccSurf_ctor(
00147         Vmem *mem,
00148         double probe_radius,
00149         int nsphere
00150         );
00151
00159 VEXTERNC int VaccSurf_ctor2(
00160         VaccSurf *thee,
00161         Vmem *mem,
00162         double probe_radius,
00163         int nsphere
00164         );
00165
00171 VEXTERNC void VaccSurf_dtor(
00172         VaccSurf **thee
00173         );
00174
00180 VEXTERNC void VaccSurf_dtor2(
00181         VaccSurf *thee
00182         );
00183
00198 VEXTERNC VaccSurf* VaccSurf_refSphere(
00199         Vmem *mem,
00200         int npts
00201         );
00202
00210 VEXTERNC VaccSurf* Vacc_atomSurf(
00211         Vacc *thee,
00212         Vatom *atom,
00213         VaccSurf *ref,
00215         double probe_radius
00216         );
00217
00218
00223 VEXTERNC Vacc* Vacc_ctor(
00224         Valist *alist,
00225         Vclist *clist,
00227         double surf_density
00229         );
00230
00235 VEXTERNC int Vacc_ctor2(
00236         Vacc *thee,
00237         Valist *alist,
00238         Vclist *clist,
00240         double surf_density
00242         );
00243
```

```
00248 VEXTERNC void Vacc_dtor(
00249         Vacc **thee
00250         );
00251
00256 VEXTERNC void Vacc_dtor2(
00257         Vacc *thee
00258         );
00259
00270 VEXTERNC double Vacc_vdwAcc(
00271         Vacc *thee,
00272         double center[VAPBS_DIM]
00273         );
00274
00286 VEXTERNC double Vacc_ivdwAcc(
00287         Vacc *thee,
00288         double center[VAPBS_DIM],
00289         double radius
00290         );
00291
00306 VEXTERNC double Vacc_molAcc(
00307         Vacc *thee,
00308         double center[VAPBS_DIM],
00309         double radius
00310         );
00311
00330 VEXTERNC double Vacc_fastMolAcc(
00331         Vacc *thee,
00332         double center[VAPBS_DIM],
00333         double radius
00334         );
00335
00347 VEXTERNC double Vacc_splineAcc(
00348         Vacc *thee,
00349         double center[VAPBS_DIM],
00350         double win,
00351         double infrad
00352         );
00353
00359 VEXTERNC void Vacc_splineAccGrad(
00360         Vacc *thee,
00361         double center[VAPBS_DIM],
00362         double win,
00363         double infrad,
00364         double *grad
00365         );
00366
00378 VEXTERNC double Vacc_splineAccAtom(
00379         Vacc *thee,
00380         double center[VAPBS_DIM],
00381         double win,
00382         double infrad,
00383         Vatom *atom
00384         );
00385
00396 VEXTERNC void Vacc_splineAccGradAtomUnnorm(
00397         Vacc *thee,
00398         double center[VAPBS_DIM],
```

```
00399         double win,
00400         double infrad,
00401         Vatom *atom,
00402         double *force
00403         );
00404
00416 VEXTERNC void Vacc_splineAccGradAtomNorm(
00417         Vacc *thee,
00418         double center[VAPBS_DIM],
00419         double win,
00420         double infrad,
00421         Vatom *atom,
00422         double *force
00423         );
00424
00432 VEXTERNC void Vacc_splineAccGradAtomNorm4(
00433         Vacc *thee,
00434         double center[VAPBS_DIM],
00435         double win,
00436         double infrad,
00437         Vatom *atom,
00438         double *force
00439         );
00440
00448 VEXTERNC void Vacc_splineAccGradAtomNorm3(
00449   Vacc *thee,
00450   double center[VAPBS_DIM],
00451   double win,
00452   double infrad,
00453   Vatom *atom,
00454   double *force
00455   );
00456
00457
00467 VEXTERNC double Vacc_SASA(
00468         Vacc *thee,
00469         double radius
00470         );
00471
00479 VEXTERNC double Vacc_totalSASA(
00480         Vacc *thee,
00481         double radius
00482         );
00483
00491 VEXTERNC double Vacc_atomSASA(
00492         Vacc *thee,
00493         double radius,
00494         Vatom *atom
00495         );
00496
00503 VEXTERNC VaccSurf* Vacc_atomSASPoints(
00504         Vacc *thee,
00505         double radius,
00506         Vatom *atom
00507         );
00508
00514 VEXTERNC void Vacc_atomdSAV(
```

```
00515         Vacc *thee,
00516         double radius,
00517         Vatom *atom,
00518         double *dSA
00519         );
00520
00526 VEXTERNC void Vacc_atomdSASA(
00527         Vacc *thee,
00528         double dpos,
00529         double radius,
00530         Vatom *atom,
00531         double *dSA
00532         );
00533
00539 VEXTERNC void Vacc_totalAtomdSASA(
00540          Vacc *thee,
00541          double dpos,
00542          double radius,
00543          Vatom *atom,
00544          double *dSA
00545          );
00546
00552 VEXTERNC void Vacc_totalAtomdSAV(
00553           Vacc *thee,
00554           double dpos,
00555           double radius,
00556           Vatom *atom,
00557           double *dSA,
00558           Vclist *clist
00559           );
00560
00568 VEXTERNC double Vacc_totalSAV(
00569  Vacc *thee,
00570  Vclist *clist,
00571  APOLparm *apolparm,
00573  double radius
00574  );
00575
00582 VEXTERNC int Vacc_wcaEnergy(
00583           Vacc *thee,
00584          APOLparm *apolparm,
00585          Valist *alist,
00586          Vclist *clist
00587          );
00594 VEXTERNC int Vacc_wcaForceAtom(Vacc *thee,
00595           APOLparm *apolparm,
00596           Vclist *clist,
00597           Vatom *atom,
00598           double *force
00599           );
00600
00606 VEXTERNC int Vacc_wcaEnergyAtom(
00607  Vacc *thee,
00608  APOLparm *apolparm,
00609  Valist *alist,
00610  Vclist *clist,
00611  int iatom,
```

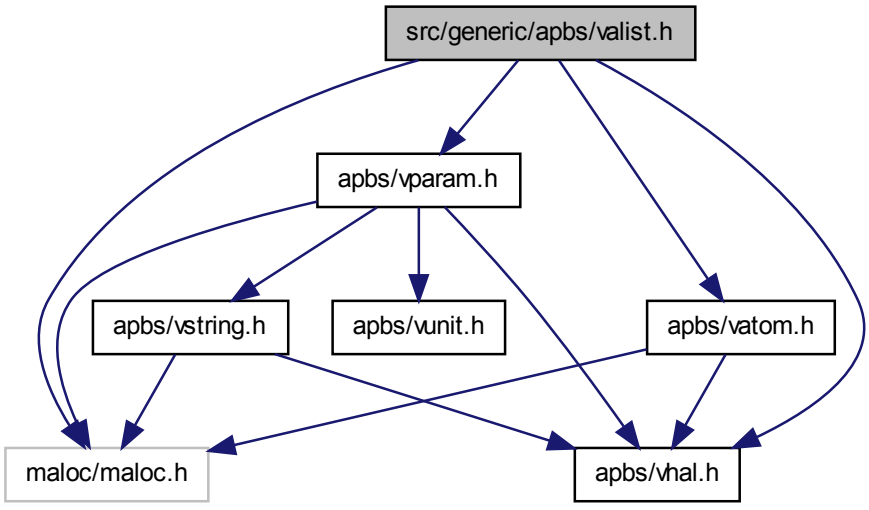
```
00612  double *value
00613  );
00614
00615 #endif    /* ifndef _VACC_H_ */
```

10.31 src/generic/apbs/valist.h File Reference

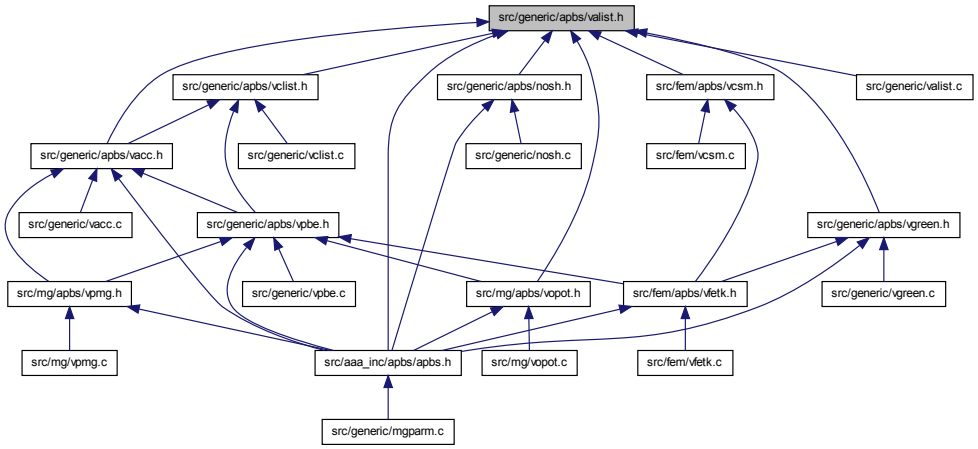
Contains declarations for class Valist.

```
#include "maloc/maloc.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 1552 2010-02-10 17:46:27Z yhuang01

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
* 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 Washington University in St. Louis 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 src/generic/apbs/valist.h

```
00001
00054 #ifndef _VALIST_H_
00055 #define _VALIST_H_
00056
00057 /* Generic headers */
00058 #include "maloc/maloc.h"
00059 #include "apbs/vhal.h"
00060
00061 /* Headers specific to this file */
00062 #include "apbs/vatom.h"
00063 #include "apbs/vparam.h"
00064
00070 struct sValist {
00071
00072   int number;
00073   double center[3];
00074   double mincrd[3];
00075   double maxcrd[3];
00076   double maxrad;
00077   double charge;
00078   Vatom *atoms;
00079   Vmem *vmem;
00081 };
00082
00087 typedef struct sValist Valist;
00088
00089 #if !defined(VINLINE_VATOM)
00090
00097 VEXTERNC Vatom* Valist_getAtomList(
00098         Valist *thee
00099         );
00100
00106 VEXTERNC double Valist_getCenterX(
00107         Valist *thee
00108         );
00109
00115 VEXTERNC double Valist_getCenterY(
00116         Valist *thee
00117         );
00118
00124 VEXTERNC double Valist_getCenterZ(
00125         Valist *thee
00126         );
00127
00133 VEXTERNC int Valist_getNumberAtoms(
00134         Valist *thee
00135         );
00136
00142 VEXTERNC Vatom* Valist_getAtom(
00143         Valist *thee,
```

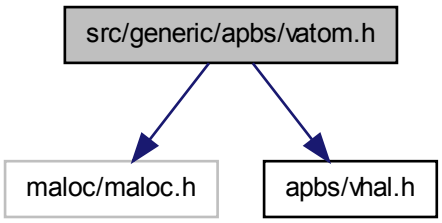
```
00144         int i
00145         );
00146
00152 VEXTERNC unsigned long int Valist_memChk(
00153         Valist *thee
00154         );
00155
00156 #else /* if defined(VINLINE_VATOM) */
00157 #   define Valist_getAtomList(thee) ((thee)->atoms)
00158 #   define Valist_getNumberAtoms(thee) ((thee)->number)
00159 #   define Valist_getAtom(thee, i) (&((thee)->atoms[i]))
00160 #   define Valist_memChk(thee) (Vmem_bytes((thee)->vmem))
00161 #   define Valist_getCenterX(thee) ((thee)->center[0])
00162 #   define Valist_getCenterY(thee) ((thee)->center[1])
00163 #   define Valist_getCenterZ(thee) ((thee)->center[2])
00164 #endif /* if !defined(VINLINE_VATOM) */
00165
00171 VEXTERNC Valist* Valist_ctor();
00172
00178 VEXTERNC Vrc_Codes Valist_ctor2(
00179         Valist *thee
00180         );
00181
00186 VEXTERNC void Valist_dtor(
00187         Valist **thee
00188         );
00189
00194 VEXTERNC void Valist_dtor2(
00195         Valist *thee
00196         );
00197
00209 VEXTERNC Vrc_Codes Valist_readPQR(
00210         Valist *thee,
00211   Vparam *param,
00212         Vio *sock
00213         );
00214
00224 VEXTERNC Vrc_Codes Valist_readPDB(
00225         Valist *thee,
00226         Vparam *param,
00227         Vio *sock
00228         );
00229
00239 VEXTERNC Vrc_Codes Valist_readXML(
00240         Valist *thee,
00241   Vparam *param,
00242         Vio *sock
00243         );
00244
00251 VEXTERNC Vrc_Codes Valist_getStatistics(Valist *thee);
00252
00253
00254 #endif /* ifndef _VALIST_H_ */
```

10.33 src/generic/apbs/vatom.h File Reference

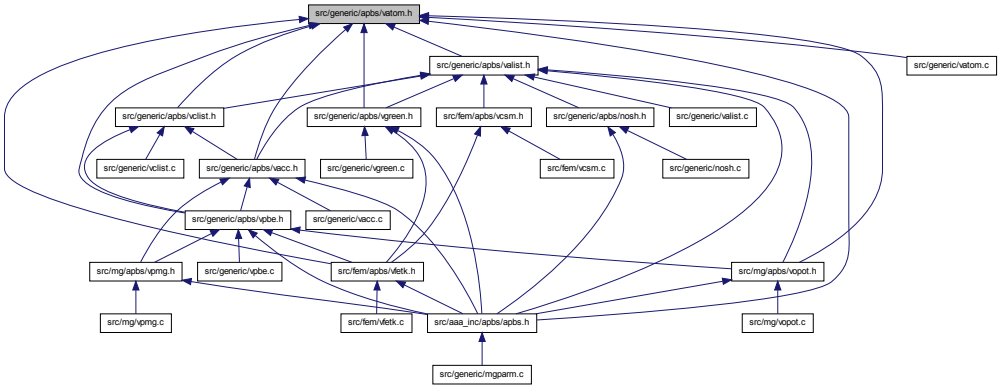
Contains declarations for class Vatom.

```
#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 1565 2010-03-07 16:06:27Z sobolevnrm

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
* 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 Washington University in St. Louis 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 src/generic/apbs/vatom.h

```
00001
00054 #ifndef _VATOM_H_
00055 #define _VATOM_H_
00056
00057 #include "maloc/maloc.h"
00058 #include "apbs/vhal.h"
00059
00066 #define VMAX_RECLEN     64
00067
00073 struct sVatom {
00074
00075     double position[3];
00076     double radius;
00077     double charge;
00078     double partID;
00080     double epsilon;
00082  int id;
00086  char resName[VMAX_RECLEN];
00087     char atomName[VMAX_RECLEN];
00089 #if defined(WITH_TINKER)
00090
00091     double dipole[3];
00092     double quadrupole[9];
00093     double inducedDipole[3];
00094     double nlInducedDipole[3];
00096 #endif /* if defined(WITH_TINKER) */
00097 };
00098
00103 typedef struct sVatom Vatom;
00104
00105 #if !defined(VINLINE_VATOM)
00106
00113     VEXTERNC double* Vatom_getPosition(Vatom *thee);
00114
```

```
00121     VEXTERNC void    Vatom_setRadius(Vatom *thee, double radius);
00122
00129     VEXTERNC double  Vatom_getRadius(Vatom *thee);
00130
00138     VEXTERNC void    Vatom_setPartID(Vatom *thee, int partID);
00139
00147     VEXTERNC double     Vatom_getPartID(Vatom *thee);
00148
00155     VEXTERNC void Vatom_setAtomID(Vatom *thee, int id);
00156
00163     VEXTERNC double Vatom_getAtomID(Vatom *thee);
00164
00171     VEXTERNC void    Vatom_setCharge(Vatom *thee, double charge);
00172
00179     VEXTERNC double  Vatom_getCharge(Vatom *thee);
00180
00187  VEXTERNC void    Vatom_setEpsilon(Vatom *thee, double epsilon);
00188
00195  VEXTERNC double  Vatom_getEpsilon(Vatom *thee);
00196
00204     VEXTERNC unsigned long int Vatom_memChk(Vatom *thee);
00205
00206 #else /* if defined(VINLINE_VATOM) */
00207 #   define Vatom_getPosition(thee) ((thee)->position)
00208 #   define Vatom_setRadius(thee, tRadius) ((thee)->radius = (tRadius))
00209 #   define Vatom_getRadius(thee) ((thee)->radius)
00210 #   define Vatom_setPartID(thee, tpartID) ((thee)->partID = (double)(tpartID))
00211 #   define Vatom_getPartID(thee) ((thee)->partID)
00212 #   define Vatom_setAtomID(thee, tatomID) ((thee)->id = (tatomID))
00213 #   define Vatom_getAtomID(thee) ((thee)->id)
00214 #   define Vatom_setCharge(thee, tCharge) ((thee)->charge = (tCharge))
00215 #   define Vatom_getCharge(thee) ((thee)->charge)
00216 #   define Vatom_setEpsilon(thee, tEpsilon) ((thee)->epsilon = (tEpsilon))
00217 #   define Vatom_getEpsilon(thee) ((thee)->epsilon)
00218 #   define Vatom_memChk(thee) (sizeof(Vatom))
00219 #endif /* if !defined(VINLINE_VATOM) */
00220
00221 /* ///////////////////////////////////////////////////////////////////////////
00222 // Class Vatom: Non-Inlineable methods (vatom.c)
00224
00231 VEXTERNC void    Vatom_setResName(Vatom *thee, char resName[VMAX_RECLEN]);
00232
00237 VEXTERNC void    Vatom_setAtomName(
00238   Vatom *thee,
00239   char atomName[VMAX_RECLEN]
00240   );
00241
00248 VEXTERNC void    Vatom_getResName(Vatom *thee, char resName[VMAX_RECLEN]);
00249
00254 VEXTERNC void   Vatom_getAtomName(
00255   Vatom *thee,
00256   char atomName[VMAX_RECLEN]
00257   );
00258
00264 VEXTERNC Vatom* Vatom_ctor();
00265
00272 VEXTERNC int     Vatom_ctor2(Vatom *thee);
```

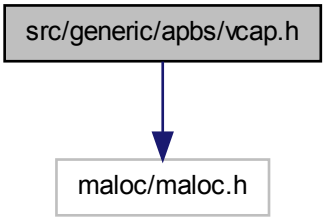
```
00273
00279 VEXTERNC void    Vatom_dtor(Vatom **thee);
00280
00286 VEXTERNC void    Vatom_dtor2(Vatom *thee);
00287
00294 VEXTERNC void   Vatom_setPosition(Vatom *thee, double position[3]);
00295
00303 VEXTERNC void Vatom_copyTo(Vatom *thee, Vatom *dest);
00304
00312 VEXTERNC void Vatom_copyFrom(Vatom *thee, Vatom *src);
00313
00314 #if defined(WITH_TINKER)
00315
00322 VEXTERNC void   Vatom_setInducedDipole(Vatom *thee,
00323                                        double inducedDipole[3]);
00324
00331 VEXTERNC void   Vatom_setNLInducedDipole(Vatom *thee,
00332                                        double nlInducedDipole[3]);
00333
00340 VEXTERNC void   Vatom_setDipole(Vatom *thee, double dipole[3]);
00341
00348 VEXTERNC void   Vatom_setQuadrupole(Vatom *thee, double quadrupole[9]);
00349
00355 VEXTERNC double*  Vatom_getDipole(Vatom *thee);
00356
00362 VEXTERNC double*  Vatom_getQuadrupole(Vatom *thee);
00363
00369 VEXTERNC double*  Vatom_getInducedDipole(Vatom *thee);
00370
00376 VEXTERNC double*  Vatom_getNLInducedDipole(Vatom *thee);
00377 #endif /* if defined(WITH_TINKER) */
00378
00379 #endif /* ifndef _VATOM_H_ */
```

10.35 src/generic/apbs/vcap.h File Reference

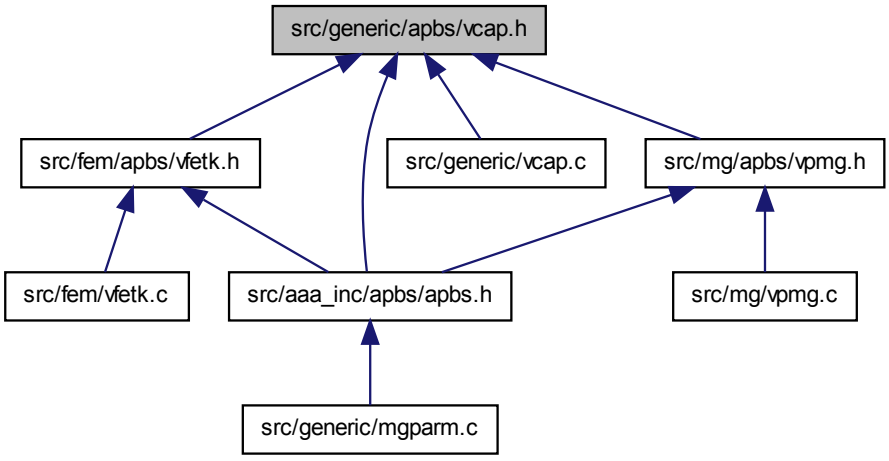
Contains declarations for class Vcap.

```
#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 1552 2010-02-10 17:46:27Z yhuang01

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
* 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 Washington University in St. Louis 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 src/generic/apbs/vcap.h

```
00001
00056 #ifndef _VCAP_H_
00057 #define _VCAP_H_
00058
00062 #define EXPMAX  85.00
00063
00067 #define EXPMIN -85.00
00068
00069 #include "maloc/maloc.h"
00070
00089 VEXTERNC double Vcap_exp(
00090         double x,
00091         int *ichop
00092         );
00093
00094
00113 VEXTERNC double Vcap_sinh(
00114         double x,
00115         int *ichop
00116         );
00117
00136 VEXTERNC double Vcap_cosh(
00137         double x,
```

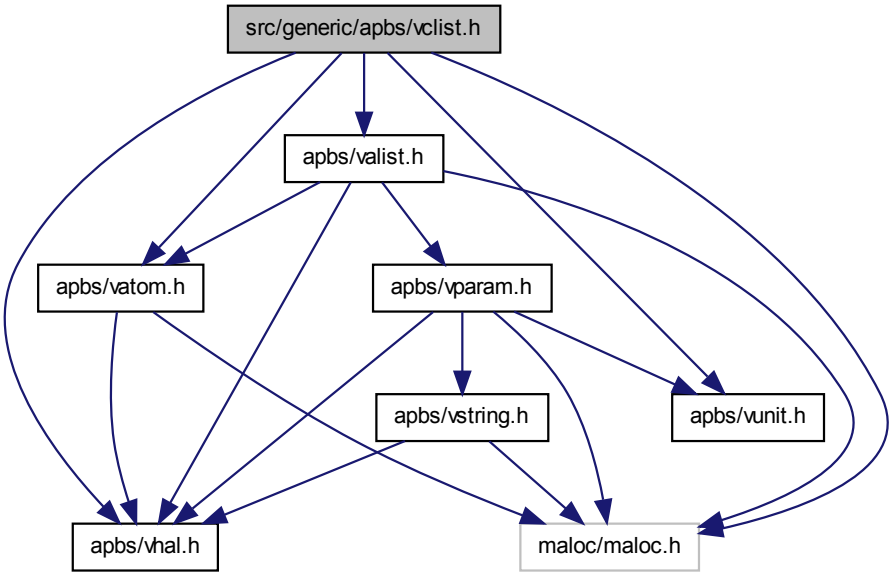
```
00138         int *ichop
00139         );
00140
00141 #endif    /* ifndef _VCAP_H_ */
```

10.37 src/generic/apbs/vclist.h File Reference

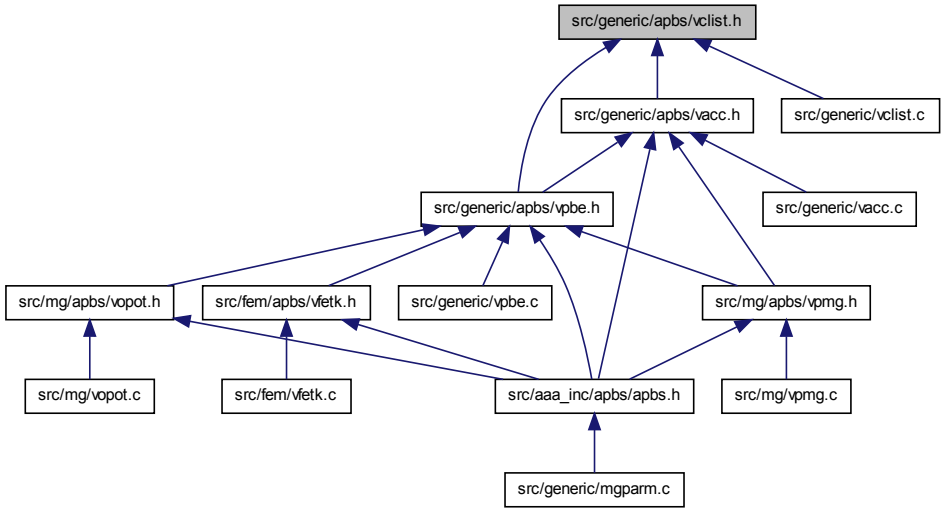
Contains declarations for class Vclist.

```
#include "maloc/maloc.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 1565 2010-03-07 16:06:27Z sobolevnrm

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
* 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 Washington University in St. Louis 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 src/generic/apbs/vclist.h

```
00001
00054 #ifndef _VCLIST_H_
00055 #define _VCLIST_H_
00056
00057 /* Generic headers */
00058 #include "maloc/maloc.h"
00059 #include "apbs/vhal.h"
00060
00061 /* Headers specific to this file */
00062 #include "apbs/valist.h"
00063 #include "apbs/vatom.h"
00064 #include "apbs/vunit.h"
00065
00071 enum eVclist_DomainMode {
00072     CLIST_AUTO_DOMAIN,
00074     CLIST_MANUAL_DOMAIN
00076 };
00077
00083 typedef enum eVclist_DomainMode Vclist_DomainMode;
00084
00090 struct sVclistCell {
00091     Vatom **atoms;
00092     int natoms;
00093 };
00094
00099 typedef struct sVclistCell VclistCell;
00100
00106 struct sVclist {
00107
00108   Vmem *vmem;
00109   Valist *alist;
00110   Vclist_DomainMode mode;
00111   int npts[VAPBS_DIM];
00112   int n;
00113   double max_radius;
00114   VclistCell *cells;
00115   double lower_corner[VAPBS_DIM];
00116   double upper_corner[VAPBS_DIM];
00117   double spacs[VAPBS_DIM];
00119 };
00120
```

```
00125 typedef struct sVclist Vclist;
00126
00127 #if !defined(VINLINE_VCLIST)
00128
00134     VEXTERNC unsigned long int Vclist_memChk(
00135             Vclist *thee
00136             );
00137
00145     VEXTERNC double Vclist_maxRadius(
00146             Vclist *thee
00147             );
00148
00149 #else /* if defined(VINLINE_VCLIST) */
00150
00151 #   define Vclist_memChk(thee) (Vmem_bytes((thee)->vmem))
00152 #   define Vclist_maxRadius(thee) ((thee)->max_radius)
00153
00154 #endif /* if !defined(VINLINE_VCLIST) */
00155
00156 /* ///////////////////////////////////////////////////////////////////////////
00157 // Class Vclist: Non-Inlineable methods (vclist.c)
00159
00164 VEXTERNC Vclist* Vclist_ctor(
00165         Valist *alist,
00166         double max_radius,
00167         int npts[VAPBS_DIM],
00169         Vclist_DomainMode mode,
00170         double lower_corner[VAPBS_DIM],
00173         double upper_corner[VAPBS_DIM]
00176         );
00177
00182 VEXTERNC Vrc_Codes Vclist_ctor2(
00183         Vclist *thee,
00184         Valist *alist,
00185         double max_radius,
00186         int npts[VAPBS_DIM],
00188         Vclist_DomainMode mode,
00189         double lower_corner[VAPBS_DIM],
00192         double upper_corner[VAPBS_DIM]
00195         );
00196
00201 VEXTERNC void Vclist_dtor(
00202         Vclist **thee
00203         );
00204
00209 VEXTERNC void Vclist_dtor2(
00210         Vclist *thee
00211         );
00212
00220 VEXTERNC VclistCell* Vclist_getCell(
00221         Vclist *thee,
00222         double position[VAPBS_DIM]
00223         );
00224
00231 VEXTERNC VclistCell* VclistCell_ctor(
00232         int natoms
00233         );
```

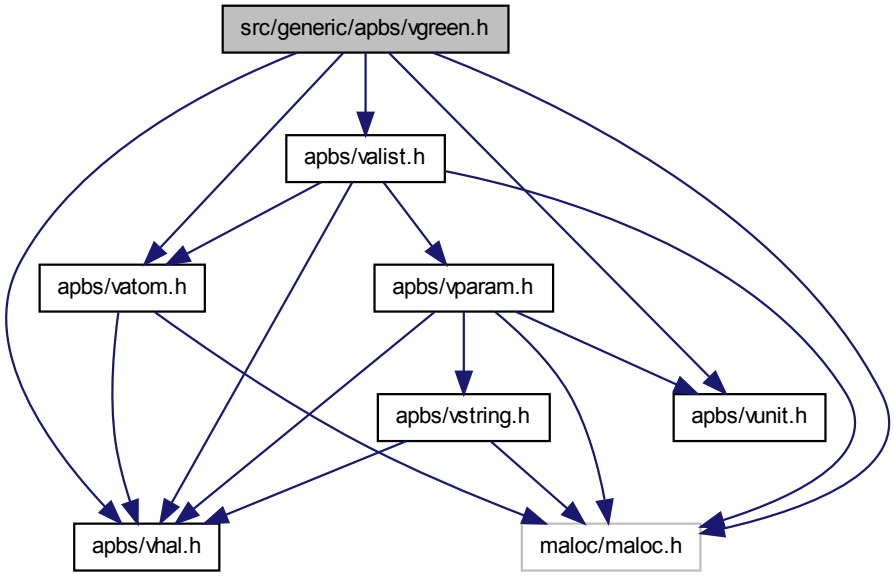
```
00234
00241 VEXTERNC Vrc_Codes VclistCell_ctor2(
00242         VclistCell *thee,
00243         int natoms
00244         );
00245
00250 VEXTERNC void VclistCell_dtor(
00251         VclistCell **thee
00252         );
00253
00258 VEXTERNC void VclistCell_dtor2(
00259         VclistCell *thee
00260         );
00261
00262 #endif    /* ifndef _VCLIST_H_ */
```

10.39 src/generic/apbs/vgreen.h File Reference

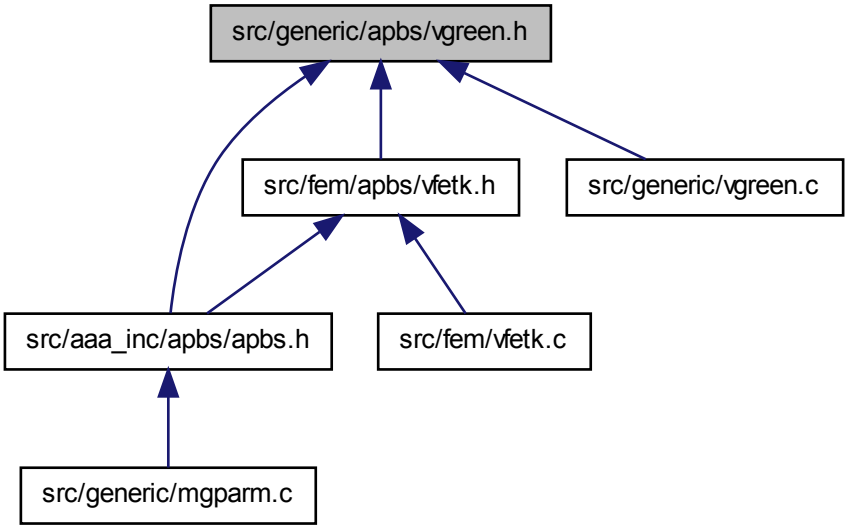
Contains declarations for class Vgreen.

```
#include "maloc/maloc.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_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)

10.39.1 Detailed Description

Contains declarations for class Vgreen.

Version

Id:

vgreen.h 1552 2010-02-10 17:46:27Z yhuang01

Author

Nathan A. Baker

Definition in file vgreen.h.

10.40 src/generic/apbs/vgreen.h

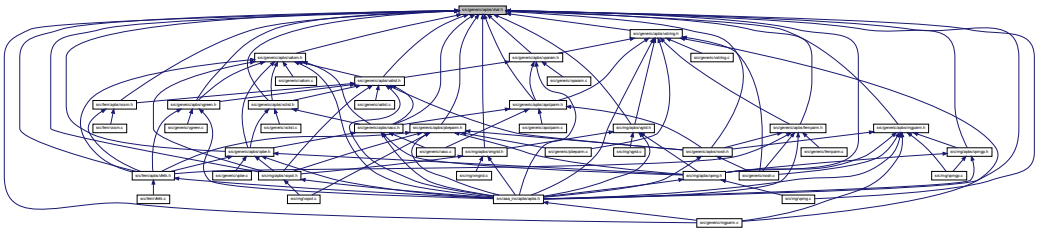
```
00001
00057 #ifndef _VGREEN_H_
00058 #define _VGREEN_H_
00059
00060 /* Generic headers */
00061 #include "maloc/maloc.h"
00062 #include "apbs/vhal.h"
00063
00064 /* Specific headers */
00065 #include "apbs/vunit.h"
00066 #include "apbs/vatom.h"
00067 #include "apbs/valist.h"
00068
00069
00075 struct sVgreen {
00076
00077   Valist *alist;
00078   Vmem *vmem;
00079   double *xp;
00081   double *yp;
00083   double *zp;
00085   double *qp;
```

```
00087   int np;
00088 };
00089
00094 typedef struct sVgreen Vgreen;
00095
00096 /* ////////////////////////////////////////////////////////////////////////
00097 // Class Vgreen: Inlineable methods (vgreen.c)
00099
00100 #if !defined(VINLINE_VGREEN)
00101
00109     VEXTERNC Valist* Vgreen_getValist(Vgreen *thee);
00110
00118     VEXTERNC unsigned long int Vgreen_memChk(Vgreen *thee);
00119
00120 #else /* if defined(VINLINE_VGREEN) */
00121 #   define Vgreen_getValist(thee) ((thee)->alist)
00122 #   define Vgreen_memChk(thee) (Vmem_bytes((thee)->vmem))
00123 #endif /* if !defined(VINLINE_VGREEN) */
00124
00125 /* ////////////////////////////////////////////////////////////////////////
00126 // Class Vgreen: Non-Inlineable methods (vgreen.c)
00128
00135 VEXTERNC Vgreen* Vgreen_ctor(Valist *alist);
00136
00144 VEXTERNC int Vgreen_ctor2(Vgreen *thee, Valist *alist);
00145
00151 VEXTERNC void Vgreen_dtor(Vgreen **thee);
00152
00158 VEXTERNC void Vgreen_dtor2(Vgreen *thee);
00159
00184 VEXTERNC int Vgreen_helmholtz(Vgreen *thee, int npos, double *x, double *y,
00185   double *z, double *val, double kappa);
00186
00214 VEXTERNC int Vgreen_helmholtzD(Vgreen *thee, int npos, double *x, double *y,
00215   double *z, double *gradx, double *grady, double *gradz, double kappa);
00216
00237 VEXTERNC int Vgreen_coulomb_direct(Vgreen *thee, int npos, double *x,
00238         double *y, double *z, double *val);
00239
00260 VEXTERNC int Vgreen_coulomb(Vgreen *thee, int npos, double *x, double *y,
00261   double *z, double *val);
00262
00286 VEXTERNC int Vgreen_coulombD_direct(Vgreen *thee, int npos, double *x,
00287         double *y, double *z, double *pot, double *gradx, double *grady, double
00288         *gradz);
00289
00314 VEXTERNC int Vgreen_coulombD(Vgreen *thee, int npos, double *x, double *y,
00315   double *z, double *pot, double *gradx, double *grady, double *gradz);
00316
00317 #endif /* ifndef _VGREEN_H_ */
```

10.41 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

  Maximum number of points on a sphere.

- #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-zer0 (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**
- typedef enum eVsol_Meth **Vsol_Meth**
- 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_TRIL = 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_AVS = 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 1605 2010-09-13 15:12:09Z yhuang01

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
* 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 Washington University in St. Louis 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 src/generic/apbs/vhal.h

```
00001
00056 #ifndef _VAPBSHAL_H_
00057 #define _VAPBSHAL_H_
00058
00065 enum eVrc_Codes {
00066
00067  VRC_WARNING=-1,
00068  VRC_FAILURE=0,
00069  VRC_SUCCESS=1
```

```
00071 };
00072 typedef enum eVrc_Codes Vrc_Codes;
00073
00080 enum eVsol_Meth {
00081
00082  VSOL_CGMG,  /* 0: conjugate gradient multigrid */
00083  VSOL_Newton, /* 1: newton */
00084  VSOL_MG,  /* 2: multigrid */
00085  VSOL_CG,  /* 3: conjugate gradient */
00086  VSOL_SOR,  /* 4: sucessive overrelaxation */
00087  VSOL_RBGS,  /* 5: red-black gauss-seidel */
00088  VSOL_WJ,  /* 6: weighted jacobi */
00089  VSOL_Richardson,/* 7: richardson  */
00090  VSOL_CGMGAqua, /* 8: conjugate gradient multigrid aqua */
00091  VSOL_NewtonAqua /* 9: newton aqua */
00092
00093 };
00094 typedef enum eVsol_Meth Vsol_Meth;
00095
00101 enum eVsurf_Meth {
00102  VSM_MOL=0,
00106     VSM_MOLSMOOTH=1,
00108  VSM_SPLINE=2,
00118  VSM_SPLINE3=3,
00122  VSM_SPLINE4=4
00126 };
00127
00132 typedef enum eVsurf_Meth Vsurf_Meth;
00133
00138 enum eVhal_PBEType {
00139     PBE_LPBE,
00140     PBE_NPBE,
00141     PBE_LRPBE,
00142     PBE_NRPBE,
00143  PBE_SMPBE
00144 };
00145
00150 typedef enum eVhal_PBEType Vhal_PBEType;
00151
00156 enum eVhal_IPKEYType {
00157     IPKEY_SMPBE = -2,
00158  IPKEY_LPBE,
00159     IPKEY_NPBE
00160 };
00161
00166 typedef enum eVhal_IPKEYType Vhal_IPKEYType;
00167
00172 enum eVhal_NONLINType {
00173     NONLIN_LPBE = 0,
00174  NONLIN_NPBE,
00175     NONLIN_SMPBE,
00176  NONLIN_LPBEAQUA,
00177  NONLIN_NPBEAQUA
00178 };
00179
00184 typedef enum eVhal_NONLINType Vhal_NONLINType;
00185
```

```
00190 enum eVoutput_Format {
00191     OUTPUT_NULL,
00192     OUTPUT_FLAT,
00193 };
00194
00199 typedef enum eVoutput_Format Voutput_Format;
00200
00206 enum eVbcfl {
00207     BCFL_ZERO=0,
00208     BCFL_SDH=1,
00210     BCFL_MDH=2,
00212     BCFL_UNUSED=3,
00213     BCFL_FOCUS=4,
00214     BCFL_MEM=5,
00215  BCFL_MAP=6
00216 };
00217
00222 typedef enum eVbcfl Vbcfl;
00223
00229 enum eVchrg_Meth {
00230  VCM_TRIL=0,
00233     VCM_BSPL2=1,
00236     VCM_BSPL4=2
00237 };
00238
00243 typedef enum eVchrg_Meth Vchrg_Meth;
00244
00250 enum eVchrg_Src {
00251  VCM_CHARGE=0,
00252  VCM_PERMANENT=1,
00253     VCM_INDUCED=2,
00254     VCM_NLINDUCED=3
00255 };
00256
00261 typedef enum eVchrg_Src Vchrg_Src;
00262
00268 enum eVdata_Type {
00269     VDT_CHARGE,
00270     VDT_POT,
00271  VDT_ATOMPOT,
00272     VDT_SMOL,
00274     VDT_SSPL,
00276     VDT_VDW,
00278     VDT_IVDW,
00280     VDT_LAP,
00281     VDT_EDENS,
00283     VDT_NDENS,
00285     VDT_QDENS,
00287     VDT_DIELX,
00289     VDT_DIELY,
00291     VDT_DIELZ,
00293     VDT_KAPPA
00295 };
00296
00301 typedef enum eVdata_Type Vdata_Type;
00302
00308 enum eVdata_Format {
```

```
00309     VDF_DX=0,
00310     VDF_UHBD=1,
00311     VDF_AVS=2,
00312  VDF_MCSF=3,
00313  VDF_GZ=4,
00314  VDF_FLAT=5
00315 };
00316
00321 typedef enum eVdata_Format Vdata_Format;
00322
00327 #define APBS_TIMER_WALL_CLOCK 26
00328
00333 #define APBS_TIMER_SETUP 27
00334
00339 #define APBS_TIMER_SOLVER 28
00340
00345 #define APBS_TIMER_ENERGY 29
00346
00351 #define APBS_TIMER_FORCE 30
00352
00357 #define APBS_TIMER_TEMP1 31
00358
00363 #define APBS_TIMER_TEMP2 32
00364
00369 #define MAXMOL 5
00370
00375 #define MAXION 10
00376
00380 #define MAXFOCUS 5
00381
00385 #define VMGNLEV 4
00386
00390 #define VREDFRAC 0.25
00391
00395 #define VAPBS_NVS 4
00396
00400 #define VAPBS_DIM 3
00401
00406 #define VAPBS_RIGHT 0
00407
00412 #define MAX_SPHERE_PTS 50000
00413
00418 #define VAPBS_FRONT 1
00419
00424 #define VAPBS_UP    2
00425
00430 #define VAPBS_LEFT  3
00431
00436 #define VAPBS_BACK  4
00437
00442 #define VAPBS_DOWN  5
00443
00448 #define VPMGSMALL 1e-12
00449
00454 #define SINH_MIN -85.0
00455
00460 #define SINH_MAX 85.0
```

```
00461
00462
00463 #if defined(VDEBUG)
00464 #   if !defined(APBS_NOINLINE)
00465 #       define APBS_NOINLINE 1
00466 #   endif
00467 #endif
00468
00469 #if !defined(APBS_NOINLINE)
00470
00474 #   define VINLINE_VACC
00475
00479 #   define VINLINE_VATOM
00480
00484 #   define VINLINE_VCSM
00485
00489 #   define VINLINE_VPBE
00490
00494 #   define VINLINE_VPEE
00495
00499 #   define VINLINE_VGREEN
00500
00504 #   define VINLINE_VFETK
00505
00509 #   define VINLINE_VPMG
00510
00515 #   define MAX_HASH_DIM 75
00516
00517 #endif
00518
00519 /* Fortran name mangling */
00520 #if defined(VF77_UPPERCASE)
00521 #   if defined(VF77_NOUNDERSCORE)
00522 #       define VF77_MANGLE(name,NAME) NAME
00523 #   elif defined(VF77_ONEUNDERSCORE)
00524 #       define VF77_MANGLE(name,NAME) NAME ## _
00525 #   else
00526 #       define VF77_MANGLE(name,NAME) name
00527 #   endif
00528 #else
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
00537 #       define VF77_MANGLE(name,NAME) name
00538 #   endif
00539 #endif
00540
00541 /* Floating Point Error */
00542 #if defined(MACHINE_EPS)
00543 #     define VFLOOR(value) \
00544               ((floor(value) != floor(value + MACHINE_EPS)) ? \
00545               floor(value + MACHINE_EPS) : floor(value))
00546 #else
00547
```

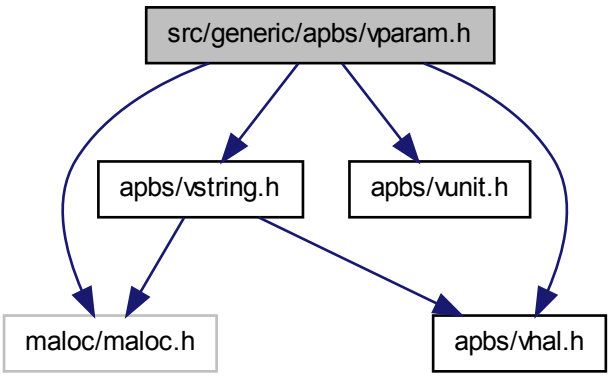
```
00552 #     define VFLOOR(value) floor(value)
00553 #endif
00554
00555 /* String embedding for ident */
00556 #if defined(HAVE_EMBED)
00557
00561 #    define VEMBED(rctag) \
00562         VPRIVATE const char* rctag; \
00563         static void* use_rcsid=(0 ? &use_rcsid : (void**)&rcsid);
00564 #else
00565
00569 #    define VEMBED(rctag)
00570 #endif /* if defined(HAVE_EMBED) */
00571
00572 #endif /* #ifndef _VAPBSHAL_H_ */
```

10.43 src/generic/apbs/vparam.h File Reference

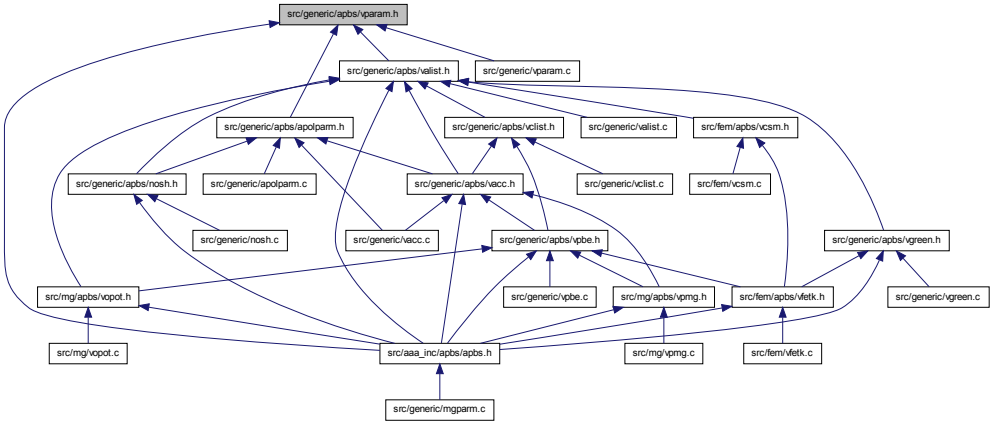
Contains declarations for class Vparam.

```
#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 1552 2010-02-10 17:46:27Z yhuang01

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
* 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 Washington University in St. Louis 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 src/generic/apbs/vparam.h

```
00001
00054 #ifndef _VPARAM_H_
00055 #define _VPARAM_H_
00056
00057 /* Generic headers */
00058 #include "maloc/maloc.h"
00059 #include "apbs/vhal.h"
00060 #include "apbs/vunit.h"
00061 #include "apbs/vstring.h"
```

```
00062
00079 struct sVparam_AtomData {
00080     char atomName[VMAX_ARGLEN];
00081     char resName[VMAX_ARGLEN];
00082     double charge;
00083     double radius;
00084     double epsilon;
00086 };
00087
00093 typedef struct sVparam_AtomData Vparam_AtomData;
00094
00101 struct Vparam_ResData {
00102     Vmem *vmem;
00103     char name[VMAX_ARGLEN];
00104     int nAtomData;
00106     Vparam_AtomData *atomData;
00107 };
00108
00114 typedef struct Vparam_ResData Vparam_ResData;
00115
00122 struct Vparam {
00123
00124   Vmem *vmem;
00125   int nResData;
00127   Vparam_ResData *resData;
00128 };
00129
00134 typedef struct Vparam Vparam;
00135
00136 /* ////////////////////////////////////////////////////////////////////////////
00137 // Class Vparam: Inlineable methods (vparam.c)
00139
00140 #if !defined(VINLINE_VPARAM)
00141
00148     VEXTERNC unsigned long int Vparam_memChk(Vparam *thee);
00149
00150 #else /* if defined(VINLINE_VPARAM) */
00151
00152 #   define Vparam_memChk(thee) (Vmem_bytes((thee)->vmem))
00153
00154 #endif /* if !defined(VINLINE_VPARAM) */
00155
00156 /* ////////////////////////////////////////////////////////////////////////////
00157 // Class Vparam: Non-Inlineable methods (vparam.c)
00159
00164 VEXTERNC Vparam_AtomData* Vparam_AtomData_ctor();
00165
00171 VEXTERNC int Vparam_AtomData_ctor2(Vparam_AtomData *thee);
00172
00177 VEXTERNC void Vparam_AtomData_dtor(Vparam_AtomData **thee);
00178
00183 VEXTERNC void Vparam_AtomData_dtor2(Vparam_AtomData *thee);
00184
00192 VEXTERNC void Vparam_AtomData_copyTo(Vparam_AtomData *thee,
00193   Vparam_AtomData *dest);
00194
00202 VEXTERNC void Vparam_ResData_copyTo(Vparam_ResData *thee,
```

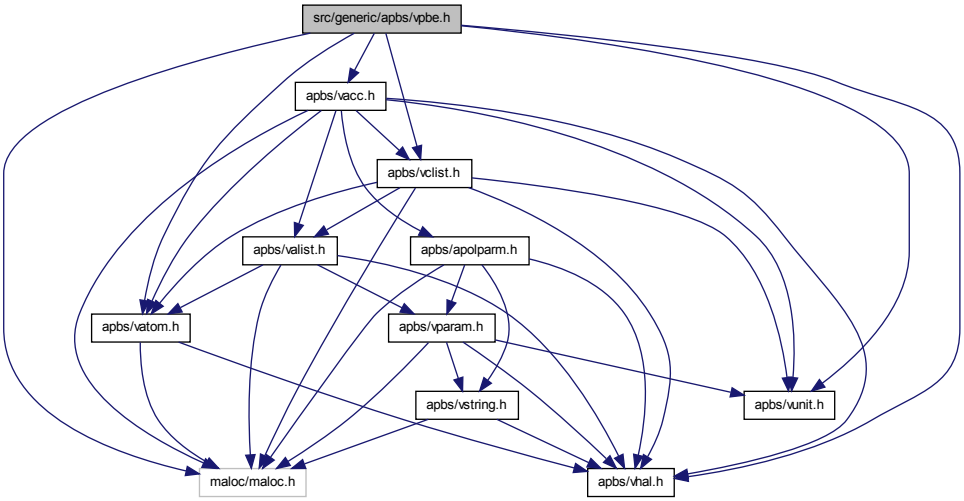
```
00203   Vparam_ResData *dest);
00204
00212 VEXTERNC void Vparam_AtomData_copyFrom(Vparam_AtomData *thee,
00213   Vparam_AtomData *src);
00214
00220 VEXTERNC Vparam_ResData* Vparam_ResData_ctor(Vmem *mem);
00221
00228 VEXTERNC int Vparam_ResData_ctor2(Vparam_ResData *thee, Vmem *mem);
00229
00234 VEXTERNC void Vparam_ResData_dtor(Vparam_ResData **thee);
00235
00240 VEXTERNC void Vparam_ResData_dtor2(Vparam_ResData *thee);
00241
00246 VEXTERNC Vparam* Vparam_ctor();
00247
00253 VEXTERNC int Vparam_ctor2(Vparam *thee);
00254
00259 VEXTERNC void Vparam_dtor(Vparam **thee);
00260
00265 VEXTERNC void Vparam_dtor2(Vparam *thee);
00266
00277 VEXTERNC Vparam_ResData* Vparam_getResData(Vparam *thee,
00278   char resName[VMAX_ARGLEN]);
00279
00291 VEXTERNC Vparam_AtomData* Vparam_getAtomData(Vparam *thee,
00292   char resName[VMAX_ARGLEN], char atomName[VMAX_ARGLEN]);
00293
00322 VEXTERNC int Vparam_readFlatFile(Vparam *thee, const char *iodev,
00323   const char *iofmt, const char *thost, const char *fname);
00324
00335 VEXTERNC int Vparam_readXMLFile(Vparam *thee, const char *iodev,
00336   const char *iofmt, const char *thost, const char *fname);
00337
00338 #endif    /* ifndef _VPARAM_H_ */
```

10.45 src/generic/apbs/vpbe.h File Reference

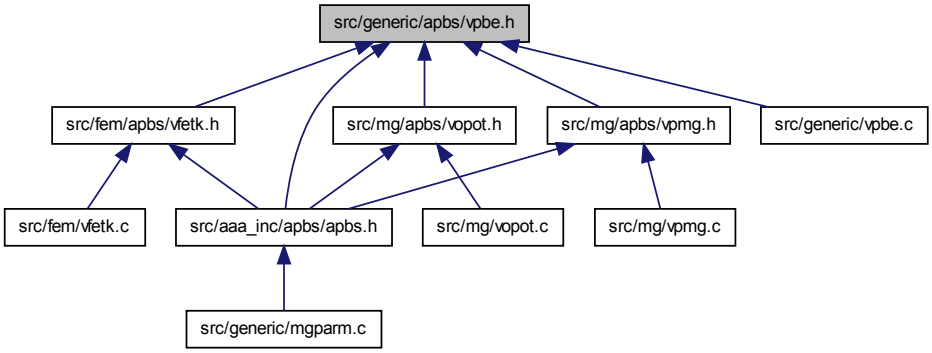
Contains declarations for class Vpbe.

```
#include "maloc/maloc.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)

  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)

  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)
  aauthor 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 1565 2010-03-07 16:06:27Z sobolevnrm

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
```

```
 * Nathan A. Baker (nathan.baker@pnl.gov)
 * Pacific Northwest National Laboratory
 *
 * Additional contributing authors listed in the code documentation.
 *
 * Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
 * 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 Washington University in St. Louis 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 src/generic/apbs/vpbe.h

```
00001
00058 #ifndef _VPBE_H_
00059 #define _VPBE_H_
00060
00061 /* Generic headers */
00062 #include "maloc/maloc.h"
00063 #include "apbs/vhal.h"
00064
00065 /* Specific headers */
00066 #include "apbs/vunit.h"
00067 #include "apbs/vatom.h"
00068 #include "apbs/vacc.h"
00069 #include "apbs/vclist.h"
00070
```

```
00076 struct sVpbe {
00077
00078  Vmem *vmem;
00080  Valist *alist;
00081  Vclist *clist;
00082  Vacc *acc;
00084  double T;
00085  double soluteDiel;
00086  double solventDiel;
00087  double solventRadius;
00091  double bulkIonicStrength;
00092  double maxIonRadius;
00095  int    numIon;
00096  double ionConc[MAXION];
00097  double ionRadii[MAXION];
00098  double ionQ[MAXION];
00100  double xkappa;
00101  double deblen;
00102  double zkappa2;
00103  double zmagic;
00105  double soluteCenter[3];
00106  double soluteRadius;
00107  double soluteXlen;
00108  double soluteYlen;
00109  double soluteZlen;
00110  double soluteCharge;
00112  double smvolume;
00113  double smsize;
00114  int ipkey;
00117  int paramFlag;
00119  /*-----------------------------------------------------------*/
00120  /* Added by Michael Grabe                                    */
00121  /*-----------------------------------------------------------*/
00122
00123  double z_mem;
00124  double L;
00125  double membraneDiel;
00126  double V;
00127  int param2Flag;
00128  /*-----------------------------------------------------------*/
00129
00130 };
00131
00136 typedef struct sVpbe Vpbe;
00137
00138 /* ///////////////////////////////////////////////////////////////////////////
00139    // Class Vpbe: Inlineable methods (vpbe.c)
00141
00142 #if !defined(VINLINE_VPBE)
00143
00150 VEXTERNC Valist* Vpbe_getValist(Vpbe *thee);
00151
00158 VEXTERNC Vacc*   Vpbe_getVacc(Vpbe *thee);
00159
00166 VEXTERNC double  Vpbe_getBulkIonicStrength(Vpbe *thee);
00167
00174 VEXTERNC double  Vpbe_getMaxIonRadius(Vpbe *thee);
```

```
00175
00182 VEXTERNC double  Vpbe_getTemperature(Vpbe *thee);
00183
00190 VEXTERNC double  Vpbe_getSoluteDiel(Vpbe *thee);
00191
00198 VEXTERNC double  Vpbe_getGamma(Vpbe *thee);
00199
00206 VEXTERNC double  Vpbe_getSoluteRadius(Vpbe *thee);
00207
00214 VEXTERNC double  Vpbe_getSoluteXlen(Vpbe *thee);
00215
00222 VEXTERNC double  Vpbe_getSoluteYlen(Vpbe *thee);
00223
00230 VEXTERNC double  Vpbe_getSoluteZlen(Vpbe *thee);
00231
00238 VEXTERNC double* Vpbe_getSoluteCenter(Vpbe *thee);
00239
00246 VEXTERNC double  Vpbe_getSoluteCharge(Vpbe *thee);
00247
00254 VEXTERNC double  Vpbe_getSolventDiel(Vpbe *thee);
00255
00262 VEXTERNC double  Vpbe_getSolventRadius(Vpbe *thee);
00263
00270 VEXTERNC double  Vpbe_getXkappa(Vpbe *thee);
00271
00278 VEXTERNC double  Vpbe_getDeblen(Vpbe *thee);
00279
00286 VEXTERNC double  Vpbe_getZkappa2(Vpbe *thee);
00287
00294 VEXTERNC double  Vpbe_getZmagic(Vpbe *thee);
00295
00296 /*-----------------------------------------------------------------*/
00297 /* Added by Michael Grabe                                          */
00298 /*-----------------------------------------------------------------*/
00299
00306 VEXTERNC double  Vpbe_getzmem(Vpbe *thee);
00307
00314 VEXTERNC double  Vpbe_getLmem(Vpbe *thee);
00315
00322 VEXTERNC double  Vpbe_getmembraneDiel(Vpbe *thee);
00323
00329 VEXTERNC double  Vpbe_getmemv(Vpbe *thee);
00330
00331 /*-----------------------------------------------------------------*/
00332
00333 #else /* if defined(VINLINE_VPBE) */
00334 #   define Vpbe_getValist(thee) ((thee)->alist)
00335 #   define Vpbe_getVacc(thee) ((thee)->acc)
00336 #   define Vpbe_getBulkIonicStrength(thee) ((thee)->bulkIonicStrength)
00337 #   define Vpbe_getTemperature(thee) ((thee)->T)
00338 #   define Vpbe_getSoluteDiel(thee) ((thee)->soluteDiel)
00339 #   define Vpbe_getSoluteCenter(thee) ((thee)->soluteCenter)
00340 #   define Vpbe_getSoluteRadius(thee) ((thee)->soluteRadius)
00341 #   define Vpbe_getSoluteXlen(thee) ((thee)->soluteXlen)
00342 #   define Vpbe_getSoluteYlen(thee) ((thee)->soluteYlen)
00343 #   define Vpbe_getSoluteZlen(thee) ((thee)->soluteZlen)
00344 #   define Vpbe_getSoluteCharge(thee) ((thee)->soluteCharge)
```

```
00345 #   define Vpbe_getSolventDiel(thee) ((thee)->solventDiel)
00346 #   define Vpbe_getSolventRadius(thee) ((thee)->solventRadius)
00347 #   define Vpbe_getMaxIonRadius(thee) ((thee)->maxIonRadius)
00348 #   define Vpbe_getXkappa(thee) ((thee)->xkappa)
00349 #   define Vpbe_getDeblen(thee) ((thee)->deblen)
00350 #   define Vpbe_getZkappa2(thee) ((thee)->zkappa2)
00351 #   define Vpbe_getZmagic(thee) ((thee)->zmagic)
00352
00353 /*-------------------------------------------------------------*/
00354 /* Added by Michael Grabe                                      */
00355 /*-------------------------------------------------------------*/
00356
00357 #   define Vpbe_getzmem(thee) ((thee)->z_mem)
00358 #   define Vpbe_getLmem(thee) ((thee)->L)
00359 #   define Vpbe_getmembraneDiel(thee) ((thee)->membraneDiel)
00360 #   define Vpbe_getmemv(thee) ((thee)->V)
00361
00362 /*-------------------------------------------------------------*/
00363
00364
00365 #endif /* if !defined(VINLINE_VPBE) */
00366
00367 /* ///////////////////////////////////////////////////////////////////////
00368    // Class Vpbe: Non-Inlineable methods (vpbe.c)
00370
00391 VEXTERNC Vpbe*   Vpbe_ctor(
00392        Valist *alist,
00393        int ionNum,
00394        double *ionConc,
00395          double *ionRadii,
00396        double *ionQ,
00397        double T,
00398        double soluteDiel,
00399        double solventDiel,
00400        double solventRadius,
00401        int focusFlag,
00402        double sdens,
00403        double z_mem,
00404        double L,
00405        double membraneDiel,
00406        double V
00407        );
00408
00429 VEXTERNC int    Vpbe_ctor2(
00430        Vpbe *thee,
00431        Valist *alist,
00432        int ionNum,
00433        double *ionConc,
00434        double *ionRadii,
00435        double *ionQ,
00436        double T,
00437        double soluteDiel,
00438        double solventDiel,
00439        double solventRadius,
00440        int focusFlag,
00441        double sdens,
00442        double z_mem,
```

```
00443         double L,
00444         double membraneDiel,
00445         double V
00446         );
00447
00458 VEXTERNC int     Vpbe_getIons(Vpbe *thee, int *nion, double ionConc[MAXION],
00459           double ionRadii[MAXION], double ionQ[MAXION]);
00460
00466 VEXTERNC void    Vpbe_dtor(Vpbe **thee);
00467
00473 VEXTERNC void    Vpbe_dtor2(Vpbe *thee);
00474
00489 VEXTERNC double  Vpbe_getCoulombEnergy1(Vpbe *thee);
00490
00498 VEXTERNC unsigned long int Vpbe_memChk(Vpbe *thee);
00499
00500 #endif /* ifndef _VPBE_H_ */
```

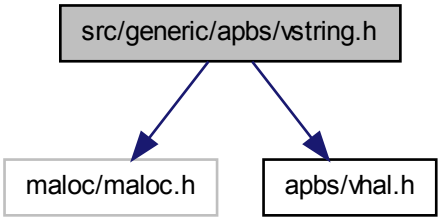
10.47 src/generic/apbs/vstring.h File Reference

Contains declarations for class Vstring.

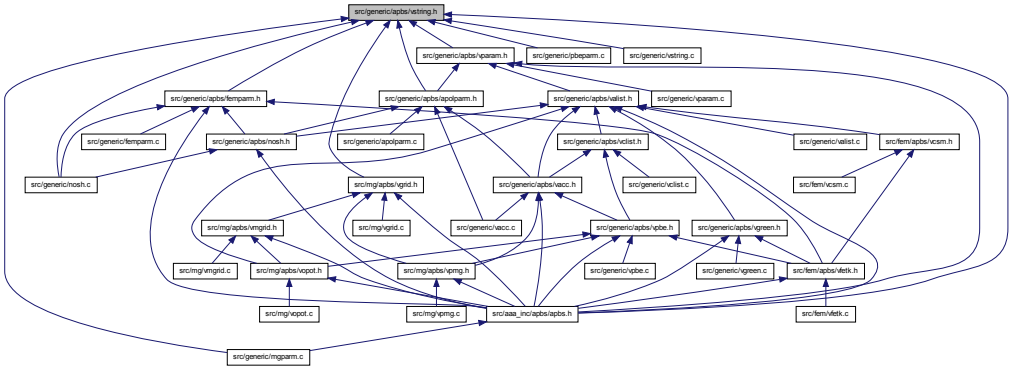
```
#include "maloc/maloc.h"
```

```
#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 1552 2010-02-10 17:46:27Z yhuang01

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
```

```
 *
 * Nathan A. Baker (nathan.baker@pnl.gov)
 * Pacific Northwest National Laboratory
 *
 * Additional contributing authors listed in the code documentation.
 *
 * Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
 * 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 Washington University in St. Louis 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.

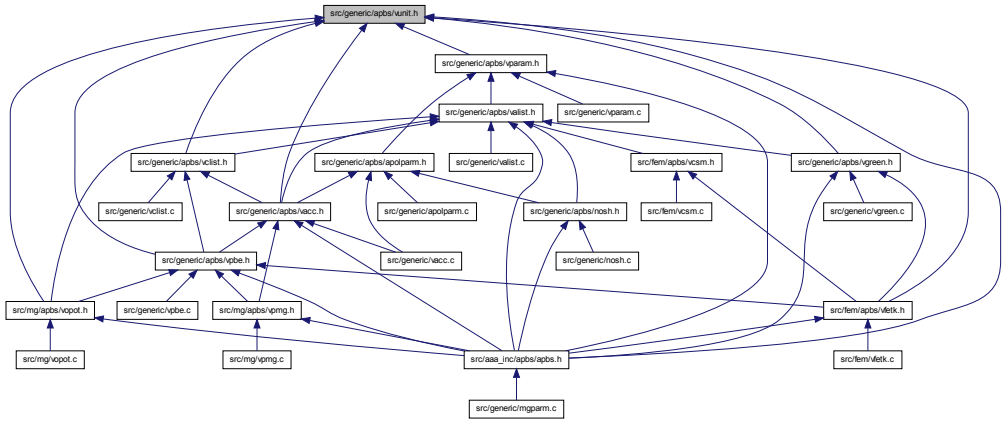
10.48 src/generic/apbs/vstring.h

```
00001
00054 #ifndef _VSTRING_H_
00055 #define _VSTRING_H_
00056
00057 #include "maloc/maloc.h"
00058 #include "apbs/vhal.h"
00059
00072 VEXTERNC int Vstring_strcasecmp(const char *s1, const char *s2);
00073
00080 VEXTERNC int Vstring_isdigit(const char *tok);
00081
00082 #endif    /* ifndef _VSTRING_H_ */
```

10.49 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 1552 2010-02-10 17:46:27Z yhuang01

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
```

```
 * Additional contributing authors listed in the code documentation.
 *
 * Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
 * 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 Washington University in St. Louis 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 src/generic/apbs/vunit.h

```
00001
00055 #ifndef _VUNIT_H_
00056 #define _VUNIT_H_
00057
00060 #define Vunit_J_to_cal  4.1840000e+00
00061
00064 #define Vunit_cal_to_J  2.3900574e-01
00065
00068 #define Vunit_amu_to_kg  1.6605402e-27
00069
00072 #define Vunit_kg_to_amu  6.0221367e+26
00073
00076 #define Vunit_ec_to_C  1.6021773e-19
00077
00080 #define Vunit_C_to_ec  6.2415065e+18
00081
00084 #define Vunit_ec  1.6021773e-19
```

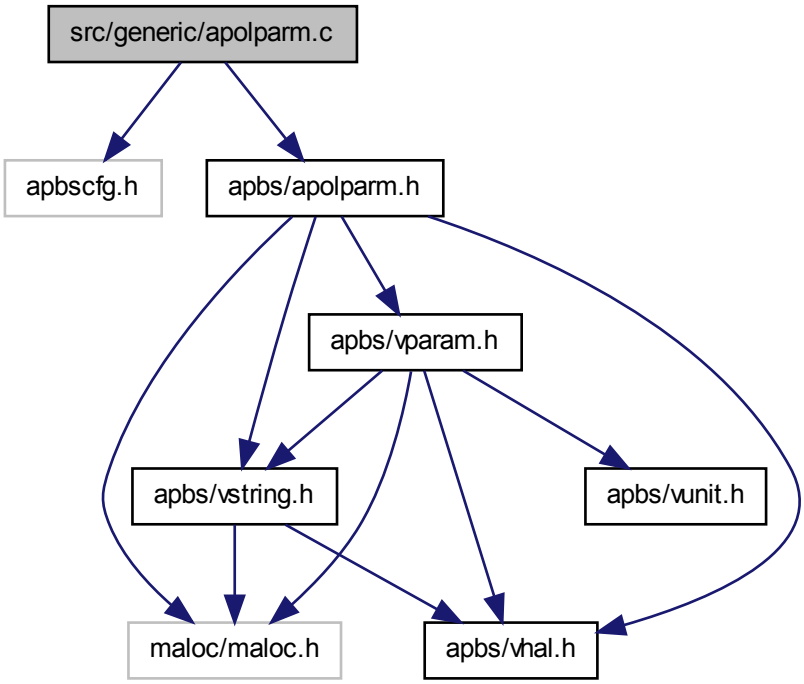
```
00085
00088 #define Vunit_kb  1.3806581e-23
00089
00092 #define Vunit_Na  6.0221367e+23
00093
00096 #define Vunit_pi  VPI
00097
00100 #define Vunit_eps0  8.8541878e-12
00101
00104 #define Vunit_esu_ec2A  3.3206364e+02
00105
00108 #define Vunit_esu_kb            1.9871913e-03
00109
00110 #endif /* ifndef _VUNIT_H_ */
```

10.51 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 1552 2010-02-10 17:46:27Z yhuang01

Attention

```
 *
 * APBS -- Adaptive Poisson-Boltzmann Solver
 *
 * Nathan A. Baker (nathan.baker@pnl.gov)
 * Pacific Northwest National Laboratory
 *
 * Additional contributing authors listed in the code documentation.
 *
 * Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
 * 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 Washington University in St. Louis 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 src/generic/apolparm.c

```
00001
00050 #include "apbscfg.h"
00051 #include "apbs/apolparm.h"
00052
00053 VEMBED(rcsid="$Id: apolparm.c 1552 2010-02-10 17:46:27Z yhuang01 $")
00054
00055 #if !defined(VINLINE_MGPARM)
00056
00057 #endif /* if !defined(VINLINE_MGPARM) */
```

```
00058
00059 VPUBLIC APOLparm* APOLparm_ctor() {
00060
00061     /* Set up the structure */
00062     APOLparm *thee = VNULL;
00063     thee = Vmem_malloc(VNULL, 1, sizeof(APOLparm));
00064     VASSERT( thee != VNULL);
00065     VASSERT( APOLparm_ctor2(thee) == VRC_SUCCESS );
00066
00067     return thee;
00068 }
00069
00070 VPUBLIC Vrc_Codes APOLparm_ctor2(APOLparm *thee) {
00071
00072     int i;
00073
00074     if (thee == VNULL) return VRC_FAILURE;
00075
00076     thee->parsed = 0;
00077
00078     thee->setgrid = 0;
00079     thee->setmolid = 0;
00080     thee->setbconc = 0;
00081     thee->setsdens = 0;
00082     thee->setdpos = 0;
00083     thee->setpress = 0;
00084     thee->setsrfm = 0;
00085     thee->setsrad = 0;
00086     thee->setswin = 0;
00087
00088  thee->settemp = 0;
00089     thee->setgamma = 0;
00090
00091  thee->setwat = 0;
00092
00093  thee->sav = 0.0;
00094  thee->sasa = 0.0;
00095  thee->wcaEnergy = 0.0;
00096
00097  for(i=0;i<3;i++) thee->totForce[i] = 0.0;
00098
00099     return VRC_SUCCESS;
00100 }
00101
00102 VPUBLIC void APOLparm_copy(
00103         APOLparm *thee,
00104         APOLparm *source
00105         ) {
00106
00107  int i;
00108
00109  thee->parsed = source->parsed;
00110
00111  for (i=0; i<3; i++) thee->grid[i] = source->grid[i];
00112  thee->setgrid = source->setgrid;
00113
00114  thee->molid = source->molid;
```

```
00115  thee->setmolid = source->setmolid;
00116
00117  thee->bconc = source->bconc ;
00118  thee->setbconc= source->setbconc ;
00119
00120  thee->sdens = source->sdens ;
00121  thee->setsdens= source->setsdens ;
00122
00123  thee->dpos = source->dpos ;
00124  thee->setdpos= source->setdpos ;
00125
00126  thee->press = source->press ;
00127  thee->setpress = source->setpress ;
00128
00129  thee->srfm = source->srfm ;
00130  thee->setsrfm = source->setsrfm ;
00131
00132  thee->srad = source->srad ;
00133  thee->setsrad = source->setsrad ;
00134
00135  thee->swin = source->swin ;
00136  thee->setswin = source->setswin ;
00137
00138  thee->temp = source->temp ;
00139  thee->settemp = source->settemp ;
00140
00141  thee->gamma = source->gamma ;
00142  thee->setgamma = source->setgamma ;
00143
00144  thee->calcenergy = source->calcenergy ;
00145  thee->setcalcenergy = source->setcalcenergy ;
00146
00147  thee->calcforce = source->calcforce ;
00148  thee->setcalcforce = source->setcalcforce ;
00149
00150  thee->setwat = source->setwat ;
00151
00152  thee->sav = source->sav;
00153  thee->sasa = source->sasa;
00154  thee->wcaEnergy = source->wcaEnergy;
00155
00156  for(i=0;i<3;i++) thee->totForce[i] = source->totForce[i];
00157
00158  return;
00159 }
00160
00161 VPUBLIC void APOLparm_dtor(APOLparm **thee) {
00162     if ((*thee) != VNULL) {
00163         APOLparm_dtor2(*thee);
00164         Vmem_free(VNULL, 1, sizeof(APOLparm), (void **)thee);
00165         (*thee) = VNULL;
00166     }
00167
00168  return;
00169 }
00170
00171 VPUBLIC void APOLparm_dtor2(APOLparm *thee) { ; }
```

```
00172
00173 VPUBLIC Vrc_Codes APOLparm_check(APOLparm *thee) {
00174
00175
00176     Vrc_Codes rc;
00177     rc = VRC_SUCCESS;
00178
00179     if (!thee->parsed) {
00180         Vnm_print(2, "APOLparm_check:  not filled!\n");
00181         return VRC_FAILURE;
00182     }
00183     if (!thee->setgrid) {
00184         Vnm_print(2, "APOLparm_check:  grid not set!\n");
00185         rc = VRC_FAILURE;
00186     }
00187     if (!thee->setmolid) {
00188         Vnm_print(2, "APOLparm_check:  molid not set!\n");
00189         rc = VRC_FAILURE;
00190     }
00191     if (!thee->setbconc) {
00192         Vnm_print(2, "APOLparm_check:  bconc not set!\n");
00193         rc = VRC_FAILURE;
00194     }
00195     if (!thee->setsdens) {
00196         Vnm_print(2, "APOLparm_check:  sdens not set!\n");
00197         rc = VRC_FAILURE;
00198     }
00199     if (!thee->setdpos) {
00200         Vnm_print(2, "APOLparm_check:  dpos not set!\n");
00201         rc = VRC_FAILURE;
00202     }
00203     if (!thee->setpress) {
00204         Vnm_print(2, "APOLparm_check:  press not set!\n");
00205         rc = VRC_FAILURE;
00206     }
00207     if (!thee->setsrfm) {
00208         Vnm_print(2, "APOLparm_check:  srfm not set!\n");
00209         rc = VRC_FAILURE;
00210     }
00211     if (!thee->setsrad) {
00212         Vnm_print(2, "APOLparm_check:  srad not set!\n");
00213         rc = VRC_FAILURE;
00214     }
00215     if (!thee->setswin) {
00216         Vnm_print(2, "APOLparm_check:  swin not set!\n");
00217         rc = VRC_FAILURE;
00218     }
00219     if (!thee->settemp) {
00220         Vnm_print(2, "APOLparm_check:  temp not set!\n");
00221         rc = VRC_FAILURE;
00222     }
00223  if (!thee->setgamma) {
00224         Vnm_print(2, "APOLparm_check:  gamma not set!\n");
00225         rc = VRC_FAILURE;
00226     }
00227     return rc;
00228
```

```
00229 }
00230
00231 VPRIVATE Vrc_Codes APOLparm_parseGRID(APOLparm *thee, Vio *sock) {
00232
00233  char tok[VMAX_BUFSIZE];
00234  double tf;
00235
00236  VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00237  if (sscanf(tok, "%lf", &tf) == 0) {
00238   Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GRID \
00239 keyword!\n", tok);
00240   return VRC_WARNING;
00241  } else thee->grid[0] = tf;
00242  VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00243  if (sscanf(tok, "%lf", &tf) == 0) {
00244   Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GRID \
00245 keyword!\n", tok);
00246   return VRC_WARNING;
00247  } else thee->grid[1] = tf;
00248  VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00249  if (sscanf(tok, "%lf", &tf) == 0) {
00250   Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GRID \
00251 keyword!\n", tok);
00252   return VRC_WARNING;
00253  } else thee->grid[2] = tf;
00254     thee->setgrid = 1;
00255     return VRC_SUCCESS;
00256
00257 VERROR1:
00258         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00259  return VRC_WARNING;
00260 }
00261
00262 VPRIVATE Vrc_Codes APOLparm_parseMOL(APOLparm *thee, Vio *sock) {
00263     int ti;
00264     char tok[VMAX_BUFSIZE];
00265
00266     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00267     if (sscanf(tok, "%d", &ti) == 0) {
00268         Vnm_print(2, "NOsh:  Read non-int (%s) while parsing MOL \
00269 keyword!\n", tok);
00270         return VRC_WARNING;
00271     }
00272     thee->molid = ti;
00273     thee->setmolid = 1;
00274     return VRC_SUCCESS;
00275
00276 VERROR1:
00277         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00278  return VRC_WARNING;
00279 }
00280
00281 VPRIVATE Vrc_Codes APOLparm_parseSRFM(APOLparm *thee, Vio *sock) {
00282     char tok[VMAX_BUFSIZE];
00283
00284     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00285
```

```
00286     if (Vstring_strcasecmp(tok, "sacc") == 0) {
00287         thee->srfm = VSM_MOL;
00288         thee->setsrfm = 1;
00289         return VRC_SUCCESS;
00290     } else {
00291         printf("parseAPOL: Unrecongnized keyword (%s) when parsing srfm!\n", tok)
    ;
00292   printf("parseAPOL: Accepted values for srfm = sacc\n");
00293         return VRC_WARNING;
00294     }
00295 
00296     return VRC_FAILURE;
00297 
00298 VERROR1:
00299         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00300  return VRC_WARNING;
00301 }
00302 
00303 VPRIVATE Vrc_Codes APOLparm_parseSRAD(APOLparm *thee, Vio *sock) {
00304     char tok[VMAX_BUFSIZE];
00305     double tf;
00306 
00307     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00308     if (sscanf(tok, "%lf", &tf) == 0) {
00309         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SRAD \
00310 keyword!\n", tok);
00311         return VRC_WARNING;
00312     }
00313     thee->srad = tf;
00314     thee->setsrad = 1;
00315     return VRC_SUCCESS;
00316 
00317 VERROR1:
00318         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00319  return VRC_WARNING;
00320 }
00321 
00322 VPRIVATE Vrc_Codes APOLparm_parseSWIN(APOLparm *thee, Vio *sock) {
00323     char tok[VMAX_BUFSIZE];
00324     double tf;
00325 
00326     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00327     if (sscanf(tok, "%lf", &tf) == 0) {
00328         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SWIN \
00329 keyword!\n", tok);
00330         return VRC_WARNING;
00331     }
00332     thee->swin = tf;
00333     thee->setswin = 1;
00334     return VRC_SUCCESS;
00335 
00336 VERROR1:
00337         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00338  return VRC_WARNING;
00339 }
00340 
00341 VPRIVATE Vrc_Codes APOLparm_parseTEMP(APOLparm *thee, Vio *sock) {
```

```
00342     char tok[VMAX_BUFSIZE];
00343     double tf;
00344
00345     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00346     if (sscanf(tok, "%lf", &tf) == 0) {
00347         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing TEMP \
00348 keyword!\n", tok);
00349         return VRC_WARNING;
00350     }
00351     thee->temp = tf;
00352     thee->settemp = 1;
00353     return VRC_SUCCESS;
00354
00355 VERROR1:
00356         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00357  return VRC_WARNING;
00358 }
00359
00360 VPRIVATE Vrc_Codes APOLparm_parseGAMMA(APOLparm *thee, Vio *sock) {
00361     char tok[VMAX_BUFSIZE];
00362     double tf;
00363
00364     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00365     if (sscanf(tok, "%lf", &tf) == 0) {
00366         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GAMMA \
00367 keyword!\n", tok);
00368         return VRC_WARNING;
00369     }
00370     thee->gamma = tf;
00371     thee->setgamma = 1;
00372     return VRC_SUCCESS;
00373
00374 VERROR1:
00375         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00376  return VRC_WARNING;
00377 }
00378
00379 VPRIVATE Vrc_Codes APOLparm_parseCALCENERGY(APOLparm *thee, Vio *sock) {
00380     char tok[VMAX_BUFSIZE];
00381     int ti;
00382
00383     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00384     /* Parse number */
00385     if (sscanf(tok, "%d", &ti) == 1) {
00386         thee->calcenergy = ti;
00387         thee->setcalcenergy = 1;
00388
00389         Vnm_print(2, "parseAPOL:  Warning -- parsed deprecated \"calcenergy \
00390 %d\" statement.\n", ti);
00391         Vnm_print(2, "parseAPOL:  Please use \"calcenergy ");
00392         switch (thee->calcenergy) {
00393             case ACE_NO:
00394                 Vnm_print(2, "no");
00395                 break;
00396             case ACE_TOTAL:
00397                 Vnm_print(2, "total");
00398                 break;
```

```
00399             case ACE_COMPS:
00400                 Vnm_print(2, "comps");
00401                 break;
00402             default:
00403                 Vnm_print(2, "UNKNOWN");
00404                 break;
00405         }
00406         Vnm_print(2, "\" instead.\n");
00407         return VRC_SUCCESS;
00408     } else if (Vstring_strcasecmp(tok, "no") == 0) {
00409         thee->calcenergy = ACE_NO;
00410         thee->setcalcenergy = 1;
00411         return VRC_SUCCESS;
00412     } else if (Vstring_strcasecmp(tok, "total") == 0) {
00413         thee->calcenergy = ACE_TOTAL;
00414         thee->setcalcenergy = 1;
00415         return VRC_SUCCESS;
00416     } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00417         thee->calcenergy = ACE_COMPS;
00418         thee->setcalcenergy = 1;
00419         return VRC_SUCCESS;
00420     } else {
00421         Vnm_print(2, "NOsh:  Unrecognized parameter (%s) while parsing \
00422 calcenergy!\n", tok);
00423         return VRC_WARNING;
00424     }
00425     return VRC_FAILURE;
00426 
00427 VERROR1:
00428         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00429  return VRC_WARNING;
00430 }
00431 
00432 VPRIVATE Vrc_Codes APOLparm_parseCALCFORCE(APOLparm *thee, Vio *sock) {
00433     char tok[VMAX_BUFSIZE];
00434     int ti;
00435 
00436     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00437     /* Parse number */
00438     if (sscanf(tok, "%d", &ti) == 1) {
00439         thee->calcforce = ti;
00440         thee->setcalcforce = 1;
00441 
00442         Vnm_print(2, "parseAPOL:  Warning -- parsed deprecated \"calcforce \
00443 %d\" statement.\n", ti);
00444         Vnm_print(2, "parseAPOL:  Please use \"calcforce ");
00445         switch (thee->calcenergy) {
00446             case ACF_NO:
00447                 Vnm_print(2, "no");
00448                 break;
00449             case ACF_TOTAL:
00450                 Vnm_print(2, "total");
00451                 break;
00452             case ACF_COMPS:
00453                 Vnm_print(2, "comps");
00454                 break;
00455             default:
```

```
00456                 Vnm_print(2, "UNKNOWN");
00457                 break;
00458         }
00459         Vnm_print(2, "\" instead.\n");
00460         return VRC_SUCCESS;
00461     } else if (Vstring_strcasecmp(tok, "no") == 0) {
00462         thee->calcforce = ACF_NO;
00463         thee->setcalcforce = 1;
00464         return VRC_SUCCESS;
00465     } else if (Vstring_strcasecmp(tok, "total") == 0) {
00466         thee->calcforce = ACF_TOTAL;
00467         thee->setcalcforce = 1;
00468         return VRC_SUCCESS;
00469     } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00470         thee->calcforce = ACF_COMPS;
00471         thee->setcalcforce = 1;
00472         return VRC_SUCCESS;
00473     } else {
00474         Vnm_print(2, "NOsh:  Unrecognized parameter (%s) while parsing \
00475 calcforce!\n", tok);
00476         return VRC_WARNING;
00477     }
00478     return VRC_FAILURE;
00479
00480 VERROR1:
00481         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00482  return VRC_WARNING;
00483 }
00484
00485 VPRIVATE Vrc_Codes APOLparm_parseBCONC(APOLparm *thee, Vio *sock) {
00486     char tok[VMAX_BUFSIZE];
00487     double tf;
00488
00489     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00490     if (sscanf(tok, "%lf", &tf) == 0) {
00491         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing BCONC \
00492 keyword!\n", tok);
00493         return VRC_WARNING;
00494     }
00495     thee->bconc = tf;
00496     thee->setbconc = 1;
00497     return VRC_SUCCESS;
00498
00499 VERROR1:
00500         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00501  return VRC_WARNING;
00502 }
00503
00504 VPRIVATE Vrc_Codes APOLparm_parseSDENS(APOLparm *thee, Vio *sock) {
00505     char tok[VMAX_BUFSIZE];
00506     double tf;
00507
00508     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00509     if (sscanf(tok, "%lf", &tf) == 0) {
00510         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SDENS \
00511 keyword!\n", tok);
00512         return VRC_WARNING;
```

```
00513     }
00514     thee->sdens = tf;
00515     thee->setsdens = 1;
00516     return VRC_SUCCESS;
00517
00518 VERROR1:
00519         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00520  return VRC_WARNING;
00521 }
00522
00523 VPRIVATE Vrc_Codes APOLparm_parseDPOS(APOLparm *thee, Vio *sock) {
00524     char tok[VMAX_BUFSIZE];
00525     double tf;
00526
00527     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00528     if (sscanf(tok, "%lf", &tf) == 0) {
00529         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SDENS \
00530 keyword!\n", tok);
00531         return VRC_WARNING;
00532     }
00533     thee->dpos = tf;
00534     thee->setdpos = 1;
00535
00536  if(thee->dpos < 0.001){
00537   Vnm_print(1,"\nWARNING WARNING WARNING WARNING WARNING\n");
00538   Vnm_print(1,"NOsh: dpos is set to a very small value.\n");
00539   Vnm_print(1,"NOsh: If you are not using a PQR file, you can \
00540 safely ignore this message.\n");
00541   Vnm_print(1,"NOsh: Otherwise please choose a value greater than \
00542 or equal to 0.001.\n\n");
00543  }
00544
00545     return VRC_SUCCESS;
00546
00547 VERROR1:
00548         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00549  return VRC_WARNING;
00550 }
00551
00552 VPRIVATE Vrc_Codes APOLparm_parsePRESS(APOLparm *thee, Vio *sock) {
00553     char tok[VMAX_BUFSIZE];
00554     double tf;
00555
00556     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00557     if (sscanf(tok, "%lf", &tf) == 0) {
00558         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing PRESS \
00559 keyword!\n", tok);
00560         return VRC_WARNING;
00561     }
00562     thee->press = tf;
00563     thee->setpress = 1;
00564     return VRC_SUCCESS;
00565
00566 VERROR1:
00567         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00568  return VRC_WARNING;
00569 }
```

```
00570
00571 VPUBLIC Vrc_Codes APOLparm_parseToken(APOLparm *thee, char tok[VMAX_BUFSIZE],
00572   Vio *sock) {
00573
00574     if (thee == VNULL) {
00575         Vnm_print(2, "parseAPOL:  got NULL thee!\n");
00576         return VRC_WARNING;
00577     }
00578
00579     if (sock == VNULL) {
00580         Vnm_print(2, "parseAPOL:  got NULL socket!\n");
00581         return VRC_WARNING;
00582     }
00583
00584  if (Vstring_strcasecmp(tok, "mol") == 0) {
00585   return APOLparm_parseMOL(thee, sock);
00586  } else if (Vstring_strcasecmp(tok, "grid") == 0) {
00587   return APOLparm_parseGRID(thee, sock);
00588  } else if (Vstring_strcasecmp(tok, "dime") == 0) {
00589   Vnm_print(2, "APOLparm_parseToken:  The DIME and GLEN keywords for APOLAR have
      been replaced with GRID.\n");
00590   Vnm_print(2, "APOLparm_parseToken:  Please see the APBS User Guide for more inf
      ormation.\n");
00591   return VRC_WARNING;
00592  } else if (Vstring_strcasecmp(tok, "glen") == 0) {
00593   Vnm_print(2, "APOLparm_parseToken:  The DIME and GLEN keywords for APOLAR have
      been replaced with GRID.\n");
00594   Vnm_print(2, "APOLparm_parseToken:  Please see the APBS User Guide for more inf
      ormation.\n");
00595   return VRC_WARNING;
00596  } else if (Vstring_strcasecmp(tok, "bconc") == 0) {
00597   return APOLparm_parseBCONC(thee, sock);
00598     } else if (Vstring_strcasecmp(tok, "sdens") == 0) {
00599         return APOLparm_parseSDENS(thee, sock);
00600     } else if (Vstring_strcasecmp(tok, "dpos") == 0) {
00601         return APOLparm_parseDPOS(thee, sock);
00602     }  else if (Vstring_strcasecmp(tok, "srfm") == 0) {
00603         return APOLparm_parseSRFM(thee, sock);
00604     } else if (Vstring_strcasecmp(tok, "srad") == 0) {
00605         return APOLparm_parseSRAD(thee, sock);
00606     } else if (Vstring_strcasecmp(tok, "swin") == 0) {
00607         return APOLparm_parseSWIN(thee, sock);
00608     } else if (Vstring_strcasecmp(tok, "temp") == 0) {
00609         return APOLparm_parseTEMP(thee, sock);
00610     } else if (Vstring_strcasecmp(tok, "gamma") == 0) {
00611         return APOLparm_parseGAMMA(thee, sock);
00612     } else if (Vstring_strcasecmp(tok, "press") == 0) {
00613         return APOLparm_parsePRESS(thee, sock);
00614     } else if (Vstring_strcasecmp(tok, "calcenergy") == 0) {
00615         return APOLparm_parseCALCENERGY(thee, sock);
00616     } else if (Vstring_strcasecmp(tok, "calcforce") == 0) {
00617         return APOLparm_parseCALCFORCE(thee, sock);
00618     } else {
00619         Vnm_print(2, "parseAPOL:  Unrecognized keyword (%s)!\n", tok);
00620         return VRC_WARNING;
00621     }
00622
```

```
00623
00624   return VRC_FAILURE;
00625
00626 }
```

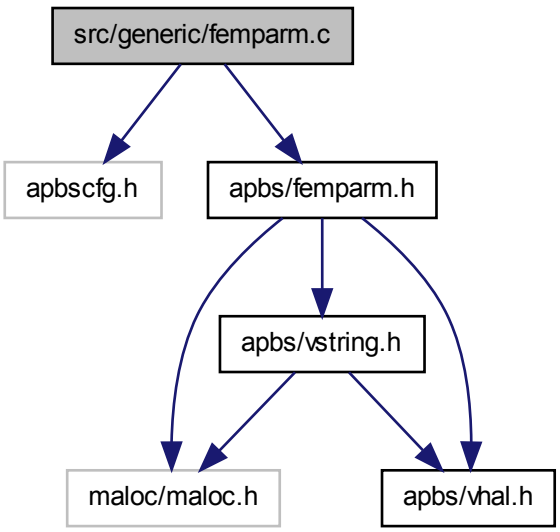
10.53 src/generic/femparm.c File Reference

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 1552 2010-02-10 17:46:27Z yhuang01

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
* 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 Washington University in St. Louis 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 src/generic/femparm.c

```
00001
00050 #include "apbscfg.h"
00051 #include "apbs/femparm.h"
00052
00053 VEMBED(rcsid="$Id: femparm.c 1552 2010-02-10 17:46:27Z yhuang01 $")
```

```
00054
00055 #if !defined(VINLINE_MGPARM)
00056
00057 #endif /* if !defined(VINLINE_MGPARM) */
00058
00059 VPUBLIC FEMparm* FEMparm_ctor(FEMparm_CalcType type) {
00060
00061     /* Set up the structure */
00062     FEMparm *thee = VNULL;
00063     thee = Vmem_malloc(VNULL, 1, sizeof(FEMparm));
00064     VASSERT( thee != VNULL);
00065     VASSERT( FEMparm_ctor2(thee, type) );
00066
00067     return thee;
00068 }
00069
00070 VPUBLIC int FEMparm_ctor2(FEMparm *thee, FEMparm_CalcType type) {
00071
00072     if (thee == VNULL) return 0;
00073
00074     thee->parsed = 0;
00075     thee->type = type;
00076     thee->settype = 1;
00077
00078     thee->setglen = 0;
00079     thee->setetol = 0;
00080     thee->setekey = 0;
00081     thee->setakeyPRE = 0;
00082     thee->setakeySOLVE = 0;
00083     thee->settargetNum = 0;
00084     thee->settargetRes = 0;
00085     thee->setmaxsolve = 0;
00086     thee->setmaxvert = 0;
00087  thee->useMesh = 0;
00088
00089     return 1;
00090 }
00091
00092 VPUBLIC void FEMparm_copy(
00093         FEMparm *thee,
00094         FEMparm *source
00095         ) {
00096
00097  int i;
00098
00099  thee->parsed = source->parsed;
00100  thee->type = source->type;
00101  thee->settype = source->settype;
00102  for (i=0; i<3; i++) thee->glen[i] = source->glen[i];
00103  thee->setglen = source->setglen;
00104  thee->etol = source->etol;
00105  thee->setetol = source->setetol;
00106  thee->ekey = source->ekey;
00107  thee->setekey = source->setekey;
00108  thee->akeyPRE = source->akeyPRE;
00109  thee->setakeyPRE = source->setakeyPRE;
00110  thee->akeySOLVE = source->akeySOLVE;
```

```
00111  thee->setakeySOLVE = source->setakeySOLVE;
00112  thee->targetNum = source->targetNum;
00113  thee->settargetNum = source->settargetNum;
00114  thee->targetRes = source->targetRes;
00115  thee->settargetRes = source->settargetRes;
00116  thee->maxsolve = source->maxsolve;
00117  thee->setmaxsolve = source->setmaxsolve;
00118  thee->maxvert = source->maxvert;
00119  thee->setmaxvert = source->setmaxvert;
00120  thee->pkey = source->pkey;
00121  thee->useMesh = source->useMesh;
00122  thee->meshID = source->meshID;
00123 }
00124
00125 VPUBLIC void FEMparm_dtor(FEMparm **thee) {
00126     if ((*thee) != VNULL) {
00127         FEMparm_dtor2(*thee);
00128         Vmem_free(VNULL, 1, sizeof(FEMparm), (void **)thee);
00129         (*thee) = VNULL;
00130     }
00131 }
00132
00133 VPUBLIC void FEMparm_dtor2(FEMparm *thee) { ; }
00134
00135 VPUBLIC int FEMparm_check(FEMparm *thee) {
00136
00137     int rc;
00138     rc = 1;
00139
00140     if (!thee->parsed) {
00141         Vnm_print(2, "FEMparm_check:  not filled!\n");
00142         return 0;
00143     }
00144     if (!thee->settype) {
00145         Vnm_print(2, "FEMparm_check:  type not set!\n");
00146         rc = 0;
00147     }
00148     if (!thee->setglen) {
00149         Vnm_print(2, "FEMparm_check:  glen not set!\n");
00150         rc = 0;
00151     }
00152     if (!thee->setetol) {
00153         Vnm_print(2, "FEMparm_check:  etol not set!\n");
00154         rc = 0;
00155     }
00156     if (!thee->setekey) {
00157         Vnm_print(2, "FEMparm_check:  ekey not set!\n");
00158         rc = 0;
00159     }
00160     if (!thee->setakeyPRE) {
00161         Vnm_print(2, "FEMparm_check:  akeyPRE not set!\n");
00162         rc = 0;
00163     }
00164     if (!thee->setakeySOLVE) {
00165         Vnm_print(2, "FEMparm_check:  akeySOLVE not set!\n");
00166         rc = 0;
00167     }
```

```
00168     if (!thee->settargetNum) {
00169         Vnm_print(2, "FEMparm_check:  targetNum not set!\n");
00170         rc = 0;
00171     }
00172     if (!thee->settargetRes) {
00173         Vnm_print(2, "FEMparm_check:  targetRes not set!\n");
00174         rc = 0;
00175     }
00176     if (!thee->setmaxsolve) {
00177         Vnm_print(2, "FEMparm_check:  maxsolve not set!\n");
00178         rc = 0;
00179     }
00180     if (!thee->setmaxvert) {
00181         Vnm_print(2, "FEMparm_check:  maxvert not set!\n");
00182         rc = 0;
00183     }
00184
00185     return rc;
00186 }
00187
00188 VPRIVATE Vrc_Codes FEMparm_parseDOMAINLENGTH(FEMparm *thee, Vio *sock) {
00189
00190     int i;
00191     double tf;
00192     char tok[VMAX_BUFSIZE];
00193
00194     for (i=0; i<3; i++) {
00195         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00196         if (sscanf(tok, "%lf", &tf) == 0) {
00197             Vnm_print(2, "parseFE:  Read non-double (%s) while parsing \
00198 DOMAINLENGTH keyword!\n", tok);
00199             return VRC_FAILURE;
00200         }
00201         thee->glen[i] = tf;
00202     }
00203     thee->setglen = 1;
00204     return VRC_SUCCESS;
00205 VERROR1:
00206     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00207     return VRC_FAILURE;
00208
00209 }
00210
00211 VPRIVATE Vrc_Codes FEMparm_parseETOL(FEMparm *thee, Vio *sock) {
00212
00213     double tf;
00214     char tok[VMAX_BUFSIZE];
00215
00216     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00217     if (sscanf(tok, "%lf", &tf) == 0) {
00218         Vnm_print(2, "parseFE:  Read non-double (%s) while parsing \
00219 ETOL keyword!\n", tok);
00220         return VRC_FAILURE;
00221     }
00222     thee->etol = tf;
00223     thee->setetol = 1;
00224     return VRC_SUCCESS;
```

```
00225 VERROR1:
00226     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00227     return VRC_FAILURE;
00228
00229
00230 }
00231
00232 VPRIVATE Vrc_Codes FEMparm_parseEKEY(FEMparm *thee, Vio *sock) {
00233
00234     char tok[VMAX_BUFSIZE];
00235     Vrc_Codes vrc = VRC_FAILURE;
00236
00237     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00238     if (Vstring_strcasecmp(tok, "simp") == 0) {
00239         thee->ekey = FET_SIMP;
00240         thee->setekey = 1;
00241         vrc = VRC_SUCCESS;
00242     } else if (Vstring_strcasecmp(tok, "glob") == 0) {
00243         thee->ekey = FET_GLOB;
00244         thee->setekey = 1;
00245         vrc = VRC_SUCCESS;
00246     } else if (Vstring_strcasecmp(tok, "frac") == 0) {
00247         thee->ekey = FET_FRAC;
00248         thee->setekey = 1;
00249         vrc = VRC_SUCCESS;
00250     } else {
00251         Vnm_print(2, "parseFE:  undefined value (%s) for ekey!\n", tok);
00252         vrc = VRC_FAILURE;
00253     }
00254
00255     return vrc;
00256 VERROR1:
00257     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00258     return VRC_FAILURE;
00259
00260 }
00261
00262 VPRIVATE Vrc_Codes FEMparm_parseAKEYPRE(FEMparm *thee, Vio *sock) {
00263
00264     char tok[VMAX_BUFSIZE];
00265     Vrc_Codes vrc = VRC_FAILURE;
00266
00267     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00268     if (Vstring_strcasecmp(tok, "unif") == 0) {
00269         thee->akeyPRE = FRT_UNIF;
00270         thee->setakeyPRE = 1;
00271         vrc =  VRC_SUCCESS;
00272     } else if (Vstring_strcasecmp(tok, "geom") == 0) {
00273         thee->akeyPRE = FRT_GEOM;
00274         thee->setakeyPRE = 1;
00275         vrc =  VRC_SUCCESS;
00276     } else {
00277         Vnm_print(2, "parseFE:  undefined value (%s) for akeyPRE!\n", tok);
00278         vrc =  VRC_FAILURE;
00279     }
00280
00281     return vrc;
```

```
00282
00283 VERROR1:
00284     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00285     return VRC_FAILURE;
00286
00287 }
00288
00289 VPRIVATE Vrc_Codes FEMparm_parseAKEYSOLVE(FEMparm *thee, Vio *sock) {
00290
00291     char tok[VMAX_BUFSIZE];
00292     Vrc_Codes vrc = VRC_FAILURE;
00293
00294     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00295     if (Vstring_strcasecmp(tok, "resi") == 0) {
00296         thee->akeySOLVE = FRT_RESI;
00297         thee->setakeySOLVE = 1;
00298         vrc =  VRC_SUCCESS;
00299     } else if (Vstring_strcasecmp(tok, "dual") == 0) {
00300         thee->akeySOLVE = FRT_DUAL;
00301         thee->setakeySOLVE = 1;
00302         vrc =  VRC_SUCCESS;
00303     } else if (Vstring_strcasecmp(tok, "loca") == 0) {
00304         thee->akeySOLVE = FRT_LOCA;
00305         thee->setakeySOLVE = 1;
00306         vrc =  VRC_SUCCESS;
00307     } else {
00308         Vnm_print(2, "parseFE:  undefined value (%s) for akeyPRE!\n", tok);
00309         vrc =  VRC_FAILURE;
00310     }
00311
00312     return vrc;
00313 VERROR1:
00314     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00315     return VRC_SUCCESS;
00316
00317 }
00318
00319 VPRIVATE Vrc_Codes FEMparm_parseTARGETNUM(FEMparm *thee, Vio *sock) {
00320
00321     char tok[VMAX_BUFSIZE];
00322     int ti;
00323
00324     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00325     if (sscanf(tok, "%d", &ti) == 0) {
00326         Vnm_print(2, "parseFE:  read non-int (%s) for targetNum!\n", tok);
00327         return VRC_FAILURE;
00328     }
00329     thee->targetNum = ti;
00330     thee->settargetNum = 1;
00331     return VRC_SUCCESS;
00332 VERROR1:
00333     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00334     return VRC_FAILURE;
00335
00336 }
00337
00338 VPRIVATE Vrc_Codes FEMparm_parseTARGETRES(FEMparm *thee, Vio *sock) {
```

```
00339
00340     char tok[VMAX_BUFSIZE];
00341     double tf;
00342
00343     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00344     if (sscanf(tok, "%lf", &tf) == 0) {
00345         Vnm_print(2, "parseFE:  read non-double (%s) for targetNum!\n",
00346           tok);
00347         return VRC_FAILURE;
00348     }
00349     thee->targetRes = tf;
00350     thee->settargetRes = 1;
00351     return VRC_SUCCESS;
00352 VERROR1:
00353     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00354     return VRC_FAILURE;
00355
00356 }
00357
00358 VPRIVATE Vrc_Codes FEMparm_parseMAXSOLVE(FEMparm *thee, Vio *sock) {
00359
00360     char tok[VMAX_BUFSIZE];
00361     int ti;
00362
00363     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00364     if (sscanf(tok, "%d", &ti) == 0) {
00365         Vnm_print(2, "parseFE:  read non-int (%s) for maxsolve!\n", tok);
00366         return VRC_FAILURE;
00367     }
00368     thee->maxsolve = ti;
00369     thee->setmaxsolve = 1;
00370     return VRC_SUCCESS;
00371 VERROR1:
00372     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00373     return VRC_FAILURE;
00374
00375 }
00376
00377 VPRIVATE Vrc_Codes FEMparm_parseMAXVERT(FEMparm *thee, Vio *sock) {
00378
00379     char tok[VMAX_BUFSIZE];
00380     int ti;
00381
00382     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00383     if (sscanf(tok, "%d", &ti) == 0) {
00384         Vnm_print(2, "parseFE:  read non-int (%s) for maxvert!\n", tok);
00385         return VRC_FAILURE;
00386     }
00387     thee->maxvert = ti;
00388     thee->setmaxvert = 1;
00389     return VRC_SUCCESS;
00390
00391 VERROR1:
00392     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00393     return VRC_FAILURE;
00394
00395 }
```

```
00396
00397 VPRIVATE Vrc_Codes FEMparm_parseUSEMESH(FEMparm *thee, Vio *sock) {
00398     char tok[VMAX_BUFSIZE];
00399     int ti;
00400
00401     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00402  if (sscanf(tok, "%d", &ti) == 0) {
00403         Vnm_print(2, "parseFE:  read non-int (%s) for usemesh!\n", tok);
00404         return VRC_FAILURE;
00405     }
00406  thee->useMesh = 1;
00407  thee->meshID = ti;
00408
00409  return VRC_SUCCESS;
00410
00411 VERROR1:
00412  Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00413  return VRC_FAILURE;
00414 }
00415
00416
00417 VPUBLIC Vrc_Codes FEMparm_parseToken(FEMparm *thee, char tok[VMAX_BUFSIZE],
00418   Vio *sock) {
00419
00420     int i, ti;
00421     double tf;
00422
00423     if (thee == VNULL) {
00424         Vnm_print(2, "parseFE:  got NULL thee!\n");
00425         return VRC_FAILURE;
00426     }
00427
00428     if (sock == VNULL) {
00429         Vnm_print(2, "parseFE:  got NULL socket!\n");
00430         return VRC_FAILURE;
00431     }
00432
00433     if (Vstring_strcasecmp(tok, "domainLength") == 0) {
00434         return FEMparm_parseDOMAINLENGTH(thee, sock);
00435     } else if (Vstring_strcasecmp(tok, "etol") == 0) {
00436         return FEMparm_parseETOL(thee, sock);
00437     } else if (Vstring_strcasecmp(tok, "ekey") == 0) {
00438         return FEMparm_parseEKEY(thee, sock);
00439     } else if (Vstring_strcasecmp(tok, "akeyPRE") == 0) {
00440         return FEMparm_parseAKEYPRE(thee, sock);
00441     } else if (Vstring_strcasecmp(tok, "akeySOLVE") == 0) {
00442         return FEMparm_parseAKEYSOLVE(thee, sock);
00443     } else if (Vstring_strcasecmp(tok, "targetNum") == 0) {
00444         return FEMparm_parseTARGETNUM(thee, sock);
00445     } else if (Vstring_strcasecmp(tok, "targetRes") == 0) {
00446         return FEMparm_parseTARGETRES(thee, sock);
00447     } else if (Vstring_strcasecmp(tok, "maxsolve") == 0) {
00448         return FEMparm_parseMAXSOLVE(thee, sock);
00449     } else if (Vstring_strcasecmp(tok, "maxvert") == 0) {
00450         return FEMparm_parseMAXVERT(thee, sock);
00451     } else if (Vstring_strcasecmp(tok, "usemesh") == 0) {
00452         return FEMparm_parseUSEMESH(thee, sock);
```

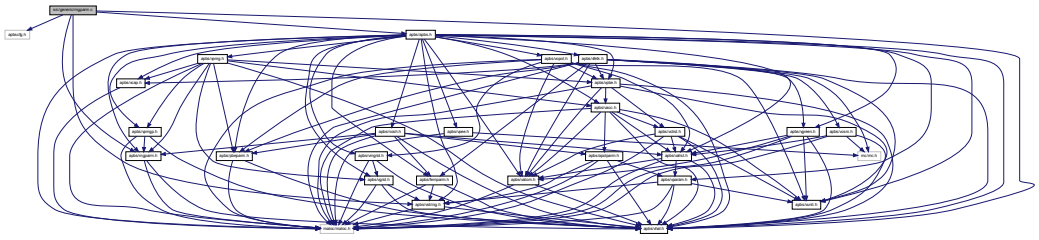
```
00453     }
00454
00455     return VRC_WARNING;
00456
00457 }
```

10.55 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)

  Get center y-coordinate.

- VPUBLIC double MGparm_getCenterZ (MGparm *thee)

  Get center z-coordinate.

- VPUBLIC int MGparm_getNx (MGparm *thee)

  Get number of grid points in x direction.

- VPUBLIC int MGparm_getNy (MGparm *thee)

  Get number of grid points in y direction.

- VPUBLIC int MGparm_getNz (MGparm *thee)

  Get number of grid points in z direction.

- VPUBLIC double MGparm_getHx (MGparm *thee)

  Get grid spacing in x direction (Å)

- VPUBLIC double MGparm_getHy (MGparm *thee)

  Get grid spacing in y direction (Å)

- VPUBLIC double MGparm_getHz (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 1552 2010-02-10 17:46:27Z yhuang01

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
```

```
 *
 * Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
 * 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 Washington University in St. Louis 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 src/generic/mgparm.c

```
00001
00049 #include "apbscfg.h"
00050 #include "apbs/apbs.h"
00051 #include "apbs/vhal.h"
00052 #include "apbs/mgparm.h"
00053 #include "apbs/vstring.h"
00054
00055 VEMBED(rcsid="$Id: mgparm.c 1552 2010-02-10 17:46:27Z yhuang01 $")
00056
00057 #if !defined(VINLINE_MGPARM)
00058
00059 #endif /* if !defined(VINLINE_MGPARM) */
00060
00061 VPUBLIC void MGparm_setCenterX(MGparm *thee, double x) {
00062     VASSERT(thee != VNULL);
00063     thee->center[0] = x;
00064 }
00065 VPUBLIC void MGparm_setCenterY(MGparm *thee, double y) {
```

```
00066     VASSERT(thee != VNULL);
00067     thee->center[1] = y;
00068 }
00069 VPUBLIC void MGparm_setCenterZ(MGparm *thee, double z) {
00070     VASSERT(thee != VNULL);
00071     thee->center[2] = z;
00072 }
00073 VPUBLIC double MGparm_getCenterX(MGparm *thee) {
00074     VASSERT(thee != VNULL);
00075     return thee->center[0];
00076 }
00077 VPUBLIC double MGparm_getCenterY(MGparm *thee) {
00078     VASSERT(thee != VNULL);
00079     return thee->center[1];
00080 }
00081 VPUBLIC double MGparm_getCenterZ(MGparm *thee) {
00082     VASSERT(thee != VNULL);
00083     return thee->center[2];
00084 }
00085 VPUBLIC int MGparm_getNx(MGparm *thee) {
00086     VASSERT(thee != VNULL);
00087     return thee->dime[0];
00088 }
00089 VPUBLIC int MGparm_getNy(MGparm *thee) {
00090     VASSERT(thee != VNULL);
00091     return thee->dime[1];
00092 }
00093 VPUBLIC int MGparm_getNz(MGparm *thee) {
00094     VASSERT(thee != VNULL);
00095     return thee->dime[2];
00096 }
00097 VPUBLIC double MGparm_getHx(MGparm *thee) {
00098     VASSERT(thee != VNULL);
00099     return thee->grid[0];
00100 }
00101 VPUBLIC double MGparm_getHy(MGparm *thee) {
00102     VASSERT(thee != VNULL);
00103     return thee->grid[1];
00104 }
00105 VPUBLIC double MGparm_getHz(MGparm *thee) {
00106     VASSERT(thee != VNULL);
00107     return thee->grid[2];
00108 }
00109
00110 VPUBLIC MGparm* MGparm_ctor(MGparm_CalcType type) {
00111
00112     /* Set up the structure */
00113     MGparm *thee = VNULL;
00114     thee = Vmem_malloc(VNULL, 1, sizeof(MGparm));
00115     VASSERT( thee != VNULL);
00116     VASSERT( MGparm_ctor2(thee, type) == VRC_SUCCESS );
00117
00118     return thee;
00119 }
00120
00121 VPUBLIC Vrc_Codes MGparm_ctor2(MGparm *thee, MGparm_CalcType type) {
00122
```

```
00123     int i;
00124
00125     if (thee == VNULL) return VRC_FAILURE;
00126
00127     for (i=0; i<3; i++) {
00128         thee->dime[i] = -1;
00129         thee->pdime[i] = 1;
00130     }
00131
00132     thee->parsed = 0;
00133     thee->type = type;
00134
00135     /* *** GENERIC PARAMETERS *** */
00136     thee->setdime = 0;
00137     thee->setchgm = 0;
00138
00139     /* *** TYPE 0 PARAMETERS *** */
00140     thee->nlev = VMGNLEV;
00141     thee->setnlev = 1;
00142     thee->etol = 1.0e-6;
00143     thee->setetol = 0;
00144     thee->setgrid = 0;
00145     thee->setglen = 0;
00146     thee->setgcent = 0;
00147
00148     /* *** TYPE 1 & 2 PARAMETERS *** */
00149     thee->setcglen = 0;
00150     thee->setfglen = 0;
00151     thee->setcgcent = 0;
00152     thee->setfgcent = 0;
00153
00154     /* *** TYPE 2 PARAMETERS *** */
00155     thee->setpdime = 0;
00156     thee->setrank = 0;
00157     thee->setsize = 0;
00158     thee->setofrac = 0;
00159     for (i=0; i<6; i++) thee->partDisjOwnSide[i] = 0;
00160  thee->setasync = 0;
00161
00162     /* *** Default parameters for TINKER *** */
00163     thee->chgs = VCM_CHARGE;
00164
00165  thee->useAqua = 0;
00166  thee->setUseAqua = 0;
00167
00168     return VRC_SUCCESS;
00169 }
00170
00171 VPUBLIC void MGparm_dtor(MGparm **thee) {
00172     if ((*thee) != VNULL) {
00173         MGparm_dtor2(*thee);
00174         Vmem_free(VNULL, 1, sizeof(MGparm), (void **)thee);
00175         (*thee) = VNULL;
00176     }
00177 }
00178
00179 VPUBLIC void MGparm_dtor2(MGparm *thee) { ; }
```

```
00180
00181 VPUBLIC Vrc_Codes MGparm_check(MGparm *thee) {
00182
00183     Vrc_Codes rc;
00184  int i, tdime[3], ti, tnlev[3], nlev;
00185
00186     rc = VRC_SUCCESS;
00187
00188  Vnm_print(0, "MGparm_check:  checking MGparm object of type %d.\n",
00189      thee->type);
00190
00191     /* Check to see if we were even filled... */
00192     if (!thee->parsed) {
00193         Vnm_print(2, "MGparm_check:  not filled!\n");
00194         return VRC_FAILURE;
00195     }
00196
00197     /* Check generic settings */
00198     if (!thee->setdime) {
00199         Vnm_print(2, "MGparm_check:  DIME not set!\n");
00200         rc = VRC_FAILURE;
00201     }
00202     if (!thee->setchgm) {
00203         Vnm_print(2, "MGparm_check: CHGM not set!\n");
00204         return VRC_FAILURE;
00205     }
00206
00207
00208     /* Check sequential manual & dummy settings */
00209     if ((thee->type == MCT_MANUAL) || (thee->type == MCT_DUMMY)) {
00210         if ((!thee->setgrid) && (!thee->setglen)) {
00211             Vnm_print(2, "MGparm_check:  Neither GRID nor GLEN set!\n");
00212             rc = VRC_FAILURE;
00213         }
00214         if ((thee->setgrid) && (thee->setglen)) {
00215             Vnm_print(2, "MGparm_check:  Both GRID and GLEN set!\n");
00216             rc = VRC_FAILURE;
00217         }
00218         if (!thee->setgcent) {
00219             Vnm_print(2, "MGparm_check:  GCENT not set!\n");
00220             rc = VRC_FAILURE;
00221         }
00222     }
00223
00224     /* Check sequential and parallel automatic focusing settings */
00225     if ((thee->type == MCT_AUTO) || (thee->type == MCT_PARALLEL)) {
00226         if (!thee->setcglen) {
00227             Vnm_print(2, "MGparm_check:  CGLEN not set!\n");
00228             rc = VRC_FAILURE;
00229         }
00230         if (!thee->setfglen) {
00231             Vnm_print(2, "MGparm_check:  FGLEN not set!\n");
00232             rc = VRC_FAILURE;
00233         }
00234         if (!thee->setcgcent) {
00235             Vnm_print(2, "MGparm_check:  CGCENT not set!\n");
00236             rc = VRC_FAILURE;
```

```
00237         }
00238         if (!thee->setfgcent) {
00239             Vnm_print(2, "MGparm_check:  FGCENT not set!\n");
00240             rc = VRC_FAILURE;
00241         }
00242     }
00243
00244     /* Check parallel automatic focusing settings */
00245     if (thee->type == MCT_PARALLEL) {
00246         if (!thee->setpdime) {
00247             Vnm_print(2, "MGparm_check:  PDIME not set!\n");
00248             rc = VRC_FAILURE;
00249         }
00250         if (!thee->setrank) {
00251             Vnm_print(2, "MGparm_check:  PROC_RANK not set!\n");
00252             rc = VRC_FAILURE;
00253         }
00254         if (!thee->setsize) {
00255             Vnm_print(2, "MGparm_check:  PROC_SIZE not set!\n");
00256             rc = VRC_FAILURE;
00257         }
00258         if (!thee->setofrac) {
00259             Vnm_print(2, "MGparm_check:  OFRAC not set!\n");
00260             rc = VRC_FAILURE;
00261         }
00262     }
00263
00264     /* Perform a sanity check on nlev and dime, resetting values as necessary */
00265     if (rc == 1) {
00266  /* Calculate the actual number of grid points and nlev to satisfy the
00267   * formula:  n = c * 2^(l+1) + 1, where n is the number of grid points,
00268   * c is an integer, and l is the number of levels */
00269  if (thee->type != MCT_DUMMY) {
00270   for (i=0; i<3; i++) {
00271             /* See if the user picked a reasonable value, if not then fix it */
00272             ti = thee->dime[i] - 1;
00273             if (ti == VPOW(2, (thee->nlev+1))) {
00274                 tnlev[i] = thee->nlev;
00275                 tdime[i] = thee->dime[i];
00276             } else {
00277                 tdime[i] = thee->dime[i];
00278                 ti = tdime[i] - 1;
00279                 tnlev[i] = 0;
00280                 /* Find the maximum number of times this dimension can be
00281                  * divided by two */
00282                 while (VEVEN(ti)) {
00283                     (tnlev[i])++;
00284                     ti = (int)ceil(0.5*ti);
00285                 }
00286                 (tnlev[i])--;
00287                 /* We'd like to have at least VMGNLEV levels in the multigrid
00288                  * hierarchy.  This means that the dimension needs to be
00289                  * c*2^VMGNLEV + 1, where c is an integer. */
00290                 if ((tdime[i] > 65) && (tnlev[i] < VMGNLEV)) {
00291                     Vnm_print(2, "NOsh:  Bad dime[%d]  = %d (%d nlev)!\n",
00292                       i, tdime[i], tnlev[i]);
00293                     ti = (int)(tdime[i]/VPOW(2.,(VMGNLEV+1)));
```

```
00294                     if (ti < 1) ti = 1;
00295                     tdime[i] = ti*(int)(VPOW(2.,(VMGNLEV+1))) + 1;
00296                     tnlev[i] = 4;
00297                     Vnm_print(2, "NOsh:  Reset dime[%d] to %d and (nlev = %d).\n"
      , i, tdime[i], VMGNLEV);
00298                   }
00299               }
00300           }
00301  } else { /* We are a dummy calculation, but we still need positive numbers of po
      ints */
00302   for (i=0; i<3; i++) {
00303       tnlev[i] = thee->nlev;
00304    tdime[i] = thee->dime[i];
00305     if (thee->dime[i] <= 0) {
00306      Vnm_print(2, "NOsh:  Resetting dime[%d] from %d to 3.\n", i, thee->dime[i]);

00307      thee->dime[i] = 3;
00308    }
00309   }
00310  }
00311
00312  /* The actual number of levels we'll be using is the smallest number of
00313           * possible levels in any dimensions */
00314         nlev = VMIN2(tnlev[0], tnlev[1]);
00315         nlev = VMIN2(nlev, tnlev[2]);
00316         /* Set the number of levels and dimensions */
00317         Vnm_print(0, "NOsh:  nlev = %d, dime = (%d, %d, %d)\n", nlev, tdime[0],
00318           tdime[1], tdime[2]);
00319         thee->nlev = nlev;
00320   if (thee->nlev <= 0) {
00321    Vnm_print(2, "MGparm_check:  illegal nlev (%d); check your grid dimensions!\n"
      , thee->nlev);
00322    rc = VRC_FAILURE;
00323   }
00324   if (thee->nlev < 2) {
00325    Vnm_print(2, "MGparm_check:  you're using a very small nlev (%d) and therefore
      \n", thee->nlev);
00326    Vnm_print(2, "MGparm_check:  will not get the optimal performance of the multi
      grid\n");
00327    Vnm_print(2, "MGparm_check:  algorithm.  Please check your grid dimensions.\n"
      );
00328   }
00329         for (i=0; i<3; i++) thee->dime[i] = tdime[i];
00330     }
00331
00332  if (!thee->setUseAqua) thee->useAqua = 0;
00333
00334     return rc;
00335 }
00336
00337 VPUBLIC void MGparm_copy(MGparm *thee, MGparm *parm) {
00338
00339     int i;
00340
00341     VASSERT(thee != VNULL);
00342     VASSERT(parm != VNULL);
00343
```

```
00344
00345     thee->type = parm->type;
00346     thee->parsed = parm->parsed;
00347
00348     /* *** GENERIC PARAMETERS *** */
00349     for (i=0; i<3; i++) thee->dime[i] = parm->dime[i];
00350     thee->setdime = parm->setdime;
00351     thee->chgm = parm->chgm;
00352     thee->setchgm = parm->setchgm;
00353     thee->chgs = parm->chgs;
00354
00355     /* *** TYPE 0 PARMS *** */
00356     thee->nlev = parm->nlev;
00357     thee->setnlev = parm->setnlev;
00358     thee->etol = parm->etol;
00359     thee->setetol = parm->setetol;
00360     for (i=0; i<3; i++) thee->grid[i] = parm->grid[i];
00361     thee->setgrid = parm->setgrid;
00362     for (i=0; i<3; i++) thee->glen[i] = parm->glen[i];
00363     thee->setglen = parm->setglen;
00364     thee->cmeth = parm->cmeth;
00365     for (i=0; i<3; i++) thee->center[i] = parm->center[i];
00366     thee->setgcent = parm->setgcent;
00367     thee->centmol = parm->centmol;
00368
00369     /* *** TYPE 1 & 2 PARMS *** */
00370     for (i=0; i<3; i++) thee->cglen[i] = parm->cglen[i];
00371     thee->setcglen = parm->setcglen;
00372     for (i=0; i<3; i++) thee->fglen[i] = parm->fglen[i];
00373     thee->setfglen = parm->setfglen;
00374     thee->ccmeth = parm->ccmeth;
00375     for (i=0; i<3; i++) thee->ccenter[i] = parm->ccenter[i];
00376     thee->setcgcent = parm->setcgcent;
00377     thee->ccentmol = parm->ccentmol;
00378     thee->fcmeth = parm->fcmeth;
00379     for (i=0; i<3; i++) thee->fcenter[i] = parm->fcenter[i];
00380     thee->setfgcent = parm->setfgcent;
00381     thee->fcentmol = parm->fcentmol;
00382
00383     /* *** TYPE 2 PARMS *** */
00384     for (i=0; i<3; i++)
00385       thee->partDisjCenter[i] = parm->partDisjCenter[i];
00386     for (i=0; i<3; i++)
00387       thee->partDisjLength[i] = parm->partDisjLength[i];
00388     for (i=0; i<6; i++)
00389       thee->partDisjOwnSide[i] = parm->partDisjOwnSide[i];
00390     for (i=0; i<3; i++) thee->pdime[i] = parm->pdime[i];
00391     thee->setpdime = parm->setpdime;
00392     thee->proc_rank = parm->proc_rank;
00393     thee->setrank = parm->setrank;
00394     thee->proc_size = parm->proc_size;
00395     thee->setsize = parm->setsize;
00396     thee->ofrac = parm->ofrac;
00397     thee->setofrac = parm->setofrac;
00398   thee->setasync = parm->setasync;
00399   thee->async = parm->async;
00400
```

```
00401  thee->nonlintype = parm->nonlintype;
00402  thee->setnonlintype = parm->setnonlintype;
00403
00404  thee->method = parm->method;
00405  thee->method = parm->method;
00406
00407  thee->useAqua = parm->useAqua;
00408  thee->setUseAqua = parm->setUseAqua;
00409 }
00410
00411 VPRIVATE Vrc_Codes MGparm_parseDIME(MGparm *thee, Vio *sock) {
00412
00413     char tok[VMAX_BUFSIZE];
00414     int ti;
00415
00416     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00417     if (sscanf(tok, "%d", &ti) == 0){
00418         Vnm_print(2, "parseMG:  Read non-integer (%s) while parsing DIME \
00419 keyword!\n", tok);
00420         return VRC_WARNING;
00421     } else thee->dime[0] = ti;
00422     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00423     if (sscanf(tok, "%d", &ti) == 0) {
00424         Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing DIME \
00425 keyword!\n", tok);
00426         return VRC_WARNING;
00427     } else thee->dime[1] = ti;
00428     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00429     if (sscanf(tok, "%d", &ti) == 0) {
00430         Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing DIME \
00431 keyword!\n", tok);
00432         return VRC_WARNING;
00433     } else thee->dime[2] = ti;
00434     thee->setdime = 1;
00435     return VRC_SUCCESS;
00436
00437     VERROR1:
00438         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00439         return VRC_WARNING;
00440 }
00441
00442 VPRIVATE Vrc_Codes MGparm_parseCHGM(MGparm *thee, Vio *sock) {
00443
00444     char tok[VMAX_BUFSIZE];
00445     Vchrg_Meth ti;
00446
00447     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00448     if (sscanf(tok, "%d", &ti) == 1) {
00449         thee->chgm = ti;
00450         thee->setchgm = 1;
00451         Vnm_print(2, "NOsh:  Warning -- parsed deprecated statment \"chgm %d\".\n
      ", ti);
00452         Vnm_print(2, "NOsh:  Please use \"chgm ");
00453         switch (thee->chgm) {
00454             case VCM_TRIL:
00455                 Vnm_print(2, "spl0");
00456                 break;
```

```
00457             case VCM_BSPL2:
00458                 Vnm_print(2, "spl2");
00459                 break;
00460             case VCM_BSPL4:
00461                 Vnm_print(2, "spl4");
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, "spl0") == 0) {
00470         thee->chgm = VCM_TRIL;
00471         thee->setchgm = 1;
00472         return VRC_SUCCESS;
00473     } else if (Vstring_strcasecmp(tok, "spl2") == 0) {
00474         thee->chgm = VCM_BSPL2;
00475         thee->setchgm = 1;
00476         return VRC_SUCCESS;
00477     } else if (Vstring_strcasecmp(tok, "spl4") == 0) {
00478         thee->chgm = VCM_BSPL4;
00479         thee->setchgm = 1;
00480         return VRC_SUCCESS;
00481     } else {
00482         Vnm_print(2, "NOsh:  Unrecognized parameter (%s) when parsing \
00483 chgm!\n", tok);
00484         return VRC_WARNING;
00485     }
00486     return VRC_WARNING;
00487 
00488     VERROR1:
00489         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00490         return VRC_WARNING;
00491 }
00492 
00493 VPRIVATE Vrc_Codes MGparm_parseNLEV(MGparm *thee, Vio *sock) {
00494 
00495     char tok[VMAX_BUFSIZE];
00496     int ti;
00497 
00498     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00499     if (sscanf(tok, "%d", &ti) == 0) {
00500         Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing NLEV \
00501 keyword!\n", tok);
00502         return VRC_WARNING;
00503     } else thee->nlev = ti;
00504     thee->setnlev = 1;
00505     return VRC_SUCCESS;
00506 
00507     VERROR1:
00508         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00509         return VRC_WARNING;
00510 }
00511 
00512 VPRIVATE Vrc_Codes MGparm_parseETOL(MGparm *thee, Vio *sock) {
00513 
```

```
00514     char tok[VMAX_BUFSIZE];
00515     double tf;
00516
00517     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00518     if (sscanf(tok, "%lf", &tf) == 0) {
00519         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing etol \
00520 keyword!\n", tok);
00521         return VRC_WARNING;
00522     } else if (tf <= 0.0) {
00523         Vnm_print(2, "parseMG:  etol must be greater than 0!\n");
00524         return VRC_WARNING;
00525     } else thee->etol = tf;
00526     thee->setetol = 1;
00527     return VRC_SUCCESS;
00528
00529     VERROR1:
00530         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00531         return VRC_WARNING;
00532 }
00533
00534
00535 VPRIVATE Vrc_Codes MGparm_parseGRID(MGparm *thee, Vio *sock) {
00536
00537     char tok[VMAX_BUFSIZE];
00538     double tf;
00539
00540     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00541     if (sscanf(tok, "%lf", &tf) == 0) {
00542         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GRID \
00543 keyword!\n", tok);
00544         return VRC_WARNING;
00545     } else thee->grid[0] = tf;
00546     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00547     if (sscanf(tok, "%lf", &tf) == 0) {
00548         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GRID \
00549 keyword!\n", tok);
00550         return VRC_WARNING;
00551     } else thee->grid[1] = tf;
00552     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00553     if (sscanf(tok, "%lf", &tf) == 0) {
00554         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GRID \
00555 keyword!\n", tok);
00556         return VRC_WARNING;
00557     } else thee->grid[2] = tf;
00558     thee->setgrid = 1;
00559     return VRC_SUCCESS;
00560
00561     VERROR1:
00562         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00563         return VRC_WARNING;
00564 }
00565
00566 VPRIVATE Vrc_Codes MGparm_parseGLEN(MGparm *thee, Vio *sock) {
00567
00568     char tok[VMAX_BUFSIZE];
00569     double tf;
00570
```

```
00571     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00572     if (sscanf(tok, "%lf", &tf) == 0) {
00573         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GLEN \
00574 keyword!\n", tok);
00575         return VRC_WARNING;
00576     } else thee->glen[0] = tf;
00577     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00578     if (sscanf(tok, "%lf", &tf) == 0) {
00579         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GLEN \
00580 keyword!\n", tok);
00581         return VRC_WARNING;
00582     } else thee->glen[1] = tf;
00583     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00584     if (sscanf(tok, "%lf", &tf) == 0) {
00585         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GLEN \
00586 keyword!\n", tok);
00587         return VRC_WARNING;
00588     } else thee->glen[2] = tf;
00589     thee->setglen = 1;
00590     return VRC_SUCCESS;
00591
00592     VERROR1:
00593         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00594         return VRC_WARNING;
00595 }
00596
00597 VPRIVATE Vrc_Codes MGparm_parseGAMMA(MGparm *thee, Vio *sock) {
00598
00599     char tok[VMAX_BUFSIZE];
00600
00601     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00602  Vnm_print(2, "parseMG:  GAMMA keyword deprecated!\n");
00603  Vnm_print(2, "parseMG:  If you are using PyMOL or VMD and still seeing this mess
      age,\n");
00604  Vnm_print(2, "parseMG:  please contact the developers of those programs regardin
      g this message.\n");
00605  return VRC_SUCCESS;
00606
00607 VERROR1:
00608  Vnm_print(2, "parseMG:  ran out of tokens!\n");
00609  return VRC_WARNING;
00610 }
00611
00612 VPRIVATE Vrc_Codes MGparm_parseGCENT(MGparm *thee, Vio *sock) {
00613
00614     char tok[VMAX_BUFSIZE];
00615     double tf;
00616     int ti;
00617
00618     /* If the next token isn't a float, it probably means we want to
00619      * center on a molecule */
00620     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00621     if (sscanf(tok, "%lf", &tf) == 0) {
00622         if (Vstring_strcasecmp(tok, "mol") == 0) {
00623             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00624             if (sscanf(tok, "%d", &ti) == 0) {
00625                 Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
```

```
00626 GCENT MOL keyword!\n", tok);
00627                 return VRC_WARNING;
00628             } else {
00629                 thee->cmeth = MCM_MOLECULE;
00630     /* Subtract 1 here to convert user numbering (1, 2, 3, ...) into
00631     array index */
00632                 thee->centmol = ti - 1;
00633             }
00634         } else {
00635             Vnm_print(2, "NOsh:  Unexpected keyword (%s) while parsing \
00636 GCENT!\n", tok);
00637             return VRC_WARNING;
00638         }
00639     } else {
00640         thee->center[0] = tf;
00641         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00642         if (sscanf(tok, "%lf", &tf) == 0) {
00643             Vnm_print(2, "NOsh:  Read non-float (%s) while parsing \
00644 GCENT keyword!\n", tok);
00645             return VRC_WARNING;
00646         }
00647         thee->center[1] = tf;
00648         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00649         if (sscanf(tok, "%lf", &tf) == 0) {
00650             Vnm_print(2, "NOsh:  Read non-float (%s) while parsing \
00651 GCENT keyword!\n", tok);
00652             return VRC_WARNING;
00653         }
00654         thee->center[2] = tf;
00655     }
00656     thee->setgcent = 1;
00657     return VRC_SUCCESS;
00658
00659     VERROR1:
00660         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00661         return VRC_WARNING;
00662 }
00663
00664 VPRIVATE Vrc_Codes MGparm_parseCGLEN(MGparm *thee, Vio *sock) {
00665
00666     char tok[VMAX_BUFSIZE];
00667     double tf;
00668
00669     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00670     if (sscanf(tok, "%lf", &tf) == 0) {
00671         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing CGLEN \
00672 keyword!\n", tok);
00673         return VRC_WARNING;
00674     } else thee->cglen[0] = tf;
00675     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00676     if (sscanf(tok, "%lf", &tf) == 0) {
00677         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing CGLEN \
00678 keyword!\n", tok);
00679         return VRC_WARNING;
00680     } else thee->cglen[1] = tf;
00681     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00682     if (sscanf(tok, "%lf", &tf) == 0) {
```

```
00683         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing CGLEN \
00684 keyword!\n", tok);
00685         return VRC_WARNING;
00686     } else thee->cglen[2] = tf;
00687     thee->setcglen = 1;
00688     return VRC_SUCCESS;
00689 
00690     VERROR1:
00691         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00692         return VRC_WARNING;
00693 }
00694 
00695 VPRIVATE Vrc_Codes MGparm_parseFGLEN(MGparm *thee, Vio *sock) {
00696 
00697     char tok[VMAX_BUFSIZE];
00698     double tf;
00699 
00700     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00701     if (sscanf(tok, "%lf", &tf) == 0) {
00702         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing FGLEN \
00703 keyword!\n", tok);
00704         return VRC_WARNING;
00705     } else thee->fglen[0] = tf;
00706     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00707     if (sscanf(tok, "%lf", &tf) == 0) {
00708         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing FGLEN \
00709 keyword!\n", tok);
00710         return VRC_WARNING;
00711     } else thee->fglen[1] = tf;
00712     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00713     if (sscanf(tok, "%lf", &tf) == 0) {
00714         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing FGLEN \
00715 keyword!\n", tok);
00716         return VRC_WARNING;
00717     } else thee->fglen[2] = tf;
00718     thee->setfglen = 1;
00719     return VRC_SUCCESS;
00720 
00721     VERROR1:
00722         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00723         return VRC_WARNING;
00724 }
00725 
00726 VPRIVATE Vrc_Codes MGparm_parseCGCENT(MGparm *thee, Vio *sock) {
00727 
00728     char tok[VMAX_BUFSIZE];
00729     double tf;
00730     int ti;
00731 
00732     /* If the next token isn't a float, it probably means we want to
00733      * center on a molecule */
00734     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00735     if (sscanf(tok, "%lf", &tf) == 0) {
00736         if (Vstring_strcasecmp(tok, "mol") == 0) {
00737             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00738             if (sscanf(tok, "%d", &ti) == 0) {
00739                 Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
```

```
00740 CGCENT MOL keyword!\n", tok);
00741                 return VRC_WARNING;
00742             } else {
00743     thee->ccmeth = MCM_MOLECULE;
00744     /* Subtract 1 here to convert user numbering (1, 2, 3, ...) into
00745     array index */
00746                 thee->ccentmol = ti - 1;
00747    }
00748         } else {
00749             Vnm_print(2, "NOsh:  Unexpected keyword (%s) while parsing \
00750 CGCENT!\n", tok);
00751             return VRC_WARNING;
00752             }
00753     } else {
00754         thee->ccenter[0] = tf;
00755         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00756         if (sscanf(tok, "%lf", &tf) == 0) {
00757             Vnm_print(2, "NOsh:  Read non-float (%s) while parsing \
00758 CGCENT keyword!\n", tok);
00759             return VRC_WARNING;
00760         }
00761         thee->ccenter[1] = tf;
00762         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00763         if (sscanf(tok, "%lf", &tf) == 0) {
00764             Vnm_print(2, "NOsh:  Read non-float (%s) while parsing \
00765 CGCENT keyword!\n", tok);
00766             return VRC_WARNING;
00767         }
00768         thee->ccenter[2] = tf;
00769     }
00770     thee->setcgcent = 1;
00771     return VRC_SUCCESS;
00772
00773     VERROR1:
00774         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00775         return VRC_WARNING;
00776 }
00777
00778 VPRIVATE Vrc_Codes MGparm_parseFGCENT(MGparm *thee, Vio *sock) {
00779
00780     char tok[VMAX_BUFSIZE];
00781     double tf;
00782     int ti;
00783
00784     /* If the next token isn't a float, it probably means we want to
00785      * center on a molecule */
00786     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00787     if (sscanf(tok, "%lf", &tf) == 0) {
00788         if (Vstring_strcasecmp(tok, "mol") == 0) {
00789             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00790             if (sscanf(tok, "%d", &ti) == 0) {
00791                 Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
00792 FGCENT MOL keyword!\n", tok);
00793                 return VRC_WARNING;
00794             } else {
00795                 thee->fcmeth = MCM_MOLECULE;
00796     /* Subtract 1 here to convert user numbering (1, 2, 3, ...) into
```

```
00797     array index */
00798                 thee->fcentmol = ti - 1;
00799             }
00800         } else {
00801             Vnm_print(2, "NOsh:  Unexpected keyword (%s) while parsing \
00802 FGCENT!\n", tok);
00803             return VRC_WARNING;
00804         }
00805     } else {
00806         thee->fcenter[0] = tf;
00807         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00808         if (sscanf(tok, "%lf", &tf) == 0) {
00809             Vnm_print(2, "NOsh:  Read non-float (%s) while parsing \
00810 FGCENT keyword!\n", tok);
00811             return VRC_WARNING;
00812         }
00813         thee->fcenter[1] = tf;
00814         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00815         if (sscanf(tok, "%lf", &tf) == 0) {
00816             Vnm_print(2, "NOsh:  Read non-float (%s) while parsing \
00817 FGCENT keyword!\n", tok);
00818             return VRC_WARNING;
00819         }
00820         thee->fcenter[2] = tf;
00821     }
00822     thee->setfgcent = 1;
00823     return VRC_SUCCESS;
00824 
00825     VERROR1:
00826         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00827         return VRC_WARNING;
00828 }
00829 
00830 VPRIVATE Vrc_Codes MGparm_parsePDIME(MGparm *thee, Vio *sock) {
00831 
00832     char tok[VMAX_BUFSIZE];
00833     int ti;
00834 
00835     /* Read the number of grid points */
00836     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00837     if (sscanf(tok, "%d", &ti) == 0) {
00838         Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing PDIME \
00839 keyword!\n", tok);
00840         return VRC_WARNING;
00841     } else {
00842         thee->pdime[0] = ti;
00843     }
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[1] = 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[2] = ti;
00859     }
00860     thee->setpdime = 1;
00861     return VRC_SUCCESS;
00862 
00863     VERROR1:
00864         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00865         return VRC_WARNING;
00866 }
00867 
00868 VPRIVATE Vrc_Codes MGparm_parseOFRAC(MGparm *thee, Vio *sock) {
00869 
00870     char tok[VMAX_BUFSIZE];
00871     double tf;
00872 
00873     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00874     if (sscanf(tok, "%lf", &tf) == 0) {
00875         Vnm_print(2, "NOsh:  Read non-int (%s) while parsing OFRAC \
00876 keyword!\n", tok);
00877         return VRC_WARNING;
00878     }
00879     thee->ofrac = tf;
00880     thee->setofrac = 1;
00881     return VRC_SUCCESS;
00882 
00883     VERROR1:
00884         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00885         return VRC_WARNING;
00886 }
00887 
00888 VPRIVATE Vrc_Codes MGparm_parseASYNC(MGparm *thee, Vio *sock) {
00889 
00890     char tok[VMAX_BUFSIZE];
00891     int ti;
00892 
00893     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00894     if (sscanf(tok, "%i", &ti) == 0) {
00895         Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing ASYNC \
00896 keyword!\n", tok);
00897         return VRC_WARNING;
00898     }
00899     thee->async = ti;
00900     thee->setasync = 1;
00901     return VRC_SUCCESS;
00902 
00903     VERROR1:
00904         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00905         return VRC_WARNING;
00906 }
00907 
00908 VPRIVATE Vrc_Codes MGparm_parseUSEAQUA(MGparm *thee, Vio *sock) {
00909     Vnm_print(0, "NOsh: parsed useaqua\n");
00910     thee->useAqua = 1;
```

```
00911     thee->setUseAqua = 1;
00912     return VRC_SUCCESS;
00913 }
00914
00915 VPUBLIC Vrc_Codes MGparm_parseToken(MGparm *thee, char tok[VMAX_BUFSIZE],
00916   Vio *sock) {
00917
00918     if (thee == VNULL) {
00919         Vnm_print(2, "parseMG:  got NULL thee!\n");
00920         return VRC_WARNING;
00921     }
00922     if (sock == VNULL) {
00923         Vnm_print(2, "parseMG:  got NULL socket!\n");
00924         return VRC_WARNING;
00925     }
00926
00927  Vnm_print(0, "MGparm_parseToken:  trying %s...\n", tok);
00928
00929
00930     if (Vstring_strcasecmp(tok, "dime") == 0) {
00931         return MGparm_parseDIME(thee, sock);
00932     } else if (Vstring_strcasecmp(tok, "chgm") == 0) {
00933         return MGparm_parseCHGM(thee, sock);
00934     } else if (Vstring_strcasecmp(tok, "nlev") == 0) {
00935         Vnm_print(2, "Warning: The 'nlev' keyword is now deprecated!\n");
00936         return MGparm_parseNLEV(thee, sock);
00937     } else if (Vstring_strcasecmp(tok, "etol") == 0) {
00938         return MGparm_parseETOL(thee, sock);
00939     } else if (Vstring_strcasecmp(tok, "grid") == 0) {
00940         return MGparm_parseGRID(thee, sock);
00941     } else if (Vstring_strcasecmp(tok, "glen") == 0) {
00942         return MGparm_parseGLEN(thee, sock);
00943     } else if (Vstring_strcasecmp(tok, "gcent") == 0) {
00944         return MGparm_parseGCENT(thee, sock);
00945     } else if (Vstring_strcasecmp(tok, "cglen") == 0) {
00946         return MGparm_parseCGLEN(thee, sock);
00947     } else if (Vstring_strcasecmp(tok, "fglen") == 0) {
00948         return MGparm_parseFGLEN(thee, sock);
00949     } else if (Vstring_strcasecmp(tok, "cgcent") == 0) {
00950         return MGparm_parseCGCENT(thee, sock);
00951     } else if (Vstring_strcasecmp(tok, "fgcent") == 0) {
00952         return MGparm_parseFGCENT(thee, sock);
00953     } else if (Vstring_strcasecmp(tok, "pdime") == 0) {
00954         return MGparm_parsePDIME(thee, sock);
00955     } else if (Vstring_strcasecmp(tok, "ofrac") == 0) {
00956         return MGparm_parseOFRAC(thee, sock);
00957     } else if (Vstring_strcasecmp(tok, "async") == 0) {
00958         return MGparm_parseASYNC(thee, sock);
00959  } else if (Vstring_strcasecmp(tok, "gamma") == 0) {
00960         return MGparm_parseGAMMA(thee, sock);
00961  } else if (Vstring_strcasecmp(tok, "useaqua") == 0) {
00962         return MGparm_parseUSEAQUA(thee, sock);
00963     } else {
00964         Vnm_print(2, "parseMG:  Unrecognized keyword (%s)!\n", tok);
00965         return VRC_WARNING;
00966     }
00967
```

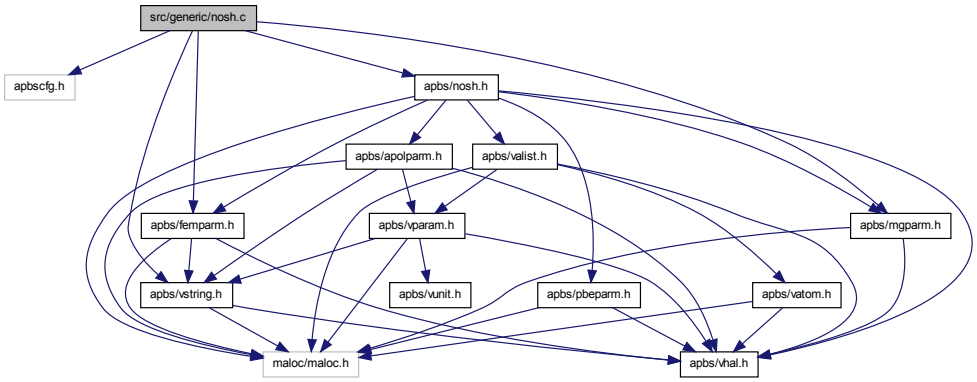
```
00968     return VRC_FAILURE;
00969
00970 }
```

10.57 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 1585 2010-05-13 16:21:17Z sdg0919

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
```

```
 * Additional contributing authors listed in the code documentation.
 *
 * Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
 * 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 Washington University in St. Louis 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 src/generic/nosh.c

```
00001
00050 #include "apbscfg.h"
00051 #include "apbs/nosh.h"
00052 #include "apbs/vstring.h"
00053 #include "apbs/mgparm.h"
00054 #include "apbs/femparm.h"
00055
00056 VEMBED(rcsid="$Id: nosh.c 1585 2010-05-13 16:21:17Z sdg0919 $")
00057
00058
00059 VPRIVATE int NOsh_parseREAD(
00060        NOsh *thee,
00061        Vio *sock);
00062
00063 VPRIVATE int NOsh_parsePRINT(
00064         NOsh *thee,
00065         Vio *sock);
```

```
00066
00067 VPRIVATE int NOsh_parseELEC(
00068        NOsh *thee,
00069        Vio *sock
00070        );
00071
00072 VPRIVATE int NOsh_parseAPOLAR(
00073        NOsh *thee,
00074        Vio *sock
00075        );
00076
00077 VEXTERNC int NOsh_parseFEM(
00078          NOsh *thee,
00079          Vio *sock,
00080          NOsh_calc *elec
00081          );
00082
00083 VEXTERNC int NOsh_parseMG(
00084         NOsh *thee,
00085         Vio *sock,
00086         NOsh_calc *elec
00087         );
00088
00089 VEXTERNC int NOsh_parseAPOL(
00090          NOsh *thee,
00091          Vio *sock,
00092          NOsh_calc *elec
00093          );
00094
00095 VPRIVATE int NOsh_setupCalcMG(
00096          NOsh *thee,
00097          NOsh_calc *elec
00098          );
00099
00100 VPRIVATE int NOsh_setupCalcMGAUTO(
00101           NOsh *thee,
00102           NOsh_calc *elec
00103           );
00104
00105 VPRIVATE int NOsh_setupCalcMGMANUAL(
00106          NOsh *thee,
00107          NOsh_calc *elec
00108          );
00109
00110 VPRIVATE int NOsh_setupCalcMGPARA(
00111           NOsh *thee,
00112           NOsh_calc *elec
00113           );
00114
00115 VPRIVATE int NOsh_setupCalcFEM(
00116           NOsh *thee,
00117           NOsh_calc *elec
00118           );
00119
00120 VPRIVATE int NOsh_setupCalcFEMANUAL(
00121           NOsh *thee,
00122           NOsh_calc *elec
```

```
00123           );
00124
00125 VPRIVATE int NOsh_setupCalcAPOL(
00126         NOsh *thee,
00127         NOsh_calc *elec
00128         );
00129
00130 #if !defined(VINLINE_NOSH)
00131
00132 VPUBLIC char* NOsh_getMolpath(NOsh *thee, int imol) {
00133  VASSERT(thee != VNULL);
00134  VASSERT(imol < thee->nmol);
00135  return thee->molpath[imol];
00136 }
00137 VPUBLIC char* NOsh_getDielXpath(NOsh *thee, int imol) {
00138  VASSERT(thee != VNULL);
00139  VASSERT(imol < thee->nmol);
00140  return thee->dielXpath[imol];
00141 }
00142 VPUBLIC char* NOsh_getDielYpath(NOsh *thee, int imol) {
00143  VASSERT(thee != VNULL);
00144  VASSERT(imol < thee->nmol);
00145  return thee->dielYpath[imol];
00146 }
00147 VPUBLIC char* NOsh_getDielZpath(NOsh *thee, int imol) {
00148  VASSERT(thee != VNULL);
00149  VASSERT(imol < thee->nmol);
00150  return thee->dielZpath[imol];
00151 }
00152 VPUBLIC char* NOsh_getKappapath(NOsh *thee, int imol) {
00153  VASSERT(thee != VNULL);
00154  VASSERT(imol < thee->nmol);
00155  return thee->kappapath[imol];
00156 }
00157 VPUBLIC char* NOsh_getPotpath(NOsh *thee, int imol) {
00158  VASSERT(thee != VNULL);
00159  VASSERT(imol < thee->nmol);
00160  return thee->potpath[imol];
00161 }
00162 VPUBLIC char* NOsh_getChargepath(NOsh *thee, int imol) {
00163  VASSERT(thee != VNULL);
00164  VASSERT(imol < thee->nmol);
00165  return thee->chargepath[imol];
00166 }
00167 VPUBLIC NOsh_calc* NOsh_getCalc(NOsh *thee, int icalc) {
00168  VASSERT(thee != VNULL);
00169  VASSERT(icalc < thee->ncalc);
00170  return thee->calc[icalc];
00171 }
00172 VPUBLIC int NOsh_getDielfmt(NOsh *thee, int i) {
00173  VASSERT(thee != VNULL);
00174  VASSERT(i < thee->ndiel);
00175  return (thee->dielfmt[i]);
00176 }
00177 VPUBLIC int NOsh_getKappafmt(NOsh *thee, int i) {
00178  VASSERT(thee != VNULL);
00179  VASSERT(i < thee->nkappa);
```

```
00180  return (thee->kappafmt[i]);
00181 }
00182 VPUBLIC int NOsh_getPotfmt(NOsh *thee, int i) {
00183  VASSERT(thee != VNULL);
00184  VASSERT(i < thee->npot);
00185  return (thee->potfmt[i]);
00186 }
00187 VPUBLIC int NOsh_getChargefmt(NOsh *thee, int i) {
00188  VASSERT(thee != VNULL);
00189  VASSERT(i < thee->ncharge);
00190  return (thee->chargefmt[i]);
00191 }
00192
00193
00194 #endif /* if !defined(VINLINE_NOSH) */
00195
00196 VPUBLIC NOsh_PrintType NOsh_printWhat(NOsh *thee, int iprint) {
00197  VASSERT(thee != VNULL);
00198  VASSERT(iprint < thee->nprint);
00199  return thee->printwhat[iprint];
00200 }
00201
00202 VPUBLIC int NOsh_printNarg(NOsh *thee, int iprint) {
00203  VASSERT(thee != VNULL);
00204  VASSERT(iprint < thee->nprint);
00205  return thee->printnarg[iprint];
00206 }
00207
00208 VPUBLIC int NOsh_elec2calc(NOsh *thee, int icalc) {
00209  VASSERT(thee != VNULL);
00210  VASSERT(icalc < thee->ncalc);
00211  return thee->elec2calc[icalc];
00212 }
00213
00214 VPUBLIC int NOsh_apol2calc(NOsh *thee, int icalc) {
00215  VASSERT(thee != VNULL);
00216  VASSERT(icalc < thee->ncalc);
00217  return thee->apol2calc[icalc];
00218 }
00219
00220 VPUBLIC char* NOsh_elecname(NOsh *thee, int ielec) {
00221  VASSERT(thee != VNULL);
00222  VASSERT(ielec < thee->nelec + 1);
00223  return thee->elecname[ielec];
00224 }
00225
00226 VPUBLIC int NOsh_printOp(NOsh *thee, int iprint, int iarg) {
00227  VASSERT(thee != VNULL);
00228  VASSERT(iprint < thee->nprint);
00229  VASSERT(iarg < thee->printnarg[iprint]);
00230  return thee->printop[iprint][iarg];
00231 }
00232
00233 VPUBLIC int NOsh_printCalc(NOsh *thee, int iprint, int iarg) {
00234  VASSERT(thee != VNULL);
00235  VASSERT(iprint < thee->nprint);
00236  VASSERT(iarg < thee->printnarg[iprint]);
```

```
00237  return thee->printcalc[iprint][iarg];
00238 }
00239
00240 VPUBLIC NOsh* NOsh_ctor(int rank, int size) {
00241
00242  /* Set up the structure */
00243  NOsh *thee = VNULL;
00244  thee = Vmem_malloc(VNULL, 1, sizeof(NOsh) );
00245  VASSERT( thee != VNULL);
00246  VASSERT( NOsh_ctor2(thee, rank, size) );
00247
00248  return thee;
00249 }
00250
00251 VPUBLIC int NOsh_ctor2(NOsh *thee, int rank, int size) {
00252
00253  int i;
00254
00255  if (thee == VNULL) return 0;
00256
00257  thee->proc_rank = rank;
00258  thee->proc_size = size;
00259
00260  thee->ispara = 0;
00261     thee->parsed = 0;
00262
00263     thee->nmol = 0;
00264     thee->gotparm = 0;
00265     thee->ncharge = 0;
00266     thee->ndiel = 0;
00267     thee->nkappa = 0;
00268  thee->npot = 0;
00269     thee->nprint = 0;
00270
00271     for (i=0; i<NOSH_MAXCALC; i++) {
00272   thee->calc[i] = VNULL;
00273   thee->elec[i] = VNULL;
00274   thee->apol[i] = VNULL;
00275  }
00276  for (i=0; i<NOSH_MAXMOL; i++) {
00277   thee->alist[i] = VNULL;
00278  }
00279  thee->ncalc = 0;
00280  thee->nelec = 0;
00281  thee->napol = 0;
00282
00283     return 1;
00284 }
00285
00286 VPUBLIC void NOsh_dtor(NOsh **thee) {
00287     if ((*thee) != VNULL) {
00288         NOsh_dtor2(*thee);
00289         Vmem_free(VNULL, 1, sizeof(NOsh), (void **)thee);
00290         (*thee) = VNULL;
00291     }
00292 }
00293
```

```
00294 VPUBLIC void NOsh_dtor2(NOsh *thee) {
00295
00296  int i;
00297
00298  if (thee != VNULL) {
00299   for (i=0; i<(thee->ncalc); i++) NOsh_calc_dtor(&(thee->calc[i]));
00300   for (i=0; i<(thee->nelec); i++) NOsh_calc_dtor(&(thee->elec[i]));
00301   for (i=0; i<(thee->napol); i++) NOsh_calc_dtor(&(thee->apol[i]));
00302     }
00303
00304 }
00305
00306 VPUBLIC NOsh_calc* NOsh_calc_ctor(
00307           NOsh_CalcType calctype
00308           ) {
00309  NOsh_calc *thee;
00310  thee = (NOsh_calc *)Vmem_malloc(VNULL, 1, sizeof(NOsh_calc));
00311  thee->calctype = calctype;
00312  switch (calctype) {
00313   case NCT_MG:
00314    thee->mgparm = MGparm_ctor(MCT_NONE);
00315    thee->femparm = VNULL;
00316    thee->apolparm = VNULL;
00317    break;
00318   case NCT_FEM:
00319    thee->mgparm = VNULL;
00320    thee->femparm = FEMparm_ctor(FCT_NONE);
00321    thee->apolparm = VNULL;
00322    break;
00323   case NCT_APOL:
00324    thee->mgparm = VNULL;
00325    thee->femparm = VNULL;
00326    thee->apolparm = APOLparm_ctor();
00327    break;
00328   default:
00329    Vnm_print(2, "NOsh_calc_ctor:  unknown calculation type (%d)!\n",
00330        calctype);
00331    VASSERT(0);
00332  }
00333  thee->pbeparm = PBEparm_ctor();
00334
00335  return thee;
00336 }
00337
00338 VPUBLIC void NOsh_calc_dtor(
00339       NOsh_calc **thee
00340       ) {
00341
00342  NOsh_calc *calc = VNULL;
00343  calc = *thee;
00344  if (calc == VNULL) return;
00345
00346  switch (calc->calctype) {
00347   case NCT_MG:
00348    MGparm_dtor(&(calc->mgparm));
00349    break;
00350   case NCT_FEM:
```

```
00351     FEMparm_dtor(&(calc->femparm));
00352     break;
00353   case NCT_APOL:
00354     APOLparm_dtor(&(calc->apolparm));
00355     break;
00356   default:
00357     Vnm_print(2, "NOsh_calc_ctor:  unknown calculation type (%d)!\n",
00358         calc->calctype);
00359     VASSERT(0);
00360  }
00361  PBEparm_dtor(&(calc->pbeparm));
00362
00363  Vmem_free(VNULL, 1, sizeof(NOsh_calc), (void **)thee);
00364  calc = VNULL;
00365
00366 }
00367
00368 VPUBLIC int NOsh_calc_copy(
00369          NOsh_calc *thee,
00370          NOsh_calc *source
00371          ) {
00372
00373  VASSERT(thee != VNULL);
00374  VASSERT(source != VNULL);
00375  VASSERT(thee->calctype == source->calctype);
00376  if (source->mgparm != VNULL)
00377   MGparm_copy(thee->mgparm, source->mgparm);
00378  if (source->femparm != VNULL)
00379   FEMparm_copy(thee->femparm, source->femparm);
00380  if (source->pbeparm != VNULL)
00381   PBEparm_copy(thee->pbeparm, source->pbeparm);
00382  if (source->apolparm != VNULL)
00383   APOLparm_copy(thee->apolparm, source->apolparm);
00384
00385  return 1;
00386
00387 }
00388
00389 VPUBLIC int NOsh_parseInputFile(
00390         NOsh *thee,
00391         char *filename
00392         ) {
00393
00394  Vio *sock;
00395  int rc;
00396
00397  sock = Vio_ctor("FILE", "ASC", VNULL, filename, "r");
00398  rc = NOsh_parseInput(thee, sock);
00399  Vio_dtor(&sock);
00400
00401  return rc;
00402 }
00403
00404 VPUBLIC int NOsh_parseInput(
00405        NOsh *thee,
00406        Vio *sock
00407        ) {
```

```
00408
00409  char *MCwhiteChars = " =,;\t\r\n";
00410  char *MCcommChars  = "#%";
00411  char tok[VMAX_BUFSIZE];
00412
00413  if (thee == VNULL) {
00414   Vnm_print(2, "NOsh_parseInput:  Got NULL thee!\n");
00415   return 0;
00416  }
00417
00418  if (sock == VNULL) {
00419   Vnm_print(2, "NOsh_parseInput:  Got pointer to NULL socket!\n");
00420   Vnm_print(2, "NOsh_parseInput:  The specified input file was not found!\n");
00421   return 0;
00422  }
00423
00424  if (thee->parsed) {
00425   Vnm_print(2, "NOsh_parseInput:  Already parsed an input file!\n");
00426   return 0;
00427  }
00428
00429  if (Vio_accept(sock, 0) < 0) {
00430   Vnm_print(2, "NOsh_parseInput:  Problem reading from socket!\n");
00431   return 0;
00432  }
00433
00434  /* Set up the whitespace and comment character definitions */
00435  Vio_setWhiteChars(sock, MCwhiteChars);
00436  Vio_setCommChars(sock, MCcommChars);
00437
00438  /* We parse the file until we run out of tokens */
00439  Vnm_print(0, "NOsh_parseInput:  Starting file parsing...\n");
00440  while (Vio_scanf(sock, "%s", tok) == 1) {
00441   /* At the highest level, we look for keywords that indicate functions like:
00442
00443   read => Read in a molecule file
00444   elec => Do an electrostatics calculation
00445   print => Print some results
00446   apolar => do a non-polar calculation
00447   quit => Quit
00448
00449   These cause the code to go to a lower-level parser routine which
00450   handles keywords specific to the particular function.  Each
00451   lower-level parser routine then returns when it hits the "end"
00452   keyword.  Due to this simple layout, no nesting of these "function"
00453   sections is allowed.
00454   */
00455   if (Vstring_strcasecmp(tok, "read") == 0) {
00456    Vnm_print(0, "NOsh: Parsing READ section\n");
00457    if (!NOsh_parseREAD(thee, sock)) return 0;
00458    Vnm_print(0, "NOsh: Done parsing READ section \
00459 (nmol=%d, ndiel=%d, nkappa=%d, ncharge=%d, npot=%d)\n", thee->nmol, thee->ndiel,

00460         thee->nkappa, thee->ncharge,thee->npot);
00461   } else if (Vstring_strcasecmp(tok, "print") == 0) {
00462    Vnm_print(0, "NOsh: Parsing PRINT section\n");
00463    if (!NOsh_parsePRINT(thee, sock)) return 0;
```

```
00464    Vnm_print(0, "NOsh: Done parsing PRINT section\n");
00465   } else if (Vstring_strcasecmp(tok, "elec") == 0) {
00466    Vnm_print(0, "NOsh: Parsing ELEC section\n");
00467    if (!NOsh_parseELEC(thee, sock)) return 0;
00468    Vnm_print(0, "NOsh: Done parsing ELEC section (nelec = %d)\n",
00469        thee->nelec);
00470   } else if (Vstring_strcasecmp(tok, "apolar") == 0) {
00471    Vnm_print(0, "NOsh: Parsing APOLAR section\n");
00472    if (!NOsh_parseAPOLAR(thee, sock)) return 0;
00473    Vnm_print(0, "NOsh: Done parsing APOLAR section (nelec = %d)\n",
00474        thee->nelec);
00475   } else if (Vstring_strcasecmp(tok, "quit") == 0) {
00476    Vnm_print(0, "NOsh: Done parsing file (got QUIT)\n");
00477    break;
00478   } else {
00479    Vnm_print(2, "NOsh_parseInput: Ignoring undefined keyword %s!\n", tok);
00480   }
00481  }
00482
00483  thee->parsed = 1;
00484  return 1;
00485
00486 }
00487
00488 VPRIVATE int NOsh_parseREAD_MOL(NOsh *thee, Vio *sock) {
00489
00490     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00491     NOsh_MolFormat molfmt;
00492
00493     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00494     if (Vstring_strcasecmp(tok, "pqr") == 0) {
00495         molfmt = NMF_PQR;
00496         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00497         if (tok[0]=='"') {
00498             strcpy(strnew, "");
00499             while (tok[strlen(tok)-1] != '"') {
00500                 strcat(str, tok);
00501                 strcat(str, " ");
00502                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00503    }
00504             strcat(str, tok);
00505             strncpy(strnew, str+1, strlen(str)-2);
00506             strcpy(tok, strnew);
00507         }
00508         Vnm_print(0, "NOsh: Storing molecule %d path %s\n",
00509       thee->nmol, tok);
00510         thee->molfmt[thee->nmol] = molfmt;
00511         strncpy(thee->molpath[thee->nmol], tok, VMAX_ARGLEN);
00512         (thee->nmol)++;
00513     } else if (Vstring_strcasecmp(tok, "pdb") == 0) {
00514         molfmt = NMF_PDB;
00515         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00516         if (tok[0]=='"') {
00517             strcpy(strnew, "");
00518             while (tok[strlen(tok)-1] != '"') {
00519                 strcat(str, tok);
00520                 strcat(str, " ");
```

```
00521                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00522     }
00523             strcat(str, tok);
00524             strncpy(strnew, str+1, strlen(str)-2);
00525             strcpy(tok, strnew);
00526         }
00527         Vnm_print(0, "NOsh: Storing molecule %d path %s\n",
00528       thee->nmol, tok);
00529         thee->molfmt[thee->nmol] = molfmt;
00530         strncpy(thee->molpath[thee->nmol], tok, VMAX_ARGLEN);
00531         (thee->nmol)++;
00532     } else if (Vstring_strcasecmp(tok, "xml") == 0) {
00533         molfmt = NMF_XML;
00534         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00535         if (tok[0]=='"') {
00536             strcpy(strnew, "");
00537             while (tok[strlen(tok)-1] != '"') {
00538                 strcat(str, tok);
00539                 strcat(str, " ");
00540                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00541     }
00542             strcat(str, tok);
00543             strncpy(strnew, str+1, strlen(str)-2);
00544             strcpy(tok, strnew);
00545         }
00546         Vnm_print(0, "NOsh: Storing molecule %d path %s\n",
00547       thee->nmol, tok);
00548         thee->molfmt[thee->nmol] = molfmt;
00549         strncpy(thee->molpath[thee->nmol], tok, VMAX_ARGLEN);
00550         (thee->nmol)++;
00551     } else {
00552         Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined mol format \
00553 %s!\n", tok);
00554     }
00555
00556     return 1;
00557
00558
00559 VERROR1:
00560         Vnm_print(2, "NOsh_parseREAD_MOL:  Ran out of tokens while parsing READ s
      ection!\n");
00561  return 0;
00562
00563 }
00564
00565 VPRIVATE int NOsh_parseREAD_PARM(NOsh *thee, Vio *sock) {
00566
00567     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00568     NOsh_ParmFormat parmfmt;
00569
00570     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00571     if (Vstring_strcasecmp(tok, "flat") == 0) {
00572         parmfmt = NPF_FLAT;
00573         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00574         if (tok[0]=='"') {
00575             strcpy(strnew, "");
00576             while (tok[strlen(tok)-1] != '"') {
```

```
00577                 strcat(str, tok);
00578                 strcat(str, " ");
00579                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00580     }
00581             strcat(str, tok);
00582             strncpy(strnew, str+1, strlen(str)-2);
00583             strcpy(tok, strnew);
00584         }
00585         if (thee->gotparm) {
00586             Vnm_print(2, "NOsh:  Hey!  You already specified a parameterfile (%s)
      !\n", thee->parmpath);
00587             Vnm_print(2, "NOsh:  I'm going to ignore this one (%s)!\n", tok);
00588         } else {
00589             thee->parmfmt = parmfmt;
00590             thee->gotparm = 1;
00591             strncpy(thee->parmpath, tok, VMAX_ARGLEN);
00592         }
00593     } else if(Vstring_strcasecmp(tok, "xml") == 0) {
00594         parmfmt = NPF_XML;
00595         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00596         if (tok[0]=='"') {
00597             strcpy(strnew, "");
00598             while (tok[strlen(tok)-1] != '"') {
00599                 strcat(str, tok);
00600                 strcat(str, " ");
00601                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00602     }
00603             strcat(str, tok);
00604             strncpy(strnew, str+1, strlen(str)-2);
00605             strcpy(tok, strnew);
00606         }
00607         if (thee->gotparm) {
00608             Vnm_print(2, "NOsh:  Hey!  You already specified a parameterfile (%s)
      !\n", thee->parmpath);
00609             Vnm_print(2, "NOsh:  I'm going to ignore this one (%s)!\n", tok);
00610         } else {
00611             thee->parmfmt = parmfmt;
00612             thee->gotparm = 1;
00613             strncpy(thee->parmpath, tok, VMAX_ARGLEN);
00614         }
00615
00616     } else {
00617         Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined parm format \
00618 %s!\n", tok);
00619     }
00620
00621     return 1;
00622
00623 VERROR1:
00624         Vnm_print(2, "NOsh_parseREAD_PARM:  Ran out of tokens while parsing READ
      section!\n");
00625  return 0;
00626
00627 }
00628
00629 VPRIVATE int NOsh_parseREAD_DIEL(NOsh *thee, Vio *sock) {
00630
```

```
00631     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00632     Vdata_Format dielfmt;
00633
00634     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00635     if (Vstring_strcasecmp(tok, "dx") == 0) {
00636         dielfmt = VDF_DX;
00637  } else if (Vstring_strcasecmp(tok, "gz") == 0) {
00638         dielfmt = VDF_GZ;
00639  } else {
00640         Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined format \
00641       %s!\n", tok);
00642   return VRC_FAILURE;
00643  }
00644
00645  VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00646  if (tok[0]=='"') {
00647   strcpy(strnew, "");
00648   while (tok[strlen(tok)-1] != '"') {
00649    strcat(str, tok);
00650    strcat(str, " ");
00651    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00652   }
00653   strcat(str, tok);
00654   strncpy(strnew, str+1, strlen(str)-2);
00655   strcpy(tok, strnew);
00656  }
00657  Vnm_print(0, "NOsh: Storing x-shifted dielectric map %d path \
00658      %s\n", thee->ndiel, tok);
00659  strncpy(thee->dielXpath[thee->ndiel], tok, VMAX_ARGLEN);
00660  VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00661  Vnm_print(0, "NOsh: Storing y-shifted dielectric map %d path \
00662      %s\n", thee->ndiel, tok);
00663  strncpy(thee->dielYpath[thee->ndiel], tok, VMAX_ARGLEN);
00664  VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00665  Vnm_print(0, "NOsh: Storing z-shifted dielectric map %d path \
00666      %s\n", thee->ndiel, tok);
00667  strncpy(thee->dielZpath[thee->ndiel], tok, VMAX_ARGLEN);
00668  thee->dielfmt[thee->ndiel] = dielfmt;
00669  (thee->ndiel)++;
00670
00671  return 1;
00672
00673 VERROR1:
00674         Vnm_print(2, "NOsh_parseREAD_DIEL:  Ran out of tokens while parsing READ
      \
00675 section!\n");
00676  return 0;
00677
00678 }
00679
00680 VPRIVATE int NOsh_parseREAD_KAPPA(NOsh *thee, Vio *sock) {
00681
00682     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00683     Vdata_Format kappafmt;
00684
00685     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00686     if (Vstring_strcasecmp(tok, "dx") == 0) {
```

```
00687         kappafmt = VDF_DX;
00688  } else if (Vstring_strcasecmp(tok, "gz") == 0) {
00689         kappafmt = VDF_GZ;
00690  } else {
00691         Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined format \
00692       %s!\n", tok);
00693   return VRC_FAILURE;
00694  }
00695
00696  VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00697  if (tok[0]=='"') {
00698   strcpy(strnew, "");
00699   while (tok[strlen(tok)-1] != '"') {
00700    strcat(str, tok);
00701    strcat(str, " ");
00702    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00703   }
00704   strcat(str, tok);
00705   strncpy(strnew, str+1, strlen(str)-2);
00706   strcpy(tok, strnew);
00707  }
00708  Vnm_print(0, "NOsh: Storing kappa map %d path %s\n",
00709      thee->nkappa, tok);
00710  thee->kappafmt[thee->nkappa] = kappafmt;
00711  strncpy(thee->kappapath[thee->nkappa], tok, VMAX_ARGLEN);
00712  (thee->nkappa)++;
00713
00714     return 1;
00715
00716 VERROR1:
00717         Vnm_print(2, "NOsh_parseREAD:  Ran out of tokens while parsing READ \
00718 section!\n");
00719  return 0;
00720
00721 }
00722
00723 VPRIVATE int NOsh_parseREAD_POTENTIAL(NOsh *thee, Vio *sock) {
00724
00725     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00726     Vdata_Format potfmt;
00727
00728     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00729     if (Vstring_strcasecmp(tok, "dx") == 0) {
00730         potfmt = VDF_DX;
00731  } else if (Vstring_strcasecmp(tok, "gz") == 0) {
00732         potfmt = VDF_GZ;
00733  } else {
00734         Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined format \
00735       %s!\n", tok);
00736   return VRC_FAILURE;
00737    }
00738
00739  VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00740  if (tok[0]=='"') {
00741   strcpy(strnew, "");
00742   while (tok[strlen(tok)-1] != '"') {
00743    strcat(str, tok);
```

```
00744    strcat(str, " ");
00745    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00746   }
00747   strcat(str, tok);
00748   strncpy(strnew, str+1, strlen(str)-2);
00749   strcpy(tok, strnew);
00750  }
00751  Vnm_print(0, "NOsh: Storing potential map %d path %s\n",
00752      thee->npot, tok);
00753  thee->potfmt[thee->npot] = potfmt;
00754  strncpy(thee->potpath[thee->npot], tok, VMAX_ARGLEN);
00755  (thee->npot)++;
00756
00757  return 1;
00758
00759 VERROR1:
00760  Vnm_print(2, "NOsh_parseREAD:  Ran out of tokens while parsing READ \
00761      section!\n");
00762  return 0;
00763
00764 }
00765
00766 VPRIVATE int NOsh_parseREAD_CHARGE(NOsh *thee, Vio *sock) {
00767
00768     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00769     Vdata_Format chargefmt;
00770
00771     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00772     if (Vstring_strcasecmp(tok, "dx") == 0) {
00773         chargefmt = VDF_DX;
00774  } else if (Vstring_strcasecmp(tok, "gz") == 0) {
00775         chargefmt = VDF_GZ;
00776  } else {
00777         Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined format \
00778       %s!\n", tok);
00779   return VRC_FAILURE;
00780  }
00781
00782  VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00783  if (tok[0]=='"') {
00784   strcpy(strnew, "");
00785   while (tok[strlen(tok)-1] != '"') {
00786    strcat(str, tok);
00787    strcat(str, " ");
00788    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00789   }
00790   strcat(str, tok);
00791   strncpy(strnew, str+1, strlen(str)-2);
00792   strcpy(tok, strnew);
00793  }
00794  Vnm_print(0, "NOsh: Storing charge map %d path %s\n",
00795      thee->ncharge, tok);
00796  thee->chargefmt[thee->ncharge] = chargefmt;
00797  strncpy(thee->chargepath[thee->ncharge], tok, VMAX_ARGLEN);
00798  (thee->ncharge)++;
00799
00800     return 1;
```

```
00801
00802 VERROR1:
00803         Vnm_print(2, "NOsh_parseREAD:  Ran out of tokens while parsing READ \
00804 section!\n");
00805  return 0;
00806
00807 }
00808
00809 VPRIVATE int NOsh_parseREAD_MESH(NOsh *thee, Vio *sock) {
00810
00811     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00812     Vdata_Format meshfmt;
00813
00814     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00815     if (Vstring_strcasecmp(tok, "mcsf") == 0) {
00816         meshfmt = VDF_MCSF;
00817         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00818         if (tok[0]=='"') {
00819             strcpy(strnew, "");
00820             while (tok[strlen(tok)-1] != '"') {
00821                 strcat(str, tok);
00822                 strcat(str, " ");
00823                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00824    }
00825             strcat(str, tok);
00826             strncpy(strnew, str+1, strlen(str)-2);
00827             strcpy(tok, strnew);
00828         }
00829         Vnm_print(0, "NOsh: Storing mesh %d path %s\n",
00830       thee->nmesh, tok);
00831         thee->meshfmt[thee->nmesh] = meshfmt;
00832         strncpy(thee->meshpath[thee->nmesh], tok, VMAX_ARGLEN);
00833         (thee->nmesh)++;
00834     } else {
00835         Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined mesh format \
00836       %s!\n", tok);
00837     }
00838
00839     return 1;
00840
00841 VERROR1:
00842  Vnm_print(2, "NOsh_parseREAD:  Ran out of tokens while parsing READ \
00843      section!\n");
00844  return 0;
00845
00846 }
00847
00848
00849 VPRIVATE int NOsh_parseREAD(NOsh *thee, Vio *sock) {
00850
00851     char tok[VMAX_BUFSIZE];
00852
00853     if (thee == VNULL) {
00854         Vnm_print(2, "NOsh_parseREAD:  Got NULL thee!\n");
00855         return 0;
00856     }
00857
```

```
00858     if (sock == VNULL) {
00859         Vnm_print(2, "NOsh_parseREAD:  Got pointer to NULL socket!\n");
00860         return 0;
00861     }
00862
00863     if (thee->parsed) {
00864         Vnm_print(2, "NOsh_parseREAD:  Already parsed an input file!\n");
00865         return 0;
00866     }
00867
00868     /* Read until we run out of tokens (bad) or hit the "END" keyword (good) */
00869     while (Vio_scanf(sock, "%s", tok) == 1) {
00870         if (Vstring_strcasecmp(tok, "end") == 0) {
00871             Vnm_print(0, "NOsh: Done parsing READ section\n");
00872             return 1;
00873         } else if (Vstring_strcasecmp(tok, "mol") == 0) {
00874             NOsh_parseREAD_MOL(thee, sock);
00875         } else if (Vstring_strcasecmp(tok, "parm") == 0) {
00876             NOsh_parseREAD_PARM(thee,sock);
00877         } else if (Vstring_strcasecmp(tok, "diel") == 0) {
00878             NOsh_parseREAD_DIEL(thee,sock);
00879         } else if (Vstring_strcasecmp(tok, "kappa") == 0) {
00880             NOsh_parseREAD_KAPPA(thee,sock);
00881   } else if (Vstring_strcasecmp(tok, "pot") == 0) {
00882             NOsh_parseREAD_POTENTIAL(thee,sock);
00883         } else if (Vstring_strcasecmp(tok, "charge") == 0) {
00884             NOsh_parseREAD_CHARGE(thee,sock);
00885   } else if (Vstring_strcasecmp(tok, "mesh") == 0) {
00886    NOsh_parseREAD_MESH(thee,sock);
00887         } else {
00888             Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined keyword %s!\n",
00889        tok);
00890         }
00891     }
00892
00893     /* We ran out of tokens! */
00894     Vnm_print(2, "NOsh_parseREAD:  Ran out of tokens while parsing READ \
00895 section!\n");
00896     return 0;
00897
00898 }
00899
00900 VPRIVATE int NOsh_parsePRINT(NOsh *thee, Vio *sock) {
00901
00902     char tok[VMAX_BUFSIZE];
00903  char name[VMAX_BUFSIZE];
00904     int ti, idx, expect, ielec, iapol;
00905
00906     if (thee == VNULL) {
00907         Vnm_print(2, "NOsh_parsePRINT:  Got NULL thee!\n");
00908         return 0;
00909     }
00910
00911     if (sock == VNULL) {
00912         Vnm_print(2, "NOsh_parsePRINT:  Got pointer to NULL socket!\n");
00913         return 0;
00914     }
```

```
00915
00916     if (thee->parsed) {
00917         Vnm_print(2, "NOsh_parsePRINT:  Already parsed an input file!\n");
00918         return 0;
00919     }
00920
00921     idx = thee->nprint;
00922     if (thee->nprint >= NOSH_MAXPRINT) {
00923         Vnm_print(2, "NOsh_parsePRINT:  Exceeded max number (%d) of PRINT \
00924 sections\n",
00925       NOSH_MAXPRINT);
00926         return 0;
00927     }
00928
00929
00930     /* The first thing we read is the thing we want to print */
00931     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00932     if (Vstring_strcasecmp(tok, "energy") == 0) {
00933         thee->printwhat[idx] = NPT_ENERGY;
00934         thee->printnarg[idx] = 0;
00935     } else if (Vstring_strcasecmp(tok, "force") == 0) {
00936         thee->printwhat[idx] = NPT_FORCE;
00937         thee->printnarg[idx] = 0;
00938  } else if (Vstring_strcasecmp(tok, "elecEnergy") == 0) {
00939         thee->printwhat[idx] = NPT_ELECENERGY;
00940         thee->printnarg[idx] = 0;
00941  } else if (Vstring_strcasecmp(tok, "elecForce") == 0) {
00942         thee->printwhat[idx] = NPT_ELECFORCE;
00943         thee->printnarg[idx] = 0;
00944     } else if (Vstring_strcasecmp(tok, "apolEnergy") == 0) {
00945         thee->printwhat[idx] = NPT_APOLENERGY;
00946         thee->printnarg[idx] = 0;
00947  } else if (Vstring_strcasecmp(tok, "apolForce") == 0) {
00948         thee->printwhat[idx] = NPT_APOLFORCE;
00949         thee->printnarg[idx] = 0;
00950     } else {
00951         Vnm_print(2, "NOsh_parsePRINT:  Undefined keyword %s while parsing \
00952 PRINT section!\n", tok);
00953         return 0;
00954     }
00955
00956     expect = 0;   /* We first expect a calculation ID (0) then an op (1) */
00957
00958     /* Read until we run out of tokens (bad) or hit the "END" keyword (good) */
00959     while (Vio_scanf(sock, "%s", tok) == 1) {
00960
00961         /* The next thing we read is either END or an ARG OP ARG statement */
00962         if (Vstring_strcasecmp(tok, "end") == 0) {
00963             if (expect != 0) {
00964                 (thee->nprint)++;
00965                 (thee->printnarg[idx])++;
00966                 Vnm_print(0, "NOsh: Done parsing PRINT section\n");
00967                 return 1;
00968             } else {
00969                 Vnm_print(2, "NOsh_parsePRINT:  Got premature END to PRINT!\n");
00970                 return 0;
00971             }
```

```
00972         } else {
00973
00974             /* Grab a calculation ID */
00975             if ((sscanf(tok, "%d", &ti) == 1) &&
00976     (Vstring_isdigit(tok) == 1)) {
00977                 if (expect == 0) {
00978                     thee->printcalc[idx][thee->printnarg[idx]] = ti-1;
00979                     expect = 1;
00980                 } else {
00981                     Vnm_print(2, "NOsh_parsePRINT:  Syntax error in PRINT \
00982 section while reading %s!\n", tok);
00983                     return 0;
00984                 }
00985     /* Grab addition operation */
00986             } else if (Vstring_strcasecmp(tok, "+") == 0) {
00987                 if (expect == 1) {
00988                     thee->printop[idx][thee->printnarg[idx]] = 0;
00989                     (thee->printnarg[idx])++;
00990                     expect = 0;
00991                     if (thee->printnarg[idx] >= NOSH_MAXPOP) {
00992                         Vnm_print(2, "NOsh_parsePRINT:  Exceeded max number \
00993 (%d) of arguments for PRINT section!\n",
00994           NOSH_MAXPOP);
00995                         return 0;
00996                     }
00997                 } else {
00998                     Vnm_print(2, "NOsh_parsePRINT:  Syntax error in PRINT \
00999 section while reading %s!\n", tok);
01000                     return 0;
01001                 }
01002     /* Grab subtraction operation */
01003             } else if (Vstring_strcasecmp(tok, "-") == 0) {
01004                 if (expect == 1) {
01005                     thee->printop[idx][thee->printnarg[idx]] = 1;
01006                     (thee->printnarg[idx])++;
01007                     expect = 0;
01008                     if (thee->printnarg[idx] >= NOSH_MAXPOP) {
01009                         Vnm_print(2, "NOsh_parseREAD:  Exceeded max number \
01010 (%d) of arguments for PRINT section!\n",
01011           NOSH_MAXPOP);
01012                         return 0;
01013                     }
01014                 } else {
01015                     Vnm_print(2, "NOsh_parsePRINT:  Syntax error in PRINT \
01016 section while reading %s!\n", tok);
01017                     return 0;
01018                 }
01019     /* Grab a calculation name from elec ID */
01020    } else if (sscanf(tok, "%s", name) == 1) {
01021       if (expect == 0) {
01022     for (ielec=0; ielec<thee->nelec; ielec++) {
01023         if (Vstring_strcasecmp(thee->elecname[ielec], name) == 0) {
01024       thee->printcalc[idx][thee->printnarg[idx]] = ielec;
01025       expect = 1;
01026       break;
01027         }
01028        }
```

```
01029      for (iapol=0; iapol<thee->napol; iapol++) {
01030          if (Vstring_strcasecmp(thee->apolname[iapol], name) == 0) {
01031        thee->printcalc[idx][thee->printnarg[idx]] = iapol;
01032        expect = 1;
01033        break;
01034          }
01035         }
01036      if (expect == 0) {
01037       Vnm_print(2, "No ELEC or APOL statement has been named %s!\n",
01038           name);
01039       return 0;
01040      }
01041                  } else {
01042                      Vnm_print(2, "NOsh_parsePRINT:  Syntax error in PRINT \
01043 section while reading %s!\n", tok);
01044                      return 0;
01045                  }
01046     /* Got bad operation */
01047              } else {
01048                  Vnm_print(2, "NOsh_parsePRINT:  Undefined keyword %s while \
01049 parsing PRINT section!\n", tok);
01050                  return 0;
01051              }
01052          } /* end parse token */
01053
01054     } /* end while */
01055
01056     VJMPERR1(0);
01057
01058     /* We ran out of tokens! */
01059 VERROR1:
01060        Vnm_print(2, "NOsh_parsePRINT:  Ran out of tokens while parsing PRINT \
01061 section!\n");
01062        return 0;
01063
01064 }
01065
01066 VPRIVATE int NOsh_parseELEC(NOsh *thee, Vio *sock) {
01067
01068  NOsh_calc *calc = VNULL;
01069
01070     char tok[VMAX_BUFSIZE];
01071
01072     if (thee == VNULL) {
01073         Vnm_print(2, "NOsh_parseELEC:  Got NULL thee!\n");
01074         return 0;
01075     }
01076
01077     if (sock == VNULL) {
01078         Vnm_print(2, "NOsh_parseELEC:  Got pointer to NULL socket!\n");
01079         return 0;
01080     }
01081
01082     if (thee->parsed) {
01083         Vnm_print(2, "NOsh_parseELEC:  Already parsed an input file!\n");
01084         return 0;
01085     }
```

```
01086
01087     /* Get a pointer to the latest ELEC calc object and update the ELEC
01088   statement number */
01089     if (thee->nelec >= NOSH_MAXCALC) {
01090         Vnm_print(2, "NOsh:  Too many electrostatics calculations in this \
01091 run!\n");
01092         Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
01093       NOSH_MAXCALC);
01094         return 1;
01095     }
01096
01097     /* The next token HAS to be the method OR "name" */
01098  if (Vio_scanf(sock, "%s", tok) == 1) {
01099      if (Vstring_strcasecmp(tok, "name") == 0) {
01100       Vio_scanf(sock, "%s", tok);
01101             strncpy(thee->elecname[thee->nelec], tok, VMAX_ARGLEN);
01102    if (Vio_scanf(sock, "%s", tok) != 1) {
01103     Vnm_print(2, "NOsh_parseELEC:  Ran out of tokens while reading \
01104 ELEC section!\n");
01105     return 0;
01106    }
01107   }
01108   if (Vstring_strcasecmp(tok, "mg-manual") == 0) {
01109    thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_MG);
01110    calc = thee->elec[thee->nelec];
01111    (thee->nelec)++;
01112    calc->mgparm->type = MCT_MANUAL;
01113       return NOsh_parseMG(thee, sock, calc);
01114   } else if (Vstring_strcasecmp(tok, "mg-auto") == 0) {
01115    thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_MG);
01116    calc = thee->elec[thee->nelec];
01117    (thee->nelec)++;
01118    calc->mgparm->type = MCT_AUTO;
01119       return NOsh_parseMG(thee, sock, calc);
01120   } else if (Vstring_strcasecmp(tok, "mg-para") == 0) {
01121    thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_MG);
01122    calc = thee->elec[thee->nelec];
01123    (thee->nelec)++;
01124    calc->mgparm->type = MCT_PARALLEL;
01125       return NOsh_parseMG(thee, sock, calc);
01126   } else if (Vstring_strcasecmp(tok, "mg-dummy") == 0) {
01127    thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_MG);
01128    calc = thee->elec[thee->nelec];
01129    (thee->nelec)++;
01130    calc->mgparm->type = MCT_DUMMY;
01131       return NOsh_parseMG(thee, sock, calc);
01132   } else if (Vstring_strcasecmp(tok, "fe-manual") == 0) {
01133    thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_FEM);
01134    calc = thee->elec[thee->nelec];
01135    (thee->nelec)++;
01136    calc->femparm->type = FCT_MANUAL;
01137       return NOsh_parseFEM(thee, sock, calc);
01138   } else {
01139       Vnm_print(2, "NOsh_parseELEC: The method (\"mg\" or \"fem\") or \
01140 \"name\" must be the first keyword in the ELEC section\n");
01141    return 0;
01142   }
```

```
01143  }
01144
01145     Vnm_print(2, "NOsh_parseELEC:  Ran out of tokens while reading ELEC section!\
      n");
01146     return 0;
01147
01148 }
01149
01150 VPRIVATE int NOsh_parseAPOLAR(NOsh *thee, Vio *sock) {
01151
01152  NOsh_calc *calc = VNULL;
01153
01154     char tok[VMAX_BUFSIZE];
01155
01156     if (thee == VNULL) {
01157         Vnm_print(2, "NOsh_parseAPOLAR:  Got NULL thee!\n");
01158         return 0;
01159     }
01160
01161     if (sock == VNULL) {
01162         Vnm_print(2, "NOsh_parseAPOLAR:  Got pointer to NULL socket!\n");
01163         return 0;
01164     }
01165
01166     if (thee->parsed) {
01167         Vnm_print(2, "NOsh_parseAPOLAR:  Already parsed an input file!\n");
01168         return 0;
01169     }
01170
01171  /* Get a pointer to the latest ELEC calc object and update the ELEC
01172   statement number */
01173     if (thee->napol >= NOSH_MAXCALC) {
01174         Vnm_print(2, "NOsh:  Too many non-polar calculations in this \
01175 run!\n");
01176         Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
01177       NOSH_MAXCALC);
01178         return 1;
01179     }
01180
01181  /* The next token HAS to be the method OR "name" */
01182  if (Vio_scanf(sock, "%s", tok) == 1) {
01183      if (Vstring_strcasecmp(tok, "name") == 0) {
01184       Vio_scanf(sock, "%s", tok);
01185             strncpy(thee->apolname[thee->napol], tok, VMAX_ARGLEN);
01186
01187    /* Parse the non-polar parameters */
01188    thee->apol[thee->napol] = NOsh_calc_ctor(NCT_APOL);
01189    calc = thee->apol[thee->napol];
01190    (thee->napol)++;
01191    return NOsh_parseAPOL(thee, sock, calc);
01192
01193    if (Vio_scanf(sock, "%s", tok) != 1) {
01194     Vnm_print(2, "NOsh_parseAPOLAR:  Ran out of tokens while reading \
01195 APOLAR section!\n");
01196     return 0;
01197    }
01198   }
```

```
01199  }
01200
01201  return 1;
01202
01203 }
01204
01205 VPUBLIC int NOsh_setupElecCalc(
01206          NOsh *thee,
01207          Valist *alist[NOSH_MAXMOL]
01208          ) {
01209  int ielec, imol, i;
01210  NOsh_calc *elec = VNULL;
01211  MGparm *mgparm = VNULL;
01212  Valist *mymol = VNULL;
01213
01214  VASSERT(thee != VNULL);
01215  for (imol=0; imol<thee->nmol; imol++) {
01216   thee->alist[imol] = alist[imol];
01217  }
01218
01219
01220  for (ielec=0; ielec<(thee->nelec); ielec++) {
01221   /* Unload the calculation object containing the ELEC information */
01222   elec = thee->elec[ielec];
01223
01224   if (((thee->ndiel != 0) || (thee->nkappa != 0) ||
01225     (thee->ncharge != 0) || (thee->npot != 0)) &&
01226     (elec->pbeparm->calcforce != PCF_NO)) {
01227    Vnm_print(2, "NOsh_setupElecCalc:  Calculation of forces disabled because surf
      ace \
01228 map is used!\n");
01229    elec->pbeparm->calcforce = PCF_NO;
01230   }
01231
01232   /* Setup the calculation */
01233   switch (elec->calctype) {
01234    case NCT_MG:
01235     /* Center on the molecules, if requested */
01236     mgparm = elec->mgparm;
01237     VASSERT(mgparm != VNULL);
01238     if (elec->mgparm->cmeth == MCM_MOLECULE) {
01239      VASSERT(mgparm->centmol >= 0);
01240      VASSERT(mgparm->centmol < thee->nmol);
01241      mymol = thee->alist[mgparm->centmol];
01242      VASSERT(mymol != VNULL);
01243      for (i=0; i<3; i++) {
01244       mgparm->center[i] = mymol->center[i];
01245      }
01246     }
01247     if (elec->mgparm->fcmeth == MCM_MOLECULE) {
01248      VASSERT(mgparm->fcentmol >= 0);
01249      VASSERT(mgparm->fcentmol < thee->nmol);
01250      mymol = thee->alist[mgparm->fcentmol];
01251      VASSERT(mymol != VNULL);
01252      for (i=0; i<3; i++) {
01253       mgparm->fcenter[i] = mymol->center[i];
01254      }
```

```
01255     }
01256     if (elec->mgparm->ccmeth == MCM_MOLECULE) {
01257      VASSERT(mgparm->ccentmol >= 0);
01258      VASSERT(mgparm->ccentmol < thee->nmol);
01259      mymol = thee->alist[mgparm->ccentmol];
01260      VASSERT(mymol != VNULL);
01261      for (i=0; i<3; i++) {
01262       mgparm->ccenter[i] = mymol->center[i];
01263      }
01264     }
01265     NOsh_setupCalcMG(thee, elec);
01266     break;
01267    case NCT_FEM:
01268     NOsh_setupCalcFEM(thee, elec);
01269     break;
01270    default:
01271     Vnm_print(2, "NOsh_setupCalc:  Invalid calculation type (%d)!\n",
01272         elec->calctype);
01273     return 0;
01274   }
01275
01276   /* At this point, the most recently-created NOsh_calc object should be the
01277    one we use for results for this ELEC statement.  Assign it. */
01278   /* Associate ELEC statement with the calculation */
01279   thee->elec2calc[ielec] = thee->ncalc-1;
01280   Vnm_print(0, "NOsh_setupCalc:  Mapping ELEC statement %d (%d) to \
01281 calculation %d (%d)\n", ielec, ielec+1, thee->elec2calc[ielec],
01282       thee->elec2calc[ielec]+1);
01283  }
01284
01285  return 1;
01286 }
01287
01288 VPUBLIC int NOsh_setupApolCalc(
01289           NOsh *thee,
01290           Valist *alist[NOSH_MAXMOL]
01291           ) {
01292  int iapol, imol;
01293  int doCalc = ACD_NO;
01294  NOsh_calc *calc = VNULL;
01295
01296  VASSERT(thee != VNULL);
01297  for (imol=0; imol<thee->nmol; imol++) {
01298   thee->alist[imol] = alist[imol];
01299  }
01300
01301  for (iapol=0; iapol<(thee->napol); iapol++) {
01302   /* Unload the calculation object containing the APOL information */
01303   calc = thee->apol[iapol];
01304
01305   /* Setup the calculation */
01306   switch (calc->calctype) {
01307    case NCT_APOL:
01308     NOsh_setupCalcAPOL(thee, calc);
01309     doCalc = ACD_YES;
01310     break;
01311    default:
```

```
01312     Vnm_print(2, "NOsh_setupCalc:  Invalid calculation type (%d)!\n", calc->
      calctype);
01313     return ACD_ERROR;
01314   }
01315   /* At this point, the most recently-created NOsh_calc object should be the
01316    one we use for results for this APOL statement.  Assign it. */
01317   /* Associate APOL statement with the calculation */
01318   thee->apol2calc[iapol] = thee->ncalc-1;
01319   Vnm_print(0, "NOsh_setupCalc:  Mapping APOL statement %d (%d) to calculation %d
      (%d)\n", iapol, iapol+1, thee->apol2calc[iapol], thee->apol2calc[iapol]+1);
01320  }
01321
01322  if(doCalc == ACD_YES){
01323   return ACD_YES;
01324  }else{
01325   return ACD_NO;
01326  }
01327 }
01328
01329 VPUBLIC int NOsh_parseMG(
01330        NOsh *thee,
01331        Vio *sock,
01332        NOsh_calc *elec
01333        ) {
01334
01335  char tok[VMAX_BUFSIZE];
01336  MGparm *mgparm = VNULL;
01337  PBEparm *pbeparm = VNULL;
01338  int rc;
01339
01340  /* Check the arguments */
01341  if (thee == VNULL) {
01342   Vnm_print(2, "NOsh:  Got NULL thee!\n");
01343   return 0;
01344  }
01345  if (sock == VNULL) {
01346   Vnm_print(2, "NOsh:  Got pointer to NULL socket!\n");
01347   return 0;
01348  }
01349  if (elec == VNULL) {
01350   Vnm_print(2, "NOsh:  Got pointer to NULL elec object!\n");
01351   return 0;
01352  }
01353  mgparm = elec->mgparm;
01354  if (mgparm == VNULL) {
01355   Vnm_print(2, "NOsh:  Got pointer to NULL mgparm object!\n");
01356   return 0;
01357  }
01358  pbeparm = elec->pbeparm;
01359  if (pbeparm == VNULL) {
01360   Vnm_print(2, "NOsh:  Got pointer to NULL pbeparm object!\n");
01361   return 0;
01362  }
01363
01364  Vnm_print(0, "NOsh_parseMG: Parsing parameters for MG calculation\n");
01365
01366  /* Parallel stuff */
```

```
01367  if (mgparm->type == MCT_PARALLEL) {
01368   mgparm->proc_rank = thee->proc_rank;
01369   mgparm->proc_size = thee->proc_size;
01370   mgparm->setrank = 1;
01371   mgparm->setsize = 1;
01372  }
01373
01374
01375  /* Start snarfing tokens from the input stream */
01376  rc = 1;
01377  while (Vio_scanf(sock, "%s", tok) == 1) {
01378
01379   Vnm_print(0, "NOsh_parseMG:  Parsing %s...\n", tok);
01380
01381   /* See if it's an END token */
01382   if (Vstring_strcasecmp(tok, "end") == 0) {
01383    mgparm->parsed = 1;
01384    pbeparm->parsed = 1;
01385    rc = 1;
01386    break;
01387   }
01388
01389   /* Pass the token through a series of parsers */
01390   rc = PBEparm_parseToken(pbeparm, tok, sock);
01391   if (rc == -1) {
01392    Vnm_print(0, "NOsh_parseMG:  parsePBE error!\n");
01393    break;
01394   } else if (rc == 0) {
01395    /* Pass the token to the generic MG parser */
01396    rc = MGparm_parseToken(mgparm, tok, sock);
01397    if (rc == -1) {
01398     Vnm_print(0, "NOsh_parseMG:  parseMG error!\n");
01399     break;
01400    } else if (rc == 0) {
01401     /* We ran out of parsers! */
01402     Vnm_print(2, "NOsh:  Unrecognized keyword: %s\n", tok);
01403     break;
01404    }
01405   }
01406  }
01407
01408  /* Handle various errors arising in the token-snarfing loop -- these all
01409   just result in simple returns right now */
01410  if (rc == -1) return 0;
01411  if (rc == 0) return 0;
01412
01413  /* Check the status of the parameter objects */
01414  if ((MGparm_check(mgparm) == VRC_FAILURE) || (!PBEparm_check(pbeparm))) {
01415   Vnm_print(2, "NOsh:  MG parameters not set correctly!\n");
01416   return 0;
01417  }
01418
01419  return 1;
01420 }
01421
01422 VPRIVATE int NOsh_setupCalcMG(
01423          NOsh *thee,
```

```
01424           NOsh_calc *calc
01425           ) {
01426
01427   MGparm *mgparm = VNULL;
01428
01429   VASSERT(thee != VNULL);
01430   VASSERT(calc != VNULL);
01431   mgparm = calc->mgparm;
01432   VASSERT(mgparm != VNULL);
01433
01434
01435   /* Now we're ready to whatever sorts of post-processing operations that are
01436    necessary for the various types of calculations */
01437   switch (mgparm->type) {
01438    case MCT_MANUAL:
01439     return NOsh_setupCalcMGMANUAL(thee, calc);
01440    case MCT_DUMMY:
01441     return NOsh_setupCalcMGMANUAL(thee, calc);
01442    case MCT_AUTO:
01443     return NOsh_setupCalcMGAUTO(thee, calc);
01444    case MCT_PARALLEL:
01445     return NOsh_setupCalcMGPARA(thee, calc);
01446    default:
01447     Vnm_print(2, "NOsh_setupCalcMG:  undefined MG calculation type (%d)!\n",
01448         mgparm->type);
01449     return 0;
01450   }
01451
01452   /* Shouldn't get here */
01453   return 0;
01454 }
01455
01456 VPRIVATE int NOsh_setupCalcFEM(
01457           NOsh *thee,
01458           NOsh_calc *calc
01459           ) {
01460
01461   VASSERT(thee != VNULL);
01462   VASSERT(calc != VNULL);
01463   VASSERT(calc->femparm != VNULL);
01464
01465   /* Now we're ready to whatever sorts of post-processing operations that are
01466    * necessary for the various types of calculations */
01467   switch (calc->femparm->type) {
01468    case FCT_MANUAL:
01469     return NOsh_setupCalcFEMANUAL(thee, calc);
01470    default:
01471     Vnm_print(2, "NOsh_parseFEM:  unknown calculation type (%d)!\n",
01472         calc->femparm->type);
01473     return 0;
01474   }
01475
01476   /* Shouldn't get here */
01477   return 0;
01478 }
01479
01480
```

```
01481 VPRIVATE int NOsh_setupCalcMGMANUAL(
01482            NOsh *thee,
01483            NOsh_calc *elec
01484            ) {
01485
01486  MGparm *mgparm = VNULL;
01487  PBEparm *pbeparm = VNULL;
01488  NOsh_calc *calc = VNULL;
01489
01490  if (thee == VNULL) {
01491   Vnm_print(2, "NOsh_setupCalcMGMANUAL:  Got NULL thee!\n");
01492   return 0;
01493  }
01494  if (elec == VNULL) {
01495   Vnm_print(2, "NOsh_setupCalcMGMANUAL:  Got NULL calc!\n");
01496   return 0;
01497  }
01498  mgparm = elec->mgparm;
01499  if (mgparm == VNULL) {
01500   Vnm_print(2, "NOsh_setupCalcMGMANUAL:  Got NULL mgparm -- was this calculation
     \
01501 set up?\n");
01502   return 0;
01503  }
01504  pbeparm = elec->pbeparm;
01505  if (pbeparm == VNULL) {
01506   Vnm_print(2, "NOsh_setupCalcMGMANUAL:  Got NULL pbeparm -- was this calculation
     \
01507 set up?\n");
01508   return 0;
01509  }
01510
01511  /* Set up missing MG parameters */
01512  if (mgparm->setgrid == 0) {
01513   VASSERT(mgparm->setglen);
01514   mgparm->grid[0] = mgparm->glen[0]/((double)(mgparm->dime[0]-1));
01515   mgparm->grid[1] = mgparm->glen[1]/((double)(mgparm->dime[1]-1));
01516   mgparm->grid[2] = mgparm->glen[2]/((double)(mgparm->dime[2]-1));
01517  }
01518  if (mgparm->setglen == 0) {
01519   VASSERT(mgparm->setgrid);
01520   mgparm->glen[0] = mgparm->grid[0]*((double)(mgparm->dime[0]-1));
01521   mgparm->glen[1] = mgparm->grid[1]*((double)(mgparm->dime[1]-1));
01522   mgparm->glen[2] = mgparm->grid[2]*((double)(mgparm->dime[2]-1));
01523  }
01524
01525  /* Check to see if he have any room left for this type of calculation, if
01526   so: set the calculation type, update the number of calculations of this type,
01527   and parse the rest of the section */
01528  if (thee->ncalc >= NOSH_MAXCALC) {
01529   Vnm_print(2, "NOsh:  Too many calculations in this run!\n");
01530   Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
01531       NOSH_MAXCALC);
01532   return 0;
01533  }
01534
01535     /* Get the next calculation object and increment the number of calculations *
```

```
     /
01536  thee->calc[thee->ncalc] = NOsh_calc_ctor(NCT_MG);
01537  calc = thee->calc[thee->ncalc];
01538  (thee->ncalc)++;
01539
01540
01541
01542  /* Copy over contents of ELEC */
01543  NOsh_calc_copy(calc, elec);
01544
01545
01546     return 1;
01547 }
01548
01549 VPUBLIC int NOsh_setupCalcMGAUTO(
01550          NOsh *thee,
01551          NOsh_calc *elec
01552          ) {
01553
01554  NOsh_calc *calcf = VNULL;
01555  NOsh_calc *calcc = VNULL;
01556  double fgrid[3], cgrid[3];
01557  double d[3], minf[3], maxf[3], minc[3], maxc[3];
01558  double redfrac, redrat[3], td;
01559  int ifocus, nfocus, tnfocus[3];
01560  int j;
01561  int icalc;
01562  int dofix;
01563
01564  /* A comment about the coding style in this function.  I use lots and lots
01565   and lots of pointer deferencing.  I could (and probably should) save
01566   these in temporary variables.  However, since there are so many MGparm,
01567   etc. and NOsh_calc, etc. objects running around in this function, the
01568   current scheme is easiest to debug. */
01569
01570
01571  if (thee == VNULL) {
01572   Vnm_print(2, "NOsh_setupCalcMGAUTO:  Got NULL thee!\n");
01573   return 0;
01574  }
01575  if (elec == VNULL) {
01576   Vnm_print(2, "NOsh_setupCalcMGAUTO:  Got NULL elec!\n");
01577   return 0;
01578  }
01579  if (elec->mgparm == VNULL) {
01580   Vnm_print(2, "NOsh_setupCalcMGAUTO:  Got NULL mgparm!\n");
01581   return 0;
01582  }
01583  if (elec->pbeparm == VNULL) {
01584   Vnm_print(2, "NOsh_setupCalcMGAUTO:  Got NULL pbeparm!\n");
01585   return 0;
01586  }
01587
01588  Vnm_print(0, "NOsh_setupCalcMGAUTO(%s, %d):  coarse grid center = %g %g %g\n",
01589      __FILE__, __LINE__,
01590      elec->mgparm->ccenter[0],
01591      elec->mgparm->ccenter[1],
```

```
01592       elec->mgparm->ccenter[2]);
01593   Vnm_print(0, "NOsh_setupCalcMGAUTO(%s, %d):  fine grid center = %g %g %g\n",
01594       __FILE__, __LINE__,
01595       elec->mgparm->fcenter[0],
01596       elec->mgparm->fcenter[1],
01597       elec->mgparm->fcenter[2]);
01598
01599   /* Calculate the grid spacing on the coarse and fine levels */
01600   for (j=0; j<3; j++) {
01601    cgrid[j] = (elec->mgparm->cglen[j])/((double)(elec->mgparm->dime[j]-1));
01602    fgrid[j] = (elec->mgparm->fglen[j])/((double)(elec->mgparm->dime[j]-1));
01603    d[j] = elec->mgparm->fcenter[j] - elec->mgparm->ccenter[j];
01604   }
01605   Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  Coarse grid spacing = %g, %g, %g\n
      ",
01606       __FILE__, __LINE__, cgrid[0], cgrid[1], cgrid[2]);
01607   Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  Fine grid spacing = %g, %g, %g\n",

01608       __FILE__, __LINE__, fgrid[0], fgrid[1], fgrid[2]);
01609   Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  Displacement between fine and \
01610 coarse grids = %g, %g, %g\n", __FILE__, __LINE__, d[0], d[1], d[2]);
01611
01612   /* Now calculate the number of focusing levels, never reducing the grid
01613    spacing by more than redfrac at each level */
01614   for (j=0; j<3; j++) {
01615    if (fgrid[j]/cgrid[j] < VREDFRAC) {
01616     redfrac = fgrid[j]/cgrid[j];
01617     td = log(redfrac)/log(VREDFRAC);
01618     tnfocus[j] = (int)ceil(td) + 1;
01619    } else tnfocus[j] = 2;
01620   }
01621   nfocus = VMAX2(VMAX2(tnfocus[0], tnfocus[1]), tnfocus[2]);
01622
01623   /* Now set redrat to the actual value by which the grid spacing is reduced
01624    at each level of focusing */
01625   for (j=0; j<3; j++) {
01626    redrat[j] = VPOW((fgrid[j]/cgrid[j]), 1.0/((double)nfocus-1.0));
01627   }
01628   Vnm_print(0, "NOsh:  %d levels of focusing with %g, %g, %g reductions\n",
01629       nfocus, redrat[0], redrat[1], redrat[2]);
01630
01631   /* Now that we know how many focusing levels to use, we're ready to set up
01632    the parameter objects */
01633   if (nfocus > (NOSH_MAXCALC-(thee->ncalc))) {
01634    Vnm_print(2, "NOsh:  Require more calculations than max (%d)!\n",
01635        NOSH_MAXCALC);
01636    return 0;
01637   }
01638
01639   for (ifocus=0; ifocus<nfocus; ifocus++) {
01640
01641    /* Generate the new calc object */
01642    icalc = thee->ncalc;
01643    thee->calc[icalc] = NOsh_calc_ctor(NCT_MG);
01644    (thee->ncalc)++;
01645
01646    /* This is the _current_ NOsh_calc object */
```

```
01647   calcf = thee->calc[icalc];
01648   /* This is the _previous_ Nosh_calc object */
01649   if (ifocus != 0) {
01650    calcc = thee->calc[icalc-1];
01651   } else {
01652    calcc = VNULL;
01653   }
01654
01655   /* Copy over most of the parameters from the ELEC object */
01656   NOsh_calc_copy(calcf, elec);
01657
01658   /* Set up the grid lengths and spacings */
01659   if (ifocus == 0) {
01660    for (j=0; j<3; j++) {
01661     calcf->mgparm->grid[j] = cgrid[j];
01662     calcf->mgparm->glen[j] = elec->mgparm->cglen[j];
01663    }
01664   } else {
01665    for (j=0; j<3; j++) {
01666     calcf->mgparm->grid[j] = redrat[j]*(calcc->mgparm->grid[j]);
01667                 calcf->mgparm->glen[j] = redrat[j]*(calcc->mgparm->glen[j]);
01668               }
01669           }
01670           calcf->mgparm->setgrid = 1;
01671           calcf->mgparm->setglen = 1;
01672
01673   /* Get centers and centering method from coarse and fine meshes */
01674         if (ifocus == 0) {
01675    calcf->mgparm->cmeth = elec->mgparm->ccmeth;
01676    calcf->mgparm->centmol = elec->mgparm->ccentmol;
01677    for (j=0; j<3; j++) {
01678     calcf->mgparm->center[j] = elec->mgparm->ccenter[j];
01679    }
01680   } else if (ifocus == (nfocus-1)) {
01681    calcf->mgparm->cmeth = elec->mgparm->fcmeth;
01682    calcf->mgparm->centmol = elec->mgparm->fcentmol;
01683    for (j=0; j<3; j++) {
01684     calcf->mgparm->center[j] = elec->mgparm->fcenter[j];
01685    }
01686   } else {
01687    calcf->mgparm->cmeth = MCM_FOCUS;
01688    /* TEMPORARILY move the current grid center
01689    to the fine grid center.  In general, this will move portions of
01690    the current mesh off the immediately-coarser mesh.  We'll fix that
01691    in the next step. */
01692    for (j=0; j<3; j++) {
01693     calcf->mgparm->center[j] = elec->mgparm->fcenter[j];
01694    }
01695   }
01696
01697
01698   /* As mentioned above, it is highly likely that the previous "jump"
01699    to the fine grid center put portions of the current mesh off the
01700    previous (coarser) mesh.  Fix this by displacing the current mesh
01701    back onto the previous coarser mesh.  */
01702   if (ifocus != 0) {
01703    Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  starting mesh \
```

```
01704 repositioning.\n", __FILE__, __LINE__);
01705    Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  coarse mesh center = \
01706 %g %g %g\n", __FILE__, __LINE__,
01707          calcc->mgparm->center[0],
01708            calcc->mgparm->center[1],
01709            calcc->mgparm->center[2]);
01710    Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  coarse mesh upper corner = \
01711 %g %g %g\n", __FILE__, __LINE__,
01712          calcc->mgparm->center[0]+0.5*(calcc->mgparm->glen[0]),
01713            calcc->mgparm->center[1]+0.5*(calcc->mgparm->glen[1]),
01714            calcc->mgparm->center[2]+0.5*(calcc->mgparm->glen[2]));
01715    Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  coarse mesh lower corner = \
01716 %g %g %g\n", __FILE__, __LINE__,
01717          calcc->mgparm->center[0]-0.5*(calcc->mgparm->glen[0]),
01718            calcc->mgparm->center[1]-0.5*(calcc->mgparm->glen[1]),
01719            calcc->mgparm->center[2]-0.5*(calcc->mgparm->glen[2]));
01720    Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  initial fine mesh upper corner =
      \
01721 %g %g %g\n", __FILE__, __LINE__,
01722          calcf->mgparm->center[0]+0.5*(calcf->mgparm->glen[0]),
01723            calcf->mgparm->center[1]+0.5*(calcf->mgparm->glen[1]),
01724            calcf->mgparm->center[2]+0.5*(calcf->mgparm->glen[2]));
01725    Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  initial fine mesh lower corner =
      \
01726 %g %g %g\n", __FILE__, __LINE__,
01727          calcf->mgparm->center[0]-0.5*(calcf->mgparm->glen[0]),
01728            calcf->mgparm->center[1]-0.5*(calcf->mgparm->glen[1]),
01729            calcf->mgparm->center[2]-0.5*(calcf->mgparm->glen[2]));
01730    for (j=0; j<3; j++) {
01731     /* Check if we've fallen off of the lower end of the mesh */
01732     dofix = 0;
01733     minf[j] = calcf->mgparm->center[j]
01734      - 0.5*(calcf->mgparm->glen[j]);
01735     minc[j] = calcc->mgparm->center[j]
01736      - 0.5*(calcc->mgparm->glen[j]);
01737     d[j] = minc[j] - minf[j];
01738     if (d[j] >= VSMALL) {
01739      if (ifocus == (nfocus-1)) {
01740       Vnm_print(2, "NOsh_setupCalcMGAUTO:  Error!  Finest \
01741 mesh has fallen off the coarser meshes!\n");
01742       Vnm_print(2, "NOsh_setupCalcMGAUTO:  difference in min %d-\
01743 direction = %g\n", j, d[j]);
01744       Vnm_print(2, "NOsh_setupCalcMGAUTO:  min fine = %g %g %g\n",
01745           minf[0], minf[1], minf[2]);
01746       Vnm_print(2, "NOsh_setupCalcMGAUTO:  min coarse = %g %g %g\n",
01747           minc[0], minc[1], minc[2]);
01748       VASSERT(0);
01749      } else {
01750       Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  ifocus = %d, \
01751 fixing mesh min violation (%g in %d-direction).\n", __FILE__, __LINE__, ifocus,
01752           d[j], j);
01753       calcf->mgparm->center[j] += d[j];
01754       dofix = 1;
01755      }
01756     }
01757     /* Check if we've fallen off of the upper end of the mesh */
01758     maxf[j] = calcf->mgparm->center[j] \
```

```
01759       + 0.5*(calcf->mgparm->glen[j]);
01760      maxc[j] = calcc->mgparm->center[j] \
01761       + 0.5*(calcc->mgparm->glen[j]);
01762      d[j] = maxf[j] - maxc[j];
01763      if (d[j] >= VSMALL) {
01764       if (ifocus == (nfocus-1)) {
01765        Vnm_print(2, "NOsh_setupCalcMGAUTO:  Error!  Finest \
01766 mesh has fallen off the coarser meshes!\n");
01767        Vnm_print(2, "NOsh_setupCalcMGAUTO:  difference in %d-\
01768 direction = %g\n", j, d[j]);
01769        VASSERT(0);
01770       } else {
01771        /* If we already fixed the lower boundary and we now need
01772        to fix the upper boundary, we have a serious problem. */
01773        if (dofix) {
01774         Vnm_print(2, "NOsh_setupCalcMGAUTO:  Error!  Both \
01775 ends of the finer mesh do not fit in the bigger mesh!\n");
01776         VASSERT(0);
01777        }
01778        Vnm_print(0, "NOsh_setupCalcMGAUTO(%s, %d):  ifocus = %d, \
01779 fixing mesh max violation (%g in %d-direction).\n", __FILE__, __LINE__, ifocus,
01780            d[j], j);
01781        calcf->mgparm->center[j] -= d[j];
01782        dofix = 1;
01783       }
01784      }
01785     }
01786     Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  final fine mesh upper corner = \

01787 %g %g %g\n", __FILE__, __LINE__,
01788         calcf->mgparm->center[0]+0.5*(calcf->mgparm->glen[0]),
01789           calcf->mgparm->center[1]+0.5*(calcf->mgparm->glen[1]),
01790           calcf->mgparm->center[2]+0.5*(calcf->mgparm->glen[2]));
01791     Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  final fine mesh lower corner = \

01792 %g %g %g\n", __FILE__, __LINE__,
01793         calcf->mgparm->center[0]-0.5*(calcf->mgparm->glen[0]),
01794           calcf->mgparm->center[1]-0.5*(calcf->mgparm->glen[1]),
01795           calcf->mgparm->center[2]-0.5*(calcf->mgparm->glen[2]));
01796   }
01797
01798   /* Finer levels have focusing boundary conditions */
01799   if (ifocus != 0) calcf->pbeparm->bcfl = BCFL_FOCUS;
01800
01801   /* Only the finest level handles I/O and needs to worry about disjoint
01802    partitioning */
01803   if (ifocus != (nfocus-1)) calcf->pbeparm->numwrite = 0;
01804
01805   /* Reset boundary flags for everything except parallel focusing */
01806         if (calcf->mgparm->type != MCT_PARALLEL)  {
01807    Vnm_print(0, "NOsh_setupMGAUTO:  Resetting boundary flags\n");
01808    for (j=0; j<6; j++) calcf->mgparm->partDisjOwnSide[j] = 0;
01809    for (j=0; j<3; j++) {
01810     calcf->mgparm->partDisjCenter[j] = 0;
01811     calcf->mgparm->partDisjLength[j] = calcf->mgparm->glen[j];
01812    }
01813         }
```

```
01814
01815
01816         calcf->mgparm->parsed = 1;
01817     }
01818
01819
01820     return 1;
01821 }
01822
01823 /* Author:   Nathan Baker and Todd Dolinsky */
01824 VPUBLIC int NOsh_setupCalcMGPARA(
01825          NOsh *thee,
01826          NOsh_calc *elec
01827          ) {
01828
01829  /* NEW (25-Jul-2006):  This code should produce modify the ELEC statement
01830  and pass it on to MGAUTO for further processing. */
01831
01832  MGparm *mgparm = VNULL;
01833  double ofrac;
01834  double hx, hy, hzed;
01835  double xofrac, yofrac, zofrac;
01836  int rank, size, npx, npy, npz, nproc, ip, jp, kp;
01837  int xeffGlob, yeffGlob, zeffGlob, xDisj, yDisj, zDisj;
01838  int xigminDisj, xigmaxDisj, yigminDisj, yigmaxDisj, zigminDisj, zigmaxDisj;
01839  int xigminOlap, xigmaxOlap, yigminOlap, yigmaxOlap, zigminOlap, zigmaxOlap;
01840  int xOlapReg, yOlapReg, zOlapReg;
01841  double xlenDisj, ylenDisj, zlenDisj;
01842  double xcentDisj, ycentDisj, zcentDisj;
01843  double xcentOlap, ycentOlap, zcentOlap;
01844  double xlenOlap, ylenOlap, zlenOlap;
01845  double xminOlap, xmaxOlap, yminOlap, ymaxOlap, zminOlap, zmaxOlap;
01846  double xminDisj, xmaxDisj, yminDisj, ymaxDisj, zminDisj, zmaxDisj;
01847  double xcent, ycent, zcent;
01848
01849  /* Grab some useful variables */
01850  VASSERT(thee != VNULL);
01851  VASSERT(elec != VNULL);
01852  mgparm = elec->mgparm;
01853  VASSERT(mgparm != VNULL);
01854
01855  /* Grab some useful variables */
01856     ofrac = mgparm->ofrac;
01857     npx = mgparm->pdime[0];
01858     npy = mgparm->pdime[1];
01859     npz = mgparm->pdime[2];
01860     nproc = npx*npy*npz;
01861
01862  /* If this is not an asynchronous calculation, then we need to make sure we
01863   have all the necessary MPI information */
01864     if (mgparm->setasync == 0) {
01865
01866 #ifndef HAVE_MPI_H
01867
01868         Vnm_tprint(2, "NOsh_setupCalcMGPARA:  Oops!  You're trying to perform \
01869 an 'mg-para' (parallel) calculation\n");
01870         Vnm_tprint(2, "NOsh_setupCalcMGPARA:  with a version of APBS that wasn't
```

```
      \
01871 compiled with MPI!\n");
01872   Vnm_tprint(2, "NOsh_setupCalcMGPARA:  Perhaps you meant to use the \
01873 'async' flag?\n");
01874         Vnm_tprint(2, "NOsh_setupCalcMGPARA:  Bailing out!\n");
01875
01876         return 0;
01877
01878 #endif
01879
01880   rank = thee->proc_rank;
01881   size = thee->proc_size;
01882   Vnm_print(0, "NOsh_setupCalcMGPARA:  Hello from processor %d of %d\n", rank,
01883       size);
01884
01885   /* Check to see if we have too many processors.  If so, then simply set
01886    this processor to duplicating the work of processor 0. */
01887   if (rank > (nproc-1)) {
01888    Vnm_print(2, "NOsh_setupMGPARA:  There are more processors available than\
01889 the %d you requested.\n", nproc);
01890    Vnm_print(2, "NOsh_setupMGPARA:  Eliminating processor %d\n", rank);
01891    thee->bogus = 1;
01892    rank = 0;
01893   }
01894
01895   /* Check to see if we have too few processors.  If so, this is a fatal
01896    error. */
01897   if (size < nproc) {
01898    Vnm_print(2, "NOsh_setupMGPARA:  There are too few processors (%d) to \
01899 satisfy requirements (%d)\n", size, nproc);
01900    return 0;
01901   }
01902
01903   Vnm_print(0, "NOsh_setupMGPARA:  Hello (again) from processor %d of %d\n",
01904       rank, size);
01905
01906     } else { /* Setting up for an asynchronous calculation. */
01907
01908   rank = mgparm->async;
01909
01910   thee->ispara = 1;
01911   thee->proc_rank = rank;
01912
01913         /* Check to see if the async id is greater than the number of
01914   * processors.  If so, this is a fatal error. */
01915         if (rank > (nproc-1)) {
01916             Vnm_print(2, "NOsh_setupMGPARA:  The processor id you requested (%d)
      \
01917 is not within the range of processors available (0-%d)\n", rank, (nproc-1));
01918             return 0;
01919         }
01920     }
01921
01922     /* Calculate the processor's coordinates in the processor grid */
01923  kp = (int)floor(rank/(npx*npy));
01924  jp = (int)floor((rank-kp*npx*npy)/npx);
01925  ip = rank - kp*npx*npy - jp*npx;
```

```
01926  Vnm_print(0, "NOsh_setupMGPARA:  Hello world from PE (%d, %d, %d)\n",
01927      ip, jp, kp);
01928
01929  /* Calculate effective overlap fractions for uneven processor distributions */
01930  if (npx == 1) xofrac = 0.0;
01931  else xofrac = ofrac;
01932  if (npy == 1) yofrac = 0.0;
01933  else yofrac = ofrac;
01934  if (npz == 1) zofrac = 0.0;
01935  else zofrac = ofrac;
01936
01937     /* Calculate the global grid size and spacing */
01938     xDisj = (int)VFLOOR(mgparm->dime[0]/(1 + 2*xofrac) + 0.5);
01939     xeffGlob = npx*xDisj;
01940     hx = mgparm->fglen[0]/(double)(xeffGlob-1);
01941     yDisj = (int)VFLOOR(mgparm->dime[1]/(1 + 2*yofrac) + 0.5);
01942     yeffGlob = npy*yDisj;
01943     hy = mgparm->fglen[1]/(double)(yeffGlob-1);
01944     zDisj = (int)VFLOOR(mgparm->dime[2]/(1 + 2*zofrac) + 0.5);
01945     zeffGlob = npz*zDisj;
01946     hzed = mgparm->fglen[2]/(double)(zeffGlob-1);
01947     Vnm_print(0, "NOsh_setupMGPARA:  Global Grid size = (%d, %d, %d)\n",
01948               xeffGlob, yeffGlob, zeffGlob);
01949     Vnm_print(0, "NOsh_setupMGPARA:  Global Grid Spacing = (%.3f, %.3f, %.3f)\n",

01950               hx, hy, hzed);
01951     Vnm_print(0, "NOsh_setupMGPARA:  Processor Grid Size = (%d, %d, %d)\n",
01952               xDisj, yDisj, zDisj);
01953
01954     /* Calculate the maximum and minimum processor grid points */
01955     xigminDisj = ip*xDisj;
01956     xigmaxDisj = xigminDisj + xDisj - 1;
01957     yigminDisj = jp*yDisj;
01958     yigmaxDisj = yigminDisj + yDisj - 1;
01959     zigminDisj = kp*zDisj;
01960     zigmaxDisj = zigminDisj + zDisj - 1;
01961     Vnm_print(0, "NOsh_setupMGPARA:  Min Grid Points for this proc. (%d, %d, %d)\
       n",
01962               xigminDisj, yigminDisj, zigminDisj);
01963     Vnm_print(0, "NOsh_setupMGPARA:  Max Grid Points for this proc. (%d, %d, %d)\
       n",
01964               xigmaxDisj, yigmaxDisj, zigmaxDisj);
01965
01966
01967     /* Calculate the disjoint partition length and center displacement */
01968     xminDisj = VMAX2(hx*(xigminDisj-0.5), 0.0);
01969     xmaxDisj = VMIN2(hx*(xigmaxDisj+0.5), mgparm->fglen[0]);
01970     xlenDisj = xmaxDisj - xminDisj;
01971     yminDisj = VMAX2(hy*(yigminDisj-0.5), 0.0);
01972     ymaxDisj = VMIN2(hy*(yigmaxDisj+0.5), mgparm->fglen[1]);
01973     ylenDisj = ymaxDisj - yminDisj;
01974     zminDisj = VMAX2(hzed*(zigminDisj-0.5), 0.0);
01975     zmaxDisj = VMIN2(hzed*(zigmaxDisj+0.5), mgparm->fglen[2]);
01976     zlenDisj = zmaxDisj - zminDisj;
01977
01978     xcent = 0.5*mgparm->fglen[0];
01979     ycent = 0.5*mgparm->fglen[1];
```

```
01980     zcent = 0.5*mgparm->fglen[2];
01981
01982     xcentDisj = xminDisj + 0.5*xlenDisj - xcent;
01983     ycentDisj = yminDisj + 0.5*ylenDisj - ycent;
01984     zcentDisj = zminDisj + 0.5*zlenDisj - zcent;
01985     if (VABS(xcentDisj) < VSMALL) xcentDisj = 0.0;
01986     if (VABS(ycentDisj) < VSMALL) ycentDisj = 0.0;
01987     if (VABS(zcentDisj) < VSMALL) zcentDisj = 0.0;
01988
01989     Vnm_print(0, "NOsh_setupMGPARA:  Disj part length = (%g, %g, %g)\n",
01990      xlenDisj, ylenDisj, zlenDisj);
01991     Vnm_print(0, "NOsh_setupMGPARA:  Disj part center displacement = (%g, %g, %g)
      \n",
01992      xcentDisj, ycentDisj, zcentDisj);
01993
01994     /* Calculate the overlapping partition length and center displacement */
01995     xOlapReg = 0;
01996     yOlapReg = 0;
01997     zOlapReg = 0;
01998     if (npx != 1) xOlapReg = (int)VFLOOR(xofrac*mgparm->fglen[0]/npx/hx + 0.5) +
      1;
01999     if (npy != 1) yOlapReg = (int)VFLOOR(yofrac*mgparm->fglen[1]/npy/hy + 0.5) +
      1;
02000     if (npz != 1) zOlapReg = (int)VFLOOR(zofrac*mgparm->fglen[2]/npz/hzed + 0.5)
      + 1;
02001
02002     Vnm_print(0, "NOsh_setupMGPARA:  No. of Grid Points in Overlap (%d, %d, %d)\n
      ",
02003               xOlapReg, yOlapReg, zOlapReg);
02004
02005     if (ip == 0) xigminOlap = 0;
02006     else if (ip == (npx - 1)) xigminOlap = xeffGlob - mgparm->dime[0];
02007     else xigminOlap = xigminDisj - xOlapReg;
02008     xigmaxOlap = xigminOlap + mgparm->dime[0] - 1;
02009
02010     if (jp == 0) yigminOlap = 0;
02011     else if (jp == (npy - 1)) yigminOlap = yeffGlob - mgparm->dime[1];
02012     else yigminOlap = yigminDisj - yOlapReg;
02013     yigmaxOlap = yigminOlap + mgparm->dime[1] - 1;
02014
02015     if (kp == 0) zigminOlap = 0;
02016     else if (kp == (npz - 1)) zigminOlap = zeffGlob - mgparm->dime[2];
02017     else zigminOlap = zigminDisj - zOlapReg;
02018     zigmaxOlap = zigminOlap + mgparm->dime[2] - 1;
02019
02020     Vnm_print(0, "NOsh_setupMGPARA:  Min Grid Points with Overlap (%d, %d, %d)\n"
      ,
02021               xigminOlap, yigminOlap, zigminOlap);
02022     Vnm_print(0, "NOsh_setupMGPARA:  Max Grid Points with Overlap (%d, %d, %d)\n"
      ,
02023               xigmaxOlap, yigmaxOlap, zigmaxOlap);
02024
02025     xminOlap = hx * xigminOlap;
02026     xmaxOlap = hx * xigmaxOlap;
02027     yminOlap = hy * yigminOlap;
02028     ymaxOlap = hy * yigmaxOlap;
02029     zminOlap = hzed * zigminOlap;
```

```
02030     zmaxOlap = hzed * zigmaxOlap;
02031
02032     xlenOlap = xmaxOlap - xminOlap;
02033     ylenOlap = ymaxOlap - yminOlap;
02034     zlenOlap = zmaxOlap - zminOlap;
02035
02036     xcentOlap = (xminOlap + 0.5*xlenOlap) - xcent;
02037     ycentOlap = (yminOlap + 0.5*ylenOlap) - ycent;
02038     zcentOlap = (zminOlap + 0.5*zlenOlap) - zcent;
02039     if (VABS(xcentOlap) < VSMALL) xcentOlap = 0.0;
02040     if (VABS(ycentOlap) < VSMALL) ycentOlap = 0.0;
02041     if (VABS(zcentOlap) < VSMALL) zcentOlap = 0.0;
02042
02043     Vnm_print(0, "NOsh_setupMGPARA:  Olap part length = (%g, %g, %g)\n",
02044      xlenOlap, ylenOlap, zlenOlap);
02045     Vnm_print(0, "NOsh_setupMGPARA:  Olap part center displacement = (%g, %g, %g)
      \n",
02046      xcentOlap, ycentOlap, zcentOlap);
02047
02048
02049     /* Calculate the boundary flags:
02050   Flags are set to 1 when another processor is present along the boundary
02051   Flags are otherwise set to 0. */
02052
02053     if (ip == 0) mgparm->partDisjOwnSide[VAPBS_LEFT] = 0;
02054  else mgparm->partDisjOwnSide[VAPBS_LEFT] = 1;
02055  if (ip == (npx-1)) mgparm->partDisjOwnSide[VAPBS_RIGHT] = 0;
02056  else mgparm->partDisjOwnSide[VAPBS_RIGHT] = 1;
02057  if (jp == 0) mgparm->partDisjOwnSide[VAPBS_BACK] = 0;
02058  else mgparm->partDisjOwnSide[VAPBS_BACK] = 1;
02059  if (jp == (npy-1)) mgparm->partDisjOwnSide[VAPBS_FRONT] = 0;
02060  else mgparm->partDisjOwnSide[VAPBS_FRONT] = 1;
02061  if (kp == 0) mgparm->partDisjOwnSide[VAPBS_DOWN] = 0;
02062  else mgparm->partDisjOwnSide[VAPBS_DOWN] = 1;
02063  if (kp == (npz-1)) mgparm->partDisjOwnSide[VAPBS_UP] = 0;
02064  else mgparm->partDisjOwnSide[VAPBS_UP] = 1;
02065
02066  Vnm_print(0, "NOsh_setupMGPARA:  partDisjOwnSide[LEFT] = %d\n",
02067      mgparm->partDisjOwnSide[VAPBS_LEFT]);
02068  Vnm_print(0, "NOsh_setupMGPARA:  partDisjOwnSide[RIGHT] = %d\n",
02069      mgparm->partDisjOwnSide[VAPBS_RIGHT]);
02070  Vnm_print(0, "NOsh_setupMGPARA:  partDisjOwnSide[FRONT] = %d\n",
02071      mgparm->partDisjOwnSide[VAPBS_FRONT]);
02072  Vnm_print(0, "NOsh_setupMGPARA:  partDisjOwnSide[BACK] = %d\n",
02073      mgparm->partDisjOwnSide[VAPBS_BACK]);
02074  Vnm_print(0, "NOsh_setupMGPARA:  partDisjOwnSide[UP] = %d\n",
02075      mgparm->partDisjOwnSide[VAPBS_UP]);
02076  Vnm_print(0, "NOsh_setupMGPARA:  partDisjOwnSide[DOWN] = %d\n",
02077      mgparm->partDisjOwnSide[VAPBS_DOWN]);
02078
02079  /* Set the mesh parameters */
02080  mgparm->fglen[0] = xlenOlap;
02081  mgparm->fglen[1] = ylenOlap;
02082  mgparm->fglen[2] = zlenOlap;
02083  mgparm->partDisjLength[0] = xlenDisj;
02084  mgparm->partDisjLength[1] = ylenDisj;
02085  mgparm->partDisjLength[2] = zlenDisj;
```

```
02086  mgparm->partDisjCenter[0] = mgparm->fcenter[0] + xcentDisj;
02087  mgparm->partDisjCenter[1] = mgparm->fcenter[1] + ycentDisj;
02088  mgparm->partDisjCenter[2] = mgparm->fcenter[2] + zcentDisj;
02089  mgparm->fcenter[0] += xcentOlap;
02090  mgparm->fcenter[1] += ycentOlap;
02091  mgparm->fcenter[2] += zcentOlap;
02092
02093  Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d):  Set up *relative* partition \
02094 centers...\n", __FILE__, __LINE__);
02095  Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d):  Absolute centers will be set \
02096 in  NOsh_setupMGAUTO\n", __FILE__, __LINE__);
02097  Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d):  partDisjCenter = %g %g %g\n",
02098      __FILE__, __LINE__,
02099      mgparm->partDisjCenter[0],
02100      mgparm->partDisjCenter[1],
02101      mgparm->partDisjCenter[2]);
02102  Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d):  ccenter = %g %g %g\n",
02103      __FILE__, __LINE__,
02104      mgparm->ccenter[0],
02105      mgparm->ccenter[1],
02106      mgparm->ccenter[2]);
02107  Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d):  fcenter = %g %g %g\n",
02108      __FILE__, __LINE__,
02109      mgparm->fcenter[0],
02110      mgparm->fcenter[1],
02111      mgparm->fcenter[2]);
02112
02113
02114  /* Setup the automatic focusing calculations associated with this processor */
02115  return NOsh_setupCalcMGAUTO(thee, elec);
02116
02117 }
02118
02119 VPUBLIC int NOsh_parseFEM(
02120         NOsh *thee,
02121         Vio *sock,
02122         NOsh_calc *elec
02123         ) {
02124
02125  char tok[VMAX_BUFSIZE];
02126  FEMparm *feparm = VNULL;
02127  PBEparm *pbeparm = VNULL;
02128  int rc;
02129  Vrc_Codes vrc;
02130
02131  /* Check the arguments */
02132  if (thee == VNULL) {
02133   Vnm_print(2, "NOsh_parseFEM:  Got NULL thee!\n");
02134   return 0;
02135  }
02136  if (sock == VNULL) {
02137   Vnm_print(2, "NOsh_parseFEM:  Got pointer to NULL socket!\n");
02138   return 0;
02139  }
02140  if (elec == VNULL) {
02141   Vnm_print(2, "NOsh_parseFEM:  Got pointer to NULL elec object!\n");
02142   return 0;
```

```
02143  }
02144  feparm = elec->femparm;
02145  if (feparm == VNULL) {
02146   Vnm_print(2, "NOsh_parseFEM:  Got pointer to NULL feparm object!\n");
02147   return 0;
02148  }
02149  pbeparm = elec->pbeparm;
02150  if (feparm == VNULL) {
02151   Vnm_print(2, "NOsh_parseFEM:  Got pointer to NULL pbeparm object!\n");
02152   return 0;
02153  }
02154
02155  Vnm_print(0, "NOsh_parseFEM: Parsing parameters for FEM calculation\n");
02156
02157  /* Start snarfing tokens from the input stream */
02158  rc = 1;
02159  while (Vio_scanf(sock, "%s", tok) == 1) {
02160
02161   Vnm_print(0, "NOsh_parseFEM:  Parsing %s...\n", tok);
02162
02163   /* See if it's an END token */
02164   if (Vstring_strcasecmp(tok, "end") == 0) {
02165    feparm->parsed = 1;
02166    pbeparm->parsed = 1;
02167    rc = 1;
02168    break;
02169   }
02170
02171   /* Pass the token through a series of parsers */
02172   rc = PBEparm_parseToken(pbeparm, tok, sock);
02173   if (rc == -1) {
02174    Vnm_print(0, "NOsh_parseFEM:  parsePBE error!\n");
02175    break;
02176   } else if (rc == 0) {
02177    /* Pass the token to the generic MG parser */
02178    vrc = FEMparm_parseToken(feparm, tok, sock);
02179    if (vrc == VRC_FAILURE) {
02180     Vnm_print(0, "NOsh_parseFEM:  parseMG error!\n");
02181     break;
02182    } else if (vrc == VRC_WARNING) {
02183     /* We ran out of parsers! */
02184     Vnm_print(2, "NOsh:  Unrecognized keyword: %s\n", tok);
02185     break;
02186    }
02187   }
02188  }
02189
02190  /* Handle various errors arising in the token-snarfing loop -- these all
02191   * just result in simple returns right now */
02192  if (rc == -1) return 0;
02193  if (rc == 0) return 0;
02194
02195  /* Check the status of the parameter objects */
02196  if ((!FEMparm_check(feparm)) || (!PBEparm_check(pbeparm))) {
02197   Vnm_print(2, "NOsh:  FEM parameters not set correctly!\n");
02198   return 0;
02199  }
```

```
02200
02201  return 1;
02202
02203 }
02204
02205 VPRIVATE int NOsh_setupCalcFEMANUAL(
02206            NOsh *thee,
02207            NOsh_calc *elec
02208            ) {
02209
02210  FEMparm *feparm = VNULL;
02211  PBEparm *pbeparm = VNULL;
02212  NOsh_calc *calc = VNULL;
02213
02214  VASSERT(thee != VNULL);
02215  VASSERT(elec != VNULL);
02216  feparm = elec->femparm;
02217  VASSERT(feparm != VNULL);
02218  pbeparm = elec->pbeparm;
02219  VASSERT(pbeparm);
02220
02221  /* Check to see if he have any room left for this type of
02222   * calculation, if so: set the calculation type, update the number
02223   * of calculations of this type, and parse the rest of the section
02224   */
02225  if (thee->ncalc >= NOSH_MAXCALC) {
02226   Vnm_print(2, "NOsh:  Too many calculations in this run!\n");
02227   Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
02228       NOSH_MAXCALC);
02229   return 0;
02230  }
02231  thee->calc[thee->ncalc] = NOsh_calc_ctor(NCT_FEM);
02232  calc = thee->calc[thee->ncalc];
02233  (thee->ncalc)++;
02234
02235  /* Copy over contents of ELEC */
02236  NOsh_calc_copy(calc, elec);
02237
02238
02239  return 1;
02240 }
02241
02242 VPUBLIC int NOsh_parseAPOL(
02243         NOsh *thee,
02244         Vio *sock,
02245         NOsh_calc *elec
02246         ) {
02247
02248  char tok[VMAX_BUFSIZE];
02249  APOLparm *apolparm = VNULL;
02250  int rc;
02251
02252  /* Check the arguments */
02253  if (thee == VNULL) {
02254   Vnm_print(2, "NOsh_parseAPOL:  Got NULL thee!\n");
02255   return 0;
02256  }
```

```
02257  if (sock == VNULL) {
02258   Vnm_print(2, "NOsh_parseAPOL:  Got pointer to NULL socket!\n");
02259   return 0;
02260  }
02261  if (elec == VNULL) {
02262   Vnm_print(2, "NOsh_parseAPOL:  Got pointer to NULL elec object!\n");
02263   return 0;
02264  }
02265  apolparm = elec->apolparm;
02266  if (apolparm == VNULL) {
02267   Vnm_print(2, "NOsh_parseAPOL:  Got pointer to NULL feparm object!\n");
02268   return 0;
02269  }
02270
02271  Vnm_print(0, "NOsh_parseAPOL: Parsing parameters for APOL calculation\n");
02272
02273  /* Start snarfing tokens from the input stream */
02274  rc = 1;
02275  while (Vio_scanf(sock, "%s", tok) == 1) {
02276
02277   Vnm_print(0, "NOsh_parseAPOL:  Parsing %s...\n", tok);
02278   /* See if it's an END token */
02279   if (Vstring_strcasecmp(tok, "end") == 0) {
02280    apolparm->parsed = 1;
02281    rc = 1;
02282    break;
02283   }
02284
02285   /* Pass the token through a series of parsers */
02286   /* Pass the token to the generic non-polar parser */
02287   rc = APOLparm_parseToken(apolparm, tok, sock);
02288   if (rc == -1) {
02289    Vnm_print(0, "NOsh_parseFEM:  parseMG error!\n");
02290    break;
02291   } else if (rc == 0) {
02292    /* We ran out of parsers! */
02293    Vnm_print(2, "NOsh:  Unrecognized keyword: %s\n", tok);
02294    break;
02295   }
02296
02297  }
02298
02299  /* Handle various errors arising in the token-snarfing loop -- these all
02300   * just result in simple returns right now */
02301  if (rc == -1) return 0;
02302  if (rc == 0) return 0;
02303
02304  /* Check the status of the parameter objects */
02305  if (!APOLparm_check(apolparm)) {
02306   Vnm_print(2, "NOsh:  APOL parameters not set correctly!\n");
02307   return 0;
02308  }
02309
02310  return 1;
02311
02312 }
02313
```

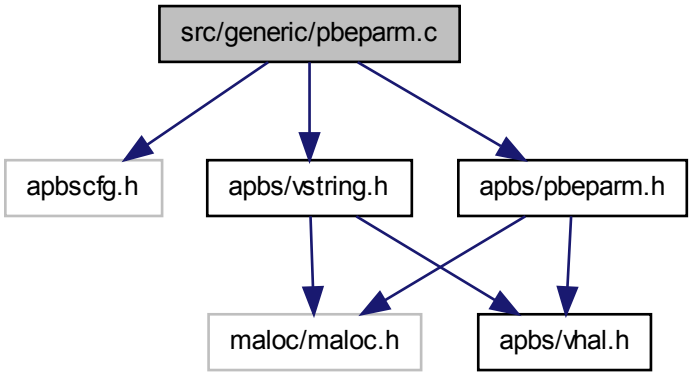
```
02314 VPRIVATE int NOsh_setupCalcAPOL(
02315         NOsh *thee,
02316         NOsh_calc *apol
02317         ) {
02318
02319  NOsh_calc *calc = VNULL;
02320
02321  VASSERT(thee != VNULL);
02322  VASSERT(apol != VNULL);
02323
02324  /* Check to see if he have any room left for this type of
02325   * calculation, if so: set the calculation type, update the number
02326   * of calculations of this type, and parse the rest of the section
02327   */
02328  if (thee->ncalc >= NOSH_MAXCALC) {
02329   Vnm_print(2, "NOsh:  Too many calculations in this run!\n");
02330   Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
02331       NOSH_MAXCALC);
02332   return 0;
02333  }
02334  thee->calc[thee->ncalc] = NOsh_calc_ctor(NCT_APOL);
02335  calc = thee->calc[thee->ncalc];
02336  (thee->ncalc)++;
02337
02338  /* Copy over contents of APOL */
02339  NOsh_calc_copy(calc, apol);
02340
02341  return 1;
02342 }
```

10.59 src/generic/pbeparm.c File Reference

Class PBEparm methods.

```
#include "apbscfg.h"

#include "apbs/pbeparm.h"

#include "apbs/vstring.h"
```

Include dependency graph for pbeparm.c:



Functions

- VPUBLIC double PBEparm_getIonCharge (PBEparm *thee, int i)

  Get charge (e) of specified ion species.

- VPUBLIC double PBEparm_getIonConc (PBEparm *thee, int i)

  Get concentration (M) of specified ion species.

- VPUBLIC double PBEparm_getIonRadius (PBEparm *thee, int i)

  Get radius (A) of specified ion species.

- VPUBLIC double **PBEparm_getzmem** (PBEparm *thee)
- VPUBLIC double **PBEparm_getLmem** (PBEparm *thee)
- VPUBLIC double **PBEparm_getmembraneDiel** (PBEparm *thee)
- VPUBLIC double **PBEparm_getmemv** (PBEparm *thee)
- VPUBLIC PBEparm * PBEparm_ctor ()

  Construct PBEparm object.

- VPUBLIC int PBEparm_ctor2 (PBEparm *thee)

  FORTRAN stub to construct PBEparm 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_parseION** (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 1585 2010-05-13 16:21:17Z sdg0919

Attention

Definition in file pbeparm.c.

10.60 src/generic/pbeparm.c

```
00001
00049 #include "apbscfg.h"
00050 #include "apbs/pbeparm.h"
00051 #include "apbs/vstring.h"
00052
00053 VEMBED(rcsid="$Id: pbeparm.c 1585 2010-05-13 16:21:17Z sdg0919 $")
00054
00055 #if !defined(VINLINE_MGPARM)
00056
00057 #endif /* if !defined(VINLINE_MGPARM) */
00058
00059 VPUBLIC double PBEparm_getIonCharge(PBEparm *thee, int i) {
00060     VASSERT(thee != VNULL);
00061  VASSERT(i < thee->nion);
00062     return thee->ionq[i];
00063 }
00064
00065 VPUBLIC double PBEparm_getIonConc(PBEparm *thee, int i) {
00066     VASSERT(thee != VNULL);
00067     VASSERT(i < thee->nion);
00068     return thee->ionc[i];
00069 }
00070
00071 VPUBLIC double PBEparm_getIonRadius(PBEparm *thee, int i) {
00072     VASSERT(thee != VNULL);
00073     VASSERT(i < thee->nion);
00074     return thee->ionr[i];
00075 }
00076
00077 /*-------------------------------------------------------------------------*/
00078 /* Added by Michael Grabe                                                  */
00079 /*-------------------------------------------------------------------------*/
00080
00081 VPUBLIC double PBEparm_getzmem(PBEparm *thee) {
00082     VASSERT(thee != VNULL);
00083     return thee->zmem;
00084 }
00085 VPUBLIC double PBEparm_getLmem(PBEparm *thee) {
00086     VASSERT(thee != VNULL);
00087     return thee->Lmem;
00088 }
00089 VPUBLIC double PBEparm_getmembraneDiel(PBEparm *thee) {
00090     VASSERT(thee != VNULL);
00091     return thee->mdie;
00092 }
00093 VPUBLIC double PBEparm_getmemv(PBEparm *thee) {
00094     VASSERT(thee != VNULL);
00095     return thee->memv;
00096 }
00097
```

```
00098 VPUBLIC PBEparm* PBEparm_ctor() {
00099
00100     /* Set up the structure */
00101     PBEparm *thee = VNULL;
00102     thee = Vmem_malloc(VNULL, 1, sizeof(PBEparm));
00103     VASSERT( thee != VNULL);
00104     VASSERT( PBEparm_ctor2(thee) );
00105
00106     return thee;
00107 }
00108
00109 VPUBLIC int PBEparm_ctor2(PBEparm *thee) {
00110
00111     int i;
00112
00113     if (thee == VNULL) return 0;
00114
00115     thee->parsed = 0;
00116
00117     thee->setmolid = 0;
00118     thee->setpbetype = 0;
00119     thee->setbcfl = 0;
00120     thee->setnion = 0;
00121     for (i=0; i<MAXION; i++){
00122   thee->setion[i] = 0;
00123   thee->ionq[i] = 0.0;
00124   thee->ionc[i] = 0.0;
00125   thee->ionr[i] = 0.0;
00126  }
00127     thee->setpdie = 0;
00128     thee->setsdie = 0;
00129     thee->setsrfm = 0;
00130     thee->setsrad = 0;
00131     thee->setswin = 0;
00132     thee->settemp = 0;
00133     thee->setcalcenergy = 0;
00134     thee->setcalcforce = 0;
00135     thee->setsdens = 0;
00136     thee->numwrite = 0;
00137     thee->setwritemat = 0;
00138     thee->nion = 0;
00139     thee->sdens = 0;
00140     thee->swin = 0;
00141     thee->srad = 1.4;
00142     thee->useDielMap = 0;
00143     thee->useKappaMap = 0;
00144  thee->usePotMap = 0;
00145     thee->useChargeMap = 0;
00146
00147  /*-----------------------------------------------*/
00148  /* Added by Michael Grabe                        */
00149  /*-----------------------------------------------*/
00150
00151     thee->setzmem = 0;
00152     thee->setLmem = 0;
00153     thee->setmdie = 0;
00154     thee->setmemv = 0;
```

```
00155
00156  /*-------------------------------------------*/
00157
00158  thee->smsize = 0.0;
00159  thee->smvolume = 0.0;
00160
00161  thee->setsmsize = 0;
00162  thee->setsmvolume = 0;
00163
00164     return 1;
00165 }
00166
00167 VPUBLIC void PBEparm_dtor(PBEparm **thee) {
00168     if ((*thee) != VNULL) {
00169         PBEparm_dtor2(*thee);
00170         Vmem_free(VNULL, 1, sizeof(PBEparm), (void **)thee);
00171         (*thee) = VNULL;
00172     }
00173 }
00174
00175 VPUBLIC void PBEparm_dtor2(PBEparm *thee) { ; }
00176
00177 VPUBLIC int PBEparm_check(PBEparm *thee) {
00178
00179     int i;
00180
00181     /* Check to see if we were even filled... */
00182     if (!thee->parsed) {
00183         Vnm_print(2, "PBEparm_check:  not filled!\n");
00184         return 0;
00185     }
00186
00187     if (!thee->setmolid) {
00188         Vnm_print(2, "PBEparm_check:  MOL not set!\n");
00189         return 0;
00190     }
00191     if (!thee->setpbetype) {
00192         Vnm_print(2, "PBEparm_check:  LPBE/NPBE/LRPBE/NRPBE/SMPBE not set!\n");
00193         return 0;
00194     }
00195     if (!thee->setbcfl) {
00196         Vnm_print(2, "PBEparm_check:  BCFL not set!\n");
00197         return 0;
00198     }
00199     if (!thee->setnion) {
00200         thee->setnion = 1;
00201         thee->nion = 0;
00202     }
00203     for (i=0; i<thee->nion; i++) {
00204         if (!thee->setion[i]) {
00205             Vnm_print(2, "PBEparm_check: ION #%d not set!\n",i);
00206             return 0;
00207         }
00208     }
00209     if (!thee->setpdie) {
00210         Vnm_print(2, "PBEparm_check: PDIE not set!\n");
00211         return 0;
```

```
00212     }
00213     if (((thee->srfm==VSM_MOL) || (thee->srfm==VSM_MOLSMOOTH)) \
00214       && (!thee->setsdens) && (thee->srad > VSMALL)) {
00215         Vnm_print(2, "PBEparm_check: SDENS not set!\n");
00216         return 0;
00217     }
00218     if (!thee->setsdie) {
00219         Vnm_print(2, "PBEparm_check: SDIE not set!\n");
00220         return 0;
00221     }
00222     if (!thee->setsrfm) {
00223         Vnm_print(2, "PBEparm_check: SRFM not set!\n");
00224         return 0;
00225     }
00226     if (((thee->srfm==VSM_MOL) || (thee->srfm==VSM_MOLSMOOTH)) \
00227       && (!thee->setsrad)) {
00228         Vnm_print(2, "PBEparm_check: SRAD not set!\n");
00229         return 0;
00230     }
00231     if ((thee->srfm==VSM_SPLINE) && (!thee->setswin)) {
00232         Vnm_print(2, "PBEparm_check: SWIN not set!\n");
00233         return 0;
00234     }
00235     if ((thee->srfm==VSM_SPLINE3) && (!thee->setswin)) {
00236         Vnm_print(2, "PBEparm_check: SWIN not set!\n");
00237         return 0;
00238     }
00239     if ((thee->srfm==VSM_SPLINE4) && (!thee->setswin)) {
00240         Vnm_print(2, "PBEparm_check: SWIN not set!\n");
00241         return 0;
00242     }
00243     if (!thee->settemp) {
00244         Vnm_print(2, "PBEparm_check: TEMP not set!\n");
00245         return 0;
00246     }
00247     if (!thee->setcalcenergy) thee->calcenergy = PCE_NO;
00248     if (!thee->setcalcforce) thee->calcforce = PCF_NO;
00249     if (!thee->setwritemat) thee->writemat = 0;
00250 
00251 /*-----------------------------------------------------------*/
00252 /* Added by Michael Grabe                                    */
00253 /*-----------------------------------------------------------*/
00254 
00255     if ((!thee->setzmem) && (thee->bcfl == 3)){
00256         Vnm_print(2, "PBEparm_check: ZMEM not set!\n");
00257         return 0;
00258     }
00259     if ((!thee->setLmem) && (thee->bcfl == 3)){
00260         Vnm_print(2, "PBEparm_check: LMEM not set!\n");
00261         return 0;
00262     }
00263     if ((!thee->setmdie) && (thee->bcfl == 3)){
00264         Vnm_print(2, "PBEparm_check: MDIE not set!\n");
00265         return 0;
00266     }
00267     if ((!thee->setmemv) && (thee->bcfl == 3)){
00268         Vnm_print(2, "PBEparm_check: MEMV not set!\n");
```

```
00269         return 0;
00270     }
00271
00272  /*-------------------------------------------------------*/
00273
00274     return 1;
00275 }
00276
00277 VPUBLIC void PBEparm_copy(PBEparm *thee, PBEparm *parm) {
00278
00279     int i, j;
00280
00281     VASSERT(thee != VNULL);
00282     VASSERT(parm != VNULL);
00283
00284     thee->molid = parm->molid;
00285     thee->setmolid = parm->setmolid;
00286     thee->useDielMap = parm->useDielMap;
00287     thee->dielMapID = parm->dielMapID;
00288     thee->useKappaMap = parm->useKappaMap;
00289     thee->kappaMapID = parm->kappaMapID;
00290  thee->usePotMap = parm->usePotMap;
00291  thee->potMapID = parm->potMapID;
00292     thee->useChargeMap = parm->useChargeMap;
00293     thee->chargeMapID = parm->chargeMapID;
00294     thee->pbetype = parm->pbetype;
00295     thee->setpbetype = parm->setpbetype;
00296     thee->bcfl = parm->bcfl;
00297     thee->setbcfl = parm->setbcfl;
00298     thee->nion = parm->nion;
00299     thee->setnion = parm->setnion;
00300     for (i=0; i<MAXION; i++) {
00301         thee->ionq[i] = parm->ionq[i];
00302         thee->ionc[i] = parm->ionc[i];
00303         thee->ionr[i] = parm->ionr[i];
00304         thee->setion[i] = parm->setion[i];
00305     };
00306     thee->pdie = parm->pdie;
00307     thee->setpdie = parm->setpdie;
00308     thee->sdens = parm->sdens;
00309     thee->setsdens = parm->setsdens;
00310     thee->sdie = parm->sdie;
00311     thee->setsdie = parm->setsdie;
00312     thee->srfm = parm->srfm;
00313     thee->setsrfm = parm->setsrfm;
00314     thee->srad = parm->srad;
00315     thee->setsrad = parm->setsrad;
00316     thee->swin = parm->swin;
00317     thee->setswin = parm->setswin;
00318     thee->temp = parm->temp;
00319     thee->settemp = parm->settemp;
00320     thee->calcenergy = parm->calcenergy;
00321     thee->setcalcenergy = parm->setcalcenergy;
00322     thee->calcforce = parm->calcforce;
00323     thee->setcalcforce = parm->setcalcforce;
00324
00325  /*-------------------------------------------------------*/
```

```
00326  /* Added by Michael Grabe                                   */
00327  /*-----------------------------------------------------------*/
00328
00329     thee->zmem = parm->zmem;
00330     thee->setzmem = parm->setzmem;
00331     thee->Lmem = parm->Lmem;
00332     thee->setLmem = parm->setLmem;
00333     thee->mdie = parm->mdie;
00334     thee->setmdie = parm->setmdie;
00335     thee->memv = parm->memv;
00336     thee->setmemv = parm->setmemv;
00337
00338  /*-----------------------------------------------------------*/
00339
00340     thee->numwrite = parm->numwrite;
00341     for (i=0; i<PBEPARM_MAXWRITE; i++) {
00342         thee->writetype[i] = parm->writetype[i];
00343         thee->writefmt[i] = parm->writefmt[i];
00344         for (j=0; j<VMAX_ARGLEN; j++)
00345           thee->writestem[i][j] = parm->writestem[i][j];
00346     }
00347     thee->writemat = parm->writemat;
00348     thee->setwritemat = parm->setwritemat;
00349     for (i=0; i<VMAX_ARGLEN; i++) thee->writematstem[i] = parm->writematstem[i];
00350     thee->writematflag = parm->writematflag;
00351
00352  thee->smsize = parm->smsize;
00353  thee->smvolume = parm->smvolume;
00354
00355  thee->setsmsize = parm->setsmsize;
00356  thee->setsmvolume = parm->setsmvolume;
00357
00358     thee->parsed = parm->parsed;
00359
00360 }
00361
00362 VPRIVATE int PBEparm_parseLPBE(PBEparm *thee, Vio *sock) {
00363     Vnm_print(0, "NOsh: parsed lpbe\n");
00364     thee->pbetype = PBE_LPBE;
00365     thee->setpbetype = 1;
00366     return 1;
00367 }
00368
00369 VPRIVATE int PBEparm_parseNPBE(PBEparm *thee, Vio *sock) {
00370     Vnm_print(0, "NOsh: parsed npbe\n");
00371     thee->pbetype = PBE_NPBE;
00372     thee->setpbetype = 1;
00373     return 1;
00374 }
00375
00376 VPRIVATE int PBEparm_parseMOL(PBEparm *thee, Vio *sock) {
00377     int ti;
00378     char tok[VMAX_BUFSIZE];
00379
00380     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00381     if (sscanf(tok, "%d", &ti) == 0) {
00382         Vnm_print(2, "NOsh:  Read non-int (%s) while parsing MOL \
```

```
00383 keyword!\n", tok);
00384         return -1;
00385     }
00386     thee->molid = ti;
00387     thee->setmolid = 1;
00388     return 1;
00389 
00390     VERROR1:
00391         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00392         return -1;
00393 }
00394 
00395 VPRIVATE int PBEparm_parseLRPBE(PBEparm *thee, Vio *sock) {
00396     Vnm_print(0, "NOsh: parsed lrpbe\n");
00397     thee->pbetype = PBE_LRPBE;
00398     thee->setpbetype = 1;
00399     return 1;
00400 }
00401 
00402 VPRIVATE int PBEparm_parseNRPBE(PBEparm *thee, Vio *sock) {
00403     Vnm_print(0, "NOsh: parsed nrpbe\n");
00404     thee->pbetype = PBE_NRPBE;
00405     thee->setpbetype = 1;
00406     return 1;
00407 }
00408 
00409 VPRIVATE int PBEparm_parseSMPBE(PBEparm *thee, Vio *sock) {
00410 
00411  int i;
00412 
00413  char type[VMAX_BUFSIZE]; /* vol or size (keywords) */
00414  char value[VMAX_BUFSIZE]; /* floating point value */
00415 
00416  char setVol = 1;
00417  char setSize = 1;
00418  char keyValuePairs = 2;
00419 
00420     double size, volume;
00421 
00422  for(i=0;i<keyValuePairs;i++){
00423 
00424   /* The line two tokens at a time */
00425   VJMPERR1(Vio_scanf(sock, "%s", type) == 1);
00426   VJMPERR1(Vio_scanf(sock, "%s", value) == 1);
00427 
00428   if(!strcmp(type,"vol")){
00429    if ((setVol = sscanf(value, "%lf", &volume)) == 0){
00430     Vnm_print(2,"NOsh:  Read non-float (%s) while parsing smpbe keyword!\n", valu
      e);
00431     return VRC_FAILURE;
00432    }
00433   }else if(!strcmp(type,"size")){
00434    if ((setSize = sscanf(value, "%lf", &size)) == 0){
00435     Vnm_print(2,"NOsh:  Read non-float (%s) while parsing smpbe keyword!\n", valu
      e);
00436     return VRC_FAILURE;
00437    }
```

```
00438   }else{
00439    Vnm_print(2,"NOsh:  Read non-float (%s) while parsing smpbe keyword!\n", value
      );
00440    return VRC_FAILURE;
00441   }
00442  }
00443
00444  /* If either the volume or size isn't set, throw an error */
00445  if((setVol == 0) || (setSize == 0)){
00446   Vnm_print(2,"NOsh:  Error while parsing smpbe keywords! Only size or vol was sp
      ecified.\n");
00447   return VRC_FAILURE;
00448  }
00449
00450  Vnm_print(0, "NOsh: parsed smpbe\n");
00451     thee->pbetype = PBE_SMPBE;
00452     thee->setpbetype = 1;
00453
00454  thee->smsize = size;
00455  thee->setsmsize = 1;
00456
00457  thee->smvolume = volume;
00458  thee->setsmvolume = 1;
00459
00460  return VRC_SUCCESS;
00461
00462 VERROR1:
00463     Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00464  return VRC_FAILURE;
00465
00466 }
00467
00468 VPRIVATE int PBEparm_parseBCFL(PBEparm *thee, Vio *sock) {
00469     char tok[VMAX_BUFSIZE];
00470     int ti;
00471
00472     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00473
00474     /* We can either parse int flag... */
00475     if (sscanf(tok, "%d", &ti) == 1) {
00476
00477         thee->bcfl = ti;
00478         thee->setbcfl = 1;
00479         /* Warn that this usage is deprecated */
00480         Vnm_print(2, "parsePBE:  Warning -- parsed deprecated \"bcfl %d\" \
00481 statement\n", ti);
00482         Vnm_print(2, "parsePBE:  Please use \"bcfl ");
00483         switch (thee->bcfl) {
00484             case BCFL_ZERO:
00485                 Vnm_print(2, "zero");
00486                 break;
00487             case BCFL_SDH:
00488                 Vnm_print(2, "sdh");
00489                 break;
00490             case BCFL_MDH:
00491                 Vnm_print(2, "mdh");
00492                 break;
```

```
00493             case BCFL_FOCUS:
00494                 Vnm_print(2, "focus");
00495                 break;
00496     case BCFL_MEM:
00497                 Vnm_print(2, "mem");
00498                 break;
00499     case BCFL_MAP:
00500                 Vnm_print(2, "map");
00501                 break;
00502             default:
00503                 Vnm_print(2, "UKNOWN");
00504                 break;
00505         }
00506         Vnm_print(2, "\" instead.\n");
00507         return 1;
00508
00509      /* ...or the word */
00510      } else {
00511
00512         if (Vstring_strcasecmp(tok, "zero") == 0) {
00513             thee->bcfl = BCFL_ZERO;
00514             thee->setbcfl = 1;
00515             return 1;
00516         } else if (Vstring_strcasecmp(tok, "sdh") == 0) {
00517             thee->bcfl = BCFL_SDH;
00518             thee->setbcfl = 1;
00519             return 1;
00520         } else if (Vstring_strcasecmp(tok, "mdh") == 0) {
00521             thee->bcfl = BCFL_MDH;
00522             thee->setbcfl = 1;
00523             return 1;
00524         } else if (Vstring_strcasecmp(tok, "focus") == 0) {
00525             thee->bcfl = BCFL_FOCUS;
00526             thee->setbcfl = 1;
00527             return 1;
00528    } else if (Vstring_strcasecmp(tok, "mem") == 0) {
00529             thee->bcfl = BCFL_MEM;
00530             thee->setbcfl = 1;
00531             return 1;
00532         } else if (Vstring_strcasecmp(tok, "map") == 0) {
00533             thee->bcfl = BCFL_MAP;
00534             thee->setbcfl = 1;
00535             return 1;
00536         } else {
00537             Vnm_print(2, "NOsh:  parsed unknown BCFL parameter (%s)!\n",
00538               tok);
00539             return -1;
00540         }
00541      }
00542      return 0;
00543
00544      VERROR1:
00545         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00546         return -1;
00547 }
00548
00549 VPRIVATE int PBEparm_parseION(PBEparm *thee, Vio *sock) {
```

```
00550
00551  int i;
00552  int meth = 0;
00553
00554  char tok[VMAX_BUFSIZE];
00555  char value[VMAX_BUFSIZE];
00556
00557  double tf;
00558  double charge, conc, radius;
00559
00560  int setCharge = 0;
00561  int setConc = 0;
00562  int setRadius = 0;
00563  int keyValuePairs = 3;
00564
00565  /* Get the initial token for the ION statement */
00566     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00567
00568  /* Scan the token once to determine the type (old style or new keyValue pair) */

00569  meth = sscanf(tok, "%lf", &tf);
00570  /* If tok is a non-zero float value, we are using the old method */
00571  if(meth != 0){
00572
00573   Vnm_print(2, "NOsh:  Deprecated use of ION keyword! Use key-value pairs\n", tok
      );
00574
00575   if (sscanf(tok, "%lf", &tf) == 0) {
00576    Vnm_print(2, "NOsh:  Read non-float (%s) while parsing ION keyword!\n", tok);
00577    return VRC_FAILURE;
00578   }
00579   thee->ionq[thee->nion] = tf;
00580   VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00581   if (sscanf(tok, "%lf", &tf) == 0) {
00582    Vnm_print(2, "NOsh:  Read non-float (%s) while parsing ION keyword!\n", tok);
00583    return VRC_FAILURE;
00584   }
00585   thee->ionc[thee->nion] = tf;
00586   VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00587   if (sscanf(tok, "%lf", &tf) == 0) {
00588    Vnm_print(2, "NOsh:  Read non-float (%s) while parsing ION keyword!\n", tok);
00589    return VRC_FAILURE;
00590   }
00591   thee->ionr[thee->nion] = tf;
00592
00593  }else{
00594
00595   /* Three key-value pairs (charge, radius and conc) */
00596   for(i=0;i<keyValuePairs;i++){
00597
00598    /* Now scan for the value (float) to be used with the key token parsed
00599     * above the if-else statement */
00600    VJMPERR1(Vio_scanf(sock, "%s", value) == 1);
00601    if(!strcmp(tok,"charge")){
00602     setCharge = sscanf(value, "%lf", &charge);
00603     if (setCharge == 0){
00604      Vnm_print(2,"NOsh:  Read non-float (%s) while parsing ION %s keyword!\n", va
```

```
      lue, tok);
00605      return VRC_FAILURE;
00606     }
00607     thee->ionq[thee->nion] = charge;
00608    }else if(!strcmp(tok,"radius")){
00609     setRadius = sscanf(value, "%lf", &radius);
00610     if (setRadius == 0){
00611      Vnm_print(2,"NOsh:  Read non-float (%s) while parsing ION %s keyword!\n", va
      lue, tok);
00612      return VRC_FAILURE;
00613     }
00614     thee->ionr[thee->nion] = radius;
00615    }else if(!strcmp(tok,"conc")){
00616     setConc = sscanf(value, "%lf", &conc);
00617     if (setConc == 0){
00618      Vnm_print(2,"NOsh:  Read non-float (%s) while parsing ION %s keyword!\n", va
      lue, tok);
00619      return VRC_FAILURE;
00620     }
00621     thee->ionc[thee->nion] = conc;
00622    }else{
00623     Vnm_print(2,"NOsh:  Illegal or missing key-value pair for ION keyword!\n");
00624     return VRC_FAILURE;
00625    }
00626
00627    /* If all three values haven't be set (setValue = 0) then read the next token
      */
00628    if((setCharge != 1) || (setConc != 1) || (setRadius != 1)){
00629     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00630    }
00631
00632   } /* end for */
00633  } /* end if */
00634
00635  /* Finally set the setion, nion and setnion flags and return success */
00636  thee->setion[thee->nion] = 1;
00637  (thee->nion)++;
00638  thee->setnion = 1;
00639  return VRC_SUCCESS;
00640
00641     VERROR1:
00642         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00643         return VRC_FAILURE;
00644 }
00645
00646 VPRIVATE int PBEparm_parsePDIE(PBEparm *thee, Vio *sock) {
00647     char tok[VMAX_BUFSIZE];
00648     double tf;
00649
00650     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00651     if (sscanf(tok, "%lf", &tf) == 0) {
00652         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing PDIE \
00653 keyword!\n", tok);
00654         return -1;
00655     }
00656     thee->pdie = tf;
00657     thee->setpdie = 1;
```

```
00658     return 1;
00659
00660     VERROR1:
00661         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00662         return -1;
00663 }
00664
00665 VPRIVATE int PBEparm_parseSDENS(PBEparm *thee, Vio *sock) {
00666     char tok[VMAX_BUFSIZE];
00667     double tf;
00668
00669     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00670     if (sscanf(tok, "%lf", &tf) == 0) {
00671         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SDENS \
00672 keyword!\n", tok);
00673         return -1;
00674     }
00675     thee->sdens = tf;
00676     thee->setsdens = 1;
00677     return 1;
00678
00679     VERROR1:
00680         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00681         return -1;
00682 }
00683
00684 VPRIVATE int PBEparm_parseSDIE(PBEparm *thee, Vio *sock) {
00685     char tok[VMAX_BUFSIZE];
00686     double tf;
00687
00688     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00689     if (sscanf(tok, "%lf", &tf) == 0) {
00690         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SDIE \
00691 keyword!\n", tok);
00692         return -1;
00693     }
00694     thee->sdie = tf;
00695     thee->setsdie = 1;
00696     return 1;
00697
00698     VERROR1:
00699         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00700         return -1;
00701 }
00702
00703 VPRIVATE int PBEparm_parseSRFM(PBEparm *thee, Vio *sock) {
00704     char tok[VMAX_BUFSIZE];
00705     int ti;
00706
00707     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00708
00709     /* Parse old-style int arg */
00710     if (sscanf(tok, "%d", &ti) == 1) {
00711         thee->srfm = ti;
00712         thee->setsrfm = 1;
00713
00714         Vnm_print(2, "parsePBE:  Warning -- parsed deprecated \"srfm %d\" \
```

```
00715 statement.\n", ti);
00716         Vnm_print(2, "parsePBE:  Please use \"srfm ");
00717         switch (thee->srfm) {
00718             case VSM_MOL:
00719                 Vnm_print(2, "mol");
00720                 break;
00721             case VSM_MOLSMOOTH:
00722                 Vnm_print(2, "smol");
00723                 break;
00724             case VSM_SPLINE:
00725                 Vnm_print(2, "spl2");
00726                 break;
00727             case VSM_SPLINE3:
00728                 Vnm_print(2, "spl3");
00729                 break;
00730             case VSM_SPLINE4:
00731                 Vnm_print(2, "spl4");
00732                 break;
00733             default:
00734                 Vnm_print(2, "UNKNOWN");
00735                 break;
00736         }
00737         Vnm_print(2, "\" instead.\n");
00738         return 1;
00739 
00740     /* Parse newer text-based args */
00741     } else if (Vstring_strcasecmp(tok, "mol") == 0) {
00742         thee->srfm = VSM_MOL;
00743         thee->setsrfm = 1;
00744         return 1;
00745     } else if (Vstring_strcasecmp(tok, "smol") == 0) {
00746         thee->srfm = VSM_MOLSMOOTH;
00747         thee->setsrfm = 1;
00748         return 1;
00749     } else if (Vstring_strcasecmp(tok, "spl2") == 0) {
00750         thee->srfm = VSM_SPLINE;
00751         thee->setsrfm = 1;
00752         return 1;
00753     } else if (Vstring_strcasecmp(tok, "spl3") == 0) {
00754         thee->srfm = VSM_SPLINE3;
00755         thee->setsrfm = 1;
00756         return 1;
00757  } else if (Vstring_strcasecmp(tok, "spl4") == 0) {
00758         thee->srfm = VSM_SPLINE4;
00759         thee->setsrfm = 1;
00760         return 1;
00761     } else {
00762         Vnm_print(2, "NOsh:  Unrecongnized keyword (%s) when parsing \
00763 srfm!\n", tok);
00764         return -1;
00765     }
00766 
00767     return 0;
00768 
00769     VERROR1:
00770         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00771         return -1;
```

```
00772 }
00773
00774 VPRIVATE int PBEparm_parseSRAD(PBEparm *thee, Vio *sock) {
00775     char tok[VMAX_BUFSIZE];
00776     double tf;
00777
00778     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00779     if (sscanf(tok, "%lf", &tf) == 0) {
00780         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SRAD \
00781 keyword!\n", tok);
00782         return -1;
00783     }
00784     thee->srad = tf;
00785     thee->setsrad = 1;
00786     return 1;
00787
00788     VERROR1:
00789         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00790         return -1;
00791 }
00792
00793 VPRIVATE int PBEparm_parseSWIN(PBEparm *thee, Vio *sock) {
00794     char tok[VMAX_BUFSIZE];
00795     double tf;
00796
00797     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00798     if (sscanf(tok, "%lf", &tf) == 0) {
00799         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SWIN \
00800 keyword!\n", tok);
00801         return -1;
00802     }
00803     thee->swin = tf;
00804     thee->setswin = 1;
00805     return 1;
00806
00807     VERROR1:
00808         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00809         return -1;
00810 }
00811
00812 VPRIVATE int PBEparm_parseTEMP(PBEparm *thee, Vio *sock) {
00813     char tok[VMAX_BUFSIZE];
00814     double tf;
00815
00816     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00817     if (sscanf(tok, "%lf", &tf) == 0) {
00818         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing TEMP \
00819 keyword!\n", tok);
00820         return -1;
00821     }
00822     thee->temp = tf;
00823     thee->settemp = 1;
00824     return 1;
00825
00826     VERROR1:
00827         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00828         return -1;
```

```
00829 }
00830
00831 VPRIVATE int PBEparm_parseUSEMAP(PBEparm *thee, Vio *sock) {
00832     char tok[VMAX_BUFSIZE];
00833     int ti;
00834
00835     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00836     Vnm_print(0, "PBEparm_parseToken:  Read %s...\n", tok);
00837     if (Vstring_strcasecmp(tok, "diel") == 0) {
00838         thee->useDielMap = 1;
00839         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00840         if (sscanf(tok, "%d", &ti) == 0) {
00841             Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
00842 USEMAP DIEL keyword!\n", tok);
00843             return -1;
00844         }
00845         thee->dielMapID = ti;
00846         return 1;
00847     } else if (Vstring_strcasecmp(tok, "kappa") == 0) {
00848         thee->useKappaMap = 1;
00849         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00850         if (sscanf(tok, "%d", &ti) == 0) {
00851             Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
00852 USEMAP KAPPA keyword!\n", tok);
00853             return -1;
00854         }
00855         thee->kappaMapID = ti;
00856         return 1;
00857     } else if (Vstring_strcasecmp(tok, "pot") == 0) {
00858         thee->usePotMap = 1;
00859         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00860         if (sscanf(tok, "%d", &ti) == 0) {
00861             Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
00862        USEMAP POT keyword!\n", tok);
00863             return -1;
00864         }
00865         thee->potMapID = ti;
00866         return 1;
00867     } else if (Vstring_strcasecmp(tok, "charge") == 0) {
00868         thee->useChargeMap = 1;
00869         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00870         if (sscanf(tok, "%d", &ti) == 0) {
00871             Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
00872 USEMAP CHARGE keyword!\n", tok);
00873             return -1;
00874         }
00875         thee->chargeMapID = ti;
00876         return 1;
00877     } else {
00878         Vnm_print(2, "NOsh:  Read undefined keyword (%s) while parsing \
00879 USEMAP statement!\n", tok);
00880         return -1;
00881     }
00882     return 0;
00883
00884     VERROR1:
00885         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
```

```
00886         return -1;
00887 }
00888
00889 VPRIVATE int PBEparm_parseCALCENERGY(PBEparm *thee, Vio *sock) {
00890     char tok[VMAX_BUFSIZE];
00891     int ti;
00892
00893     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00894     /* Parse number */
00895     if (sscanf(tok, "%d", &ti) == 1) {
00896         thee->calcenergy = ti;
00897         thee->setcalcenergy = 1;
00898
00899         Vnm_print(2, "parsePBE:  Warning -- parsed deprecated \"calcenergy \
00900 %d\" statement.\n", ti);
00901         Vnm_print(2, "parsePBE:  Please use \"calcenergy ");
00902         switch (thee->calcenergy) {
00903             case PCE_NO:
00904                 Vnm_print(2, "no");
00905                 break;
00906             case PCE_TOTAL:
00907                 Vnm_print(2, "total");
00908                 break;
00909             case PCE_COMPS:
00910                 Vnm_print(2, "comps");
00911                 break;
00912             default:
00913                 Vnm_print(2, "UNKNOWN");
00914                 break;
00915         }
00916         Vnm_print(2, "\" instead.\n");
00917         return 1;
00918     } else if (Vstring_strcasecmp(tok, "no") == 0) {
00919         thee->calcenergy = PCE_NO;
00920         thee->setcalcenergy = 1;
00921         return 1;
00922     } else if (Vstring_strcasecmp(tok, "total") == 0) {
00923         thee->calcenergy = PCE_TOTAL;
00924         thee->setcalcenergy = 1;
00925         return 1;
00926     } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00927         thee->calcenergy = PCE_COMPS;
00928         thee->setcalcenergy = 1;
00929         return 1;
00930     } else {
00931         Vnm_print(2, "NOsh:  Unrecognized parameter (%s) while parsing \
00932 calcenergy!\n", tok);
00933         return -1;
00934     }
00935     return 0;
00936
00937     VERROR1:
00938         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00939         return -1;
00940 }
00941
00942 VPRIVATE int PBEparm_parseCALCFORCE(PBEparm *thee, Vio *sock) {
```

```
00943     char tok[VMAX_BUFSIZE];
00944     int ti;
00945
00946     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00947     /* Parse number */
00948     if (sscanf(tok, "%d", &ti) == 1) {
00949         thee->calcforce = ti;
00950         thee->setcalcforce = 1;
00951
00952         Vnm_print(2, "parsePBE:  Warning -- parsed deprecated \"calcforce \
00953 %d\" statement.\n", ti);
00954         Vnm_print(2, "parsePBE:  Please use \"calcforce ");
00955         switch (thee->calcenergy) {
00956             case PCF_NO:
00957                 Vnm_print(2, "no");
00958                 break;
00959             case PCF_TOTAL:
00960                 Vnm_print(2, "total");
00961                 break;
00962             case PCF_COMPS:
00963                 Vnm_print(2, "comps");
00964                 break;
00965             default:
00966                 Vnm_print(2, "UNKNOWN");
00967                 break;
00968         }
00969         Vnm_print(2, "\" instead.\n");
00970         return 1;
00971     } else if (Vstring_strcasecmp(tok, "no") == 0) {
00972         thee->calcforce = PCF_NO;
00973         thee->setcalcforce = 1;
00974         return 1;
00975     } else if (Vstring_strcasecmp(tok, "total") == 0) {
00976         thee->calcforce = PCF_TOTAL;
00977         thee->setcalcforce = 1;
00978         return 1;
00979     } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00980         thee->calcforce = PCF_COMPS;
00981         thee->setcalcforce = 1;
00982         return 1;
00983     } else {
00984         Vnm_print(2, "NOsh:  Unrecognized parameter (%s) while parsing \
00985 calcforce!\n", tok);
00986         return -1;
00987     }
00988     return 0;
00989
00990     VERROR1:
00991         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00992         return -1;
00993 }
00994
00995 /*-----------------------------------------------------------*/
00996 /* Added by Michael Grabe                                    */
00997 /*-----------------------------------------------------------*/
00998
00999 VPRIVATE int PBEparm_parseZMEM(PBEparm *thee, Vio *sock) {
```

```
01000     char tok[VMAX_BUFSIZE];
01001     double tf;
01002
01003     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01004     if (sscanf(tok, "%lf", &tf) == 0) {
01005         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing ZMEM \
01006       keyword!\n", tok);
01007         return -1;
01008     }
01009     thee->zmem = tf;
01010     thee->setzmem = 1;
01011     return 1;
01012
01013 VERROR1:
01014  Vnm_print(2, "parsePBE:  ran out of tokens!\n");
01015  return -1;
01016 }
01017
01018
01019 VPRIVATE int PBEparm_parseLMEM(PBEparm *thee, Vio *sock) {
01020     char tok[VMAX_BUFSIZE];
01021     double tf;
01022
01023     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01024     if (sscanf(tok, "%lf", &tf) == 0) {
01025         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing LMEM \
01026       keyword!\n", tok);
01027         return -1;
01028     }
01029     thee->Lmem = tf;
01030     thee->setLmem = 1;
01031     return 1;
01032
01033 VERROR1:
01034  Vnm_print(2, "parsePBE:  ran out of tokens!\n");
01035  return -1;
01036 }
01037
01038 VPRIVATE int PBEparm_parseMDIE(PBEparm *thee, Vio *sock) {
01039     char tok[VMAX_BUFSIZE];
01040     double tf;
01041
01042     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01043     if (sscanf(tok, "%lf", &tf) == 0) {
01044         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing MDIE \
01045       keyword!\n", tok);
01046         return -1;
01047     }
01048     thee->mdie = tf;
01049     thee->setmdie  = 1;
01050     return 1;
01051
01052 VERROR1:
01053  Vnm_print(2, "parsePBE:  ran out of tokens!\n");
01054  return -1;
01055 }
01056
```

```
01057 VPRIVATE int PBEparm_parseMEMV(PBEparm *thee, Vio *sock) {
01058     char tok[VMAX_BUFSIZE];
01059     double tf;
01060
01061     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01062     if (sscanf(tok, "%lf", &tf) == 0) {
01063         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing MEMV \
01064       keyword!\n", tok);
01065         return -1;
01066     }
01067     thee->memv = tf;
01068     thee->setmemv = 1;
01069     return 1;
01070
01071 VERROR1:
01072  Vnm_print(2, "parsePBE:  ran out of tokens!\n");
01073  return -1;
01074 }
01075
01076 /*-----------------------------------------------------------*/
01077
01078 VPRIVATE int PBEparm_parseWRITE(PBEparm *thee, Vio *sock) {
01079     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
01080     Vdata_Type writetype;
01081     Vdata_Format writefmt;
01082
01083     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01084     if (Vstring_strcasecmp(tok, "pot") == 0) {
01085         writetype = VDT_POT;
01086     } else if (Vstring_strcasecmp(tok, "atompot") == 0) {
01087         writetype = VDT_ATOMPOT;
01088     }  else if (Vstring_strcasecmp(tok, "charge") == 0) {
01089         writetype = VDT_CHARGE;
01090     } else if (Vstring_strcasecmp(tok, "smol") == 0) {
01091         writetype = VDT_SMOL;
01092     } else if (Vstring_strcasecmp(tok, "dielx") == 0) {
01093         writetype = VDT_DIELX;
01094     } else if (Vstring_strcasecmp(tok, "diely") == 0) {
01095         writetype = VDT_DIELY;
01096     } else if (Vstring_strcasecmp(tok, "dielz") == 0) {
01097         writetype = VDT_DIELZ;
01098     } else if (Vstring_strcasecmp(tok, "kappa") == 0) {
01099         writetype = VDT_KAPPA;
01100     } else if (Vstring_strcasecmp(tok, "sspl") == 0) {
01101         writetype = VDT_SSPL;
01102     } else if (Vstring_strcasecmp(tok, "vdw") == 0) {
01103         writetype = VDT_VDW;
01104     } else if (Vstring_strcasecmp(tok, "ivdw") == 0) {
01105         writetype = VDT_IVDW;
01106     } else if (Vstring_strcasecmp(tok, "lap") == 0) {
01107         writetype = VDT_LAP;
01108     } else if (Vstring_strcasecmp(tok, "edens") == 0) {
01109         writetype = VDT_EDENS;
01110     } else if (Vstring_strcasecmp(tok, "ndens") == 0) {
01111         writetype = VDT_NDENS;
01112     } else if (Vstring_strcasecmp(tok, "qdens") == 0) {
01113         writetype = VDT_QDENS;
```

```
01114     } else {
01115         Vnm_print(2, "PBEparm_parse:  Invalid data type (%s) to write!\n",
01116            tok);
01117         return -1;
01118     }
01119     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01120     if (Vstring_strcasecmp(tok, "dx") == 0) {
01121         writefmt = VDF_DX;
01122     } else if (Vstring_strcasecmp(tok, "uhbd") == 0) {
01123         writefmt = VDF_UHBD;
01124     } else if (Vstring_strcasecmp(tok, "avs") == 0) {
01125         writefmt = VDF_AVS;
01126     } else if (Vstring_strcasecmp(tok, "gz") == 0) {
01127         writefmt = VDF_GZ;
01128     } else if (Vstring_strcasecmp(tok, "flat") == 0) {
01129         writefmt = VDF_FLAT;
01130     } else {
01131         Vnm_print(2, "PBEparm_parse:  Invalid data format (%s) to write!\n",
01132            tok);
01133         return -1;
01134     }
01135     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01136     if (tok[0]=='"') {
01137         strcpy(strnew, "");
01138         while (tok[strlen(tok)-1] != '"') {
01139             strcat(str, tok);
01140             strcat(str, " ");
01141             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01142         }
01143         strcat(str, tok);
01144         strncpy(strnew, str+1, strlen(str)-2);
01145         strcpy(tok, strnew);
01146     }
01147  if (thee->numwrite < (PBEPARM_MAXWRITE-1)) {
01148   strncpy(thee->writestem[thee->numwrite], tok, VMAX_ARGLEN);
01149   thee->writetype[thee->numwrite] = writetype;
01150   thee->writefmt[thee->numwrite] = writefmt;
01151   (thee->numwrite)++;
01152  } else {
01153   Vnm_print(2, "PBEparm_parse:  You have exceeded the maximum number of write sta
      tements!\n");
01154   Vnm_print(2, "PBEparm_parse:  Ignoring additional write statements!\n");
01155  }
01156     return 1;
01157
01158     VERROR1:
01159         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
01160         return -1;
01161 }
01162
01163 VPRIVATE int PBEparm_parseWRITEMAT(PBEparm *thee, Vio *sock) {
01164     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
01165
01166     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01167     if (Vstring_strcasecmp(tok, "poisson") == 0) {
01168         thee->writematflag = 0;
01169     } else if (Vstring_strcasecmp(tok, "full") == 0) {
```

```
01170         thee->writematflag = 1;
01171     } else {
01172         Vnm_print(2, "NOsh:  Invalid format (%s) while parsing \
01173 WRITEMAT keyword!\n", tok);
01174         return -1;
01175     }
01176 
01177     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01178     if (tok[0]=='"') {
01179         strcpy(strnew, "");
01180         while (tok[strlen(tok)-1] != '"') {
01181             strcat(str, tok);
01182             strcat(str, " ");
01183             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01184         }
01185         strcat(str, tok);
01186         strncpy(strnew, str+1, strlen(str)-2);
01187         strcpy(tok, strnew);
01188     }
01189     strncpy(thee->writematstem, tok, VMAX_ARGLEN);
01190     thee->setwritemat = 1;
01191     thee->writemat = 1;
01192     return 1;
01193 
01194     VERROR1:
01195         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
01196         return -1;
01197 
01198 }
01199 
01200 VPUBLIC int PBEparm_parseToken(PBEparm *thee, char tok[VMAX_BUFSIZE],
01201   Vio *sock) {
01202 
01203     if (thee == VNULL) {
01204         Vnm_print(2, "parsePBE:  got NULL thee!\n");
01205         return -1;
01206     }
01207     if (sock == VNULL) {
01208         Vnm_print(2, "parsePBE:  got NULL socket!\n");
01209         return -1;
01210     }
01211 
01212     Vnm_print(0, "PBEparm_parseToken:  trying %s...\n", tok);
01213 
01214     if (Vstring_strcasecmp(tok, "mol") == 0) {
01215         return PBEparm_parseMOL(thee, sock);
01216     } else if (Vstring_strcasecmp(tok, "lpbe") == 0) {
01217         return PBEparm_parseLPBE(thee, sock);
01218     } else if (Vstring_strcasecmp(tok, "npbe") == 0) {
01219         return PBEparm_parseNPBE(thee, sock);
01220     } else if (Vstring_strcasecmp(tok, "lrpbe") == 0) {
01221         return PBEparm_parseLRPBE(thee, sock);
01222     } else if (Vstring_strcasecmp(tok, "nrpbe") == 0) {
01223         return PBEparm_parseNRPBE(thee, sock);
01224     } else if (Vstring_strcasecmp(tok, "smpbe") == 0) {
01225         return PBEparm_parseSMPBE(thee, sock);
01226     } else if (Vstring_strcasecmp(tok, "bcfl") == 0) {
```

```
01227         return PBEparm_parseBCFL(thee, sock);
01228     } else if (Vstring_strcasecmp(tok, "ion") == 0) {
01229         return PBEparm_parseION(thee, sock);
01230     } else if (Vstring_strcasecmp(tok, "pdie") == 0) {
01231         return PBEparm_parsePDIE(thee, sock);
01232     } else if (Vstring_strcasecmp(tok, "sdens") == 0) {
01233         return PBEparm_parseSDENS(thee, sock);
01234     } else if (Vstring_strcasecmp(tok, "sdie") == 0) {
01235         return PBEparm_parseSDIE(thee, sock);
01236     } else if (Vstring_strcasecmp(tok, "srfm") == 0) {
01237         return PBEparm_parseSRFM(thee, sock);
01238     } else if (Vstring_strcasecmp(tok, "srad") == 0) {
01239         return PBEparm_parseSRAD(thee, sock);
01240     } else if (Vstring_strcasecmp(tok, "swin") == 0) {
01241         return PBEparm_parseSWIN(thee, sock);
01242     } else if (Vstring_strcasecmp(tok, "temp") == 0) {
01243         return PBEparm_parseTEMP(thee, sock);
01244     } else if (Vstring_strcasecmp(tok, "usemap") == 0) {
01245         return PBEparm_parseUSEMAP(thee, sock);
01246     } else if (Vstring_strcasecmp(tok, "calcenergy") == 0) {
01247         return PBEparm_parseCALCENERGY(thee, sock);
01248     } else if (Vstring_strcasecmp(tok, "calcforce") == 0) {
01249         return PBEparm_parseCALCFORCE(thee, sock);
01250     } else if (Vstring_strcasecmp(tok, "write") == 0) {
01251         return PBEparm_parseWRITE(thee, sock);
01252     } else if (Vstring_strcasecmp(tok, "writemat") == 0) {
01253         return PBEparm_parseWRITEMAT(thee, sock);
01254 
01255 /*-----------------------------------------------------------*/
01256 /* Added by Michael Grabe                                    */
01257 /*-----------------------------------------------------------*/
01258 
01259     } else if (Vstring_strcasecmp(tok, "zmem") == 0) {
01260         return PBEparm_parseZMEM(thee, sock);
01261     } else if (Vstring_strcasecmp(tok, "Lmem") == 0) {
01262         return PBEparm_parseLMEM(thee, sock);
01263     } else if (Vstring_strcasecmp(tok, "mdie") == 0) {
01264         return PBEparm_parseMDIE(thee, sock);
01265     } else if (Vstring_strcasecmp(tok, "memv") == 0) {
01266         return PBEparm_parseMEMV(thee, sock);
01267     }
01268 
01269 /*-----------------------------------------------------------*/
01270 
01271     return 0;
01272 
01273 }
```

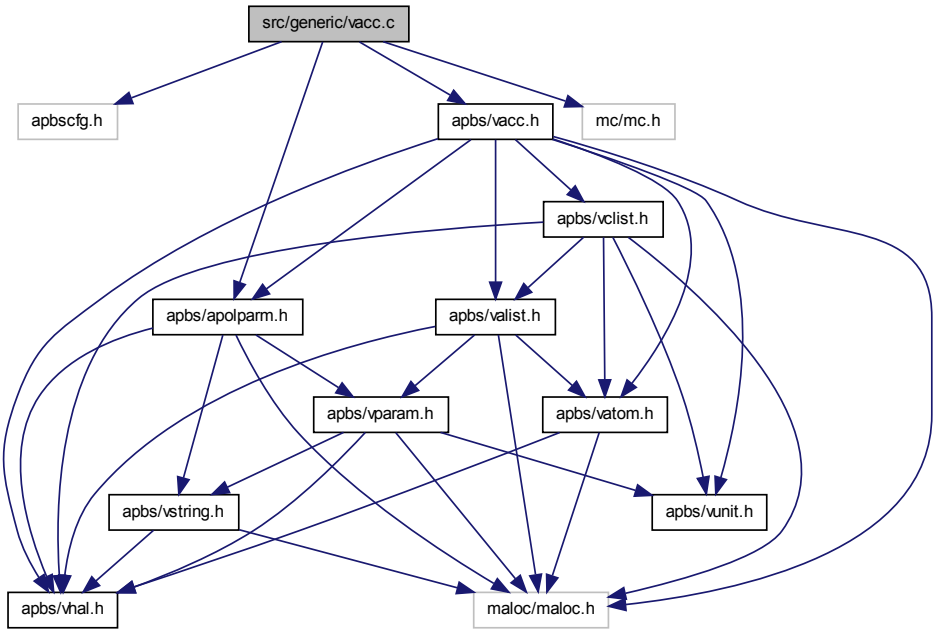
10.61 src/generic/vacc.c File Reference

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 derivatve 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 derivatve 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 1605 2010-09-13 15:12:09Z yhuang01

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
* 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 Washington University in St. Louis 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

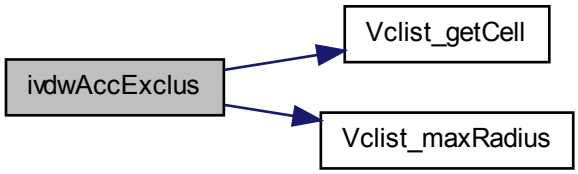
Nathan Baker

Parameters

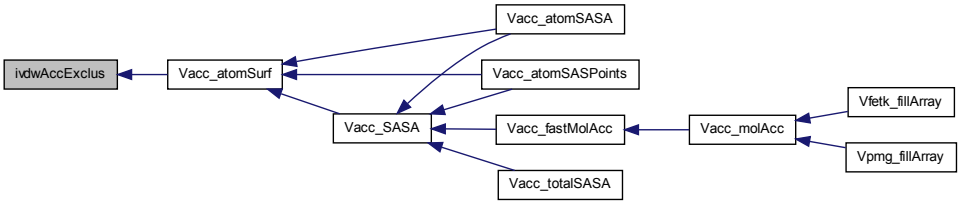
center	Accessibility object
radius	Position to test
atomID	Radius of probe ID of atom to ignore

Definition at line 77 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

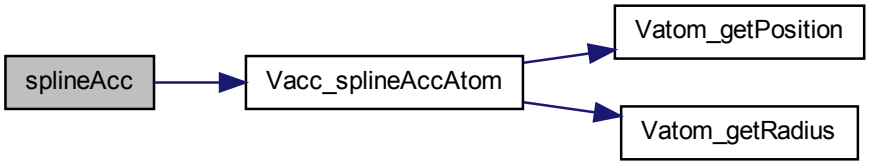
Todd Dolinsky and Nathan Baker

Parameters

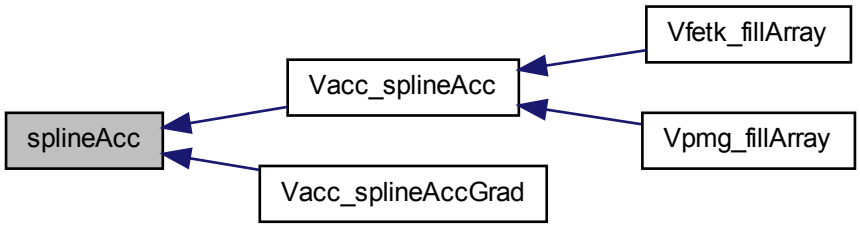
center	Accessibility object
win	Point at which the acc is to be evaluated
infrad	Spline window
cell	Radius to inflate atomic radius Cell of atom objects

Definition at line 429 of file vacc.c.

Here is the call graph for this function:



Here is the caller graph for this function:

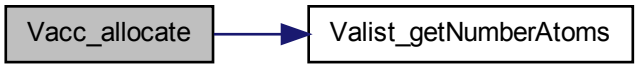


10.61.2.3 VPRIVATE int Vacc_allocate (Vacc * *thee*)

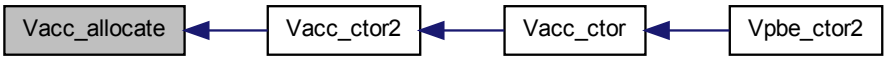
Allocate (and clear) space for storage

Definition at line 176 of file vacc.c.

Here is the call graph for this function:



Here is the caller graph for this function:

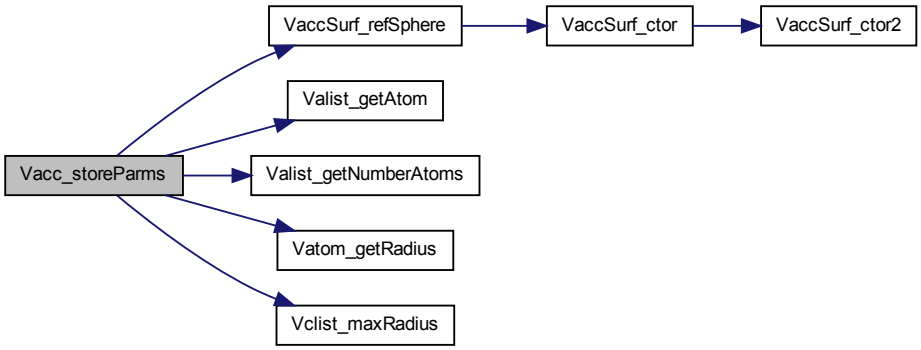


10.61.2.4 VPRIVATE int Vacc_storeParms (Vacc * *thee,* Valist * *alist,* Vclist * *clist,* double *surf_density*)

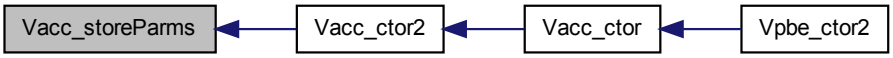
Check and store parameters passed to constructor

Definition at line 137 of file vacc.c.

Here is the call graph for this function:



Here is the caller graph for this function:



10.62 src/generic/vacc.c

```
00001
00049 #include "apbscfg.h"
00050 #include "apbs/vacc.h"
00051 #include "apbs/apolparm.h"
00052
00053 #if defined(HAVE_MC_H)
00054 #include "mc/mc.h"
00055 #endif
00056
00057 VEMBED(rcsid="$Id: vacc.c 1605 2010-09-13 15:12:09Z yhuang01 $")
00058
00059 #if !defined(VINLINE_VACC)
00060
00061 VPUBLIC unsigned long int Vacc_memChk(Vacc *thee) {
```

```
00062     if (thee == VNULL) return 0;
00063     return Vmem_bytes(thee->mem);
00064 }
00065
00066 #endif /* if !defined(VINLINE_VACC) */
00067
00077 VPRIVATE int ivdwAccExclus(
00078          Vacc *thee,
00079          double center[3],
00080          double radius,
00081          int atomID
00082          ) {
00083
00084     int iatom;
00085     double dist2, *apos;
00086     Vatom *atom;
00087     VclistCell *cell;
00088
00089     VASSERT(thee != VNULL);
00090
00091     /* We can only test probes with radii less than the max specified */
00092     if (radius > Vclist_maxRadius(thee->clist)) {
00093         Vnm_print(2,
00094       "Vacc_ivdwAcc: got radius (%g) bigger than max radius (%g)\n",
00095       radius, Vclist_maxRadius(thee->clist));
00096   VASSERT(0);
00097     }
00098
00099     /* Get the relevant cell from the cell list */
00100     cell = Vclist_getCell(thee->clist, center);
00101
00102     /* If we have no cell, then no atoms are nearby and we're definitely
00103      * accessible */
00104     if (cell == VNULL) {
00105         return 1;
00106     }
00107
00108     /* Otherwise, check for overlap with the atoms in the cell */
00109     for (iatom=0; iatom<cell->natoms; iatom++) {
00110         atom = cell->atoms[iatom];
00111   apos = atom->position;
00112   dist2 = VSQR(center[0]-apos[0]) + VSQR(center[1]-apos[1])
00113       + VSQR(center[2]-apos[2]);
00114   if (dist2 < VSQR(atom->radius+radius)){
00115    if (atom->id != atomID) return 0;
00116   }
00117  }
00118
00119     /* If we're still here, then the point is accessible */
00120     return 1;
00121
00122 }
00123
00124 VPUBLIC Vacc* Vacc_ctor(Valist *alist, Vclist *clist, double surf_density) {
00125
00126
00127     Vacc *thee = VNULL;
```

```
00128
00129     /* Set up the structure */
00130     thee = Vmem_malloc(VNULL, 1, sizeof(Vacc) );
00131     VASSERT( thee != VNULL);
00132     VASSERT( Vacc_ctor2(thee, alist, clist, surf_density));
00133     return thee;
00134 }
00135
00137 VPRIVATE int Vacc_storeParms(Vacc *thee, Valist *alist, Vclist *clist,
00138         double surf_density) {
00139
00140     int nsphere, iatom;
00141     double maxrad, maxarea, rad;
00142     Vatom *atom;
00143
00144     if (alist == VNULL) {
00145         Vnm_print(2, "Vacc_storeParms:  Got NULL Valist!\n");
00146         return 0;
00147     } else thee->alist = alist;
00148     if (clist == VNULL) {
00149         Vnm_print(2, "Vacc_storeParms:  Got NULL Vclist!\n");
00150         return 0;
00151     } else thee->clist = clist;
00152     thee->surf_density = surf_density;
00153
00154     /* Loop through the atoms to determine the maximum radius */
00155     maxrad = 0.0;
00156     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
00157         atom = Valist_getAtom(alist, iatom);
00158         rad = Vatom_getRadius(atom);
00159         if (rad > maxrad) maxrad = rad;
00160     }
00161     maxrad = maxrad + Vclist_maxRadius(thee->clist);
00162
00163     maxarea = 4.0*VPI*maxrad*maxrad;
00164     nsphere = (int)ceil(maxarea*surf_density);
00165
00166     Vnm_print(0, "Vacc_storeParms:  Surf. density = %g\n", surf_density);
00167     Vnm_print(0, "Vacc_storeParms:  Max area = %g\n", maxarea);
00168     thee->refSphere = VaccSurf_refSphere(thee->mem, nsphere);
00169     Vnm_print(0, "Vacc_storeParms:  Using %d-point reference sphere\n",
00170             thee->refSphere->npts);
00171
00172     return 1;
00173 }
00174
00176 VPRIVATE int Vacc_allocate(Vacc *thee) {
00177
00178     int i, natoms;
00179
00180     natoms = Valist_getNumberAtoms(thee->alist);
00181
00182     thee->atomFlags = Vmem_malloc(thee->mem, natoms, sizeof(int));
00183     if (thee->atomFlags == VNULL) {
00184         Vnm_print(2,
00185                 "Vacc_allocate:  Failed to allocate %d (int)s for atomFlags!\n",
00186                  natoms);
```

```
00187         return 0;
00188     }
00189     for (i=0; i<natoms; i++) (thee->atomFlags)[i] = 0;
00190
00191     return 1;
00192 }
00193
00194
00195 VPUBLIC int Vacc_ctor2(Vacc *thee, Valist *alist, Vclist *clist,
00196     double surf_density) {
00197
00198     /* Check and store parameters */
00199     if (!Vacc_storeParms(thee, alist, clist, surf_density)) {
00200         Vnm_print(2, "Vacc_ctor2:  parameter check failed!\n");
00201         return 0;
00202     }
00203
00204     /* Set up memory management object */
00205     thee->mem = Vmem_ctor("APBS::VACC");
00206     if (thee->mem == VNULL) {
00207         Vnm_print(2, "Vacc_ctor2:  memory object setup failed!\n");
00208         return 0;
00209     }
00210
00211     /* Setup and check probe */
00212     thee->surf = VNULL;
00213
00214     /* Allocate space */
00215     if (!Vacc_allocate(thee)) {
00216         Vnm_print(2, "Vacc_ctor2:  memory allocation failed!\n");
00217         return 0;
00218     }
00219
00220     return 1;
00221 }
00222
00223
00224 VPUBLIC void Vacc_dtor(Vacc **thee) {
00225
00226     if ((*thee) != VNULL) {
00227         Vacc_dtor2(*thee);
00228         Vmem_free(VNULL, 1, sizeof(Vacc), (void **)thee);
00229         (*thee) = VNULL;
00230     }
00231
00232 }
00233
00234 VPUBLIC void Vacc_dtor2(Vacc *thee) {
00235
00236     int i, natoms;
00237
00238     natoms = Valist_getNumberAtoms(thee->alist);
00239     Vmem_free(thee->mem, natoms, sizeof(int), (void **)&(thee->atomFlags));
00240
00241     if (thee->refSphere != VNULL) {
00242         VaccSurf_dtor(&(thee->refSphere));
00243         thee->refSphere = VNULL;
```

```
00244     }
00245     if (thee->surf != VNULL) {
00246         for (i=0; i<natoms; i++) VaccSurf_dtor(&(thee->surf[i]));
00247         Vmem_free(thee->mem, natoms, sizeof(VaccSurf *),
00248                   (void **)&(thee->surf));
00249         thee->surf = VNULL;
00250     }
00251
00252     Vmem_dtor(&(thee->mem));
00253 }
00254
00255 VPUBLIC double Vacc_vdwAcc(Vacc *thee, double center[3]) {
00256
00257     VclistCell *cell;
00258     Vatom *atom;
00259     int iatom;
00260     double *apos;
00261     double dist2;
00262
00263     /* Get the relevant cell from the cell list */
00264     cell = Vclist_getCell(thee->clist, center);
00265
00266     /* If we have no cell, then no atoms are nearby and we're definitely
00267      * accessible */
00268     if (cell == VNULL) return 1.0;
00269
00270     /* Otherwise, check for overlap with the atoms in the cell */
00271     for (iatom=0; iatom<cell->natoms; iatom++) {
00272         atom = cell->atoms[iatom];
00273         apos = Vatom_getPosition(atom);
00274         dist2 = VSQR(center[0]-apos[0]) + VSQR(center[1]-apos[1])
00275                 + VSQR(center[2]-apos[2]);
00276         if (dist2 < VSQR(Vatom_getRadius(atom))) return 0.0;
00277     }
00278
00279     /* If we're still here, then the point is accessible */
00280     return 1.0;
00281 }
00282
00283 VPUBLIC double Vacc_ivdwAcc(Vacc *thee, double center[3], double radius) {
00284
00285     return (double)ivdwAccExclus(thee, center, radius, -1);
00286
00287 }
00288
00289 VPUBLIC void Vacc_splineAccGradAtomNorm(Vacc *thee, double center[VAPBS_DIM],
00290         double win, double infrad, Vatom *atom, double *grad) {
00291
00292     int i;
00293     double dist, *apos, arad, sm, sm2, w2i, w3i, mygrad;
00294     double mychi = 1.0;           /* Char. func. value for given atom */
00295
00296     VASSERT(thee != NULL);
00297
00298     /* Inverse squared window parameter */
00299     w2i = 1.0/(win*win);
00300     w3i = 1.0/(win*win*win);
```

```
00301
00302     /* The grad is zero by default */
00303     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
00304
00305     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
00306      * *** MAGNITUDE OF THE FORCE *** */
00307     apos = Vatom_getPosition(atom);
00308     /* Zero-radius atoms don't contribute */
00309     if (Vatom_getRadius(atom) > 0.0) {
00310         arad = Vatom_getRadius(atom) + infrad;
00311         dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
00312           + VSQR(apos[2]-center[2]));
00313         /* If we're inside an atom, the entire characteristic function
00314          * will be zero and the grad will be zero, so we can stop */
00315         if (dist < (arad - win)) return;
00316         /* Likewise, if we're outside the smoothing window, the characteristic
00317          * function is unity and the grad will be zero, so we can stop */
00318         else if (dist > (arad + win)) return;
00319         /* Account for floating point error at the border
00320          * NAB:  COULDN'T THESE TESTS BE COMBINED AS BELOW
00321          * (Vacc_splineAccAtom)? */
00322         else if ((VABS(dist - (arad - win)) < VSMALL) ||
00323                  (VABS(dist - (arad + win)) < VSMALL)) return;
00324         /* If we're inside the smoothing window */
00325         else {
00326             sm = dist - arad + win;
00327             sm2 = VSQR(sm);
00328             mychi = 0.75*sm2*w2i -0.25*sm*sm2*w3i;
00329             mygrad = 1.5*sm*w2i - 0.75*sm2*w3i;
00330         }
00331         /* Now assemble the grad vector */
00332         VASSERT(mychi > 0.0);
00333         for (i=0; i<VAPBS_DIM; i++)
00334             grad[i] = -(mygrad/mychi)*((center[i] - apos[i])/dist);
00335     }
00336 }
00337
00338 VPUBLIC void Vacc_splineAccGradAtomUnnorm(Vacc *thee, double center[VAPBS_DIM],
00339         double win, double infrad, Vatom *atom, double *grad) {
00340
00341     int i;
00342     double dist, *apos, arad, sm, sm2, w2i, w3i, mygrad;
00343     double mychi = 1.0;           /* Char. func. value for given atom */
00344
00345     VASSERT(thee != NULL);
00346
00347     /* Inverse squared window parameter */
00348     w2i = 1.0/(win*win);
00349     w3i = 1.0/(win*win*win);
00350
00351     /* The grad is zero by default */
00352     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
00353
00354     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
00355      * *** MAGNITUDE OF THE FORCE *** */
00356     apos = Vatom_getPosition(atom);
00357     /* Zero-radius atoms don't contribute */
```

```
00358     if (Vatom_getRadius(atom) > 0.0) {
00359         arad = Vatom_getRadius(atom) + infrad;
00360         dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
00361           + VSQR(apos[2]-center[2]));
00362         /* If we're inside an atom, the entire characteristic function
00363          * will be zero and the grad will be zero, so we can stop */
00364         if (dist < (arad - win)) return;
00365         /* Likewise, if we're outside the smoothing window, the characteristic
00366          * function is unity and the grad will be zero, so we can stop */
00367         else if (dist > (arad + win)) return;
00368         /* Account for floating point error at the border
00369          * NAB:  COULDN'T THESE TESTS BE COMBINED AS BELOW
00370          * (Vacc_splineAccAtom)? */
00371         else if ((VABS(dist - (arad - win)) < VSMALL) ||
00372                  (VABS(dist - (arad + win)) < VSMALL)) return;
00373         /* If we're inside the smoothing window */
00374         else {
00375             sm = dist - arad + win;
00376             sm2 = VSQR(sm);
00377             mychi = 0.75*sm2*w2i -0.25*sm*sm2*w3i;
00378             mygrad = 1.5*sm*w2i - 0.75*sm2*w3i;
00379         }
00380         /* Now assemble the grad vector */
00381         VASSERT(mychi > 0.0);
00382         for (i=0; i<VAPBS_DIM; i++)
00383             grad[i] = -(mygrad)*((center[i] - apos[i])/dist);
00384     }
00385 }
00386
00387 VPUBLIC double Vacc_splineAccAtom(Vacc *thee, double center[VAPBS_DIM],
00388         double win, double infrad, Vatom *atom) {
00389
00390     double dist, *apos, arad, sm, sm2, w2i, w3i, value, stot, sctot;
00391
00392     VASSERT(thee != NULL);
00393
00394     /* Inverse squared window parameter */
00395     w2i = 1.0/(win*win);
00396     w3i = 1.0/(win*win*win);
00397
00398     apos = Vatom_getPosition(atom);
00399     /* Zero-radius atoms don't contribute */
00400     if (Vatom_getRadius(atom) > 0.0) {
00401         arad = Vatom_getRadius(atom) + infrad;
00402         stot = arad + win;
00403         sctot = VMAX2(0, (arad - win));
00404         dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
00405           + VSQR(apos[2]-center[2]));
00406         /* If we're inside an atom, the entire characteristic function
00407          * will be zero */
00408         if ((dist < sctot) || (VABS(dist - sctot) < VSMALL)){
00409             value = 0.0;
00410         /* We're outside the smoothing window */
00411         } else if ((dist > stot) || (VABS(dist - stot) < VSMALL)) {
00412             value = 1.0;
00413         /* We're inside the smoothing window */
00414         } else {
```

```
00415             sm = dist - arad + win;
00416             sm2 = VSQR(sm);
00417             value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
00418         }
00419     } else value = 1.0;
00420
00421     return value;
00422 }
00423
00429 VPRIVATE double splineAcc(
00430         Vacc *thee,
00431         double center[VAPBS_DIM],
00433         double win,
00434         double infrad,
00435         VclistCell *cell
00436         ) {
00437
00438     int atomID, iatom;
00439     Vatom *atom;
00440     double value = 1.0;
00441
00442     VASSERT(thee != NULL);
00443
00444     /* Now loop through the atoms assembling the characteristic function */
00445     for (iatom=0; iatom<cell->natoms; iatom++) {
00446
00447         atom = cell->atoms[iatom];
00448         atomID = atom->id;
00449
00450         /* Check to see if we've counted this atom already */
00451         if ( !(thee->atomFlags[atomID]) ) {
00452
00453             thee->atomFlags[atomID] = 1;
00454             value *= Vacc_splineAccAtom(thee, center, win, infrad, atom);
00455
00456             if (value < VSMALL) return value;
00457         }
00458     }
00459
00460     return value;
00461 }
00462
00463
00464 VPUBLIC double Vacc_splineAcc(Vacc *thee, double center[VAPBS_DIM], double win,
00465   double infrad) {
00466
00467     VclistCell *cell;
00468     Vatom *atom;
00469     int iatom, atomID;
00470
00471
00472     VASSERT(thee != NULL);
00473
00474     if (Vclist_maxRadius(thee->clist) < (win + infrad)) {
00475         Vnm_print(2, "Vacc_splineAcc:  Vclist has max_radius=%g;\n",
00476                 Vclist_maxRadius(thee->clist));
00477         Vnm_print(2, "Vacc_splineAcc:  Insufficient for win=%g, infrad=%g\n",
```

```
00478                 win, infrad);
00479         VASSERT(0);
00480     }
00481
00482     /* Get a cell or VNULL; in the latter case return 1.0 */
00483     cell = Vclist_getCell(thee->clist, center);
00484     if (cell == VNULL) return 1.0;
00485
00486     /* First, reset the list of atom flags
00487      * NAB:  THIS SEEMS VERY INEFFICIENT */
00488     for (iatom=0; iatom<cell->natoms; iatom++) {
00489         atom = cell->atoms[iatom];
00490         atomID = atom->id;
00491         thee->atomFlags[atomID] = 0;
00492     }
00493
00494     return splineAcc(thee, center, win, infrad, cell);
00495 }
00496
00497 VPUBLIC void Vacc_splineAccGrad(Vacc *thee, double center[VAPBS_DIM],
00498         double win, double infrad, double *grad) {
00499
00500     int iatom, i, atomID;
00501     double acc = 1.0;
00502     double tgrad[VAPBS_DIM];
00503     VclistCell *cell;
00504     Vatom *atom = VNULL;
00505
00506     VASSERT(thee != NULL);
00507
00508     if (Vclist_maxRadius(thee->clist) < (win + infrad)) {
00509         Vnm_print(2, "Vacc_splineAccGrad: Vclist max_radius=%g;\n",
00510                 Vclist_maxRadius(thee->clist));
00511         Vnm_print(2, "Vacc_splineAccGrad: Insufficient for win=%g, infrad=%g\n",

00512                 win, infrad);
00513         VASSERT(0);
00514     }
00515
00516     /* Reset the gradient */
00517     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
00518
00519     /* Get the cell; check for nullity */
00520     cell = Vclist_getCell(thee->clist, center);
00521     if (cell == VNULL) return;
00522
00523     /* Reset the list of atom flags */
00524     for (iatom=0; iatom<cell->natoms; iatom++) {
00525         atom = cell->atoms[iatom];
00526         atomID = atom->id;
00527         thee->atomFlags[atomID] = 0;
00528     }
00529
00530     /* Get the local accessibility */
00531     acc = splineAcc(thee, center, win, infrad, cell);
00532
00533     /* Accumulate the gradient of all local atoms */
```

```
00534     if (acc > VSMALL) {
00535         for (iatom=0; iatom<cell->natoms; iatom++) {
00536             atom = cell->atoms[iatom];
00537             Vacc_splineAccGradAtomNorm(thee, center, win, infrad, atom, tgrad);
00538         }
00539         for (i=0; i<VAPBS_DIM; i++) grad[i] += tgrad[i];
00540     }
00541     for (i=0; i<VAPBS_DIM; i++) grad[i] *= -acc;
00542 }
00543
00544 VPUBLIC double Vacc_molAcc(Vacc *thee, double center[VAPBS_DIM],
00545         double radius) {
00546
00547     double rc;
00548
00549     /* ******* CHECK IF OUTSIDE ATOM+PROBE RADIUS SURFACE ***** */
00550     if (Vacc_ivdwAcc(thee, center, radius) == 1.0) {
00551
00552         /* Vnm_print(2, "DEBUG:  ivdwAcc = 1.0\n"); */
00553         rc = 1.0;
00554
00555     /* ******* CHECK IF INSIDE ATOM RADIUS SURFACE ***** */
00556     } else if (Vacc_vdwAcc(thee, center) == 0.0) {
00557
00558         /* Vnm_print(2, "DEBUG:  vdwAcc = 0.0\n"); */
00559         rc = 0.0;
00560
00561     /* ******* CHECK IF OUTSIDE MOLECULAR SURFACE ***** */
00562     } else {
00563
00564         /* Vnm_print(2, "DEBUG:  calling fastMolAcc...\n"); */
00565         rc = Vacc_fastMolAcc(thee, center, radius);
00566
00567     }
00568
00569     return rc;
00570
00571 }
00572
00573 VPUBLIC double Vacc_fastMolAcc(Vacc *thee, double center[VAPBS_DIM],
00574         double radius) {
00575
00576     Vatom *atom;
00577     VaccSurf *surf;
00578     VclistCell *cell;
00579     int ipt, iatom, atomID;
00580     double dist2, rad2;
00581
00582     rad2 = radius*radius;
00583
00584     /* Check to see if the SAS has been defined */
00585     if (thee->surf == VNULL) Vacc_SASA(thee, radius);
00586
00587     /* Get the cell associated with this point */
00588     cell = Vclist_getCell(thee->clist, center);
00589     if (cell == VNULL) {
00590         Vnm_print(2, "Vacc_fastMolAcc:  unexpected VNULL VclistCell!\n");
```

```
00591         return 1.0;
00592     }
00593
00594     /* Loop through all the atoms in the cell */
00595     for (iatom=0; iatom<cell->natoms; iatom++) {
00596         atom = cell->atoms[iatom];
00597         atomID = Vatom_getAtomID(atom);
00598         surf = thee->surf[atomID];
00599         /* Loop through all SAS points associated with this atom */
00600         for (ipt=0; ipt<surf->npts; ipt++) {
00601             /* See if we're within a probe radius of the point */
00602             dist2 = VSQR(center[0]-(surf->xpts[ipt]))
00603                 + VSQR(center[1]-(surf->ypts[ipt]))
00604                 + VSQR(center[2]-(surf->zpts[ipt]));
00605             if (dist2 < rad2) return 1.0;
00606         }
00607     }
00608
00609     /* If all else failed, we are not inside the molecular surface */
00610     return 0.0;
00611 }
00612
00613
00614 #if defined(HAVE_MC_H)
00615 VPUBLIC void Vacc_writeGMV(Vacc *thee, double radius, int meth, Gem *gm,
00616   char *iodev, char *iofmt, char *iohost, char *iofile) {
00617
00618     double *accVals[MAXV], coord[3];
00619     Vio *sock;
00620     int ivert, icoord;
00621
00622     for (ivert=0; ivert<MAXV; ivert++) accVals[ivert] = VNULL;
00623     accVals[0] = (void *)Vmem_malloc(thee->mem, Gem_numVV(gm), sizeof(double));
00624     accVals[1] = (void *)Vmem_malloc(thee->mem, Gem_numVV(gm), sizeof(double));
00625     for (ivert=0; ivert<Gem_numVV(gm); ivert++) {
00626         for (icoord=0;icoord<3;icoord++)
00627           coord[icoord] = VV_coord(Gem_VV(gm, ivert), icoord);
00628         if (meth == 0) {
00629             accVals[0][ivert] = Vacc_molAcc(thee, coord, radius);
00630             accVals[1][ivert] = Vacc_molAcc(thee, coord, radius);
00631         } else if (meth == 1) {
00632             accVals[0][ivert] = Vacc_ivdwAcc(thee, coord, radius);
00633             accVals[1][ivert] = Vacc_ivdwAcc(thee, coord, radius);
00634         } else if (meth == 2) {
00635             accVals[0][ivert] = Vacc_vdwAcc(thee, coord);
00636             accVals[1][ivert] = Vacc_vdwAcc(thee, coord);
00637         } else VASSERT(0);
00638     }
00639     sock = Vio_ctor(iodev, iofmt, iohost, iofile, "w");
00640     Gem_writeGMV(gm, sock, 1, accVals);
00641     Vio_dtor(&sock);
00642     Vmem_free(thee->mem, Gem_numVV(gm), sizeof(double),
00643       (void **)&(accVals[0]));
00644     Vmem_free(thee->mem, Gem_numVV(gm), sizeof(double),
00645       (void **)&(accVals[1]));
00646 }
00647 #endif /* defined(HAVE_MC_H) */
```

```
00648
00649 VPUBLIC double Vacc_SASA(Vacc *thee, double radius) {
00650
00651     int i, natom;
00652     double area, *apos;
00653     Vatom *atom;
00654     VaccSurf *asurf;
00655
00656  unsigned long long mbeg;
00657
00658     natom = Valist_getNumberAtoms(thee->alist);
00659
00660     /* Check to see if we need to build the surface */
00661     if (thee->surf == VNULL) {
00662         thee->surf = Vmem_malloc(thee->mem, natom, sizeof(VaccSurf *));
00663
00664 #if defined(DEBUG_MAC_OSX_OCL) || defined(DEBUG_MAC_OSX_STANDARD)
00665 #include "mach_chud.h"
00666   machm_(&mbeg);
00667 #pragma omp parallel for private(i,atom)
00668 #endif
00669         for (i=0; i<natom; i++) {
00670             atom = Valist_getAtom(thee->alist, i);
00671             /* NOTE:  RIGHT NOW WE DO THIS FOR THE ENTIRE MOLECULE WHICH IS
00672              * INCREDIBLY INEFFICIENT, PARTICULARLY DURING FOCUSING!!! */
00673             thee->surf[i] = Vacc_atomSurf(thee, atom, thee->refSphere,
00674             radius);
00675         }
00676     }
00677
00678     /* Calculate the area */
00679     area = 0.0;
00680     for (i=0; i<natom; i++) {
00681         atom = Valist_getAtom(thee->alist, i);
00682         asurf = thee->surf[i];
00683         /* See if this surface needs to be rebuilt */
00684         if (asurf->probe_radius != radius) {
00685             Vnm_print(2, "Vacc_SASA:  Warning -- probe radius changed from %g to
      %g!\n",
00686        asurf->probe_radius, radius);
00687             VaccSurf_dtor2(asurf);
00688             thee->surf[i] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
00689             asurf = thee->surf[i];
00690         }
00691         area += (asurf->area);
00692     }
00693
00694 #if defined(DEBUG_MAC_OSX_OCL) || defined(DEBUG_MAC_OSX_STANDARD)
00695  mets_(&mbeg, "Vacc_SASA - Parallel");
00696 #endif
00697
00698     return area;
00699
00700 }
00701
00702 VPUBLIC double Vacc_totalSASA(Vacc *thee, double radius) {
00703
```

```
00704     return Vacc_SASA(thee, radius);
00705
00706 }
00707
00708 VPUBLIC double Vacc_atomSASA(Vacc *thee, double radius, Vatom *atom) {
00709
00710     VaccSurf *asurf;
00711     int id;
00712
00713     if (thee->surf == VNULL) Vacc_SASA(thee, radius);
00714
00715     id = Vatom_getAtomID(atom);
00716     asurf = thee->surf[id];
00717
00718     /* See if this surface needs to be rebuilt */
00719     if (asurf->probe_radius != radius) {
00720         Vnm_print(2, "Vacc_SASA:  Warning -- probe radius changed from %g to %g!\
      n",
00721                 asurf->probe_radius, radius);
00722         VaccSurf_dtor2(asurf);
00723         thee->surf[id] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
00724         asurf = thee->surf[id];
00725     }
00726
00727     return asurf->area;
00728
00729 }
00730
00731 VPUBLIC VaccSurf* VaccSurf_ctor(Vmem *mem, double probe_radius, int nsphere) {
00732     VaccSurf *thee;
00733
00734     //thee = Vmem_malloc(mem, 1, sizeof(Vacc) );
00735  if (nsphere >= MAX_SPHERE_PTS) {
00736   Vnm_print(2, "VaccSurf_ctor:  Error!  The requested number of grid points (%d)
      exceeds the maximum (%d)!\n", nsphere, MAX_SPHERE_PTS);
00737   Vnm_print(2, "VaccSurf_ctor:  Please check the variable MAX_SPHERE_PTS to reset
      .\n");
00738   VASSERT(0);
00739  }
00740  thee = (VaccSurf*)calloc(1,sizeof(Vacc));
00741     VASSERT( VaccSurf_ctor2(thee, mem, probe_radius, nsphere) );
00742
00743     return thee;
00744 }
00745
00746 VPUBLIC int VaccSurf_ctor2(VaccSurf *thee, Vmem *mem, double probe_radius,
00747         int nsphere) {
00748
00749     if (thee == VNULL) return 0;
00750
00751     thee->mem = mem;
00752     thee->npts = nsphere;
00753     thee->probe_radius = probe_radius;
00754     thee->area = 0.0;
00755
00756     if (thee->npts > 0) {
00757   /*
```

```
00758         thee->xpts = Vmem_malloc(thee->mem, thee->npts, sizeof(double));
00759         thee->ypts = Vmem_malloc(thee->mem, thee->npts, sizeof(double));
00760         thee->zpts = Vmem_malloc(thee->mem, thee->npts, sizeof(double));
00761         thee->bpts = Vmem_malloc(thee->mem, thee->npts, sizeof(char));
00762     */
00763   thee->xpts = (double*)calloc(thee->npts,sizeof(double));
00764         thee->ypts = (double*)calloc(thee->npts,sizeof(double));
00765         thee->zpts = (double*)calloc(thee->npts,sizeof(double));
00766         thee->bpts = (char*)calloc(thee->npts,sizeof(char));
00767     } else {
00768         thee->xpts = VNULL;
00769         thee->ypts = VNULL;
00770         thee->zpts = VNULL;
00771         thee->bpts = VNULL;
00772     }
00773
00774     return 1;
00775 }
00776
00777 VPUBLIC void VaccSurf_dtor(VaccSurf **thee) {
00778
00779     Vmem *mem;
00780
00781     if ((*thee) != VNULL) {
00782         mem = (*thee)->mem;
00783         VaccSurf_dtor2(*thee);
00784         //Vmem_free(mem, 1, sizeof(VaccSurf), (void **)thee);
00785   free(*thee);
00786         (*thee) = VNULL;
00787     }
00788
00789 }
00790
00791 VPUBLIC void VaccSurf_dtor2(VaccSurf *thee) {
00792
00793     if (thee->npts > 0) {
00794   /*
00795         Vmem_free(thee->mem, thee->npts, sizeof(double),
00796                 (void **)&(thee->xpts));
00797         Vmem_free(thee->mem, thee->npts, sizeof(double),
00798                 (void **)&(thee->ypts));
00799         Vmem_free(thee->mem, thee->npts, sizeof(double),
00800                 (void **)&(thee->zpts));
00801         Vmem_free(thee->mem, thee->npts, sizeof(char),
00802                 (void **)&(thee->bpts));
00803    */
00804   free(thee->xpts);
00805   free(thee->ypts);
00806   free(thee->zpts);
00807   free(thee->bpts);
00808     }
00809 }
00810
00811 VPUBLIC VaccSurf* Vacc_atomSurf(Vacc *thee, Vatom *atom,
00812         VaccSurf *ref, double prad) {
00813
00814     VaccSurf *surf;
```

```
00815     int i, j, npts, atomID;
00816     double arad, rad, pos[3], *apos;
00817  char bpts[MAX_SPHERE_PTS];
00818
00819     /* Get atom information */
00820     arad = Vatom_getRadius(atom);
00821     apos = Vatom_getPosition(atom);
00822     atomID = Vatom_getAtomID(atom);
00823
00824     if (arad < VSMALL) {
00825         return VaccSurf_ctor(thee->mem, prad, 0);
00826     }
00827
00828     rad = arad + prad;
00829
00830     /* Determine which points will contribute */
00831     npts = 0;
00832     for (i=0; i<ref->npts; i++) {
00833         /* Reset point flag: zero-radius atoms do not contribute */
00834         pos[0] = rad*(ref->xpts[i]) + apos[0];
00835         pos[1] = rad*(ref->ypts[i]) + apos[1];
00836         pos[2] = rad*(ref->zpts[i]) + apos[2];
00837         if (ivdwAccExclus(thee, pos, prad, atomID)) {
00838             npts++;
00839             bpts[i] = 1;
00840         } else {
00841             bpts[i] = 0;
00842         }
00843     }
00844
00845     /* Allocate space for the points */
00846     surf = VaccSurf_ctor(thee->mem, prad, npts);
00847
00848     /* Assign the points */
00849     j = 0;
00850     for (i=0; i<ref->npts; i++) {
00851         if (bpts[i]) {
00852             surf->bpts[j] = 1;
00853             surf->xpts[j] = rad*(ref->xpts[i]) + apos[0];
00854             surf->ypts[j] = rad*(ref->ypts[i]) + apos[1];
00855             surf->zpts[j] = rad*(ref->zpts[i]) + apos[2];
00856             j++;
00857         }
00858     }
00859
00860     /* Assign the area */
00861     surf->area = 4.0*VPI*rad*rad*((double)(surf->npts))/((double)(ref->npts));
00862
00863     return surf;
00864
00865 }
00866
00867 VPUBLIC VaccSurf* VaccSurf_refSphere(Vmem *mem, int npts) {
00868
00869     VaccSurf *surf;
00870     int nactual, i, itheta, ntheta, iphi, nphimax, nphi;
00871     double frac;
```

```
00872     double sintheta, costheta, theta, dtheta;
00873     double sinphi, cosphi, phi, dphi;
00874
00875     /* Setup "constants" */
00876     frac = ((double)(npts))/4.0;
00877     ntheta = VRINT(VSQRT(Vunit_pi*frac));
00878     dtheta = Vunit_pi/((double)(ntheta));
00879     nphimax = 2*ntheta;
00880
00881     /* Count the actual number of points to be used */
00882     nactual = 0;
00883     for (itheta=0; itheta<ntheta; itheta++) {
00884         theta = dtheta*((double)(itheta));
00885         sintheta = VSIN(theta);
00886         costheta = VCOS(theta);
00887         nphi = VRINT(sintheta*nphimax);
00888         nactual += nphi;
00889     }
00890
00891     /* Allocate space for the points */
00892     surf = VaccSurf_ctor(mem, 1.0, nactual);
00893
00894     /* Clear out the boolean array */
00895     for (i=0; i<nactual; i++) surf->bpts[i] = 1;
00896
00897     /* Assign the points */
00898     nactual = 0;
00899     for (itheta=0; itheta<ntheta; itheta++) {
00900         theta = dtheta*((double)(itheta));
00901         sintheta = VSIN(theta);
00902         costheta = VCOS(theta);
00903         nphi = VRINT(sintheta*nphimax);
00904         if (nphi != 0) {
00905             dphi = 2*Vunit_pi/((double)(nphi));
00906             for (iphi=0; iphi<nphi; iphi++) {
00907                 phi = dphi*((double)(iphi));
00908                 sinphi = VSIN(phi);
00909                 cosphi = VCOS(phi);
00910                 surf->xpts[nactual] = cosphi * sintheta;
00911                 surf->ypts[nactual] = sinphi * sintheta;
00912                 surf->zpts[nactual] = costheta;
00913                 nactual++;
00914             }
00915         }
00916     }
00917
00918     surf->npts = nactual;
00919
00920     return surf;
00921 }
00922
00923 VPUBLIC VaccSurf* Vacc_atomSASPoints(Vacc *thee, double radius,
00924         Vatom *atom) {
00925
00926     VaccSurf *asurf = VNULL;
00927     int id;
00928
```

```
00929     if (thee->surf == VNULL) Vacc_SASA(thee, radius);
00930     id = Vatom_getAtomID(atom);
00931
00932     asurf = thee->surf[id];
00933
00934     /* See if this surface needs to be rebuilt */
00935     if (asurf->probe_radius != radius) {
00936         Vnm_print(2, "Vacc_SASA:  Warning -- probe radius changed from %g to %g!\
      n",
00937                 asurf->probe_radius, radius);
00938         VaccSurf_dtor2(asurf);
00939         thee->surf[id] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
00940         asurf = thee->surf[id];
00941     }
00942
00943     return asurf;
00944
00945 }
00946
00947 VPUBLIC void Vacc_splineAccGradAtomNorm4(Vacc *thee, double center[VAPBS_DIM],
00948           double win, double infrad, Vatom *atom, double *grad) {
00949
00950     int i;
00951     double dist, *apos, arad, sm, sm2, sm3, sm4, sm5, sm6, sm7;
00952     double e, e2, e3, e4, e5, e6, e7;
00953     double b, b2, b3, b4, b5, b6, b7;
00954     double c0, c1, c2, c3, c4, c5, c6, c7;
00955     double denom, mygrad;
00956     double mychi = 1.0;           /* Char. func. value for given atom */
00957
00958     VASSERT(thee != NULL);
00959
00960     /* The grad is zero by default */
00961     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
00962
00963     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
00964      * *** MAGNITUDE OF THE FORCE *** */
00965     apos = Vatom_getPosition(atom);
00966     /* Zero-radius atoms don't contribute */
00967     if (Vatom_getRadius(atom) > 0.0) {
00968
00969         arad = Vatom_getRadius(atom);
00970         arad = arad + infrad;
00971         b = arad - win;
00972         e = arad + win;
00973
00974         e2 = e * e;
00975         e3 = e2 * e;
00976         e4 = e3 * e;
00977         e5 = e4 * e;
00978         e6 = e5 * e;
00979         e7 = e6 * e;
00980         b2 = b * b;
00981         b3 = b2 * b;
00982         b4 = b3 * b;
00983         b5 = b4 * b;
00984         b6 = b5 * b;
```

```
00985         b7 = b6 * b;
00986
00987         denom = e7  - 7.0*b*e6 + 21.0*b2*e5 - 35.0*e4*b3
00988     + 35.0*e3*b4 - 21.0*b5*e2  + 7.0*e*b6 - b7;
00989         c0 = b4*(35.0*e3 - 21.0*b*e2 + 7*e*b2 - b3)/denom;
00990         c1 = -140.0*b3*e3/denom;
00991         c2 = 210.0*e2*b2*(e + b)/denom;
00992         c3 = -140.0*e*b*(e2 + 3.0*b*e + b2)/denom;
00993         c4 =  35.0*(e3 + 9.0*b*e2 + + 9.0*e*b2 + b3)/denom;
00994         c5 = -84.0*(e2 + 3.0*b*e + b2)/denom;
00995         c6 =  70.0*(e + b)/denom;
00996         c7 = -20.0/denom;
00997
00998         dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
00999       + VSQR(apos[2]-center[2]));
01000
01001         /* If we're inside an atom, the entire characteristic function
01002     * will be zero and the grad will be zero, so we can stop */
01003         if (dist < (arad - win)) return;
01004         /* Likewise, if we're outside the smoothing window, the characteristic
01005     * function is unity and the grad will be zero, so we can stop */
01006         else if (dist > (arad + win)) return;
01007         /* Account for floating point error at the border
01008     * NAB:  COULDN'T THESE TESTS BE COMBINED AS BELOW
01009     * (Vacc_splineAccAtom)? */
01010         else if ((VABS(dist - (arad - win)) < VSMALL) ||
01011                  (VABS(dist - (arad + win)) < VSMALL)) return;
01012         /* If we're inside the smoothing window */
01013         else {
01014             sm = dist;
01015             sm2 = sm * sm;
01016             sm3 = sm2 * sm;
01017             sm4 = sm3 * sm;
01018             sm5 = sm4 * sm;
01019             sm6 = sm5 * sm;
01020             sm7 = sm6 * sm;
01021             mychi = c0 + c1*sm + c2*sm2 + c3*sm3
01022      + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
01023             mygrad = c1 + 2.0*c2*sm  + 3.0*c3*sm2 + 4.0*c4*sm3
01024      + 5.0*c5*sm4 + 6.0*c6*sm5 + 7.0*c7*sm6;
01025             if (mychi <= 0.0) {
01026      /* Avoid numerical round off errors */
01027      return;
01028             } else if (mychi > 1.0) {
01029      /* Avoid numerical round off errors */
01030      mychi = 1.0;
01031             }
01032         }
01033         /* Now assemble the grad vector */
01034         VASSERT(mychi > 0.0);
01035         for (i=0; i<VAPBS_DIM; i++)
01036             grad[i] = -(mygrad/mychi)*((center[i] - apos[i])/dist);
01037      }
01038 }
01039
01040 VPUBLIC void Vacc_splineAccGradAtomNorm3(Vacc *thee, double center[VAPBS_DIM],
01041            double win, double infrad, Vatom *atom, double *grad) {
```

```
01042
01043     int i;
01044     double dist, *apos, arad, sm, sm2, sm3, sm4, sm5;
01045     double e, e2, e3, e4, e5;
01046     double b, b2, b3, b4, b5;
01047     double c0, c1, c2, c3, c4, c5;
01048     double denom, mygrad;
01049     double mychi = 1.0;           /* Char. func. value for given atom */
01050
01051     VASSERT(thee != NULL);
01052
01053     /* The grad is zero by default */
01054     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
01055
01056     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
01057    * *** MAGNITUDE OF THE FORCE *** */
01058     apos = Vatom_getPosition(atom);
01059     /* Zero-radius atoms don't contribute */
01060     if (Vatom_getRadius(atom) > 0.0) {
01061
01062         arad = Vatom_getRadius(atom);
01063         arad = arad + infrad;
01064         b = arad - win;
01065         e = arad + win;
01066
01067         e2 = e * e;
01068         e3 = e2 * e;
01069         e4 = e3 * e;
01070         e5 = e4 * e;
01071         b2 = b * b;
01072         b3 = b2 * b;
01073         b4 = b3 * b;
01074         b5 = b4 * b;
01075
01076         denom = pow((e - b), 5.0);
01077         c0 = -10.0*e2*b3 + 5.0*e*b4 - b5;
01078         c1 = 30.0*e2*b2;
01079         c2 = -30.0*(e2*b + e*b2);
01080         c3 = 10.0*(e2 + 4.0*e*b + b2);
01081         c4 = -15.0*(e + b);
01082         c5 = 6;
01083         c0 = c0/denom;
01084         c1 = c1/denom;
01085         c2 = c2/denom;
01086         c3 = c3/denom;
01087         c4 = c4/denom;
01088         c5 = c5/denom;
01089
01090         dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
01091       + VSQR(apos[2]-center[2]));
01092
01093         /* If we're inside an atom, the entire characteristic function
01094    * will be zero and the grad will be zero, so we can stop */
01095         if (dist < (arad - win)) return;
01096         /* Likewise, if we're outside the smoothing window, the characteristic
01097    * function is unity and the grad will be zero, so we can stop */
01098         else if (dist > (arad + win)) return;
```

```
01099         /* Account for floating point error at the border
01100     * NAB:  COULDN'T THESE TESTS BE COMBINED AS BELOW
01101     * (Vacc_splineAccAtom)? */
01102         else if ((VABS(dist - (arad - win)) < VSMALL) ||
01103                  (VABS(dist - (arad + win)) < VSMALL)) return;
01104         /* If we're inside the smoothing window */
01105         else {
01106             sm = dist;
01107             sm2 = sm * sm;
01108             sm3 = sm2 * sm;
01109             sm4 = sm3 * sm;
01110             sm5 = sm4 * sm;
01111             mychi = c0 + c1*sm + c2*sm2 + c3*sm3
01112      + c4*sm4 + c5*sm5;
01113             mygrad = c1 + 2.0*c2*sm  + 3.0*c3*sm2 + 4.0*c4*sm3
01114      + 5.0*c5*sm4;
01115             if (mychi <= 0.0) {
01116      /* Avoid numerical round off errors */
01117      return;
01118             } else if (mychi > 1.0) {
01119      /* Avoid numerical round off errors */
01120      mychi = 1.0;
01121             }
01122         }
01123         /* Now assemble the grad vector */
01124         VASSERT(mychi > 0.0);
01125         for (i=0; i<VAPBS_DIM; i++)
01126             grad[i] = -(mygrad/mychi)*((center[i] - apos[i])/dist);
01127     }
01128 }
01129
01130 /* ///////////////////////////////////////////////////////////////////////////
01131     // Routine:  Vacc_atomdSAV
01132     //
01133     // Purpose:  Calculates the vector valued atomic derivative of volume
01134     //
01135     // Args:     radius  The radius of the solvent probe in Angstroms
01136     //           iatom   Index of the atom in thee->alist
01137     //
01138     // Author:   Jason Wagoner
01139     //           Nathan Baker (original FORTRAN routine from UHBD by Brock Luty)
01141 VPUBLIC void Vacc_atomdSAV(Vacc *thee, double srad, Vatom *atom, double *dSA) {
01142
01143     int ipt, iatom;
01144
01145     double area;
01146     double *tPos, tRad, vec[3];
01147     double dx,dy,dz;
01148     VaccSurf *ref;
01149     dx = 0.0;
01150     dy = 0.0;
01151     dz = 0.0;
01152     /* Get the atom information */
01153     ref = thee->refSphere;
01154     iatom = Vatom_getAtomID(atom);
01155
01156     dSA[0] = 0.0;
```

```
01157     dSA[1] = 0.0;
01158     dSA[2] = 0.0;
01159
01160     tPos = Vatom_getPosition(atom);
01161     tRad = Vatom_getRadius(atom);
01162
01163  if(tRad == 0.0) return;
01164
01165     area = 4.0*VPI*(tRad+srad)*(tRad+srad)/((double)(ref->npts));
01166     for (ipt=0; ipt<ref->npts; ipt++) {
01167         vec[0] = (tRad+srad)*ref->xpts[ipt] + tPos[0];
01168         vec[1] = (tRad+srad)*ref->ypts[ipt] + tPos[1];
01169         vec[2] = (tRad+srad)*ref->zpts[ipt] + tPos[2];
01170         if (ivdwAccExclus(thee, vec, srad, iatom)) {
01171    dx = dx+vec[0]-tPos[0];
01172    dy = dy+vec[1]-tPos[1];
01173    dz = dz+vec[2]-tPos[2];
01174      }
01175     }
01176
01177     if ((tRad+srad) != 0){
01178   dSA[0] = dx*area/(tRad+srad);
01179   dSA[1] = dy*area/(tRad+srad);
01180   dSA[2] = dz*area/(tRad+srad);
01181     }
01182
01183 }
01184
01185 /* Note: This is purely test code to make certain that the dSASA code is
01186    behaving properly. This function should NEVER be called by anyone
01187    other than an APBS developer at Wash U.
01188 */
01189 VPRIVATE double Vacc_SASAPos(Vacc *thee, double radius) {
01190
01191     int i, natom;
01192     double area;
01193     Vatom *atom;
01194     VaccSurf *asurf;
01195
01196     natom = Valist_getNumberAtoms(thee->alist);
01197
01198     /* Calculate the area */
01199     area = 0.0;
01200     for (i=0; i<natom; i++) {
01201         atom = Valist_getAtom(thee->alist, i);
01202         asurf = thee->surf[i];
01203
01204   VaccSurf_dtor2(asurf);
01205   thee->surf[i] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
01206   asurf = thee->surf[i];
01207         area += (asurf->area);
01208     }
01209
01210     return area;
01211
01212 }
01213
```

```
01214 VPRIVATE double Vacc_atomSASAPos(Vacc *thee, double radius, Vatom *atom,int mode)
       {
01215
01216     VaccSurf *asurf;
01217     int id;
01218  static int warned = 0;
01219
01220     if ((thee->surf == VNULL) || (mode == 1)){
01221   if(!warned){
01222    printf("WARNING: Recalculating entire surface!!!!\n");
01223    warned = 1;
01224   }
01225   Vacc_SASAPos(thee, radius);
01226  }
01227
01228     id = Vatom_getAtomID(atom);
01229     asurf = thee->surf[id];
01230
01231  VaccSurf_dtor(&asurf);
01232  thee->surf[id] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
01233  asurf = thee->surf[id];
01234
01235     return asurf->area;
01236
01237 }
01238
01239 /* ///////////////////////////////////////////////////////////////////////////
01240    // Routine:  Vacc_atomdSASA
01241    //
01242    // Purpose:  Calculates the derivative of surface area with respect to atomic
01243    //           displacement using finite difference methods.
01244    //
01245    // Args:     radius  The radius of the solvent probe in Angstroms
01246    //           iatom   Index of the atom in thee->alist
01247    //
01248    // Author:   Jason Wagoner
01249    //   David Gohara
01250    //           Nathan Baker (original FORTRAN routine from UHBD by Brock Luty)
01252 VPUBLIC void Vacc_atomdSASA(Vacc *thee, double dpos, double srad, Vatom *atom, do
      uble *dSA) {
01253
01254     int iatom;
01255     double *temp_Pos, tRad;
01256     double tPos[3];
01257  double axb1,axt1,ayb1,ayt1,azb1,azt1;
01258     VaccSurf *ref;
01259
01260     /* Get the atom information */
01261     ref = thee->refSphere;
01262     temp_Pos = Vatom_getPosition(atom);
01263     tRad = Vatom_getRadius(atom);
01264     iatom = Vatom_getAtomID(atom);
01265
01266     dSA[0] = 0.0;
01267     dSA[1] = 0.0;
01268     dSA[2] = 0.0;
01269
```

```
01270  tPos[0] = temp_Pos[0];
01271     tPos[1] = temp_Pos[1];
01272     tPos[2] = temp_Pos[2];
01273
01274  /* Shift by pos -/+ on x */
01275  temp_Pos[0] -= dpos;
01276     axb1 = Vacc_atomSASAPos(thee, srad, atom,0);
01277  temp_Pos[0] = tPos[0];
01278
01279     temp_Pos[0] += dpos;
01280     axt1 = Vacc_atomSASAPos(thee, srad, atom,0);
01281  temp_Pos[0] = tPos[0];
01282
01283  /* Shift by pos -/+ on y */
01284     temp_Pos[1] -= dpos;
01285     ayb1 = Vacc_atomSASAPos(thee, srad, atom,0);
01286  temp_Pos[1] = tPos[1];
01287
01288     temp_Pos[1] += dpos;
01289     ayt1 = Vacc_atomSASAPos(thee, srad, atom,0);
01290  temp_Pos[1] = tPos[1];
01291
01292  /* Shift by pos -/+ on z */
01293     temp_Pos[2] -= dpos;
01294     azb1 = Vacc_atomSASAPos(thee, srad, atom,0);
01295  temp_Pos[2] = tPos[2];
01296
01297     temp_Pos[2] += dpos;
01298     azt1 = Vacc_atomSASAPos(thee, srad, atom,0);
01299  temp_Pos[2] = tPos[2];
01300
01301  /* Reset the atom SASA to zero displacement */
01302  Vacc_atomSASAPos(thee, srad, atom,0);
01303
01304  /* Calculate the final value */
01305     dSA[0] = (axt1-axb1)/(2.0 * dpos);
01306     dSA[1] = (ayt1-ayb1)/(2.0 * dpos);
01307     dSA[2] = (azt1-azb1)/(2.0 * dpos);
01308 }
01309
01310 /* Note: This is purely test code to make certain that the dSASA code is
01311    behaving properly. This function should NEVER be called by anyone
01312    other than an APBS developer at Wash U.
01313 */
01314 VPUBLIC void Vacc_totalAtomdSASA(Vacc *thee, double dpos, double srad, Vatom *ato
      m, double *dSA) {
01315
01316     int iatom;
01317     double *temp_Pos, tRad;
01318     double tPos[3];
01319  double axb1,axt1,ayb1,ayt1,azb1,azt1;
01320     VaccSurf *ref;
01321
01322     /* Get the atom information */
01323     ref = thee->refSphere;
01324     temp_Pos = Vatom_getPosition(atom);
01325     tRad = Vatom_getRadius(atom);
```

```
01326     iatom = Vatom_getAtomID(atom);
01327
01328     dSA[0] = 0.0;
01329     dSA[1] = 0.0;
01330     dSA[2] = 0.0;
01331
01332  tPos[0] = temp_Pos[0];
01333     tPos[1] = temp_Pos[1];
01334     tPos[2] = temp_Pos[2];
01335
01336  /* Shift by pos -/+ on x */
01337  temp_Pos[0] -= dpos;
01338     axb1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01339  temp_Pos[0] = tPos[0];
01340
01341     temp_Pos[0] += dpos;
01342     axt1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01343  temp_Pos[0] = tPos[0];
01344
01345  /* Shift by pos -/+ on y */
01346     temp_Pos[1] -= dpos;
01347     ayb1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01348  temp_Pos[1] = tPos[1];
01349
01350     temp_Pos[1] += dpos;
01351     ayt1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01352  temp_Pos[1] = tPos[1];
01353
01354  /* Shift by pos -/+ on z */
01355     temp_Pos[2] -= dpos;
01356     azb1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01357  temp_Pos[2] = tPos[2];
01358
01359     temp_Pos[2] += dpos;
01360     azt1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01361  temp_Pos[2] = tPos[2];
01362
01363  /* Calculate the final value */
01364     dSA[0] = (axt1-axb1)/(2.0 * dpos);
01365     dSA[1] = (ayt1-ayb1)/(2.0 * dpos);
01366     dSA[2] = (azt1-azb1)/(2.0 * dpos);
01367 }
01368
01369 /* Note: This is purely test code to make certain that the dSASA code is
01370    behaving properly. This function should NEVER be called by anyone
01371    other than an APBS developer at Wash U.
01372 */
01373 VPUBLIC void Vacc_totalAtomdSAV(Vacc *thee, double dpos, double srad, Vatom *atom
      , double *dSA, Vclist *clist) {
01374
01375     int iatom;
01376     double *temp_Pos, tRad;
01377     double tPos[3];
01378  double axb1,axt1,ayb1,ayt1,azb1,azt1;
01379     VaccSurf *ref;
01380
01381     /* Get the atom information */
```

```
01382     ref = thee->refSphere;
01383     temp_Pos = Vatom_getPosition(atom);
01384     tRad = Vatom_getRadius(atom);
01385     iatom = Vatom_getAtomID(atom);
01386 
01387     dSA[0] = 0.0;
01388     dSA[1] = 0.0;
01389     dSA[2] = 0.0;
01390 
01391  tPos[0] = temp_Pos[0];
01392     tPos[1] = temp_Pos[1];
01393     tPos[2] = temp_Pos[2];
01394 
01395  /* Shift by pos -/+ on x */
01396  temp_Pos[0] -= dpos;
01397     axb1 = Vacc_totalSAV(thee,clist, VNULL, srad);
01398  temp_Pos[0] = tPos[0];
01399 
01400     temp_Pos[0] += dpos;
01401     axt1 = Vacc_totalSAV(thee,clist, VNULL, srad);
01402  temp_Pos[0] = tPos[0];
01403 
01404  /* Shift by pos -/+ on y */
01405     temp_Pos[1] -= dpos;
01406     ayb1 = Vacc_totalSAV(thee,clist, VNULL, srad);
01407  temp_Pos[1] = tPos[1];
01408 
01409     temp_Pos[1] += dpos;
01410     ayt1 = Vacc_totalSAV(thee,clist, VNULL, srad);
01411  temp_Pos[1] = tPos[1];
01412 
01413  /* Shift by pos -/+ on z */
01414     temp_Pos[2] -= dpos;
01415     azb1 = Vacc_totalSAV(thee,clist, VNULL, srad);
01416  temp_Pos[2] = tPos[2];
01417 
01418     temp_Pos[2] += dpos;
01419     azt1 = Vacc_totalSAV(thee,clist, VNULL, srad);
01420  temp_Pos[2] = tPos[2];
01421 
01422  /* Calculate the final value */
01423     dSA[0] = (axt1-axb1)/(2.0 * dpos);
01424     dSA[1] = (ayt1-ayb1)/(2.0 * dpos);
01425     dSA[2] = (azt1-azb1)/(2.0 * dpos);
01426 }
01427 
01428 VPUBLIC double Vacc_totalSAV(Vacc *thee, Vclist *clist, APOLparm *apolparm, doubl
      e radius) {
01429 
01430  int i;
01431  int npts[3];
01432 
01433  double spacs[3], vec[3];
01434     double w, wx, wy, wz, len, fn, x, y, z, vol;
01435  double vol_density,sav;
01436     double *lower_corner, *upper_corner;
01437 
```

```
01438  sav = 0.0;
01439  vol = 1.0;
01440  vol_density = 2.0;
01441
01442  lower_corner = clist->lower_corner;
01443  upper_corner = clist->upper_corner;
01444
01445  for (i=0; i<3; i++) {
01446   len = upper_corner[i] - lower_corner[i];
01447   vol *= len;
01448   fn = len*vol_density + 1;
01449   npts[i] = (int)ceil(fn);
01450   spacs[i] = len/((double)(npts[i])-1.0);
01451   if (apolparm != VNULL) {
01452    if (apolparm->setgrid) {
01453     if (apolparm->grid[i] > spacs[i]) {
01454      Vnm_print(2, "Vacc_totalSAV:  Warning, your GRID value (%g) is larger than t
      he recommended value (%g)!\n",
01455       apolparm->grid[i], spacs[i]);
01456     }
01457     spacs[i] = apolparm->grid[i];
01458
01459    }
01460   }
01461  }
01462
01463  for (x=lower_corner[0]; x<=upper_corner[0]; x=x+spacs[0]) {
01464   if ( VABS(x - lower_corner[0]) < VSMALL) {
01465    wx = 0.5;
01466   } else if ( VABS(x - upper_corner[0]) < VSMALL) {
01467    wx = 0.5;
01468   } else {
01469    wx = 1.0;
01470   }
01471   vec[0] = x;
01472   for (y=lower_corner[1]; y<=upper_corner[1]; y=y+spacs[1]) {
01473    if ( VABS(y - lower_corner[1]) < VSMALL) {
01474     wy = 0.5;
01475    } else if ( VABS(y - upper_corner[1]) < VSMALL) {
01476     wy = 0.5;
01477    } else {
01478     wy = 1.0;
01479    }
01480    vec[1] = y;
01481    for (z=lower_corner[2]; z<=upper_corner[2]; z=z+spacs[2]) {
01482     if ( VABS(z - lower_corner[2]) < VSMALL) {
01483      wz = 0.5;
01484     } else if ( VABS(z - upper_corner[2]) < VSMALL) {
01485      wz = 0.5;
01486     } else {
01487      wz = 1.0;
01488     }
01489     vec[2] = z;
01490
01491     w = wx*wy*wz;
01492
01493     sav += (w*(1.0-Vacc_ivdwAcc(thee, vec, radius)));
```

```
01494
01495    } /* z loop */
01496   } /* y loop */
01497  } /* x loop */
01498
01499  w  = spacs[0]*spacs[1]*spacs[2];
01500  sav *= w;
01501
01502  return sav;
01503 }
01504
01505 int Vacc_wcaEnergyAtom(Vacc *thee, APOLparm *apolparm, Valist *alist,
01506          Vclist *clist, int iatom, double *value) {
01507
01508  int i;
01509  int npts[3];
01510  int pad = 14;
01511
01512         int xmin, ymin, zmin;
01513         int xmax, ymax, zmax;
01514
01515         double sigma6, sigma12;
01516
01517  double spacs[3], vec[3];
01518     double w, wx, wy, wz, len, fn, x, y, z, vol;
01519  double x2,y2,z2,r;
01520  double vol_density, energy, rho, srad;
01521  double psig, epsilon, watepsilon, sigma, watsigma, eni, chi;
01522
01523  double *pos;
01524     double *lower_corner, *upper_corner;
01525
01526  Vatom *atom = VNULL;
01527  VASSERT(apolparm != VNULL);
01528
01529  energy = 0.0;
01530  vol = 1.0;
01531  vol_density = 2.0;
01532
01533  lower_corner = clist->lower_corner;
01534  upper_corner = clist->upper_corner;
01535
01536  atom = Valist_getAtom(alist, iatom);
01537  pos = Vatom_getPosition(atom);
01538
01539  /* Note:  these are the original temporary water parameters... they have been
01540   replaced by entries in a parameter file:
01541  watsigma = 1.7683;
01542  watepsilon =  0.1521;
01543  watepsilon = watepsilon*4.184;
01544  */
01545
01546  srad = apolparm->srad;
01547  rho = apolparm->bconc;
01548  watsigma = apolparm->watsigma;
01549  watepsilon = apolparm->watepsilon;
01550  psig = atom->radius;
```

```
01551  epsilon = atom->epsilon;
01552  sigma = psig + watsigma;
01553     epsilon = VSQRT((epsilon * watepsilon));
01554
01555  /* parameters */
01556     sigma6 = VPOW(sigma,6);
01557     sigma12 = VPOW(sigma,12);
01558     /* OPLS-style radius:  double sigmar = sigma*VPOW(2, (1.0/6.0)); */
01559
01560  xmin = pos[0] - pad;
01561  xmax = pos[0] + pad;
01562  ymin = pos[1] - pad;
01563  ymax = pos[1] + pad;
01564  zmin = pos[2] - pad;
01565  zmax = pos[2] + pad;
01566
01567  for (i=0; i<3; i++) {
01568   len = (upper_corner[i] + pad) - (lower_corner[i] - pad);
01569   vol *= len;
01570   fn = len*vol_density + 1;
01571   npts[i] = (int)ceil(fn);
01572   spacs[i] = 0.5;
01573   if (apolparm->setgrid) {
01574    if (apolparm->grid[i] > spacs[i]) {
01575     Vnm_print(2, "Vacc_totalSAV:  Warning, your GRID value (%g) is larger than th
      e recommended value (%g)!\n",
01576       apolparm->grid[i], spacs[i]);
01577    }
01578    spacs[i] = apolparm->grid[i];
01579   }
01580  }
01581
01582  for (x=xmin; x<=xmax; x=x+spacs[0]) {
01583   if ( VABS(x - xmin) < VSMALL) {
01584    wx = 0.5;
01585   } else if ( VABS(x - xmax) < VSMALL) {
01586    wx = 0.5;
01587   } else {
01588    wx = 1.0;
01589   }
01590   vec[0] = x;
01591   for (y=ymin; y<=ymax; y=y+spacs[1]) {
01592    if ( VABS(y - ymin) < VSMALL) {
01593     wy = 0.5;
01594    } else if ( VABS(y - ymax) < VSMALL) {
01595     wy = 0.5;
01596    } else {
01597     wy = 1.0;
01598    }
01599    vec[1] = y;
01600    for (z=zmin; z<=zmax; z=z+spacs[2]) {
01601     if ( VABS(z - zmin) < VSMALL) {
01602      wz = 0.5;
01603     } else if ( VABS(z - zmax) < VSMALL) {
01604      wz = 0.5;
01605     } else {
01606      wz = 1.0;
```

```
01607     }
01608     vec[2] = z;
01609
01610     w = wx*wy*wz;
01611
01612     chi = Vacc_ivdwAcc(thee, vec, srad);
01613
01614     if (VABS(chi) > VSMALL) {
01615
01616      x2 = VSQR(vec[0]-pos[0]);
01617      y2 = VSQR(vec[1]-pos[1]);
01618      z2 = VSQR(vec[2]-pos[2]);
01619      r = VSQRT(x2+y2+z2);
01620
01621      if (r <= 14 && r >= sigma) {
01622       eni = chi*rho*epsilon*(-2.0*sigma6/VPOW(r,6)+sigma12/VPOW(r,12));
01623      }else if (r <= 14){
01624       eni = -1.0*epsilon*chi*rho;
01625      }else{
01626       eni = 0.0;
01627      }
01628     }else{
01629      eni = 0.0;
01630     }
01631
01632     energy += eni*w;
01633
01634    } /* z loop */
01635   } /* y loop */
01636  } /* x loop */
01637
01638  w  = spacs[0]*spacs[1]*spacs[2];
01639  energy *= w;
01640
01641  *value = energy;
01642
01643  return VRC_SUCCESS;
01644 }
01645
01646 VPUBLIC int Vacc_wcaEnergy(Vacc *acc, APOLparm *apolparm, Valist *alist,
01647         Vclist *clist){
01648
01649  int iatom;
01650  int rc = 0;
01651
01652     double energy = 0.0;
01653  double tenergy = 0.0;
01654  double rho = apolparm->bconc;
01655
01656  /* Do a sanity check to make sure that watepsilon and watsigma are set
01657   * If not, return with an error. */
01658  if(apolparm->setwat == 0){
01659   Vnm_print(2,"Vacc_wcaEnergy: Error. No value was set for watsigma and watepsilo
      n.\n");
01660   return VRC_FAILURE;
01661  }
01662
```

```
01663  if (VABS(rho) < VSMALL) {
01664   apolparm->wcaEnergy = tenergy;
01665   return 1;
01666  }
01667
01668     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++){
01669         rc = Vacc_wcaEnergyAtom(acc, apolparm, alist, clist, iatom, &energy);
01670   if(rc == 0) return 0;
01671
01672   tenergy += energy;
01673     }
01674
01675  apolparm->wcaEnergy = tenergy;
01676
01677     return VRC_SUCCESS;
01678
01679 }
01680
01681 VPUBLIC int Vacc_wcaForceAtom(Vacc *thee, APOLparm *apolparm, Vclist *clist,
01682          Vatom *atom, double *force){
01683  int i,si;
01684  int npts[3];
01685  int pad = 14;
01686
01687         int xmin, ymin, zmin;
01688         int xmax, ymax, zmax;
01689
01690         double sigma6, sigma12;
01691
01692  double spacs[3], vec[3], fpt[3];
01693     double w, wx, wy, wz, len, fn, x, y, z, vol;
01694  double x2,y2,z2,r;
01695  double vol_density, fo;
01696  double rho;
01697  double srad, psig, epsilon, watepsilon, sigma, watsigma, chi;
01698
01699  double *pos;
01700     double *lower_corner, *upper_corner;
01701
01702  VASSERT(apolparm != VNULL);
01703
01704  /* Do a sanity check to make sure that watepsilon and watsigma are set
01705   * If not, return with an error. */
01706  if(apolparm->setwat == 0){
01707   Vnm_print(2,"Vacc_wcaEnergy: Error. No value was set for watsigma and watepsilo
      n.\n");
01708   return VRC_FAILURE;
01709  }
01710
01711  vol = 1.0;
01712  vol_density = 2.0;
01713
01714  lower_corner = clist->lower_corner;
01715  upper_corner = clist->upper_corner;
01716
01717  pos = Vatom_getPosition(atom);
01718
```

```
01719  /* Note:  these are the temporary water parameters we used to use in this
01720   routine... they have been replaced by entries in a parameter file
01721  watsigma = 1.7683;
01722  watepsilon =  0.1521;
01723  watepsilon = watepsilon*4.184;
01724  */
01725  srad = apolparm->srad;
01726  rho = apolparm->bconc;
01727  watsigma = apolparm->watsigma;
01728  watepsilon = apolparm->watepsilon;
01729
01730  psig = atom->radius;
01731  epsilon = atom->epsilon;
01732  sigma = psig + watsigma;
01733     epsilon = VSQRT((epsilon * watepsilon));
01734
01735  /* parameters */
01736     sigma6 = VPOW(sigma,6);
01737     sigma12 = VPOW(sigma,12);
01738     /* OPLS-style radius:  double sigmar = sigma*VPOW(2, (1.0/6.0));  */
01739
01740  for (i=0; i<3; i++) {
01741   len = (upper_corner[i] + pad) - (lower_corner[i] - pad);
01742   vol *= len;
01743   fn = len*vol_density + 1;
01744   npts[i] = (int)ceil(fn);
01745   spacs[i] = 0.5;
01746   force[i] = 0.0;
01747   if (apolparm->setgrid) {
01748    if (apolparm->grid[i] > spacs[i]) {
01749     Vnm_print(2, "Vacc_totalSAV:  Warning, your GRID value (%g) is larger than th
       e recommended value (%g)!\n",
01750       apolparm->grid[i], spacs[i]);
01751    }
01752    spacs[i] = apolparm->grid[i];
01753   }
01754  }
01755
01756  xmin = pos[0] - pad;
01757  xmax = pos[0] + pad;
01758  ymin = pos[1] - pad;
01759  ymax = pos[1] + pad;
01760  zmin = pos[2] - pad;
01761  zmax = pos[2] + pad;
01762
01763  for (x=xmin; x<=xmax; x=x+spacs[0]) {
01764   if ( VABS(x - xmin) < VSMALL) {
01765    wx = 0.5;
01766   } else if ( VABS(x - xmax) < VSMALL) {
01767    wx = 0.5;
01768   } else {
01769    wx = 1.0;
01770   }
01771   vec[0] = x;
01772   for (y=ymin; y<=ymax; y=y+spacs[1]) {
01773    if ( VABS(y - ymin) < VSMALL) {
01774     wy = 0.5;
```

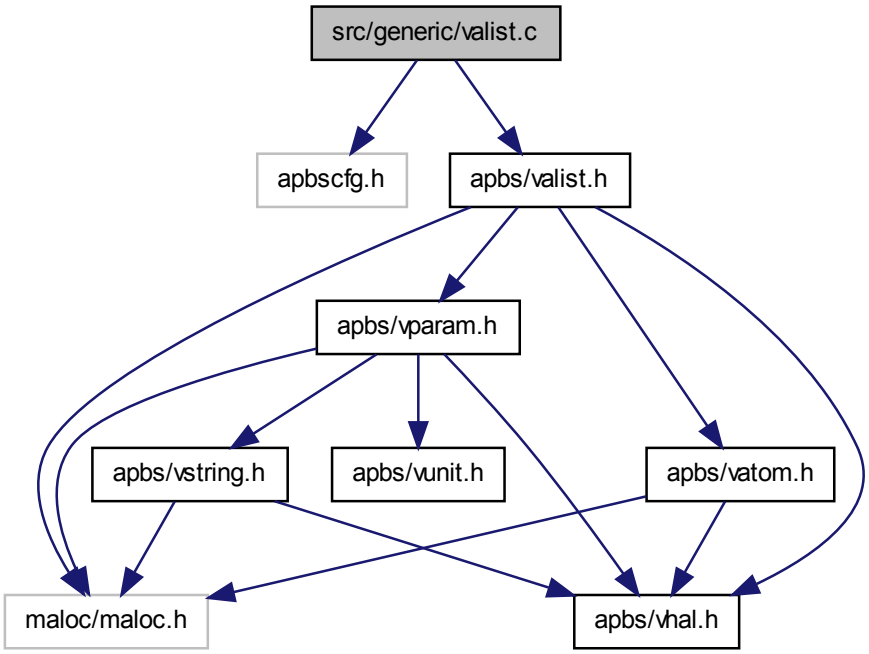
```
01775     } else if ( VABS(y - ymax) < VSMALL) {
01776      wy = 0.5;
01777     } else {
01778      wy = 1.0;
01779     }
01780     vec[1] = y;
01781     for (z=zmin; z<=zmax; z=z+spacs[2]) {
01782      if ( VABS(z - zmin) < VSMALL) {
01783       wz = 0.5;
01784      } else if ( VABS(z - zmax) < VSMALL) {
01785       wz = 0.5;
01786      } else {
01787       wz = 1.0;
01788      }
01789      vec[2] = z;
01790
01791      w = wx*wy*wz;
01792
01793      chi = Vacc_ivdwAcc(thee, vec, srad);
01794
01795      if (chi != 0.0) {
01796
01797       x2 = VSQR(vec[0]-pos[0]);
01798       y2 = VSQR(vec[1]-pos[1]);
01799       z2 = VSQR(vec[2]-pos[2]);
01800       r = VSQRT(x2+y2+z2);
01801
01802       if (r <= 14 && r >= sigma){
01803
01804        fo = 12.0*chi*rho*epsilon*(sigma6/VPOW(r,7)-sigma12/VPOW(r,13));
01805
01806        fpt[0] = -1.0*(pos[0]-vec[0])*fo/r;
01807        fpt[1] = -1.0*(pos[1]-vec[1])*fo/r;
01808        fpt[2] = -1.0*(pos[2]-vec[2])*fo/r;
01809
01810       }else {
01811        for (si=0; si < 3; si++) fpt[si] = 0.0;
01812       }
01813      }else {
01814       for (si=0; si < 3; si++) fpt[si] = 0.0;
01815      }
01816
01817      for(i=0;i<3;i++){
01818       force[i] += (w*fpt[i]);
01819      }
01820
01821     } /* z loop */
01822    } /* y loop */
01823   } /* x loop */
01824
01825   w  = spacs[0]*spacs[1]*spacs[2];
01826   for(i=0;i<3;i++) force[i] *= w;
01827
01828   return VRC_SUCCESS;
01829  }
01830
```

10.63 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 1613 2010-10-19 14:58:22Z sobolevnrm

Attention

*

```
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
* 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 Washington University in St. Louis 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 src/generic/valist.c

```
00001
00048 #include "apbscfg.h"
00049 #include "apbs/valist.h"
00050
00051 VEMBED(rcsid="$Id: valist.c 1613 2010-10-19 14:58:22Z sobolevnrm $")
00052
00053 VPRIVATE char *Valist_whiteChars = " \t\r\n";
00054 VPRIVATE char *Valist_commChars  = "#%";
00055 VPRIVATE char *Valist_xmlwhiteChars = " \t\r\n<>";
00056
00057 #if !defined(VINLINE_VATOM)
00058
```

```
00059 VPUBLIC double Valist_getCenterX(Valist *thee) {
00060
00061     if (thee == NULL) {
00062   Vnm_print(2, "Valist_getCenterX:  Found null pointer when getting the center of
       X coordinate!\n");
00063   VASSERT(0);
00064  }
00065     return thee->center[0];
00066
00067 }
00068
00069 VPUBLIC double Valist_getCenterY(Valist *thee) {
00070
00071     if (thee == NULL) {
00072   Vnm_print(2, "Valist_getCenterY:  Found null pointer when getting the center of
       Y coordinate!\n");
00073   VASSERT(0);
00074  }
00075     return thee->center[1];
00076
00077 }
00078 VPUBLIC double Valist_getCenterZ(Valist *thee) {
00079
00080     if (thee == NULL) {
00081   Vnm_print(2, "Valist_getCenterZ:  Found null pointer when getting the center of
       Z coordinate!\n");
00082   VASSERT(0);
00083  }
00084     return thee->center[2];
00085
00086 }
00087
00088 VPUBLIC Vatom* Valist_getAtomList(Valist *thee) {
00089
00090     if (thee == NULL) {
00091   Vnm_print(2, "Valist_getAtomList:  Found null pointer when getting the atom lis
      t!\n");
00092   VASSERT(0);
00093  }
00094     return thee->atoms;
00095
00096 }
00097
00098 VPUBLIC int Valist_getNumberAtoms(Valist *thee) {
00099
00100     if (thee == NULL) {
00101   Vnm_print(2, "Valist_getNumberAtoms:  Found null pointer when getting the numbe
      r of atoms!\n");
00102   VASSERT(0);
00103  }
00104     return thee->number;
00105
00106 }
00107
00108 VPUBLIC Vatom* Valist_getAtom(Valist *thee, int i) {
00109
00110     if (thee == NULL) {
```

```
00111   Vnm_print(2, "Valist_getAtom:  Found null pointer when getting atoms!\n");
00112   VASSERT(0);
00113  }
00114     if (i >= thee->number) {
00115   Vnm_print(2, "Valist_getAtom:  Requested atom number (%d) outside of atom list
      range (%d)!\n", i, thee->number);
00116   VASSERT(0);
00117  }
00118     return &(thee->atoms[i]);
00119
00120 }
00121
00122 VPUBLIC unsigned long int Valist_memChk(Valist *thee) {
00123
00124     if (thee == NULL) return 0;
00125     return Vmem_bytes(thee->vmem);
00126
00127 }
00128
00129 #endif /* if !defined(VINLINE_VATOM) */
00130
00131 VPUBLIC Valist* Valist_ctor() {
00132
00133     /* Set up the structure */
00134     Valist *thee = VNULL;
00135     thee = Vmem_malloc(VNULL, 1, sizeof(Valist));
00136     if ( thee == VNULL) {
00137   Vnm_print(2, "Valist_ctor:  Got NULL pointer when constructing the atom list ob
      ject!\n");
00138   VASSERT(0);
00139  }
00140     if ( Valist_ctor2(thee) != VRC_SUCCESS) {
00141   Vnm_print(2, "Valist_ctor:   Error in constructing the atom list object!\n");
00142   VASSERT(0);
00143  }
00144
00145     return thee;
00146 }
00147
00148 VPUBLIC Vrc_Codes Valist_ctor2(Valist *thee) {
00149
00150     thee->atoms = VNULL;
00151     thee->number = 0;
00152
00153     /* Initialize the memory management object */
00154     thee->vmem = Vmem_ctor("APBS:VALIST");
00155
00156     return VRC_SUCCESS;
00157
00158 }
00159
00160 VPUBLIC void Valist_dtor(Valist **thee)
00161 {
00162     if ((*thee) != VNULL) {
00163         Valist_dtor2(*thee);
00164         Vmem_free(VNULL, 1, sizeof(Valist), (void **)thee);
00165         (*thee) = VNULL;
```

```
00166     }
00167 }
00168
00169 VPUBLIC void Valist_dtor2(Valist *thee) {
00170
00171     Vmem_free(thee->vmem, thee->number, sizeof(Vatom), (void **)&(thee->atoms));
00172     thee->atoms = VNULL;
00173     thee->number = 0;
00174
00175     Vmem_dtor(&(thee->vmem));
00176 }
00177
00178 /* Read serial number from PDB ATOM/HETATM field */
00179 VPRIVATE Vrc_Codes Valist_readPDBSerial(Valist *thee, Vio *sock, int *serial) {
00180
00181     char tok[VMAX_BUFSIZE];
00182     int ti = 0;
00183
00184     if (Vio_scanf(sock, "%s", tok) != 1) {
00185         Vnm_print(2, "Valist_readPDB:  Ran out of tokens while parsing serial!\n"
      );
00186         return VRC_FAILURE;
00187     }
00188     if (sscanf(tok, "%d", &ti) != 1) {
00189         Vnm_print(2, "Valist_readPDB:  Unable to parse serial token (%s) as int!\
      n",
00190                 tok);
00191         return VRC_FAILURE;
00192     }
00193     *serial = ti;
00194
00195     return VRC_SUCCESS;
00196 }
00197
00198 /* Read atom name from PDB ATOM/HETATM field */
00199 VPRIVATE Vrc_Codes Valist_readPDBAtomName(Valist *thee, Vio *sock,
00200         char atomName[VMAX_ARGLEN]) {
00201
00202     char tok[VMAX_BUFSIZE];
00203
00204     if (Vio_scanf(sock, "%s", tok) != 1) {
00205         Vnm_print(2, "Valist_readPDB:  Ran out of tokens while parsing atom name!
      \n");
00206         return VRC_FAILURE;
00207     }
00208     if (strlen(tok) < VMAX_ARGLEN) strcpy(atomName, tok);
00209     else {
00210         Vnm_print(2, "Valist_readPDB:  Atom name (%s) too long!\n", tok);
00211         return VRC_FAILURE;
00212     }
00213     return VRC_SUCCESS;
00214 }
00215
00216 /* Read residue name from PDB ATOM/HETATM field */
00217 VPRIVATE Vrc_Codes Valist_readPDBResidueName(Valist *thee, Vio *sock,
00218         char resName[VMAX_ARGLEN]) {
00219
```

```
00220     char tok[VMAX_BUFSIZE];
00221
00222     if (Vio_scanf(sock, "%s", tok) != 1) {
00223         Vnm_print(2, "Valist_readPDB:  Ran out of tokens while parsing residue na
      me!\n");
00224         return VRC_FAILURE;
00225     }
00226     if (strlen(tok) < VMAX_ARGLEN) strcpy(resName, tok);
00227     else {
00228         Vnm_print(2, "Valist_readPDB:  Residue name (%s) too long!\n", tok);
00229         return VRC_FAILURE;
00230     }
00231     return VRC_SUCCESS;
00232 }
00233
00234 /* Read residue number from PDB ATOM/HETATM field */
00235 VPRIVATE Vrc_Codes Valist_readPDBResidueNumber(
00236         Valist *thee, Vio *sock, int *resSeq) {
00237
00238     char tok[VMAX_BUFSIZE];
00239  char *resstring;
00240     int ti = 0;
00241
00242     if (Vio_scanf(sock, "%s", tok) != 1) {
00243         Vnm_print(2, "Valist_readPDB:  Ran out of tokens while parsing resSeq!\n"
      );
00244         return VRC_FAILURE;
00245     }
00246     if (sscanf(tok, "%d", &ti) != 1) {
00247
00248      /* One of three things can happen here:
00249    1)  There is a chainID in the line:    THR A   1
00250    2)  The chainID is merged with resSeq: THR A1001
00251    3)  An actual error:                   THR foo
00252
00253   */
00254
00255   if (strlen(tok) == 1) {
00256    /* Case 1: Chain ID Present
00257                        Read the next field and hope its a float */
00258
00259    if (Vio_scanf(sock, "%s", tok) != 1) {
00260           Vnm_print(2, "Valist_readPDB:  Ran out of tokens while parsing resSeq!\
      n");
00261           return VRC_FAILURE;
00262       }
00263       if (sscanf(tok, "%d", &ti) != 1) {
00264     Vnm_print(2, "Valist_readPDB:  Unable to parse resSeq token (%s) as int!\n",
00265                 tok);
00266           return VRC_FAILURE;
00267    }
00268
00269   } else {
00270    /* Case 2: Chain ID, merged string.
00271         Move pointer forward past the chainID and check
00272          */
00273       //strcpy(resstring, tok);
```

```
00274    resstring = tok;
00275    resstring++;
00276
00277    if (sscanf(resstring, "%d", &ti) != 1) {
00278     /* Case 3:  More than one non-numeral char is present. Error.*/
00279                 Vnm_print(2, "Valist_readPDB:  Unable to parse resSeq token (%s)
      as int!\n",
00280                 resstring);
00281              return VRC_FAILURE;
00282    }
00283   }
00284     }
00285     *resSeq = ti;
00286
00287     return VRC_SUCCESS;
00288 }
00289
00290 /* Read atom coordinate from PDB ATOM/HETATM field */
00291 VPRIVATE Vrc_Codes Valist_readPDBAtomCoord(Valist *thee, Vio *sock, double *coord
      ) {
00292
00293     char tok[VMAX_BUFSIZE];
00294     double tf = 0;
00295
00296     if (Vio_scanf(sock, "%s", tok) != 1) {
00297         Vnm_print(2, "Valist_readPDB:  Ran out of tokens while parsing atom coord
      inate!\n");
00298         return VRC_FAILURE;
00299     }
00300     if (sscanf(tok, "%lf", &tf) != 1) {
00301         return VRC_FAILURE;
00302     }
00303     *coord = tf;
00304
00305     return VRC_SUCCESS;
00306 }
00307
00308 /* Read charge and radius from PQR ATOM/HETATM field */
00309 VPRIVATE Vrc_Codes Valist_readPDBChargeRadius(Valist *thee, Vio *sock,
00310         double *charge, double *radius) {
00311
00312     char tok[VMAX_BUFSIZE];
00313     double tf = 0;
00314
00315     if (Vio_scanf(sock, "%s", tok) != 1) {
00316         Vnm_print(2, "Valist_readPQR:  Ran out of tokens while parsing charge!\n"
      );
00317         return VRC_FAILURE;
00318     }
00319     if (sscanf(tok, "%lf", &tf) != 1) {
00320         return VRC_FAILURE;
00321     }
00322     *charge = tf;
00323
00324     if (Vio_scanf(sock, "%s", tok) != 1) {
00325         Vnm_print(2, "Valist_readPQR:  Ran out of tokens while parsing radius!\n"
      );
```

```
00326         return VRC_FAILURE;
00327     }
00328     if (sscanf(tok, "%lf", &tf) != 1) {
00329         return VRC_FAILURE;
00330     }
00331     *radius = tf;
00332 
00333     return VRC_SUCCESS;
00334 }
00335 
00336 /* Read ATOM/HETATM field of PDB through the X/Y/Z fields */
00337 VPRIVATE Vrc_Codes Valist_readPDB_throughXYZ(
00338         Valist *thee,
00339         Vio *sock, /* Socket ready for reading */
00340         int *serial, /* Set to atom number */
00341         char atomName[VMAX_ARGLEN], /* Set to atom name */
00342         char resName[VMAX_ARGLEN], /* Set to residue name */
00343         int *resSeq, /* Set to residue number */
00344         double *x, /* Set to x-coordinate */
00345         double *y, /* Set to y-coordinate */
00346         double *z  /* Set to z-coordinate */
00347         ) {
00348 
00349 
00350     int i, njunk, gotit;
00351 
00352     /* Grab serial */
00353     if (Valist_readPDBSerial(thee, sock, serial) == VRC_FAILURE) {
00354         Vnm_print(2, "Valist_readPDB:  Error while parsing serial!\n");
00355     }
00356 
00357     /* Grab atom name */
00358     if (Valist_readPDBAtomName(thee, sock, atomName) == VRC_FAILURE) {
00359         Vnm_print(2, "Valist_readPDB:  Error while parsing atom name!\n");
00360         return VRC_FAILURE;
00361     }
00362 
00363     /* Grab residue name */
00364     if (Valist_readPDBResidueName(thee, sock, resName) == VRC_FAILURE) {
00365         Vnm_print(2, "Valist_readPDB:  Error while parsing residue name!\n");
00366         return VRC_FAILURE;
00367     }
00368 
00369 
00370     /* Grab residue number */
00371     if (Valist_readPDBResidueNumber(thee, sock, resSeq) == VRC_FAILURE) {
00372         Vnm_print(2, "Valist_readPDB:  Error while parsing residue name!\n");
00373         return VRC_FAILURE;
00374     }
00375 
00376 
00377     /* Read tokens until we find one that can be parsed as an atom
00378      * x-coordinate.  We will allow njunk=1 intervening field that
00379      * cannot be parsed as a coordinate */
00380     njunk = 1;
00381     gotit = 0;
00382     for (i=0; i<(njunk+1); i++) {
```

```
00383         if (Valist_readPDBAtomCoord(thee, sock, x) == VRC_SUCCESS) {
00384             gotit = 1;
00385             break;
00386         }
00387     }
00388     if (!gotit) {
00389         Vnm_print(2, "Valist_readPDB:  Can't find x!\n");
00390         return VRC_FAILURE;
00391     }
00392     /* Read y-coordinate */
00393     if (Valist_readPDBAtomCoord(thee, sock, y) == VRC_FAILURE) {
00394         Vnm_print(2, "Valist_readPDB:  Can't find y!\n");
00395         return VRC_FAILURE;
00396     }
00397     /* Read z-coordinate */
00398     if (Valist_readPDBAtomCoord(thee, sock, z) == VRC_FAILURE) {
00399         Vnm_print(2, "Valist_readPDB:  Can't find z!\n");
00400         return VRC_FAILURE;
00401     }
00402
00403 #if 0 /* Set to 1 if you want to debug */
00404     Vnm_print(1, "Valist_readPDB:  serial = %d\n", *serial);
00405     Vnm_print(1, "Valist_readPDB:  atomName = %s\n", atomName);
00406     Vnm_print(1, "Valist_readPDB:  resName = %s\n", resName);
00407     Vnm_print(1, "Valist_readPDB:  resSeq = %d\n", *resSeq);
00408     Vnm_print(1, "Valist_readPDB:  pos = (%g, %g, %g)\n",
00409               *x, *y, *z);
00410 #endif
00411
00412     return VRC_SUCCESS;
00413 }
00414
00415 /* Get a the next available atom storage location, increasing the storage
00416  * space if necessary.  Return VNULL if something goes wrong. */
00417 VPRIVATE Vatom* Valist_getAtomStorage(
00418         Valist *thee,
00419         Vatom **plist, /* Pointer to existing list of atoms */
00420         int *pnlist, /* Size of existing list, may be changed */
00421         int *pnatoms /* Existing number of atoms in list; incremented
00422                         before exit */
00423         ) {
00424
00425     Vatom *oldList, *newList, *theList;
00426     Vatom *oldAtom, *newAtom;
00427     int iatom, inext, oldLength, newLength, natoms;
00428
00429     newList = VNULL;
00430
00431     /* See if we need more space */
00432     if (*pnatoms >= *pnlist) {
00433
00434         /* Double the storage space */
00435         oldLength = *pnlist;
00436         newLength = 2*oldLength;
00437         newList = Vmem_malloc(thee->vmem, newLength, sizeof(Vatom));
00438         oldList = *plist;
00439
```

```
00440         /* Check the allocation */
00441         if (newList == VNULL) {
00442             Vnm_print(2, "Valist_readPDB:  failed to allocate space for %d (Vatom
      )s!\n", newLength);
00443             return VNULL;
00444         }
00445 
00446         /* Copy the atoms over */
00447         natoms = *pnatoms;
00448         for (iatom=0; iatom<natoms; iatom++) {
00449             oldAtom = &(oldList[iatom]);
00450             newAtom = &(newList[iatom]);
00451             Vatom_copyTo(oldAtom, newAtom);
00452             Vatom_dtor2(oldAtom);
00453         }
00454 
00455         /* Free the old list */
00456         Vmem_free(thee->vmem, oldLength, sizeof(Vatom), (void **)plist);
00457 
00458         /* Copy new list to plist */
00459         *plist = newList;
00460         *pnlist = newLength;
00461     }
00462 
00463     theList = *plist;
00464     inext = *pnatoms;
00465 
00466     /* Get the next available spot and increment counters */
00467     newAtom = &(theList[inext]);
00468     *pnatoms = inext + 1;
00469 
00470     return newAtom;
00471 }
00472 
00473 VPRIVATE Vrc_Codes Valist_setAtomArray(Valist *thee,
00474         Vatom **plist, /* Pointer to list of atoms to store */
00475         int nlist, /* Length of list */
00476         int natoms /* Number of real atom entries in list */
00477         ) {
00478 
00479     Vatom *list, *newAtom, *oldAtom;
00480     int i;
00481 
00482     list = *plist;
00483 
00484     /* Allocate necessary space */
00485     thee->number = 0;
00486     thee->atoms = Vmem_malloc(thee->vmem, natoms, sizeof(Vatom));
00487     if (thee->atoms == VNULL) {
00488         Vnm_print(2, "Valist_readPDB:  Unable to allocate space for %d (Vatom)s!\
      n",
00489                 natoms);
00490         return VRC_FAILURE;
00491     }
00492     thee->number = natoms;
00493 
00494     /* Copy over data */
```

```
00495     for (i=0; i<thee->number; i++) {
00496         newAtom = &(thee->atoms[i]);
00497         oldAtom = &(list[i]);
00498         Vatom_copyTo(oldAtom, newAtom);
00499         Vatom_dtor2(oldAtom);
00500     }
00501 
00502     /* Free old array */
00503     Vmem_free(thee->vmem, nlist, sizeof(Vatom), (void **)plist);
00504 
00505     return VRC_SUCCESS;
00506 }
00507 
00508 VPUBLIC Vrc_Codes Valist_readPDB(Valist *thee, Vparam *param, Vio *sock) {
00509 
00510     /* WE DO NOT DIRECTLY CONFORM TO PDB STANDARDS -- TO ALLOW LARGER FILES, WE
00511      * REQUIRE ALL FIELDS TO BE WHITESPACE DELIMITED */
00512 
00513     Vatom *atoms = VNULL;
00514     Vatom *nextAtom = VNULL;
00515     Vparam_AtomData *atomData = VNULL;
00516 
00517     char tok[VMAX_BUFSIZE];
00518     char atomName[VMAX_ARGLEN], resName[VMAX_ARGLEN];
00519 
00520  int nlist, natoms, serial, resSeq;
00521 
00522  double x, y, z, charge, radius, epsilon;
00523     double pos[3];
00524 
00525     if (thee == VNULL) {
00526  Vnm_print(2, "Valist_readPDB:  Got NULL pointer when reading PDB file!\n");
00527  VASSERT(0);
00528  }
00529     thee->number = 0;
00530 
00531     Vio_setWhiteChars(sock, Valist_whiteChars);
00532     Vio_setCommChars(sock, Valist_commChars);
00533 
00534     /* Allocate some initial space for the atoms */
00535     nlist = 200;
00536     atoms = Vmem_malloc(thee->vmem, nlist, sizeof(Vatom));
00537 
00538     natoms = 0;
00539     /* Read until we run out of lines */
00540     while (Vio_scanf(sock, "%s", tok) == 1) {
00541 
00542         /* Parse only ATOM/HETATOM fields */
00543         if ((Vstring_strcasecmp(tok, "ATOM") == 0) ||
00544             (Vstring_strcasecmp(tok, "HETATM") == 0)) {
00545 
00546             /* Read ATOM/HETATM field of PDB through the X/Y/Z fields */
00547             if (Valist_readPDB_throughXYZ(thee, sock, &serial, atomName,
00548                       resName, &resSeq, &x, &y, &z) == VRC_FAILURE) {
00549                 Vnm_print(2, "Valist_readPDB:  Error parsing atom %d!\n",
00550                           serial);
00551                 return VRC_FAILURE;
```

```
00552             }
00553
00554             /* Try to find the parameters. */
00555             atomData = Vparam_getAtomData(param, resName, atomName);
00556             if (atomData == VNULL) {
00557                 Vnm_print(2, "Valist_readPDB:  Couldn't find parameters for \
00558 atom = %s, residue = %s\n", atomName, resName);
00559                 return VRC_FAILURE;
00560             }
00561             charge = atomData->charge;
00562             radius = atomData->radius;
00563    epsilon = atomData->epsilon;
00564
00565             /* Get pointer to next available atom position */
00566             nextAtom = Valist_getAtomStorage(thee, &atoms, &nlist, &natoms);
00567             if (nextAtom == VNULL) {
00568                 Vnm_print(2, "Valist_readPDB:  Error in allocating spacing for at
      oms!\n");
00569                 return VRC_FAILURE;
00570             }
00571
00572             /* Store the information */
00573             pos[0] = x; pos[1] = y; pos[2] = z;
00574             Vatom_setPosition(nextAtom, pos);
00575             Vatom_setCharge(nextAtom, charge);
00576             Vatom_setRadius(nextAtom, radius);
00577    Vatom_setEpsilon(nextAtom, epsilon);
00578             Vatom_setAtomID(nextAtom, natoms-1);
00579    Vatom_setResName(nextAtom, resName);
00580             Vatom_setAtomName(nextAtom, atomName);
00581
00582         } /* if ATOM or HETATM */
00583     } /* while we haven't run out of tokens */
00584
00585     Vnm_print(0, "Valist_readPDB: Counted %d atoms\n", natoms);
00586     fflush(stdout);
00587
00588     /* Store atoms internally */
00589     if (Valist_setAtomArray(thee, &atoms, nlist, natoms) == VRC_FAILURE) {
00590         Vnm_print(2, "Valist_readPDB:  unable to store atoms!\n");
00591         return VRC_FAILURE;
00592     }
00593
00594     return Valist_getStatistics(thee);
00595
00596
00597 }
00598
00599 VPUBLIC Vrc_Codes Valist_readPQR(Valist *thee, Vparam *params, Vio *sock) {
00600
00601     /* WE DO NOT DIRECTLY CONFORM TO PDB STANDARDS -- TO ALLOW LARGER FILES, WE
00602      * REQUIRE ALL FIELDS TO BE WHITESPACE DELIMITED */
00603
00604
00605     Vatom *atoms = VNULL;
00606     Vatom *nextAtom = VNULL;
00607  Vparam_AtomData *atomData = VNULL;
```

```
00608
00609     char tok[VMAX_BUFSIZE];
00610     char atomName[VMAX_ARGLEN], resName[VMAX_ARGLEN];
00611
00612  int use_params = 0;
00613  int nlist, natoms, serial, resSeq;
00614
00615     double x, y, z, charge, radius, epsilon;
00616     double pos[3];
00617
00618  epsilon = 0.0;
00619
00620     if (thee == VNULL) {
00621   Vnm_print(2, "Valist_readPQR:  Got NULL pointer when reading PQR file!\n");
00622   VASSERT(0);
00623  }
00624     thee->number = 0;
00625
00626     Vio_setWhiteChars(sock, Valist_whiteChars);
00627     Vio_setCommChars(sock, Valist_commChars);
00628
00629     /* Allocate some initial space for the atoms */
00630     nlist = 200;
00631     atoms = Vmem_malloc(thee->vmem, nlist, sizeof(Vatom));
00632
00633  /* Check if we are using a parameter file or not */
00634  if(params != VNULL) use_params = 1;
00635
00636     natoms = 0;
00637     /* Read until we run out of lines */
00638     while (Vio_scanf(sock, "%s", tok) == 1) {
00639
00640         /* Parse only ATOM/HETATOM fields */
00641         if ((Vstring_strcasecmp(tok, "ATOM") == 0) ||
00642             (Vstring_strcasecmp(tok, "HETATM") == 0)) {
00643
00644             /* Read ATOM/HETATM field of PDB through the X/Y/Z fields */
00645             if (Valist_readPDB_throughXYZ(thee, sock, &serial, atomName,
00646                       resName, &resSeq, &x, &y, &z) == VRC_FAILURE) {
00647                 Vnm_print(2, "Valist_readPQR:  Error parsing atom %d!\n",serial);

00648                 Vnm_print(2, "Please double check this atom in the pqr file, e.g.
      , make sure there are no concatenated fields.\n");
00649                 return VRC_FAILURE;
00650             }
00651
00652             /* Read Q/R fields */
00653             if (Valist_readPDBChargeRadius(thee, sock, &charge, &radius) ==
      VRC_FAILURE) {
00654                 Vnm_print(2, "Valist_readPQR:  Error parsing atom %d!\n",
00655                           serial);
00656                 Vnm_print(2, "Please double check this atom in the pqr file, e.g.
      , make sure there are no concatenated fields.\n");
00657                 return VRC_FAILURE;
00658             }
00659
00660    if(use_params){
```

```
00661     /* Try to find the parameters. */
00662     atomData = Vparam_getAtomData(params, resName, atomName);
00663     if (atomData == VNULL) {
00664      Vnm_print(2, "Valist_readPDB:  Couldn't find parameters for \
00665 atom = %s, residue = %s\n", atomName, resName);
00666      return VRC_FAILURE;
00667     }
00668     charge = atomData->charge;
00669     radius = atomData->radius;
00670     epsilon = atomData->epsilon;
00671    }
00672
00673             /* Get pointer to next available atom position */
00674             nextAtom = Valist_getAtomStorage(thee, &atoms, &nlist, &natoms);
00675             if (nextAtom == VNULL) {
00676                 Vnm_print(2, "Valist_readPQR:  Error in allocating spacing for at
      oms!\n");
00677                 return VRC_FAILURE;
00678             }
00679
00680             /* Store the information */
00681             pos[0] = x; pos[1] = y; pos[2] = z;
00682             Vatom_setPosition(nextAtom, pos);
00683             Vatom_setCharge(nextAtom, charge);
00684             Vatom_setRadius(nextAtom, radius);
00685    Vatom_setEpsilon(nextAtom, epsilon);
00686             Vatom_setAtomID(nextAtom, natoms-1);
00687    Vatom_setResName(nextAtom, resName);
00688             Vatom_setAtomName(nextAtom, atomName);
00689
00690         } /* if ATOM or HETATM */
00691     } /* while we haven't run out of tokens */
00692
00693     Vnm_print(0, "Valist_readPQR: Counted %d atoms\n", natoms);
00694     fflush(stdout);
00695
00696     /* Store atoms internally */
00697     if (Valist_setAtomArray(thee, &atoms, nlist, natoms) == VRC_FAILURE) {
00698         Vnm_print(2, "Valist_readPDB:  unable to store atoms!\n");
00699         return VRC_FAILURE;
00700     }
00701
00702     return Valist_getStatistics(thee);
00703
00704
00705 }
00706
00707 VPUBLIC Vrc_Codes Valist_readXML(Valist *thee, Vparam *params, Vio *sock) {
00708
00709     Vatom *atoms = VNULL;
00710     Vatom *nextAtom = VNULL;
00711
00712     char tok[VMAX_BUFSIZE];
00713     char endtag[VMAX_BUFSIZE];
00714
00715  int nlist, natoms;
00716     int xset, yset, zset, chgset, radset;
```

```
00717
00718  double x, y, z, charge, radius, dtmp;
00719     double pos[3];
00720
00721     if (thee == VNULL) {
00722   Vnm_print(2, "Valist_readXML:  Got NULL pointer when reading XML file!\n");
00723   VASSERT(0);
00724  }
00725     thee->number = 0;
00726
00727     Vio_setWhiteChars(sock, Valist_xmlwhiteChars);
00728     Vio_setCommChars(sock, Valist_commChars);
00729
00730     /* Allocate some initial space for the atoms */
00731     nlist = 200;
00732     atoms = Vmem_malloc(thee->vmem, nlist, sizeof(Vatom));
00733
00734     /* Initialize some variables */
00735     natoms = 0;
00736     xset = 0;
00737     yset = 0;
00738     zset = 0;
00739     chgset = 0;
00740     radset = 0;
00741     strcpy(endtag,"/");
00742
00743  if(params == VNULL){
00744   Vnm_print(1,"\nValist_readXML: Warning Warning Warning Warning Warning\n");
00745   Vnm_print(1,"Valist_readXML: The use of XML input files with parameter\n");
00746   Vnm_print(1,"Valist_readXML: files is currently not supported.\n");
00747   Vnm_print(1,"Valist_readXML: Warning Warning Warning Warning Warning\n\n");
00748  }
00749
00750     /* Read until we run out of lines */
00751     while (Vio_scanf(sock, "%s", tok) == 1) {
00752
00753         /* The first tag taken is the start tag - save it to detect end */
00754         if (Vstring_strcasecmp(endtag, "/") == 0) strcat(endtag, tok);
00755
00756         if (Vstring_strcasecmp(tok, "x") == 0) {
00757             Vio_scanf(sock, "%s", tok);
00758             if (sscanf(tok, "%lf", &dtmp) != 1) {
00759                 Vnm_print(2, "Valist_readXML:  Unexpected token (%s) while \
00760 reading x!\n", tok);
00761                 return VRC_FAILURE;
00762               }
00763             x = dtmp;
00764             xset = 1;
00765         } else if (Vstring_strcasecmp(tok, "y") == 0) {
00766             Vio_scanf(sock, "%s", tok);
00767             if (sscanf(tok, "%lf", &dtmp) != 1) {
00768                 Vnm_print(2, "Valist_readXML:  Unexpected token (%s) while \
00769 reading y!\n", tok);
00770                 return VRC_FAILURE;
00771               }
00772             y = dtmp;
00773             yset = 1;
```

```
00774         } else if (Vstring_strcasecmp(tok, "z") == 0) {
00775             Vio_scanf(sock, "%s", tok);
00776             if (sscanf(tok, "%lf", &dtmp) != 1) {
00777                 Vnm_print(2, "Valist_readXML:  Unexpected token (%s) while \
00778 reading z!\n", tok);
00779                 return VRC_FAILURE;
00780               }
00781             z = dtmp;
00782             zset = 1;
00783         } else if (Vstring_strcasecmp(tok, "charge") == 0) {
00784             Vio_scanf(sock, "%s", tok);
00785             if (sscanf(tok, "%lf", &dtmp) != 1) {
00786                 Vnm_print(2, "Valist_readXML:  Unexpected token (%s) while \
00787 reading charge!\n", tok);
00788                 return VRC_FAILURE;
00789               }
00790             charge = dtmp;
00791             chgset = 1;
00792         } else if (Vstring_strcasecmp(tok, "radius") == 0) {
00793             Vio_scanf(sock, "%s", tok);
00794             if (sscanf(tok, "%lf", &dtmp) != 1) {
00795                 Vnm_print(2, "Valist_readXML:  Unexpected token (%s) while \
00796 reading radius!\n", tok);
00797                 return VRC_FAILURE;
00798               }
00799             radius = dtmp;
00800             radset = 1;
00801         } else if (Vstring_strcasecmp(tok, "/atom") == 0) {
00802 
00803           /* Get pointer to next available atom position */
00804             nextAtom = Valist_getAtomStorage(thee, &atoms, &nlist, &natoms);
00805             if (nextAtom == VNULL) {
00806                 Vnm_print(2, "Valist_readXML:  Error in allocating spacing for at
      oms!\n");
00807                 return VRC_FAILURE;
00808             }
00809 
00810             if (xset && yset && zset && chgset && radset){
00811 
00812                 /* Store the information */
00813                 pos[0] = x; pos[1] = y; pos[2] = z;
00814                 Vatom_setPosition(nextAtom, pos);
00815                 Vatom_setCharge(nextAtom, charge);
00816                 Vatom_setRadius(nextAtom, radius);
00817                 Vatom_setAtomID(nextAtom, natoms-1);
00818 
00819                 /* Reset the necessary flags */
00820                 xset = 0;
00821                 yset = 0;
00822                 zset = 0;
00823                 chgset = 0;
00824                 radset = 0;
00825             } else {
00826                 Vnm_print(2,  "Valist_readXML:  Missing field(s) in atom tag:\n")
      ;
00827                 if (!xset) Vnm_print(2,"\tx value not set!\n");
00828                 if (!yset) Vnm_print(2,"\ty value not set!\n");
```

```
00829                 if (!zset) Vnm_print(2,"\tz value not set!\n");
00830                 if (!chgset) Vnm_print(2,"\tcharge value not set!\n");
00831                 if (!radset) Vnm_print(2,"\tradius value not set!\n");
00832                 return VRC_FAILURE;
00833             }
00834         } else if (Vstring_strcasecmp(tok, endtag) == 0) break;
00835     }
00836
00837     Vnm_print(0, "Valist_readXML: Counted %d atoms\n", natoms);
00838     fflush(stdout);
00839
00840     /* Store atoms internally */
00841     if (Valist_setAtomArray(thee, &atoms, nlist, natoms) == VRC_FAILURE) {
00842         Vnm_print(2, "Valist_readXML:  unable to store atoms!\n");
00843         return VRC_FAILURE;
00844     }
00845
00846     return Valist_getStatistics(thee);
00847
00848 }
00849
00850 /* Load up Valist with various statistics */
00851 VPUBLIC Vrc_Codes Valist_getStatistics(Valist *thee) {
00852
00853     Vatom *atom;
00854     int i, j;
00855
00856     if (thee == VNULL) {
00857   Vnm_print(2, "Valist_getStatistics:  Got NULL pointer when loading up Valist wi
      th various statistics!\n");
00858   VASSERT(0);
00859  }
00860
00861     thee->center[0] = 0.;
00862     thee->center[1] = 0.;
00863     thee->center[2] = 0.;
00864     thee->maxrad = 0.;
00865     thee->charge = 0.;
00866
00867     if (thee->number == 0) return VRC_FAILURE;
00868
00869     /* Reset stat variables */
00870     atom = &(thee->atoms[0]);
00871     for (i=0; i<3; i++) {
00872         thee->maxcrd[i] = thee->mincrd[i] = atom->position[i];
00873     }
00874     thee->maxrad = atom->radius;
00875     thee->charge = 0.0;
00876
00877     for (i=0; i<thee->number; i++) {
00878
00879         atom = &(thee->atoms[i]);
00880         for (j=0; j<3; j++) {
00881             if (atom->position[j] < thee->mincrd[j])
00882               thee->mincrd[j] = atom->position[j];
00883             if (atom->position[j] > thee->maxcrd[j])
00884               thee->maxcrd[j] = atom->position[j];
```

```
00885         }
00886         if (atom->radius > thee->maxrad) thee->maxrad = atom->radius;
00887         thee->charge = thee->charge + atom->charge;
00888     }
00889
00890     thee->center[0] = 0.5*(thee->maxcrd[0] + thee->mincrd[0]);
00891     thee->center[1] = 0.5*(thee->maxcrd[1] + thee->mincrd[1]);
00892     thee->center[2] = 0.5*(thee->maxcrd[2] + thee->mincrd[2]);
00893
00894  Vnm_print(0, "Valist_getStatistics:  Max atom coordinate:  (%g, %g, %g)\n",
00895      thee->maxcrd[0], thee->maxcrd[1], thee->maxcrd[2]);
00896  Vnm_print(0, "Valist_getStatistics:  Min atom coordinate:  (%g, %g, %g)\n",
00897      thee->mincrd[0], thee->mincrd[1], thee->mincrd[2]);
00898  Vnm_print(0, "Valist_getStatistics:  Molecule center:  (%g, %g, %g)\n",
00899      thee->center[0], thee->center[1], thee->center[2]);
00900
00901     return VRC_SUCCESS;
00902 }
```

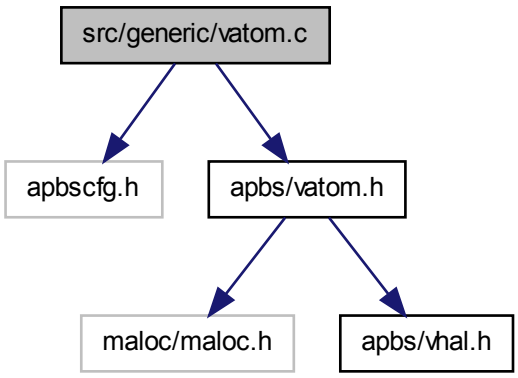
10.65 src/generic/vatom.c File Reference

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 1552 2010-02-10 17:46:27Z yhuang01

Attention

*

Definition in file vatom.c.

10.66 src/generic/vatom.c

```
00001
00049 #include "apbscfg.h"
00050 #include "apbs/vatom.h"
00051
00052 VEMBED(rcsid="$Id: vatom.c 1552 2010-02-10 17:46:27Z yhuang01 $")
00053
00054 #if !defined(VINLINE_VATOM)
00055
00056 VPUBLIC double *Vatom_getPosition(Vatom *thee) {
00057
00058    VASSERT(thee != VNULL);
00059    return thee->position;
```

```
00060
00061 }
00062
00063 VPUBLIC double Vatom_getPartID(Vatom *thee) {
00064
00065    VASSERT(thee != VNULL);
00066    return thee->partID;
00067
00068 }
00069
00070 VPUBLIC void Vatom_setPartID(Vatom *thee, int partID) {
00071
00072    VASSERT(thee != VNULL);
00073    thee->partID = (double)partID;
00074
00075 }
00076
00077 VPUBLIC double Vatom_getAtomID(Vatom *thee) {
00078
00079    VASSERT(thee != VNULL);
00080    return thee->atomID;
00081
00082 }
00083
00084 VPUBLIC void Vatom_setAtomID(Vatom *thee, int atomID) {
00085
00086    VASSERT(thee != VNULL);
00087    thee->atomID = atomID;
00088
00089 }
00090
00091 VPUBLIC void Vatom_setRadius(Vatom *thee, double radius) {
00092
00093    VASSERT(thee != VNULL);
00094    thee->radius = radius;
00095
00096 }
00097
00098 VPUBLIC double Vatom_getRadius(Vatom *thee) {
00099
00100    VASSERT(thee != VNULL);
00101    return thee->radius;
00102
00103 }
00104
00105 VPUBLIC void Vatom_setCharge(Vatom *thee, double charge) {
00106
00107    VASSERT(thee != VNULL);
00108    thee->charge = charge;
00109
00110 }
00111
00112 VPUBLIC double Vatom_getCharge(Vatom *thee) {
00113
00114    VASSERT(thee != VNULL);
00115    return thee->charge;
00116
```

```
00117 }
00118
00119 VPUBLIC unsigned long int Vatom_memChk(Vatom *thee) { return sizeof(Vatom); }
00120
00121 #endif /* if !defined(VINLINE_VATOM) */
00122
00123 VPUBLIC Vatom* Vatom_ctor() {
00124
00125     /* Set up the structure */
00126     Vatom *thee = VNULL;
00127     thee = (Vatom *)Vmem_malloc( VNULL, 1, sizeof(Vatom) );
00128     VASSERT( thee != VNULL);
00129     VASSERT( Vatom_ctor2(thee));
00130
00131     return thee;
00132 }
00133
00134 VPUBLIC int Vatom_ctor2(Vatom *thee) {
00135     thee->partID = -1;
00136     return 1;
00137 }
00138
00139 VPUBLIC void Vatom_dtor(Vatom **thee) {
00140     if ((*thee) != VNULL) {
00141         Vatom_dtor2(*thee);
00142         Vmem_free(VNULL, 1, sizeof(Vatom), (void **)thee);
00143         (*thee) = VNULL;
00144     }
00145 }
00146
00147 VPUBLIC void Vatom_dtor2(Vatom *thee) { ; }
00148
00149 VPUBLIC void Vatom_setPosition(Vatom *thee, double position[3]) {
00150
00151    VASSERT(thee != VNULL);
00152    (thee->position)[0] = position[0];
00153    (thee->position)[1] = position[1];
00154    (thee->position)[2] = position[2];
00155
00156 }
00157
00158 VPUBLIC void Vatom_copyTo(Vatom *thee, Vatom *dest) {
00159
00160     VASSERT(thee != VNULL);
00161     VASSERT(dest != VNULL);
00162
00163     memcpy(dest, thee, sizeof(Vatom));
00164
00165 }
00166
00167 VPUBLIC void Vatom_copyFrom(Vatom *thee, Vatom *src) {
00168
00169     Vatom_copyTo(src, thee);
00170
00171 }
00172
00173 VPUBLIC void Vatom_setResName(Vatom *thee, char resName[VMAX_RECLEN]) {
```

```
00174
00175  VASSERT(thee != VNULL);
00176  strcpy(thee->resName, resName);
00177
00178 }
00179
00180 VPUBLIC void Vatom_getResName(Vatom *thee, char resName[VMAX_RECLEN]) {
00181
00182
00183  VASSERT(thee != VNULL);
00184  strcpy(resName,thee->resName);
00185
00186 }
00187
00188 VPUBLIC void Vatom_setAtomName(Vatom *thee, char atomName[VMAX_RECLEN]) {
00189
00190  VASSERT(thee != VNULL);
00191  strcpy(thee->atomName, atomName);
00192
00193 }
00194
00195 VPUBLIC void Vatom_getAtomName(Vatom *thee, char atomName[VMAX_RECLEN]) {
00196
00197  VASSERT(thee != VNULL);
00198  strcpy(atomName,thee->atomName);
00199
00200 }
00201
00202 #if defined(WITH_TINKER)
00203
00204 VPUBLIC void Vatom_setDipole(Vatom *thee, double dipole[3]) {
00205
00206    VASSERT(thee != VNULL);
00207    (thee->dipole)[0] = dipole[0];
00208    (thee->dipole)[1] = dipole[1];
00209    (thee->dipole)[2] = dipole[2];
00210
00211 }
00212
00213 VPUBLIC void Vatom_setQuadrupole(Vatom *thee, double quadrupole[9]) {
00214
00215    int i;
00216    VASSERT(thee != VNULL);
00217    for (i=0; i<9; i++)  (thee->quadrupole)[i] = quadrupole[i];
00218 }
00219
00220 VPUBLIC void Vatom_setInducedDipole(Vatom *thee, double dipole[3]) {
00221
00222    VASSERT(thee != VNULL);
00223    (thee->inducedDipole)[0] = dipole[0];
00224    (thee->inducedDipole)[1] = dipole[1];
00225    (thee->inducedDipole)[2] = dipole[2];
00226 }
00227
00228 VPUBLIC void Vatom_setNLInducedDipole(Vatom *thee, double dipole[3]) {
00229
00230    VASSERT(thee != VNULL);
```

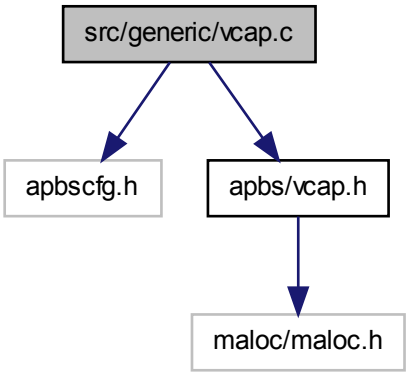
```
00231     (thee->nlInducedDipole)[0] = dipole[0];
00232     (thee->nlInducedDipole)[1] = dipole[1];
00233     (thee->nlInducedDipole)[2] = dipole[2];
00234 
00235 }
00236 
00237 VPUBLIC double *Vatom_getDipole(Vatom *thee) {
00238 
00239     VASSERT(thee != VNULL);
00240     return thee->dipole;
00241 
00242 }
00243 
00244 VPUBLIC double *Vatom_getQuadrupole(Vatom *thee) {
00245 
00246     VASSERT(thee != VNULL);
00247     return thee->quadrupole;
00248 
00249 }
00250 
00251 VPUBLIC double *Vatom_getInducedDipole(Vatom *thee) {
00252 
00253     VASSERT(thee != VNULL);
00254     return thee->inducedDipole;
00255 
00256 }
00257 
00258 VPUBLIC double *Vatom_getNLInducedDipole(Vatom *thee) {
00259 
00260     VASSERT(thee != VNULL);
00261     return thee->nlInducedDipole;
00262 
00263 }
00264 
00265 #endif /* if defined(WITH_TINKER) */
```

10.67 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 1552 2010-02-10 17:46:27Z yhuang01

Attention

```
 *
 * APBS -- Adaptive Poisson-Boltzmann Solver
 *
 * Nathan A. Baker (nathan.baker@pnl.gov)
 * Pacific Northwest National Laboratory
 *
 * Additional contributing authors listed in the code documentation.
 *
 * Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
 * 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 Washington University in St. Louis 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 src/generic/vcap.c

```
00001
00049 #include "apbscfg.h"
00050 #include "apbs/vcap.h"
00051
00052 VPUBLIC double Vcap_exp(double x, int *ichop) {
```

```
00053
00054     /* The two chopped arguments */
00055     if (x > EXPMAX) {
00056        (*ichop) = 1;
00057        return VEXP(EXPMAX);
00058     } else if (x < EXPMIN) {
00059        (*ichop) = 1;
00060        return VEXP(EXPMIN);
00061     }
00062
00063     /* The normal EXP */
00064     (*ichop) = 0;
00065     return VEXP(x);
00066 }
00067
00068 VPUBLIC double Vcap_sinh(double x, int *ichop) {
00069
00070     /* The two chopped arguments */
00071     if (x > EXPMAX) {
00072        (*ichop) = 1;
00073        return VSINH(EXPMAX);
00074     } else if (x < EXPMIN) {
00075        (*ichop) = 1;
00076        return VSINH(EXPMIN);
00077     }
00078
00079     /* The normal SINH */
00080     (*ichop) = 0;
00081     return VSINH(x);
00082 }
00083
00084 VPUBLIC double Vcap_cosh(double x, int *ichop) {
00085
00086     /* The two chopped arguments */
00087     if (x > EXPMAX) {
00088        (*ichop) = 1;
00089        return VCOSH(EXPMAX);
00090     } else if (x < EXPMIN) {
00091        (*ichop) = 1;
00092        return VCOSH(EXPMIN);
00093     }
00094
00095     /* The normal COSH */
00096     (*ichop) = 0;
00097     return VCOSH(x);
00098 }
00099
```

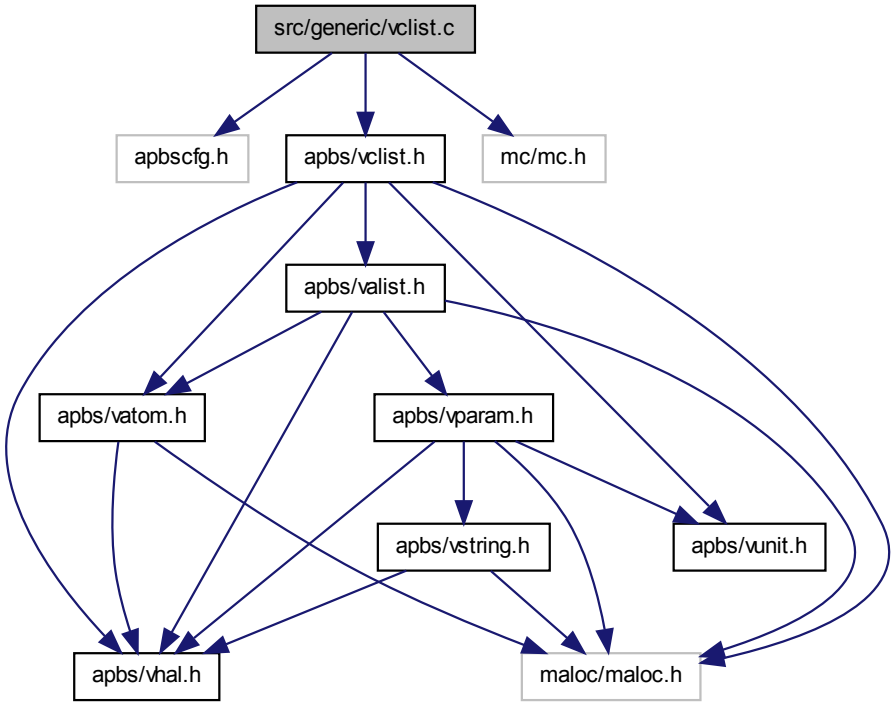
10.69 src/generic/vclist.c File Reference

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 1552 2010-02-10 17:46:27Z yhuang01

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
* 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 Washington University in St. Louis 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 src/generic/vclist.c

```
00001
00049 #include "apbscfg.h"
00050 #include "apbs/vclist.h"
00051
00052 #if defined(HAVE_MC_H)
00053 #include "mc/mc.h"
00054 #endif
00055
00056 VEMBED(rcsid="$Id: vclist.c 1552 2010-02-10 17:46:27Z yhuang01 $")
00057
00058 #if !defined(VINLINE_VCLIST)
00059
00060 VPUBLIC unsigned long int Vclist_memChk(Vclist *thee) {
00061     if (thee == VNULL) return 0;
00062     return Vmem_bytes(thee->vmem);
00063 }
00064
00065 VPUBLIC double Vclist_maxRadius(Vclist *thee) {
00066     VASSERT(thee != VNULL);
00067     return thee->max_radius;
00068 }
00069
00070 #endif /* if !defined(VINLINE_VCLIST) */
00071
00072 VPUBLIC Vclist* Vclist_ctor(Valist *alist, double max_radius,
00073         int npts[VAPBS_DIM], Vclist_DomainMode mode,
00074         double lower_corner[VAPBS_DIM],  double upper_corner[VAPBS_DIM]) {
00075
00076     Vclist *thee = VNULL;
00077
00078     /* Set up the structure */
00079     thee = Vmem_malloc(VNULL, 1, sizeof(Vclist) );
00080     VASSERT( thee != VNULL);
00081     VASSERT( Vclist_ctor2(thee, alist, max_radius, npts, mode, lower_corner,
00082                 upper_corner) == VRC_SUCCESS );
00083     return thee;
00084 }
00085
00086 /* Get the dimensions of the molecule stored in thee->alist */
00087 VPRIVATE void Vclist_getMolDims(
00088         Vclist *thee,
00089         double lower_corner[VAPBS_DIM], /* Set to lower corner of molecule */
00090         double upper_corner[VAPBS_DIM], /* Set to lower corner of molecule */
00091         double *r_max /* Set to max atom radius */
00092         ) {
00093
00094     int i, j;
00095     double pos;
00096     Valist *alist;
00097     Vatom *atom;
```

```
00098
00099     alist = thee->alist;
00100
00101     /* Initialize */
00102     for (i=0; i<VAPBS_DIM; i++) {
00103         lower_corner[i] = VLARGE;
00104         upper_corner[i] = -VLARGE;
00105     }
00106     *r_max = -1.0;
00107
00108     /* Check each atom */
00109     for (i=0; i<Valist_getNumberAtoms(alist); i++) {
00110         atom = Valist_getAtom(alist, i);
00111   for (j=0; j<VAPBS_DIM; j++) {
00112             pos = (Vatom_getPosition(atom))[j];
00113             if ( pos < lower_corner[j] ) lower_corner[j] = pos;
00114             if ( pos > upper_corner[j] ) upper_corner[j] = pos;
00115         }
00116         if (Vatom_getRadius(atom) > *r_max) *r_max = Vatom_getRadius(atom);
00117     }
00118
00119 }
00120
00121 /* Setup lookup grid */
00122 VPRIVATE Vrc_Codes Vclist_setupGrid(Vclist *thee) {
00123
00124     /* Inflation factor ~ sqrt(2)*/
00125     #define VCLIST_INFLATE 1.42
00126
00127     int i;
00128     double length[VAPBS_DIM], r_max;
00129
00130     /* Set up the grid corners */
00131     switch (thee->mode) {
00132         case CLIST_AUTO_DOMAIN:
00133             /* Get molecule dimensions */
00134             Vclist_getMolDims(thee, thee->lower_corner, thee->upper_corner,
00135                     &r_max);
00136             /* Set up grid spacings */
00137             for (i=0; i<VAPBS_DIM; i++) {
00138                 thee->upper_corner[i] = thee->upper_corner[i]
00139                     + VCLIST_INFLATE*(r_max+thee->max_radius);
00140                 thee->lower_corner[i] = thee->lower_corner[i]
00141                     - VCLIST_INFLATE*(r_max+thee->max_radius);
00142             }
00143             break;
00144         case CLIST_MANUAL_DOMAIN:
00145             /* Grid corners established in constructor */
00146             break;
00147         default:
00148             Vnm_print(2, "Vclist_setupGrid:  invalid setup mode (%d)!\n",
00149                     thee->mode);
00150             return VRC_FAILURE;
00151     }
00152
00153     /* Set up the grid lengths and spacings */
00154     for (i=0; i<VAPBS_DIM; i++) {
```

```
00155         length[i] = thee->upper_corner[i] - thee->lower_corner[i];
00156         thee->spacs[i] = length[i]/((double)(thee->npts[i] - 1));
00157     }
00158     Vnm_print(0, "Vclist_setupGrid:  Grid lengths = (%g, %g, %g)\n",
00159             length[0], length[1], length[2]);
00160
00161     Vnm_print(0, "Vclist_setupGrid:  Grid lower corner = (%g, %g, %g)\n",
00162       (thee->lower_corner)[0], (thee->lower_corner)[1],
00163       (thee->lower_corner)[2]);
00164
00165     return VRC_SUCCESS;
00166
00167     #undef VCLIST_INFLATE
00168 }
00169
00170 /* Check and store parameters passed to constructor */
00171 VPRIVATE Vrc_Codes Vclist_storeParms(Vclist *thee, Valist *alist,
00172         double max_radius, int npts[VAPBS_DIM], Vclist_DomainMode mode,
00173         double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM] ) {
00174
00175     int i = 0;
00176
00177     if (alist == VNULL) {
00178         Vnm_print(2, "Vclist_ctor2:  Got NULL Valist!\n");
00179         return VRC_FAILURE;
00180     } else thee->alist = alist;
00181
00182     thee->n = 1;
00183     for (i=0; i<VAPBS_DIM; i++) {
00184         if (npts[i] < 3) {
00185             Vnm_print(2,
00186                     "Vclist_ctor2:  n[%d] (%d) must be greater than 2!\n",
00187                     i, npts[i]);
00188             return VRC_FAILURE;
00189         }
00190         thee->npts[i] = npts[i];
00191         thee->n *= npts[i];
00192     }
00193     Vnm_print(0, "Vclist_ctor2:  Using %d x %d x %d hash table\n",
00194             npts[0], npts[1], npts[2]);
00195
00196     thee->mode = mode;
00197     switch (thee->mode) {
00198         case CLIST_AUTO_DOMAIN:
00199             Vnm_print(0, "Vclist_ctor2:  automatic domain setup.\n");
00200             break;
00201         case CLIST_MANUAL_DOMAIN:
00202             Vnm_print(0, "Vclist_ctor2:  manual domain setup.\n");
00203             Vnm_print(0, "Vclist_ctor2:  lower corner = [ \n");
00204             for (i=0; i<VAPBS_DIM; i++) {
00205                 thee->lower_corner[i] = lower_corner[i];
00206                 Vnm_print(0, "%g ", lower_corner[i]);
00207             }
00208             Vnm_print(0, "]\n");
00209             Vnm_print(0, "Vclist_ctor2:  upper corner = [ \n");
00210             for (i=0; i<VAPBS_DIM; i++) {
00211                 thee->upper_corner[i] = upper_corner[i];
```

```
00212                 Vnm_print(0, "%g ", upper_corner[i]);
00213             }
00214             Vnm_print(0, "]\n");
00215             break;
00216         default:
00217             Vnm_print(2, "Vclist_ctor2:  invalid setup mode (%d)!\n", mode);
00218             return VRC_FAILURE;
00219     }
00220
00221     thee->max_radius = max_radius;
00222     Vnm_print(0, "Vclist_ctor2:  Using %g max radius\n", max_radius);
00223
00224     return VRC_SUCCESS;
00225 }
00226
00227 /* Calculate the gridpoints an atom spans */
00228 VPRIVATE void Vclist_gridSpan(Vclist *thee,
00229         Vatom *atom, /* Atom */
00230         int imin[VAPBS_DIM], /* Set to min grid indices */
00231         int imax[VAPBS_DIM]  /* Set to max grid indices */
00232         ) {
00233
00234     int i;
00235     double *coord, dc, idc, rtot;
00236
00237     /* Get the position in the grid's frame of reference */
00238     coord = Vatom_getPosition(atom);
00239
00240     /* Get the range the atom radius + probe radius spans */
00241     rtot = Vatom_getRadius(atom) + thee->max_radius;
00242
00243     /* Calculate the range of grid points the inflated atom spans in the x
00244      * direction. */
00245     for (i=0; i<VAPBS_DIM; i++) {
00246         dc = coord[i] - (thee->lower_corner)[i];
00247         idc = (dc + rtot)/(thee->spacs[i]);
00248         imax[i] = (int)(ceil(idc));
00249         imax[i] = VMIN2(imax[i], thee->npts[i]-1);
00250         idc = (dc - rtot)/(thee->spacs[i]);
00251         imin[i] = (int)(floor(idc));
00252         imin[i] = VMAX2(imin[i], 0);
00253     }
00254
00255 }
00256
00257 /* Get the array index for a particular cell based on its i,j,k
00258  * coordinates */
00259 VPRIVATE int Vclist_arrayIndex(Vclist *thee, int i, int j, int k) {
00260
00261     return (thee->npts[2])*(thee->npts[1])*i + (thee->npts[2])*j + k;
00262
00263 }
00264
00265
00266 /* Assign atoms to cells */
00267 VPRIVATE Vrc_Codes Vclist_assignAtoms(Vclist *thee) {
00268
```

```
00269     int iatom, i, j, k, ui, inext;
00270     int imax[VAPBS_DIM], imin[VAPBS_DIM];
00271     int totatoms;
00272     Vatom *atom;
00273     VclistCell *cell;
00274
00275
00276     /* Find out how many atoms are associated with each grid point */
00277     totatoms = 0;
00278     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
00279
00280         /* Get grid span for atom */
00281         atom = Valist_getAtom(thee->alist, iatom);
00282         Vclist_gridSpan(thee, atom, imin, imax);
00283
00284         /* Now find and assign the grid points */
00285         VASSERT(VAPBS_DIM == 3);
00286         for ( i = imin[0]; i <= imax[0]; i++) {
00287             for ( j = imin[1]; j <= imax[1]; j++) {
00288                 for ( k = imin[2]; k <= imax[2]; k++) {
00289                     /* Get index to array */
00290                     ui = Vclist_arrayIndex(thee, i, j, k);
00291                     /* Increment number of atoms for this grid point */
00292                     cell = &(thee->cells[ui]);
00293                     (cell->natoms)++;
00294                     totatoms++;
00295                 }
00296             }
00297         }
00298     }
00299     Vnm_print(0, "Vclist_assignAtoms:  Have %d atom entries\n", totatoms);
00300
00301     /* Allocate the space to store the pointers to the atoms */
00302     for (ui=0; ui<thee->n; ui++) {
00303         cell = &(thee->cells[ui]);
00304         if ( VclistCell_ctor2(cell, cell->natoms) == VRC_FAILURE ) {
00305             Vnm_print(2, "Vclist_assignAtoms:  cell error!\n");
00306             return VRC_FAILURE;
00307         }
00308         /* Clear the counter for later use */
00309         cell->natoms = 0;
00310     }
00311
00312     /* Assign the atoms to grid points */
00313     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
00314
00315         /* Get grid span for atom */
00316         atom = Valist_getAtom(thee->alist, iatom);
00317         Vclist_gridSpan(thee, atom, imin, imax);
00318
00319         /* Now find and assign the grid points */
00320         for (i = imin[0]; i <= imax[0]; i++) {
00321             for (j = imin[1]; j <= imax[1]; j++) {
00322                 for (k = imin[2]; k <= imax[2]; k++) {
00323                     /* Get index to array */
00324                     ui = Vclist_arrayIndex(thee, i, j, k);
00325                     cell = &(thee->cells[ui]);
```

```
00326                     /* Index of next available array location */
00327                     inext = cell->natoms;
00328                     cell->atoms[inext] = atom;
00329                     /* Increment number of atoms */
00330                     (cell->natoms)++;
00331                 }
00332             }
00333         }
00334     }
00335
00336     return VRC_SUCCESS;
00337 }
00338
00339 /* Main (FORTRAN stub) constructor */
00340 VPUBLIC Vrc_Codes Vclist_ctor2(Vclist *thee, Valist *alist, double max_radius,
00341         int npts[VAPBS_DIM], Vclist_DomainMode mode,
00342         double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM]) {
00343
00344     int i;
00345     VclistCell *cell;
00346
00347     /* Check and store parameters */
00348     if ( Vclist_storeParms(thee, alist, max_radius, npts, mode, lower_corner,
00349                 upper_corner) == VRC_FAILURE ) {
00350         Vnm_print(2, "Vclist_ctor2:  parameter check failed!\n");
00351         return VRC_FAILURE;
00352     }
00353
00354     /* Set up memory */
00355     thee->vmem = Vmem_ctor("APBS::VCLIST");
00356     if (thee->vmem == VNULL) {
00357         Vnm_print(2, "Vclist_ctor2:  memory object setup failed!\n");
00358         return VRC_FAILURE;
00359     }
00360
00361     /* Set up cells */
00362     thee->cells = Vmem_malloc( thee->vmem, thee->n, sizeof(VclistCell) );
00363     if (thee->cells == VNULL) {
00364         Vnm_print(2,
00365                 "Vclist_ctor2:  Failed allocating %d VclistCell objects!\n",
00366                 thee->n);
00367         return VRC_FAILURE;
00368     }
00369     for (i=0; i<thee->n; i++) {
00370         cell = &(thee->cells[i]);
00371         cell->natoms = 0;
00372     }
00373
00374     /* Set up the grid */
00375     if ( Vclist_setupGrid(thee) == VRC_FAILURE ) {
00376         Vnm_print(2, "Vclist_ctor2:  grid setup failed!\n");
00377         return VRC_FAILURE;
00378     }
00379
00380     /* Assign atoms to grid cells */
00381     if (Vclist_assignAtoms(thee) == VRC_FAILURE) {
00382         Vnm_print(2, "Vclist_ctor2:  atom assignment failed!\n");
```

```
00383         return VRC_FAILURE;
00384     }
00385
00386
00387
00388
00389
00390     return VRC_SUCCESS;
00391 }
00392
00393 /* Destructor */
00394 VPUBLIC void Vclist_dtor(Vclist **thee) {
00395
00396     if ((*thee) != VNULL) {
00397         Vclist_dtor2(*thee);
00398         Vmem_free(VNULL, 1, sizeof(Vclist), (void **)thee);
00399         (*thee) = VNULL;
00400     }
00401
00402 }
00403
00404 /* Main (stub) destructor */
00405 VPUBLIC void Vclist_dtor2(Vclist *thee) {
00406
00407     VclistCell *cell;
00408     int i;
00409
00410     for (i=0; i<thee->n; i++) {
00411         cell = &(thee->cells[i]);
00412         VclistCell_dtor2(cell);
00413     }
00414     Vmem_free(thee->vmem, thee->n, sizeof(VclistCell),
00415             (void **)&(thee->cells));
00416     Vmem_dtor(&(thee->vmem));
00417
00418 }
00419
00420 VPUBLIC VclistCell* Vclist_getCell(Vclist *thee, double pos[VAPBS_DIM]) {
00421
00422     int i, ic[VAPBS_DIM], ui;
00423     double c[VAPBS_DIM];
00424
00425     /* Convert to grid based coordinates */
00426     for (i=0; i<VAPBS_DIM; i++) {
00427         c[i] = pos[i] - (thee->lower_corner)[i];
00428         ic[i] = (int)(c[i]/thee->spacs[i]);
00429         if (ic[i] < 0) {
00430             /* printf("OFF LOWER CORNER!\n"); */
00431             return VNULL;
00432         } else if (ic[i] >= thee->npts[i]) {
00433             /* printf("OFF UPPER CORNER!\n"); */
00434             return VNULL;
00435         }
00436     }
00437
00438     /* Get the array index */
00439     VASSERT(VAPBS_DIM == 3);
```

```
00440     ui = Vclist_arrayIndex(thee, ic[0], ic[1], ic[2]);
00441
00442     return &(thee->cells[ui]);
00443
00444 }
00445
00446 VPUBLIC VclistCell* VclistCell_ctor(int natoms) {
00447
00448     VclistCell *thee = VNULL;
00449
00450     /* Set up the structure */
00451     thee = Vmem_malloc(VNULL, 1, sizeof(VclistCell));
00452     VASSERT( thee != VNULL);
00453     VASSERT( VclistCell_ctor2(thee, natoms) == VRC_SUCCESS );
00454
00455     return thee;
00456 }
00457
00458 VPUBLIC Vrc_Codes VclistCell_ctor2(VclistCell *thee, int natoms) {
00459
00460     if (thee == VNULL) {
00461         Vnm_print(2, "VclistCell_ctor2:  NULL thee!\n");
00462         return VRC_FAILURE;
00463     }
00464
00465     thee->natoms = natoms;
00466     if (thee->natoms > 0) {
00467         thee->atoms = Vmem_malloc(VNULL, natoms, sizeof(Vatom *));
00468         if (thee->atoms == VNULL) {
00469             Vnm_print(2,
00470           "VclistCell_ctor2:  unable to allocate space for %d atom pointers!\n",
00471               natoms);
00472           return VRC_FAILURE;
00473         }
00474     }
00475
00476     return VRC_SUCCESS;
00477
00478 }
00479
00480 VPUBLIC void VclistCell_dtor(VclistCell **thee) {
00481
00482     if ((*thee) != VNULL) {
00483         VclistCell_dtor2(*thee);
00484         Vmem_free(VNULL, 1, sizeof(VclistCell), (void **)thee);
00485         (*thee) = VNULL;
00486     }
00487
00488 }
00489
00490 /* Main (stub) destructor */
00491 VPUBLIC void VclistCell_dtor2(VclistCell *thee) {
00492
00493     if (thee->natoms > 0) {
00494         Vmem_free(VNULL, thee->natoms, sizeof(Vatom *),
00495                 (void **)&(thee->atoms));
00496     }
```

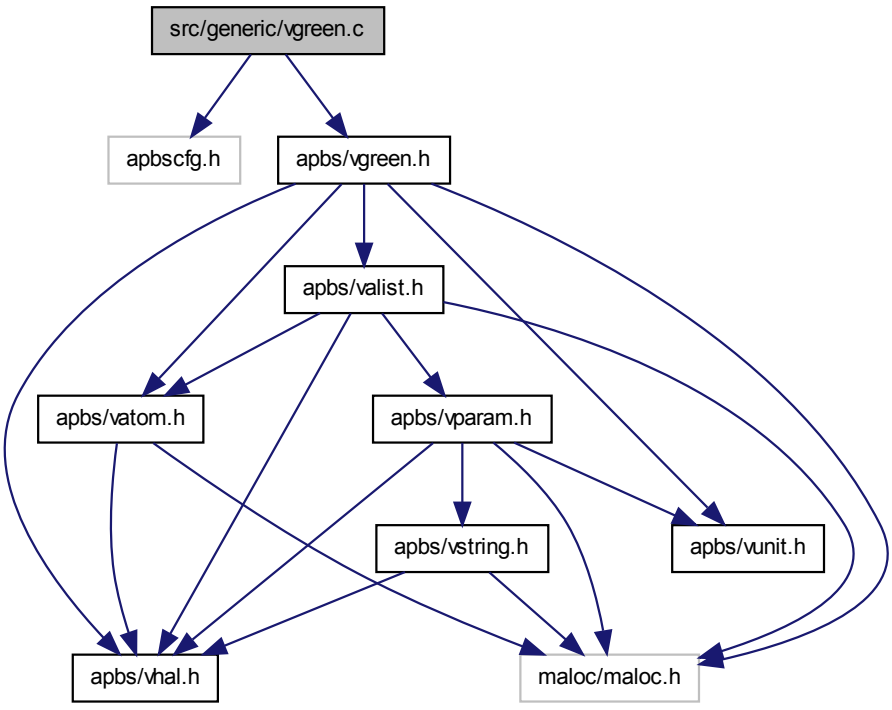
```
00497
00498 }
00499
```

10.71 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 **treecleanup** (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 1552 2010-02-10 17:46:27Z yhuang01

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
* 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 Washington University in St. Louis 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 src/generic/vgreen.c

```
00001
00049 #include "apbscfg.h"
00050 #include "apbs/vgreen.h"
00051
00052 /* Define wrappers for F77 treecode routines */
00053 #ifdef HAVE_TREE
00054 #  define F77TREEPEFORCE VF77_MANGLE(treepeforce, TREEPEFORCE)
00055 #  define F77DIRECT_ENG_FORCE VF77_MANGLE(direct_eng_force, DIRECT_ENG_FORCE)
00056 #  define F77CLEANUP VF77_MANGLE(mycleanup, MYCLEANUP)
00057 #  define F77TREE_COMPP VF77_MANGLE(mytree_compp, MYTREE_COMP)
00058 #  define F77TREE_COMPFP VF77_MANGLE(mytree_compfp, MYTREE_COMPFP)
00059 #  define F77CREATE_TREE VF77_MANGLE(mycreate_tree, MYCREATE_TREE)
00060 #  define F77INITLEVELS VF77_MANGLE(myinitlevels, MYINITLEVELS)
00061 #  define F77SETUP VF77_MANGLE(mysetup, MYSETUP)
00062 #endif  /* ifdef HAVE_TREE */
00063
00064 /* Some constants associated with the tree code */
00065 #ifdef HAVE_TREE
00066
00069 #   define FMM_DIST_TOL VSMALL
00070
00075 #   define FMM_IFLAG 2
00076
00079 #   define FMM_ORDER 4
00080
00083 #   define FMM_THETA 0.5
00084
```

```
00087 #   define FMM_MAXPARNODE 150
00088
00091 #   define FMM_SHRINK 1
00092
00095 #   define FMM_MINLEVEL 50000
00096
00099 #   define FMM_MAXLEVEL 0
00100 #endif  /* ifdef HAVE_TREE */
00101
00102
00103 /*
00104  * @brief  Setup treecode internal structures
00105  * @ingroup  Vgreen
00106  * @author  Nathan Baker
00107  * @param  thee  Vgreen object
00108  * @return  1 if successful, 0 otherwise
00109  */
00110 VPRIVATE int treesetup(Vgreen *thee);
00111
00112 /*
00113  * @brief  Clean up treecode internal structures
00114  * @ingroup  Vgreen
00115  * @author  Nathan Baker
00116  * @param  thee  Vgreen object
00117  * @return  1 if successful, 0 otherwise
00118  */
00119 VPRIVATE int treecleanup(Vgreen *thee);
00120
00121 /*
00122  * @brief  Calculate forces or potential
00123  * @ingroup  Vgreen
00124  * @author  Nathan Baker
00125  * @param  thee  Vgreen object
00126  * @return  1 if successful, 0 otherwise
00127  */
00128 VPRIVATE int treecalc(Vgreen *thee, double *xtar, double *ytar, double *ztar,
00129         double *qtar, int numtars, double *tpengtar, double *x, double *y,
00130         double *z, double *q, int numpars, double *fx, double *fy, double *fz,
00131         int iflag, int farrdim, int arrdim);
00132
00133 #if !defined(VINLINE_VGREEN)
00134
00135 VPUBLIC Valist* Vgreen_getValist(Vgreen *thee) {
00136
00137    VASSERT(thee != VNULL);
00138    return thee->alist;
00139
00140 }
00141
00142 VPUBLIC unsigned long int Vgreen_memChk(Vgreen *thee) {
00143     if (thee == VNULL) return 0;
00144     return Vmem_bytes(thee->vmem);
00145 }
00146
00147 #endif /* if !defined(VINLINE_VGREEN) */
00148
00149 VPUBLIC Vgreen* Vgreen_ctor(Valist *alist) {
```

```
00150
00151     /* Set up the structure */
00152     Vgreen *thee = VNULL;
00153     thee = (Vgreen *)Vmem_malloc(VNULL, 1, sizeof(Vgreen) );
00154     VASSERT( thee != VNULL);
00155     VASSERT( Vgreen_ctor2(thee, alist));
00156
00157     return thee;
00158 }
00159
00160 VPUBLIC int Vgreen_ctor2(Vgreen *thee, Valist *alist) {
00161
00162     VASSERT( thee != VNULL );
00163
00164     /* Memory management object */
00165     thee->vmem = Vmem_ctor("APBS:VGREEN");
00166
00167     /* Set up the atom list and grid manager */
00168     if (alist == VNULL) {
00169         Vnm_print(2,"Vgreen_ctor2: got null pointer to Valist object!\n");
00170     }
00171
00172     thee->alist = alist;
00173
00174     /* Setup FMM tree (if applicable) */
00175 #ifdef HAVE_TREE
00176     if (!treesetup(thee)) {
00177         Vnm_print(2, "Vgreen_ctor2:  Error setting up FMM tree!\n");
00178         return 0;
00179     }
00180 #endif /* ifdef HAVE_TREE */
00181
00182     return 1;
00183 }
00184
00185 VPUBLIC void Vgreen_dtor(Vgreen **thee) {
00186     if ((*thee) != VNULL) {
00187         Vgreen_dtor2(*thee);
00188         Vmem_free(VNULL, 1, sizeof(Vgreen), (void **)thee);
00189         (*thee) = VNULL;
00190     }
00191 }
00192
00193 VPUBLIC void Vgreen_dtor2(Vgreen *thee) {
00194
00195 #ifdef HAVE_TREE
00196     treecleanup(thee);
00197 #endif
00198     Vmem_dtor(&(thee->vmem));
00199
00200 }
00201
00202 VPUBLIC int Vgreen_helmholtz(Vgreen *thee, int npos, double *x, double *y,
00203   double *z, double *val, double kappa) {
00204
00205     Vnm_print(2, "Error -- Vgreen_helmholtz not implemented yet!\n");
00206     return 0;
```

```
00207 }
00208
00209 VPUBLIC int Vgreen_helmholtzD(Vgreen *thee, int npos, double *x, double *y,
00210   double *z, double *gradx, double *grady, double *gradz, double kappa) {
00211
00212     Vnm_print(2, "Error -- Vgreen_helmholtzD not implemented yet!\n");
00213     return 0;
00214
00215 }
00216
00217 VPUBLIC int Vgreen_coulomb_direct(Vgreen *thee, int npos, double *x,
00218         double *y, double *z, double *val) {
00219
00220     Vatom *atom;
00221     double *apos, charge, dist, dx, dy, dz, scale;
00222     double *q, qtemp, fx, fy, fz;
00223     int iatom, ipos;
00224
00225     if (thee == VNULL) {
00226         Vnm_print(2, "Vgreen_coulomb:  Got NULL thee!\n");
00227         return 0;
00228     }
00229
00230     for (ipos=0; ipos<npos; ipos++) val[ipos] = 0.0;
00231
00232     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
00233         atom = Valist_getAtom(thee->alist, iatom);
00234         apos = Vatom_getPosition(atom);
00235         charge = Vatom_getCharge(atom);
00236         for (ipos=0; ipos<npos; ipos++) {
00237             dx = apos[0] - x[ipos];
00238             dy = apos[1] - y[ipos];
00239             dz = apos[2] - z[ipos];
00240             dist = VSQRT(VSQR(dx) + VSQR(dy) + VSQR(dz));
00241             if (dist > VSMALL) val[ipos] += (charge/dist);
00242         }
00243     }
00244
00245     scale = Vunit_ec/(4*Vunit_pi*Vunit_eps0*1.0e-10);
00246     for (ipos=0; ipos<npos; ipos++) val[ipos] = val[ipos]*scale;
00247
00248     return 1;
00249 }
00250
00251 VPUBLIC int Vgreen_coulomb(Vgreen *thee, int npos, double *x, double *y,
00252   double *z, double *val) {
00253
00254     Vatom *atom;
00255     double *apos, charge, dist, dx, dy, dz, scale;
00256     double *q, qtemp, fx, fy, fz;
00257     int iatom, ipos;
00258
00259     if (thee == VNULL) {
00260         Vnm_print(2, "Vgreen_coulomb:  Got NULL thee!\n");
00261         return 0;
00262     }
00263
```

```
00264     for (ipos=0; ipos<npos; ipos++) val[ipos] = 0.0;
00265
00266 #ifdef HAVE_TREE
00267
00268     /* Allocate charge array (if necessary) */
00269     if (Valist_getNumberAtoms(thee->alist) > 1) {
00270         if (npos > 1) {
00271             q = VNULL;
00272             q = Vmem_malloc(thee->vmem, npos, sizeof(double));
00273             if (q == VNULL) {
00274                 Vnm_print(2, "Vgreen_coulomb:  Error allocating charge array!\n")
      ;
00275                 return 0;
00276             }
00277         } else {
00278             q = &(qtemp);
00279         }
00280         for (ipos=0; ipos<npos; ipos++) q[ipos] = 1.0;
00281
00282         /* Calculate */
00283         treecalc(thee, x, y, z, q, npos, val, thee->xp, thee->yp, thee->zp,
00284           thee->qp, thee->np, &fx, &fy, &fz, 1, 1, thee->np);
00285     } else return Vgreen_coulomb_direct(thee, npos, x, y, z, val);
00286
00287     /* De-allocate charge array (if necessary) */
00288     if (npos > 1) Vmem_free(thee->vmem, npos, sizeof(double), (void **)&q);
00289
00290     scale = Vunit_ec/(4*Vunit_pi*Vunit_eps0*1.0e-10);
00291     for (ipos=0; ipos<npos; ipos++) val[ipos] = val[ipos]*scale;
00292
00293     return 1;
00294
00295 #else /* ifdef HAVE_TREE */
00296
00297     return Vgreen_coulomb_direct(thee, npos, x, y, z, val);
00298
00299 #endif
00300
00301 }
00302
00303 VPUBLIC int Vgreen_coulombD_direct(Vgreen *thee, int npos,
00304         double *x, double *y, double *z, double *pot, double *gradx,
00305         double *grady, double *gradz) {
00306
00307     Vatom *atom;
00308     double *apos, charge, dist, dist2, idist3, dy, dz, dx, scale;
00309     double *q, qtemp;
00310     int iatom, ipos;
00311
00312     if (thee == VNULL) {
00313         Vnm_print(2, "Vgreen_coulombD:  Got VNULL thee!\n");
00314         return 0;
00315     }
00316
00317     for (ipos=0; ipos<npos; ipos++) {
00318         pot[ipos] = 0.0;
00319         gradx[ipos] = 0.0;
```

```
00320         grady[ipos] = 0.0;
00321         gradz[ipos] = 0.0;
00322     }
00323
00324     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
00325         atom = Valist_getAtom(thee->alist, iatom);
00326         apos = Vatom_getPosition(atom);
00327         charge = Vatom_getCharge(atom);
00328         for (ipos=0; ipos<npos; ipos++) {
00329             dx = apos[0] - x[ipos];
00330             dy = apos[1] - y[ipos];
00331             dz = apos[2] - z[ipos];
00332             dist2 = VSQR(dx) + VSQR(dy) + VSQR(dz);
00333             dist = VSQRT(dist2);
00334             if (dist > VSMALL) {
00335                 idist3 = 1.0/(dist*dist2);
00336                 gradx[ipos] -= (charge*dx*idist3);
00337                 grady[ipos] -= (charge*dy*idist3);
00338                 gradz[ipos] -= (charge*dz*idist3);
00339                 pot[ipos] += (charge/dist);
00340             }
00341         }
00342     }
00343
00344     scale = Vunit_ec/(4*VPI*Vunit_eps0*(1.0e-10));
00345     for (ipos=0; ipos<npos; ipos++) {
00346         gradx[ipos] = gradx[ipos]*scale;
00347         grady[ipos] = grady[ipos]*scale;
00348         gradz[ipos] = gradz[ipos]*scale;
00349         pot[ipos] = pot[ipos]*scale;
00350     }
00351
00352     return 1;
00353 }
00354
00355 VPUBLIC int Vgreen_coulombD(Vgreen *thee, int npos, double *x, double *y,
00356         double *z, double *pot, double *gradx, double *grady, double *gradz) {
00357
00358     Vatom *atom;
00359     double *apos, charge, dist, dist2, idist3, dy, dz, dx, scale;
00360     double *q, qtemp;
00361     int iatom, ipos;
00362
00363     if (thee == VNULL) {
00364         Vnm_print(2, "Vgreen_coulombD:  Got VNULL thee!\n");
00365         return 0;
00366     }
00367
00368     for (ipos=0; ipos<npos; ipos++) {
00369         pot[ipos] = 0.0;
00370         gradx[ipos] = 0.0;
00371         grady[ipos] = 0.0;
00372         gradz[ipos] = 0.0;
00373     }
00374
00375 #ifdef HAVE_TREE
00376
```

```
00377     if (Valist_getNumberAtoms(thee->alist) > 1) {
00378         if (npos > 1) {
00379             q = VNULL;
00380             q = Vmem_malloc(thee->vmem, npos, sizeof(double));
00381             if (q == VNULL) {
00382                 Vnm_print(2, "Vgreen_coulomb:  Error allocating charge array!\n")
      ;
00383                 return 0;
00384             }
00385         } else {
00386             q = &(qtemp);
00387         }
00388         for (ipos=0; ipos<npos; ipos++) q[ipos] = 1.0;
00389 
00390         /* Calculate */
00391         treecalc(thee, x, y, z, q, npos, pot, thee->xp, thee->yp, thee->zp,
00392                 thee->qp, thee->np, gradx, grady, gradz, 2, npos, thee->np);
00393 
00394         /* De-allocate charge array (if necessary) */
00395         if (npos > 1) Vmem_free(thee->vmem, npos, sizeof(double), (void **)&q);
00396     } else return Vgreen_coulombD_direct(thee, npos, x, y, z, pot,
00397             gradx, grady, gradz);
00398 
00399     scale = Vunit_ec/(4*VPI*Vunit_eps0*(1.0e-10));
00400     for (ipos=0; ipos<npos; ipos++) {
00401         gradx[ipos] = gradx[ipos]*scale;
00402         grady[ipos] = grady[ipos]*scale;
00403         gradz[ipos] = gradz[ipos]*scale;
00404         pot[ipos] = pot[ipos]*scale;
00405     }
00406 
00407     return 1;
00408 
00409 #else /* ifdef HAVE_TREE */
00410 
00411     return Vgreen_coulombD_direct(thee, npos, x, y, z, pot,
00412             gradx, grady, gradz);
00413 
00414 #endif
00415 
00416 }
00417 
00418 VPRIVATE int treesetup(Vgreen *thee) {
00419 
00420 #ifdef HAVE_TREE
00421 
00422     double dist_tol = FMM_DIST_TOL;
00423     int iflag = FMM_IFLAG;
00424     double order = FMM_ORDER;
00425     int theta = FMM_THETA;
00426     int shrink = FMM_SHRINK;
00427     int maxparnode = FMM_MAXPARNODE;
00428     int minlevel = FMM_MINLEVEL;
00429     int maxlevel = FMM_MAXLEVEL;
00430     int level = 0;
00431     int one = 1;
00432     Vatom *atom;
```

```
00433     double xyzminmax[6], *pos;
00434     int i;
00435
00436     /* Set up particle arrays with atomic coordinates and charges */
00437     Vnm_print(0, "treesetup:  Initializing FMM particle arrays...\n");
00438     thee->np = Valist_getNumberAtoms(thee->alist);
00439     thee->xp = VNULL;
00440     thee->xp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(double));
00441     if (thee->xp == VNULL) {
00442         Vnm_print(2, "Vgreen_ctor2:  Failed to allocate %d*sizeof(double)!\n",
00443           thee->np);
00444         return 0;
00445     }
00446     thee->yp = VNULL;
00447     thee->yp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(double));
00448     if (thee->yp == VNULL) {
00449         Vnm_print(2, "Vgreen_ctor2:  Failed to allocate %d*sizeof(double)!\n",
00450           thee->np);
00451         return 0;
00452     }
00453     thee->zp = VNULL;
00454     thee->zp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(double));
00455     if (thee->zp == VNULL) {
00456         Vnm_print(2, "Vgreen_ctor2:  Failed to allocate %d*sizeof(double)!\n",
00457           thee->np);
00458         return 0;
00459     }
00460     thee->qp = VNULL;
00461     thee->qp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(double));
00462     if (thee->qp == VNULL) {
00463         Vnm_print(2, "Vgreen_ctor2:  Failed to allocate %d*sizeof(double)!\n",
00464           thee->np);
00465         return 0;
00466     }
00467     for (i=0; i<thee->np; i++) {
00468         atom = Valist_getAtom(thee->alist, i);
00469         pos = Vatom_getPosition(atom);
00470         thee->xp[i] = pos[0];
00471         thee->yp[i] = pos[1];
00472         thee->zp[i] = pos[2];
00473         thee->qp[i] = Vatom_getCharge(atom);
00474     }
00475
00476     Vnm_print(0, "treesetup:  Setting things up...\n");
00477     F77SETUP(thee->xp, thee->yp, thee->zp, &(thee->np), &order, &theta, &iflag,
00478             &dist_tol, xyzminmax, &(thee->np));
00479
00480
00481     Vnm_print(0, "treesetup:  Initializing levels...\n");
00482     F77INITLEVELS(&minlevel, &maxlevel);
00483
00484     Vnm_print(0, "treesetup:  Creating tree...\n");
00485     F77CREATE_TREE(&one, &(thee->np), thee->xp, thee->yp, thee->zp, thee->qp,
00486       &shrink, &maxparnode, xyzminmax, &level, &(thee->np));
00487
00488     return 1;
00489
```

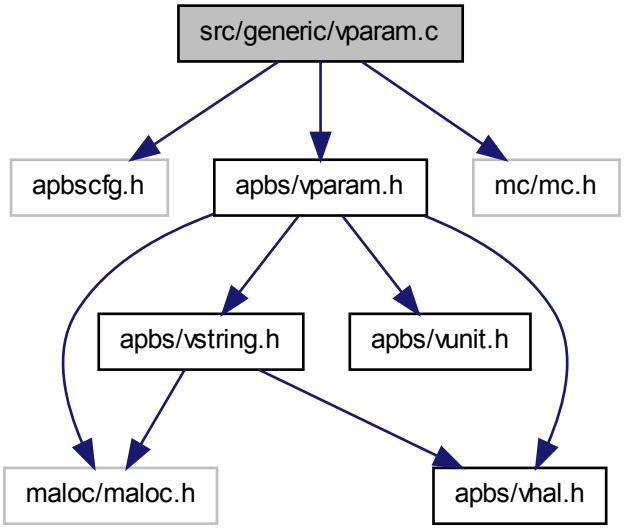
```
00490 #else /* ifdef HAVE_TREE */
00491
00492     Vnm_print(2, "treesetup:  Error!  APBS not linked with treecode!\n");
00493     return 0;
00494
00495 #endif /* ifdef HAVE_TREE */
00496 }
00497
00498 VPRIVATE int treecleanup(Vgreen *thee) {
00499
00500 #ifdef HAVE_TREE
00501
00502     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->xp));
00503     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->yp));
00504     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->zp));
00505     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->qp));
00506     F77CLEANUP();
00507
00508     return 1;
00509
00510 #else /* ifdef HAVE_TREE */
00511
00512     Vnm_print(2, "treecleanup:  Error!  APBS not linked with treecode!\n");
00513     return 0;
00514
00515 #endif /* ifdef HAVE_TREE */
00516 }
00517
00518 VPRIVATE int treecalc(Vgreen *thee, double *xtar, double *ytar, double *ztar,
00519         double *qtar, int numtars, double *tpengtar, double *x, double *y,
00520         double *z, double *q, int numpars, double *fx, double *fy, double *fz,
00521         int iflag, int farrdim, int arrdim) {
00522
00523 #ifdef HAVE_TREE
00524     int i, level, err, maxlevel, minlevel, one;
00525     double xyzminmax[6];
00526
00527
00528     if (iflag != 1) {
00529         F77TREE_COMPFP(xtar, ytar, ztar, qtar, &numtars, tpengtar, x, y, z, q,
00530                 fx, fy, fz, &numpars, &farrdim, &arrdim);
00531     } else {
00532         F77TREE_COMPP(xtar, ytar, ztar, qtar, &numtars, tpengtar, &farrdim, x,
00533                 y, z, q, &numpars, &arrdim);
00534     }
00535
00536
00537     return 1;
00538
00539 #else /* ifdef HAVE_TREE */
00540
00541     Vnm_print(2, "treecalc:  Error!  APBS not linked with treecode!\n");
00542     return 0;
00543
00544 #endif /* ifdef HAVE_TREE */
00545 }
00546
```

10.73 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

- VPRIVATE char * MCwhiteChars = " =,;\t\n\r"

  Whitespace characters for socket reads.

- VPRIVATE char * MCcommChars = "#%"

  Comment characters for socket reads.

- VPRIVATE 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 1552 2010-02-10 17:46:27Z yhuang01

Attention

Definition in file vparam.c.

10.74 src/generic/vparam.c

```
00001
00049 #include "apbscfg.h"
00050 #include "apbs/vparam.h"
00051
00052 #if defined(HAVE_MC_H)
00053 #include "mc/mc.h"
00054 #endif
00055
00056 VEMBED(rcsid="$Id: vparam.c 1552 2010-02-10 17:46:27Z yhuang01 $")
00057
00058
00062 VPRIVATE char *MCwhiteChars = " =,;\t\n\r";
00063
00068 VPRIVATE char *MCcommChars  = "#%";
00069
00074 VPRIVATE char *MCxmlwhiteChars = " =,;\t\n\r<>";
00075
00084 VPRIVATE int readFlatFileLine(Vio *sock, Vparam_AtomData *atom);
00085
00094 VPRIVATE int readXMLFileAtom(Vio *sock, Vparam_AtomData *atom);
00095
00096
00097 #if !defined(VINLINE_VPARAM)
00098
00099 VPUBLIC unsigned long int Vparam_memChk(Vparam *thee) {
00100     if (thee == VNULL) return 0;
00101     return Vmem_bytes(thee->vmem);
00102 }
00103
00104 #endif /* if !defined(VINLINE_VPARAM) */
00105
00106 VPUBLIC Vparam_AtomData* Vparam_AtomData_ctor() {
00107
00108     Vparam_AtomData *thee = VNULL;
00109
00110     /* Set up the structure */
00111     thee = Vmem_malloc(VNULL, 1, sizeof(Vparam_AtomData) );
00112     VASSERT(thee != VNULL);
00113     VASSERT(Vparam_AtomData_ctor2(thee));
00114
00115     return thee;
00116 }
00117
00118 VPUBLIC int Vparam_AtomData_ctor2(Vparam_AtomData *thee) { return 1; }
00119
00120 VPUBLIC void Vparam_AtomData_dtor(Vparam_AtomData **thee) {
00121
00122     if ((*thee) != VNULL) {
00123         Vparam_AtomData_dtor2(*thee);
00124         Vmem_free(VNULL, 1, sizeof(Vparam_AtomData), (void **)thee);
00125         (*thee) = VNULL;
00126     }
00127
00128 }
00129
```

```
00130 VPUBLIC void Vparam_AtomData_dtor2(Vparam_AtomData *thee) { ; }
00131
00132 VPUBLIC Vparam_ResData* Vparam_ResData_ctor(Vmem *mem) {
00133
00134     Vparam_ResData *thee = VNULL;
00135
00136     /* Set up the structure */
00137     thee = Vmem_malloc(mem, 1, sizeof(Vparam_ResData) );
00138     VASSERT(thee != VNULL);
00139     VASSERT(Vparam_ResData_ctor2(thee, mem));
00140
00141     return thee;
00142 }
00143
00144 VPUBLIC int Vparam_ResData_ctor2(Vparam_ResData *thee, Vmem *mem) {
00145
00146     if (thee == VNULL) {
00147         Vnm_print(2, "Vparam_ResData_ctor2:  Got VNULL thee!\n");
00148         return 0;
00149     }
00150     thee->vmem = mem;
00151     thee->nAtomData = 0;
00152     thee->atomData = VNULL;
00153
00154     return 1;
00155 }
00156
00157 VPUBLIC void Vparam_ResData_dtor(Vparam_ResData **thee) {
00158
00159     if ((*thee) != VNULL) {
00160         Vparam_ResData_dtor2(*thee);
00161         Vmem_free((*thee)->vmem, 1, sizeof(Vparam_ResData), (void **)thee);
00162         (*thee) = VNULL;
00163     }
00164
00165 }
00166
00167 VPUBLIC void Vparam_ResData_dtor2(Vparam_ResData *thee) {
00168
00169     if (thee == VNULL) return;
00170     if (thee->nAtomData > 0) {
00171         Vmem_free(thee->vmem, thee->nAtomData, sizeof(Vparam_AtomData),
00172           (void **)&(thee->atomData));
00173     }
00174     thee->nAtomData = 0;
00175     thee->atomData = VNULL;
00176 }
00177
00178 VPUBLIC Vparam* Vparam_ctor() {
00179
00180     Vparam *thee = VNULL;
00181
00182     /* Set up the structure */
00183     thee = Vmem_malloc(VNULL, 1, sizeof(Vparam) );
00184     VASSERT(thee != VNULL);
00185     VASSERT(Vparam_ctor2(thee));
00186
```

```
00187     return thee;
00188 }
00189
00190 VPUBLIC int Vparam_ctor2(Vparam *thee) {
00191
00192     if (thee == VNULL) {
00193         Vnm_print(2, "Vparam_ctor2: got VNULL thee!\n");
00194         return 0;
00195     }
00196
00197     thee->vmem = VNULL;
00198     thee->vmem = Vmem_ctor("APBS:VPARAM");
00199     if (thee->vmem == VNULL) {
00200         Vnm_print(2, "Vparam_ctor2:  failed to init Vmem!\n");
00201         return 0;
00202     }
00203
00204     thee->nResData = 0;
00205     thee->resData = VNULL;
00206
00207     return 1;
00208 }
00209
00210 VPUBLIC void Vparam_dtor(Vparam **thee) {
00211
00212     if ((*thee) != VNULL) {
00213         Vparam_dtor2(*thee);
00214         Vmem_free(VNULL, 1, sizeof(Vparam), (void **)thee);
00215         (*thee) = VNULL;
00216     }
00217
00218 }
00219
00220 VPUBLIC void Vparam_dtor2(Vparam *thee) {
00221
00222     int i;
00223
00224     if (thee == VNULL) return;
00225
00226     /* Destroy the residue data */
00227     for (i=0; i<thee->nResData; i++) Vparam_ResData_dtor2(&(thee->resData[i]));
00228     if (thee->nResData > 0) Vmem_free(thee->vmem, thee->nResData,
00229       sizeof(Vparam_ResData), (void **)&(thee->resData));
00230     thee->nResData = 0;
00231     thee->resData = VNULL;
00232
00233     if (thee->vmem != VNULL) Vmem_dtor(&(thee->vmem));
00234     thee->vmem = VNULL;
00235
00236 }
00237
00238 VPUBLIC Vparam_ResData* Vparam_getResData(Vparam *thee,
00239   char resName[VMAX_ARGLEN]) {
00240
00241     int i;
00242     Vparam_ResData *res = VNULL;
00243
```

```
00244     VASSERT(thee != VNULL);
00245
00246     if ((thee->nResData == 0) || (thee->resData == VNULL)) {
00247         res = VNULL;
00248         return res;
00249     }
00250
00251     /* Look for the matching residue */
00252     for (i=0; i<thee->nResData; i++) {
00253         res = &(thee->resData[i]);
00254         if (Vstring_strcasecmp(resName, res->name) == 0) return res;
00255
00256     }
00257
00258     /* Didn't find a matching residue */
00259     res = VNULL;
00260     Vnm_print(2, "Vparam_getResData:  unable to find res=%s\n", resName);
00261     return res;
00262 }
00263
00264 VPUBLIC Vparam_AtomData* Vparam_getAtomData(Vparam *thee,
00265   char resName[VMAX_ARGLEN], char atomName[VMAX_ARGLEN]) {
00266
00267     int i;
00268     Vparam_ResData *res = VNULL;
00269     Vparam_AtomData *atom = VNULL;
00270
00271     VASSERT(thee != VNULL);
00272
00273     if ((thee->nResData == 0) || (thee->resData == VNULL)) {
00274         atom = VNULL;
00275         return atom;
00276     }
00277
00278     /* Look for the matching residue */
00279     res = Vparam_getResData(thee, resName);
00280     if (res == VNULL) {
00281         atom = VNULL;
00282   Vnm_print(2, "Vparam_getAtomData:  Unable to find residue %s!\n", resName);
00283         return atom;
00284     }
00285     for (i=0; i<res->nAtomData; i++) {
00286         atom = &(res->atomData[i]);
00287   if (atom == VNULL) {
00288    Vnm_print(2, "Vparam_getAtomData:  got NULL atom!\n");
00289    return VNULL;
00290   }
00291         if (Vstring_strcasecmp(atomName, atom->atomName) == 0) {
00292             return atom;
00293         }
00294     }
00295
00296     /* Didn't find a matching atom/residue */
00297     atom = VNULL;
00298     Vnm_print(2, "Vparam_getAtomData:  unable to find atom '%s', res '%s'\n",
00299       atomName, resName);
00300     return atom;
```

```
00301 }
00302
00303 VPUBLIC int Vparam_readXMLFile(Vparam *thee, const char *iodev,
00304   const char *iofmt, const char *thost, const char *fname) {
00305
00306     int i, ires, natoms, nalloc, ralloc;
00307     Vparam_AtomData *atoms = VNULL;
00308     Vparam_AtomData *tatoms = VNULL;
00309     Vparam_AtomData *atom = VNULL;
00310     Vparam_ResData *res = VNULL;
00311     Vparam_ResData *residues = VNULL;
00312     Vparam_ResData *tresidues = VNULL;
00313     Vio *sock = VNULL;
00314     char currResName[VMAX_ARGLEN];
00315     char tok[VMAX_ARGLEN];
00316     char endtag[VMAX_ARGLEN];
00317
00318     VASSERT(thee != VNULL);
00319
00320     /* Setup communication */
00321     sock = Vio_ctor(iodev,iofmt,thost,fname,"r");
00322     if (sock == VNULL) {
00323         Vnm_print(2, "Vparam_readXMLFile: Problem opening virtual socket %s\n",
00324           fname);
00325         return 0;
00326     }
00327     if (Vio_accept(sock, 0) < 0) {
00328         Vnm_print(2, "Vparam_readXMLFile: Problem accepting virtual socket %s\n",

00329           fname);
00330         return 0;
00331     }
00332     Vio_setWhiteChars(sock, MCxmlwhiteChars);
00333     Vio_setCommChars(sock, MCcommChars);
00334
00335     /* Clear existing parameters */
00336     if (thee->nResData > 0) {
00337         Vnm_print(2, "WARNING -- CLEARING PARAMETER DATABASE!\n");
00338         for (i=0; i<thee->nResData; i++) {
00339             Vparam_ResData_dtor2(&(thee->resData[i]));
00340         }
00341         Vmem_free(thee->vmem, thee->nResData,
00342           sizeof(Vparam_ResData), (void **)&(thee->resData));
00343     }
00344
00345     strcpy(endtag,"/");
00346
00347     /* Set up temporary residue list */
00348
00349     ralloc = 50;
00350     residues = Vmem_malloc(thee->vmem, ralloc, sizeof(Vparam_ResData));
00351
00352     /* Read until we run out of entries, allocating space as needed */
00353     while (1) {
00354
00355         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00356
```

```
00357         /* The first token should be the start tag */
00358
00359         if (Vstring_strcasecmp(endtag, "/") == 0) strcat(endtag, tok);
00360
00361         if (Vstring_strcasecmp(tok, "residue") == 0) {
00362           if (thee->nResData >= ralloc) {
00363                 tresidues = Vmem_malloc(thee->vmem, 2*ralloc, sizeof(
      Vparam_ResData));
00364                 VASSERT(tresidues != VNULL);
00365                 for (i=0; i<thee->nResData; i++) {
00366                     Vparam_ResData_copyTo(&(residues[i]), &(tresidues[i]));
00367                 }
00368                 Vmem_free(thee->vmem, ralloc, sizeof(Vparam_ResData),
00369                           (void **)&(residues));
00370                 residues = tresidues;
00371                 tresidues = VNULL;
00372                 ralloc = 2*ralloc;
00373             }
00374
00375           /* Initial space for this residue's atoms */
00376           nalloc = 20;
00377           natoms = 0;
00378           atoms = Vmem_malloc(thee->vmem, nalloc, sizeof(Vparam_AtomData));
00379
00380         } else if (Vstring_strcasecmp(tok, "name") == 0) {
00381             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);  /* value */
00382             strcpy(currResName, tok);
00383             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1); /* </name> */
00384         } else if (Vstring_strcasecmp(tok, "atom") == 0) {
00385             if (natoms >= nalloc) {
00386                 tatoms = Vmem_malloc(thee->vmem, 2*nalloc, sizeof(
      Vparam_AtomData));
00387                 VASSERT(tatoms != VNULL);
00388                 for (i=0; i<natoms; i++) {
00389                     Vparam_AtomData_copyTo(&(atoms[i]), &(tatoms[i]));
00390                 }
00391                 Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData),
00392                           (void **)&(atoms));
00393                 atoms = tatoms;
00394                 tatoms = VNULL;
00395                 nalloc = 2*nalloc;
00396             }
00397             atom = &(atoms[natoms]);
00398             if (!readXMLFileAtom(sock, atom)) break;
00399             natoms++;
00400
00401         } else if (Vstring_strcasecmp(tok, "/residue") == 0) {
00402
00403           res = &(residues[thee->nResData]);
00404           Vparam_ResData_ctor2(res, thee->vmem);
00405           res->atomData = Vmem_malloc(thee->vmem, natoms,
00406                                       sizeof(Vparam_AtomData));
00407           res->nAtomData = natoms;
00408           strcpy(res->name, currResName);
00409           for (i=0; i<natoms; i++) {
00410               strcpy(atoms[i].resName, currResName);
00411               Vparam_AtomData_copyTo(&(atoms[i]), &(res->atomData[i]));
```

```
00412           }
00413           Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData), (void **)&(atoms
      ));
00414           (thee->nResData)++;
00415
00416         } else if (Vstring_strcasecmp(tok, endtag) == 0) break;
00417     }
00418
00419     /* Initialize and copy the residues into the Vparam object */
00420
00421     thee->resData = Vmem_malloc(thee->vmem, thee->nResData,
00422                                 sizeof(Vparam_ResData));
00423     for (ires=0; ires<thee->nResData; ires++) {
00424         Vparam_ResData_copyTo(&(residues[ires]), &(thee->resData[ires]));
00425     }
00426
00427     /* Destroy temporary atom space */
00428     Vmem_free(thee->vmem, ralloc, sizeof(Vparam_ResData), (void **)&(residues));
00429
00430     /* Shut down communication */
00431     Vio_acceptFree(sock);
00432     Vio_dtor(&sock);
00433
00434     return 1;
00435
00436 VERROR1:
00437     Vnm_print(2, "Vparam_readXMLFile: Got unexpected EOF reading parameter file!\
      n");
00438     return 0;
00439
00440 }
00441
00442 VPUBLIC int Vparam_readFlatFile(Vparam *thee, const char *iodev,
00443   const char *iofmt, const char *thost, const char *fname) {
00444
00445     int i, iatom, jatom, ires, natoms, nalloc;
00446     Vparam_AtomData *atoms = VNULL;
00447     Vparam_AtomData *tatoms = VNULL;
00448     Vparam_AtomData *atom = VNULL;
00449     Vparam_ResData *res = VNULL;
00450     Vio *sock = VNULL;
00451     char currResName[VMAX_ARGLEN];
00452
00453     VASSERT(thee != VNULL);
00454
00455     /* Setup communication */
00456     sock = Vio_ctor(iodev,iofmt,thost,fname,"r");
00457     if (sock == VNULL) {
00458         Vnm_print(2, "Vparam_readFlatFile: Problem opening virtual socket %s\n",
00459           fname);
00460         return 0;
00461     }
00462     if (Vio_accept(sock, 0) < 0) {
00463         Vnm_print(2, "Vparam_readFlatFile: Problem accepting virtual socket %s\n"
      ,
00464           fname);
00465         return 0;
```

```
00466     }
00467     Vio_setWhiteChars(sock, MCwhiteChars);
00468     Vio_setCommChars(sock, MCcommChars);
00469
00470     /* Clear existing parameters */
00471     if (thee->nResData > 0) {
00472         Vnm_print(2, "WARNING -- CLEARING PARAMETER DATABASE!\n");
00473         for (i=0; i<thee->nResData; i++) {
00474             Vparam_ResData_dtor2(&(thee->resData[i]));
00475         }
00476         Vmem_free(thee->vmem, thee->nResData,
00477           sizeof(Vparam_ResData), (void **)&(thee->resData));
00478     }
00479
00480     /* Initial space for atoms */
00481     nalloc = 200;
00482     natoms = 0;
00483     atoms = Vmem_malloc(thee->vmem, nalloc, sizeof(Vparam_AtomData));
00484
00485     /* Read until we run out of entries, allocating space as needed */
00486     while (1) {
00487         if (natoms >= nalloc) {
00488             tatoms = Vmem_malloc(thee->vmem, 2*nalloc, sizeof(Vparam_AtomData));
00489             VASSERT(tatoms != VNULL);
00490             for (i=0; i<natoms; i++) {
00491                 Vparam_AtomData_copyTo(&(atoms[i]), &(tatoms[i]));
00492             }
00493             Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData),
00494               (void **)&(atoms));
00495             atoms = tatoms;
00496             tatoms = VNULL;
00497             nalloc = 2*nalloc;
00498         }
00499         atom = &(atoms[natoms]);
00500         if (!readFlatFileLine(sock, atom)) break;
00501         natoms++;
00502     }
00503     if (natoms == 0) return 0;
00504
00505     /* Count the number of residues */
00506     thee->nResData = 1;
00507     strcpy(currResName, atoms[0].resName);
00508     for (i=1; i<natoms; i++) {
00509         if (Vstring_strcasecmp(atoms[i].resName, currResName) != 0) {
00510             strcpy(currResName, atoms[i].resName);
00511             (thee->nResData)++;
00512         }
00513     }
00514
00515     /* Create the residues */
00516     thee->resData = Vmem_malloc(thee->vmem, thee->nResData,
00517       sizeof(Vparam_ResData));
00518     VASSERT(thee->resData != VNULL);
00519     for (i=0; i<(thee->nResData); i++) {
00520         res = &(thee->resData[i]);
00521         Vparam_ResData_ctor2(res, thee->vmem);
00522     }
```

```
00523
00524     /* Count the number of atoms per residue */
00525     ires = 0;
00526     res = &(thee->resData[ires]);
00527     res->nAtomData = 1;
00528     strcpy(res->name, atoms[0].resName);
00529     for (i=1; i<natoms; i++) {
00530         if (Vstring_strcasecmp(atoms[i].resName, res->name) != 0) {
00531             (ires)++;
00532             res = &(thee->resData[ires]);
00533             res->nAtomData = 1;
00534             strcpy(res->name, atoms[i].resName);
00535         } else (res->nAtomData)++;
00536     }
00537
00538     /* Allocate per-residue space for atoms */
00539     for (ires=0; ires<thee->nResData; ires++) {
00540         res = &(thee->resData[ires]);
00541   res->atomData = Vmem_malloc(thee->vmem, res->nAtomData,
00542           sizeof(Vparam_AtomData));
00543     }
00544
00545     /* Copy atoms into residues */
00546     iatom = 0;
00547     Vparam_AtomData_copyTo(&(atoms[0]), &(res->atomData[iatom]));
00548     for (ires=0; ires<thee->nResData; ires++) {
00549         res = &(thee->resData[ires]);
00550         for (jatom=0; jatom<res->nAtomData; jatom++) {
00551             Vparam_AtomData_copyTo(&(atoms[iatom]), &(res->atomData[jatom]));
00552             iatom++;
00553         }
00554     }
00555
00556
00557     /* Shut down communication */
00558     Vio_acceptFree(sock);
00559     Vio_dtor(&sock);
00560
00561     /* Destroy temporary atom space */
00562     Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData), (void **)&(atoms));
00563
00564     return 1;
00565
00566 }
00567
00568 VEXTERNC void Vparam_AtomData_copyTo(Vparam_AtomData *thee,
00569   Vparam_AtomData *dest) {
00570
00571     VASSERT(thee != VNULL);
00572     VASSERT(dest != VNULL);
00573
00574     strcpy(dest->atomName, thee->atomName);
00575     strcpy(dest->resName, thee->resName);
00576     dest->charge = thee->charge;
00577     dest->radius = thee->radius;
00578     dest->epsilon = thee->epsilon;
00579
```

```
00580 }
00581
00582 VEXTERNC void Vparam_ResData_copyTo(Vparam_ResData *thee,
00583   Vparam_ResData *dest) {
00584
00585     int i;
00586
00587     VASSERT(thee != VNULL);
00588     VASSERT(dest != VNULL);
00589
00590     strcpy(dest->name, thee->name);
00591     dest->vmem = thee->vmem;
00592     dest->nAtomData = thee->nAtomData;
00593
00594     dest->atomData = Vmem_malloc(thee->vmem, dest->nAtomData,
00595                                  sizeof(Vparam_AtomData));
00596
00597     for (i=0; i<dest->nAtomData; i++) {
00598       Vparam_AtomData_copyTo(&(thee->atomData[i]), &(dest->atomData[i]));
00599     }
00600     Vmem_free(thee->vmem, thee->nAtomData, sizeof(Vparam_AtomData),
00601               (void **)&(thee->atomData));
00602 }
00603
00604 VEXTERNC void Vparam_AtomData_copyFrom(Vparam_AtomData *thee,
00605   Vparam_AtomData *src) {  Vparam_AtomData_copyTo(src, thee); }
00606
00607 VPRIVATE int readXMLFileAtom(Vio *sock, Vparam_AtomData *atom) {
00608
00609     double dtmp;
00610     char tok[VMAX_BUFSIZE];
00611     int chgflag, radflag, nameflag;
00612
00613     VASSERT(atom != VNULL);
00614
00615     if (Vio_scanf(sock, "%s", tok) != 1) return 0;
00616
00617     chgflag = 0;
00618     radflag = 0;
00619     nameflag = 0;
00620
00621     while (1)
00622       {
00623           if (Vstring_strcasecmp(tok, "name") == 0) {
00624               VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00625               if (strlen(tok) > VMAX_ARGLEN) {
00626                   Vnm_print(2, "Vparam_readXMLFileAtom:  string (%s) too long \
00627 (%d)!\n", tok, strlen(tok));
00628                   return 0;
00629               }
00630               nameflag = 1;
00631               strcpy(atom->atomName, tok);
00632           } else if (Vstring_strcasecmp(tok, "charge") == 0) {
00633               VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00634               if (sscanf(tok, "%lf", &dtmp) != 1) {
00635                   Vnm_print(2, "Vparam_readXMLFileAtom:  Unexpected token (%s) wh
      ile \
```

```
00636 parsing charge!\n", tok);
00637                     return 0;
00638                 }
00639                 chgflag = 1;
00640                 atom->charge = dtmp;
00641             }  else if (Vstring_strcasecmp(tok, "radius") == 0) {
00642                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00643                 if (sscanf(tok, "%lf", &dtmp) != 1) {
00644                     Vnm_print(2, "Vparam_readXMLFileAtom:  Unexpected token (%s) wh
      ile \
00645 parsing radius!\n", tok);
00646                     return 0;
00647                 }
00648                 radflag = 1;
00649                 atom->radius = dtmp;
00650             }  else if (Vstring_strcasecmp(tok, "epsilon") == 0) {
00651                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00652                 if (sscanf(tok, "%lf", &dtmp) != 1) {
00653                     Vnm_print(2, "Vparam_readXMLFileAtom:  Unexpected token (%s) wh
      ile \
00654 parsing epsilon!\n", tok);
00655                     return 0;
00656                 }
00657                 atom->epsilon = dtmp;
00658             } else if ((Vstring_strcasecmp(tok, "/atom") == 0) ||
00659                        (Vstring_strcasecmp(tok, "atom") == 0)){
00660                   if (chgflag && radflag && nameflag) return 1;
00661                   else if (!chgflag) {
00662                     Vnm_print(2, "Vparam_readXMLFileAtom: Reached end of atom witho
      ut \
00663 setting the charge!\n");
00664                     return 0;
00665                   } else if (!radflag) {
00666                     Vnm_print(2, "Vparam_readXMLFileAtom: Reached end of atom witho
      ut \
00667 setting the radius!\n");
00668                     return 0;
00669                   } else if (!nameflag) {
00670                     Vnm_print(2, "Vparam_readXMLFileAtom: Reached end of atom witho
      ut \
00671 setting the name!\n");
00672                     return 0;
00673                   }
00674             }
00675             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00676         }
00677 
00678     /* If we get here something wrong has happened */
00679 
00680     VJMPERR1(1);
00681 
00682 VERROR1:
00683     Vnm_print(2, "Vparam_readXMLFileAtom: Got unexpected EOF reading parameter fi
      le!\n");
00684     return 0;
00685 
00686 }
```

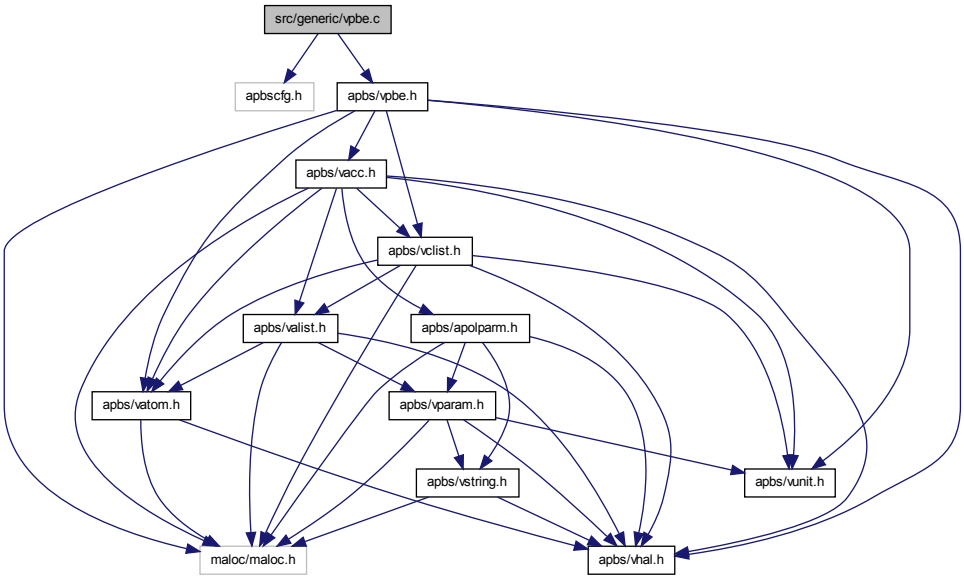
```
00687
00688 VPRIVATE int readFlatFileLine(Vio *sock, Vparam_AtomData *atom) {
00689
00690     double dtmp;
00691     char tok[VMAX_BUFSIZE];
00692
00693     VASSERT(atom != VNULL);
00694
00695     if (Vio_scanf(sock, "%s", tok) != 1) return 0;
00696     if (strlen(tok) > VMAX_ARGLEN) {
00697         Vnm_print(2, "Vparam_readFlatFile:  string (%s) too long (%d)!\n",
00698           tok, strlen(tok));
00699         return 0;
00700     }
00701     strcpy(atom->resName, tok);
00702     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00703     if (strlen(tok) > VMAX_ARGLEN) {
00704         Vnm_print(2, "Vparam_readFlatFile:  string (%s) too long (%d)!\n",
00705           tok, strlen(tok));
00706         return 0;
00707     }
00708     strcpy(atom->atomName, tok);
00709     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00710     if (sscanf(tok, "%lf", &dtmp) != 1) {
00711         Vnm_print(2, "Vparam_readFlatFile:  Unexpected token (%s) while \
00712 parsing charge!\n", tok);
00713         return 0;
00714     }
00715     atom->charge = dtmp;
00716     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00717     if (sscanf(tok, "%lf", &dtmp) != 1) {
00718         Vnm_print(2, "Vparam_readFlatFile:  Unexpected token (%s) while \
00719 parsing radius!\n", tok);
00720         return 0;
00721     }
00722     atom->radius = dtmp;
00723     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00724     if (sscanf(tok, "%lf", &dtmp) != 1) {
00725         Vnm_print(2, "Vparam_readFlatFile:  Unexpected token (%s) while \
00726 parsing radius!\n", tok);
00727         return 0;
00728     }
00729     atom->epsilon = dtmp;
00730
00731     return 1;
00732
00733 VERROR1:
00734     Vnm_print(2, "Vparam_readFlatFile: Got unexpected EOF reading parameter file!
      \n");
00735     return 0;
00736 }
```

10.75 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-Huckel parameter.
- VPUBLIC double Vpbe_getDeblen (Vpbe *thee)
  Get Debye-Huckel screening length.
- VPUBLIC double Vpbe_getZkappa2 (Vpbe *thee)
  Get modified squared Debye-Huckel 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 1552 2010-02-10 17:46:27Z yhuang01

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
* 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 Washington University in St. Louis 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 src/generic/vpbe.c

```
00001
00049 #include "apbscfg.h"
00050 #include "apbs/vpbe.h"
00051
00052 /* ///////////////////////////////////////////////////////////////////////////
00053 // Class Vpbe: Private method declaration
00055 #define MAX_SPLINE_WINDOW 0.5
00056
00057 /* ///////////////////////////////////////////////////////////////////////////
00058 // Class Vpbe: Inlineable methods
00060 #if !defined(VINLINE_VPBE)
00061
00062 VPUBLIC Valist* Vpbe_getValist(Vpbe *thee) {
00063
00064    VASSERT(thee != VNULL);
00065    return thee->alist;
00066
00067 }
00068
00069 VPUBLIC Vacc* Vpbe_getVacc(Vpbe *thee) {
00070
00071    VASSERT(thee != VNULL);
00072    VASSERT(thee->paramFlag);
00073    return thee->acc;
00074
00075 }
00076
00077 VPUBLIC double Vpbe_getBulkIonicStrength(Vpbe *thee) {
00078
00079    VASSERT(thee != VNULL);
00080    VASSERT(thee->paramFlag);
00081    return thee->bulkIonicStrength;
00082 }
00083
00084 VPUBLIC double Vpbe_getTemperature(Vpbe *thee) {
00085
00086    VASSERT(thee != VNULL);
00087    VASSERT(thee->paramFlag);
00088    return thee->T;
00089
```

```
00090 }
00091
00092 VPUBLIC double Vpbe_getSoluteDiel(Vpbe *thee) {
00093
00094    VASSERT(thee != VNULL);
00095    VASSERT(thee->paramFlag);
00096    return thee->soluteDiel;
00097
00098 }
00099
00100 VPUBLIC double* Vpbe_getSoluteCenter(Vpbe *thee) {
00101
00102    VASSERT(thee != VNULL);
00103    return thee->soluteCenter;
00104 }
00105
00106 VPUBLIC double Vpbe_getSolventDiel(Vpbe *thee) {
00107
00108    VASSERT(thee != VNULL);
00109    VASSERT(thee->paramFlag);
00110    return thee->solventDiel;
00111 }
00112
00113 VPUBLIC double Vpbe_getSolventRadius(Vpbe *thee) {
00114
00115    VASSERT(thee != VNULL);
00116    VASSERT(thee->paramFlag);
00117    return thee->solventRadius;
00118 }
00119
00120 VPUBLIC double Vpbe_getMaxIonRadius(Vpbe *thee) {
00121
00122    VASSERT(thee != VNULL);
00123    VASSERT(thee->paramFlag);
00124    return thee->maxIonRadius;
00125 }
00126
00127 VPUBLIC double Vpbe_getXkappa(Vpbe *thee) {
00128
00129    VASSERT(thee != VNULL);
00130    VASSERT(thee->paramFlag);
00131    return thee->xkappa;
00132 }
00133
00134 VPUBLIC double Vpbe_getDeblen(Vpbe *thee) {
00135
00136    VASSERT(thee != VNULL);
00137    VASSERT(thee->paramFlag);
00138    return thee->deblen;
00139 }
00140
00141 VPUBLIC double Vpbe_getZkappa2(Vpbe *thee) {
00142
00143    VASSERT(thee != VNULL);
00144    VASSERT(thee->paramFlag);
00145    return thee->zkappa2;
00146 }
```

```
00147
00148 VPUBLIC double Vpbe_getZmagic(Vpbe *thee) {
00149
00150    VASSERT(thee != VNULL);
00151    VASSERT(thee->paramFlag);
00152    return thee->zmagic;
00153 }
00154
00155 VPUBLIC double Vpbe_getSoluteRadius(Vpbe *thee) {
00156
00157    VASSERT(thee != VNULL);
00158    return thee->soluteRadius;
00159 }
00160
00161 VPUBLIC double Vpbe_getSoluteXlen(Vpbe *thee) {
00162
00163    VASSERT(thee != VNULL);
00164    return thee->soluteXlen;
00165 }
00166
00167 VPUBLIC double Vpbe_getSoluteYlen(Vpbe *thee) {
00168
00169    VASSERT(thee != VNULL);
00170    return thee->soluteYlen;
00171 }
00172
00173 VPUBLIC double Vpbe_getSoluteZlen(Vpbe *thee) {
00174
00175    VASSERT(thee != VNULL);
00176    return thee->soluteZlen;
00177 }
00178
00179 VPUBLIC double Vpbe_getSoluteCharge(Vpbe *thee) {
00180
00181    VASSERT(thee != VNULL);
00182    return thee->soluteCharge;
00183 }
00184
00185 /* ///////////////////////////////////////////////////////////////////////////
00186  // Routine:  Vpbe_getzmem
00187  // Purpose: This routine returns values stored in the structure thee.
00188  // Author:  Michael Grabe
00190 VPUBLIC double Vpbe_getzmem(Vpbe *thee) {
00191
00192  VASSERT(thee != VNULL);
00193  VASSERT(thee->param2Flag);
00194  return thee->z_mem;
00195 }
00196
00197 /* ///////////////////////////////////////////////////////////////////////////
00198  // Routine:  Vpbe_getLmem
00199  // Purpose: This routine returns values stored in the structure thee.
00200  // Author:  Michael Grabe
00202 VPUBLIC double Vpbe_getLmem(Vpbe *thee) {
00203
00204  VASSERT(thee != VNULL);
00205  VASSERT(thee->param2Flag);
```

```
00206  return thee->L;
00207 }
00208
00209 /* ///////////////////////////////////////////////////////////////////////////
00210  // Routine:  Vpbe_getmembraneDiel
00211  // Purpose: This routine returns values stored in the structure thee.
00212  // Author:  Michael Grabe
00214 VPUBLIC double Vpbe_getmembraneDiel(Vpbe *thee) {
00215
00216  VASSERT(thee != VNULL);
00217  VASSERT(thee->param2Flag);
00218  return thee->mmembraneDiel;
00219 }
00220
00221 /* ///////////////////////////////////////////////////////////////////////////
00222  // Routine:  Vpbe_getmemv
00223  // Purpose: This routine returns values stored in the structure thee.
00224  // Author:  Michael Grabe
00226 VPUBLIC double Vpbe_getmemv(Vpbe *thee) {
00227
00228  VASSERT(thee != VNULL);
00229  VASSERT(thee->param2Flag);
00230  return thee->V;
00231 }
00232
00233 #endif /* if !defined(VINLINE_VPBE) */
00234
00235 /* ///////////////////////////////////////////////////////////////////////////
00236 // Class Vpbe: Non-inlineable methods
00238
00239 VPUBLIC Vpbe* Vpbe_ctor(Valist *alist, int ionNum, double *ionConc,
00240       double *ionRadii, double *ionQ, double T,
00241       double soluteDiel, double solventDiel,
00242       double solventRadius, int focusFlag, double sdens,
00243       double z_mem, double L, double membraneDiel, double V ) {
00244
00245     /* Set up the structure */
00246     Vpbe *thee = VNULL;
00247     thee = Vmem_malloc(VNULL, 1, sizeof(Vpbe) );
00248     VASSERT( thee != VNULL);
00249     VASSERT( Vpbe_ctor2(thee, alist, ionNum, ionConc, ionRadii, ionQ,
00250       T, soluteDiel, solventDiel, solventRadius, focusFlag, sdens,
00251       z_mem, L, membraneDiel, V) );
00252
00253     return thee;
00254 }
00255
00256
00257 VPUBLIC int Vpbe_ctor2(Vpbe *thee, Valist *alist, int ionNum,
00258         double *ionConc, double *ionRadii,
00259         double *ionQ, double T, double soluteDiel,
00260         double solventDiel, double solventRadius, int focusFlag,
00261         double sdens, double z_mem, double L, double membraneDiel,
00262         double V) {
00263
00264     int i, iatom, inhash[3];
00265     double atomRadius;
```

```
00266     Vatom *atom;
00267     double center[3] = {0.0, 0.0, 0.0};
00268     double lower_corner[3] = {0.0, 0.0, 0.0};
00269     double upper_corner[3] = {0.0, 0.0, 0.0};
00270     double disp[3], dist, radius, charge, xmin, xmax, ymin, ymax, zmin, zmax;
00271     double x, y, z, netCharge;
00272     double nhash[3];
00273     const double N_A = 6.022045000e+23;
00274     const double e_c = 4.803242384e-10;
00275     const double k_B = 1.380662000e-16;
00276     const double pi  = 4. * VATAN(1.);
00277
00278     /* Set up memory management object */
00279     thee->vmem = Vmem_ctor("APBS::VPBE");
00280
00281     VASSERT(thee != VNULL);
00282     if (alist == VNULL) {
00283         Vnm_print(2, "Vpbe_ctor2: Got null pointer to Valist object!\n");
00284         return 0;
00285     }
00286
00287     /* **** STUFF THAT GETS DONE FOR EVERYONE **** */
00288     /* Set pointers */
00289     thee->alist = alist;
00290     thee->paramFlag = 0;
00291
00292     /* Determine solute center */
00293     center[0] = thee->alist->center[0];
00294     center[1] = thee->alist->center[1];
00295     center[2] = thee->alist->center[2];
00296     thee->soluteCenter[0] = center[0];
00297     thee->soluteCenter[1] = center[1];
00298     thee->soluteCenter[2] = center[2];
00299
00300     /* Determine solute length and charge*/
00301     radius = 0;
00302     atom = Valist_getAtom(thee->alist, 0);
00303     xmin = Vatom_getPosition(atom)[0];
00304     xmax = Vatom_getPosition(atom)[0];
00305     ymin = Vatom_getPosition(atom)[1];
00306     ymax = Vatom_getPosition(atom)[1];
00307     zmin = Vatom_getPosition(atom)[2];
00308     zmax = Vatom_getPosition(atom)[2];
00309     charge = 0;
00310     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
00311         atom = Valist_getAtom(thee->alist, iatom);
00312         atomRadius = Vatom_getRadius(atom);
00313         x = Vatom_getPosition(atom)[0];
00314         y = Vatom_getPosition(atom)[1];
00315         z = Vatom_getPosition(atom)[2];
00316         if ((x+atomRadius) > xmax) xmax = x + atomRadius;
00317         if ((x-atomRadius) < xmin) xmin = x - atomRadius;
00318         if ((y+atomRadius) > ymax) ymax = y + atomRadius;
00319         if ((y-atomRadius) < ymin) ymin = y - atomRadius;
00320         if ((z+atomRadius) > zmax) zmax = z + atomRadius;
00321         if ((z-atomRadius) < zmin) zmin = z - atomRadius;
00322         disp[0] = (x - center[0]);
```

```
00323         disp[1] = (y - center[1]);
00324         disp[2] = (z - center[2]);
00325         dist = (disp[0]*disp[0]) + (disp[1]*disp[1]) + (disp[2]*disp[2]);
00326         dist = VSQRT(dist) + atomRadius;
00327         if (dist > radius) radius = dist;
00328         charge += Vatom_getCharge(Valist_getAtom(thee->alist, iatom));
00329     }
00330     thee->soluteRadius = radius;
00331     Vnm_print(0, "Vpbe_ctor2:  solute radius = %g\n", radius);
00332     thee->soluteXlen = xmax - xmin;
00333     thee->soluteYlen = ymax - ymin;
00334     thee->soluteZlen = zmax - zmin;
00335     Vnm_print(0, "Vpbe_ctor2:  solute dimensions = %g x %g x %g\n",
00336             thee->soluteXlen, thee->soluteYlen, thee->soluteZlen);
00337     thee->soluteCharge = charge;
00338     Vnm_print(0, "Vpbe_ctor2:  solute charge = %g\n", charge);
00339 
00340     /* Set parameters */
00341     thee->numIon = ionNum;
00342     if (thee->numIon >= MAXION) {
00343         Vnm_print(2, "Vpbe_ctor2:  Too many ion species (MAX = %d)!\n",
00344           MAXION);
00345         return 0;
00346     }
00347     thee->bulkIonicStrength = 0.0;
00348     thee->maxIonRadius = 0.0;
00349     netCharge = 0.0;
00350     for (i=0; i<thee->numIon; i++) {
00351         thee->ionConc[i] = ionConc[i];
00352         thee->ionRadii[i] = ionRadii[i];
00353         if (ionRadii[i] > thee->maxIonRadius) thee->maxIonRadius = ionRadii[i];
00354         thee->ionQ[i] = ionQ[i];
00355         thee->bulkIonicStrength += (0.5*ionConc[i]*VSQR(ionQ[i]));
00356         netCharge += (ionConc[i]*ionQ[i]);
00357     }
00358 #ifndef VAPBSQUIET
00359     Vnm_print(1, "  Vpbe_ctor:  Using max ion radius (%g A) for exclusion \
00360 function\n", thee->maxIonRadius);
00361 #endif
00362     if (VABS(netCharge) > VSMALL) {
00363         Vnm_print(2, "Vpbe_ctor2:  You have a counterion charge imbalance!\n");
00364         Vnm_print(2, "Vpbe_ctor2:  Net charge conc. = %g M\n", netCharge);
00365         return 0;
00366     }
00367     thee->T = T;
00368     thee->soluteDiel = soluteDiel;
00369     thee->solventDiel = solventDiel;
00370     thee->solventRadius = solventRadius;
00371 
00372     /* Compute parameters:
00373      *
00374      * kappa^2 = (8 pi N_A e_c^2) I_s / (1000 eps_w k_B T)
00375      * kappa   = 0.325567 * I_s^{1/2}   angstroms^{-1}
00376      * deblen  = 1 / kappa
00377      *         = 3.071564378 * I_s^{1/2}   angstroms
00378      * \bar{kappa}^2 = eps_w * kappa^2
00379      * zmagic  = (4 * pi * e_c^2) / (k_B T)   (we scale the diagonal later)
```

```
00380      *         = 7046.528838
00381      */
00382     if (thee->T == 0.0) {
00383         Vnm_print(2, "Vpbe_ctor2:  You set the temperature to 0 K.\n");
00384         Vnm_print(2, "Vpbe_ctor2:  That violates the 3rd Law of Thermo.!");
00385         return 0;
00386     }
00387     if (thee->bulkIonicStrength == 0.) {
00388         thee->xkappa  = 0.;
00389         thee->deblen  = 0.;
00390         thee->zkappa2 = 0.;
00391     } else {
00392         thee->xkappa  = VSQRT( thee->bulkIonicStrength * 1.0e-16 *
00393             ((8.0 * pi * N_A * e_c*e_c) /
00394             (1000.0 * thee->solventDiel * k_B * T))
00395         );
00396         thee->deblen  = 1. / thee->xkappa;
00397         thee->zkappa2 = thee->solventDiel * VSQR(thee->xkappa);
00398     }
00399     Vnm_print(0, "Vpbe_ctor2:  bulk ionic strength = %g\n",
00400             thee->bulkIonicStrength);
00401     Vnm_print(0, "Vpbe_ctor2:  xkappa = %g\n", thee->xkappa);
00402     Vnm_print(0, "Vpbe_ctor2:  Debye length = %g\n", thee->deblen);
00403     Vnm_print(0, "Vpbe_ctor2:  zkappa2 = %g\n", thee->zkappa2);
00404     thee->zmagic  = ((4.0 * pi * e_c*e_c) / (k_B * thee->T)) * 1.0e+8;
00405     Vnm_print(0, "Vpbe_ctor2:  zmagic = %g\n", thee->zmagic);
00406
00407     /* Compute accessibility objects:
00408      *   - Allow for extra room in the case of spline windowing
00409      *   - Place some limits on the size of the hash table in the case of very
00410      *     large molecules
00411      */
00412     if (thee->maxIonRadius > thee->solventRadius)
00413         radius = thee->maxIonRadius + MAX_SPLINE_WINDOW;
00414     else radius = thee->solventRadius + MAX_SPLINE_WINDOW;
00415
00416  nhash[0] = (thee->soluteXlen)/0.5;
00417  nhash[1] = (thee->soluteYlen)/0.5;
00418  nhash[2] = (thee->soluteZlen)/0.5;
00419     for (i=0; i<3; i++) inhash[i] = (int)(nhash[i]);
00420
00421  for (i=0;i<3;i++){
00422         if (inhash[i] < 3) inhash[i] = 3;
00423         if (inhash[i] > MAX_HASH_DIM) inhash[i] = MAX_HASH_DIM;
00424  }
00425     Vnm_print(0, "Vpbe_ctor2:  Constructing Vclist with %d x %d x %d table\n",
00426             inhash[0], inhash[1], inhash[2]);
00427
00428     thee->clist = Vclist_ctor(thee->alist, radius, inhash,
00429             CLIST_AUTO_DOMAIN, lower_corner, upper_corner);
00430
00431     VASSERT(thee->clist != VNULL);
00432     thee->acc = Vacc_ctor(thee->alist, thee->clist, sdens);
00433
00434     VASSERT(thee->acc != VNULL);
00435
00436  /* SMPBE Added */
```

```
00437  thee->smsize = 0.0;
00438  thee->smvolume = 0.0;
00439  thee->ipkey = 0;
00440
00441     thee->paramFlag = 1;
00442
00443  /*-----------------------------------------------------------*/
00444  /* added by Michael Grabe                                    */
00445  /*-----------------------------------------------------------*/
00446
00447     thee->z_mem = z_mem;
00448     thee->L = L;
00449     thee->membraneDiel = membraneDiel;
00450     thee->V = V;
00451
00452  //    if (V != VNULL) thee->param2Flag = 1;
00453  //    else thee->param2Flag = 0;
00454
00455  /*-----------------------------------------------------------*/
00456
00457     return 1;
00458 }
00459
00460 VPUBLIC void Vpbe_dtor(Vpbe **thee) {
00461     if ((*thee) != VNULL) {
00462         Vpbe_dtor2(*thee);
00463         Vmem_free(VNULL, 1, sizeof(Vpbe), (void **)thee);
00464         (*thee) = VNULL;
00465     }
00466 }
00467
00468 VPUBLIC void Vpbe_dtor2(Vpbe *thee) {
00469     Vclist_dtor(&(thee->clist));
00470     Vacc_dtor(&(thee->acc));
00471     Vmem_dtor(&(thee->vmem));
00472 }
00473
00474 VPUBLIC double Vpbe_getCoulombEnergy1(Vpbe *thee) {
00475
00476     int i, j, k, natoms;
00477
00478     double dist, *ipos, *jpos, icharge, jcharge;
00479     double energy = 0.0;
00480     double eps, T;
00481     Vatom *iatom, *jatom;
00482     Valist *alist;
00483
00484     VASSERT(thee != VNULL);
00485     alist = Vpbe_getValist(thee);
00486     VASSERT(alist != VNULL);
00487     natoms = Valist_getNumberAtoms(alist);
00488
00489     /* Do the sum */
00490     for (i=0; i<natoms; i++) {
00491         iatom = Valist_getAtom(alist,i);
00492         icharge = Vatom_getCharge(iatom);
00493         ipos = Vatom_getPosition(iatom);
```

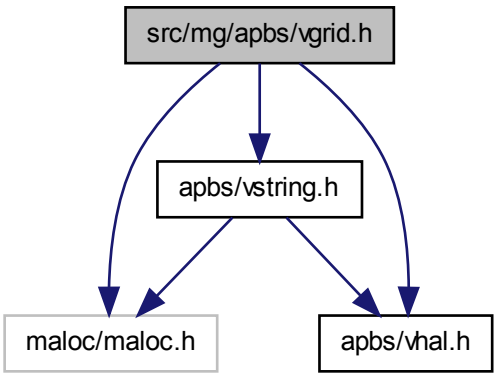
```
00494         for (j=i+1; j<natoms; j++) {
00495             jatom = Valist_getAtom(alist,j);
00496             jcharge = Vatom_getCharge(jatom);
00497             jpos = Vatom_getPosition(jatom);
00498             dist = 0;
00499             for (k=0; k<3; k++) dist += ((ipos[k]-jpos[k])*(ipos[k]-jpos[k]));
00500             dist = VSQRT(dist);
00501             energy = energy + icharge*jcharge/dist;
00502         }
00503     }
00504 
00505     /* Convert the result to J */
00506     T = Vpbe_getTemperature(thee);
00507     eps = Vpbe_getSoluteDiel(thee);
00508     energy = energy*Vunit_ec*Vunit_ec/(4*Vunit_pi*Vunit_eps0*eps*(1.0e-10));
00509 
00510     /* Scale by Boltzmann energy */
00511     energy = energy/(Vunit_kb*T);
00512 
00513     return energy;
00514 }
00515 
00516 VPUBLIC unsigned long int Vpbe_memChk(Vpbe *thee) {
00517 
00518     unsigned long int memUse = 0;
00519 
00520     if (thee == VNULL) return 0;
00521 
00522     memUse = memUse + sizeof(Vpbe);
00523     memUse = memUse + (unsigned long int)Vacc_memChk(thee->acc);
00524 
00525     return memUse;
00526 }
00527 
00528 VPUBLIC int Vpbe_getIons(Vpbe *thee, int *nion, double ionConc[MAXION],
00529   double ionRadii[MAXION], double ionQ[MAXION]) {
00530 
00531     int i;
00532 
00533     VASSERT(thee != VNULL);
00534 
00535     *nion = thee->numIon;
00536     for (i=0; i<(*nion); i++) {
00537         ionConc[i] = thee->ionConc[i];
00538         ionRadii[i] = thee->ionRadii[i];
00539         ionQ[i] = thee->ionQ[i];
00540     }
00541 
00542     return *nion;
00543 }
```

10.77 src/mg/apbs/vgrid.h File Reference

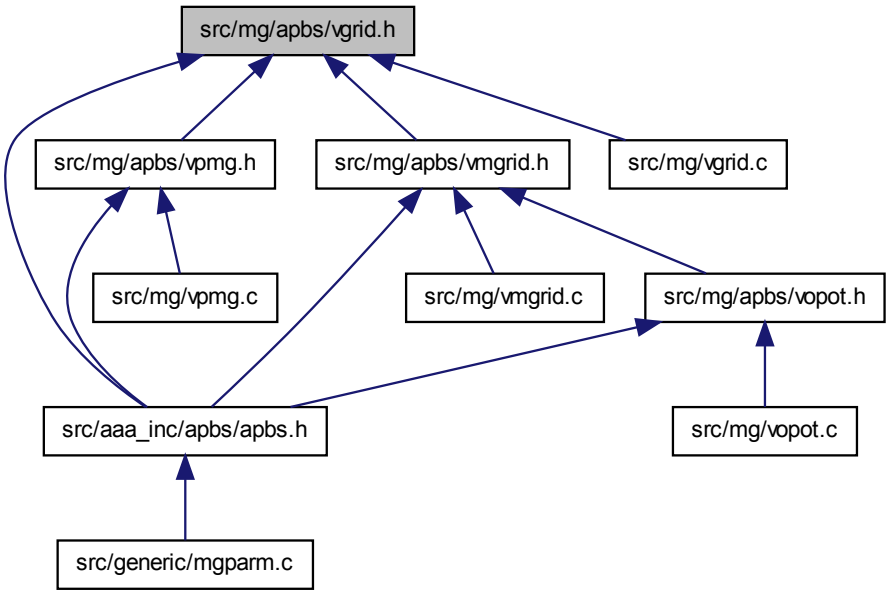
Potential oracle for Cartesian mesh data.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/vstring.h"
```

Include dependency graph for vgrid.h:



This graph shows which files directly or indirectly include this file:



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:

$$\|u\|_{H_1} = \left(\int_\Omega |\nabla u(x)|^2 dx + \int_\Omega |u(x)|^2 dx \right)^{1/2}$$

  .

10.77.1 Detailed Description

Potential oracle for Cartesian mesh data.

Author

Nathan Baker and Steve Bond

Version

Id:

vgrid.h 1609 2010-10-01 18:40:36Z sobolevnrm

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
* 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 Washington University in St. Louis 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.

10.77.2 Function Documentation

10.77.2.1 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

thee	Object to hold new grid data
iodev	I/O device
iofmt	I/O format
thost	Remote host name
fname	File name
title	Data title
pvec	Masking vector (0 = not written)

Definition at line 779 of file vgrid.c.

10.78 src/mg/apbs/vgrid.h

```
00001
00055 #ifndef _VGRID_H_
00056 #define _VGRID_H_
00057
00058 #include "maloc/maloc.h"
00059 #include "apbs/vhal.h"
00060 #include "apbs/vstring.h"
00061
00062
00065 #define VGRID_DIGITS 6
00066
00072 struct sVgrid {
00073
00074     int nx;
00075     int ny;
00076     int nz;
00077     double hx;
00078     double hy;
00079     double hzed;
00080     double xmin;
00081     double ymin;
```

```
00082     double zmin;
00083     double xmax;
00084     double ymax;
00085     double zmax;
00086     double *data;
00087     int readdata;
00088     int ctordata;
00090     Vmem *mem;
00091 };
00092
00097 typedef struct sVgrid Vgrid;
00098
00099 #if !defined(VINLINE_VGRID)
00100
00108     VEXTERNC unsigned long int Vgrid_memChk(Vgrid *thee);
00109
00110 #else /* if defined(VINLINE_VGRID) */
00111
00119 #   define Vgrid_memChk(thee) (Vmem_bytes((thee)->vmem))
00120
00121 #endif /* if !defined(VINLINE_VPMG) */
00122
00140 VEXTERNC Vgrid*  Vgrid_ctor(int nx, int ny, int nz,
00141                             double hx, double hy, double hzed,
00142                             double xmin, double ymin, double zmin,
00143                             double *data);
00144
00163 VEXTERNC int Vgrid_ctor2(Vgrid *thee, int nx, int ny, int nz,
00164                             double hx, double hy, double hzed,
00165                             double xmin, double ymin, double zmin,
00166                             double *data);
00167
00176 VEXTERNC int Vgrid_value(Vgrid *thee, double x[3], double *value);
00177
00183 VEXTERNC void Vgrid_dtor(Vgrid **thee);
00184
00190 VEXTERNC void Vgrid_dtor2(Vgrid *thee);
00191
00205 VEXTERNC int Vgrid_curvature(Vgrid *thee, double pt[3], int cflag,
00206   double *curv);
00207
00216 VEXTERNC int Vgrid_gradient(Vgrid *thee, double pt[3], double grad[3] );
00217
00222 VEXTERNC int Vgrid_readGZ(
00223         Vgrid *thee,
00224         const char *fname
00225         );
00226
00230 VEXTERNC void Vgrid_writeGZ(
00231        Vgrid *thee,
00232        const char *iodev,
00233        const char *iofmt,
00234        const char *thost,
00235        const char *fname,
00236        char *title,
00237        double *pvec
00238        );
```

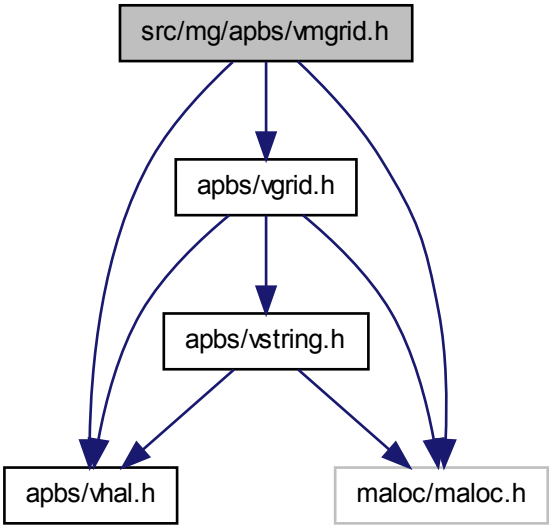
```
00239
00257 VEXTERNC void Vgrid_writeUHBD(Vgrid *thee, const char *iodev,
00258   const char *iofmt, const char *thost, const char *fname, char *title,
00259   double *pvec);
00260
00275 VEXTERNC void Vgrid_writeDX(Vgrid *thee, const char *iodev,
00276   const char *iofmt,  const char *thost, const char *fname, char *title,
00277   double *pvec);
00278
00290 VEXTERNC int Vgrid_readDX(Vgrid *thee, const char *iodev, const char *iofmt,
00291   const char *thost, const char *fname);
00292
00299 VEXTERNC double Vgrid_integrate(Vgrid *thee);
00300
00309 VEXTERNC double Vgrid_normL1(Vgrid *thee);
00310
00319 VEXTERNC double Vgrid_normL2(Vgrid *thee);
00320
00329 VEXTERNC double Vgrid_normLinf(Vgrid *thee);
00330
00340 VEXTERNC double Vgrid_seminormH1(Vgrid *thee);
00341
00352 VEXTERNC double Vgrid_normH1(Vgrid *thee);
00353
00354 #endif
```

10.79 src/mg/apbs/vmgrid.h File Reference

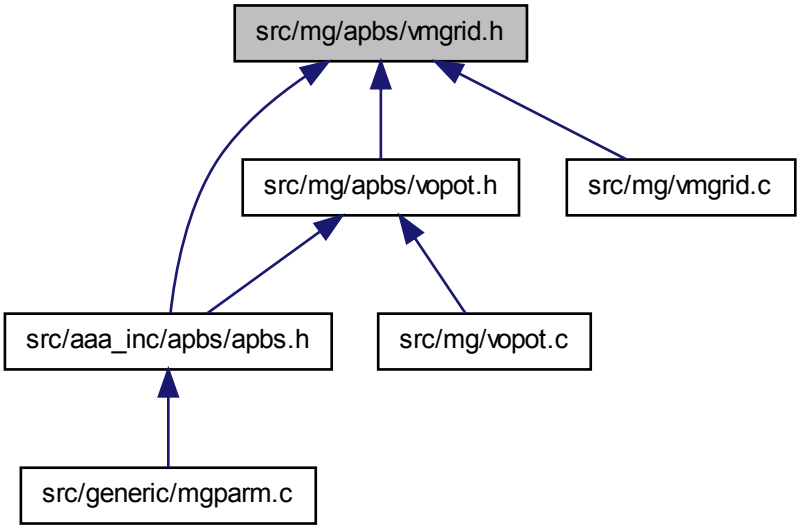
Multiresolution oracle for Cartesian mesh data.

```
#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

  Multiresoltion oracle for Cartesian mesh data.

Defines

- #define VMGRIDMAX 20

  The maximum number of levels in the grid hiearchy.

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 hiearchy.
- VEXTERNC Vgrid * Vmgrid_getGridByPoint (Vmgrid *thee, double pt[3])
  Get grid in hiearchy which contains specified point or VNULL.

10.79.1 Detailed Description

Multiresolution oracle for Cartesian mesh data.

Author

Nathan Baker

Version

Id:

vmgrid.h 1565 2010-03-07 16:06:27Z sobolevnrm

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
* 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 Washington University in St. Louis 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.

10.80 src/mg/apbs/vmgrid.h

```
00001
```

```
00054 #ifndef _VMGRID_H_
00055 #define _VMGRID_H_
00056
00057 /* Generic headers */
00058 #include "maloc/maloc.h"
00059 #include "apbs/vhal.h"
00060
00061 /* Headers specific to this file */
00062 #include "apbs/vgrid.h"
00063
00068 #define VMGRIDMAX 20
00069
00070
00076 struct sVmgrid {
00077
00078     int ngrids;
00079     Vgrid *grids[VMGRIDMAX];
00084 };
00085
00090 typedef struct sVmgrid Vmgrid;
00091
00097 VEXTERNC Vmgrid*  Vmgrid_ctor();
00098
00105 VEXTERNC int Vmgrid_ctor2(Vmgrid *thee);
00106
00115 VEXTERNC int Vmgrid_value(Vmgrid *thee, double x[3], double *value);
00116
00122 VEXTERNC void Vmgrid_dtor(Vmgrid **thee);
00123
00129 VEXTERNC void Vmgrid_dtor2(Vmgrid *thee);
00130
00143 VEXTERNC int Vmgrid_addGrid(Vmgrid *thee, Vgrid *grid);
00144
00145
00159 VEXTERNC int Vmgrid_curvature(Vmgrid *thee, double pt[3], int cflag,
00160   double *curv);
00161
00170 VEXTERNC int Vmgrid_gradient(Vmgrid *thee, double pt[3], double grad[3] );
00171
00179 VEXTERNC Vgrid* Vmgrid_getGridByNum(Vmgrid *thee, int num);
00180
00188 VEXTERNC Vgrid* Vmgrid_getGridByPoint(Vmgrid *thee, double pt[3]);
00189
00190 #endif
00191
```

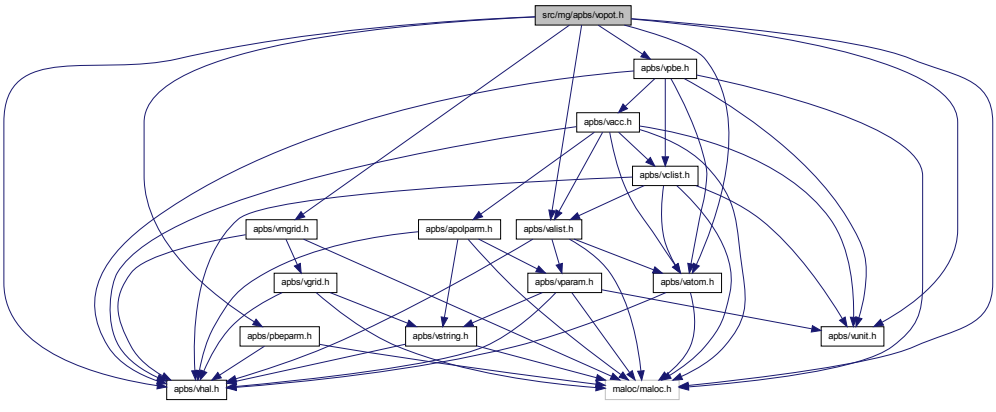
10.81 src/mg/apbs/vopot.h File Reference

Potential oracle for Cartesian mesh data.

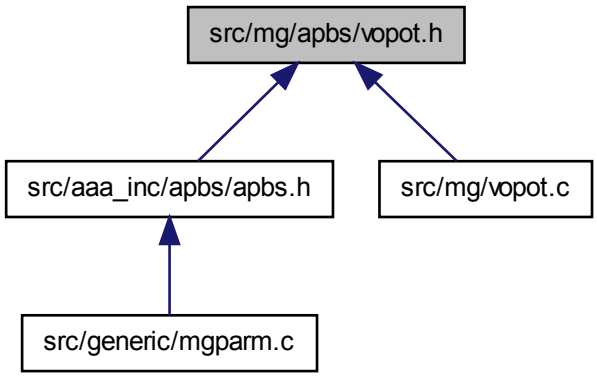
```
#include "maloc/maloc.h"

#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.

10.81.1 Detailed Description

Potential oracle for Cartesian mesh data.

Author

Nathan Baker

Version

Id:

vopot.h 1552 2010-02-10 17:46:27Z yhuang01

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
* 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 Washington University in St. Louis 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.

10.82 src/mg/apbs/vopot.h

```
00001
00054 #ifndef _VOPOT_H_
00055 #define _VOPOT_H_
00056
00057 /* Generic headers */
00058 #include "maloc/maloc.h"
00059 #include "apbs/vhal.h"
00060
00061 /* Specific headers */
00062 #include "apbs/vatom.h"
00063 #include "apbs/valist.h"
00064 #include "apbs/vmgrid.h"
00065 #include "apbs/vunit.h"
00066 #include "apbs/vpbe.h"
00067 #include "apbs/pbeparm.h"
00068
00074 struct sVopot {
00075
00076     Vmgrid *mgrid;
00078     Vpbe   *pbe;
00079     Vbcfl bcfl;
00081 };
00082
```

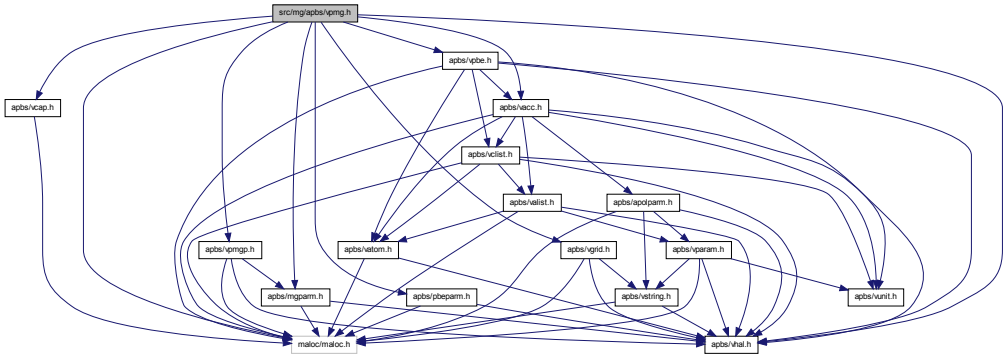
```
00087 typedef struct sVopot Vopot;
00088
00099 VEXTERNC Vopot*  Vopot_ctor(Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl);
00100
00112 VEXTERNC int Vopot_ctor2(Vopot *thee, Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl);
00113
00122 VEXTERNC int Vopot_pot(Vopot *thee, double x[3], double *pot);
00123
00129 VEXTERNC void Vopot_dtor(Vopot **thee);
00130
00136 VEXTERNC void Vopot_dtor2(Vopot *thee);
00137
00151 VEXTERNC int Vopot_curvature(Vopot *thee, double pt[3], int cflag, double
00152   *curv);
00153
00162 VEXTERNC int Vopot_gradient(Vopot *thee, double pt[3], double grad[3] );
00163
00164
00165 #endif
```

10.83 src/mg/apbs/vpmg.h File Reference

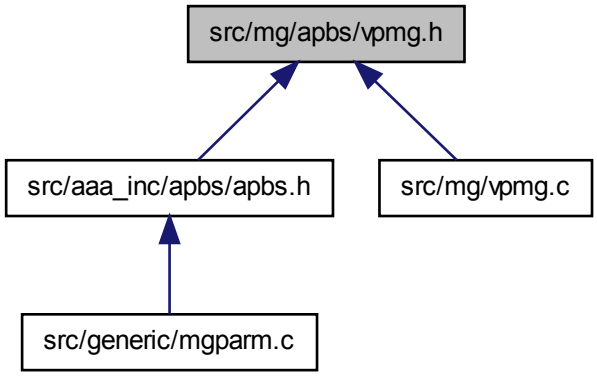
Contains declarations for class Vpmg.

```
#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, 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 *nlInduced, 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 dipooles 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 *nlInduced, 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 dipooles 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 *nlInduced, 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.

- VEXTERNC void Vpmg_qfMutualPolForce (Vpmg *thee, Vgrid *induced, Vgrid *nlInduced, 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 *nlInduced, 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 *nlInduced, 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.

10.83.1 Detailed Description

Contains declarations for class Vpmg.

Version

Id:

vpmg.h 1578 2010-03-19 22:13:30Z sdg0919

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
* 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 Washington University in St. Louis 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.

10.84 src/mg/apbs/vpmg.h

```
00001
00057 #ifndef _VPMG_H_
00058 #define _VPMG_H_
00059
00060 /* Generic headers */
00061 #include "maloc/maloc.h"
00062 #include "apbs/vhal.h"
00063
00064 /* Headers specific to this file */
00065 #include "apbs/vpmgp.h"
00066 #include "apbs/vacc.h"
00067 #include "apbs/vcap.h"
00068 #include "apbs/vpbe.h"
00069 #include "apbs/vgrid.h"
00070 #include "apbs/mgparm.h"
00071 #include "apbs/pbeparm.h"
00072
00077 #define VPMGMAXPART 2000
00078
00088 struct sVpmg {
00089
00090   Vmem *vmem;
00091   Vpmgp *pmgp;
00092   Vpbe *pbe;
00094   double *epsx;
00095   double *epsy;
00096   double *epsz;
00097   double *kappa;
00098   double *pot;
00099   double *charge;
00101   int *iparm;
00102   double *rparm;
00103   int *iwork;
00104   double *rwork;
00105   double *a1cf;
00107   double *a2cf;
00109   double *a3cf;
00111   double *ccf;
00112   double *fcf;
00113   double *tcf;
00114   double *u;
00115   double *xf;
00116   double *yf;
00117   double *zf;
00118   double *gxcf;
00119   double *gycf;
00120   double *gzcf;
00121   double *pvec;
00122   double extDiEnergy;
00124   double extQmEnergy;
```

```
00126   double extQfEnergy;
00128   double extNpEnergy;
00130   Vsurf_Meth surfMeth;
00131   double splineWin;
00132   Vchrg_Meth chargeMeth;
00133   Vchrg_Src chargeSrc;
00135   int filled;
00137   int useDielXMap;
00139   Vgrid *dielXMap;
00140   int useDielYMap;
00142   Vgrid *dielYMap;
00143   int useDielZMap;
00145   Vgrid *dielZMap;
00146   int useKappaMap;
00148   Vgrid *kappaMap;
00149   int usePotMap;
00151   Vgrid *potMap;
00153   int useChargeMap;
00155   Vgrid *chargeMap;
00156 };
00157
00162 typedef struct sVpmg Vpmg;
00163
00164 /* ////////////////////////////////////////////////////////////////////////
00167 #if !defined(VINLINE_VPMG)
00168
00175     VEXTERNC unsigned long int Vpmg_memChk(
00176             Vpmg *thee
00177             );
00178
00179 #else /* if defined(VINLINE_VPMG) */
00180
00181 #   define Vpmg_memChk(thee) (Vmem_bytes((thee)->vmem))
00182
00183 #endif /* if !defined(VINLINE_VPMG) */
00184
00185 /* ////////////////////////////////////////////////////////////////////////
00188
00193 VEXTERNC Vpmg* Vpmg_ctor(
00194         Vpmgp *parms,
00195         Vpbe *pbe,
00196         int focusFlag,
00197         Vpmg *pmgOLD,
00198         MGparm *mgparm,
00199         PBEparm_calcEnergy energyFlag
00200         );
00201
00209 VEXTERNC int Vpmg_ctor2(
00210         Vpmg *thee,
00211         Vpmgp *parms,
00212         Vpbe *pbe,
00213         int focusFlag,
00214         Vpmg *pmgOLD,
00216         MGparm *mgparm,
00218         PBEparm_calcEnergy energyFlag
00221         );
00222
```

```
00227 VEXTERNC void Vpmg_dtor(
00228         Vpmg **thee
00230         );
00231
00236 VEXTERNC void Vpmg_dtor2(
00237         Vpmg *thee
00238         );
00239
00245 VEXTERNC int Vpmg_fillco(
00246         Vpmg *thee,
00247         Vsurf_Meth surfMeth,
00248         double splineWin,
00250         Vchrg_Meth chargeMeth,
00251         int useDielXMap,
00252         Vgrid *dielXMap,
00253         int useDielYMap,
00254         Vgrid *dielYMap,
00255         int useDielZMap,
00256         Vgrid *dielZMap,
00257         int useKappaMap,
00258         Vgrid *kappaMap,
00259         int usePotMap,
00260   Vgrid *potMap,
00261         int useChargeMap,
00262         Vgrid *chargeMap
00263         );
00264
00270 VEXTERNC int Vpmg_solve(
00271         Vpmg *thee
00272         );
00273
00285 VEXTERNC int Vpmg_solveLaplace(
00286         Vpmg *thee
00287         );
00288
00298 VEXTERNC double Vpmg_energy(
00299         Vpmg *thee,
00300         int extFlag
00304         );
00305
00323 VEXTERNC double Vpmg_qfEnergy(
00324         Vpmg *thee,
00325         int extFlag
00329         );
00330
00350 VEXTERNC double Vpmg_qfAtomEnergy(
00351         Vpmg *thee,
00352         Vatom *atom
00353         );
00354
00379 VEXTERNC double Vpmg_qmEnergy(
00380         Vpmg *thee,
00381         int extFlag
00385         );
00386
00387
00406 VEXTERNC double Vpmg_dielEnergy(
```

```
00407         Vpmg *thee,
00408         int extFlag
00412         );
00413
00414
00431 VEXTERNC double Vpmg_dielGradNorm(
00432         Vpmg *thee
00433         );
00434
00446 VEXTERNC int Vpmg_force(
00447         Vpmg *thee,
00448         double *force,
00450         int atomID,
00451         Vsurf_Meth srfm,
00452         Vchrg_Meth chgm
00453         );
00454
00466 VEXTERNC int Vpmg_qfForce(
00467         Vpmg *thee,
00468         double *force,
00470         int atomID,
00471         Vchrg_Meth chgm
00472         );
00473
00485 VEXTERNC int Vpmg_dbForce(
00486         Vpmg *thee,
00487         double *dbForce,
00489         int atomID,
00490         Vsurf_Meth srfm
00491         );
00492
00504 VEXTERNC int Vpmg_ibForce(
00505         Vpmg *thee,
00506         double *force,
00508         int atomID,
00509         Vsurf_Meth srfm
00510         );
00511
00517 VEXTERNC void Vpmg_setPart(
00518         Vpmg *thee,
00519         double lowerCorner[3],
00520         double upperCorner[3],
00521         int bflags[6]
00525         );
00526
00531 VEXTERNC void Vpmg_unsetPart(
00532         Vpmg *thee
00533         );
00534
00540 VEXTERNC int Vpmg_fillArray(
00541         Vpmg *thee,
00542         double *vec,
00544         Vdata_Type type,
00545         double parm,
00546         Vhal_PBEType pbetype,
00547   PBEparm * pbeparm
00548         );
```

```
00549
00555 VPUBLIC void Vpmg_fieldSpline4(
00556             Vpmg *thee,
00557             int atomID,
00558             double field[3]
00559             );
00560
00568 VEXTERNC double Vpmg_qfPermanentMultipoleEnergy(
00569             Vpmg *thee,
00570             int atomID
00571             );
00572
00578 VEXTERNC void Vpmg_qfPermanentMultipoleForce(
00579             Vpmg *thee,
00580             int atomID,
00581             double force[3],
00582             double torque[3]
00583             );
00584
00589 VEXTERNC void Vpmg_ibPermanentMultipoleForce(
00590             Vpmg *thee,
00591             int atomID,
00592             double force[3]
00593             );
00594
00599 VEXTERNC void Vpmg_dbPermanentMultipoleForce(
00600             Vpmg *thee,
00601             int atomID,
00602             double force[3]
00603             );
00604
00611 VEXTERNC void Vpmg_qfDirectPolForce(
00612             Vpmg *thee,
00613             Vgrid *perm,
00614             Vgrid *induced,
00615             int atomID,
00616             double force[3],
00617             double torque[3]
00618             );
00619
00628 VEXTERNC void Vpmg_qfNLDirectPolForce(
00629             Vpmg *thee,
00630             Vgrid *perm,
00631             Vgrid *nlInduced,
00632             int atomID,
00633             double force[3],
00634             double torque[3]
00635             );
00636
00644 VEXTERNC void Vpmg_ibDirectPolForce(
00645             Vpmg *thee,
00646             Vgrid *perm,
00647             Vgrid *induced,
00648             int atomID,
00649             double force[3]
00650             );
00651
```

```
00660 VEXTERNC void Vpmg_ibNLDirectPolForce(
00661              Vpmg *thee,
00662              Vgrid *perm,
00663              Vgrid *nlInduced,
00664              int atomID,
00665              double force[3]
00666              );
00667
00675 VEXTERNC void Vpmg_dbDirectPolForce(
00676              Vpmg *thee,
00677              Vgrid *perm,
00678              Vgrid *induced,
00679              int atomID,
00680              double force[3]
00681              );
00682
00691 VEXTERNC void Vpmg_dbNLDirectPolForce(
00692              Vpmg *thee,
00693              Vgrid *perm,
00694              Vgrid *nlInduced,
00695              int atomID,
00696              double force[3]
00697              );
00698
00705 VEXTERNC void Vpmg_qfMutualPolForce(
00706              Vpmg *thee,
00707              Vgrid *induced,
00708              Vgrid *nlInduced,
00709              int atomID,
00710              double force[3]
00711              );
00712
00720 VEXTERNC void Vpmg_ibMutualPolForce(
00721              Vpmg *thee,
00722              Vgrid *induced,
00723              Vgrid *nlInduced,
00724              int atomID,
00725              double force[3]
00726              );
00727
00735 VEXTERNC void Vpmg_dbMutualPolForce(
00736              Vpmg *thee,
00737              Vgrid *induced,
00738              Vgrid *nlInduced,
00739              int atomID,
00740              double force[3]
00741              );
00742
00749 VEXTERNC void Vpmg_printColComp(
00750         Vpmg *thee,
00751         char path[72],
00752         char title[72],
00753         char mxtype[3],
00761         int flag
00765         );
00766
00767 #endif    /* ifndef _VPMG_H_ */
```

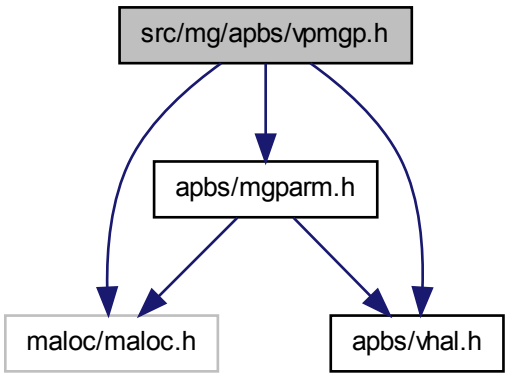
```
00768
```

10.85 src/mg/apbs/vpmgp.h File Reference

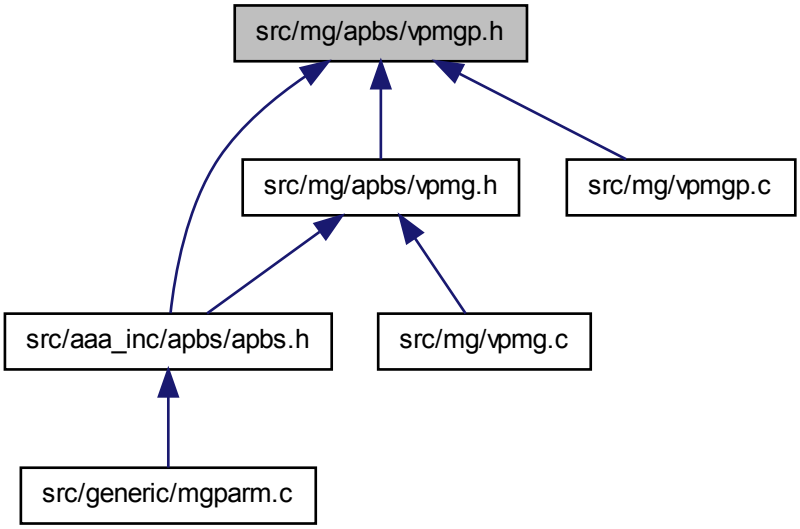
Contains declarations for class Vpmgp.

```
#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.

10.85.1 Detailed Description

Contains declarations for class Vpmgp.

Version

Id:

vpmgp.h 1605 2010-09-13 15:12:09Z yhuang01

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@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
```

```
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
* 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 Washington University in St. Louis 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.

10.86 src/mg/apbs/vpmgp.h

```
00001
00057 #ifndef _VPMGP_H_
00058 #define _VPMGP_H_
00059
00060 #include "maloc/maloc.h"
00061 #include "apbs/vhal.h"
00062 #include "apbs/mgparm.h"
00063
00070 struct sVpmgp {
00071
00072     /* ********** USER-SPECIFIED PARAMETERS ********** */
00073     int nx;
00074     int ny;
00075     int nz;
00076     int nlev;
00077     double hx;
00078     double hy;
00079     double hzed;
```

```
00080     int nonlin;
00085     /* ********** DERIVED PARAMETERS ********** */
00086     int nxc;
00087     int nyc;
00088     int nzc;
00089     int nf;
00090     int nc;
00091     int narrc;
00092     int n_rpc;
00093     int n_iz;
00094     int n_ipc;
00096     int nrwk;
00097     int niwk;
00098     int narr;
00099     int ipkey;
00107     /* ********** PARAMETERS WITH DEFAULT VALUES ********** */
00108     double xcent;
00109     double ycent;
00110     double zcent;
00111     double errtol;
00112     int itmax;
00113     int istop;
00120     int iinfo;
00125     Vbcfl bcfl;
00126     int key;
00129     int iperf;
00134     int meth;
00145     int mgkey;
00148     int nu1;
00149     int nu2;
00150     int mgsmoo;
00156     int mgprol;
00160     int mgcoar;
00164     int mgsolv;
00167     int mgdisc;
00170     double omegal;
00171     double omegan;
00172     int irite;
00173     int ipcon;
00179     double xlen;
00180     double ylen;
00181     double zlen;
00182     double xmin;
00183     double ymin;
00184     double zmin;
00185     double xmax;
00186     double ymax;
00187     double zmax;
00188 };
00189
00194 typedef struct sVpmgp Vpmgp;
00195
00196 /* ////////////////////////////////////////////////////////////////////////////
00197 // Class Vpmgp: Inlineable methods (vpmgp.c)
00199
00200 #if !defined(VINLINE_VPMGP)
00201 #else /* if defined(VINLINE_VPMGP) */
```

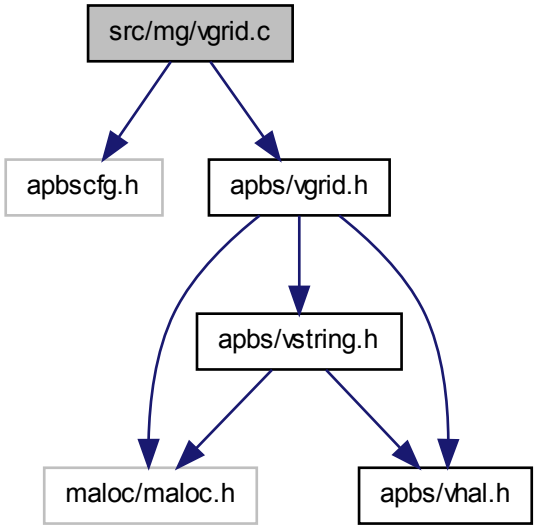
```
00202 #endif /* if !defined(VINLINE_VPMGP) */
00203
00204 /* ////////////////////////////////////////////////////////////////////////
00205 // Class Vpmgp: Non-Inlineable methods (vpmgp.c)
00207
00214 VEXTERNC Vpmgp* Vpmgp_ctor(MGparm *mgparm);
00215
00224 VEXTERNC int Vpmgp_ctor2(Vpmgp *thee, MGparm *mgparm);
00225
00231 VEXTERNC void Vpmgp_dtor(Vpmgp **thee);
00232
00238 VEXTERNC void Vpmgp_dtor2(Vpmgp *thee);
00239
00244 VEXTERNC void Vpmgp_size(
00245  Vpmgp *thee
00246  );
00247
00252 VEXTERNC void Vpmgp_makeCoarse(
00253  int numLevel,
00254  int nxOld,
00255  int nyOld,
00256  int nzOld,
00257  int *nxNew,
00258  int *nyNew,
00259  int *nzNew
00260  );
00261
00262
00263
00264 #endif    /* ifndef _VPMGP_H_ */
```

10.87 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)

  FORTRAN 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]

10.87.1 Detailed Description

Class Vgrid methods.

Author

Nathan Baker

Version

Id:

vgrid.c 1608 2010-10-01 18:39:30Z sobolevnrm

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
* 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 Washington University in St. Louis 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.

10.87.2 Function Documentation

10.87.2.1 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

thee	Object to hold new grid data
iodev	I/O device
iofmt	I/O format
thost	Remote host name
fname	File name
title	Data title
pvec	Masking vector (0 = not written)

Definition at line 779 of file vgrid.c.

10.88 src/mg/vgrid.c

```
00001
00049 #include "apbscfg.h"
00050 #include "apbs/vgrid.h"
00051
00052 VEMBED(rcsid="$Id: vgrid.c 1608 2010-10-01 18:39:30Z sobolevnrm $")
00053
00054 #if !defined(VINLINE_VGRID)
00055     VPUBLIC unsigned long int Vgrid_memChk(Vgrid *thee) {
00056         return Vmem_bytes(thee->mem);
00057     }
00058 #endif
00059 #define IJK(i,j,k)  (((k)*(nx)*(ny))+((j)*(nx))+(i))
00060
00061 VPRIVATE char *MCwhiteChars = " =,;\t\n";
00062 VPRIVATE char *MCcommChars  = "#%";
00063 VPRIVATE double Vcompare;
00064 VPRIVATE char Vprecision[26];
00065
00066 /* ///////////////////////////////////////////////////////////////////////////
00067 // Routine:  Vgrid_ctor
00068 // Author:   Nathan Baker
00070 VPUBLIC Vgrid* Vgrid_ctor(int nx, int ny, int nz,
00071                           double hx, double hy, double hzed,
00072                           double xmin, double ymin, double zmin,
00073                           double *data) {
```

```
00074
00075     Vgrid *thee = VNULL;
00076
00077     thee = Vmem_malloc(VNULL, 1, sizeof(Vgrid));
00078     VASSERT(thee != VNULL);
00079     VASSERT(Vgrid_ctor2(thee, nx, ny, nz, hx, hy, hzed,
00080                 xmin, ymin, zmin, data));
00081
00082     return thee;
00083 }
00084
00085 /* ///////////////////////////////////////////////////////////////////////////
00086 // Routine:  Vgrid_ctor2
00087 // Author:   Nathan Baker
00089 VPUBLIC int Vgrid_ctor2(Vgrid *thee, int nx, int ny, int nz,
00090                 double hx, double hy, double hzed,
00091                 double xmin, double ymin, double zmin,
00092                 double *data) {
00093
00094     if (thee == VNULL) return 0;
00095     thee->nx = nx;
00096     thee->ny = ny;
00097     thee->nz = nz;
00098     thee->hx = hx;
00099     thee->hy = hy;
00100     thee->hzed = hzed;
00101     thee->xmin = xmin;
00102     thee->xmax = xmin + (nx-1)*hx;
00103     thee->ymin = ymin;
00104     thee->ymax = ymin + (ny-1)*hy;
00105     thee->zmin = zmin;
00106     thee->zmax = zmin + (nz-1)*hzed;
00107     if (data == VNULL) {
00108         thee->ctordata = 0;
00109         thee->readdata = 0;
00110     } else {
00111         thee->ctordata = 1;
00112         thee->readdata = 0;
00113         thee->data = data;
00114     }
00115
00116     thee->mem = Vmem_ctor("APBS:VGRID");
00117
00118     Vcompare = pow(10,-1*(VGRID_DIGITS - 2));
00119     sprintf(Vprecision,"%%12.%de %%12.%de %%12.%de", VGRID_DIGITS,
00120             VGRID_DIGITS, VGRID_DIGITS);
00121
00122     return 1;
00123 }
00124
00125 /* ///////////////////////////////////////////////////////////////////////////
00126 // Routine:  Vgrid_dtor
00127 // Author:   Nathan Baker
00129 VPUBLIC void Vgrid_dtor(Vgrid **thee) {
00130
00131     if ((*thee) != VNULL) {
00132         Vgrid_dtor2(*thee);
```

```
00133         Vmem_free(VNULL, 1, sizeof(Vgrid), (void **)thee);
00134         (*thee) = VNULL;
00135     }
00136 }
00137
00138 /* ///////////////////////////////////////////////////////////////////////////
00139 // Routine:  Vgrid_dtor2
00140 // Author:   Nathan Baker
00142 VPUBLIC void Vgrid_dtor2(Vgrid *thee) {
00143
00144     if (thee->readdata) {
00145         Vmem_free(thee->mem, (thee->nx*thee->ny*thee->nz), sizeof(double),
00146           (void **)&(thee->data));
00147     }
00148     Vmem_dtor(&(thee->mem));
00149
00150 }
00151
00152 /* ///////////////////////////////////////////////////////////////////////////
00153 // Routine:  Vgrid_value
00154 // Author:   Nathan Baker
00156 VPUBLIC int Vgrid_value(Vgrid *thee, double pt[3], double *value) {
00157
00158     int nx, ny, nz, ihi, jhi, khi, ilo, jlo, klo;
00159     double hx, hy, hzed, xmin, ymin, zmin, ifloat, jfloat, kfloat;
00160     double xmax, ymax, zmax;
00161     double u, dx, dy, dz;
00162
00163     if (thee == VNULL) {
00164         Vnm_print(2, "Vgrid_value:  Error -- got VNULL thee!\n");
00165         VASSERT(0);
00166     }
00167     if (!(thee->ctordata || thee->readdata)) {
00168         Vnm_print(2, "Vgrid_value:  Error -- no data available!\n");
00169         VASSERT(0);
00170     }
00171
00172     nx = thee->nx;
00173     ny = thee->ny;
00174     nz = thee->nz;
00175     hx = thee->hx;
00176     hy = thee->hy;
00177     hzed = thee->hzed;
00178     xmin = thee->xmin;
00179     ymin = thee->ymin;
00180     zmin = thee->zmin;
00181     xmax = thee->xmax;
00182     ymax = thee->ymax;
00183     zmax = thee->zmax;
00184
00185     u = 0;
00186
00187     ifloat = (pt[0] - xmin)/hx;
00188     jfloat = (pt[1] - ymin)/hy;
00189     kfloat = (pt[2] - zmin)/hzed;
00190
00191     ihi = (int)ceil(ifloat);
```

```
00192     jhi = (int)ceil(jfloat);
00193     khi = (int)ceil(kfloat);
00194     ilo = (int)floor(ifloat);
00195     jlo = (int)floor(jfloat);
00196     klo = (int)floor(kfloat);
00197  if (VABS(pt[0] - xmin) < Vcompare) ilo = 0;
00198  if (VABS(pt[1] - ymin) < Vcompare) jlo = 0;
00199  if (VABS(pt[2] - zmin) < Vcompare) klo = 0;
00200  if (VABS(pt[0] - xmax) < Vcompare) ihi = nx-1;
00201  if (VABS(pt[1] - ymax) < Vcompare) jhi = ny-1;
00202  if (VABS(pt[2] - zmax) < Vcompare) khi = nz-1;
00203
00204     /* See if we're on the mesh */
00205     if ((ihi<nx) && (jhi<ny) && (khi<nz) &&
00206         (ilo>=0) && (jlo>=0) && (klo>=0)) {
00207
00208         dx = ifloat - (double)(ilo);
00209         dy = jfloat - (double)(jlo);
00210         dz = kfloat - (double)(klo);
00211         u = dx      *dy      *dz      *(thee->data[IJK(ihi,jhi,khi)])
00212           + dx      *(1.0-dy)*dz      *(thee->data[IJK(ihi,jlo,khi)])
00213           + dx      *dy      *(1.0-dz)*(thee->data[IJK(ihi,jhi,klo)])
00214           + dx      *(1.0-dy)*(1.0-dz)*(thee->data[IJK(ihi,jlo,klo)])
00215           + (1.0-dx)*dy      *dz      *(thee->data[IJK(ilo,jhi,khi)])
00216           + (1.0-dx)*(1.0-dy)*dz      *(thee->data[IJK(ilo,jlo,khi)])
00217           + (1.0-dx)*dy      *(1.0-dz)*(thee->data[IJK(ilo,jhi,klo)])
00218           + (1.0-dx)*(1.0-dy)*(1.0-dz)*(thee->data[IJK(ilo,jlo,klo)]);
00219
00220         *value = u;
00221
00222         if (isnan(u)) {
00223             Vnm_print(2, "Vgrid_value:  Got NaN!\n");
00224             Vnm_print(2, "Vgrid_value:  (x, y, z) = (%4.3f, %4.3f, %4.3f)\n",
00225                     pt[0], pt[1], pt[2]);
00226             Vnm_print(2, "Vgrid_value:  (ihi, jhi, khi) = (%d, %d, %d)\n",
00227                     ihi, jhi, khi);
00228             Vnm_print(2, "Vgrid_value:  (ilo, jlo, klo) = (%d, %d, %d)\n",
00229                     ilo, jlo, klo);
00230             Vnm_print(2, "Vgrid_value:  (nx, ny, nz) = (%d, %d, %d)\n",
00231                     nx, ny, nz);
00232             Vnm_print(2, "Vgrid_value:  (dx, dy, dz) = (%4.3f, %4.3f, %4.3f)\n",

00233                     dx, dy, dz);
00234             Vnm_print(2, "Vgrid_value:  data[IJK(ihi,jhi,khi)] = %g\n",
00235                     thee->data[IJK(ihi,jhi,khi)]);
00236             Vnm_print(2, "Vgrid_value:  data[IJK(ihi,jlo,khi)] = %g\n",
00237                     thee->data[IJK(ihi,jlo,khi)]);
00238             Vnm_print(2, "Vgrid_value:  data[IJK(ihi,jhi,klo)] = %g\n",
00239                     thee->data[IJK(ihi,jhi,klo)]);
00240             Vnm_print(2, "Vgrid_value:  data[IJK(ihi,jlo,klo)] = %g\n",
00241                     thee->data[IJK(ihi,jlo,klo)]);
00242             Vnm_print(2, "Vgrid_value:  data[IJK(ilo,jhi,khi)] = %g\n",
00243                     thee->data[IJK(ilo,jhi,khi)]);
00244             Vnm_print(2, "Vgrid_value:  data[IJK(ilo,jlo,khi)] = %g\n",
00245                     thee->data[IJK(ilo,jlo,khi)]);
00246             Vnm_print(2, "Vgrid_value:  data[IJK(ilo,jhi,klo)] = %g\n",
00247                     thee->data[IJK(ilo,jhi,klo)]);
```

```
00248             Vnm_print(2, "Vgrid_value:  data[IJK(ilo,jlo,klo)] = %g\n",
00249                     thee->data[IJK(ilo,jlo,klo)]);
00250         }
00251         return 1;
00252
00253     } else {
00254
00255         *value = 0;
00256         return 0;
00257
00258     }
00259
00260     return 0;
00261
00262 }
00263
00264 /* ///////////////////////////////////////////////////////////////////////////
00265 // Routine:  Vgrid_curvature
00266 //
00267 //   Notes:  cflag=0 ==> Reduced Maximal Curvature
00268 //           cflag=1 ==> Mean Curvature (Laplace)
00269 //           cflag=2 ==> Gauss Curvature
00270 //           cflag=3 ==> True Maximal Curvature
00271 //
00272 // Authors:  Stephen Bond and Nathan Baker
00274 VPUBLIC int Vgrid_curvature(Vgrid *thee, double pt[3], int cflag,
00275   double *value) {
00276
00277     double hx, hy, hzed, curv;
00278     double dxx, dyy, dzz;
00279     double uleft, umid, uright, testpt[3];
00280
00281     if (thee == VNULL) {
00282         Vnm_print(2, "Vgrid_curvature:  Error -- got VNULL thee!\n");
00283         VASSERT(0);
00284     }
00285     if (!(thee->ctordata || thee->readdata)) {
00286         Vnm_print(2, "Vgrid_curvature:  Error -- no data available!\n");
00287         VASSERT(0);
00288     }
00289
00290     hx = thee->hx;
00291     hy = thee->hy;
00292     hzed = thee->hzed;
00293
00294     curv = 0.0;
00295
00296     testpt[0] = pt[0];
00297     testpt[1] = pt[1];
00298     testpt[2] = pt[2];
00299
00300     /* Compute 2nd derivative in the x-direction */
00301     VJMPERR1(Vgrid_value( thee, testpt, &umid));
00302     testpt[0] = pt[0] - hx;
00303     VJMPERR1(Vgrid_value( thee, testpt, &uleft));
00304     testpt[0] = pt[0] + hx;
00305     VJMPERR1(Vgrid_value( thee, testpt, &uright));
```

```
00306     testpt[0] = pt[0];
00307
00308     dxx = (uright - 2*umid + uleft)/(hx*hx);
00309
00310     /* Compute 2nd derivative in the y-direction */
00311     VJMPERR1(Vgrid_value( thee, testpt, &umid));
00312     testpt[1] = pt[1] - hy;
00313     VJMPERR1(Vgrid_value( thee, testpt, &uleft));
00314     testpt[1] = pt[1] + hy;
00315     VJMPERR1(Vgrid_value( thee, testpt, &uright));
00316     testpt[1] = pt[1];
00317
00318     dyy = (uright - 2*umid + uleft)/(hy*hy);
00319
00320     /* Compute 2nd derivative in the z-direction */
00321     VJMPERR1(Vgrid_value( thee, testpt, &umid));
00322     testpt[2] = pt[2] - hzed;
00323     VJMPERR1(Vgrid_value( thee, testpt, &uleft));
00324     testpt[2] = pt[2] + hzed;
00325     VJMPERR1(Vgrid_value( thee, testpt, &uright));
00326
00327     dzz = (uright - 2*umid + uleft)/(hzed*hzed);
00328
00329
00330     if ( cflag == 0 ) {
00331         curv = fabs(dxx);
00332         curv = ( curv > fabs(dyy) ) ? curv : fabs(dyy);
00333         curv = ( curv > fabs(dzz) ) ? curv : fabs(dzz);
00334     } else if ( cflag == 1 ) {
00335         curv = (dxx + dyy + dzz)/3.0;
00336     } else {
00337         Vnm_print(2, "Vgrid_curvature:  support for cflag = %d not available!\n",
       cflag);
00338         VASSERT( 0 ); /* Feature Not Coded Yet! */
00339     }
00340
00341     *value = curv;
00342     return 1;
00343
00344     VERROR1:
00345         return 0;
00346
00347 }
00348
00349 /* ///////////////////////////////////////////////////////////////////////////
00350 // Routine:  Vgrid_gradient
00351 //
00352 // Authors:  Nathan Baker and Stephen Bond
00354 VPUBLIC int Vgrid_gradient(Vgrid *thee, double pt[3], double grad[3]) {
00355
00356     double hx, hy, hzed;
00357     double uleft, umid, uright, testpt[3];
00358     int haveleft, haveright;
00359
00360     if (thee == VNULL) {
00361         Vnm_print(2, "Vgrid_gradient:  Error -- got VNULL thee!\n");
00362         VASSERT(0);
```

```
00363     }
00364     if (!(thee->ctordata || thee->readdata)) {
00365         Vnm_print(2, "Vgrid_gradient:  Error -- no data available!\n");
00366         VASSERT(0);
00367     }
00368 
00369     hx = thee->hx;
00370     hy = thee->hy;
00371     hzed = thee->hzed;
00372 
00373     /* Compute derivative in the x-direction */
00374     testpt[0] = pt[0];
00375     testpt[1] = pt[1];
00376     testpt[2] = pt[2];
00377     VJMPERR1( Vgrid_value( thee, testpt, &umid));
00378     testpt[0] = pt[0] - hx;
00379     if (Vgrid_value( thee, testpt, &uleft)) haveleft = 1;
00380     else haveleft = 0;
00381     testpt[0] = pt[0] + hx;
00382     if (Vgrid_value( thee, testpt, &uright)) haveright = 1;
00383     else haveright = 0;
00384     if (haveright && haveleft) grad[0] = (uright - uleft)/(2*hx);
00385     else if (haveright) grad[0] = (uright - umid)/hx;
00386     else if (haveleft) grad[0] = (umid - uleft)/hx;
00387     else VJMPERR1(0);
00388 
00389     /* Compute derivative in the y-direction */
00390     testpt[0] = pt[0];
00391     testpt[1] = pt[1];
00392     testpt[2] = pt[2];
00393     VJMPERR1(Vgrid_value(thee, testpt, &umid));
00394     testpt[1] = pt[1] - hy;
00395     if (Vgrid_value( thee, testpt, &uleft)) haveleft = 1;
00396     else haveleft = 0;
00397     testpt[1] = pt[1] + hy;
00398     if (Vgrid_value( thee, testpt, &uright)) haveright = 1;
00399     else haveright = 0;
00400     if (haveright && haveleft) grad[1] = (uright - uleft)/(2*hy);
00401     else if (haveright) grad[1] = (uright - umid)/hy;
00402     else if (haveleft) grad[1] = (umid - uleft)/hy;
00403     else VJMPERR1(0);
00404 
00405     /* Compute derivative in the z-direction */
00406     testpt[0] = pt[0];
00407     testpt[1] = pt[1];
00408     testpt[2] = pt[2];
00409     VJMPERR1(Vgrid_value(thee, testpt, &umid));
00410     testpt[2] = pt[2] - hzed;
00411     if (Vgrid_value( thee, testpt, &uleft)) haveleft = 1;
00412     else haveleft = 0;
00413     testpt[2] = pt[2] + hzed;
00414     if (Vgrid_value( thee, testpt, &uright)) haveright = 1;
00415     else haveright = 0;
00416     if (haveright && haveleft) grad[2] = (uright - uleft)/(2*hzed);
00417     else if (haveright) grad[2] = (uright - umid)/hzed;
00418     else if (haveleft) grad[2] = (umid - uleft)/hzed;
00419     else VJMPERR1(0);
```

```
00420
00421     return 1;
00422
00423     VERROR1:
00424         return 0;
00425
00426 }
00427
00428 /* ////////////////////////////////////////////////////////////////////////
00429  // Routine:  Vgrid_readGZ
00430  //
00431  // Author:   David Gohara
00433 #ifdef HAVE_ZLIB
00434 #define off_t long
00435 #include "../../contrib/zlib/zlib.h"
00436 #endif
00437 VPUBLIC int Vgrid_readGZ(Vgrid *thee, const char *fname) {
00438
00439 #ifdef HAVE_ZLIB
00440  int i, j, k, q, itmp, u, header;
00441  int length, incr;
00442  double * temp;
00443  double dtmp1, dtmp2, dtmp3;
00444  gzFile infile;
00445  char line[VMAX_ARGLEN];
00446
00447  header = 0;
00448
00449  /* Check to see if the existing data is null and, if not, clear it out */
00450     if (thee->data != VNULL) {
00451         Vnm_print(1, "Vgrid_readDX:  destroying existing data!\n");
00452   Vmem_free(thee->mem, (thee->nx*thee->ny*thee->nz), sizeof(double),
00453       (void **)&(thee->data)); }
00454     thee->readdata = 1;
00455     thee->ctordata = 0;
00456
00457  infile = gzopen(fname, "rb");
00458  if (infile == Z_NULL) {
00459         Vnm_print(2, "Vgrid_writeDX:  Problem opening compressed file %s\n",
00460       fname);
00461         return VRC_FAILURE;
00462     }
00463
00464  thee->hx = 0.0;
00465  thee->hy = 0.0;
00466  thee->hzed = 0.0;
00467
00468  //read data here
00469  while (header < 7) {
00470   if(gzgets(infile, line, VMAX_ARGLEN) == Z_NULL){
00471    return VRC_FAILURE;
00472   }
00473   if(strncmp(line, "#", 1) == 0) continue;
00474   if(line[0] == '\n') continue;
00475
00476   switch (header) {
00477    case 0:
```

```
00478     sscanf(line, "object 1 class gridpositions counts %d %d %d",
00479         &(thee->nx),&(thee->ny),&(thee->nz));
00480     break;
00481    case 1:
00482     sscanf(line, "origin %lf %lf %lf",
00483         &(thee->xmin),&(thee->ymin),&(thee->zmin));
00484     break;
00485    case 2:
00486    case 3:
00487    case 4:
00488     sscanf(line, "delta %lf %lf %lf",&dtmp1,&dtmp2,&dtmp3);
00489     thee->hx += dtmp1;
00490     thee->hy += dtmp2;
00491     thee->hzed += dtmp3;
00492     break;
00493    default:
00494     break;
00495   }
00496
00497   header++;
00498  }
00499
00500  /* Allocate space for the data */
00501     Vnm_print(0, "Vgrid_readGZ:  allocating %d x %d x %d doubles for storage\n",
00502      thee->nx, thee->ny, thee->nz);
00503     thee->data = VNULL;
00504     thee->data = Vmem_malloc(thee->mem, (thee->nx)*(thee->ny)*(thee->nz),
00505         sizeof(double));
00506     if (thee->data == VNULL) {
00507         Vnm_print(2, "Vgrid_readGZ:  Unable to allocate space for data!\n");
00508         return 0;
00509     }
00510
00511  /* Allocate a temporary buffer to store the compressed
00512   * data into (column major order). Add 2 to ensure the buffer is
00513   * big enough to take extra data on the final read loop.
00514   */
00515  temp = (double *)malloc(thee->nx*thee->ny*thee->nz*sizeof(double) + 2);
00516  for(i=0;i<thee->nx*thee->ny*thee->nz;i+=3){
00517   gzgets(infile, line, length);
00518   sscanf(line, "%lf %lf %lf",&temp[i],&temp[i+1],&temp[i+2]);
00519  }
00520
00521  /* Now move the data to row major order */
00522  incr = 0;
00523  for (i=0; i<thee->nx; i++) {
00524   for (j=0; j<thee->ny; j++) {
00525    for (k=0; k<thee->nz; k++) {
00526     u = k*(thee->nx)*(thee->ny)+j*(thee->nx)+i;
00527     (thee->data)[u] = temp[incr++];
00528    }
00529   }
00530  }
00531
00532  /* calculate grid maxima */
00533  thee->xmax = thee->xmin + (thee->nx-1)*thee->hx;
00534  thee->ymax = thee->ymin + (thee->ny-1)*thee->hy;
```

```
00535  thee->zmax = thee->zmin + (thee->nz-1)*thee->hzed;
00536
00537  /* Close off the socket */
00538  gzclose(infile);
00539  free(temp);
00540 #else
00541
00542  Vnm_print(0, "WARNING\n");
00543  Vnm_print(0, "Vgrid_readGZ:  gzip read/write support is disabled in this build\n
      ");
00544  Vnm_print(0, "Vgrid_readGZ:  configure and compile without the --disable-zlib fl
      ag.\n");
00545  Vnm_print(0, "WARNING\n");
00546 #endif
00547  return VRC_SUCCESS;
00548 }
00549
00550 /* ///////////////////////////////////////////////////////////////////////////
00551 // Routine:  Vgrid_readDX
00552 //
00553 // Author:   Nathan Baker
00555 VPUBLIC int Vgrid_readDX(Vgrid *thee, const char *iodev, const char *iofmt,
00556   const char *thost, const char *fname) {
00557
00558     int i, j, k, itmp, u;
00559     double dtmp;
00560     char tok[VMAX_BUFSIZE];
00561     Vio *sock;
00562
00563     /* Check to see if the existing data is null and, if not, clear it out */
00564     if (thee->data != VNULL) {
00565         Vnm_print(1, "Vgrid_readDX:  destroying existing data!\n");
00566  Vmem_free(thee->mem, (thee->nx*thee->ny*thee->nz), sizeof(double),
00567           (void **)&(thee->data)); }
00568     thee->readdata = 1;
00569     thee->ctordata = 0;
00570
00571     /* Set up the virtual socket */
00572     sock = Vio_ctor(iodev,iofmt,thost,fname,"r");
00573     if (sock == VNULL) {
00574         Vnm_print(2, "Vgrid_readDX: Problem opening virtual socket %s\n",
00575           fname);
00576         return 0;
00577     }
00578     if (Vio_accept(sock, 0) < 0) {
00579         Vnm_print(2, "Vgrid_readDX: Problem accepting virtual socket %s\n",
00580           fname);
00581         return 0;
00582     }
00583
00584     Vio_setWhiteChars(sock, MCwhiteChars);
00585     Vio_setCommChars(sock, MCcommChars);
00586
00587     /* Read in the DX regular positions */
00588     /* Get "object" */
00589     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00590     VJMPERR1(!strcmp(tok, "object"));
```

```
00591     /* Get "1" */
00592     VJMPERR2(1 == Vio_scanf(sock, "%d", &itmp));
00593     /* Get "class" */
00594     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00595     VJMPERR1(!strcmp(tok, "class"));
00596     /* Get "gridpositions" */
00597     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00598     VJMPERR1(!strcmp(tok, "gridpositions"));
00599     /* Get "counts" */
00600     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00601     VJMPERR1(!strcmp(tok, "counts"));
00602     /* Get nx */
00603     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00604     VJMPERR1(1 == sscanf(tok, "%d", &(thee->nx)));
00605     /* Get ny */
00606     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00607     VJMPERR1(1 == sscanf(tok, "%d", &(thee->ny)));
00608     /* Get nz */
00609     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00610     VJMPERR1(1 == sscanf(tok, "%d", &(thee->nz)));
00611     Vnm_print(0, "Vgrid_readDX:  Grid dimensions %d x %d x %d grid\n",
00612      thee->nx, thee->ny, thee->nz);
00613     /* Get "origin" */
00614     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00615     VJMPERR1(!strcmp(tok, "origin"));
00616     /* Get xmin */
00617     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00618     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->xmin)));
00619     /* Get ymin */
00620     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00621     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->ymin)));
00622     /* Get zmin */
00623     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00624     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->zmin)));
00625     Vnm_print(0, "Vgrid_readDX:  Grid origin = (%g, %g, %g)\n",
00626       thee->xmin, thee->ymin, thee->zmin);
00627     /* Get "delta" */
00628     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00629     VJMPERR1(!strcmp(tok, "delta"));
00630     /* Get hx */
00631     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00632     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->hx)));
00633     /* Get 0.0 */
00634     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00635     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00636     VJMPERR1(dtmp == 0.0);
00637     /* Get 0.0 */
00638     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00639     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00640     VJMPERR1(dtmp == 0.0);
00641     /* Get "delta" */
00642     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00643     VJMPERR1(!strcmp(tok, "delta"));
00644     /* Get 0.0 */
00645     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00646     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00647     VJMPERR1(dtmp == 0.0);
```

```
00648     /* Get hy */
00649     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00650     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->hy)));
00651     /* Get 0.0 */
00652     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00653     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00654     VJMPERR1(dtmp == 0.0);
00655     /* Get "delta" */
00656     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00657     VJMPERR1(!strcmp(tok, "delta"));
00658     /* Get 0.0 */
00659     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00660     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00661     VJMPERR1(dtmp == 0.0);
00662     /* Get 0.0 */
00663     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00664     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00665     VJMPERR1(dtmp == 0.0);
00666     /* Get hz */
00667     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00668     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->hzed)));
00669     Vnm_print(0, "Vgrid_readDX:  Grid spacings = (%g, %g, %g)\n",
00670       thee->hx, thee->hy, thee->hzed);
00671     /* Get "object" */
00672     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00673     VJMPERR1(!strcmp(tok, "object"));
00674     /* Get "2" */
00675     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00676     /* Get "class" */
00677     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00678     VJMPERR1(!strcmp(tok, "class"));
00679     /* Get "gridconnections" */
00680     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00681     VJMPERR1(!strcmp(tok, "gridconnections"));
00682     /* Get "counts" */
00683     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00684     VJMPERR1(!strcmp(tok, "counts"));
00685     /* Get the dimensions again */
00686     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00687     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00688     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00689     /* Get "object" */
00690     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00691     VJMPERR1(!strcmp(tok, "object"));
00692     /* Get # */
00693     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00694     /* Get "class" */
00695     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00696     VJMPERR1(!strcmp(tok, "class"));
00697     /* Get "array" */
00698     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00699     VJMPERR1(!strcmp(tok, "array"));
00700     /* Get "type" */
00701     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00702     VJMPERR1(!strcmp(tok, "type"));
00703     /* Get "double" */
00704     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
```

```
00705     VJMPERR1(!strcmp(tok, "double"));
00706     /* Get "rank" */
00707     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00708     VJMPERR1(!strcmp(tok, "rank"));
00709     /* Get # */
00710     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00711     /* Get "items" */
00712     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00713     VJMPERR1(!strcmp(tok, "items"));
00714     /* Get # */
00715     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00716     VJMPERR1(1 == sscanf(tok, "%d", &itmp));
00717     VJMPERR1(((thee->nx)*(thee->ny)*(thee->nz)) == itmp);
00718     /* Get "data" */
00719     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00720     VJMPERR1(!strcmp(tok, "data"));
00721     /* Get "follows" */
00722     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00723     VJMPERR1(!strcmp(tok, "follows"));
00724
00725     /* Allocate space for the data */
00726     Vnm_print(0, "Vgrid_readDX:  allocating %d x %d x %d doubles for storage\n",
00727       thee->nx, thee->ny, thee->nz);
00728     thee->data = VNULL;
00729     thee->data = Vmem_malloc(thee->mem, (thee->nx)*(thee->ny)*(thee->nz),
00730       sizeof(double));
00731     if (thee->data == VNULL) {
00732         Vnm_print(2, "Vgrid_readDX:  Unable to allocate space for data!\n");
00733         return 0;
00734     }
00735
00736     for (i=0; i<thee->nx; i++) {
00737         for (j=0; j<thee->ny; j++) {
00738             for (k=0; k<thee->nz; k++) {
00739                 u = k*(thee->nx)*(thee->ny)+j*(thee->nx)+i;
00740                 VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00741                 VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00742                 (thee->data)[u] = dtmp;
00743             }
00744         }
00745     }
00746
00747     /* calculate grid maxima */
00748     thee->xmax = thee->xmin + (thee->nx-1)*thee->hx;
00749     thee->ymax = thee->ymin + (thee->ny-1)*thee->hy;
00750     thee->zmax = thee->zmin + (thee->nz-1)*thee->hzed;
00751
00752     /* Close off the socket */
00753     Vio_acceptFree(sock);
00754     Vio_dtor(&sock);
00755
00756     return 1;
00757
00758   VERROR1:
00759     Vio_dtor(&sock);
00760     Vnm_print(2, "Vgrid_readDX:  Format problem with input file <%s>\n",
00761       fname);
```

```
00762     return 0;
00763
00764   VERROR2:
00765     Vio_dtor(&sock);
00766     Vnm_print(2, "Vgrid_readDX:  I/O problem with input file <%s>\n",
00767       fname);
00768     return 0;
00769
00770
00771
00772 }
00773
00774 /* ///////////////////////////////////////////////////////////////////////////
00775  // Routine:  Vgrid_writeGZ
00776  //
00777  // Author:   Nathan Baker
00779 VPUBLIC void Vgrid_writeGZ(Vgrid *thee, const char *iodev, const char *iofmt,
00780        const char *thost, const char *fname, char *title, double *pvec) {
00781
00782 #ifdef HAVE_ZLIB
00783  double xmin, ymin, zmin, hx, hy, hzed;
00784
00785  int nx, ny, nz;
00786  int icol, i, j, k, u, usepart, nxPART, nyPART, nzPART, gotit;
00787  double x, y, z, xminPART, yminPART, zminPART;
00788
00789  int txyz;
00790  double txmin, tymin, tzmin;
00791
00792  char header[8196];
00793  char footer[8196];
00794  char line[80];
00795  char newline[] = "\n";
00796  gzFile outfile;
00797  char precFormat[VMAX_BUFSIZE];
00798
00799  if (thee == VNULL) {
00800   Vnm_print(2, "Vgrid_writeGZ:  Error -- got VNULL thee!\n");
00801   VASSERT(0);
00802  }
00803  if (!(thee->ctordata || thee->readdata)) {
00804   Vnm_print(2, "Vgrid_writeGZ:  Error -- no data available!\n");
00805   VASSERT(0);
00806  }
00807
00808  hx = thee->hx;
00809  hy = thee->hy;
00810  hzed = thee->hzed;
00811  nx = thee->nx;
00812  ny = thee->ny;
00813  nz = thee->nz;
00814  xmin = thee->xmin;
00815  ymin = thee->ymin;
00816  zmin = thee->zmin;
00817
00818  if (pvec == VNULL) usepart = 0;
00819  else usepart = 1;
```

```
00820
00821  /* Set up the virtual socket */
00822  Vnm_print(0, "Vgrid_writeGZ:  Opening file...\n");
00823  outfile = gzopen(fname, "wb");
00824
00825  if (usepart) {
00826   /* Get the lower corner and number of grid points for the local
00827    * partition */
00828   xminPART = VLARGE;
00829   yminPART = VLARGE;
00830   zminPART = VLARGE;
00831   nxPART = 0;
00832   nyPART = 0;
00833   nzPART = 0;
00834   /* First, search for the lower corner */
00835   for (k=0; k<nz; k++) {
00836    z = k*hzed + zmin;
00837    for (j=0; j<ny; j++) {
00838     y = j*hy + ymin;
00839     for (i=0; i<nx; i++) {
00840      x = i*hx + xmin;
00841      if (pvec[IJK(i,j,k)] > 0.0) {
00842       if (x < xminPART) xminPART = x;
00843       if (y < yminPART) yminPART = y;
00844       if (z < zminPART) zminPART = z;
00845      }
00846     }
00847    }
00848   }
00849   /* Now search for the number of grid points in the z direction */
00850   for (k=0; k<nz; k++) {
00851    gotit = 0;
00852    for (j=0; j<ny; j++) {
00853     for (i=0; i<nx; i++) {
00854      if (pvec[IJK(i,j,k)] > 0.0) {
00855       gotit = 1;
00856       break;
00857      }
00858     }
00859     if (gotit) break;
00860    }
00861    if (gotit) nzPART++;
00862   }
00863   /* Now search for the number of grid points in the y direction */
00864   for (j=0; j<ny; j++) {
00865    gotit = 0;
00866    for (k=0; k<nz; k++) {
00867     for (i=0; i<nx; i++) {
00868      if (pvec[IJK(i,j,k)] > 0.0) {
00869       gotit = 1;
00870       break;
00871      }
00872     }
00873     if (gotit) break;
00874    }
00875    if (gotit) nyPART++;
00876   }
```

```
00877   /* Now search for the number of grid points in the x direction */
00878   for (i=0; i<nx; i++) {
00879    gotit = 0;
00880    for (k=0; k<nz; k++) {
00881     for (j=0; j<ny; j++) {
00882      if (pvec[IJK(i,j,k)] > 0.0) {
00883       gotit = 1;
00884       break;
00885      }
00886     }
00887     if (gotit) break;
00888    }
00889    if (gotit) nxPART++;
00890   }
00891
00892   if ((nxPART != nx) || (nyPART != ny) || (nzPART != nz)) {
00893    Vnm_print(0, "Vgrid_writeGZ:  printing only subset of domain\n");
00894   }
00895
00896   txyz = (nxPART*nyPART*nzPART);
00897   txmin = xminPART;
00898   tymin = yminPART;
00899   tzmin = zminPART;
00900
00901  }else {
00902
00903   txyz = (nx*ny*nz);
00904   txmin = xmin;
00905   tymin = ymin;
00906   tzmin = zmin;
00907
00908  }
00909
00910  /* Write off the title (if we're not XDR) */
00911  sprintf(header,
00912    "# Data from %s\n" \
00913    "# \n"       \
00914    "# %s\n"    \
00915    "# \n"       \
00916    "object 1 class gridpositions counts %i %i %i\n" \
00917    "origin %12.6e %12.6e %12.6e\n" \
00918    "delta %12.6e 0.000000e+00 0.000000e+00\n"  \
00919    "delta 0.000000e+00 %12.6e 0.000000e+00\n"  \
00920    "delta 0.000000e+00 0.000000e+00 %12.6e\n"  \
00921    "object 2 class gridconnections counts %i %i %i\n"\
00922    "object 3 class array type double rank 0 items %i data follows\n",
00923    PACKAGE_STRING,title,nx,ny,nz,txmin,tymin,tzmin,
00924    hx,hy,hzed,nx,ny,nz,txyz);
00925  gzwrite(outfile, header, strlen(header)*sizeof(char));
00926
00927  /* Now write the data */
00928  icol = 0;
00929  for (i=0; i<nx; i++) {
00930   for (j=0; j<ny; j++) {
00931    for (k=0; k<nz; k++) {
00932     u = k*(nx)*(ny)+j*(nx)+i;
00933     if (pvec[u] > 0.0) {
```

```
00934       sprintf(line, "%12.6e ", thee->data[u]);
00935       gzwrite(outfile, line, strlen(line)*sizeof(char));
00936       icol++;
00937       if (icol == 3) {
00938        icol = 0;
00939        gzwrite(outfile, newline, strlen(newline)*sizeof(char));
00940       }
00941      }
00942     }
00943    }
00944  }
00945  if(icol < 3){
00946   char newline[] = "\n";
00947   gzwrite(outfile, newline, strlen(newline)*sizeof(char));
00948  }
00949
00950  /* Create the field */
00951  sprintf(footer, "attribute \"dep\" string \"positions\"\n" \
00952    "object \"regular positions regular connections\" class field\n" \
00953    "component \"positions\" value 1\n" \
00954    "component \"connections\" value 2\n" \
00955    "component \"data\" value 3\n");
00956  gzwrite(outfile, footer, strlen(footer)*sizeof(char));
00957
00958  gzclose(outfile);
00959 #else
00960
00961  Vnm_print(0, "WARNING\n");
00962  Vnm_print(0, "Vgrid_readGZ:  gzip read/write support is disabled in this build\n
      ");
00963  Vnm_print(0, "Vgrid_readGZ:  configure and compile without the --disable-zlib fl
      ag.\n");
00964  Vnm_print(0, "WARNING\n");
00965 #endif
00966 }
00967
00968 /* ///////////////////////////////////////////////////////////////////////////
00969 // Routine:  Vgrid_writeDX
00970 //
00971 // Author:   Nathan Baker
00973 VPUBLIC void Vgrid_writeDX(Vgrid *thee, const char *iodev, const char *iofmt,
00974   const char *thost, const char *fname, char *title, double *pvec) {
00975
00976     double xmin, ymin, zmin, hx, hy, hzed;
00977     int nx, ny, nz;
00978     int icol, i, j, k, u, usepart, nxPART, nyPART, nzPART, gotit;
00979     double x, y, z, xminPART, yminPART, zminPART;
00980     Vio *sock;
00981     char precFormat[VMAX_BUFSIZE];
00982
00983     if (thee == VNULL) {
00984         Vnm_print(2, "Vgrid_writeDX:  Error -- got VNULL thee!\n");
00985         VASSERT(0);
00986     }
00987     if (!(thee->ctordata || thee->readdata)) {
00988         Vnm_print(2, "Vgrid_writeDX:  Error -- no data available!\n");
00989         VASSERT(0);
```

```
00990     }
00991
00992     hx = thee->hx;
00993     hy = thee->hy;
00994     hzed = thee->hzed;
00995     nx = thee->nx;
00996     ny = thee->ny;
00997     nz = thee->nz;
00998     xmin = thee->xmin;
00999     ymin = thee->ymin;
01000     zmin = thee->zmin;
01001
01002     if (pvec == VNULL) usepart = 0;
01003     else usepart = 1;
01004
01005     /* Set up the virtual socket */
01006     Vnm_print(0, "Vgrid_writeDX:  Opening virtual socket...\n");
01007     sock = Vio_ctor(iodev,iofmt,thost,fname,"w");
01008     if (sock == VNULL) {
01009         Vnm_print(2, "Vgrid_writeDX:  Problem opening virtual socket %s\n",
01010           fname);
01011         return;
01012     }
01013     if (Vio_connect(sock, 0) < 0) {
01014         Vnm_print(2, "Vgrid_writeDX: Problem connecting virtual socket %s\n",
01015           fname);
01016         return;
01017     }
01018
01019     Vio_setWhiteChars(sock, MCwhiteChars);
01020     Vio_setCommChars(sock, MCcommChars);
01021
01022     Vnm_print(0, "Vgrid_writeDX:  Writing to virtual socket...\n");
01023
01024     if (usepart) {
01025         /* Get the lower corner and number of grid points for the local
01026          * partition */
01027         xminPART = VLARGE;
01028         yminPART = VLARGE;
01029         zminPART = VLARGE;
01030         nxPART = 0;
01031         nyPART = 0;
01032         nzPART = 0;
01033         /* First, search for the lower corner */
01034         for (k=0; k<nz; k++) {
01035             z = k*hzed + zmin;
01036             for (j=0; j<ny; j++) {
01037                 y = j*hy + ymin;
01038                 for (i=0; i<nx; i++) {
01039                     x = i*hx + xmin;
01040                     if (pvec[IJK(i,j,k)] > 0.0) {
01041                         if (x < xminPART) xminPART = x;
01042                         if (y < yminPART) yminPART = y;
01043                         if (z < zminPART) zminPART = z;
01044                     }
01045                 }
01046             }
```

```
01047         }
01048         /* Now search for the number of grid points in the z direction */
01049         for (k=0; k<nz; k++) {
01050             gotit = 0;
01051             for (j=0; j<ny; j++) {
01052                 for (i=0; i<nx; i++) {
01053                     if (pvec[IJK(i,j,k)] > 0.0) {
01054                         gotit = 1;
01055                         break;
01056                     }
01057                 }
01058                 if (gotit) break;
01059             }
01060             if (gotit) nzPART++;
01061         }
01062         /* Now search for the number of grid points in the y direction */
01063         for (j=0; j<ny; j++) {
01064             gotit = 0;
01065             for (k=0; k<nz; k++) {
01066                 for (i=0; i<nx; i++) {
01067                     if (pvec[IJK(i,j,k)] > 0.0) {
01068                         gotit = 1;
01069                         break;
01070                     }
01071                 }
01072                 if (gotit) break;
01073             }
01074             if (gotit) nyPART++;
01075         }
01076         /* Now search for the number of grid points in the x direction */
01077         for (i=0; i<nx; i++) {
01078             gotit = 0;
01079             for (k=0; k<nz; k++) {
01080                 for (j=0; j<ny; j++) {
01081                     if (pvec[IJK(i,j,k)] > 0.0) {
01082                         gotit = 1;
01083                         break;
01084                     }
01085                 }
01086                 if (gotit) break;
01087             }
01088             if (gotit) nxPART++;
01089         }
01090
01091         if ((nxPART != nx) || (nyPART != ny) || (nzPART != nz)) {
01092             Vnm_print(0, "Vgrid_writeDX:  printing only subset of domain\n");
01093         }
01094
01095
01096         /* Write off the title (if we're not XDR) */
01097         if (Vstring_strcasecmp(iofmt, "XDR") == 0) {
01098             Vnm_print(0, "Vgrid_writeDX:  Skipping comments for XDR format.\n");
01099         } else {
01100             Vnm_print(0, "Vgrid_writeDX:  Writing comments for %s format.\n",
01101               iofmt);
01102             Vio_printf(sock, "# Data from %s\n", PACKAGE_STRING);
01103             Vio_printf(sock, "# \n");
```

```
01104             Vio_printf(sock, "# %s\n", title);
01105             Vio_printf(sock, "# \n");
01106         }
01107
01108         /* Write off the DX regular positions */
01109         Vio_printf(sock, "object 1 class gridpositions counts %d %d %d\n",
01110           nxPART, nyPART, nzPART);
01111
01112         sprintf(precFormat, Vprecision, xminPART, yminPART, zminPART);
01113         Vio_printf(sock, "origin %s\n", precFormat);
01114         sprintf(precFormat, Vprecision, hx, 0.0, 0.0);
01115         Vio_printf(sock, "delta %s\n", precFormat);
01116         sprintf(precFormat, Vprecision, 0.0, hy, 0.0);
01117         Vio_printf(sock, "delta %s\n", precFormat);
01118         sprintf(precFormat, Vprecision, 0.0, 0.0, hzed);
01119         Vio_printf(sock, "delta %s\n", precFormat);
01120
01121         /* Write off the DX regular connections */
01122         Vio_printf(sock, "object 2 class gridconnections counts %d %d %d\n",
01123           nxPART, nyPART, nzPART);
01124
01125         /* Write off the DX data */
01126         Vio_printf(sock, "object 3 class array type double rank 0 items %d \
01127 data follows\n", (nxPART*nyPART*nzPART));
01128         icol = 0;
01129         for (i=0; i<nx; i++) {
01130             for (j=0; j<ny; j++) {
01131                 for (k=0; k<nz; k++) {
01132                     u = k*(nx)*(ny)+j*(nx)+i;
01133                     if (pvec[u] > 0.0) {
01134                         Vio_printf(sock, "%12.6e ", thee->data[u]);
01135                         icol++;
01136                         if (icol == 3) {
01137                             icol = 0;
01138                             Vio_printf(sock, "\n");
01139                         }
01140                     }
01141                 }
01142             }
01143         }
01144
01145         if (icol != 0) Vio_printf(sock, "\n");
01146
01147         /* Create the field */
01148         Vio_printf(sock, "attribute \"dep\" string \"positions\"\n");
01149         Vio_printf(sock, "object \"regular positions regular connections\" \
01150 class field\n");
01151         Vio_printf(sock, "component \"positions\" value 1\n");
01152         Vio_printf(sock, "component \"connections\" value 2\n");
01153         Vio_printf(sock, "component \"data\" value 3\n");
01154
01155     } else {
01156         /* Write off the title (if we're not XDR) */
01157         if (Vstring_strcasecmp(iofmt, "XDR") == 0) {
01158             Vnm_print(0, "Vgrid_writeDX:  Skipping comments for XDR format.\n");
01159         } else {
01160             Vnm_print(0, "Vgrid_writeDX:  Writing comments for %s format.\n",
```

```
01161               iofmt);
01162             Vio_printf(sock, "# Data from %s\n", PACKAGE_STRING);
01163             Vio_printf(sock, "# \n");
01164             Vio_printf(sock, "# %s\n", title);
01165             Vio_printf(sock, "# \n");
01166         }
01167 
01168 
01169         /* Write off the DX regular positions */
01170         Vio_printf(sock, "object 1 class gridpositions counts %d %d %d\n",
01171           nx, ny, nz);
01172 
01173         sprintf(precFormat, Vprecision, xmin, ymin, zmin);
01174         Vio_printf(sock, "origin %s\n", precFormat);
01175         sprintf(precFormat, Vprecision, hx, 0.0, 0.0);
01176         Vio_printf(sock, "delta %s\n", precFormat);
01177         sprintf(precFormat, Vprecision, 0.0, hy, 0.0);
01178         Vio_printf(sock, "delta %s\n", precFormat);
01179         sprintf(precFormat, Vprecision, 0.0, 0.0, hzed);
01180         Vio_printf(sock, "delta %s\n", precFormat);
01181 
01182         /* Write off the DX regular connections */
01183         Vio_printf(sock, "object 2 class gridconnections counts %d %d %d\n",
01184           nx, ny, nz);
01185 
01186         /* Write off the DX data */
01187         Vio_printf(sock, "object 3 class array type double rank 0 items %d \
01188 data follows\n", (nx*ny*nz));
01189         icol = 0;
01190         for (i=0; i<nx; i++) {
01191             for (j=0; j<ny; j++) {
01192                 for (k=0; k<nz; k++) {
01193                     u = k*(nx)*(ny)+j*(nx)+i;
01194                     Vio_printf(sock, "%12.6e ", thee->data[u]);
01195                     icol++;
01196                     if (icol == 3) {
01197                         icol = 0;
01198                         Vio_printf(sock, "\n");
01199                     }
01200                 }
01201             }
01202         }
01203         if (icol != 0) Vio_printf(sock, "\n");
01204 
01205         /* Create the field */
01206         Vio_printf(sock, "attribute \"dep\" string \"positions\"\n");
01207         Vio_printf(sock, "object \"regular positions regular connections\" \
01208 class field\n");
01209         Vio_printf(sock, "component \"positions\" value 1\n");
01210         Vio_printf(sock, "component \"connections\" value 2\n");
01211         Vio_printf(sock, "component \"data\" value 3\n");
01212     }
01213 
01214     /* Close off the socket */
01215     Vio_connectFree(sock);
01216     Vio_dtor(&sock);
01217 }
```

```
01218
01219 /* ///////////////////////////////////////////////////////////////////////////
01220 // Routine:  Vgrid_writeUHBD
01221 // Author:   Nathan Baker
01223 VPUBLIC void Vgrid_writeUHBD(Vgrid *thee, const char *iodev, const char *iofmt,
01224   const char *thost, const char *fname, char *title, double *pvec) {
01225
01226     int icol, i, j, k, u, nx, ny, nz, gotit;
01227     double xmin, ymin, zmin, hzed, hy, hx;
01228     Vio *sock;
01229
01230     if (thee == VNULL) {
01231         Vnm_print(2, "Vgrid_writeUHBD:  Error -- got VNULL thee!\n");
01232         VASSERT(0);
01233     }
01234     if (!(thee->ctordata || thee->readdata)) {
01235         Vnm_print(2, "Vgrid_writeUHBD:  Error -- no data available!\n");
01236         VASSERT(0);
01237     }
01238
01239     if ((thee->hx!=thee->hy) || (thee->hy!=thee->hzed)
01240       || (thee->hx!=thee->hzed)) {
01241         Vnm_print(2, "Vgrid_writeUHBD: can't write UHBD mesh with non-uniform \
01242 spacing\n");
01243         return;
01244     }
01245
01246     /* Set up the virtual socket */
01247     sock = Vio_ctor(iodev,iofmt,thost,fname,"w");
01248     if (sock == VNULL) {
01249         Vnm_print(2, "Vgrid_writeUHBD: Problem opening virtual socket %s\n",
01250           fname);
01251         return;
01252     }
01253     if (Vio_connect(sock, 0) < 0) {
01254         Vnm_print(2, "Vgrid_writeUHBD: Problem connecting virtual socket %s\n",
01255           fname);
01256         return;
01257     }
01258
01259     /* Get the lower corner and number of grid points for the local
01260      * partition */
01261     hx = thee->hx;
01262     hy = thee->hy;
01263     hzed = thee->hzed;
01264     nx = thee->nx;
01265     ny = thee->ny;
01266     nz = thee->nz;
01267     xmin = thee->xmin;
01268     ymin = thee->ymin;
01269     zmin = thee->zmin;
01270
01271     /* Let interested folks know that partition information is ignored */
01272     if (pvec != VNULL) {
01273         gotit = 0;
01274         for (i=0; i<(nx*ny*nz); i++) {
01275             if (pvec[i] == 0) {
```

```
01276                 gotit = 1;
01277                 break;
01278             }
01279         }
01280         if (gotit) {
01281             Vnm_print(2, "Vgrid_writeUHBD:  IGNORING PARTITION INFORMATION!\n");
01282             Vnm_print(2, "Vgrid_writeUHBD:  This means I/O from parallel runs \
01283 will have significant overlap.\n");
01284         }
01285     }
01286 
01287     /* Write out the header */
01288     Vio_printf(sock, "%72s\n", title);
01289     Vio_printf(sock, "%12.5e%12.5e%7d%7d%7d%7d%7d\n", 1.0, 0.0, -1, 0,
01290       nz, 1, nz);
01291     Vio_printf(sock, "%7d%7d%7d%12.5e%12.5e%12.5e%12.5e\n", nx, ny, nz,
01292       hx, (xmin-hx), (ymin-hx), (zmin-hx));
01293     Vio_printf(sock, "%12.5e%12.5e%12.5e%12.5e\n", 0.0, 0.0, 0.0, 0.0);
01294     Vio_printf(sock, "%12.5e%12.5e%7d%7d", 0.0, 0.0, 0, 0);
01295 
01296     /* Write out the entries */
01297     icol = 0;
01298     for (k=0; k<nz; k++) {
01299         Vio_printf(sock, "\n%7d%7d%7d\n", k+1, thee->nx, thee->ny);
01300         icol = 0;
01301         for (j=0; j<ny; j++) {
01302             for (i=0; i<nx; i++) {
01303                 u = k*(nx)*(ny)+j*(nx)+i;
01304                 icol++;
01305                 Vio_printf(sock, " %12.5e", thee->data[u]);
01306                 if (icol == 6) {
01307                     icol = 0;
01308                     Vio_printf(sock, "\n");
01309                 }
01310             }
01311         }
01312     }
01313     if (icol != 0) Vio_printf(sock, "\n");
01314 
01315     /* Close off the socket */
01316     Vio_connectFree(sock);
01317     Vio_dtor(&sock);
01318 }
01319 
01320 VPUBLIC double Vgrid_integrate(Vgrid *thee) {
01321 
01322     int i, j, k, nx, ny, nz;
01323     double sum, w;
01324 
01325     if (thee == VNULL) {
01326         Vnm_print(2, "Vgrid_integrate:  Got VNULL thee!\n");
01327         VASSERT(0);
01328     }
01329 
01330     nx = thee->nx;
01331     ny = thee->ny;
01332     nz = thee->nz;
```

```
01333
01334     sum = 0.0;
01335
01336     for (k=0; k<nz; k++) {
01337   w = 1.0;
01338   if ((k==0) || (k==(nz-1))) w = w * 0.5;
01339         for (j=0; j<ny; j++) {
01340    w = 1.0;
01341    if ((j==0) || (j==(ny-1))) w = w * 0.5;
01342               for (i=0; i<nx; i++) {
01343     w = 1.0;
01344     if ((i==0) || (i==(nx-1))) w = w * 0.5;
01345                 sum = sum + w*(thee->data[IJK(i,j,k)]);
01346             }
01347         }
01348     }
01349
01350     sum = sum*(thee->hx)*(thee->hy)*(thee->hzed);
01351
01352     return sum;
01353
01354 }
01355
01356
01357 VPUBLIC double Vgrid_normL1(Vgrid *thee) {
01358
01359     int i, j, k, nx, ny, nz;
01360     double sum;
01361
01362     if (thee == VNULL) {
01363         Vnm_print(2, "Vgrid_normL1:  Got VNULL thee!\n");
01364         VASSERT(0);
01365     }
01366
01367     nx = thee->nx;
01368     ny = thee->ny;
01369     nz = thee->nz;
01370
01371     sum = 0.0;
01372     for (k=0; k<nz; k++) {
01373         for (j=0; j<ny; j++) {
01374             for (i=0; i<nx; i++) {
01375                 sum = sum + VABS(thee->data[IJK(i,j,k)]);
01376             }
01377         }
01378     }
01379
01380     sum = sum*(thee->hx)*(thee->hy)*(thee->hzed);
01381
01382     return sum;
01383
01384 }
01385
01386 VPUBLIC double Vgrid_normL2(Vgrid *thee) {
01387
01388     int i, j, k, nx, ny, nz;
01389     double sum;
```

```
01390
01391     if (thee == VNULL) {
01392         Vnm_print(2, "Vgrid_normL2:  Got VNULL thee!\n");
01393         VASSERT(0);
01394     }
01395
01396     nx = thee->nx;
01397     ny = thee->ny;
01398     nz = thee->nz;
01399
01400     sum = 0.0;
01401     for (k=0; k<nz; k++) {
01402         for (j=0; j<ny; j++) {
01403             for (i=0; i<nx; i++) {
01404                 sum = sum + VSQR(thee->data[IJK(i,j,k)]);
01405             }
01406         }
01407     }
01408
01409     sum = sum*(thee->hx)*(thee->hy)*(thee->hzed);
01410
01411     return VSQRT(sum);
01412
01413 }
01414
01415 VPUBLIC double Vgrid_seminormH1(Vgrid *thee) {
01416
01417     int i, j, k, d, nx, ny, nz;
01418     double pt[3], grad[3], sum, hx, hy, hzed, xmin, ymin, zmin;
01419
01420     if (thee == VNULL) {
01421         Vnm_print(2, "Vgrid_seminormH1:  Got VNULL thee!\n");
01422         VASSERT(0);
01423     }
01424
01425     nx = thee->nx;
01426     ny = thee->ny;
01427     nz = thee->nz;
01428     hx = thee->hx;
01429     hy = thee->hy;
01430     hzed = thee->hzed;
01431     xmin = thee->xmin;
01432     ymin = thee->ymin;
01433     zmin = thee->zmin;
01434
01435     sum = 0.0;
01436     for (k=0; k<nz; k++) {
01437         pt[2] = k*hzed + zmin;
01438         for (j=0; j<ny; j++) {
01439             pt[1] = j*hy + ymin;
01440             for (i=0; i<nx; i++) {
01441                 pt[0] = i*hx + xmin;
01442                 VASSERT(Vgrid_gradient(thee, pt, grad));
01443                 for (d=0; d<3; d++) sum = sum + VSQR(grad[d]);
01444             }
01445         }
01446     }
```

```
01447
01448     sum = sum*(hx)*(hy)*(hzed);
01449
01450     if (VABS(sum) < VSMALL) sum = 0.0;
01451     else sum = VSQRT(sum);
01452
01453     return sum;
01454
01455 }
01456
01457 VPUBLIC double Vgrid_normH1(Vgrid *thee) {
01458
01459     double sum = 0.0;
01460
01461     if (thee == VNULL) {
01462         Vnm_print(2, "Vgrid_normH1:  Got VNULL thee!\n");
01463         VASSERT(0);
01464     }
01465
01466     sum = VSQR(Vgrid_seminormH1(thee)) + VSQR(Vgrid_normL2(thee));
01467
01468     return VSQRT(sum);
01469
01470 }
01471
01472 VPUBLIC double Vgrid_normLinf(Vgrid *thee) {
01473
01474     int i, j, k, nx, ny, nz, gotval;
01475     double sum, val;
01476
01477     if (thee == VNULL) {
01478         Vnm_print(2, "Vgrid_normLinf:  Got VNULL thee!\n");
01479         VASSERT(0);
01480     }
01481
01482     nx = thee->nx;
01483     ny = thee->ny;
01484     nz = thee->nz;
01485
01486     sum = 0.0;
01487     gotval = 0;
01488     for (k=0; k<nz; k++) {
01489         for (j=0; j<ny; j++) {
01490             for (i=0; i<nx; i++) {
01491                 val = VABS(thee->data[IJK(i,j,k)]);
01492                 if (!gotval) {
01493                     gotval = 1;
01494                     sum = val;
01495                 }
01496                 if (val > sum) sum = val;
01497             }
01498         }
01499     }
01500
01501     return sum;
01502
01503 }
```

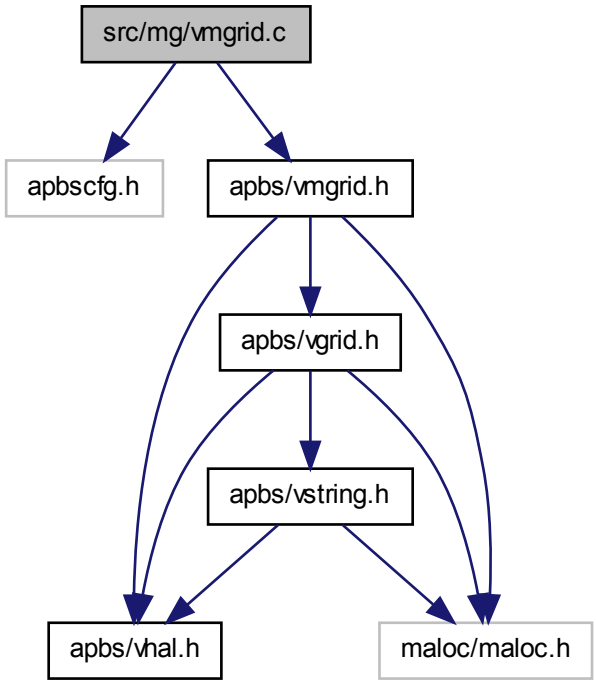
```
01504
```

10.89 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)

  FORTRAN 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.

10.89.1 Detailed Description

Class Vmgrid methods.

Author

Nathan Baker

Version

Id:

vmgrid.c 1552 2010-02-10 17:46:27Z yhuang01

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
```

Definition in file vmgrid.c.

10.90 src/mg/vmgrid.c

```
00001
00049 #include "apbscfg.h"
00050 #include "apbs/vmgrid.h"
00051
00052 VEMBED(rcsid="$Id: vmgrid.c 1552 2010-02-10 17:46:27Z yhuang01 $")
00053
00054 /* ////////////////////////////////////////////////////////////////////////////
00055 // Routine:  Vmgrid_ctor
00056 // Author:   Nathan Baker
00058 VPUBLIC Vmgrid* Vmgrid_ctor() {
00059
00060     Vmgrid *thee = VNULL;
00061
00062     thee = Vmem_malloc(VNULL, 1, sizeof(Vmgrid));
```

```
00063     VASSERT(thee != VNULL);
00064     VASSERT(Vmgrid_ctor2(thee));
00065
00066     return thee;
00067 }
00068
00069 /* ///////////////////////////////////////////////////////////////////////////
00070 // Routine:  Vmgrid_ctor2
00071 // Author:   Nathan Baker
00073 VPUBLIC int Vmgrid_ctor2(Vmgrid *thee) {
00074
00075     int i;
00076
00077     if (thee == VNULL) return 0;
00078
00079     thee->ngrids = 0;
00080     for (i=0; i<VMGRIDMAX; i++) thee->grids[i] = VNULL;
00081
00082     return 1;
00083 }
00084
00085 /* ///////////////////////////////////////////////////////////////////////////
00086 // Routine:  Vmgrid_dtor
00087 // Author:   Nathan Baker
00089 VPUBLIC void Vmgrid_dtor(Vmgrid **thee) {
00090
00091     if ((*thee) != VNULL) {
00092         Vmgrid_dtor2(*thee);
00093         Vmem_free(VNULL, 1, sizeof(Vmgrid), (void **)thee);
00094         (*thee) = VNULL;
00095     }
00096 }
00097
00098 /* ///////////////////////////////////////////////////////////////////////////
00099 // Routine:  Vmgrid_dtor2
00100 // Author:   Nathan Baker
00102 VPUBLIC void Vmgrid_dtor2(Vmgrid *thee) { ; }
00103
00104 /* ///////////////////////////////////////////////////////////////////////////
00105 // Routine:  Vmgrid_value
00106 // Author:   Nathan Baker
00108 VPUBLIC int Vmgrid_value(Vmgrid *thee, double pt[3], double *value) {
00109
00110     int i, rc;
00111     double tvalue;
00112
00113     VASSERT(thee != VNULL);
00114
00115     for (i=0; i<thee->ngrids; i++) {
00116         rc = Vgrid_value(thee->grids[i], pt, &tvalue);
00117         if (rc) {
00118             *value = tvalue;
00119             return 1;
00120         }
00121     }
00122
00123     Vnm_print(2, "Vmgrid_value:  Point (%g, %g, %g) not found in \
```

```
00124 hiearchy!\n", pt[0], pt[1], pt[2]);
00125
00126     return 0;
00127 }
00128
00129 /* ////////////////////////////////////////////////////////////////////////
00130 // Routine:  Vmgrid_curvature
00131 //
00132 //   Notes:  cflag=0 ==> Reduced Maximal Curvature
00133 //           cflag=1 ==> Mean Curvature (Laplace)
00134 //           cflag=2 ==> Gauss Curvature
00135 //           cflag=3 ==> True Maximal Curvature
00136 //
00137 // Authors:  Nathan Baker
00139 VPUBLIC int Vmgrid_curvature(Vmgrid *thee, double pt[3], int cflag,
00140   double *value) {
00141
00142     int i, rc;
00143     double tvalue;
00144
00145     VASSERT(thee != VNULL);
00146
00147     for (i=0; i<thee->ngrids; i++) {
00148         rc = Vgrid_curvature(thee->grids[i], pt, cflag, &tvalue);
00149         if (rc) {
00150             *value = tvalue;
00151             return 1;
00152         }
00153     }
00154
00155     Vnm_print(2, "Vmgrid_curvature:  Point (%g, %g, %g) not found in \
00156 hiearchy!\n", pt[0], pt[1], pt[2]);
00157
00158     return 0;
00159
00160
00161 }
00162
00163 /* ////////////////////////////////////////////////////////////////////////
00164 // Routine:  Vmgrid_gradient
00165 //
00166 // Authors:  Nathan Baker
00168 VPUBLIC int Vmgrid_gradient(Vmgrid *thee, double pt[3], double grad[3]) {
00169
00170     int i, j, rc;
00171     double tgrad[3];
00172
00173     VASSERT(thee != VNULL);
00174
00175     for (i=0; i<thee->ngrids; i++) {
00176         rc = Vgrid_gradient(thee->grids[i], pt, tgrad);
00177         if (rc) {
00178             for (j=0; j<3; j++) grad[j] = tgrad[j];
00179             return 1;
00180         }
00181     }
00182
```

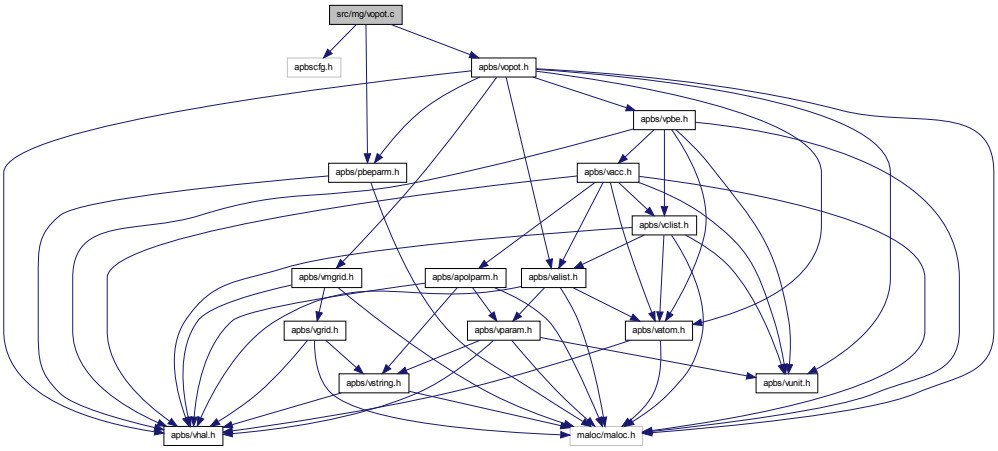
```
00183     Vnm_print(2, "Vmgrid_gradient:  Point (%g, %g, %g) not found in \
00184 hiearchy!\n", pt[0], pt[1], pt[2]);
00185
00186     return 0;
00187
00188
00189 }
00190
00191 /* ////////////////////////////////////////////////////////////////////////
00192 // Routine:  Vmgrid_addGrid
00193 //
00194 // Authors:  Nathan Baker
00196 VPUBLIC int Vmgrid_addGrid(Vmgrid *thee, Vgrid *grid) {
00197
00198     int i, j, rc;
00199     double tgrad[3];
00200
00201     VASSERT(thee != VNULL);
00202
00203     if (grid == VNULL) {
00204         Vnm_print(2, "Vmgrid_addGrid:  Not adding VNULL grid!\n");
00205         return 0;
00206     }
00207
00208     if (thee->ngrids >= VMGRIDMAX) {
00209         Vnm_print(2, "Vmgrid_addGrid:  Too many grids in hierarchy (max = \
00210 %d)!\n", VMGRIDMAX);
00211         Vnm_print(2, "Vmgrid_addGrid:  Not adding grid!\n");
00212         return 0;
00213     }
00214
00215     thee->grids[thee->ngrids] = grid;
00216     (thee->ngrids)++;
00217
00218     return 1;
00219
00220 }
```

10.91 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.

10.91.1 Detailed Description

Class Vopot methods.

Author

Nathan Baker

Version

Id:

vopot.c 1552 2010-02-10 17:46:27Z yhuang01

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2010, Washi
* 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 Washington University in St. Louis 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.

10.92 src/mg/vopot.c

```
00001
00049 #include "apbscfg.h"
00050 #include "apbs/vopot.h"
00051 #include "apbs/pbeparm.h"
00052
00053 VEMBED(rcsid="$Id: vopot.c 1552 2010-02-10 17:46:27Z yhuang01 $")
00054
00055 /* ////////////////////////////////////////////////////////////////////////////
00056 // Routine:  Vopot_ctor
00057 // Author:   Nathan Baker
00059 VPUBLIC Vopot* Vopot_ctor(Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl) {
00060
00061     Vopot *thee = VNULL;
00062
00063     thee = Vmem_malloc(VNULL, 1, sizeof(Vopot));
00064     VASSERT(thee != VNULL);
00065     VASSERT(Vopot_ctor2(thee, mgrid, pbe, bcfl));
00066
00067     return thee;
00068 }
00069
00070 /* ////////////////////////////////////////////////////////////////////////////
00071 // Routine:  Vopot_ctor2
00072 // Author:   Nathan Baker
00074 VPUBLIC int Vopot_ctor2(Vopot *thee, Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl) {
00075
00076     if (thee == VNULL) return 0;
00077     thee->bcfl = bcfl;
00078     thee->mgrid = mgrid;
00079     thee->pbe = pbe;
00080
00081     return 1;
00082 }
00083
00084 /* ////////////////////////////////////////////////////////////////////////////
00085 // Routine:  Vopot_dtor
00086 // Author:   Nathan Baker
00088 VPUBLIC void Vopot_dtor(Vopot **thee) {
00089
00090     if ((*thee) != VNULL) {
```

```
00091         Vopot_dtor2(*thee);
00092         Vmem_free(VNULL, 1, sizeof(Vopot), (void **)thee);
00093         (*thee) = VNULL;
00094     }
00095 }
00096
00097 /* ////////////////////////////////////////////////////////////////////////////
00098 // Routine:  Vopot_dtor2
00099 // Author:   Nathan Baker
00101 VPUBLIC void Vopot_dtor2(Vopot *thee) { return; }
00102
00103 /* ////////////////////////////////////////////////////////////////////////////
00104 // Routine:  Vopot_pot
00105 // Author:   Nathan Baker
00107 #define IJK(i,j,k)  (((k)*(nx)*(ny))+((j)*(nx))+(i))
00108 VPUBLIC int Vopot_pot(Vopot *thee, double pt[3], double *value) {
00109
00110     Vatom *atom;
00111     int i, iatom;
00112     double u, T, charge, eps_w, xkappa, dist, size, val, *position;
00113     Valist *alist;
00114
00115     VASSERT(thee != VNULL);
00116
00117     eps_w = Vpbe_getSolventDiel(thee->pbe);
00118     xkappa = (1.0e10)*Vpbe_getXkappa(thee->pbe);
00119     T = Vpbe_getTemperature(thee->pbe);
00120     alist = Vpbe_getValist(thee->pbe);
00121
00122     u = 0;
00123
00124     /* See if we're on the mesh */
00125     if (Vmgrid_value(thee->mgrid, pt, &u)) {
00126
00127         *value = u;
00128
00129     } else {
00130
00131         switch (thee->bcfl) {
00132
00133             case BCFL_ZERO:
00134                 u = 0;
00135                 break;
00136
00137             case BCFL_SDH:
00138                 size = (1.0e-10)*Vpbe_getSoluteRadius(thee->pbe);
00139                 position = Vpbe_getSoluteCenter(thee->pbe);
00140                 charge = Vunit_ec*Vpbe_getSoluteCharge(thee->pbe);
00141                 dist = 0;
00142                 for (i=0; i<3; i++)
00143                   dist += VSQR(position[i] - pt[i]);
00144                 dist = (1.0e-10)*VSQRT(dist);
00145                 val = (charge)/(4*VPI*Vunit_eps0*eps_w*dist);
00146                 if (xkappa != 0.0)
00147                   val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00148                 val = val*Vunit_ec/(Vunit_kb*T);
00149                 u = val;
```

```
00150                 break;
00151
00152             case BCFL_MDH:
00153                 u = 0;
00154                 for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
00155                     atom = Valist_getAtom(alist, iatom);
00156                     position = Vatom_getPosition(atom);
00157                     charge = Vunit_ec*Vatom_getCharge(atom);
00158                     size = (1e-10)*Vatom_getRadius(atom);
00159                     dist = 0;
00160                     for (i=0; i<3; i++)
00161                       dist += VSQR(position[i] - pt[i]);
00162                     dist = (1.0e-10)*VSQRT(dist);
00163                     val = (charge)/(4*VPI*Vunit_eps0*eps_w*dist);
00164                     if (xkappa != 0.0)
00165                       val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00166                     val = val*Vunit_ec/(Vunit_kb*T);
00167                     u = u + val;
00168                 }
00169                 break;
00170
00171             case BCFL_UNUSED:
00172                 Vnm_print(2, "Vopot_pot:  Invalid bcfl flag (%d)!\n",
00173                   thee->bcfl);
00174                 return 0;
00175
00176             case BCFL_FOCUS:
00177                 Vnm_print(2, "Vopot_pot:  Invalid bcfl flag (%d)!\n",
00178                   thee->bcfl);
00179                 return 0;
00180
00181             default:
00182                 Vnm_print(2, "Vopot_pot:  Bogus thee->bcfl flag (%d)!\n",
00183                   thee->bcfl);
00184                 return 0;
00185                 break;
00186         }
00187
00188         *value = u;
00189
00190     }
00191
00192     return 1;
00193
00194 }
00195
00196 /* ///////////////////////////////////////////////////////////////////////////
00197 // Routine:  Vopot_curvature
00198 //
00199 //   Notes:  cflag=0 ==> Reduced Maximal Curvature
00200 //           cflag=1 ==> Mean Curvature (Laplace)
00201 //           cflag=2 ==> Gauss Curvature
00202 //           cflag=3 ==> True Maximal Curvature
00203 //   If we are off the grid, we can still evaluate the Laplacian; assuming, we
00204 //   are away from the molecular surface, it is simply equal to the DH factor.
00205 //
00206 // Authors:  Nathan Baker
```

```
00208 VPUBLIC int Vopot_curvature(Vopot *thee, double pt[3], int cflag,
00209   double *value) {
00210
00211     Vatom *atom;
00212     int i, iatom;
00213     double u, T, charge, eps_w, xkappa, dist, size, val, *position, zkappa2;
00214     Valist *alist;
00215
00216     VASSERT(thee != VNULL);
00217
00218     eps_w = Vpbe_getSolventDiel(thee->pbe);
00219     xkappa = (1.0e10)*Vpbe_getXkappa(thee->pbe);
00220     zkappa2 = Vpbe_getZkappa2(thee->pbe);
00221     T = Vpbe_getTemperature(thee->pbe);
00222     alist = Vpbe_getValist(thee->pbe);
00223
00224     u = 0;
00225
00226     if (Vmgrid_curvature(thee->mgrid, pt, cflag, value)) return 1;
00227     else if (cflag != 1) {
00228         Vnm_print(2, "Vopot_curvature:  Off mesh!\n");
00229         return 1;
00230     } else {
00231
00232         switch (thee->bcfl) {
00233
00234             case BCFL_ZERO:
00235                 u = 0;
00236                 break;
00237
00238             case BCFL_SDH:
00239                 size = (1.0e-10)*Vpbe_getSoluteRadius(thee->pbe);
00240                 position = Vpbe_getSoluteCenter(thee->pbe);
00241                 charge = Vunit_ec*Vpbe_getSoluteCharge(thee->pbe);
00242                 dist = 0;
00243                 for (i=0; i<3; i++)
00244                   dist += VSQR(position[i] - pt[i]);
00245                 dist = (1.0e-10)*VSQRT(dist);
00246                 if (xkappa != 0.0)
00247                   u = zkappa2*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00248                 break;
00249
00250             case BCFL_MDH:
00251                 u = 0;
00252                 for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
00253                     atom = Valist_getAtom(alist, iatom);
00254                     position = Vatom_getPosition(atom);
00255                     charge = Vunit_ec*Vatom_getCharge(atom);
00256                     size = (1e-10)*Vatom_getRadius(atom);
00257                     dist = 0;
00258                     for (i=0; i<3; i++)
00259                       dist += VSQR(position[i] - pt[i]);
00260                     dist = (1.0e-10)*VSQRT(dist);
00261                     if (xkappa != 0.0)
00262                       val = zkappa2*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00263                     u = u + val;
00264                 }
```

```
00265                 break;
00266
00267             case BCFL_UNUSED:
00268                 Vnm_print(2, "Vopot_pot:  Invlid bcfl (%d)!\n", thee->bcfl);
00269                 return 0;
00270
00271             case BCFL_FOCUS:
00272                 Vnm_print(2, "Vopot_pot:  Invlid bcfl (%d)!\n", thee->bcfl);
00273                 return 0;
00274
00275             default:
00276                 Vnm_print(2, "Vopot_pot:  Bogus thee->bcfl flag (%d)!\n",
00277                   thee->bcfl);
00278                 return 0;
00279                 break;
00280         }
00281
00282         *value = u;
00283     }
00284
00285     return 1;
00286
00287 }
00288
00289 /* ///////////////////////////////////////////////////////////////////////////
00290 // Routine:  Vopot_gradient
00291 //
00292 // Authors:  Nathan Baker
00294 VPUBLIC int Vopot_gradient(Vopot *thee, double pt[3], double grad[3]) {
00295
00296     Vatom *atom;
00297     int iatom;
00298     double T, charge, eps_w, xkappa, size, val, *position;
00299     double dx, dy, dz, dist;
00300     Valist *alist;
00301
00302     VASSERT(thee != VNULL);
00303
00304     eps_w = Vpbe_getSolventDiel(thee->pbe);
00305     xkappa = (1.0e10)*Vpbe_getXkappa(thee->pbe);
00306     T = Vpbe_getTemperature(thee->pbe);
00307     alist = Vpbe_getValist(thee->pbe);
00308
00309
00310     if (!Vmgrid_gradient(thee->mgrid, pt, grad)) {
00311
00312         switch (thee->bcfl) {
00313
00314             case BCFL_ZERO:
00315                 grad[0] = 0.0;
00316                 grad[1] = 0.0;
00317                 grad[2] = 0.0;
00318                 break;
00319
00320             case BCFL_SDH:
00321                 grad[0] = 0.0;
00322                 grad[1] = 0.0;
```

```
00323                 grad[2] = 0.0;
00324                 size = (1.0e-10)*Vpbe_getSoluteRadius(thee->pbe);
00325                 position = Vpbe_getSoluteCenter(thee->pbe);
00326                 charge = Vunit_ec*Vpbe_getSoluteCharge(thee->pbe);
00327                 dx = position[0] - pt[0];
00328                 dy = position[1] - pt[1];
00329                 dz = position[2] - pt[2];
00330                 dist = VSQR(dx) + VSQR(dy) + VSQR(dz);
00331                 dist = (1.0e-10)*VSQRT(dist);
00332                 val = (charge)/(4*VPI*Vunit_eps0*eps_w);
00333                 if (xkappa != 0.0)
00334                   val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00335                 val = val*Vunit_ec/(Vunit_kb*T);
00336                 grad[0] = val*dx/dist*(-1.0/dist/dist + xkappa/dist);
00337                 grad[1] = val*dy/dist*(-1.0/dist/dist + xkappa/dist);
00338                 grad[2] = val*dz/dist*(-1.0/dist/dist + xkappa/dist);
00339                 break;
00340 
00341             case BCFL_MDH:
00342                 grad[0] = 0.0;
00343                 grad[1] = 0.0;
00344                 grad[2] = 0.0;
00345                 for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
00346                     atom = Valist_getAtom(alist, iatom);
00347                     position = Vatom_getPosition(atom);
00348                     charge = Vunit_ec*Vatom_getCharge(atom);
00349                     size = (1e-10)*Vatom_getRadius(atom);
00350                     dx = position[0] - pt[0];
00351                     dy = position[1] - pt[1];
00352                     dz = position[2] - pt[2];
00353                     dist = VSQR(dx) + VSQR(dy) + VSQR(dz);
00354                     dist = (1.0e-10)*VSQRT(dist);
00355                     val = (charge)/(4*VPI*Vunit_eps0*eps_w);
00356                     if (xkappa != 0.0)
00357                       val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00358                     val = val*Vunit_ec/(Vunit_kb*T);
00359                     grad[0] += (val*dx/dist*(-1.0/dist/dist + xkappa/dist));
00360                     grad[1] += (val*dy/dist*(-1.0/dist/dist + xkappa/dist));
00361                     grad[2] += (val*dz/dist*(-1.0/dist/dist + xkappa/dist));
00362                 }
00363                 break;
00364 
00365             case BCFL_UNUSED:
00366                 Vnm_print(2, "Vopot:  Invalid bcfl (%d)!\n", thee->bcfl);
00367                 return 0;
00368 
00369             case BCFL_FOCUS:
00370                 Vnm_print(2, "Vopot:  Invalid bcfl (%d)!\n", thee->bcfl);
00371                 return 0;
00372 
00373             default:
00374                 Vnm_print(2, "Vopot_pot:  Bogus thee->bcfl flag (%d)!\n",
00375                   thee->bcfl);
00376                 return 0;
00377                 break;
00378         }
00379 
```

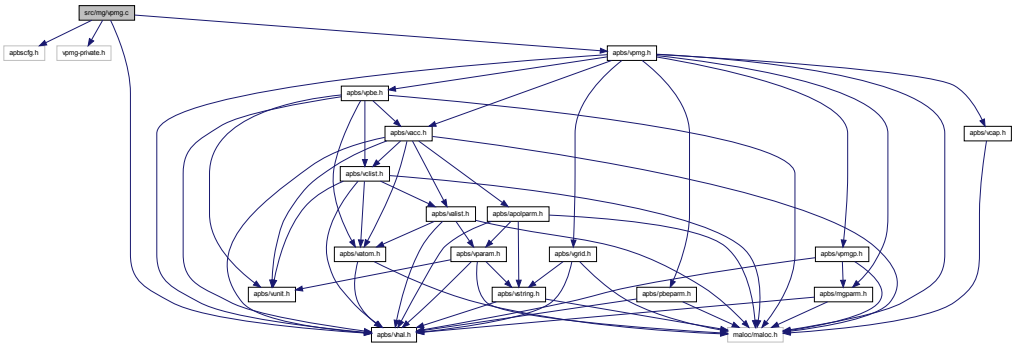
```
00380         return 1;
00381     }
00382
00383     return 1;
00384
00385 }
00386
```

10.93 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, Vpbe *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 pbetype, 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 **fillcoCoefMolIon** (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 hzed, 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)

10.93.1 Detailed Description

Class Vpmg methods.

Author

Nathan Baker

Version

Id:

vpmg.c 1611 2010-10-07 20:35:26Z yhuang01

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
* 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 Washington University in St. Louis 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.

10.94 src/mg/vpmg.c

```
00001
00049 #include "apbscfg.h"
00050 #include "vpmg-private.h"
00051 #include "apbs/vpmg.h"
00052 #include "apbs/vhal.h"
00053
00054 VEMBED(rcsid="$Id: vpmg.c 1611 2010-10-07 20:35:26Z yhuang01 $")
00055
00056 #if !defined(VINLINE_VPMG)
00057
00058 VPUBLIC unsigned long int Vpmg_memChk(Vpmg *thee) {
00059     if (thee == VNULL) return 0;
00060     return Vmem_bytes(thee->vmem);
00061 }
00062
00063 #endif /* if !defined(VINLINE_VPMG) */
00064
00065
00066 VPUBLIC void Vpmg_printColComp(Vpmg *thee, char path[72], char title[72],
00067   char mxtype[3], int flag) {
00068
00069     int nn, nxm2, nym2, nzm2, ncol, nrow, nonz;
00070     double *nzval;
00071     int *colptr, *rowind;
00072
00073     /* Calculate the total number of unknowns */
00074     nxm2 = thee->pmgp->nx - 2;
00075     nym2 = thee->pmgp->ny - 2;
00076     nzm2 = thee->pmgp->nz - 2;
```

```
00077     nn = nxm2*nym2*nzm2;
00078     ncol = nn;
00079     nrow = nn;
00080
00081     /* Calculate the number of non-zero matrix entries:
00082      *    nn       nonzeros on diagonal
00083      *    nn-1     nonzeros on first off-diagonal
00084      *    nn-nx    nonzeros on second off-diagonal
00085      *    nn-nx*ny nonzeros on third off-diagonal
00086      *
00087      *    7*nn-2*nx*ny-2*nx-2 TOTAL non-zeros
00088      */
00089     nonz = 7*nn - 2*nxm2*nym2 - 2*nxm2 - 2;
00090     nzval  = Vmem_malloc(thee->vmem, nonz, sizeof(double));
00091     rowind = Vmem_malloc(thee->vmem, nonz, sizeof(int));
00092     colptr = Vmem_malloc(thee->vmem, (ncol+1), sizeof(int));
00093
00094 #ifndef VAPBSQUIET
00095     Vnm_print(1, "Vpmg_printColComp:  Allocated space for %d nonzeros\n",
00096       nonz);
00097 #endif
00098
00099     F77BCOLCOMP(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00100       nzval, rowind, colptr, &flag);
00101
00102 #if 0
00103     for (i=0; i<nn; i++) {
00104         Vnm_print(1, "nnz(%d) = %g\n", i, nzval[i]);
00105     }
00106 #endif
00107
00108     /* I do not understand why I need to pass nzval in this way, but it
00109      * works... */
00110     F77PCOLCOMP(&nrow, &ncol, &nonz, &(nzval[0]), rowind, colptr, path, title,
00111       mxtype);
00112
00113     Vmem_free(thee->vmem, (ncol+1), sizeof(int), (void **)&colptr);
00114     Vmem_free(thee->vmem, nonz, sizeof(int), (void **)&rowind);
00115     Vmem_free(thee->vmem, nonz, sizeof(double), (void **)&nzval);
00116
00117 }
00118
00119 VPUBLIC Vpmg* Vpmg_ctor(Vpmgp *pmgp, Vpbe *pbe, int focusFlag,
00120         Vpmg *pmgOLD, MGparm *mgparm, PBEparm_calcEnergy energyFlag) {
00121
00122     Vpmg *thee = VNULL;
00123
00124     thee = Vmem_malloc(VNULL, 1, sizeof(Vpmg) );
00125     VASSERT(thee != VNULL);
00126     VASSERT( Vpmg_ctor2(thee, pmgp, pbe, focusFlag, pmgOLD, mgparm,
00127                 energyFlag) );
00128     return thee;
00129 }
00130
00131 VPUBLIC int Vpmg_ctor2(Vpmg *thee, Vpmgp *pmgp, Vpbe *pbe, int focusFlag,
00132         Vpmg *pmgOLD, MGparm *mgparm, PBEparm_calcEnergy energyFlag) {
00133
```

```
00134     int i, j, nion;
00135     double ionConc[MAXION], ionQ[MAXION], ionRadii[MAXION], zkappa2, zks2;
00136     double ionstr, partMin[3], partMax[3];
00137
00138     /* Get the parameters */
00139     VASSERT(pmgp != VNULL);
00140     VASSERT(pbe != VNULL);
00141     thee->pmgp = pmgp;
00142     thee->pbe = pbe;
00143
00144     /* Set up the memory */
00145     thee->vmem = Vmem_ctor("APBS:VPMG");
00146
00147  /* TEMPORARY USEAQUA */
00148     /* Calculate storage requirements */
00149  if(mgparm->useAqua == 0){
00150   Vpmgp_size(thee->pmgp);
00151  }else{
00152   F77MGSZAQUA(
00153     &(thee->pmgp->mgcoar), &(thee->pmgp->mgdisc),
00154     &(thee->pmgp->mgsolv),
00155     &(thee->pmgp->nx), &(thee->pmgp->ny), &(thee->pmgp->nz),
00156     &(thee->pmgp->nlev),
00157     &(thee->pmgp->nxc), &(thee->pmgp->nyc), &(thee->pmgp->nzc),
00158     &(thee->pmgp->nf), &(thee->pmgp->nc),
00159     &(thee->pmgp->narr), &(thee->pmgp->narrc),
00160     &(thee->pmgp->n_rpc), &(thee->pmgp->n_iz), &(thee->pmgp->n_ipc),
00161     &(thee->pmgp->nrwk), &(thee->pmgp->niwk)
00162     );
00163  }
00164
00165     /* We need some additional storage if: nonlinear & newton OR cgmg */
00166  /* SMPBE Added - nonlin = 2 added since it mimics NPBE */
00167     if ( ( ((thee->pmgp->nonlin == NONLIN_NPBE) || (thee->pmgp->nonlin == NONLIN_
      SMPBE))
00168      && (thee->pmgp->meth == VSOL_Newton) ) || (thee->pmgp->meth == VSOL_CGMG) )

00169  {
00170   thee->pmgp->nrwk += (2*(thee->pmgp->nf));
00171     }
00172
00173  Vnm_print(0, "Vpmg_ctor2:  PMG chose nx = %d, ny = %d, nz = %d\n",
00174      thee->pmgp->nx, thee->pmgp->ny, thee->pmgp->nz);
00175  Vnm_print(0, "Vpmg_ctor2:  PMG chose nlev = %d\n",
00176      thee->pmgp->nlev);
00177  Vnm_print(0, "Vpmg_ctor2:  PMG chose nxc = %d, nyc = %d, nzc = %d\n",
00178      thee->pmgp->nxc, thee->pmgp->nyc, thee->pmgp->nzc);
00179  Vnm_print(0, "Vpmg_ctor2:  PMG chose nf = %d, nc = %d\n",
00180      thee->pmgp->nf, thee->pmgp->nc);
00181  Vnm_print(0, "Vpmg_ctor2:  PMG chose narr = %d, narrc = %d\n",
00182      thee->pmgp->narr, thee->pmgp->narrc);
00183  Vnm_print(0, "Vpmg_ctor2:  PMG chose n_rpc = %d, n_iz = %d, n_ipc = %d\n",
00184      thee->pmgp->n_rpc, thee->pmgp->n_iz, thee->pmgp->n_ipc);
00185  Vnm_print(0, "Vpmg_ctor2:  PMG chose nrwk = %d, niwk = %d\n",
00186      thee->pmgp->nrwk, thee->pmgp->niwk);
00187
00188  /* Allocate boundary storage */
```

```
00189  thee->gxcf = (double *)Vmem_malloc(thee->vmem,
00190             10*(thee->pmgp->ny)*(thee->pmgp->nz), sizeof(double));
00191  thee->gycf = (double *)Vmem_malloc(thee->vmem,
00192             10*(thee->pmgp->nx)*(thee->pmgp->nz), sizeof(double));
00193  thee->gzcf = (double *)Vmem_malloc(thee->vmem,
00194             10*(thee->pmgp->nx)*(thee->pmgp->ny), sizeof(double));
00195
00196  /* Warn users if they are using BCFL_MAP that
00197     we do not include external energies */
00198  if (thee->pmgp->bcfl == BCFL_MAP)
00199   Vnm_print(2,"Vpmg_ctor2: \nWarning: External energies are not used in BCFL_MAP
      calculations!\n");
00200
00201  if (focusFlag) {
00202   /* Overwrite any default or user-specified boundary condition
00203   * arguments; we are now committed to a calculation via focusing */
00204   if (thee->pmgp->bcfl != BCFL_FOCUS) {
00205    Vnm_print(2,
00206        "Vpmg_ctor2: reset boundary condition flag to BCFL_FOCUS!\n");
00207    thee->pmgp->bcfl = BCFL_FOCUS;
00208   }
00209
00210   /* Fill boundaries */
00211   Vnm_print(0, "Vpmg_ctor2:  Filling boundary with old solution!\n");
00212   focusFillBound(thee, pmgOLD);
00213
00214   /* Calculate energetic contributions from region outside focusing
00215    * domain */
00216   if (energyFlag != PCE_NO) {
00217
00218    if (mgparm->type == MCT_PARALLEL) {
00219
00220     for (j=0; j<3; j++) {
00221      partMin[j] = mgparm->partDisjCenter[j]
00222      - 0.5*mgparm->partDisjLength[j];
00223      partMax[j] = mgparm->partDisjCenter[j]
00224       + 0.5*mgparm->partDisjLength[j];
00225     }
00226
00227    } else {
00228     for (j=0; j<3; j++) {
00229      partMin[j] = mgparm->center[j] - 0.5*mgparm->glen[j];
00230      partMax[j] = mgparm->center[j] + 0.5*mgparm->glen[j];
00231     }
00232    }
00233    extEnergy(thee, pmgOLD, energyFlag, partMin, partMax,
00234        mgparm->partDisjOwnSide);
00235   }
00236
00237  } else {
00238
00239   /* Ignore external energy contributions */
00240   thee->extQmEnergy = 0;
00241   thee->extDiEnergy = 0;
00242   thee->extQfEnergy = 0;
00243  }
00244
```

```
00245  /*
00246   * TODO: Move the dtor out of here. The current ctor is done in routines.c,
00247   *       This was originally moved out to kill a memory leak. The dtor has
00248   *       has been removed from initMG and placed back here to keep memory
00249   *       usage low. killMG has been modified accordingly.
00250   */
00251  Vpmg_dtor(&pmgOLD);
00252
00253  /* Allocate partition vector storage */
00254  thee->pvec = (double *)Vmem_malloc(thee->vmem,
00255              (thee->pmgp->nx)*(thee->pmgp->ny)*(thee->pmgp->nz), sizeof(double));
00256
00257  /* Allocate remaining storage */
00258  thee->iparm = (int *)Vmem_malloc(thee->vmem, 100, sizeof(int));
00259  thee->rparm = (double *)Vmem_malloc(thee->vmem, 100, sizeof(double));
00260  thee->iwork = (int *)Vmem_malloc(thee->vmem, thee->pmgp->niwk,
00261            sizeof(int));
00262  thee->rwork = (double *)Vmem_malloc(thee->vmem, thee->pmgp->nrwk,
00263            sizeof(double));
00264  thee->charge = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00265             sizeof(double));
00266  thee->kappa = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00267            sizeof(double));
00268  thee->pot = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00269            sizeof(double));
00270  thee->epsx = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00271              sizeof(double));
00272  thee->epsy = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00273              sizeof(double));
00274  thee->epsz = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00275              sizeof(double));
00276  thee->a1cf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00277              sizeof(double));
00278  thee->a2cf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00279              sizeof(double));
00280  thee->a3cf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00281              sizeof(double));
00282  thee->ccf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00283             sizeof(double));
00284  thee->fcf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00285             sizeof(double));
00286  thee->tcf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00287             sizeof(double));
00288  thee->u = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00289           sizeof(double));
00290  thee->xf = (double *)Vmem_malloc(thee->vmem, 5*(thee->pmgp->nx),
00291            sizeof(double));
00292  thee->yf = (double *)Vmem_malloc(thee->vmem, 5*(thee->pmgp->ny),
00293            sizeof(double));
00294  thee->zf = (double *)Vmem_malloc(thee->vmem, 5*(thee->pmgp->nz),
00295            sizeof(double));
00296
00297  /* Plop some of the parameters into the iparm and rparm arrays */
00298  F77PACKMG(thee->iparm, thee->rparm, &(thee->pmgp->nrwk), &(thee->pmgp->niwk),
00299      &(thee->pmgp->nx), &(thee->pmgp->ny), &(thee->pmgp->nz),
00300      &(thee->pmgp->nlev), &(thee->pmgp->nu1), &(thee->pmgp->nu2),
00301      &(thee->pmgp->mgkey), &(thee->pmgp->itmax), &(thee->pmgp->istop),
```

```
00302       &(thee->pmgp->ipcon), &(thee->pmgp->nonlin), &(thee->pmgp->mgsmoo),
00303       &(thee->pmgp->mgprol), &(thee->pmgp->mgcoar), &(thee->pmgp->mgsolv),
00304       &(thee->pmgp->mgdisc), &(thee->pmgp->iinfo), &(thee->pmgp->errtol),
00305       &(thee->pmgp->ipkey), &(thee->pmgp->omegal), &(thee->pmgp->omegan),
00306       &(thee->pmgp->irite), &(thee->pmgp->iperf));
00307
00308
00309  /* Initialize ion concentrations and valencies in PMG routines */
00310  zkappa2 = Vpbe_getZkappa2(thee->pbe);
00311  ionstr = Vpbe_getBulkIonicStrength(thee->pbe);
00312  if (ionstr > 0.0) zks2 = 0.5/ionstr;
00313  else zks2 = 0.0;
00314  Vpbe_getIons(thee->pbe, &nion, ionConc, ionRadii, ionQ);
00315
00316  /* Currently for SMPBE type calculations we do not want to apply a scale
00317   factor to the ionConc */
00318  switch(pmgp->ipkey){
00319   case IPKEY_SMPBE:
00320    F77MYPDEFINITSMPBE(&nion, ionQ, ionConc, &pbe->smvolume,&pbe->smsize);
00321    break;
00322   case IPKEY_NPBE:
00323    /* Else adjust the inoConc by scaling factor zks2 */
00324    for (i=0; i<nion; i++) ionConc[i] = zks2 * ionConc[i];
00325    F77MYPDEFINITNPBE(&nion, ionQ, ionConc);
00326    break;
00327   case IPKEY_LPBE:
00328    /* Else adjust the inoConc by scaling factor zks2 */
00329    for (i=0; i<nion; i++) ionConc[i] = zks2 * ionConc[i];
00330    F77MYPDEFINITLPBE(&nion, ionQ, ionConc);
00331    break;
00332   default:
00333    /* Else adjust the inoConc by scaling factor zks2 */
00334    for (i=0; i<nion; i++) ionConc[i] = zks2 * ionConc[i];
00335    break;
00336  }
00337
00338  /* Set the default chargeSrc for 5th order splines */
00339  thee->chargeSrc = mgparm->chgs;
00340
00341  /* Turn off restriction of observable calculations to a specific
00342  * partition */
00343  Vpmg_unsetPart(thee);
00344
00345  /* The coefficient arrays have not been filled */
00346  thee->filled = 0;
00347
00348  return 1;
00349 }
00350
00351 VPUBLIC int Vpmg_solve(Vpmg *thee) {
00352
00353     int i, nx, ny, nz, n;
00354     double zkappa2;
00355
00356     nx = thee->pmgp->nx;
00357     ny = thee->pmgp->ny;
00358     nz = thee->pmgp->nz;
```

```
00359     n = nx*ny*nz;
00360
00361     if (!(thee->filled)) {
00362         Vnm_print(2, "Vpmg_solve:  Need to call Vpmg_fillco()!\n");
00363         return 0;
00364     }
00365
00366     /* Fill the "true solution" array */
00367     for (i=0; i<n; i++) {
00368         thee->tcf[i] = 0.0;
00369     }
00370
00371     /* Fill the RHS array */
00372     for (i=0; i<n; i++) {
00373         thee->fcf[i] = thee->charge[i];
00374     }
00375
00376     /* Fill the operator coefficient array. */
00377     for (i=0; i<n; i++) {
00378         thee->a1cf[i] = thee->epsx[i];
00379         thee->a2cf[i] = thee->epsy[i];
00380         thee->a3cf[i] = thee->epsz[i];
00381     }
00382
00383     /* Fill the nonlinear coefficient array by multiplying the kappa
00384      * accessibility array (containing values between 0 and 1) by zkappa2. */
00385     zkappa2 = Vpbe_getZkappa2(thee->pbe);
00386     if (zkappa2 > VPMGSMALL) {
00387         for (i=0; i<n; i++) {
00388             thee->ccf[i] = zkappa2*thee->kappa[i];
00389         }
00390     } else {
00391         for (i=0; i<n; i++) {
00392             thee->ccf[i] = 0.0;
00393         }
00394     }
00395
00396     switch(thee->pmgp->meth) {
00397         /* CGMG (linear) */
00398         case VSOL_CGMG:
00399             F77CGMGDRIV(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00400       thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00401       thee->gzcf, thee->a1cf, thee->a2cf, thee->a3cf, thee->ccf,
00402       thee->fcf, thee->tcf);
00403             break;
00404    /* Newton (nonlinear) */
00405         case VSOL_Newton:
00406             F77NEWDRIV(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00407         thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00408         thee->gzcf, thee->a1cf, thee->a2cf, thee->a3cf, thee->ccf,
00409         thee->fcf, thee->tcf);
00410             break;
00411    /* MG (linear/nonlinear) */
00412         case VSOL_MG:
00413 #if 1
00414    F77MGDRIV(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 #else
00419     mgdrivc(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00420         thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00421         thee->gzcf, thee->a1cf, thee->a2cf, thee->a3cf, thee->ccf,
00422         thee->fcf, thee->tcf);
00423 #endif
00424             break;
00425     /* CGHS (linear/nonlinear) */
00426          case VSOL_CG:
00427     F77NCGHSDRIV(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             break;
00432     /* SOR (linear/nonlinear) */
00433          case VSOL_SOR:
00434     F77NSORDRIV(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00435        thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00436        thee->gzcf, thee->a1cf, thee->a2cf, thee->a3cf, thee->ccf,
00437        thee->fcf, thee->tcf);
00438             break;
00439     /* GSRB (linear/nonlinear) */
00440          case VSOL_RBGS:
00441     F77NGSRBDRIV(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00442         thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00443         thee->gzcf, thee->a1cf, thee->a2cf, thee->a3cf, thee->ccf,
00444         thee->fcf, thee->tcf);
00445             break;
00446     /* WJAC (linear/nonlinear) */
00447          case VSOL_WJ:
00448     F77NWJACDRIV(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00449         thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00450         thee->gzcf, thee->a1cf, thee->a2cf, thee->a3cf, thee->ccf,
00451         thee->fcf, thee->tcf);
00452             break;
00453     /* RICH (linear/nonlinear) */
00454          case VSOL_Richardson:
00455     F77NRICHDRIV(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00456         thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00457         thee->gzcf, thee->a1cf, thee->a2cf, thee->a3cf, thee->ccf,
00458         thee->fcf, thee->tcf);
00459             break;
00460     /* CGMG (linear) TEMPORARY USEAQUA */
00461          case VSOL_CGMGAqua:
00462              F77CGMGDRIVAQUA(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00463        thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00464        thee->gzcf, thee->a1cf, thee->a2cf, thee->a3cf, thee->ccf,
00465        thee->fcf);
00466             break;
00467     /* Newton (nonlinear) TEMPORARY USEAQUA */
00468          case VSOL_NewtonAqua:
00469              F77NEWDRIVAQUA(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00470         thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00471         thee->gzcf, thee->a1cf, thee->a2cf, thee->a3cf, thee->ccf,
00472         thee->fcf);
```

```
00473             break;
00474         /* Error handling */
00475         default:
00476             Vnm_print(2, "Vpmg_solve: invalid solver method key (%d)\n",
00477               thee->pmgp->key);
00478             return 0;
00479             break;
00480     }
00481
00482     return 1;
00483
00484 }
00485
00486
00487 VPUBLIC void Vpmg_dtor(Vpmg **thee) {
00488
00489     if ((*thee) != VNULL) {
00490         Vpmg_dtor2(*thee);
00491         Vmem_free(VNULL, 1, sizeof(Vpmg), (void **)thee);
00492         (*thee) = VNULL;
00493     }
00494
00495 }
00496
00497 VPUBLIC void Vpmg_dtor2(Vpmg *thee) {
00498
00499     /* Clear out the FORTRAN arrays */
00500     F77MYPDEFCLEAR();
00501
00502     /* Clean up the storage */
00503     Vmem_free(thee->vmem, 100, sizeof(int), (void **)&(thee->iparm));
00504     Vmem_free(thee->vmem, 100, sizeof(double), (void **)&(thee->rparm));
00505     Vmem_free(thee->vmem, thee->pmgp->niwk, sizeof(int),
00506       (void **)&(thee->iwork));
00507     Vmem_free(thee->vmem, thee->pmgp->nrwk, sizeof(double),
00508       (void **)&(thee->rwork));
00509     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00510       (void **)&(thee->charge));
00511     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00512       (void **)&(thee->kappa));
00513  Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00514      (void **)&(thee->pot));
00515     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00516       (void **)&(thee->epsx));
00517     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00518       (void **)&(thee->epsy));
00519     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00520       (void **)&(thee->epsz));
00521     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00522       (void **)&(thee->a1cf));
00523     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00524       (void **)&(thee->a2cf));
00525     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00526       (void **)&(thee->a3cf));
00527     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00528       (void **)&(thee->ccf));
00529     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
```

```
00530       (void **)&(thee->fcf));
00531     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00532       (void **)&(thee->tcf));
00533     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00534       (void **)&(thee->u));
00535     Vmem_free(thee->vmem, 5*(thee->pmgp->nx), sizeof(double),
00536       (void **)&(thee->xf));
00537     Vmem_free(thee->vmem, 5*(thee->pmgp->ny), sizeof(double),
00538       (void **)&(thee->yf));
00539     Vmem_free(thee->vmem, 5*(thee->pmgp->nz), sizeof(double),
00540       (void **)&(thee->zf));
00541     Vmem_free(thee->vmem, 10*(thee->pmgp->ny)*(thee->pmgp->nz), sizeof(double),
00542       (void **)&(thee->gxcf));
00543     Vmem_free(thee->vmem, 10*(thee->pmgp->nx)*(thee->pmgp->nz), sizeof(double),
00544       (void **)&(thee->gycf));
00545     Vmem_free(thee->vmem, 10*(thee->pmgp->nx)*(thee->pmgp->ny), sizeof(double),
00546       (void **)&(thee->gzcf));
00547     Vmem_free(thee->vmem, (thee->pmgp->nx)*(thee->pmgp->ny)*(thee->pmgp->nz),
00548       sizeof(double), (void **)&(thee->pvec));
00549
00550     Vmem_dtor(&(thee->vmem));
00551 }
00552
00553 VPUBLIC void Vpmg_setPart(Vpmg *thee, double lowerCorner[3],
00554         double upperCorner[3], int bflags[6]) {
00555
00556     Valist *alist;
00557     Vatom *atom;
00558     int i, j, k, nx, ny, nz;
00559     double xmin, ymin, zmin, x, y, z, hx, hy, hzed, xok, yok, zok;
00560     double x0,x1,y0,y1,z0,z1;
00561
00562     nx = thee->pmgp->nx;
00563     ny = thee->pmgp->ny;
00564     nz = thee->pmgp->nz;
00565     hx = thee->pmgp->hx;
00566     hy = thee->pmgp->hy;
00567     hzed = thee->pmgp->hzed;
00568     xmin = thee->pmgp->xcent - 0.5*hx*(nx-1);
00569     ymin = thee->pmgp->ycent - 0.5*hy*(ny-1);
00570     zmin = thee->pmgp->zcent - 0.5*hzed*(nz-1);
00571
00572     xok = 0;
00573     yok = 0;
00574     zok = 0;
00575
00576     /* We need have called Vpmg_fillco first */
00577
00578     alist = thee->pbe->alist;
00579
00580     Vnm_print(0, "Vpmg_setPart:  lower corner = (%g, %g, %g)\n",
00581       lowerCorner[0], lowerCorner[1], lowerCorner[2]);
00582     Vnm_print(0, "Vpmg_setPart:  upper corner = (%g, %g, %g)\n",
00583       upperCorner[0], upperCorner[1], upperCorner[2]);
00584     Vnm_print(0, "Vpmg_setPart:  actual minima = (%g, %g, %g)\n",
00585       xmin, ymin, zmin);
00586     Vnm_print(0, "Vpmg_setPart:  actual maxima = (%g, %g, %g)\n",
```

```
00587       xmin+hx*(nx-1), ymin+hy*(ny-1), zmin+hzed*(nz-1));
00588     Vnm_print(0, "Vpmg_setPart:  bflag[FRONT] = %d\n",
00589       bflags[VAPBS_FRONT]);
00590     Vnm_print(0, "Vpmg_setPart:  bflag[BACK] = %d\n",
00591       bflags[VAPBS_BACK]);
00592     Vnm_print(0, "Vpmg_setPart:  bflag[LEFT] = %d\n",
00593       bflags[VAPBS_LEFT]);
00594     Vnm_print(0, "Vpmg_setPart:  bflag[RIGHT] = %d\n",
00595       bflags[VAPBS_RIGHT]);
00596     Vnm_print(0, "Vpmg_setPart:  bflag[UP] = %d\n",
00597       bflags[VAPBS_UP]);
00598     Vnm_print(0, "Vpmg_setPart:  bflag[DOWN] = %d\n",
00599       bflags[VAPBS_DOWN]);
00600
00601     /* Identify atoms as inside, outside, or on the border
00602        If on the border, use the bflags to determine if there
00603        is an adjacent processor - if so, this atom should be equally
00604        shared. */
00605
00606     for (i=0; i<Valist_getNumberAtoms(alist); i++) {
00607         atom = Valist_getAtom(alist, i);
00608
00609         if ((atom->position[0] < upperCorner[0]) &&
00610             (atom->position[0] > lowerCorner[0])) xok = 1;
00611         else {
00612             if ((VABS(atom->position[0] - lowerCorner[0]) < VPMGSMALL) &&
00613                 (bflags[VAPBS_LEFT] == 0)) xok = 1;
00614             else if ((VABS(atom->position[0] - lowerCorner[0]) < VPMGSMALL) &&
00615                 (bflags[VAPBS_LEFT] == 1)) xok = 0.5;
00616             else if ((VABS(atom->position[0] - upperCorner[0]) < VPMGSMALL) &&
00617                 (bflags[VAPBS_RIGHT] == 0)) xok = 1;
00618             else if ((VABS(atom->position[0] - upperCorner[0]) < VPMGSMALL) &&
00619                 (bflags[VAPBS_RIGHT] == 1)) xok = 0.5;
00620             else xok = 0;
00621         }
00622         if ((atom->position[1] < upperCorner[1]) &&
00623             (atom->position[1] > lowerCorner[1])) yok = 1;
00624         else {
00625             if ((VABS(atom->position[1] - lowerCorner[1]) < VPMGSMALL) &&
00626                 (bflags[VAPBS_BACK] == 0)) yok = 1;
00627             else if ((VABS(atom->position[1] - lowerCorner[1]) < VPMGSMALL) &&
00628                 (bflags[VAPBS_BACK] == 1)) yok = 0.5;
00629             else if ((VABS(atom->position[1] - upperCorner[1]) < VPMGSMALL) &&
00630                 (bflags[VAPBS_FRONT] == 0)) yok = 1;
00631             else if ((VABS(atom->position[1] - upperCorner[1]) < VPMGSMALL) &&
00632                 (bflags[VAPBS_FRONT] == 1)) yok = 0.5;
00633             else yok = 0;
00634         }
00635         if ((atom->position[2] < upperCorner[2]) &&
00636             (atom->position[2] > lowerCorner[2])) zok = 1;
00637         else {
00638             if ((VABS(atom->position[2] - lowerCorner[2]) < VPMGSMALL) &&
00639                 (bflags[VAPBS_DOWN] == 0)) zok = 1;
00640             else if ((VABS(atom->position[2] - lowerCorner[2]) < VPMGSMALL) &&
00641                 (bflags[VAPBS_DOWN] == 1)) zok = 0.5;
00642             else if ((VABS(atom->position[2] - upperCorner[2]) < VPMGSMALL) &&
00643                 (bflags[VAPBS_UP] == 0)) zok = 1;
```

```
00644             else if ((VABS(atom->position[2] - upperCorner[2]) < VPMGSMALL) &&
00645                 (bflags[VAPBS_UP] == 1)) zok = 0.5;
00646             else zok = 0;
00647         }
00648
00649         atom->partID = xok*yok*zok;
00650   /*
00651   Vnm_print(1, "DEBUG (%s, %d):  atom->position[0] - upperCorner[0] = %g\n",
00652       __FILE__, __LINE__, atom->position[0] - upperCorner[0]);
00653   Vnm_print(1, "DEBUG (%s, %d):  atom->position[0] - lowerCorner[0] = %g\n",
00654       __FILE__, __LINE__, atom->position[0] - lowerCorner[0]);
00655   Vnm_print(1, "DEBUG (%s, %d):  atom->position[1] - upperCorner[1] = %g\n",
00656       __FILE__, __LINE__, atom->position[1] - upperCorner[1]);
00657   Vnm_print(1, "DEBUG (%s, %d):  atom->position[1] - lowerCorner[1] = %g\n",
00658       __FILE__, __LINE__, atom->position[1] - lowerCorner[1]);
00659   Vnm_print(1, "DEBUG (%s, %d):  atom->position[2] - upperCorner[2] = %g\n",
00660       __FILE__, __LINE__, atom->position[2] - upperCorner[2]);
00661   Vnm_print(1, "DEBUG (%s, %d):  atom->position[2] - lowerCorner[0] = %g\n",
00662       __FILE__, __LINE__, atom->position[2] - lowerCorner[2]);
00663   Vnm_print(1, "DEBUG (%s, %d):  xok = %g, yok = %g, zok = %g\n",
00664       __FILE__, __LINE__, xok, yok, zok);
00665    */
00666
00667  }
00668
00669     /* Load up pvec -
00670        For all points within h{axis}/2 of a border - use a gradient
00671        to determine the pvec weight.
00672        Points on the boundary depend on the presence of an adjacent
00673        processor. */
00674
00675     for (i=0; i<(nx*ny*nz); i++) thee->pvec[i] = 0.0;
00676
00677     for (i=0; i<nx; i++) {
00678         xok = 0.0;
00679         x = i*hx + xmin;
00680         if ( (x < (upperCorner[0]-hx/2)) &&
00681              (x > (lowerCorner[0]+hx/2))
00682            ) xok = 1.0;
00683         else if ( (VABS(x - lowerCorner[0]) < VPMGSMALL) &&
00684                   (bflags[VAPBS_LEFT] == 0)) xok = 1.0;
00685         else if ((VABS(x - lowerCorner[0]) < VPMGSMALL) &&
00686                  (bflags[VAPBS_LEFT] == 1)) xok = 0.5;
00687         else if ((VABS(x - upperCorner[0]) < VPMGSMALL) &&
00688                  (bflags[VAPBS_RIGHT] == 0)) xok = 1.0;
00689         else if ((VABS(x - upperCorner[0]) < VPMGSMALL) &&
00690                  (bflags[VAPBS_RIGHT] == 1)) xok = 0.5;
00691         else if ((x > (upperCorner[0] + hx/2)) || (x < (lowerCorner[0] - hx/2)))
      xok = 0.0;
00692         else if ((x < (upperCorner[0] + hx/2)) || (x > (lowerCorner[0] - hx/2)))
      {
00693             x0 = VMAX2(x - hx/2, lowerCorner[0]);
00694             x1 = VMIN2(x + hx/2, upperCorner[0]);
00695             xok = VABS(x1-x0)/hx;
00696
00697             if (xok < 0.0) {
00698                 if (VABS(xok) < VPMGSMALL) xok = 0.0;
```

```
00699                 else {
00700                     Vnm_print(2, "Vpmg_setPart:  fell off x-interval (%1.12E)!\n"
      ,
00701                             xok);
00702                     VASSERT(0);
00703                 }
00704             }
00705             if (xok > 1.0) {
00706                 if (VABS(xok - 1.0) < VPMGSMALL) xok = 1.0;
00707                 else {
00708                     Vnm_print(2, "Vpmg_setPart:  fell off x-interval (%1.12E)!\n"
      ,
00709                             xok);
00710                     VASSERT(0);
00711                 }
00712             }
00713 
00714         } else xok = 0.0;
00715 
00716         for (j=0; j<ny; j++) {
00717             yok = 0.0;
00718             y = j*hy + ymin;
00719             if ((y < (upperCorner[1]-hy/2)) && (y > (lowerCorner[1]+hy/2))) yok =
       1.0;
00720             else if ((VABS(y - lowerCorner[1]) < VPMGSMALL) &&
00721                      (bflags[VAPBS_BACK] == 0)) yok = 1.0;
00722             else if ((VABS(y - lowerCorner[1]) < VPMGSMALL) &&
00723                      (bflags[VAPBS_BACK] == 1)) yok = 0.5;
00724             else if ((VABS(y - upperCorner[1]) < VPMGSMALL) &&
00725                      (bflags[VAPBS_FRONT] == 0)) yok = 1.0;
00726             else if ((VABS(y - upperCorner[1]) < VPMGSMALL) &&
00727                      (bflags[VAPBS_FRONT] == 1)) yok = 0.5;
00728             else if ((y > (upperCorner[1] + hy/2)) || (y < (lowerCorner[1] - hy/2
      ))) yok=0.0;
00729             else if ((y < (upperCorner[1] + hy/2)) || (y > (lowerCorner[1] - hy/2
      ))){
00730                 y0 = VMAX2(y - hy/2, lowerCorner[1]);
00731                 y1 = VMIN2(y + hy/2, upperCorner[1]);
00732                 yok = VABS(y1-y0)/hy;
00733 
00734                 if (yok < 0.0) {
00735                     if (VABS(yok) < VPMGSMALL) yok = 0.0;
00736                     else {
00737                         Vnm_print(2, "Vpmg_setPart:  fell off y-interval (%1.12E)
      !\n",
00738                                 yok);
00739                         VASSERT(0);
00740                     }
00741                 }
00742                 if (yok > 1.0) {
00743                     if (VABS(yok - 1.0) < VPMGSMALL) yok = 1.0;
00744                     else {
00745                         Vnm_print(2, "Vpmg_setPart:  fell off y-interval (%1.12E)
      !\n",
00746                                 yok);
00747                         VASSERT(0);
00748                     }
```

```
00749                 }
00750             }
00751             else yok=0.0;
00752 
00753             for (k=0; k<nz; k++) {
00754                 zok = 0.0;
00755                 z = k*hzed + zmin;
00756                 if ((z < (upperCorner[2]-hzed/2)) && (z > (lowerCorner[2]+hzed/2)
      )) zok = 1.0;
00757                 else if ((VABS(z - lowerCorner[2]) < VPMGSMALL) &&
00758                          (bflags[VAPBS_DOWN] == 0)) zok = 1.0;
00759                 else if ((VABS(z - lowerCorner[2]) < VPMGSMALL) &&
00760                          (bflags[VAPBS_DOWN] == 1)) zok = 0.5;
00761                 else if ((VABS(z - upperCorner[2]) < VPMGSMALL) &&
00762                          (bflags[VAPBS_UP] == 0)) zok = 1.0;
00763                 else if ((VABS(z - upperCorner[2]) < VPMGSMALL) &&
00764                          (bflags[VAPBS_UP] == 1)) zok = 0.5;
00765                 else if ((z > (upperCorner[2] + hzed/2)) || (z < (lowerCorner[2]
      - hzed/2))) zok=0.0;
00766                 else if ((z < (upperCorner[2] + hzed/2)) || (z > (lowerCorner[2]
      - hzed/2))){
00767                     z0 = VMAX2(z - hzed/2, lowerCorner[2]);
00768                     z1 = VMIN2(z + hzed/2, upperCorner[2]);
00769                     zok = VABS(z1-z0)/hzed;
00770 
00771                     if (zok < 0.0) {
00772                         if (VABS(zok) < VPMGSMALL) zok = 0.0;
00773                         else {
00774                             Vnm_print(2, "Vpmg_setPart:  fell off z-interval (%1.
      12E)!\n",
00775                                       zok);
00776                             VASSERT(0);
00777                         }
00778                     }
00779                     if (zok > 1.0) {
00780                         if (VABS(zok - 1.0) < VPMGSMALL) zok = 1.0;
00781                         else {
00782                             Vnm_print(2, "Vpmg_setPart:  fell off z-interval (%1.
      12E)!\n",
00783                                       zok);
00784                             VASSERT(0);
00785                         }
00786                     }
00787                 }
00788                 else zok = 0.0;
00789 
00790                 if (VABS(xok*yok*zok) < VPMGSMALL) thee->pvec[IJK(i,j,k)] = 0.0;
00791                 else thee->pvec[IJK(i,j,k)] = xok*yok*zok;
00792 
00793             }
00794         }
00795     }
00796 }
00797 
00798 VPUBLIC void Vpmg_unsetPart(Vpmg *thee) {
00799 
00800     int i, nx, ny, nz;
```

```
00801     Vatom *atom;
00802     Valist *alist;
00803
00804     VASSERT(thee != VNULL);
00805
00806     nx = thee->pmgp->nx;
00807     ny = thee->pmgp->ny;
00808     nz = thee->pmgp->nz;
00809     alist = thee->pbe->alist;
00810
00811     for (i=0; i<(nx*ny*nz); i++) thee->pvec[i] = 1;
00812     for (i=0; i<Valist_getNumberAtoms(alist); i++) {
00813         atom = Valist_getAtom(alist, i);
00814         atom->partID = 1;
00815     }
00816 }
00817
00818 VPUBLIC int Vpmg_fillArray(Vpmg *thee, double *vec, Vdata_Type type,
00819   double parm, Vhal_PBEType pbetype, PBEparm *pbeparm) {
00820
00821     Vacc *acc = VNULL;
00822     Vpbe *pbe = VNULL;
00823     Vgrid *grid = VNULL;
00824  Vatom *atoms = VNULL;
00825  Valist *alist = VNULL;
00826     double position[3], hx, hy, hzed, xmin, ymin, zmin;
00827     double grad[3], eps, epsp, epss, zmagic;
00828     int i, j, k, l, nx, ny, nz, ichop;
00829
00830     pbe = thee->pbe;
00831     acc = Vpbe_getVacc(pbe);
00832     nx = thee->pmgp->nx;
00833     ny = thee->pmgp->ny;
00834     nz = thee->pmgp->nz;
00835     hx = thee->pmgp->hx;
00836     hy = thee->pmgp->hy;
00837     hzed = thee->pmgp->hzed;
00838     xmin = thee->pmgp->xmin;
00839     ymin = thee->pmgp->ymin;
00840     zmin = thee->pmgp->zmin;
00841     epsp = Vpbe_getSoluteDiel(pbe);
00842     epss = Vpbe_getSolventDiel(pbe);
00843     zmagic = Vpbe_getZmagic(pbe);
00844
00845     if (!(thee->filled)) {
00846         Vnm_print(2, "Vpmg_fillArray:  need to call Vpmg_fillco first!\n");
00847         return 0;
00848     }
00849
00850     switch (type) {
00851
00852         case VDT_CHARGE:
00853
00854             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->charge[i]/zmagic;
00855             break;
00856
00857         case VDT_DIELX:
```

```
00858
00859             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->epsx[i];
00860             break;
00861
00862         case VDT_DIELY:
00863
00864             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->epsy[i];
00865             break;
00866
00867         case VDT_DIELZ:
00868
00869             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->epsz[i];
00870             break;
00871
00872         case VDT_KAPPA:
00873
00874             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->kappa[i];
00875             break;
00876
00877         case VDT_POT:
00878
00879             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->u[i];
00880             break;
00881
00882    case VDT_ATOMPOT:
00883     alist = thee->pbe->alist;
00884     atoms = alist[pbeparm->molid-1].atoms;
00885     grid = Vgrid_ctor(nx, ny, nz, hx, hy,
00886           hzed, xmin, ymin, zmin,thee->u);
00887     for (i=0; i<alist[pbeparm->molid-1].number;i++) {
00888      position[0] = atoms[i].position[0];
00889      position[1] = atoms[i].position[1];
00890      position[2] = atoms[i].position[2];
00891
00892      Vgrid_value(grid, position, &vec[i]);
00893     }
00894     Vgrid_dtor(&grid);
00895     break;
00896
00897         case VDT_SMOL:
00898
00899             for (k=0; k<nz; k++) {
00900                 for (j=0; j<ny; j++) {
00901                     for (i=0; i<nx; i++) {
00902
00903                         position[0] = i*hx + xmin;
00904                         position[1] = j*hy + ymin;
00905                         position[2] = k*hzed + zmin;
00906
00907                         vec[IJK(i,j,k)] = (Vacc_molAcc(acc,position,parm));
00908                     }
00909                 }
00910             }
00911             break;
00912
00913         case VDT_SSPL:
00914
```

```
00915             for (k=0; k<nz; k++) {
00916                 for (j=0; j<ny; j++) {
00917                     for (i=0; i<nx; i++) {
00918 
00919                         position[0] = i*hx + xmin;
00920                         position[1] = j*hy + ymin;
00921                         position[2] = k*hzed + zmin;
00922 
00923                         vec[IJK(i,j,k)] = Vacc_splineAcc(acc,position,parm,0);
00924                     }
00925                 }
00926             }
00927             break;
00928 
00929         case VDT_VDW:
00930 
00931             for (k=0; k<nz; k++) {
00932                 for (j=0; j<ny; j++) {
00933                     for (i=0; i<nx; i++) {
00934 
00935                         position[0] = i*hx + xmin;
00936                         position[1] = j*hy + ymin;
00937                         position[2] = k*hzed + zmin;
00938 
00939                         vec[IJK(i,j,k)] = Vacc_vdwAcc(acc,position);
00940                     }
00941                 }
00942             }
00943             break;
00944 
00945         case VDT_IVDW:
00946 
00947             for (k=0; k<nz; k++) {
00948                 for (j=0; j<ny; j++) {
00949                     for (i=0; i<nx; i++) {
00950 
00951                         position[0] = i*hx + xmin;
00952                         position[1] = j*hy + ymin;
00953                         position[2] = k*hzed + zmin;
00954 
00955                         vec[IJK(i,j,k)] = Vacc_ivdwAcc(acc,position,parm);
00956                     }
00957                 }
00958             }
00959             break;
00960 
00961         case VDT_LAP:
00962 
00963             grid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin,
00964               thee->u);
00965             for (k=0; k<nz; k++) {
00966                 for (j=0; j<ny; j++) {
00967                     for (i=0; i<nx; i++) {
00968 
00969                         if ((k==0) || (k==(nz-1)) ||
00970                             (j==0) || (j==(ny-1)) ||
00971                             (i==0) || (i==(nx-1))) {
```

```
00972
00973                               vec[IJK(i,j,k)] = 0;
00974
00975                           } else {
00976                                   position[0] = i*hx + xmin;
00977                                   position[1] = j*hy + ymin;
00978                                   position[2] = k*hzed + zmin;
00979                                   VASSERT(Vgrid_curvature(grid,position, 1,
00980                                     &(vec[IJK(i,j,k)])));
00981                           }
00982                       }
00983                   }
00984               }
00985               Vgrid_dtor(&grid);
00986               break;
00987
00988           case VDT_EDENS:
00989
00990               grid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin,
00991                 thee->u);
00992               for (k=0; k<nz; k++) {
00993                   for (j=0; j<ny; j++) {
00994                       for (i=0; i<nx; i++) {
00995
00996                           position[0] = i*hx + xmin;
00997                           position[1] = j*hy + ymin;
00998                           position[2] = k*hzed + zmin;
00999                           VASSERT(Vgrid_gradient(grid, position, grad));
01000                           eps = epsp + (epss-epsp)*Vacc_molAcc(acc, position,
01001                             pbe->solventRadius);
01002                           vec[IJK(i,j,k)] = 0.0;
01003                           for (l=0; l<3; l++)
01004                             vec[IJK(i,j,k)] += eps*VSQR(grad[l]);
01005                       }
01006                   }
01007               }
01008               Vgrid_dtor(&grid);
01009               break;
01010
01011           case VDT_NDENS:
01012
01013               for (k=0; k<nz; k++) {
01014                   for (j=0; j<ny; j++) {
01015                       for (i=0; i<nx; i++) {
01016
01017                           position[0] = i*hx + xmin;
01018                           position[1] = j*hy + ymin;
01019                           position[2] = k*hzed + zmin;
01020                           vec[IJK(i,j,k)] = 0.0;
01021                           if ( VABS(Vacc_ivdwAcc(acc,
01022                                  position, pbe->maxIonRadius) - 1.0) < VSMALL) {
01023                               for (l=0; l<pbe->numIon; l++) {
01024                                   if (pbetype == PBE_NPBE || pbetype == PBE_SMPBE /
      *  SMPBE Added */) {
01025                                       vec[IJK(i,j,k)] += (pbe->ionConc[l]
01026                                           * Vcap_exp(-pbe->ionQ[l]*thee->u[IJK(i,j,
      k)],
```

```
01027                                       &ichop));
01028                                   } else if (pbetype == PBE_LPBE){
01029                                       vec[IJK(i,j,k)] += (pbe->ionConc[l]
01030                                           * (1 - pbe->ionQ[l]*thee->u[IJK(i,j,k)]))
      ;
01031                                   }
01032                               }
01033                           }
01034                       }
01035                   }
01036               }
01037               break;
01038
01039           case VDT_QDENS:
01040
01041               for (k=0; k<nz; k++) {
01042                   for (j=0; j<ny; j++) {
01043                       for (i=0; i<nx; i++) {
01044
01045                           position[0] = i*hx + xmin;
01046                           position[1] = j*hy + ymin;
01047                           position[2] = k*hzed + zmin;
01048                           vec[IJK(i,j,k)] = 0.0;
01049                           if ( VABS(Vacc_ivdwAcc(acc,
01050                                  position, pbe->maxIonRadius) - 1.0) < VSMALL) {
01051                               for (l=0; l<pbe->numIon; l++) {
01052                                   if (pbetype == PBE_NPBE || pbetype == PBE_SMPBE /
      *  SMPBE Added */) {
01053                                       vec[IJK(i,j,k)] += (pbe->ionConc[l]
01054                                           * pbe->ionQ[l]
01055                                           * Vcap_exp(-pbe->ionQ[l]*thee->u[IJK(i,j,
      k)],
01056                                           &ichop));
01057                                   } else if (pbetype == PBE_LPBE) {
01058                                       vec[IJK(i,j,k)] += (pbe->ionConc[l]
01059                                           * pbe->ionQ[l]
01060                                           * (1 - pbe->ionQ[l]*thee->u[IJK(i,j,k)]))
      ;
01061                                   }
01062                               }
01063                           }
01064                       }
01065                   }
01066               }
01067               break;
01068
01069           default:
01070
01071               Vnm_print(2, "main:  Bogus data type (%d)!\n", type);
01072               return 0;
01073               break;
01074
01075       }
01076
01077       return 1;
01078
01079 }
```

```
01080
01081 VPRIVATE double Vpmg_polarizEnergy(Vpmg *thee, int extFlag) {
01082
01083     int i, j, k, ijk, nx, ny, nz, iatom;
01084     double xmin, ymin, zmin, x, y, z, hx, hy, hzed, epsp, lap, pt[3];
01085     double T, pre, polq, dist2, dist, energy, q;
01086     double *charge, *pos, eps_w;
01087     Vgrid *potgrid;
01088     Vpbe *pbe;
01089     Valist *alist;
01090     Vatom *atom;
01091
01092     xmin = thee->pmgp->xmin;
01093     ymin = thee->pmgp->ymin;
01094     zmin = thee->pmgp->ymin;
01095     hx = thee->pmgp->hx;
01096     hy = thee->pmgp->hy;
01097     hzed = thee->pmgp->hzed;
01098     nx = thee->pmgp->nx;
01099     ny = thee->pmgp->ny;
01100     nz = thee->pmgp->nz;
01101     pbe = thee->pbe;
01102     epsp = Vpbe_getSoluteDiel(pbe);
01103     eps_w = Vpbe_getSolventDiel(pbe);
01104     alist = pbe->alist;
01105     charge = thee->charge;
01106
01107     /* Calculate the prefactor for Coulombic calculations */
01108     T = Vpbe_getTemperature(pbe);
01109     pre = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T);
01110     pre = pre*(1.0e10);
01111
01112     /* Set up Vgrid object with solution */
01113     potgrid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin, thee->u);
01114
01115     /* Calculate polarization charge */
01116     energy = 0.0;
01117     for (i=1; i<(nx-1); i++) {
01118         pt[0] = xmin + hx*i;
01119         for (j=1; j<(ny-1); j++) {
01120             pt[1] = ymin + hy*j;
01121             for (k=1; k<(nz-1); k++) {
01122                 pt[2] = zmin + hzed*k;
01123
01124                 /* Calculate polarization charge */
01125                 VASSERT(Vgrid_curvature(potgrid, pt, 1, &lap));
01126                 ijk = IJK(i,j,k);
01127                 polq = charge[ijk] + epsp*lap*3.0;
01128
01129                 /* Calculate interaction energy with atoms */
01130                 if (VABS(polq) > VSMALL) {
01131                     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
01132                         atom = Valist_getAtom(alist, iatom);
01133                         q = Vatom_getCharge(atom);
01134                         pos = Vatom_getPosition(atom);
01135                         dist2 = VSQR(pos[0]-pt[0]) + VSQR(pos[1]-pt[1]) \
01136                                 + VSQR(pos[2]-pt[2]);
```

```
01137                         dist = VSQRT(dist2);
01138
01139                         if (dist < VSMALL) {
01140                             Vnm_print(2, "Vpmg_polarizEnergy:  atom on grid point
      ; ignoring!\n");
01141                         } else {
01142                             energy = energy + polq*q/dist;
01143                         }
01144                     }
01145                 }
01146             }
01147         }
01148     }
01149
01150     return pre*energy;
01151 }
01152
01153 VPUBLIC double Vpmg_energy(Vpmg *thee, int extFlag) {
01154
01155     double totEnergy = 0.0;
01156     double dielEnergy = 0.0;
01157     double qmEnergy = 0.0;
01158     double qfEnergy = 0.0;
01159
01160     VASSERT(thee != VNULL);
01161
01162     if ((thee->pmgp->nonlin) && (Vpbe_getBulkIonicStrength(thee->pbe) > 0.)) {
01163         Vnm_print(0, "Vpmg_energy:  calculating full PBE energy\n");
01164         qmEnergy = Vpmg_qmEnergy(thee, extFlag);
01165         Vnm_print(0, "Vpmg_energy:  qmEnergy = %1.12E kT\n", qmEnergy);
01166         qfEnergy = Vpmg_qfEnergy(thee, extFlag);
01167         Vnm_print(0, "Vpmg_energy:  qfEnergy = %1.12E kT\n", qfEnergy);
01168         dielEnergy = Vpmg_dielEnergy(thee, extFlag);
01169         Vnm_print(0, "Vpmg_energy:  dielEnergy = %1.12E kT\n", dielEnergy);
01170         totEnergy = qfEnergy - dielEnergy - qmEnergy;
01171     } else {
01172         Vnm_print(0, "Vpmg_energy:  calculating only q-phi energy\n");
01173         qfEnergy = Vpmg_qfEnergy(thee, extFlag);
01174         Vnm_print(0, "Vpmg_energy:  qfEnergy = %1.12E kT\n", qfEnergy);
01175         totEnergy = 0.5*qfEnergy;
01176     }
01177
01178     return totEnergy;
01179
01180 }
01181
01182 VPUBLIC double Vpmg_dielEnergy(Vpmg *thee, int extFlag) {
01183
01184     double hx, hy, hzed, energy, nrgx, nrgy, nrgz, pvecx, pvecy, pvecz;
01185     int i, j, k, nx, ny, nz;
01186
01187     VASSERT(thee != VNULL);
01188
01189     /* Get the mesh information */
01190     nx = thee->pmgp->nx;
01191     ny = thee->pmgp->ny;
01192     nz = thee->pmgp->nz;
```

```
01193     hx = thee->pmgp->hx;
01194     hy = thee->pmgp->hy;
01195     hzed = thee->pmgp->hzed;
01196
01197     energy = 0.0;
01198
01199     if (!thee->filled) {
01200         Vnm_print(2, "Vpmg_dielEnergy:  Need to call Vpmg_fillco!\n");
01201         VASSERT(0);
01202     }
01203
01204     for (k=0; k<(nz-1); k++) {
01205         for (j=0; j<(ny-1); j++) {
01206             for (i=0; i<(nx-1); i++) {
01207                 pvecx = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i+1,j,k)]);
01208                 pvecy = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j+1,k)]);
01209                 pvecz = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j,k+1)]);
01210                 nrgx = thee->epsx[IJK(i,j,k)]*pvecx
01211                   * VSQR((thee->u[IJK(i,j,k)]-thee->u[IJK(i+1,j,k)])/hx);
01212                 nrgy = thee->epsy[IJK(i,j,k)]*pvecy
01213                   * VSQR((thee->u[IJK(i,j,k)]-thee->u[IJK(i,j+1,k)])/hy);
01214                 nrgz = thee->epsz[IJK(i,j,k)]*pvecz
01215                   * VSQR((thee->u[IJK(i,j,k)]-thee->u[IJK(i,j,k+1)])/hzed);
01216                 energy += (nrgx + nrgy + nrgz);
01217             }
01218         }
01219     }
01220
01221     energy = 0.5*energy*hx*hy*hzed;
01222     energy = energy/Vpbe_getZmagic(thee->pbe);
01223
01224     if (extFlag == 1) energy += (thee->extDiEnergy);
01225
01226     return energy;
01227 }
01228
01229 VPUBLIC double Vpmg_dielGradNorm(Vpmg *thee) {
01230
01231     double hx, hy, hzed, energy, nrgx, nrgy, nrgz, pvecx, pvecy, pvecz;
01232     int i, j, k, nx, ny, nz;
01233
01234     VASSERT(thee != VNULL);
01235
01236     /* Get the mesh information */
01237     nx = thee->pmgp->nx;
01238     ny = thee->pmgp->ny;
01239     nz = thee->pmgp->nz;
01240     hx = thee->pmgp->hx;
01241     hy = thee->pmgp->hy;
01242     hzed = thee->pmgp->hzed;
01243
01244     energy = 0.0;
01245
01246     if (!thee->filled) {
01247         Vnm_print(2, "Vpmg_dielGradNorm:  Need to call Vpmg_fillco!\n");
01248         VASSERT(0);
01249     }
```

```
01250
01251     for (k=1; k<nz; k++) {
01252         for (j=1; j<ny; j++) {
01253             for (i=1; i<nx; i++) {
01254                 pvecx = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i-1,j,k)]);
01255                 pvecy = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j-1,k)]);
01256                 pvecz = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j,k-1)]);
01257                 nrgx = pvecx
01258                  * VSQR((thee->epsx[IJK(i,j,k)]-thee->epsx[IJK(i-1,j,k)])/hx);
01259                 nrgy = pvecy
01260                  * VSQR((thee->epsy[IJK(i,j,k)]-thee->epsy[IJK(i,j-1,k)])/hy);
01261                 nrgz = pvecz
01262                  * VSQR((thee->epsz[IJK(i,j,k)]-thee->epsz[IJK(i,j,k-1)])/hzed);
01263                 energy += VSQRT(nrgx + nrgy + nrgz);
01264             }
01265         }
01266     }
01267
01268     energy = energy*hx*hy*hzed;
01269
01270     return energy;
01271 }
01272
01273 VPUBLIC double Vpmg_qmEnergy(Vpmg *thee, int extFlag) {
01274
01275  double energy;
01276
01277  if(thee->pbe->ipkey == IPKEY_SMPBE){
01278   energy = Vpmg_qmEnergySMPBE(thee,extFlag);
01279  }else{
01280   energy = Vpmg_qmEnergyNONLIN(thee,extFlag);
01281  }
01282
01283  return energy;
01284 }
01285
01286 VPRIVATE double Vpmg_qmEnergyNONLIN(Vpmg *thee, int extFlag) {
01287
01288     double hx, hy, hzed, energy, ionConc[MAXION], ionRadii[MAXION];
01289     double ionQ[MAXION], zkappa2, ionstr, zks2;
01290     int i, j, nx, ny, nz, nion, ichop, nchop;
01291
01292     VASSERT(thee != VNULL);
01293
01294     /* Get the mesh information */
01295     nx = thee->pmgp->nx;
01296     ny = thee->pmgp->ny;
01297     nz = thee->pmgp->nz;
01298     hx = thee->pmgp->hx;
01299     hy = thee->pmgp->hy;
01300     hzed = thee->pmgp->hzed;
01301     zkappa2 = Vpbe_getZkappa2(thee->pbe);
01302     ionstr = Vpbe_getBulkIonicStrength(thee->pbe);
01303
01304     /* Bail if we're at zero ionic strength */
01305     if (zkappa2 < VSMALL) {
01306
```

```
01307 #ifndef VAPBSQUIET
01308         Vnm_print(0, "Vpmg_qmEnergy:  Zero energy for zero ionic strength!\n");
01309 #endif
01310
01311         return 0.0;
01312     }
01313     zks2 = 0.5*zkappa2/ionstr;
01314
01315     if (!thee->filled) {
01316         Vnm_print(2, "Vpmg_qmEnergy:  Need to call Vpmg_fillco()!\n");
01317         VASSERT(0);
01318     }
01319
01320     energy = 0.0;
01321     nchop = 0;
01322     Vpbe_getIons(thee->pbe, &nion, ionConc, ionRadii, ionQ);
01323     if (thee->pmgp->nonlin) {
01324         Vnm_print(0, "Vpmg_qmEnergy:  Calculating nonlinear energy\n");
01325         for (i=0; i<(nx*ny*nz); i++) {
01326             if (thee->pvec[i]*thee->kappa[i] > VSMALL) {
01327                 for (j=0; j<nion; j++) {
01328                     energy += (thee->pvec[i]*thee->kappa[i]*zks2
01329                       * ionConc[j]
01330                       * (Vcap_exp(-ionQ[j]*thee->u[i], &ichop)-1.0));
01331                     nchop += ichop;
01332                 }
01333             }
01334         }
01335         if (nchop > 0){
01336    Vnm_print(2, "Vpmg_qmEnergy:  Chopped EXP %d times!\n",nchop);
01337    Vnm_print(2, "\nERROR!  Detected large potential values in energy evaluation!
      \nERROR!  This calculation failed -- please report to the APBS developers!\n\n");

01338    VASSERT(0);
01339   }
01340     } else {
01341         /* Zkappa2 OK here b/c LPBE approx */
01342         Vnm_print(0, "Vpmg_qmEnergy:  Calculating linear energy\n");
01343         for (i=0; i<(nx*ny*nz); i++) {
01344             if (thee->pvec[i]*thee->kappa[i] > VSMALL)
01345               energy += (thee->pvec[i]*zkappa2*thee->kappa[i]*VSQR(thee->u[i]));
01346         }
01347         energy = 0.5*energy;
01348     }
01349     energy = energy*hx*hy*hzed;
01350     energy = energy/Vpbe_getZmagic(thee->pbe);
01351
01352     if (extFlag == 1) energy += thee->extQmEnergy;
01353
01354     return energy;
01355 }
01356
01357 VPUBLIC double Vpmg_qmEnergySMPBE(Vpmg *thee, int extFlag) {
01358
01359     double hx, hy, hzed, energy, ionConc[MAXION], ionRadii[MAXION];
01360     double ionQ[MAXION], zkappa2, ionstr, zks2;
01361     int i, j, nx, ny, nz, nion, ichop, nchop;
```

```
01362
01363     /* SMPB Modification (vchu, 09/21/06)*/
01364     /* variable declarations for SMPB energy terms */
01365     double a, k, z1, z2, z3, cb1, cb2, cb3;
01366     double a1, a2, a3, c1, c2, c3, currEnergy;
01367     double fracOccA, fracOccB, fracOccC, phi, gpark, denom, Na;
01368     int ichop1, ichop2, ichop3;
01369
01370     VASSERT(thee != VNULL);
01371
01372     /* Get the mesh information */
01373     nx = thee->pmgp->nx;
01374     ny = thee->pmgp->ny;
01375     nz = thee->pmgp->nz;
01376     hx = thee->pmgp->hx;
01377     hy = thee->pmgp->hy;
01378     hzed = thee->pmgp->hzed;
01379     zkappa2 = Vpbe_getZkappa2(thee->pbe);
01380     ionstr = Vpbe_getBulkIonicStrength(thee->pbe);
01381
01382     /* Bail if we're at zero ionic strength */
01383     if (zkappa2 < VSMALL) {
01384
01385 #ifndef VAPBSQUIET
01386         Vnm_print(0, "Vpmg_qmEnergySMPBE:  Zero energy for zero ionic strength!\n
      ");
01387 #endif
01388
01389         return 0.0;
01390     }
01391     zks2 = 0.5*zkappa2/ionstr;
01392
01393     if (!thee->filled) {
01394         Vnm_print(2, "Vpmg_qmEnergySMPBE:  Need to call Vpmg_fillco()!\n");
01395         VASSERT(0);
01396     }
01397
01398     energy = 0.0;
01399     nchop = 0;
01400     Vpbe_getIons(thee->pbe, &nion, ionConc, ionRadii, ionQ);
01401
01402     /* SMPB Modification (vchu, 09/21/06) */
01403     /* Extensive modification to the first part of the if statement
01404    where that handles the thee->pmgp->nonlin part. Basically, I've
01405    deleted all of the original code and written my own code that computes
01406    the electrostatic free energy in the SMPB framework. Definitely really hacky
01407    at this stage of the game, but gets the job done. The second part of the
01408    if statement (the part that handles linear poisson-boltzmann) has been deleted
01409    because there will be no linearized SMPB energy.. */
01410
01411     z1 = ionQ[0];
01412     z2 = ionQ[1];
01413     z3 = ionQ[2];
01414     cb1 = ionConc[0];
01415     cb2 = ionConc[1];
01416     cb3 = ionConc[2];
01417     a  = thee->pbe->smvolume;
```

```
01418     k  = thee->pbe->smsize;
01419     Na = 6.022045000e-04; /* Converts from Molar to N/A^3 */
01420
01421     fracOccA = Na*cb1*VCUB(a);
01422     fracOccB = Na*cb2*VCUB(a);
01423     fracOccC = Na*cb3*VCUB(a);
01424
01425     phi = (fracOccA/k) + fracOccB + fracOccC;
01426
01427     if (thee->pmgp->nonlin) {
01428         Vnm_print(0, "Vpmg_qmEnergySMPBE:  Calculating nonlinear energy using SMP
      B functional!\n");
01429         for (i=0; i<(nx*ny*nz); i++) {
01430    if (((k-1) > VSMALL) && (thee->pvec[i]*thee->kappa[i] > VSMALL)) {
01431
01432     a1 = Vcap_exp(-1.0*z1*thee->u[i], &ichop1);
01433     a2 = Vcap_exp(-1.0*z2*thee->u[i], &ichop2);
01434     a3 = Vcap_exp(-1.0*z3*thee->u[i], &ichop3);
01435
01436     nchop += ichop1 + ichop2 + ichop3;
01437
01438     gpark = (1 - phi + (fracOccA/k)*a1);
01439     denom = VPOW(gpark, k) + VPOW(1-fracOccB-fracOccC, k-1)*(fracOccB*a2+fracOccC
      *a3);
01440
01441     if (cb1 > VSMALL) {
01442      c1 = Na*cb1*VPOW(gpark, k-1)*a1/denom;
01443      if(c1 != c1) c1 = 0.;
01444     } else c1 = 0.;
01445
01446     if (cb2 > VSMALL) {
01447      c2 = Na*cb2*VPOW(1-fracOccB-fracOccC,k-1)*a2/denom;
01448      if(c2 != c2) c2 = 0.;
01449     } else c2 = 0.;
01450
01451     if (cb3 > VSMALL) {
01452      c3 = Na*cb3*VPOW(1-fracOccB-fracOccC,k-1)*a3/denom;
01453      if(c3 != c3) c3 = 0.;
01454     } else c3 = 0.;
01455
01456     currEnergy = k*VLOG((1-(c1*VCUB(a)/k)-c2*VCUB(a)-c3*VCUB(a))/(1-phi))
01457      -(k-1)*VLOG((1-c2*VCUB(a)-c3*VCUB(a))/(1-phi+(fracOccA/k)));
01458
01459     energy += thee->pvec[i]*thee->kappa[i]*currEnergy;
01460
01461    } else if (thee->pvec[i]*thee->kappa[i] > VSMALL){
01462
01463     a1 = Vcap_exp(-1.0*z1*thee->u[i], &ichop1);
01464     a2 = Vcap_exp(-1.0*z2*thee->u[i], &ichop2);
01465     a3 = Vcap_exp(-1.0*z3*thee->u[i], &ichop3);
01466
01467     nchop += ichop1 + ichop2 + ichop3;
01468
01469     gpark = (1 - phi + (fracOccA)*a1);
01470     denom = gpark + (fracOccB*a2+fracOccC*a3);
01471
01472     if (cb1 > VSMALL) {
```

```
01473       c1 = Na*cb1*a1/denom;
01474       if(c1 != c1) c1 = 0.;
01475      } else c1 = 0.;
01476
01477      if (cb2 > VSMALL) {
01478       c2 = Na*cb2*a2/denom;
01479       if(c2 != c2) c2 = 0.;
01480      } else c2 = 0.;
01481
01482      if (cb3 > VSMALL) {
01483       c3 = Na*cb3*a3/denom;
01484       if(c3 != c3) c3 = 0.;
01485      } else c3 = 0.;
01486
01487      currEnergy = VLOG((1-c1*VCUB(a)-c2*VCUB(a)-c3*VCUB(a))/(1-fracOccA-fracOccB-f
     racOccC));
01488
01489      energy += thee->pvec[i]*thee->kappa[i]*currEnergy;
01490     }
01491          }
01492
01493    energy = -energy/VCUB(a);
01494
01495          if (nchop > 0) Vnm_print(2, "Vpmg_qmEnergySMPBE:  Chopped EXP %d times!\n
     ",
01496           nchop);
01497
01498      } else {
01499          /* Zkappa2 OK here b/c LPBE approx */
01500          Vnm_print(0, "Vpmg_qmEnergySMPBE:  ERROR: NO LINEAR ENERGY!! Returning 0!
     \n");
01501
01502    energy = 0.0;
01503
01504      }
01505      energy = energy*hx*hy*hzed;
01506
01507      if (extFlag == 1) energy += thee->extQmEnergy;
01508
01509      return energy;
01510  }
01511
01512  VPUBLIC double Vpmg_qfEnergy(Vpmg *thee, int extFlag) {
01513
01514      double energy = 0.0;
01515
01516      VASSERT(thee != VNULL);
01517
01518      if ((thee->useChargeMap) || (thee->chargeMeth == VCM_BSPL2)) {
01519          energy = Vpmg_qfEnergyVolume(thee, extFlag);
01520      } else {
01521          energy = Vpmg_qfEnergyPoint(thee, extFlag);
01522      }
01523
01524      return energy;
01525  }
01526
```

```
01527 VPRIVATE double Vpmg_qfEnergyPoint(Vpmg *thee, int extFlag) {
01528
01529     int iatom, nx, ny, nz, ihi, ilo, jhi, jlo, khi, klo;
01530     double xmax, ymax, zmax, xmin, ymin, zmin, hx, hy, hzed, ifloat, jfloat;
01531     double charge, kfloat, dx, dy, dz, energy, uval, *position;
01532     double *u;
01533     double *pvec;
01534     Valist *alist;
01535     Vatom *atom;
01536     Vpbe *pbe;
01537
01538     pbe = thee->pbe;
01539     alist = pbe->alist;
01540     VASSERT(alist != VNULL);
01541
01542     /* Get the mesh information */
01543     nx = thee->pmgp->nx;
01544     ny = thee->pmgp->ny;
01545     nz = thee->pmgp->nz;
01546     hx = thee->pmgp->hx;
01547     hy = thee->pmgp->hy;
01548     hzed = thee->pmgp->hzed;
01549     xmax = thee->pmgp->xmax;
01550     ymax = thee->pmgp->ymax;
01551     zmax = thee->pmgp->zmax;
01552     xmin = thee->pmgp->xmin;
01553     ymin = thee->pmgp->ymin;
01554     zmin = thee->pmgp->zmin;
01555
01556     u = thee->u;
01557     pvec = thee->pvec;
01558
01559     energy = 0.0;
01560
01561     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
01562
01563         /* Get atomic information */
01564         atom = Valist_getAtom(alist, iatom);
01565
01566         position = Vatom_getPosition(atom);
01567         charge = Vatom_getCharge(atom);
01568
01569         /* Figure out which vertices we're next to */
01570         ifloat = (position[0] - xmin)/hx;
01571         jfloat = (position[1] - ymin)/hy;
01572         kfloat = (position[2] - zmin)/hzed;
01573         ihi = (int)ceil(ifloat);
01574         ilo = (int)floor(ifloat);
01575         jhi = (int)ceil(jfloat);
01576         jlo = (int)floor(jfloat);
01577         khi = (int)ceil(kfloat);
01578         klo = (int)floor(kfloat);
01579
01580         if (atom->partID > 0) {
01581
01582             if ((ihi<nx) && (jhi<ny) && (khi<nz) &&
01583                 (ilo>=0) && (jlo>=0) && (klo>=0)) {
```

```
01584
01585                 /* Now get trilinear interpolation constants */
01586                 dx = ifloat - (double)(ilo);
01587                 dy = jfloat - (double)(jlo);
01588                 dz = kfloat - (double)(klo);
01589                 uval =
01590                   dx*dy*dz*u[IJK(ihi,jhi,khi)]
01591                 + dx*(1.0-dy)*dz*u[IJK(ihi,jlo,khi)]
01592                 + dx*dy*(1.0-dz)*u[IJK(ihi,jhi,klo)]
01593                 + dx*(1.0-dy)*(1.0-dz)*u[IJK(ihi,jlo,klo)]
01594                 + (1.0-dx)*dy*dz*u[IJK(ilo,jhi,khi)]
01595                 + (1.0-dx)*(1.0-dy)*dz*u[IJK(ilo,jlo,khi)]
01596                 + (1.0-dx)*dy*(1.0-dz)*u[IJK(ilo,jhi,klo)]
01597                 + (1.0-dx)*(1.0-dy)*(1.0-dz)*u[IJK(ilo,jlo,klo)];
01598                 energy += (uval*charge*atom->partID);
01599             } else if (thee->pmgp->bcfl != BCFL_FOCUS) {
01600                 Vnm_print(2, "Vpmg_qfEnergy:  Atom #%d at (%4.3f, %4.3f, \
01601 %4.3f) is off the mesh (ignoring)!\n",
01602                 iatom, position[0], position[1], position[2]);
01603             }
01604         }
01605     }
01606
01607     if (extFlag) energy += thee->extQfEnergy;
01608
01609     return energy;
01610 }
01611
01612 VPUBLIC double Vpmg_qfAtomEnergy(Vpmg *thee, Vatom *atom) {
01613
01614     int nx, ny, nz, ihi, ilo, jhi, jlo, khi, klo;
01615     double xmax, xmin, ymax, ymin, zmax, zmin, hx, hy, hzed, ifloat, jfloat;
01616     double charge, kfloat, dx, dy, dz, energy, uval, *position;
01617     double *u;
01618
01619
01620     /* Get the mesh information */
01621     nx = thee->pmgp->nx;
01622     ny = thee->pmgp->ny;
01623     nz = thee->pmgp->nz;
01624     hx = thee->pmgp->hx;
01625     hy = thee->pmgp->hy;
01626     hzed = thee->pmgp->hzed;
01627     xmax = thee->xf[nx-1];
01628     ymax = thee->yf[ny-1];
01629     zmax = thee->zf[nz-1];
01630     xmin = thee->xf[0];
01631     ymin = thee->yf[0];
01632     zmin = thee->zf[0];
01633
01634     u = thee->u;
01635
01636     energy = 0.0;
01637
01638
01639     position = Vatom_getPosition(atom);
01640     charge = Vatom_getCharge(atom);
```

```
01641
01642     /* Figure out which vertices we're next to */
01643     ifloat = (position[0] - xmin)/hx;
01644     jfloat = (position[1] - ymin)/hy;
01645     kfloat = (position[2] - zmin)/hzed;
01646     ihi = (int)ceil(ifloat);
01647     ilo = (int)floor(ifloat);
01648     jhi = (int)ceil(jfloat);
01649     jlo = (int)floor(jfloat);
01650     khi = (int)ceil(kfloat);
01651     klo = (int)floor(kfloat);
01652
01653     if (atom->partID > 0) {
01654
01655         if ((ihi<nx) && (jhi<ny) && (khi<nz) &&
01656             (ilo>=0) && (jlo>=0) && (klo>=0)) {
01657
01658             /* Now get trilinear interpolation constants */
01659             dx = ifloat - (double)(ilo);
01660             dy = jfloat - (double)(jlo);
01661             dz = kfloat - (double)(klo);
01662             uval =
01663               dx*dy*dz*u[IJK(ihi,jhi,khi)]
01664             + dx*(1.0-dy)*dz*u[IJK(ihi,jlo,khi)]
01665             + dx*dy*(1.0-dz)*u[IJK(ihi,jhi,klo)]
01666             + dx*(1.0-dy)*(1.0-dz)*u[IJK(ihi,jlo,klo)]
01667             + (1.0-dx)*dy*dz*u[IJK(ilo,jhi,khi)]
01668             + (1.0-dx)*(1.0-dy)*dz*u[IJK(ilo,jlo,khi)]
01669             + (1.0-dx)*dy*(1.0-dz)*u[IJK(ilo,jhi,klo)]
01670             + (1.0-dx)*(1.0-dy)*(1.0-dz)*u[IJK(ilo,jlo,klo)];
01671             energy += (uval*charge*atom->partID);
01672         } else if (thee->pmgp->bcfl != BCFL_FOCUS) {
01673             Vnm_print(2, "Vpmg_qfAtomEnergy:  Atom at (%4.3f, %4.3f, \
01674 %4.3f) is off the mesh (ignoring)!\n",
01675             position[0], position[1], position[2]);
01676         }
01677     }
01678
01679     return energy;
01680 }
01681
01682 VPRIVATE double Vpmg_qfEnergyVolume(Vpmg *thee, int extFlag) {
01683
01684     double hx, hy, hzed, energy;
01685     int i, nx, ny, nz;
01686
01687     VASSERT(thee != VNULL);
01688
01689     /* Get the mesh information */
01690     nx = thee->pmgp->nx;
01691     ny = thee->pmgp->ny;
01692     nz = thee->pmgp->nz;
01693     hx = thee->pmgp->hx;
01694     hy = thee->pmgp->hy;
01695     hzed = thee->pmgp->hzed;
01696
01697     if (!thee->filled) {
```

```
01698         Vnm_print(2, "Vpmg_qfEnergyVolume:  need to call Vpmg_fillco!\n");
01699         VASSERT(0);
01700     }
01701
01702     energy = 0.0;
01703     Vnm_print(0, "Vpmg_qfEnergyVolume:  Calculating energy\n");
01704     for (i=0; i<(nx*ny*nz); i++) {
01705         energy += (thee->pvec[i]*thee->u[i]*thee->charge[i]);
01706     }
01707     energy = energy*hx*hy*hzed/Vpbe_getZmagic(thee->pbe);
01708
01709     if (extFlag == 1) energy += thee->extQfEnergy;
01710
01711     return energy;
01712 }
01713
01714 VPRIVATE void Vpmg_splineSelect(int srfm,Vacc *acc,double *gpos,double win,
01715           double infrad,Vatom *atom,double *force){
01716
01717  switch (srfm) {
01718   case VSM_SPLINE :
01719    Vacc_splineAccGradAtomNorm(acc, gpos, win, infrad, atom, force);
01720    break;
01721   case VSM_SPLINE3:
01722    Vacc_splineAccGradAtomNorm3(acc, gpos, win, infrad, atom, force);
01723    break;
01724   case VSM_SPLINE4 :
01725    Vacc_splineAccGradAtomNorm4(acc, gpos, win, infrad, atom, force);
01726    break;
01727   default:
01728    Vnm_print(2, "Vpmg_dbnbForce: Unknown surface method.\n");
01729    return;
01730  }
01731
01732  return;
01733 }
01734
01735 VPRIVATE void focusFillBound(Vpmg *thee, Vpmg *pmgOLD) {
01736
01737     Vpbe *pbe;
01738     double hxOLD, hyOLD, hzOLD, xminOLD, yminOLD, zminOLD, xmaxOLD, ymaxOLD;
01739     double zmaxOLD;
01740     int nxOLD, nyOLD, nzOLD;
01741     double hxNEW, hyNEW, hzNEW, xminNEW, yminNEW, zminNEW, xmaxNEW, ymaxNEW;
01742     double zmaxNEW;
01743     int nxNEW, nyNEW, nzNEW;
01744     int i, j, k, ihi, ilo, jhi, jlo, khi, klo, nx, ny, nz;
01745     double x, y, z, dx, dy, dz, ifloat, jfloat, kfloat, uval;
01746     double eps_w, T, pre1, xkappa, size, *apos, charge, pos[3];
01747
01748  double uvalMin, uvalMax;
01749  double *data;
01750
01751     /* Calculate new problem dimensions */
01752     hxNEW = thee->pmgp->hx;
01753     hyNEW = thee->pmgp->hy;
01754     hzNEW = thee->pmgp->hzed;
```

```
01755     nx =  thee->pmgp->nx;
01756     ny =  thee->pmgp->ny;
01757     nz =  thee->pmgp->nz;
01758     nxNEW = thee->pmgp->nx;
01759     nyNEW = thee->pmgp->ny;
01760     nzNEW = thee->pmgp->nz;
01761     xminNEW = thee->pmgp->xcent - ((double)(nxNEW-1)*hxNEW)/2.0;
01762     xmaxNEW = thee->pmgp->xcent + ((double)(nxNEW-1)*hxNEW)/2.0;
01763     yminNEW = thee->pmgp->ycent - ((double)(nyNEW-1)*hyNEW)/2.0;
01764     ymaxNEW = thee->pmgp->ycent + ((double)(nyNEW-1)*hyNEW)/2.0;
01765     zminNEW = thee->pmgp->zcent - ((double)(nzNEW-1)*hzNEW)/2.0;
01766     zmaxNEW = thee->pmgp->zcent + ((double)(nzNEW-1)*hzNEW)/2.0;
01767
01768   if(pmgOLD != VNULL){
01769    /* Relevant old problem parameters */
01770    hxOLD = pmgOLD->pmgp->hx;
01771    hyOLD = pmgOLD->pmgp->hy;
01772    hzOLD = pmgOLD->pmgp->hzed;
01773    nxOLD = pmgOLD->pmgp->nx;
01774    nyOLD = pmgOLD->pmgp->ny;
01775    nzOLD = pmgOLD->pmgp->nz;
01776    xminOLD = pmgOLD->pmgp->xcent - ((double)(nxOLD-1)*hxOLD)/2.0;
01777    xmaxOLD = pmgOLD->pmgp->xcent + ((double)(nxOLD-1)*hxOLD)/2.0;
01778    yminOLD = pmgOLD->pmgp->ycent - ((double)(nyOLD-1)*hyOLD)/2.0;
01779    ymaxOLD = pmgOLD->pmgp->ycent + ((double)(nyOLD-1)*hyOLD)/2.0;
01780    zminOLD = pmgOLD->pmgp->zcent - ((double)(nzOLD-1)*hzOLD)/2.0;
01781    zmaxOLD = pmgOLD->pmgp->zcent + ((double)(nzOLD-1)*hzOLD)/2.0;
01782
01783    data = pmgOLD->u;
01784   }else{
01785    /* Relevant old problem parameters */
01786    hxOLD = thee->potMap->hx;
01787    hyOLD = thee->potMap->hy;
01788    hzOLD = thee->potMap->hzed;
01789    nxOLD = thee->potMap->nx;
01790    nyOLD = thee->potMap->ny;
01791    nzOLD = thee->potMap->nz;
01792    xminOLD = thee->potMap->xmin;
01793    xmaxOLD = thee->potMap->xmax;
01794    yminOLD = thee->potMap->ymin;
01795    ymaxOLD = thee->potMap->ymax;
01796    zminOLD = thee->potMap->zmin;
01797    zmaxOLD = thee->potMap->zmax;
01798
01799    data = thee->potMap->data;
01800   }
01801      /* BOUNDARY CONDITION SETUP FOR POINTS OFF OLD MESH:
01802       * For each "atom" (only one for bcfl=1), we use the following formula to
01803       * calculate the boundary conditions:
01804       *    g(x) = \frac{q e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
01805       *           * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
01806       *           * 1/d
01807       * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
01808       * We only need to evaluate some of these prefactors once:
01809       *    pre1 = \frac{e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
01810       * which gives the potential as
01811       *    g(x) = pre1 * q/d * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
```

```
01812      */
01813     pbe = thee->pbe;
01814     eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
01815     T = Vpbe_getTemperature(pbe);               /* K             */
01816     pre1 = (Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T);
01817
01818     /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
01819      * m/A, then we will only need to deal with distances and sizes in
01820      * Angstroms rather than meters.                                    */
01821     xkappa = Vpbe_getXkappa(pbe);              /* A^{-1}        */
01822     pre1 = pre1*(1.0e10);
01823     size = Vpbe_getSoluteRadius(pbe);
01824     apos = Vpbe_getSoluteCenter(pbe);
01825     charge = Vunit_ec*Vpbe_getSoluteCharge(pbe);
01826
01827     /* Check for rounding error */
01828     if (VABS(xminOLD-xminNEW) < VSMALL) xminNEW = xminOLD;
01829     if (VABS(xmaxOLD-xmaxNEW) < VSMALL) xmaxNEW = xmaxOLD;
01830     if (VABS(yminOLD-yminNEW) < VSMALL) yminNEW = yminOLD;
01831     if (VABS(ymaxOLD-ymaxNEW) < VSMALL) ymaxNEW = ymaxOLD;
01832     if (VABS(zminOLD-zminNEW) < VSMALL) zminNEW = zminOLD;
01833     if (VABS(zmaxOLD-zmaxNEW) < VSMALL) zmaxNEW = zmaxOLD;
01834
01835
01836     /* Sanity check: make sure we're within the old mesh */
01837     Vnm_print(0, "VPMG::focusFillBound -- New mesh mins = %g, %g, %g\n",
01838      xminNEW, yminNEW, zminNEW);
01839     Vnm_print(0, "VPMG::focusFillBound -- New mesh maxs = %g, %g, %g\n",
01840      xmaxNEW, ymaxNEW, zmaxNEW);
01841     Vnm_print(0, "VPMG::focusFillBound -- Old mesh mins = %g, %g, %g\n",
01842      xminOLD, yminOLD, zminOLD);
01843     Vnm_print(0, "VPMG::focusFillBound -- Old mesh maxs = %g, %g, %g\n",
01844      xmaxOLD, ymaxOLD, zmaxOLD);
01845
01846     /* The following is obsolete; we'll substitute analytical boundary
01847      * condition values when the new mesh falls outside the old */
01848     if ((xmaxNEW>xmaxOLD) || (ymaxNEW>ymaxOLD) || (zmaxNEW>zmaxOLD) ||
01849         (xminOLD>xminNEW) || (yminOLD>yminNEW) || (zminOLD>zminNEW)) {
01850
01851         Vnm_print(2, "Vpmg::focusFillBound -- new mesh not contained in old!\n");

01852   Vnm_print(2, "Vpmg::focusFillBound -- old mesh min = (%g, %g, %g)\n",
01853       xminOLD, yminOLD, zminOLD);
01854         Vnm_print(2, "Vpmg::focusFillBound -- old mesh max = (%g, %g, %g)\n",
01855       xmaxOLD, ymaxOLD, zmaxOLD);
01856   Vnm_print(2, "Vpmg::focusFillBound -- new mesh min = (%g, %g, %g)\n",
01857       xminNEW, yminNEW, zminNEW);
01858         Vnm_print(2, "Vpmg::focusFillBound -- new mesh max = (%g, %g, %g)\n",
01859       xmaxNEW, ymaxNEW, zmaxNEW);
01860   fflush(stderr);
01861         VASSERT(0);
01862     }
01863
01864  uvalMin = VPMGSMALL;
01865  uvalMax = -VPMGSMALL;
01866
01867     /* Fill the "i" boundaries (dirichlet) */
```

```
01868     for (k=0; k<nzNEW; k++) {
01869         for (j=0; j<nyNEW; j++) {
01870             /* Low X face */
01871             x = xminNEW;
01872             y = yminNEW + j*hyNEW;
01873             z = zminNEW + k*hzNEW;
01874             if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zmin
      OLD-VSMALL)) &&
01875                 (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmax
      OLD+VSMALL))) {
01876                 ifloat = (x - xminOLD)/hxOLD;
01877                 jfloat = (y - yminOLD)/hyOLD;
01878                 kfloat = (z - zminOLD)/hzOLD;
01879                 ihi = (int)ceil(ifloat);
01880                 if (ihi > (nxOLD-1)) ihi = nxOLD-1;
01881                 ilo = (int)floor(ifloat);
01882                 if (ilo < 0) ilo = 0;
01883                 jhi = (int)ceil(jfloat);
01884                 if (jhi > (nyOLD-1)) jhi = nyOLD-1;
01885                 jlo = (int)floor(jfloat);
01886                 if (jlo < 0) jlo = 0;
01887                 khi = (int)ceil(kfloat);
01888                 if (khi > (nzOLD-1)) khi = nzOLD-1;
01889                 klo = (int)floor(kfloat);
01890                 if (klo < 0) klo = 0;
01891                 dx = ifloat - (double)(ilo);
01892                 dy = jfloat - (double)(jlo);
01893                 dz = kfloat - (double)(klo);
01894                 nx = nxOLD; ny = nyOLD; nz = nzOLD;
01895                 uval =  dx*dy*dz*(data[IJK(ihi,jhi,khi)])
01896     + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
01897     + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
01898     + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
01899     + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
01900     + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
01901     + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
01902     + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
01903                 nx = nxNEW; ny = nyNEW; nz = nzNEW;
01904             } else {
01905     Vnm_print(2, "focusFillBound (%s, %d):  Off old mesh at %g, %g \
01906         %g!\n", __FILE__, __LINE__, x, y, z);
01907     Vnm_print(2, "focusFillBound (%s, %d):  old mesh lower corner at \
01908         %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
01909                 Vnm_print(2, "focusFillBound (%s, %d):  old mesh upper corner at
      \
01910         %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
01911     VASSERT(0);
01912             }
01913             nx = nxNEW; ny = nyNEW; nz = nzNEW;
01914             thee->gxcf[IJKx(j,k,0)] = uval;
01915    if(uval < uvalMin) uvalMin = uval;
01916    if(uval > uvalMax) uvalMax = uval;
01917
01918             /* High X face */
01919             x = xmaxNEW;
01920             if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zmin
      OLD-VSMALL)) &&
```

```
01921                 (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmax
      OLD+VSMALL))) {
01922                 ifloat = (x - xminOLD)/hxOLD;
01923                 jfloat = (y - yminOLD)/hyOLD;
01924                 kfloat = (z - zminOLD)/hzOLD;
01925                 ihi = (int)ceil(ifloat);
01926                 if (ihi > (nxOLD-1)) ihi = nxOLD-1;
01927                 ilo = (int)floor(ifloat);
01928                 if (ilo < 0) ilo = 0;
01929                 jhi = (int)ceil(jfloat);
01930                 if (jhi > (nyOLD-1)) jhi = nyOLD-1;
01931                 jlo = (int)floor(jfloat);
01932                 if (jlo < 0) jlo = 0;
01933                 khi = (int)ceil(kfloat);
01934                 if (khi > (nzOLD-1)) khi = nzOLD-1;
01935                 klo = (int)floor(kfloat);
01936                 if (klo < 0) klo = 0;
01937                 dx = ifloat - (double)(ilo);
01938                 dy = jfloat - (double)(jlo);
01939                 dz = kfloat - (double)(klo);
01940                 nx = nxOLD; ny = nyOLD; nz = nzOLD;
01941                 uval =  dx*dy*dz*(data[IJK(ihi,jhi,khi)])
01942     + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
01943     + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
01944     + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
01945     + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
01946     + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
01947     + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
01948     + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
01949                 nx = nxNEW; ny = nyNEW; nz = nzNEW;
01950             } else {
01951     Vnm_print(2, "focusFillBound (%s, %d):  Off old mesh at %g, %g \
01952         %g!\n", __FILE__, __LINE__, x, y, z);
01953     Vnm_print(2, "focusFillBound (%s, %d):  old mesh lower corner at \
01954         %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
01955                 Vnm_print(2, "focusFillBound (%s, %d):  old mesh upper corner at
      \
01956         %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
01957     VASSERT(0);
01958             }
01959             nx = nxNEW; ny = nyNEW; nz = nzNEW;
01960             thee->gxcf[IJKx(j,k,1)] = uval;
01961    if(uval < uvalMin) uvalMin = uval;
01962    if(uval > uvalMax) uvalMax = uval;
01963
01964             /* Zero Neumann conditions */
01965             nx = nxNEW; ny = nyNEW; nz = nzNEW;
01966             thee->gxcf[IJKx(j,k,2)] = 0.0;
01967             nx = nxNEW; ny = nyNEW; nz = nzNEW;
01968             thee->gxcf[IJKx(j,k,3)] = 0.0;
01969         }
01970     }
01971
01972     /* Fill the "j" boundaries (dirichlet) */
01973     for (k=0; k<nzNEW; k++) {
01974         for (i=0; i<nxNEW; i++) {
01975             /* Low Y face */
```

```
01976             x = xminNEW + i*hxNEW;
01977             y = yminNEW;
01978             z = zminNEW + k*hzNEW;
01979             if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zmin
      OLD-VSMALL)) &&
01980                 (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmax
      OLD+VSMALL))) {
01981                 ifloat = (x - xminOLD)/hxOLD;
01982                 jfloat = (y - yminOLD)/hyOLD;
01983                 kfloat = (z - zminOLD)/hzOLD;
01984                 ihi = (int)ceil(ifloat);
01985                 if (ihi > (nxOLD-1)) ihi = nxOLD-1;
01986                 ilo = (int)floor(ifloat);
01987                 if (ilo < 0) ilo = 0;
01988                 jhi = (int)ceil(jfloat);
01989                 if (jhi > (nyOLD-1)) jhi = nyOLD-1;
01990                 jlo = (int)floor(jfloat);
01991                 if (jlo < 0) jlo = 0;
01992                 khi = (int)ceil(kfloat);
01993                 if (khi > (nzOLD-1)) khi = nzOLD-1;
01994                 klo = (int)floor(kfloat);
01995                 if (klo < 0) klo = 0;
01996                 dx = ifloat - (double)(ilo);
01997                 dy = jfloat - (double)(jlo);
01998                 dz = kfloat - (double)(klo);
01999                 nx = nxOLD; ny = nyOLD; nz = nzOLD;
02000                 uval =  dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02001     + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02002     + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02003     + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02004     + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02005     + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02006     + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02007     + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02008                 nx = nxNEW; ny = nyNEW; nz = nzNEW;
02009             } else {
02010     Vnm_print(2, "focusFillBound (%s, %d):  Off old mesh at %g, %g \
02011         %g!\n", __FILE__, __LINE__, x, y, z);
02012     Vnm_print(2, "focusFillBound (%s, %d):  old mesh lower corner at \
02013         %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02014                 Vnm_print(2, "focusFillBound (%s, %d):  old mesh upper corner at
      \
02015         %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02016     VASSERT(0);
02017             }
02018             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02019             thee->gycf[IJKy(i,k,0)] = uval;
02020    if(uval < uvalMin) uvalMin = uval;
02021    if(uval > uvalMax) uvalMax = uval;
02022
02023             /* High Y face */
02024             y = ymaxNEW;
02025             if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zmin
      OLD-VSMALL)) &&
02026                 (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmax
      OLD+VSMALL))) {
02027                 ifloat = (x - xminOLD)/hxOLD;
```

```
02028                 jfloat = (y - yminOLD)/hyOLD;
02029                 kfloat = (z - zminOLD)/hzOLD;
02030                 ihi = (int)ceil(ifloat);
02031                 if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02032                 ilo = (int)floor(ifloat);
02033                 if (ilo < 0) ilo = 0;
02034                 jhi = (int)ceil(jfloat);
02035                 if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02036                 jlo = (int)floor(jfloat);
02037                 if (jlo < 0) jlo = 0;
02038                 khi = (int)ceil(kfloat);
02039                 if (khi > (nzOLD-1)) khi = nzOLD-1;
02040                 klo = (int)floor(kfloat);
02041                 if (klo < 0) klo = 0;
02042                 dx = ifloat - (double)(ilo);
02043                 dy = jfloat - (double)(jlo);
02044                 dz = kfloat - (double)(klo);
02045                 nx = nxOLD; ny = nyOLD; nz = nzOLD;
02046                 uval =  dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02047     + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02048     + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02049     + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02050     + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02051     + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02052     + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02053     + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02054                 nx = nxNEW; ny = nyNEW; nz = nzNEW;
02055             } else {
02056     Vnm_print(2, "focusFillBound (%s, %d):  Off old mesh at %g, %g \
02057         %g!\n", __FILE__, __LINE__, x, y, z);
02058     Vnm_print(2, "focusFillBound (%s, %d):  old mesh lower corner at \
02059         %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02060                 Vnm_print(2, "focusFillBound (%s, %d):  old mesh upper corner at
    \
02061         %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02062     VASSERT(0);
02063             }
02064             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02065             thee->gycf[IJKy(i,k,1)] = uval;
02066    if(uval < uvalMin) uvalMin = uval;
02067    if(uval > uvalMax) uvalMax = uval;
02068
02069             /* Zero Neumann conditions */
02070             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02071             thee->gycf[IJKy(i,k,2)] = 0.0;
02072             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02073             thee->gycf[IJKy(i,k,3)] = 0.0;
02074         }
02075     }
02076
02077     /* Fill the "k" boundaries (dirichlet) */
02078     for (j=0; j<nyNEW; j++) {
02079         for (i=0; i<nxNEW; i++) {
02080             /* Low Z face */
02081             x = xminNEW + i*hxNEW;
02082             y = yminNEW + j*hyNEW;
02083             z = zminNEW;
```

```
02084             if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zmin
      OLD-VSMALL)) &&
02085                 (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmax
      OLD+VSMALL))) {
02086                 ifloat = (x - xminOLD)/hxOLD;
02087                 jfloat = (y - yminOLD)/hyOLD;
02088                 kfloat = (z - zminOLD)/hzOLD;
02089                 ihi = (int)ceil(ifloat);
02090                 if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02091                 ilo = (int)floor(ifloat);
02092                 if (ilo < 0) ilo = 0;
02093                 jhi = (int)ceil(jfloat);
02094                 if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02095                 jlo = (int)floor(jfloat);
02096                 if (jlo < 0) jlo = 0;
02097                 khi = (int)ceil(kfloat);
02098                 if (khi > (nzOLD-1)) khi = nzOLD-1;
02099                 klo = (int)floor(kfloat);
02100                 if (klo < 0) klo = 0;
02101                 dx = ifloat - (double)(ilo);
02102                 dy = jfloat - (double)(jlo);
02103                 dz = kfloat - (double)(klo);
02104                 nx = nxOLD; ny = nyOLD; nz = nzOLD;
02105                 uval =  dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02106     + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02107     + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02108     + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02109     + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02110     + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02111     + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02112     + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02113                 nx = nxNEW; ny = nyNEW; nz = nzNEW;
02114             } else {
02115     Vnm_print(2, "focusFillBound (%s, %d):  Off old mesh at %g, %g \
02116         %g!\n", __FILE__, __LINE__, x, y, z);
02117     Vnm_print(2, "focusFillBound (%s, %d):  old mesh lower corner at \
02118         %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02119                 Vnm_print(2, "focusFillBound (%s, %d):  old mesh upper corner at
      \
02120         %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02121     VASSERT(0);
02122             }
02123             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02124             thee->gzcf[IJKz(i,j,0)] = uval;
02125    if(uval < uvalMin) uvalMin = uval;
02126    if(uval > uvalMax) uvalMax = uval;
02127
02128             /* High Z face */
02129             z = zmaxNEW;
02130             if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zmin
      OLD-VSMALL)) &&
02131                 (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmax
      OLD+VSMALL))) {
02132                 ifloat = (x - xminOLD)/hxOLD;
02133                 jfloat = (y - yminOLD)/hyOLD;
02134                 kfloat = (z - zminOLD)/hzOLD;
02135                 ihi = (int)ceil(ifloat);
```

```
02136                 if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02137                 ilo = (int)floor(ifloat);
02138                 if (ilo < 0) ilo = 0;
02139                 jhi = (int)ceil(jfloat);
02140                 if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02141                 jlo = (int)floor(jfloat);
02142                 if (jlo < 0) jlo = 0;
02143                 khi = (int)ceil(kfloat);
02144                 if (khi > (nzOLD-1)) khi = nzOLD-1;
02145                 klo = (int)floor(kfloat);
02146                 if (klo < 0) klo = 0;
02147                 dx = ifloat - (double)(ilo);
02148                 dy = jfloat - (double)(jlo);
02149                 dz = kfloat - (double)(klo);
02150                 nx = nxOLD; ny = nyOLD; nz = nzOLD;
02151                 uval =  dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02152     + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02153     + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02154     + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02155     + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02156     + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02157     + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02158     + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02159                 nx = nxNEW; ny = nyNEW; nz = nzNEW;
02160             } else {
02161     Vnm_print(2, "focusFillBound (%s, %d):  Off old mesh at %g, %g \
02162         %g!\n", __FILE__, __LINE__, x, y, z);
02163     Vnm_print(2, "focusFillBound (%s, %d):  old mesh lower corner at \
02164         %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02165                 Vnm_print(2, "focusFillBound (%s, %d):  old mesh upper corner at
    \
02166         %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02167     VASSERT(0);
02168             }
02169             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02170             thee->gzcf[IJKz(i,j,1)] = uval;
02171    if(uval < uvalMin) uvalMin = uval;
02172    if(uval > uvalMax) uvalMax = uval;
02173
02174             /* Zero Neumann conditions */
02175             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02176             thee->gzcf[IJKz(i,j,2)] = 0.0;
02177             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02178             thee->gzcf[IJKz(i,j,3)] = 0.0;
02179         }
02180     }
02181
02182  if((uvalMin < SINH_MIN) || (uvalMax > SINH_MAX)){
02183   Vnm_print(2, "\nfocusFillBound:  WARNING! Unusually large potential values\n" \

02184         "                   detected on the focusing boundary!  \n" \
02185         "                   Convergence not guaranteed for NPBE/NRPBE calculation
      s!\n");
02186  }
02187
02188 }
02189
```

```
02190 VPRIVATE void extEnergy(Vpmg *thee, Vpmg *pmgOLD, PBEparm_calcEnergy extFlag,
02191       double partMin[3], double partMax[3], int bflags[6]) {
02192
02193     Vatom *atom;
02194     double hxNEW, hyNEW, hzNEW;
02195     double lowerCorner[3], upperCorner[3];
02196     int nxNEW, nyNEW, nzNEW;
02197     int nxOLD, nyOLD, nzOLD;
02198     int i,j,k;
02199     double xmin, xmax, ymin, ymax, zmin, zmax;
02200     double hxOLD, hyOLD, hzOLD;
02201     double xval, yval, zval;
02202     double x,y,z;
02203     int nx, ny, nz;
02204
02205     /* Set the new external energy contribution to zero.  Any external
02206      * contributions from higher levels will be included in the appropriate
02207      * energy function call. */
02208     thee->extQmEnergy = 0;
02209     thee->extQfEnergy = 0;
02210     thee->extDiEnergy = 0;
02211
02212     /* New problem dimensions */
02213     hxNEW = thee->pmgp->hx;
02214     hyNEW = thee->pmgp->hy;
02215     hzNEW = thee->pmgp->hzed;
02216     nxNEW = thee->pmgp->nx;
02217     nyNEW = thee->pmgp->ny;
02218     nzNEW = thee->pmgp->nz;
02219     lowerCorner[0] = thee->pmgp->xcent - ((double)(nxNEW-1)*hxNEW)/2.0;
02220     upperCorner[0] = thee->pmgp->xcent + ((double)(nxNEW-1)*hxNEW)/2.0;
02221     lowerCorner[1] = thee->pmgp->ycent - ((double)(nyNEW-1)*hyNEW)/2.0;
02222     upperCorner[1] = thee->pmgp->ycent + ((double)(nyNEW-1)*hyNEW)/2.0;
02223     lowerCorner[2] = thee->pmgp->zcent - ((double)(nzNEW-1)*hzNEW)/2.0;
02224     upperCorner[2] = thee->pmgp->zcent + ((double)(nzNEW-1)*hzNEW)/2.0;
02225
02226     Vnm_print(0, "VPMG::extEnergy:  energy flag = %d\n", extFlag);
02227
02228     /* Old problem dimensions */
02229     nxOLD = pmgOLD->pmgp->nx;
02230     nyOLD = pmgOLD->pmgp->ny;
02231     nzOLD = pmgOLD->pmgp->nz;
02232
02233     /* Create a partition based on the new problem dimensions */
02234     /* Vnm_print(1, "DEBUG (%s, %d):  extEnergy calling Vpmg_setPart for old PMG.
      \n",
02235   __FILE__, __LINE__); */
02236     Vpmg_setPart(pmgOLD, lowerCorner, upperCorner, bflags);
02237
02238
02239     Vnm_print(0,"VPMG::extEnergy:   Finding extEnergy dimensions...\n");
02240     Vnm_print(0,"VPMG::extEnergy    Disj part lower corner = (%g, %g, %g)\n",
02241      partMin[0], partMin[1], partMin[2]);
02242     Vnm_print(0,"VPMG::extEnergy    Disj part upper corner = (%g, %g, %g)\n",
02243      partMax[0], partMax[1], partMax[2]);
02244
02245     /* Find the old dimensions */
```

```
02246
02247     hxOLD = pmgOLD->pmgp->hx;
02248     hyOLD = pmgOLD->pmgp->hy;
02249     hzOLD = pmgOLD->pmgp->hzed;
02250     xmin =  pmgOLD->pmgp->xcent - 0.5*hxOLD*(nxOLD-1);
02251     ymin =  pmgOLD->pmgp->ycent - 0.5*hyOLD*(nyOLD-1);
02252     zmin =  pmgOLD->pmgp->zcent - 0.5*hzOLD*(nzOLD-1);
02253     xmax =  xmin+hxOLD*(nxOLD-1);
02254     ymax =  ymin+hyOLD*(nyOLD-1);
02255     zmax =  zmin+hzOLD*(nzOLD-1);
02256
02257     Vnm_print(0,"VPMG::extEnergy    Old lower corner = (%g, %g, %g)\n",
02258      xmin, ymin, zmin);
02259     Vnm_print(0,"VPMG::extEnergy    Old upper corner = (%g, %g, %g)\n",
02260      xmax, ymax, zmax);
02261
02262     /* Flip the partition, but do not include any points that will
02263    be included by another processor */
02264
02265     nx = nxOLD;
02266     ny = nyOLD;
02267     nz = nzOLD;
02268
02269     for(i=0; i<nx; i++) {
02270         xval = 1;
02271         x = i*hxOLD + xmin;
02272         if (x < partMin[0] && bflags[VAPBS_LEFT] == 1) xval = 0;
02273         else if (x > partMax[0] && bflags[VAPBS_RIGHT] == 1) xval = 0;
02274
02275         for(j=0; j<ny; j++) {
02276             yval = 1;
02277             y = j*hyOLD + ymin;
02278             if (y < partMin[1] && bflags[VAPBS_BACK] == 1) yval = 0;
02279             else if (y > partMax[1] && bflags[VAPBS_FRONT] == 1) yval = 0;
02280
02281             for(k=0; k<nz; k++) {
02282                 zval = 1;
02283                 z = k*hzOLD + zmin;
02284                 if (z < partMin[2] && bflags[VAPBS_DOWN] == 1) zval = 0;
02285                 else if (z > partMax[2] && bflags[VAPBS_UP] == 1) zval = 0;
02286
02287                 if (pmgOLD->pvec[IJK(i,j,k)] > VSMALL) pmgOLD->pvec[IJK(i,j,k)] =
       1.0;
02288                 pmgOLD->pvec[IJK(i,j,k)] = (1 - (pmgOLD->pvec[IJK(i,j,k)])) * (xv
      al*yval*zval);
02289             }
02290         }
02291     }
02292
02293     for (i=0; i<Valist_getNumberAtoms(thee->pbe->alist); i++) {
02294         xval=1;
02295         yval=1;
02296         zval=1;
02297         atom = Valist_getAtom(thee->pbe->alist, i);
02298         x = atom->position[0];
02299         y = atom->position[1];
02300         z = atom->position[2];
```

```
02301         if (x < partMin[0] && bflags[VAPBS_LEFT] == 1) xval = 0;
02302         else if (x > partMax[0] && bflags[VAPBS_RIGHT] == 1) xval = 0;
02303         if (y < partMin[1] && bflags[VAPBS_BACK] == 1) yval = 0;
02304         else if (y > partMax[1] && bflags[VAPBS_FRONT] == 1) yval = 0;
02305         if (z < partMin[2] && bflags[VAPBS_DOWN] == 1) zval = 0;
02306         else if (z > partMax[2] && bflags[VAPBS_UP] == 1) zval = 0;
02307         if (atom->partID > VSMALL) atom->partID = 1.0;
02308         atom->partID = (1 - atom->partID) * (xval*yval*zval);
02309     }
02310 
02311     /* Now calculate the energy on inverted subset of the domain */
02312     thee->extQmEnergy = Vpmg_qmEnergy(pmgOLD, 1);
02313     Vnm_print(0, "VPMG::extEnergy: extQmEnergy = %g kT\n", thee->extQmEnergy);
02314     thee->extQfEnergy = Vpmg_qfEnergy(pmgOLD, 1);
02315     Vnm_print(0, "VPMG::extEnergy: extQfEnergy = %g kT\n", thee->extQfEnergy);
02316     thee->extDiEnergy = Vpmg_dielEnergy(pmgOLD, 1);
02317     Vnm_print(0, "VPMG::extEnergy: extDiEnergy = %g kT\n", thee->extDiEnergy);
02318     Vpmg_unsetPart(pmgOLD);
02319 }
02320 
02321 VPRIVATE double bcfl1sp(double size, double *apos, double charge,
02322       double xkappa, double pre1, double *pos) {
02323 
02324     double dist, val;
02325 
02326     dist = VSQRT(VSQR(pos[0]-apos[0]) + VSQR(pos[1]-apos[1])
02327      + VSQR(pos[2]-apos[2]));
02328     if (xkappa > VSMALL) {
02329         val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02330   / (1+xkappa*size);
02331     } else {
02332         val = pre1*(charge/dist);
02333     }
02334 
02335     return val;
02336 }
02337 
02338 VPRIVATE void bcfl1(double size, double *apos, double charge,
02339      double xkappa, double pre1, double *gxcf, double *gycf, double *gzcf,
02340      double *xf, double *yf, double *zf, int nx, int ny, int nz) {
02341 
02342     int i, j, k;
02343     double dist, val;
02344     double gpos[3];
02345 
02346     /* the "i" boundaries (dirichlet) */
02347     for (k=0; k<nz; k++) {
02348         gpos[2] = zf[k];
02349         for (j=0; j<ny; j++) {
02350             gpos[1] = yf[j];
02351             gpos[0] = xf[0];
02352             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02353        + VSQR(gpos[2]-apos[2]));
02354             if (xkappa > VSMALL) {
02355                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02356     / (1+xkappa*size);
02357             } else {
```

```
02358                 val = pre1*(charge/dist);
02359             }
02360             gxcf[IJKx(j,k,0)] += val;
02361             gpos[0] = xf[nx-1];
02362             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02363        + VSQR(gpos[2]-apos[2]));
02364             if (xkappa > VSMALL) {
02365                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02366     / (1+xkappa*size);
02367             } else {
02368                 val = pre1*(charge/dist);
02369             }
02370             gxcf[IJKx(j,k,1)] += val;
02371         }
02372     }
02373
02374     /* the "j" boundaries (dirichlet) */
02375     for (k=0; k<nz; k++) {
02376         gpos[2] = zf[k];
02377         for (i=0; i<nx; i++) {
02378             gpos[0] = xf[i];
02379             gpos[1] = yf[0];
02380             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02381        + VSQR(gpos[2]-apos[2]));
02382             if (xkappa > VSMALL) {
02383                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02384     / (1+xkappa*size);
02385             } else {
02386                 val = pre1*(charge/dist);
02387             }
02388             gycf[IJKy(i,k,0)] += val;
02389             gpos[1] = yf[ny-1];
02390             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02391        + VSQR(gpos[2]-apos[2]));
02392             if (xkappa > VSMALL) {
02393                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02394     / (1+xkappa*size);
02395             } else {
02396                 val = pre1*(charge/dist);
02397             }
02398             gycf[IJKy(i,k,1)] += val;
02399         }
02400     }
02401
02402     /* the "k" boundaries (dirichlet) */
02403     for (j=0; j<ny; j++) {
02404         gpos[1] = yf[j];
02405         for (i=0; i<nx; i++) {
02406             gpos[0] = xf[i];
02407             gpos[2] = zf[0];
02408             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02409        + VSQR(gpos[2]-apos[2]));
02410             if (xkappa > VSMALL) {
02411                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02412     / (1+xkappa*size);
02413             } else {
02414                 val = pre1*(charge/dist);
```

```
02415             }
02416             gzcf[IJKz(i,j,0)] += val;
02417             gpos[2] = zf[nz-1];
02418             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02419         + VSQR(gpos[2]-apos[2]));
02420             if (xkappa > VSMALL) {
02421                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02422     / (1+xkappa*size);
02423             } else {
02424                 val = pre1*(charge/dist);
02425             }
02426             gzcf[IJKz(i,j,1)] += val;
02427          }
02428     }
02429 }
02430
02431 VPRIVATE void bcfl2(double size, double *apos,
02432                     double charge, double *dipole, double *quad,
02433                     double xkappa, double eps_p, double eps_w, double T,
02434                     double *gxcf, double *gycf, double *gzcf,
02435                     double *xf, double *yf, double *zf,
02436                     int nx, int ny, int nz) {
02437
02438     int i, j, k;
02439     double val;
02440     double gpos[3],tensor[3];
02441     double ux,uy,uz,xr,yr,zr;
02442     double qxx,qxy,qxz,qyx,qyy,qyz,qzx,qzy,qzz;
02443     double dist, pre;
02444
02445     VASSERT(dipole != VNULL);
02446     ux = dipole[0];
02447     uy = dipole[1];
02448     uz = dipole[2];
02449     if (quad != VNULL) {
02450   /* The factor of 1/3 results from using a
02451    traceless quadrupole definition. See, for example,
02452    "The Theory of Intermolecular Forces" by A.J. Stone,
02453    Chapter 3. */
02454   qxx = quad[0] / 3.0;
02455   qxy = quad[1] / 3.0;
02456   qxz = quad[2] / 3.0;
02457   qyx = quad[3] / 3.0;
02458   qyy = quad[4] / 3.0;
02459   qyz = quad[5] / 3.0;
02460   qzx = quad[6] / 3.0;
02461   qzy = quad[7] / 3.0;
02462   qzz = quad[8] / 3.0;
02463     } else {
02464   qxx = 0.0;
02465   qxy = 0.0;
02466   qxz = 0.0;
02467   qyx = 0.0;
02468   qyy = 0.0;
02469   qyz = 0.0;
02470   qzx = 0.0;
02471   qzy = 0.0;
```

```
02472   qzz = 0.0;
02473     }
02474
02475     pre = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*Vunit_kb*T);
02476     pre = pre*(1.0e10);
02477
02478     /* the "i" boundaries (dirichlet) */
02479     for (k=0; k<nz; k++) {
02480         gpos[2] = zf[k];
02481         for (j=0; j<ny; j++) {
02482             gpos[1] = yf[j];
02483             gpos[0] = xf[0];
02484             xr = gpos[0] - apos[0];
02485             yr = gpos[1] - apos[1];
02486             zr = gpos[2] - apos[2];
02487             dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02488             multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02489             val = pre*charge*tensor[0];
02490             val -= pre*ux*xr*tensor[1];
02491             val -= pre*uy*yr*tensor[1];
02492             val -= pre*uz*zr*tensor[1];
02493             val += pre*qxx*xr*xr*tensor[2];
02494             val += pre*qyy*yr*yr*tensor[2];
02495             val += pre*qzz*zr*zr*tensor[2];
02496             val += pre*2.0*qxy*xr*yr*tensor[2];
02497             val += pre*2.0*qxz*xr*zr*tensor[2];
02498             val += pre*2.0*qyz*yr*zr*tensor[2];
02499             gxcf[IJKx(j,k,0)] += val;
02500
02501             gpos[0] = xf[nx-1];
02502             xr = gpos[0] - apos[0];
02503             dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02504             multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02505             val = pre*charge*tensor[0];
02506             val -= pre*ux*xr*tensor[1];
02507             val -= pre*uy*yr*tensor[1];
02508             val -= pre*uz*zr*tensor[1];
02509             val += pre*qxx*xr*xr*tensor[2];
02510             val += pre*qyy*yr*yr*tensor[2];
02511             val += pre*qzz*zr*zr*tensor[2];
02512             val += pre*2.0*qxy*xr*yr*tensor[2];
02513             val += pre*2.0*qxz*xr*zr*tensor[2];
02514             val += pre*2.0*qyz*yr*zr*tensor[2];
02515             gxcf[IJKx(j,k,1)] += val;
02516         }
02517     }
02518
02519     /* the "j" boundaries (dirichlet) */
02520     for (k=0; k<nz; k++) {
02521         gpos[2] = zf[k];
02522         for (i=0; i<nx; i++) {
02523             gpos[0] = xf[i];
02524             gpos[1] = yf[0];
02525             xr = gpos[0] - apos[0];
02526             yr = gpos[1] - apos[1];
02527             zr = gpos[2] - apos[2];
02528             dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
```

```
02529             multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02530             val = pre*charge*tensor[0];
02531             val -= pre*ux*xr*tensor[1];
02532             val -= pre*uy*yr*tensor[1];
02533             val -= pre*uz*zr*tensor[1];
02534             val += pre*qxx*xr*xr*tensor[2];
02535             val += pre*qyy*yr*yr*tensor[2];
02536             val += pre*qzz*zr*zr*tensor[2];
02537             val += pre*2.0*qxy*xr*yr*tensor[2];
02538             val += pre*2.0*qxz*xr*zr*tensor[2];
02539             val += pre*2.0*qyz*yr*zr*tensor[2];
02540             gycf[IJKy(i,k,0)] += val;
02541 
02542             gpos[1] = yf[ny-1];
02543             yr = gpos[1] - apos[1];
02544             dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02545             multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02546             val = pre*charge*tensor[0];
02547             val -= pre*ux*xr*tensor[1];
02548             val -= pre*uy*yr*tensor[1];
02549             val -= pre*uz*zr*tensor[1];
02550             val += pre*qxx*xr*xr*tensor[2];
02551             val += pre*qyy*yr*yr*tensor[2];
02552             val += pre*qzz*zr*zr*tensor[2];
02553             val += pre*2.0*qxy*xr*yr*tensor[2];
02554             val += pre*2.0*qxz*xr*zr*tensor[2];
02555             val += pre*2.0*qyz*yr*zr*tensor[2];
02556             gycf[IJKy(i,k,1)] += val;
02557         }
02558     }
02559 
02560     /* the "k" boundaries (dirichlet) */
02561     for (j=0; j<ny; j++) {
02562         gpos[1] = yf[j];
02563         for (i=0; i<nx; i++) {
02564             gpos[0] = xf[i];
02565             gpos[2] = zf[0];
02566             xr = gpos[0] - apos[0];
02567             yr = gpos[1] - apos[1];
02568             zr = gpos[2] - apos[2];
02569             dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02570             multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02571             val = pre*charge*tensor[0];
02572             val -= pre*ux*xr*tensor[1];
02573             val -= pre*uy*yr*tensor[1];
02574             val -= pre*uz*zr*tensor[1];
02575             val += pre*qxx*xr*xr*tensor[2];
02576             val += pre*qyy*yr*yr*tensor[2];
02577             val += pre*qzz*zr*zr*tensor[2];
02578             val += pre*2.0*qxy*xr*yr*tensor[2];
02579             val += pre*2.0*qxz*xr*zr*tensor[2];
02580             val += pre*2.0*qyz*yr*zr*tensor[2];
02581             gzcf[IJKz(i,j,0)] += val;
02582 
02583             gpos[2] = zf[nz-1];
02584             zr = gpos[2] - apos[2];
02585             dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
```

```
02586             multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02587             val = pre*charge*tensor[0];
02588             val -= pre*ux*xr*tensor[1];
02589             val -= pre*uy*yr*tensor[1];
02590             val -= pre*uz*zr*tensor[1];
02591             val += pre*qxx*xr*xr*tensor[2];
02592             val += pre*qyy*yr*yr*tensor[2];
02593             val += pre*qzz*zr*zr*tensor[2];
02594             val += pre*2.0*qxy*xr*yr*tensor[2];
02595             val += pre*2.0*qxz*xr*zr*tensor[2];
02596             val += pre*2.0*qyz*yr*zr*tensor[2];
02597             gzcf[IJKz(i,j,1)] += val;
02598         }
02599     }
02600 }
02601
02602 VPRIVATE void bcCalcOrig(Vpmg *thee) {
02603
02604     int nx, ny, nz;
02605     double size, *position, charge, xkappa, eps_w, T, pre1;
02606     double *dipole, *quadrupole, debye, eps_p;
02607     double xr,yr,zr,qave,*apos;
02608     double sdhcharge, sdhdipole[3], traced[9], sdhquadrupole[9];
02609     int i, j, k, iatom;
02610     Vpbe *pbe;
02611     Vatom *atom;
02612     Valist *alist;
02613
02614     pbe = thee->pbe;
02615     alist = thee->pbe->alist;
02616     nx = thee->pmgp->nx;
02617     ny = thee->pmgp->ny;
02618     nz = thee->pmgp->nz;
02619
02620     /* Zero out the boundaries */
02621     /* the "i" boundaries (dirichlet) */
02622     for (k=0; k<nz; k++) {
02623         for (j=0; j<ny; j++) {
02624             thee->gxcf[IJKx(j,k,0)] = 0.0;
02625             thee->gxcf[IJKx(j,k,1)] = 0.0;
02626             thee->gxcf[IJKx(j,k,2)] = 0.0;
02627             thee->gxcf[IJKx(j,k,3)] = 0.0;
02628         }
02629     }
02630
02631     /* the "j" boundaries (dirichlet) */
02632     for (k=0; k<nz; k++) {
02633         for (i=0; i<nx; i++) {
02634             thee->gycf[IJKy(i,k,0)] = 0.0;
02635             thee->gycf[IJKy(i,k,1)] = 0.0;
02636             thee->gycf[IJKy(i,k,2)] = 0.0;
02637             thee->gycf[IJKy(i,k,3)] = 0.0;
02638         }
02639     }
02640
02641     /* the "k" boundaries (dirichlet) */
02642     for (j=0; j<ny; j++) {
```

```
02643         for (i=0; i<nx; i++) {
02644             thee->gzcf[IJKz(i,j,0)] = 0.0;
02645             thee->gzcf[IJKz(i,j,1)] = 0.0;
02646             thee->gzcf[IJKz(i,j,2)] = 0.0;
02647             thee->gzcf[IJKz(i,j,3)] = 0.0;
02648         }
02649     }
02650 
02651     /* For each "atom" (only one for bcfl=1), we use the following formula to
02652  * calculate the boundary conditions:
02653  *    g(x) = \frac{q e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
02654  *           * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
02655  *           * 1/d
02656  * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
02657  * We only need to evaluate some of these prefactors once:
02658  *    pre1 = \frac{e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
02659  * which gives the potential as
02660  *    g(x) = pre1 * q/d * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
02661  */
02662     eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
02663     eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
02664     T = Vpbe_getTemperature(pbe);               /* K             */
02665     pre1 = (Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T);
02666 
02667     /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
02668  * m/A, then we will only need to deal with distances and sizes in
02669  * Angstroms rather than meters.                                        */
02670     xkappa = Vpbe_getXkappa(pbe);              /* A^{-1}        */
02671     pre1 = pre1*(1.0e10);
02672 
02673     switch (thee->pmgp->bcfl) {
02674         /*  If we have zero boundary conditions, we're done */
02675         case BCFL_ZERO:
02676             return;
02677 
02678     /*  For single DH sphere BC's, we only have one "atom" to deal with;
02679     *  get its information and */
02680         case BCFL_SDH:
02681             size = Vpbe_getSoluteRadius(pbe);
02682             position = Vpbe_getSoluteCenter(pbe);
02683 
02684             /*
02685     For AMOEBA SDH boundary conditions, we need to find the
02686     total monopole, dipole and traceless quadrupole moments
02687     of either the permanent multipoles, induced dipoles or
02688     non-local induced dipoles.
02689     */
02690 
02691             sdhcharge = 0.0;
02692             for (i=0; i<3; i++) sdhdipole[i] = 0.0;
02693     for (i=0; i<9; i++) sdhquadrupole[i] = 0.0;
02694 
02695     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
02696     atom = Valist_getAtom(alist, iatom);
02697     apos = Vatom_getPosition(atom);
02698     xr = apos[0] - position[0];
02699     yr = apos[1] - position[1];
```

```
02700     zr = apos[2] - position[2];
02701     switch (thee->chargeSrc) {
02702      case VCM_CHARGE:
02703       charge = Vatom_getCharge(atom);
02704       sdhcharge += charge;
02705       sdhdipole[0] += xr * charge;
02706       sdhdipole[1] += yr * charge;
02707       sdhdipole[2] += zr * charge;
02708       traced[0] = xr*xr*charge;
02709       traced[1] = xr*yr*charge;
02710       traced[2] = xr*zr*charge;
02711       traced[3] = yr*xr*charge;
02712       traced[4] = yr*yr*charge;
02713       traced[5] = yr*zr*charge;
02714       traced[6] = zr*xr*charge;
02715       traced[7] = zr*yr*charge;
02716       traced[8] = zr*zr*charge;
02717       qave = (traced[0] + traced[4] + traced[8]) / 3.0;
02718       sdhquadrupole[0] += 1.5*(traced[0] - qave);
02719       sdhquadrupole[1] += 1.5*(traced[1]);
02720       sdhquadrupole[2] += 1.5*(traced[2]);
02721       sdhquadrupole[3] += 1.5*(traced[3]);
02722       sdhquadrupole[4] += 1.5*(traced[4] - qave);
02723       sdhquadrupole[5] += 1.5*(traced[5]);
02724       sdhquadrupole[6] += 1.5*(traced[6]);
02725       sdhquadrupole[7] += 1.5*(traced[7]);
02726       sdhquadrupole[8] += 1.5*(traced[8] - qave);
02727 #if defined(WITH_TINKER)
02728      case VCM_PERMANENT:
02729       charge = Vatom_getCharge(atom);
02730       dipole = Vatom_getDipole(atom);
02731       quadrupole = Vatom_getQuadrupole(atom);
02732       sdhcharge += charge;
02733       sdhdipole[0] += xr * charge;
02734       sdhdipole[1] += yr * charge;
02735       sdhdipole[2] += zr * charge;
02736       traced[0] = xr*xr*charge;
02737       traced[1] = xr*yr*charge;
02738       traced[2] = xr*zr*charge;
02739       traced[3] = yr*xr*charge;
02740       traced[4] = yr*yr*charge;
02741       traced[5] = yr*zr*charge;
02742       traced[6] = zr*xr*charge;
02743       traced[7] = zr*yr*charge;
02744       traced[8] = zr*zr*charge;
02745       sdhdipole[0] += dipole[0];
02746       sdhdipole[1] += dipole[1];
02747       sdhdipole[2] += dipole[2];
02748       traced[0] += 2.0*xr*dipole[0];
02749       traced[1] += xr*dipole[1] + yr*dipole[0];
02750       traced[2] += xr*dipole[2] + zr*dipole[0];
02751       traced[3] += yr*dipole[0] + xr*dipole[1];
02752       traced[4] += 2.0*yr*dipole[1];
02753       traced[5] += yr*dipole[2] + zr*dipole[1];
02754       traced[6] += zr*dipole[0] + xr*dipole[2];
02755       traced[7] += zr*dipole[1] + yr*dipole[2];
02756       traced[8] += 2.0*zr*dipole[2];
```

```
02757       qave = (traced[0] + traced[4] + traced[8]) / 3.0;
02758       sdhquadrupole[0] += 1.5*(traced[0] - qave);
02759       sdhquadrupole[1] += 1.5*(traced[1]);
02760       sdhquadrupole[2] += 1.5*(traced[2]);
02761       sdhquadrupole[3] += 1.5*(traced[3]);
02762       sdhquadrupole[4] += 1.5*(traced[4] - qave);
02763       sdhquadrupole[5] += 1.5*(traced[5]);
02764       sdhquadrupole[6] += 1.5*(traced[6]);
02765       sdhquadrupole[7] += 1.5*(traced[7]);
02766       sdhquadrupole[8] += 1.5*(traced[8] - qave);
02767       sdhquadrupole[0] += quadrupole[0];
02768       sdhquadrupole[1] += quadrupole[1];
02769       sdhquadrupole[2] += quadrupole[2];
02770       sdhquadrupole[3] += quadrupole[3];
02771       sdhquadrupole[4] += quadrupole[4];
02772       sdhquadrupole[5] += quadrupole[5];
02773       sdhquadrupole[6] += quadrupole[6];
02774       sdhquadrupole[7] += quadrupole[7];
02775       sdhquadrupole[8] += quadrupole[8];
02776     case VCM_INDUCED:
02777       dipole = Vatom_getInducedDipole(atom);
02778       sdhdipole[0] += dipole[0];
02779       sdhdipole[1] += dipole[1];
02780       sdhdipole[2] += dipole[2];
02781       traced[0] = 2.0*xr*dipole[0];
02782       traced[1] = xr*dipole[1] + yr*dipole[0];
02783       traced[2] = xr*dipole[2] + zr*dipole[0];
02784       traced[3] = yr*dipole[0] + xr*dipole[1];
02785       traced[4] = 2.0*yr*dipole[1];
02786       traced[5] = yr*dipole[2] + zr*dipole[1];
02787       traced[6] = zr*dipole[0] + xr*dipole[2];
02788       traced[7] = zr*dipole[1] + yr*dipole[2];
02789       traced[8] = 2.0*zr*dipole[2];
02790       qave = (traced[0] + traced[4] + traced[8]) / 3.0;
02791       sdhquadrupole[0] += 1.5*(traced[0] - qave);
02792       sdhquadrupole[1] += 1.5*(traced[1]);
02793       sdhquadrupole[2] += 1.5*(traced[2]);
02794       sdhquadrupole[3] += 1.5*(traced[3]);
02795       sdhquadrupole[4] += 1.5*(traced[4] - qave);
02796       sdhquadrupole[5] += 1.5*(traced[5]);
02797       sdhquadrupole[6] += 1.5*(traced[6]);
02798       sdhquadrupole[7] += 1.5*(traced[7]);
02799       sdhquadrupole[8] += 1.5*(traced[8] - qave);
02800     case VCM_NLINDUCED:
02801       dipole = Vatom_getNLInducedDipole(atom);
02802       sdhdipole[0] += dipole[0];
02803       sdhdipole[1] += dipole[1];
02804       sdhdipole[2] += dipole[2];
02805       traced[0] = 2.0*xr*dipole[0];
02806       traced[1] = xr*dipole[1] + yr*dipole[0];
02807       traced[2] = xr*dipole[2] + zr*dipole[0];
02808       traced[3] = yr*dipole[0] + xr*dipole[1];
02809       traced[4] = 2.0*yr*dipole[1];
02810       traced[5] = yr*dipole[2] + zr*dipole[1];
02811       traced[6] = zr*dipole[0] + xr*dipole[2];
02812       traced[7] = zr*dipole[1] + yr*dipole[2];
02813       traced[8] = 2.0*zr*dipole[2];
```

```
02814       qave = (traced[0] + traced[4] + traced[8]) / 3.0;
02815       sdhquadrupole[0] += 1.5*(traced[0] - qave);
02816       sdhquadrupole[1] += 1.5*(traced[1]);
02817       sdhquadrupole[2] += 1.5*(traced[2]);
02818       sdhquadrupole[3] += 1.5*(traced[3]);
02819       sdhquadrupole[4] += 1.5*(traced[4] - qave);
02820       sdhquadrupole[5] += 1.5*(traced[5]);
02821       sdhquadrupole[6] += 1.5*(traced[6]);
02822       sdhquadrupole[7] += 1.5*(traced[7]);
02823       sdhquadrupole[8] += 1.5*(traced[8] - qave);
02824 #endif /* if defined(WITH_TINKER) */
02825     }
02826    }
02827    /* SDH dipole and traceless quadrupole values
02828     were checked against similar routines in TINKER
02829     for large proteins.
02830
02831     debye=4.8033324;
02832     printf("%6.3f, %6.3f, %6.3f\n", sdhdipole[0]*debye,
02833     sdhdipole[1]*debye, sdhdipole[2]*debye);
02834     printf("%6.3f\n", sdhquadrupole[0]*debye);
02835     printf("%6.3f %6.3f\n", sdhquadrupole[3]*debye,
02836     sdhquadrupole[4]*debye);
02837     printf("%6.3f %6.3f %6.3f\n", sdhquadrupole[6]*debye,
02838     sdhquadrupole[7]*debye, sdhquadrupole[8]*debye);
02839     */
02840
02841    bcfl2(size, position, sdhcharge, sdhdipole, sdhquadrupole,
02842       xkappa, eps_p, eps_w, T, thee->gxcf, thee->gycf,
02843       thee->gzcf, thee->xf, thee->yf, thee->zf, nx, ny, nz);
02844             break;
02845
02846          case BCFL_MDH:
02847    for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
02848     atom = Valist_getAtom(alist, iatom);
02849     position = Vatom_getPosition(atom);
02850     charge = Vunit_ec*Vatom_getCharge(atom);
02851     dipole = VNULL;
02852     quadrupole = VNULL;
02853     size = Vatom_getRadius(atom);
02854     switch (thee->chargeSrc)
02855     {
02856      case VCM_CHARGE:
02857       ;
02858 #if  defined(WITH_TINKER)
02859      case VCM_PERMANENT:
02860       dipole = Vatom_getDipole(atom);
02861       quadrupole = Vatom_getQuadrupole(atom);
02862
02863      case VCM_INDUCED:
02864       dipole = Vatom_getInducedDipole(atom);
02865
02866      case VCM_NLINDUCED:
02867       dipole = Vatom_getNLInducedDipole(atom);
02868 #endif
02869     }
02870     bcfl1(size, position, charge, xkappa, pre1,
```

```
02871         thee->gxcf, thee->gycf, thee->gzcf,
02872         thee->xf, thee->yf, thee->zf, nx, ny, nz);
02873     }
02874     break;
02875 
02876           case BCFL_UNUSED:
02877               Vnm_print(2, "bcCalc:  Invalid bcfl (%d)!\n", thee->pmgp->bcfl);
02878               VASSERT(0);
02879 
02880           case BCFL_FOCUS:
02881               Vnm_print(2, "VPMG::bcCalc -- not appropriate for focusing!\n");
02882               VASSERT(0);
02883 
02884           default:
02885               Vnm_print(2, "VPMG::bcCalc -- invalid boundary condition \
02886 flag (%d)!\n", thee->pmgp->bcfl);
02887               VASSERT(0);
02888       }
02889 }
02890 
02891 /*
02892  Used by bcflnew
02893  */
02894 VPRIVATE int gridPointIsValid(int i, int j, int k, int nx, int ny, int nz){
02895 
02896  int isValid = 0;
02897 
02898  if((k==0) || (k==nz-1)){
02899   isValid = 1;
02900  }else if((j==0) || (j==ny-1)){
02901   isValid = 1;
02902  }else if((i==0) || (i==nx-1)){
02903   isValid = 1;
02904  }
02905 
02906  return isValid;
02907 }
02908 
02909 /*
02910  Used by bcflnew
02911  */
02912 #ifdef DEBUG_MAC_OSX_OCL
02913 #include "mach_chud.h"
02914 VPRIVATE void packAtomsOpenCL(float *ax, float *ay, float *az,
02915       float *charge, float *size, Vpmg *thee){
02916 
02917  int i;
02918  int natoms;
02919 
02920  Vatom *atom = VNULL;
02921  Valist *alist = VNULL;
02922 
02923  alist = thee->pbe->alist;
02924  natoms = Valist_getNumberAtoms(alist);
02925 
02926  for(i=0;i<natoms;i++){
02927   atom = &(alist->atoms[i]);
```

```
02928   charge[i] = Vunit_ec*atom->charge;
02929   ax[i] = atom->position[0];
02930   ay[i] = atom->position[1];
02931   az[i] = atom->position[2];
02932   size[i] = atom->radius;
02933  }
02934 }
02935
02936 /*
02937  Used by bcflnew
02938  */
02939 VPRIVATE void packUnpackOpenCL(int nx, int ny, int nz, int ngrid,
02940        float *gx, float *gy, float *gz, float *value,
02941        Vpmg *thee, int pack){
02942
02943  int i,j,k,igrid;
02944  int x0,x1,y0,y1,z0,z1;
02945
02946  float gpos[3];
02947  double *xf, *yf, *zf;
02948  double *gxcf, *gycf, *gzcf;
02949
02950  xf = thee->xf;
02951  yf = thee->yf;
02952  zf = thee->zf;
02953
02954  gxcf = thee->gxcf;
02955  gycf = thee->gycf;
02956  gzcf = thee->gzcf;
02957
02958  igrid = 0;
02959  for(k=0;k<nz;k++){
02960   gpos[2] = zf[k];
02961   for(j=0;j<ny;j++){
02962    gpos[1] = yf[j];
02963    for(i=0;i<nx;i++){
02964     gpos[0] = xf[i];
02965     if(gridPointIsValid(i, j, k, nx, ny, nz)){
02966      if(pack != 0){
02967       gx[igrid] = gpos[0];
02968       gy[igrid] = gpos[1];
02969       gz[igrid] = gpos[2];
02970
02971       value[igrid] = 0.0;
02972      }else{
02973       x0 = IJKx(j,k,0);
02974       x1 = IJKx(j,k,1);
02975       y0 = IJKy(i,k,0);
02976       y1 = IJKy(i,k,1);
02977       z0 = IJKz(i,j,0);
02978       z1 = IJKz(i,j,1);
02979
02980       if(i==0){
02981        gxcf[x0] += value[igrid];
02982       }
02983       if(i==nx-1){
02984        gxcf[x1] += value[igrid];
```

```
02985       }
02986       if(j==0){
02987        gycf[y0] += value[igrid];
02988       }
02989       if(j==ny-1){
02990        gycf[y1] += value[igrid];
02991       }
02992       if(k==0){
02993        gzcf[z0] += value[igrid];
02994       }
02995       if(k==nz-1){
02996        gzcf[z1] += value[igrid];
02997       }
02998      }
02999
03000      igrid++;
03001     } //end is valid point
03002    } //end i
03003   } //end j
03004  } //end k
03005
03006 }
03007
03008 /*
03009  bcflnew is an optimized replacement for bcfl1. bcfl1 is still used when TINKER
03010  support is compiled in.
03011  bcflnew uses: packUnpack, packAtoms, gridPointIsValid
03012  */
03013 VPRIVATE void bcflnewOpenCL(Vpmg *thee){
03014
03015  int i,j,k, iatom, igrid;
03016  int x0, x1, y0, y1, z0, z1;
03017
03018  int nx, ny, nz;
03019  int natoms, ngrid, ngadj;
03020
03021  float dist, pre1, eps_w, eps_p, T, xkappa;
03022
03023  float *ax, *ay, *az;
03024  float *charge, *size, *val;
03025
03026  float *gx, *gy, *gz;
03027
03028  Vpbe *pbe = thee->pbe;
03029
03030  nx = thee->pmgp->nx;
03031     ny = thee->pmgp->ny;
03032     nz = thee->pmgp->nz;
03033
03034  eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
03035     eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
03036     T = Vpbe_getTemperature(pbe);              /* K             */
03037     pre1 = ((Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T))*(1.0e10);
03038  xkappa = Vpbe_getXkappa(pbe);
03039
03040  natoms = Valist_getNumberAtoms(thee->pbe->alist);
03041  ngrid = 2*((nx*ny) + (ny*nz) + (nx*nz));
```

```
03042  ngadj = ngrid + (512 - (ngrid & 511));
03043
03044  ax = (float*)malloc(natoms * sizeof(float));
03045  ay = (float*)malloc(natoms * sizeof(float));
03046  az = (float*)malloc(natoms * sizeof(float));
03047
03048  charge = (float*)malloc(natoms * sizeof(float));
03049  size = (float*)malloc(natoms * sizeof(float));
03050
03051  gx = (float*)malloc(ngrid * sizeof(float));
03052  gy = (float*)malloc(ngrid * sizeof(float));
03053  gz = (float*)malloc(ngrid * sizeof(float));
03054
03055  val = (float*)malloc(ngrid * sizeof(float));
03056
03057  packAtomsOpenCL(ax,ay,az,charge,size,thee);
03058  packUnpackOpenCL(nx,ny,nz,ngrid,gx,gy,gz,val,thee,1);
03059
03060  runMDHCL(ngrid,natoms,ngadj,ax,ay,az,gx,gy,gz,charge,size,xkappa,pre1,val);
03061
03062  packUnpackOpenCL(nx,ny,nz,ngrid,gx,gy,gz,val,thee,0);
03063
03064  free(ax);
03065  free(ay);
03066  free(az);
03067  free(charge);
03068  free(size);
03069
03070  free(gx);
03071  free(gy);
03072  free(gz);
03073  free(val);
03074 }
03075 #endif
03076
03077 VPRIVATE void packAtoms(double *ax, double *ay, double *az,
03078       double *charge, double *size, Vpmg *thee){
03079
03080  int i;
03081  int natoms;
03082
03083  Vatom *atom = VNULL;
03084  Valist *alist = VNULL;
03085
03086  alist = thee->pbe->alist;
03087  natoms = Valist_getNumberAtoms(alist);
03088
03089  for(i=0;i<natoms;i++){
03090   atom = &(alist->atoms[i]);
03091   charge[i] = Vunit_ec*atom->charge;
03092   ax[i] = atom->position[0];
03093   ay[i] = atom->position[1];
03094   az[i] = atom->position[2];
03095   size[i] = atom->radius;
03096  }
03097 }
03098
```

```
03099 /*
03100  Used by bcflnew
03101  */
03102 VPRIVATE void packUnpack(int nx, int ny, int nz, int ngrid,
03103        double *gx, double *gy, double *gz, double *value,
03104        Vpmg *thee, int pack){
03105
03106  int i,j,k,igrid;
03107  int x0,x1,y0,y1,z0,z1;
03108
03109  double gpos[3];
03110  double *xf, *yf, *zf;
03111  double *gxcf, *gycf, *gzcf;
03112
03113  xf = thee->xf;
03114  yf = thee->yf;
03115  zf = thee->zf;
03116
03117  gxcf = thee->gxcf;
03118  gycf = thee->gycf;
03119  gzcf = thee->gzcf;
03120
03121  igrid = 0;
03122  for(k=0;k<nz;k++){
03123   gpos[2] = zf[k];
03124   for(j=0;j<ny;j++){
03125    gpos[1] = yf[j];
03126    for(i=0;i<nx;i++){
03127     gpos[0] = xf[i];
03128     if(gridPointIsValid(i, j, k, nx, ny, nz)){
03129      if(pack != 0){
03130       gx[igrid] = gpos[0];
03131       gy[igrid] = gpos[1];
03132       gz[igrid] = gpos[2];
03133
03134       value[igrid] = 0.0;
03135      }else{
03136       x0 = IJKx(j,k,0);
03137       x1 = IJKx(j,k,1);
03138       y0 = IJKy(i,k,0);
03139       y1 = IJKy(i,k,1);
03140       z0 = IJKz(i,j,0);
03141       z1 = IJKz(i,j,1);
03142
03143       if(i==0){
03144        gxcf[x0] += value[igrid];
03145       }
03146       if(i==nx-1){
03147        gxcf[x1] += value[igrid];
03148       }
03149       if(j==0){
03150        gycf[y0] += value[igrid];
03151       }
03152       if(j==ny-1){
03153        gycf[y1] += value[igrid];
03154       }
03155       if(k==0){
```

```
03156         gzcf[z0] += value[igrid];
03157        }
03158        if(k==nz-1){
03159         gzcf[z1] += value[igrid];
03160        }
03161       }
03162
03163       igrid++;
03164      } //end is valid point
03165     } //end i
03166    } //end j
03167   } //end k
03168
03169 }
03170
03171 VPRIVATE void bcflnew(Vpmg *thee){
03172
03173   int i,j,k, iatom, igrid;
03174   int x0, x1, y0, y1, z0, z1;
03175
03176   int nx, ny, nz;
03177   int natoms, ngrid;
03178
03179   double dist, pre1, eps_w, eps_p, T, xkappa;
03180
03181   double *ax, *ay, *az;
03182   double *charge, *size, *val;
03183
03184   double *gx, *gy, *gz;
03185
03186   Vpbe *pbe = thee->pbe;
03187
03188   nx = thee->pmgp->nx;
03189      ny = thee->pmgp->ny;
03190      nz = thee->pmgp->nz;
03191
03192   eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
03193      eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
03194      T = Vpbe_getTemperature(pbe);               /* K             */
03195      pre1 = ((Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T))*(1.0e10);
03196   xkappa = Vpbe_getXkappa(pbe);
03197
03198   natoms = Valist_getNumberAtoms(thee->pbe->alist);
03199   ngrid = 2*((nx*ny) + (ny*nz) + (nx*nz));
03200
03201   ax = (double*)malloc(natoms * sizeof(double));
03202   ay = (double*)malloc(natoms * sizeof(double));
03203   az = (double*)malloc(natoms * sizeof(double));
03204
03205   charge = (double*)malloc(natoms * sizeof(double));
03206   size = (double*)malloc(natoms * sizeof(double));
03207
03208   gx = (double*)malloc(ngrid * sizeof(double));
03209   gy = (double*)malloc(ngrid * sizeof(double));
03210   gz = (double*)malloc(ngrid * sizeof(double));
03211
03212   val = (double*)malloc(ngrid * sizeof(double));
```

```
03213
03214  packAtoms(ax,ay,az,charge,size,thee);
03215  packUnpack(nx,ny,nz,ngrid,gx,gy,gz,val,thee,1);
03216
03217  if(xkappa > VSMALL){
03218 #pragma omp parallel for default(shared) private(igrid,iatom,dist)
03219   for(igrid=0;igrid<ngrid;igrid++){
03220    for(iatom=0; iatom<natoms; iatom++){
03221     dist = VSQRT(VSQR(gx[igrid]-ax[iatom]) + VSQR(gy[igrid]-ay[iatom])
03222         + VSQR(gz[igrid]-az[iatom]));
03223     val[igrid] += pre1*(charge[iatom]/dist)*VEXP(-xkappa*(dist-size[iatom]))
03224     / (1+xkappa*size[iatom]);
03225    }
03226   }
03227  }else{
03228 #pragma omp parallel for default(shared) private(igrid,iatom,dist)
03229   for(igrid=0;igrid<ngrid;igrid++){
03230    for(iatom=0; iatom<natoms; iatom++){
03231     dist = VSQRT(VSQR(gx[igrid]-ax[iatom]) + VSQR(gy[igrid]-ay[iatom])
03232         + VSQR(gz[igrid]-az[iatom]));
03233     val[igrid] += pre1*(charge[iatom]/dist);
03234    }
03235   }
03236  }
03237  packUnpack(nx,ny,nz,ngrid,gx,gy,gz,val,thee,0);
03238
03239  free(ax);
03240  free(ay);
03241  free(az);
03242  free(charge);
03243  free(size);
03244
03245  free(gx);
03246  free(gy);
03247  free(gz);
03248  free(val);
03249 }
03250
03251 VPRIVATE void multipolebc(double r, double kappa, double eps_p,
03252                           double eps_w, double rad, double tsr[3]) {
03253     double r2,r3,r5;
03254     double eps_r;
03255     double ka,ka2,ka3;
03256     double kr,kr2,kr3;
03257
03258     /*
03259   Below an attempt is made to explain the potential outside of a
03260   multipole located at the center of spherical cavity of dieletric
03261   eps_p, with dielectric eps_w outside (and possibly kappa > 0).
03262
03263
03264   First, eps_p = 1.0
03265   eps_w = 1.0
03266   kappa = 0.0
03267
03268   The general form for the potential of a traceless multipole tensor
03269   of rank n in vacuum is:
```

```
03270
03271   V(r) = (-1)^n * u . n . Del^n (1/r)
03272
03273   where
03274   u                    is a multipole of order n (3^n components)
03275   u . n. Del^n (1/r)    is the contraction of u with the nth
03276   derivative of 1/r
03277
03278   for example, if n = 1, the dipole potential is
03279   V_vac(r) = (-1)*[ux*x + uy*y + uz*z]/r^3
03280
03281   This function returns the parts of V(r) for multipoles of
03282   order 0, 1 and 2 that are independent of the contraction.
03283
03284   For the vacuum example, this would be 1/r, -1/r^3 and 3/r^5
03285   respectively.
03286
03287   *** Note that this requires that the quadrupole is
03288   traceless. If not, the diagonal quadrupole potential changes
03289   from
03290   qaa *  3*a^2/r^5
03291   to
03292   qaa * (3*a^2/r^5 - 1/r^3a )
03293   where we sum over the trace; a = x, y and z.
03294
03295   (In other words, the -1/r^3 term cancels for a traceless quadrupole.
03296   qxx + qyy + qzz = 0
03297   such that
03298   -(qxx + qyy + qzz)/r^3 = 0
03299
03300   For quadrupole with trace:
03301   qxx + qyy + qzz != 0
03302   such that
03303   -(qxx + qyy + qzz)/r^3 != 0
03304   )
03305
03306   ============================================================================
03307
03308   eps_p != 1 or eps_w != 1
03309   kappa = 0.0
03310
03311   If the multipole is placed at the center of a sphere with
03312   dieletric eps_p in a solvent of dielectric eps_w, the potential
03313   outside the sphere is the solution to the Laplace equation:
03314
03315   V(r) = 1/eps_w * (2*n+1)*eps_r/(n+(n+1)*eps_r)
03316   * (-1)^n * u . n . Del^n (1/r)
03317   where
03318   eps_r = eps_w / eps_p
03319   is the ratio of solvent to solute dielectric
03320
03321   ============================================================================
03322
03323   kappa > 0
03324
03325   Finally, if the region outside the sphere is treated by the linearized
03326   PB equation with Debye-Huckel parameter kappa, the solution is:
```

```
03327
03328   V(r) = kappa/eps_w * alpha_n(kappa*a) * K_n(kappa*r) * r^(n+1)/a^n
03329   * (-1)^n * u . n . Del^n (1/r)
03330   where
03331   alpha_n(x) is [(2n + 1) / x] / [(n*K_n(x)/eps_r) - x*K_n'(x)]
03332   K_n(x) are modified spherical Bessel functions of the third kind.
03333   K_n'(x) is the derivative of K_n(x)
03334      */
03335
03336     eps_r = eps_w/eps_p;
03337     r2 = r*r;
03338     r3 = r2*r;
03339     r5 = r3*r2;
03340     tsr[0] = (1.0/eps_w)/r;
03341     tsr[1] = (1.0/eps_w)*(-1.0)/r3;
03342     tsr[2] = (1.0/eps_w)*(3.0)/r5;
03343     if (kappa < VSMALL) {
03344         tsr[1] = (3.0*eps_r)/(1.0 + 2.0*eps_r)*tsr[1];
03345         tsr[2] = (5.0*eps_r)/(2.0 + 3.0*eps_r)*tsr[2];
03346     } else {
03347         ka = kappa*rad;
03348         ka2 = ka*ka;
03349         ka3 = ka2*ka;
03350         kr = kappa*r;
03351         kr2 = kr*kr;
03352         kr3 = kr2*kr;
03353         tsr[0] = exp(ka-kr) / (1.0 + ka) * tsr[0];
03354         tsr[1] = 3.0*eps_r*exp(ka-kr)*(1.0 + kr) * tsr[1];
03355         tsr[1] = tsr[1] / (1.0 + ka + eps_r*(2.0 + 2.0*ka + ka2));
03356         tsr[2] = 5.0*eps_r*exp(ka-kr)*(3.0 + 3.0*kr + kr2) * tsr[2];
03357         tsr[2] = tsr[2]/(6.0+6.0*ka+2.0*ka2+eps_r*(9.0+9.0*ka+4.0*ka2+ka3));
03358     }
03359 }
03360
03361 VPRIVATE void bcfl_sdh(Vpmg *thee){
03362
03363  int i,j,k,iatom;
03364  int nx, ny, nz;
03365
03366  double size, *position, charge, xkappa, eps_w, eps_p, T, pre, dist;
03367  double sdhcharge, sdhdipole[3], traced[9], sdhquadrupole[9];
03368  double *dipole, *quadrupole;
03369
03370  double val, *apos, gpos[3], tensor[3], qave;
03371  double ux, uy, uz, xr, yr, zr;
03372  double qxx,qxy,qxz,qyx,qyy,qyz,qzx,qzy,qzz;
03373
03374  double *xf, *yf, *zf;
03375  double *gxcf, *gycf, *gzcf;
03376
03377  Vpbe *pbe;
03378  Vatom *atom;
03379     Valist *alist;
03380
03381  pbe = thee->pbe;
03382  alist = thee->pbe->alist;
03383     nx = thee->pmgp->nx;
```

```
03384     ny = thee->pmgp->ny;
03385     nz = thee->pmgp->nz;
03386
03387   xf = thee->xf;
03388   yf = thee->yf;
03389   zf = thee->zf;
03390
03391   gxcf = thee->gxcf;
03392   gycf = thee->gycf;
03393   gzcf = thee->gzcf;
03394
03395   /* For each "atom" (only one for bcfl=1), we use the following formula to
03396    * calculate the boundary conditions:
03397    *    g(x) = \frac{q e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
03398    *           * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
03399    *           * 1/d
03400    * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
03401    * We only need to evaluate some of these prefactors once:
03402    *    pre1 = \frac{e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
03403    * which gives the potential as
03404    *    g(x) = pre1 * q/d * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
03405    */
03406     eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
03407     eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
03408     T = Vpbe_getTemperature(pbe);               /* K             */
03409
03410     pre = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*Vunit_kb*T);
03411     pre = pre*(1.0e10);
03412
03413   /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
03414    * m/A, then we will only need to deal with distances and sizes in
03415    * Angstroms rather than meters.                                        */
03416     xkappa = Vpbe_getXkappa(pbe);              /* A^{-1}        */
03417
03418   /* Solute size and position */
03419   size = Vpbe_getSoluteRadius(pbe);
03420   position = Vpbe_getSoluteCenter(pbe);
03421
03422   sdhcharge = 0.0;
03423   for (i=0; i<3; i++) sdhdipole[i] = 0.0;
03424   for (i=0; i<9; i++) sdhquadrupole[i] = 0.0;
03425
03426   for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
03427    atom = Valist_getAtom(alist, iatom);
03428    apos = Vatom_getPosition(atom);
03429    xr = apos[0] - position[0];
03430    yr = apos[1] - position[1];
03431    zr = apos[2] - position[2];
03432    switch (thee->chargeSrc) {
03433     case VCM_CHARGE:
03434      charge = Vatom_getCharge(atom);
03435      sdhcharge += charge;
03436      sdhdipole[0] += xr * charge;
03437      sdhdipole[1] += yr * charge;
03438      sdhdipole[2] += zr * charge;
03439      traced[0] = xr*xr*charge;
03440      traced[1] = xr*yr*charge;
```

```
03441     traced[2] = xr*zr*charge;
03442     traced[3] = yr*xr*charge;
03443     traced[4] = yr*yr*charge;
03444     traced[5] = yr*zr*charge;
03445     traced[6] = zr*xr*charge;
03446     traced[7] = zr*yr*charge;
03447     traced[8] = zr*zr*charge;
03448     qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03449     sdhquadrupole[0] += 1.5*(traced[0] - qave);
03450     sdhquadrupole[1] += 1.5*(traced[1]);
03451     sdhquadrupole[2] += 1.5*(traced[2]);
03452     sdhquadrupole[3] += 1.5*(traced[3]);
03453     sdhquadrupole[4] += 1.5*(traced[4] - qave);
03454     sdhquadrupole[5] += 1.5*(traced[5]);
03455     sdhquadrupole[6] += 1.5*(traced[6]);
03456     sdhquadrupole[7] += 1.5*(traced[7]);
03457     sdhquadrupole[8] += 1.5*(traced[8] - qave);
03458 #if defined(WITH_TINKER)
03459    case VCM_PERMANENT:
03460     charge = Vatom_getCharge(atom);
03461     dipole = Vatom_getDipole(atom);
03462     quadrupole = Vatom_getQuadrupole(atom);
03463     sdhcharge += charge;
03464     sdhdipole[0] += xr * charge;
03465     sdhdipole[1] += yr * charge;
03466     sdhdipole[2] += zr * charge;
03467     traced[0] = xr*xr*charge;
03468     traced[1] = xr*yr*charge;
03469     traced[2] = xr*zr*charge;
03470     traced[3] = yr*xr*charge;
03471     traced[4] = yr*yr*charge;
03472     traced[5] = yr*zr*charge;
03473     traced[6] = zr*xr*charge;
03474     traced[7] = zr*yr*charge;
03475     traced[8] = zr*zr*charge;
03476     sdhdipole[0] += dipole[0];
03477     sdhdipole[1] += dipole[1];
03478     sdhdipole[2] += dipole[2];
03479     traced[0] += 2.0*xr*dipole[0];
03480     traced[1] += xr*dipole[1] + yr*dipole[0];
03481     traced[2] += xr*dipole[2] + zr*dipole[0];
03482     traced[3] += yr*dipole[0] + xr*dipole[1];
03483     traced[4] += 2.0*yr*dipole[1];
03484     traced[5] += yr*dipole[2] + zr*dipole[1];
03485     traced[6] += zr*dipole[0] + xr*dipole[2];
03486     traced[7] += zr*dipole[1] + yr*dipole[2];
03487     traced[8] += 2.0*zr*dipole[2];
03488     qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03489     sdhquadrupole[0] += 1.5*(traced[0] - qave);
03490     sdhquadrupole[1] += 1.5*(traced[1]);
03491     sdhquadrupole[2] += 1.5*(traced[2]);
03492     sdhquadrupole[3] += 1.5*(traced[3]);
03493     sdhquadrupole[4] += 1.5*(traced[4] - qave);
03494     sdhquadrupole[5] += 1.5*(traced[5]);
03495     sdhquadrupole[6] += 1.5*(traced[6]);
03496     sdhquadrupole[7] += 1.5*(traced[7]);
03497     sdhquadrupole[8] += 1.5*(traced[8] - qave);
```

```
03498     sdhquadrupole[0] += quadrupole[0];
03499     sdhquadrupole[1] += quadrupole[1];
03500     sdhquadrupole[2] += quadrupole[2];
03501     sdhquadrupole[3] += quadrupole[3];
03502     sdhquadrupole[4] += quadrupole[4];
03503     sdhquadrupole[5] += quadrupole[5];
03504     sdhquadrupole[6] += quadrupole[6];
03505     sdhquadrupole[7] += quadrupole[7];
03506     sdhquadrupole[8] += quadrupole[8];
03507    case VCM_INDUCED:
03508     dipole = Vatom_getInducedDipole(atom);
03509     sdhdipole[0] += dipole[0];
03510     sdhdipole[1] += dipole[1];
03511     sdhdipole[2] += dipole[2];
03512     traced[0] = 2.0*xr*dipole[0];
03513     traced[1] = xr*dipole[1] + yr*dipole[0];
03514     traced[2] = xr*dipole[2] + zr*dipole[0];
03515     traced[3] = yr*dipole[0] + xr*dipole[1];
03516     traced[4] = 2.0*yr*dipole[1];
03517     traced[5] = yr*dipole[2] + zr*dipole[1];
03518     traced[6] = zr*dipole[0] + xr*dipole[2];
03519     traced[7] = zr*dipole[1] + yr*dipole[2];
03520     traced[8] = 2.0*zr*dipole[2];
03521     qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03522     sdhquadrupole[0] += 1.5*(traced[0] - qave);
03523     sdhquadrupole[1] += 1.5*(traced[1]);
03524     sdhquadrupole[2] += 1.5*(traced[2]);
03525     sdhquadrupole[3] += 1.5*(traced[3]);
03526     sdhquadrupole[4] += 1.5*(traced[4] - qave);
03527     sdhquadrupole[5] += 1.5*(traced[5]);
03528     sdhquadrupole[6] += 1.5*(traced[6]);
03529     sdhquadrupole[7] += 1.5*(traced[7]);
03530     sdhquadrupole[8] += 1.5*(traced[8] - qave);
03531    case VCM_NLINDUCED:
03532     dipole = Vatom_getNLInducedDipole(atom);
03533     sdhdipole[0] += dipole[0];
03534     sdhdipole[1] += dipole[1];
03535     sdhdipole[2] += dipole[2];
03536     traced[0] = 2.0*xr*dipole[0];
03537     traced[1] = xr*dipole[1] + yr*dipole[0];
03538     traced[2] = xr*dipole[2] + zr*dipole[0];
03539     traced[3] = yr*dipole[0] + xr*dipole[1];
03540     traced[4] = 2.0*yr*dipole[1];
03541     traced[5] = yr*dipole[2] + zr*dipole[1];
03542     traced[6] = zr*dipole[0] + xr*dipole[2];
03543     traced[7] = zr*dipole[1] + yr*dipole[2];
03544     traced[8] = 2.0*zr*dipole[2];
03545     qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03546     sdhquadrupole[0] += 1.5*(traced[0] - qave);
03547     sdhquadrupole[1] += 1.5*(traced[1]);
03548     sdhquadrupole[2] += 1.5*(traced[2]);
03549     sdhquadrupole[3] += 1.5*(traced[3]);
03550     sdhquadrupole[4] += 1.5*(traced[4] - qave);
03551     sdhquadrupole[5] += 1.5*(traced[5]);
03552     sdhquadrupole[6] += 1.5*(traced[6]);
03553     sdhquadrupole[7] += 1.5*(traced[7]);
03554     sdhquadrupole[8] += 1.5*(traced[8] - qave);
```

```
03555 #endif /* if defined(WITH_TINKER) */
03556   }
03557  }
03558
03559  ux = sdhdipole[0];
03560  uy = sdhdipole[1];
03561     uz = sdhdipole[2];
03562
03563  /* The factor of 1/3 results from using a
03564   traceless quadrupole definition. See, for example,
03565   "The Theory of Intermolecular Forces" by A.J. Stone,
03566   Chapter 3. */
03567  qxx = sdhquadrupole[0] / 3.0;
03568  qxy = sdhquadrupole[1] / 3.0;
03569  qxz = sdhquadrupole[2] / 3.0;
03570  qyx = sdhquadrupole[3] / 3.0;
03571  qyy = sdhquadrupole[4] / 3.0;
03572  qyz = sdhquadrupole[5] / 3.0;
03573  qzx = sdhquadrupole[6] / 3.0;
03574  qzy = sdhquadrupole[7] / 3.0;
03575  qzz = sdhquadrupole[8] / 3.0;
03576
03577  for(k=0;k<nz;k++){
03578   gpos[2] = zf[k];
03579   for(j=0;j<ny;j++){
03580    gpos[1] = yf[j];
03581    for(i=0;i<nx;i++){
03582     gpos[0] = xf[i];
03583     if(gridPointIsValid(i, j, k, nx, ny, nz)){
03584      xr = gpos[0] - position[0];
03585      yr = gpos[1] - position[1];
03586      zr = gpos[2] - position[2];
03587
03588      dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
03589      multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
03590
03591      val = pre*sdhcharge*tensor[0];
03592      val -= pre*ux*xr*tensor[1];
03593      val -= pre*uy*yr*tensor[1];
03594      val -= pre*uz*zr*tensor[1];
03595      val += pre*qxx*xr*xr*tensor[2];
03596      val += pre*qyy*yr*yr*tensor[2];
03597      val += pre*qzz*zr*zr*tensor[2];
03598      val += pre*2.0*qxy*xr*yr*tensor[2];
03599      val += pre*2.0*qxz*xr*zr*tensor[2];
03600      val += pre*2.0*qyz*yr*zr*tensor[2];
03601
03602      if(i==0){
03603       gxcf[IJKx(j,k,0)] = val;
03604      }
03605      if(i==nx-1){
03606       gxcf[IJKx(j,k,1)] = val;
03607      }
03608      if(j==0){
03609       gycf[IJKy(i,k,0)] = val;
03610      }
03611      if(j==ny-1){
```

```
03612       gycf[IJKy(i,k,1)] = val;
03613      }
03614      if(k==0){
03615       gzcf[IJKz(i,j,0)] = val;
03616      }
03617      if(k==nz-1){
03618       gzcf[IJKz(i,j,1)] = val;
03619      }
03620     } /* End grid point is valid */
03621    } /* End i loop */
03622   } /* End j loop */
03623  } /* End k loop */
03624
03625 }
03626
03627 VPRIVATE void bcfl_mdh(Vpmg *thee){
03628
03629  int i,j,k,iatom;
03630  int nx, ny, nz;
03631
03632  double val, *apos, gpos[3];
03633  double *dipole, *quadrupole;
03634  double size, charge, xkappa, eps_w, eps_p, T, pre1, dist;
03635
03636  double *xf, *yf, *zf;
03637  double *gxcf, *gycf, *gzcf;
03638
03639  Vpbe *pbe;
03640  Vatom *atom;
03641     Valist *alist;
03642
03643  pbe = thee->pbe;
03644  alist = thee->pbe->alist;
03645     nx = thee->pmgp->nx;
03646     ny = thee->pmgp->ny;
03647     nz = thee->pmgp->nz;
03648
03649  xf = thee->xf;
03650  yf = thee->yf;
03651  zf = thee->zf;
03652
03653  gxcf = thee->gxcf;
03654  gycf = thee->gycf;
03655  gzcf = thee->gzcf;
03656
03657  /* For each "atom" (only one for bcfl=1), we use the following formula to
03658   * calculate the boundary conditions:
03659   *    g(x) = \frac{q e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
03660   *           * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
03661   *           * 1/d
03662   * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
03663   * We only need to evaluate some of these prefactors once:
03664   *    pre1 = \frac{e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
03665   * which gives the potential as
03666   *    g(x) = pre1 * q/d * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
03667   */
03668     eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
```

```
03669     eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
03670     T = Vpbe_getTemperature(pbe);               /* K             */
03671     pre1 = (Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T);
03672
03673     /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
03674      * m/A, then we will only need to deal with distances and sizes in
03675      * Angstroms rather than meters.                                    */
03676     xkappa = Vpbe_getXkappa(pbe);              /* A^{-1}        */
03677     pre1 = pre1*(1.0e10);
03678
03679  /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 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
03684  for(k=0;k<nz;k++){
03685   gpos[2] = zf[k];
03686   for(j=0;j<ny;j++){
03687    gpos[1] = yf[j];
03688    for(i=0;i<nx;i++){
03689     gpos[0] = xf[i];
03690     if(gridPointIsValid(i, j, k, nx, ny, nz)){
03691
03692      val = 0.0;
03693
03694      for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
03695       atom = Valist_getAtom(alist, iatom);
03696       apos = Vatom_getPosition(atom);
03697       charge = Vunit_ec*Vatom_getCharge(atom);
03698       size = Vatom_getRadius(atom);
03699
03700       dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
03701           + VSQR(gpos[2]-apos[2]));
03702       if (xkappa > VSMALL) {
03703        val += pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
03704        / (1+xkappa*size);
03705       } else {
03706        val += pre1*(charge/dist);
03707       }
03708
03709      }
03710
03711      if(i==0){
03712       gxcf[IJKx(j,k,0)] = val;
03713      }
03714      if(i==nx-1){
03715       gxcf[IJKx(j,k,1)] = val;
03716      }
03717      if(j==0){
03718       gycf[IJKy(i,k,0)] = val;
03719      }
03720      if(j==ny-1){
03721       gycf[IJKy(i,k,1)] = val;
03722      }
03723      if(k==0){
03724       gzcf[IJKz(i,j,0)] = val;
03725      }
```

```
03726       if(k==nz-1){
03727        gzcf[IJKz(i,j,1)] = val;
03728       }
03729      } /* End grid point is valid */
03730     } /* End i loop */
03731    } /* End j loop */
03732   } /* End k loop */
03733
03734 }
03735
03736 /* ////////////////////////////////////////////////////////////////////////
03737  // Routine:  bcfl_mem
03738  //
03739  // Purpose:  Increment all the boundary points by the
03740  //           analytic expression for a membrane system in
03741  //           the presence of a membrane potential. This
03742  //           Boundary flag should only be used for systems
03743  //           that explicitly have membranes in the dielectric
03744  //           and solvent maps.
03745  //
03746  //           There should be several input variables add to this
03747  //           function such as membrane potential, membrane thickness
03748  //           and height.
03749  //
03750  // Args:     apos is a 3-vector
03751  //
03752  // Author: Michael Grabe
03754 VPRIVATE void bcfl_mem(double zmem, double L, double eps_m, double eps_w,
03755      double V, double xkappa, double *gxcf, double *gycf, double *gzcf,
03756      double *xf, double *yf, double *zf, int nx, int ny, int nz) {
03757
03759     /* some definitions                              */
03760     /* L = total length of the membrane              */
03761     /* xkappa = inverse Debeye length                */
03762     /* zmem = z value of membrane bottom (Cytoplasm) */
03763     /* V = electrical potential inside the cell      */
03765     int i, j, k;
03766     double dist, val, z_low, z_high, z_shift;
03767     double A, B, C, D, edge_L, l;
03768     double G, z_0, z_rel;
03769     double gpos[3];
03770
03771  printf("Here is the value of kappa: %f\n",xkappa);
03772  printf("Here is the value of L: %f\n",L);
03773  printf("Here is the value of zmem: %f\n",zmem);
03774  printf("Here is the value of mdie: %f\n",eps_m);
03775  printf("Here is the value of memv: %f\n",V);
03776
03777  /* no salt symmetric BC's at +/- infinity */
03778  // B=V/(edge_L - l*(1-eps_w/eps_m));
03779  // A=V + B*edge_L;
03780  // D=eps_w/eps_m*B;
03781  z_low = zmem;     /* this defines the bottom of the membrane */
03782  z_high = zmem + L;  /* this is the top of the membrane */
03783
03784  /*****************************************************/
03785  /* proper boundary conditions for V = 0 extracellular */
```

```
03786  /* and psi=-V cytoplasm.                                  */
03787  /* Implicit in this formulation is that the membrane  */
03788  /* center be at z = 0                                     */
03789  /*********************************************************/
03790
03791  l=L/2;                    /* half of the membrane length */
03792  z_0 = z_low + l;          /* center of the membrane      */
03793  G=l*eps_w/eps_m*xkappa;
03794  A=-V/2*(1/(G+1))*exp(xkappa*l);
03795  B=V/2;
03796  C=-V/2*eps_w/eps_m*xkappa*(1/(G+1));
03797  D=-A;
03798  /* The analytic expression for the boundary conditions      */
03799  /* had the cytoplasmic surface of the membrane set to zero. */
03800  /* This requires an off-set of the BC equations.            */
03801
03802     /* the "i" boundaries (dirichlet) */
03803     for (k=0; k<nz; k++) {
03804         gpos[2] = zf[k];
03805         z_rel = gpos[2] - z_0;    /* relative position for BCs */
03806
03807         for (j=0; j<ny; j++) {
03808
03809    if (gpos[2] <= z_low) {                       /* cytoplasmic */
03810
03811     val = A*exp(xkappa*z_rel) + V;
03812     gxcf[IJKx(j,k,0)] += val;    /* assign low side BC */
03813     gxcf[IJKx(j,k,1)] += val;    /* assign high side BC */
03814
03815    }
03816
03817    else if (gpos[2] > z_low && gpos[2] <= z_high) {  /* in membrane */
03818
03819     val = B + C*z_rel;
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_high)  {                  /* extracellular */
03826
03827     val = D*exp(-xkappa*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         }
03834     }
03835
03836     /* the "j" boundaries (dirichlet) */
03837     for (k=0; k<nz; k++) {
03838         gpos[2] = zf[k];
03839         z_rel = gpos[2] - z_0;
03840         for (i=0; i<nx; i++) {
03841
03842    if (gpos[2] <= z_low) {                       /* cytoplasmic */
```

```
03843
03844     val = A*exp(xkappa*z_rel) + V;
03845     gycf[IJKy(i,k,0)] += val;    /* assign low side BC */
03846     gycf[IJKy(i,k,1)] += val;    /* assign high side BC */
03847     //printf("%f \n",val);
03848
03849    }
03850
03851    else if (gpos[2] > z_low && gpos[2] <= z_high) {  /* in membrane */
03852
03853     val = B + C*z_rel;
03854     gycf[IJKy(i,k,0)] += val;    /* assign low side BC */
03855     gycf[IJKy(i,k,1)] += val;    /* assign high side BC */
03856     //printf("%f \n",val);
03857
03858    }
03859    else if (gpos[2] > z_high)  {                 /* extracellular */
03860
03861     val = D*exp(-xkappa*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
03868         }
03869     }
03870
03871     /* the "k" boundaries (dirichlet) */
03872     for (j=0; j<ny; j++) {
03873         for (i=0; i<nx; i++) {
03874
03875    /* first assign the bottom boundary */
03876
03877    gpos[2] = zf[0];
03878    z_rel = gpos[2] - z_0;
03879
03880    if (gpos[2] <= z_low) {                       /* cytoplasmic */
03881
03882     val = A*exp(xkappa*z_rel) + V;
03883     gzcf[IJKz(i,j,0)] += val;    /* assign low side BC */
03884     //printf("%f \n",val);
03885
03886    }
03887
03888    else if (gpos[2] > z_low && gpos[2] <= z_high) {  /* in membrane */
03889
03890     val = B + C*z_rel;
03891     gzcf[IJKz(i,j,0)] += val;    /* assign low side BC */
03892
03893    }
03894
03895    else if (gpos[2] > z_high)  {                 /* extracellular */
03896
03897     val = D*exp(-xkappa*z_rel);
03898     gzcf[IJKz(i,j,0)] += val;    /* assign low side BC */
03899
```

```
03900     }
03901
03902     /* now assign the top boundary */
03903
03904     gpos[2] = zf[nz-1];
03905     z_rel = gpos[2] - z_0;
03906
03907     if (gpos[2] <= z_low) {                        /* cytoplasmic */
03908
03909      val = A*exp(xkappa*z_rel) + V;
03910      gzcf[IJKz(i,j,1)] += val;    /* assign high side BC */
03911
03912     }
03913
03914     else if (gpos[2] > z_low && gpos[2] <= z_high) {  /* in membrane */
03915
03916      val = B + C*z_rel;
03917      gzcf[IJKz(i,j,1)] += val;    /* assign high side BC */
03918
03919     }
03920
03921     else if (gpos[2] > z_high)  {                   /* extracellular */
03922
03923      val = D*exp(-xkappa*z_rel);
03924      gzcf[IJKz(i,j,1)] += val;    /* assign high side BC */
03925      //printf("%f \n",val);
03926
03927     }
03928
03929         }
03930      }
03931 }
03932
03933 VPRIVATE void bcfl_map(Vpmg *thee){
03934
03935  Vpbe *pbe;
03936     double position[3], pot, hx, hy, hzed;
03937     int i, j, k, nx, ny, nz, rc;
03938
03939
03940     VASSERT(thee != VNULL);
03941
03942     /* Mesh info */
03943     nx = thee->pmgp->nx;
03944     ny = thee->pmgp->ny;
03945     nz = thee->pmgp->nz;
03946     hx = thee->pmgp->hx;
03947     hy = thee->pmgp->hy;
03948     hzed = thee->pmgp->hzed;
03949
03950     /* Reset the potential array */
03951     for (i=0; i<(nx*ny*nz); i++) thee->pot[i] = 0.0;
03952
03953     /* Fill in the source term (atomic potentials) */
03954     Vnm_print(0, "Vpmg_fillco:  filling in source term.\n");
03955     for (k=0; k<nz; k++) {
03956         for (j=0; j<ny; j++) {
```

```
03957             for (i=0; i<nx; i++) {
03958                 position[0] = thee->xf[i];
03959                 position[1] = thee->yf[j];
03960                 position[2] = thee->zf[k];
03961     rc = Vgrid_value(thee->potMap, position, &pot);
03962     if (!rc) {
03963      Vnm_print(2, "fillcoChargeMap:  Error -- fell off of potential map at (%g, %
      g, %g)!\n",
03964           position[0], position[1], position[2]);
03965      VASSERT(0);
03966     }
03967                 thee->pot[IJK(i,j,k)] = pot;
03968             }
03969         }
03970     }
03971
03972 }
03973
03974 #if  defined(WITH_TINKER)
03975 VPRIVATE void bcfl_mdh_tinker(Vpmg *thee){
03976
03977  int i,j,k,iatom;
03978  int nx, ny, nz;
03979
03980  double val, *apos, gpos[3], tensor[9];
03981  double *dipole, *quadrupole;
03982  double size, charge, xkappa, eps_w, eps_p, T, pre1, dist;
03983
03984  double ux,uy,uz,xr,yr,zr;
03985  double qxx,qxy,qxz,qyx,qyy,qyz,qzx,qzy,qzz;
03986
03987  double *xf, *yf, *zf;
03988  double *gxcf, *gycf, *gzcf;
03989
03990  Vpbe *pbe;
03991  Vatom *atom;
03992     Valist *alist;
03993
03994  pbe = thee->pbe;
03995  alist = thee->pbe->alist;
03996     nx = thee->pmgp->nx;
03997     ny = thee->pmgp->ny;
03998     nz = thee->pmgp->nz;
03999
04000  xf = thee->xf;
04001  yf = thee->yf;
04002  zf = thee->zf;
04003
04004  gxcf = thee->gxcf;
04005  gycf = thee->gycf;
04006  gzcf = thee->gzcf;
04007
04008  /* For each "atom" (only one for bcfl=1), we use the following formula to
04009   * calculate the boundary conditions:
04010   *    g(x) = \frac{q e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
04011   *           * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
04012   *           * 1/d
```

```
04013   * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
04014   * We only need to evaluate some of these prefactors once:
04015   *    pre1 = \frac{e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
04016   * which gives the potential as
04017   *    g(x) = pre1 * q/d * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
04018   */
04019     eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
04020     eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
04021     T = Vpbe_getTemperature(pbe);               /* K             */
04022     pre1 = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*Vunit_kb*T);
04023
04024     /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
04025   * m/A, then we will only need to deal with distances and sizes in
04026   * Angstroms rather than meters.                                        */
04027     xkappa = Vpbe_getXkappa(pbe);              /* A^{-1}        */
04028     pre1 = pre1*(1.0e10);
04029
04030  /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
04031   * m/A, then we will only need to deal with distances and sizes in
04032   * Angstroms rather than meters.                                        */
04033     xkappa = Vpbe_getXkappa(pbe);              /* A^{-1}        */
04034
04035  for(k=0;k<nz;k++){
04036   gpos[2] = zf[k];
04037   for(j=0;j<ny;j++){
04038    gpos[1] = yf[j];
04039    for(i=0;i<nx;i++){
04040     gpos[0] = xf[i];
04041     if(gridPointIsValid(i, j, k, nx, ny, nz)){
04042
04043      val = 0.0;
04044
04045      for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
04046       atom = Valist_getAtom(alist, iatom);
04047       apos = Vatom_getPosition(atom);
04048       size = Vatom_getRadius(atom);
04049
04050       charge = 0.0;
04051
04052       dipole = VNULL;
04053       quadrupole = VNULL;
04054
04055       if (thee->chargeSrc == VCM_PERMANENT) {
04056        charge = Vatom_getCharge(atom);
04057        dipole = Vatom_getDipole(atom);
04058        quadrupole = Vatom_getQuadrupole(atom);
04059       } else if (thee->chargeSrc == VCM_INDUCED) {
04060        dipole = Vatom_getInducedDipole(atom);
04061       } else {
04062        dipole = Vatom_getNLInducedDipole(atom);
04063       }
04064
04065       ux = dipole[0];
04066       uy = dipole[1];
04067       uz = dipole[2];
04068
04069       if (quadrupole != VNULL) {
```

```
04070         /* The factor of 1/3 results from using a
04071          traceless quadrupole definition. See, for example,
04072          "The Theory of Intermolecular Forces" by A.J. Stone,
04073          Chapter 3. */
04074         qxx = quadrupole[0] / 3.0;
04075         qxy = quadrupole[1] / 3.0;
04076         qxz = quadrupole[2] / 3.0;
04077         qyx = quadrupole[3] / 3.0;
04078         qyy = quadrupole[4] / 3.0;
04079         qyz = quadrupole[5] / 3.0;
04080         qzx = quadrupole[6] / 3.0;
04081         qzy = quadrupole[7] / 3.0;
04082         qzz = quadrupole[8] / 3.0;
04083        } else {
04084         qxx = 0.0;
04085         qxy = 0.0;
04086         qxz = 0.0;
04087         qyx = 0.0;
04088         qyy = 0.0;
04089         qyz = 0.0;
04090         qzx = 0.0;
04091         qzy = 0.0;
04092         qzz = 0.0;
04093        }
04094
04095        xr = gpos[0] - apos[0];
04096        yr = gpos[1] - apos[1];
04097        zr = gpos[2] - apos[2];
04098
04099        dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
04100        multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
04101
04102        val += pre1*charge*tensor[0];
04103        val -= pre1*ux*xr*tensor[1];
04104        val -= pre1*uy*yr*tensor[1];
04105        val -= pre1*uz*zr*tensor[1];
04106        val += pre1*qxx*xr*xr*tensor[2];
04107        val += pre1*qyy*yr*yr*tensor[2];
04108        val += pre1*qzz*zr*zr*tensor[2];
04109        val += pre1*2.0*qxy*xr*yr*tensor[2];
04110        val += pre1*2.0*qxz*xr*zr*tensor[2];
04111        val += pre1*2.0*qyz*yr*zr*tensor[2];
04112
04113       }
04114
04115       if(i==0){
04116        gxcf[IJKx(j,k,0)] = val;
04117       }
04118       if(i==nx-1){
04119        gxcf[IJKx(j,k,1)] = val;
04120       }
04121       if(j==0){
04122        gycf[IJKy(i,k,0)] = val;
04123       }
04124       if(j==ny-1){
04125        gycf[IJKy(i,k,1)] = val;
04126       }
```

```
04127       if(k==0){
04128        gzcf[IJKz(i,j,0)] = val;
04129       }
04130       if(k==nz-1){
04131        gzcf[IJKz(i,j,1)] = val;
04132       }
04133      } /* End grid point is valid */
04134     } /* End i loop */
04135    } /* End j loop */
04136   } /* End k loop */
04137
04138 }
04139 #endif
04140
04141 VPRIVATE void bcCalc(Vpmg *thee){
04142
04143  int i, j, k;
04144  int nx, ny, nz;
04145
04146  double zmem, eps_m, Lmem, memv, eps_w, xkappa;
04147
04148     nx = thee->pmgp->nx;
04149     ny = thee->pmgp->ny;
04150     nz = thee->pmgp->nz;
04151
04152     /* Zero out the boundaries */
04153     /* the "i" boundaries (dirichlet) */
04154     for (k=0; k<nz; k++) {
04155         for (j=0; j<ny; j++) {
04156             thee->gxcf[IJKx(j,k,0)] = 0.0;
04157             thee->gxcf[IJKx(j,k,1)] = 0.0;
04158             thee->gxcf[IJKx(j,k,2)] = 0.0;
04159             thee->gxcf[IJKx(j,k,3)] = 0.0;
04160         }
04161     }
04162
04163     /* the "j" boundaries (dirichlet) */
04164     for (k=0; k<nz; k++) {
04165         for (i=0; i<nx; i++) {
04166             thee->gycf[IJKy(i,k,0)] = 0.0;
04167             thee->gycf[IJKy(i,k,1)] = 0.0;
04168             thee->gycf[IJKy(i,k,2)] = 0.0;
04169             thee->gycf[IJKy(i,k,3)] = 0.0;
04170         }
04171     }
04172
04173     /* the "k" boundaries (dirichlet) */
04174     for (j=0; j<ny; j++) {
04175         for (i=0; i<nx; i++) {
04176             thee->gzcf[IJKz(i,j,0)] = 0.0;
04177             thee->gzcf[IJKz(i,j,1)] = 0.0;
04178             thee->gzcf[IJKz(i,j,2)] = 0.0;
04179             thee->gzcf[IJKz(i,j,3)] = 0.0;
04180         }
04181     }
04182
04183  switch (thee->pmgp->bcfl) {
```

```
04184    /*  If we have zero boundary conditions, we're done */
04185         case BCFL_ZERO:
04186             return;
04187   case BCFL_SDH:
04188    bcfl_sdh(thee);
04189    break;
04190   case BCFL_MDH:
04191 #if defined(WITH_TINKER)
04192    bcfl_mdh_tinker(thee);
04193 #else
04194
04195 #ifdef DEBUG_MAC_OSX_OCL
04196 #include "mach_chud.h"
04197    uint64_t mbeg = mach_absolute_time();
04198
04199    /*
04200     * If OpenCL is available we use it, otherwise fall back to
04201     * normal route (CPU multithreaded w/ OpenMP)
04202     */
04203    if (kOpenCLAvailable == 1) bcflnewOpenCL(thee);
04204    else bcflnew(thee);
04205
04206    mets_(&mbeg, "MDH");
04207 #else
04208    /* bcfl_mdh(thee); */
04209    bcflnew(thee);
04210 #endif /* DEBUG_MAC_OSX_OCL */
04211
04212 #endif /* WITH_TINKER */
04213    break;
04214   case BCFL_MEM:
04215
04216    zmem  = Vpbe_getzmem(thee->pbe);
04217    Lmem  = Vpbe_getLmem(thee->pbe);
04218    eps_m = Vpbe_getmembraneDiel(thee->pbe);
04219    memv =  Vpbe_getmemv(thee->pbe);
04220
04221    eps_w = Vpbe_getSolventDiel(thee->pbe);
04222    xkappa = Vpbe_getXkappa(thee->pbe);
04223
04224    bcfl_mem(zmem, Lmem, eps_m, eps_w, memv, xkappa,
04225       thee->gxcf, thee->gycf, thee->gzcf,
04226       thee->xf, thee->yf, thee->zf, nx, ny, nz);
04227    break;
04228         case BCFL_UNUSED:
04229             Vnm_print(2, "bcCalc:  Invalid bcfl (%d)!\n", thee->pmgp->bcfl);
04230             VASSERT(0);
04231    break;
04232         case BCFL_FOCUS:
04233             Vnm_print(2, "VPMG::bcCalc -- not appropriate for focusing!\n");
04234             VASSERT(0);
04235    break;
04236   case BCFL_MAP:
04237    bcfl_map(thee);
04238    focusFillBound(thee,VNULL);
04239    break;
04240         default:
```

```
04241             Vnm_print(2, "VPMG::bcCalc -- invalid boundary condition \
04242         flag (%d)!\n", thee->pmgp->bcfl);
04243             VASSERT(0);
04244     break;
04245      }
04246 }
04247
04248 VPRIVATE void fillcoCoefMap(Vpmg *thee) {
04249
04250     Vpbe *pbe;
04251     double ionstr, position[3], tkappa, eps, pot, hx, hy, hzed;
04252     int i, j, k, nx, ny, nz;
04253     double kappamax;
04254     VASSERT(thee != VNULL);
04255
04256     /* Get PBE info */
04257     pbe = thee->pbe;
04258     ionstr = Vpbe_getBulkIonicStrength(pbe);
04259
04260     /* Mesh info */
04261     nx = thee->pmgp->nx;
04262     ny = thee->pmgp->ny;
04263     nz = thee->pmgp->nz;
04264     hx = thee->pmgp->hx;
04265     hy = thee->pmgp->hy;
04266     hzed = thee->pmgp->hzed;
04267
04268     if ((!thee->useDielXMap) || (!thee->useDielYMap)
04269   || (!thee->useDielZMap) || ((!thee->useKappaMap) && (ionstr>VPMGSMALL))) {
04270
04271         Vnm_print(2, "fillcoCoefMap:  You need to use all coefficient maps!\n");
04272         VASSERT(0);
04273
04274     }
04275
04276     /* Scale the kappa map to values between 0 and 1
04277        Thus get the maximum value in the map - this
04278        is theoretically unnecessary, but a good check.*/
04279     kappamax = -1.00;
04280     for (k=0; k<nz; k++) {
04281         for (j=0; j<ny; j++) {
04282             for (i=0; i<nx; i++) {
04283                 if (ionstr > VPMGSMALL) {
04284                     position[0] = thee->xf[i];
04285                     position[1] = thee->yf[j];
04286                     position[2] = thee->zf[k];
04287                     if (!Vgrid_value(thee->kappaMap, position, &tkappa)) {
04288                         Vnm_print(2, "Vpmg_fillco:  Off kappaMap at:\n");
04289                         Vnm_print(2, "Vpmg_fillco:  (x,y,z) = (%g,%g %g)\n",
04290                                   position[0], position[1], position[2]);
04291                         VASSERT(0);
04292                     }
04293                     if (tkappa > kappamax) {
04294                         kappamax = tkappa;
04295                     }
04296                     if (tkappa < 0.0){
04297                       Vnm_print(2, "Vpmg_fillcoCoefMap: Kappa map less than 0\n"
```

```
      );
04298                         Vnm_print(2, "Vpmg_fillcoCoefMap: at (x,y,z) = (%g,%g %g)\
      n",
04299                                   position[0], position[1], position[2]);
04300                         VASSERT(0);
04301                       }
04302                   }
04303               }
04304           }
04305       }
04306
04307       if (kappamax > 1.0){
04308         Vnm_print(2, "Vpmg_fillcoCoefMap:  Maximum Kappa value\n");
04309         Vnm_print(2, "%g is greater than 1 - will scale appropriately!\n",
04310                   kappamax);
04311       }
04312       else {
04313         kappamax = 1.0;
04314       }
04315
04316       for (k=0; k<nz; k++) {
04317           for (j=0; j<ny; j++) {
04318               for (i=0; i<nx; i++) {
04319
04320                   if (ionstr > VPMGSMALL) {
04321                        position[0] = thee->xf[i];
04322                        position[1] = thee->yf[j];
04323                        position[2] = thee->zf[k];
04324                        if (!Vgrid_value(thee->kappaMap, position, &tkappa)) {
04325                            Vnm_print(2, "Vpmg_fillco:  Off kappaMap at:\n");
04326                            Vnm_print(2, "Vpmg_fillco:  (x,y,z) = (%g,%g %g)\n",
04327                              position[0], position[1], position[2]);
04328                            VASSERT(0);
04329                        }
04330                        if (tkappa < VPMGSMALL) tkappa = 0.0;
04331                        thee->kappa[IJK(i,j,k)] = (tkappa / kappamax);
04332                   }
04333
04334                   position[0] = thee->xf[i] + 0.5*hx;
04335                   position[1] = thee->yf[j];
04336                   position[2] = thee->zf[k];
04337                   if (!Vgrid_value(thee->dielXMap, position, &eps)) {
04338                       Vnm_print(2, "Vpmg_fillco:  Off dielXMap at:\n");
04339                       Vnm_print(2, "Vpmg_fillco:  (x,y,z) = (%g,%g %g)\n",
04340                         position[0], position[1], position[2]);
04341                       VASSERT(0);
04342                    }
04343                    thee->epsx[IJK(i,j,k)] = eps;
04344
04345                    position[0] = thee->xf[i];
04346                    position[1] = thee->yf[j] + 0.5*hy;
04347                    position[2] = thee->zf[k];
04348                    if (!Vgrid_value(thee->dielYMap, position, &eps)) {
04349                       Vnm_print(2, "Vpmg_fillco:  Off dielYMap at:\n");
04350                       Vnm_print(2, "Vpmg_fillco:  (x,y,z) = (%g,%g %g)\n",
04351                         position[0], position[1], position[2]);
04352                       VASSERT(0);
```

```
04353                   }
04354                   thee->epsy[IJK(i,j,k)] = eps;
04355
04356                   position[0] = thee->xf[i];
04357                   position[1] = thee->yf[j];
04358                   position[2] = thee->zf[k] + 0.5*hzed;
04359                   if (!Vgrid_value(thee->dielZMap, position, &eps)) {
04360                      Vnm_print(2, "Vpmg_fillco:  Off dielZMap at:\n");
04361                      Vnm_print(2, "Vpmg_fillco:  (x,y,z) = (%g,%g %g)\n",
04362                        position[0], position[1], position[2]);
04363                      VASSERT(0);
04364                   }
04365                   thee->epsz[IJK(i,j,k)] = eps;
04366               }
04367           }
04368       }
04369 }
04370
04371 VPRIVATE void fillcoCoefMol(Vpmg *thee) {
04372
04373       if (thee->useDielXMap || thee->useDielYMap || thee->useDielZMap ||
04374         thee->useKappaMap)  {
04375
04376           fillcoCoefMap(thee);
04377
04378       } else {
04379
04380           fillcoCoefMolDiel(thee);
04381           fillcoCoefMolIon(thee);
04382
04383       }
04384
04385 }
04386
04387 VPRIVATE void fillcoCoefMolIon(Vpmg *thee) {
04388
04389       Vacc *acc;
04390       Valist *alist;
04391       Vpbe *pbe;
04392       Vatom *atom;
04393       double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr;
04394       double xlen, ylen, zlen, irad;
04395       double hx, hy, hzed, *apos, arad;
04396       int i, nx, ny, nz, iatom;
04397       Vsurf_Meth surfMeth;
04398
04399       VASSERT(thee != VNULL);
04400       surfMeth = thee->surfMeth;
04401
04402       /* Get PBE info */
04403       pbe = thee->pbe;
04404       acc = pbe->acc;
04405       alist = pbe->alist;
04406       irad = Vpbe_getMaxIonRadius(pbe);
04407       ionstr = Vpbe_getBulkIonicStrength(pbe);
04408
04409       /* Mesh info */
```

```
04410     nx = thee->pmgp->nx;
04411     ny = thee->pmgp->ny;
04412     nz = thee->pmgp->nz;
04413     hx = thee->pmgp->hx;
04414     hy = thee->pmgp->hy;
04415     hzed = thee->pmgp->hzed;
04416 
04417     /* Define the total domain size */
04418     xlen = thee->pmgp->xlen;
04419     ylen = thee->pmgp->ylen;
04420     zlen = thee->pmgp->zlen;
04421 
04422     /* Define the min/max dimensions */
04423     xmin = thee->pmgp->xcent - (xlen/2.0);
04424     ymin = thee->pmgp->ycent - (ylen/2.0);
04425     zmin = thee->pmgp->zcent - (zlen/2.0);
04426     xmax = thee->pmgp->xcent + (xlen/2.0);
04427     ymax = thee->pmgp->ycent + (ylen/2.0);
04428     zmax = thee->pmgp->zcent + (zlen/2.0);
04429 
04430     /* This is a floating point parameter related to the non-zero nature of the
04431      * bulk ionic strength.  If the ionic strength is greater than zero; this
04432      * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
04433      * Otherwise, this parameter is set to 0.0 */
04434     if (ionstr > VPMGSMALL) ionmask = 1.0;
04435     else ionmask = 0.0;
04436 
04437     /* Reset the kappa array, marking everything accessible */
04438     for (i=0; i<(nx*ny*nz); i++) thee->kappa[i] = ionmask;
04439 
04440     if (ionstr < VPMGSMALL) return;
04441 
04442     /* Loop through the atoms and set kappa = 0.0 (inaccessible) if a point
04443      * is inside the ion-inflated van der Waals radii */
04444     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
04445 
04446         atom = Valist_getAtom(alist, iatom);
04447         apos = Vatom_getPosition(atom);
04448         arad = Vatom_getRadius(atom);
04449 
04450         if (arad > VSMALL) {
04451 
04452             /* Make sure we're on the grid */
04453             if ((apos[0]<(xmin-irad-arad)) || (apos[0]>(xmax+irad+arad))  || \
04454                 (apos[1]<(ymin-irad-arad)) || (apos[1]>(ymax+irad+arad))  || \
04455                 (apos[2]<(zmin-irad-arad)) || (apos[2]>(zmax+irad+arad))) {
04456                 if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
04457      (thee->pmgp->bcfl != BCFL_MAP)) {
04458                     Vnm_print(2,
04459     "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):\
      n",
04460                       iatom, apos[0], apos[1], apos[2]);
04461                     Vnm_print(2, "Vpmg_fillco:  xmin = %g, xmax = %g\n",
04462                       xmin, xmax);
04463                     Vnm_print(2, "Vpmg_fillco:  ymin = %g, ymax = %g\n",
04464                       ymin, ymax);
04465                     Vnm_print(2, "Vpmg_fillco:  zmin = %g, zmax = %g\n",
```

```
04466                       zmin, zmax);
04467                }
04468                fflush(stderr);
04469
04470            } else { /* if we're on the mesh */
04471
04472                /* Mark ions */
04473                markSphere((irad+arad), apos,
04474                        nx, ny, nz,
04475                        hx, hy, hzed,
04476                        xmin, ymin, zmin,
04477                        thee->kappa, 0.0);
04478
04479            } /* endif (on the mesh) */
04480        }
04481    } /* endfor (over all atoms) */
04482
04483 }
04484
04485 VPRIVATE void fillcoCoefMolDiel(Vpmg *thee) {
04486
04487     /* Always call NoSmooth to fill the epsilon arrays */
04488     fillcoCoefMolDielNoSmooth(thee);
04489
04490     /* Call the smoothing algorithm as needed */
04491     if (thee->surfMeth == VSM_MOLSMOOTH) {
04492         fillcoCoefMolDielSmooth(thee);
04493     }
04494 }
04495
04496 VPRIVATE void fillcoCoefMolDielNoSmooth(Vpmg *thee) {
04497
04498     Vacc *acc;
04499     VaccSurf *asurf;
04500     Valist *alist;
04501     Vpbe *pbe;
04502     Vatom *atom;
04503     double xmin, xmax, ymin, ymax, zmin, zmax;
04504     double xlen, ylen, zlen, position[3];
04505     double srad, epsw, epsp, deps, area;
04506     double hx, hy, hzed, *apos, arad;
04507     int i, nx, ny, nz, ntot, iatom, ipt;
04508
04509     /* Get PBE info */
04510     pbe = thee->pbe;
04511     acc = pbe->acc;
04512     alist = pbe->alist;
04513     srad = Vpbe_getSolventRadius(pbe);
04514     epsw = Vpbe_getSolventDiel(pbe);
04515     epsp = Vpbe_getSoluteDiel(pbe);
04516
04517     /* Mesh info */
04518     nx = thee->pmgp->nx;
04519     ny = thee->pmgp->ny;
04520     nz = thee->pmgp->nz;
04521     hx = thee->pmgp->hx;
04522     hy = thee->pmgp->hy;
```

```
04523     hzed = thee->pmgp->hzed;
04524
04525     /* Define the total domain size */
04526     xlen = thee->pmgp->xlen;
04527     ylen = thee->pmgp->ylen;
04528     zlen = thee->pmgp->zlen;
04529
04530     /* Define the min/max dimensions */
04531     xmin = thee->pmgp->xcent - (xlen/2.0);
04532     ymin = thee->pmgp->ycent - (ylen/2.0);
04533     zmin = thee->pmgp->zcent - (zlen/2.0);
04534     xmax = thee->pmgp->xcent + (xlen/2.0);
04535     ymax = thee->pmgp->ycent + (ylen/2.0);
04536     zmax = thee->pmgp->zcent + (zlen/2.0);
04537
04538     /* Reset the arrays */
04539  ntot = nx*ny*nz;
04540     for (i=0; i<ntot; i++) {
04541         thee->epsx[i] = epsw;
04542         thee->epsy[i] = epsw;
04543         thee->epsz[i] = epsw;
04544     }
04545
04546     /* Loop through the atoms and set a{123}cf = 0.0 (inaccessible)
04547      * if a point is inside the solvent-inflated van der Waals radii */
04548 #pragma omp parallel for default(shared) private(iatom,atom,apos,arad)
04549     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
04550
04551         atom = Valist_getAtom(alist, iatom);
04552         apos = Vatom_getPosition(atom);
04553         arad = Vatom_getRadius(atom);
04554
04555         /* Make sure we're on the grid */
04556         if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
04557             (apos[1]<=ymin) || (apos[1]>=ymax)  || \
04558             (apos[2]<=zmin) || (apos[2]>=zmax)) {
04559    if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
04560     (thee->pmgp->bcfl != BCFL_MAP)) {
04561                 Vnm_print(2, "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f,\
04562  %4.3f) is off the mesh (ignoring):\n",
04563                   iatom, apos[0], apos[1], apos[2]);
04564                 Vnm_print(2, "Vpmg_fillco:  xmin = %g, xmax = %g\n",
04565                   xmin, xmax);
04566                 Vnm_print(2, "Vpmg_fillco:  ymin = %g, ymax = %g\n",
04567                   ymin, ymax);
04568                 Vnm_print(2, "Vpmg_fillco:  zmin = %g, zmax = %g\n",
04569                   zmin, zmax);
04570             }
04571             fflush(stderr);
04572
04573         } else { /* if we're on the mesh */
04574
04575             if (arad > VSMALL) {
04576                 /* Mark x-shifted dielectric */
04577                 markSphere((arad+srad), apos,
04578                         nx, ny, nz,
04579                         hx, hy, hzed,
```

```
04580                         (xmin+0.5*hx), ymin, zmin,
04581                         thee->epsx, epsp);
04582
04583                 /* Mark y-shifted dielectric */
04584                 markSphere((arad+srad), apos,
04585                         nx, ny, nz,
04586                         hx, hy, hzed,
04587                         xmin, (ymin+0.5*hy), zmin,
04588                         thee->epsy, epsp);
04589
04590                 /* Mark z-shifted dielectric */
04591                 markSphere((arad+srad), apos,
04592                         nx, ny, nz,
04593                         hx, hy, hzed,
04594                         xmin, ymin, (zmin+0.5*hzed),
04595                         thee->epsz, epsp);
04596             }
04597
04598         } /* endif (on the mesh) */
04599     } /* endfor (over all atoms) */
04600
04601  area = Vacc_SASA(acc, srad);
04602
04603     /* We only need to do the next step for non-zero solvent radii */
04604     if (srad > VSMALL) {
04605
04606         /* Now loop over the solvent accessible surface points */
04607
04608 #pragma omp parallel for default(shared) private(iatom,atom,area,asurf,ipt,positi
      on)
04609         for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
04610             atom = Valist_getAtom(alist, iatom);
04611    area = Vacc_atomSASA(acc, srad, atom);
04612    if (area > 0.0 ) {
04613     asurf = Vacc_atomSASPoints(acc, srad, atom);
04614
04615     /* Use each point on the SAS to reset the solvent accessibility */
04616     /* TODO:  Make sure we're not still wasting time here. */
04617     for (ipt=0; ipt<(asurf->npts); ipt++) {
04618
04619      position[0] = asurf->xpts[ipt];
04620      position[1] = asurf->ypts[ipt];
04621      position[2] = asurf->zpts[ipt];
04622
04623      /* Mark x-shifted dielectric */
04624      markSphere(srad, position,
04625           nx, ny, nz,
04626           hx, hy, hzed,
04627           (xmin+0.5*hx), ymin, zmin,
04628           thee->epsx, epsw);
04629
04630      /* Mark y-shifted dielectric */
04631      markSphere(srad, position,
04632           nx, ny, nz,
04633           hx, hy, hzed,
04634           xmin, (ymin+0.5*hy), zmin,
04635           thee->epsy, epsw);
```

```
04636
04637      /* Mark z-shifted dielectric */
04638      markSphere(srad, position,
04639           nx, ny, nz,
04640           hx, hy, hzed,
04641           xmin, ymin, (zmin+0.5*hzed),
04642           thee->epsz, epsw);
04643
04644     }
04645    }
04646         }
04647     }
04648 }
04649
04650 VPRIVATE void fillcoCoefMolDielSmooth(Vpmg *thee) {
04651
04652   /* This function smoothes using a 9 point method based on
04653      Bruccoleri, et al. J Comput Chem 18 268-276 (1997).  The nine points
04654      used are the shifted grid point and the 8 points that are 1/sqrt(2)
04655      grid spacings away.  The harmonic mean of the 9 points is then used to
04656      find the overall dielectric value for the point in question. The use of
04657      this function assumes that the non-smoothed values were placed in the
04658      dielectric arrays by the fillcoCoefMolDielNoSmooth function.*/
04659
04660     Vpbe *pbe;
04661     double frac, epsw;
04662     int i, j, k, nx, ny, nz, numpts;
04663
04664     /* Mesh info */
04665     nx = thee->pmgp->nx;
04666     ny = thee->pmgp->ny;
04667     nz = thee->pmgp->nz;
04668
04669     pbe = thee->pbe;
04670     epsw = Vpbe_getSolventDiel(pbe);
04671
04672     /* Copy the existing diel arrays to work arrays */
04673     for (i=0; i<(nx*ny*nz); i++) {
04674         thee->a1cf[i] = thee->epsx[i];
04675         thee->a2cf[i] = thee->epsy[i];
04676         thee->a3cf[i] = thee->epsz[i];
04677         thee->epsx[i] = epsw;
04678         thee->epsy[i] = epsw;
04679         thee->epsz[i] = epsw;
04680     }
04681
04682     /* Smooth the dielectric values */
04683     for (i=0; i<nx; i++) {
04684         for (j=0; j<ny; j++) {
04685             for (k=0; k<nz; k++) {
04686
04687                 /* Get the 8 points that are 1/sqrt(2) grid spacings away */
04688
04689                 /* Points for the X-shifted array */
04690                 frac = 1.0/thee->a1cf[IJK(i,j,k)];
04691                 frac += 1.0/thee->a2cf[IJK(i,j,k)];
04692                 frac += 1.0/thee->a3cf[IJK(i,j,k)];
```

```
04693                 numpts = 3;
04694
04695                 if (j > 0) {
04696                     frac += 1.0/thee->a2cf[IJK(i,j-1,k)];
04697                     numpts += 1;
04698                 }
04699                 if (k > 0) {
04700                     frac += 1.0/thee->a3cf[IJK(i,j,k-1)];
04701                     numpts += 1;
04702                 }
04703                 if (i < (nx-1)){
04704                     frac += 1.0/thee->a2cf[IJK(i+1,j,k)];
04705                     frac += 1.0/thee->a3cf[IJK(i+1,j,k)];
04706                     numpts += 2;
04707                     if (j > 0) {
04708                         frac += 1.0/thee->a2cf[IJK(i+1,j-1,k)];
04709                         numpts += 1;
04710                     }
04711                     if (k > 0) {
04712                         frac += 1.0/thee->a3cf[IJK(i+1,j,k-1)];
04713                         numpts += 1;
04714                     }
04715                 }
04716                 thee->epsx[IJK(i,j,k)] = numpts/frac;
04717
04718                 /* Points for the Y-shifted array */
04719                 frac = 1.0/thee->a2cf[IJK(i,j,k)];
04720                 frac += 1.0/thee->a1cf[IJK(i,j,k)];
04721                 frac += 1.0/thee->a3cf[IJK(i,j,k)];
04722                 numpts = 3;
04723
04724                 if (i > 0) {
04725                     frac += 1.0/thee->a1cf[IJK(i-1,j,k)];
04726                     numpts += 1;
04727                 }
04728                 if (k > 0) {
04729                     frac += 1.0/thee->a3cf[IJK(i,j,k-1)];
04730                     numpts += 1;
04731                 }
04732                 if (j < (ny-1)){
04733                     frac += 1.0/thee->a1cf[IJK(i,j+1,k)];
04734                     frac += 1.0/thee->a3cf[IJK(i,j+1,k)];
04735                     numpts += 2;
04736                     if (i > 0) {
04737                         frac += 1.0/thee->a1cf[IJK(i-1,j+1,k)];
04738                         numpts += 1;
04739                     }
04740                     if (k > 0) {
04741                         frac += 1.0/thee->a3cf[IJK(i,j+1,k-1)];
04742                         numpts += 1;
04743                     }
04744                 }
04745                 thee->epsy[IJK(i,j,k)] = numpts/frac;
04746
04747                 /* Points for the Z-shifted array */
04748                 frac = 1.0/thee->a3cf[IJK(i,j,k)];
04749                 frac += 1.0/thee->a1cf[IJK(i,j,k)];
```

```
04750                 frac += 1.0/thee->a2cf[IJK(i,j,k)];
04751                 numpts = 3;
04752 
04753                 if (i > 0) {
04754                     frac += 1.0/thee->a1cf[IJK(i-1,j,k)];
04755                     numpts += 1;
04756                 }
04757                 if (j > 0) {
04758                     frac += 1.0/thee->a2cf[IJK(i,j-1,k)];
04759                     numpts += 1;
04760                 }
04761                 if (k < (nz-1)){
04762                     frac += 1.0/thee->a1cf[IJK(i,j,k+1)];
04763                     frac += 1.0/thee->a2cf[IJK(i,j,k+1)];
04764                     numpts += 2;
04765                     if (i > 0) {
04766                         frac += 1.0/thee->a1cf[IJK(i-1,j,k+1)];
04767                         numpts += 1;
04768                     }
04769                     if (j > 0) {
04770                         frac += 1.0/thee->a2cf[IJK(i,j-1,k+1)];
04771                         numpts += 1;
04772                     }
04773                 }
04774                 thee->epsz[IJK(i,j,k)] = numpts/frac;
04775             }
04776         }
04777     }
04778 }
04779 
04780 
04781 VPRIVATE void fillcoCoefSpline(Vpmg *thee) {
04782 
04783     Valist *alist;
04784     Vpbe *pbe;
04785     Vatom *atom;
04786     double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr, dist2;
04787     double xlen, ylen, zlen, position[3], itot, stot, ictot, ictot2, sctot;
04788     double irad, dx, dy, dz, epsw, epsp, w2i;
04789     double hx, hy, hzed, *apos, arad, sctot2;
04790     double dx2, dy2, dz2, stot2, itot2, rtot, rtot2, splineWin, w3i;
04791     double dist, value, sm, sm2;
04792     int i, j, k, nx, ny, nz, iatom;
04793     int imin, imax, jmin, jmax, kmin, kmax;
04794 
04795     VASSERT(thee != VNULL);
04796     splineWin = thee->splineWin;
04797     w2i = 1.0/(splineWin*splineWin);
04798     w3i = 1.0/(splineWin*splineWin*splineWin);
04799 
04800     /* Get PBE info */
04801     pbe = thee->pbe;
04802     alist = pbe->alist;
04803     irad = Vpbe_getMaxIonRadius(pbe);
04804     ionstr = Vpbe_getBulkIonicStrength(pbe);
04805     epsw = Vpbe_getSolventDiel(pbe);
04806     epsp = Vpbe_getSoluteDiel(pbe);
```

```
04807
04808     /* Mesh info */
04809     nx = thee->pmgp->nx;
04810     ny = thee->pmgp->ny;
04811     nz = thee->pmgp->nz;
04812     hx = thee->pmgp->hx;
04813     hy = thee->pmgp->hy;
04814     hzed = thee->pmgp->hzed;
04815
04816     /* Define the total domain size */
04817     xlen = thee->pmgp->xlen;
04818     ylen = thee->pmgp->ylen;
04819     zlen = thee->pmgp->zlen;
04820
04821     /* Define the min/max dimensions */
04822     xmin = thee->pmgp->xcent - (xlen/2.0);
04823     ymin = thee->pmgp->ycent - (ylen/2.0);
04824     zmin = thee->pmgp->zcent - (zlen/2.0);
04825     xmax = thee->pmgp->xcent + (xlen/2.0);
04826     ymax = thee->pmgp->ycent + (ylen/2.0);
04827     zmax = thee->pmgp->zcent + (zlen/2.0);
04828
04829     /* This is a floating point parameter related to the non-zero nature of the
04830      * bulk ionic strength.  If the ionic strength is greater than zero; this
04831      * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
04832      * Otherwise, this parameter is set to 0.0 */
04833     if (ionstr > VPMGSMALL) ionmask = 1.0;
04834     else ionmask = 0.0;
04835
04836     /* Reset the kappa, epsx, epsy, and epsz arrays */
04837     for (i=0; i<(nx*ny*nz); i++) {
04838         thee->kappa[i] = 1.0;
04839         thee->epsx[i] = 1.0;
04840         thee->epsy[i] = 1.0;
04841         thee->epsz[i] = 1.0;
04842     }
04843
04844     /* Loop through the atoms and do assign the dielectric */
04845     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
04846
04847         atom = Valist_getAtom(alist, iatom);
04848         apos = Vatom_getPosition(atom);
04849         arad = Vatom_getRadius(atom);
04850
04851         /* Make sure we're on the grid */
04852         if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
04853             (apos[1]<=ymin) || (apos[1]>=ymax)  || \
04854             (apos[2]<=zmin) || (apos[2]>=zmax)) {
04855    if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
04856     (thee->pmgp->bcfl != BCFL_MAP)) {
04857                 Vnm_print(2, "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f,\
04858  %4.3f) is off the mesh (ignoring):\n",
04859                   iatom, apos[0], apos[1], apos[2]);
04860                 Vnm_print(2, "Vpmg_fillco:    xmin = %g, xmax = %g\n",
04861                   xmin, xmax);
04862                 Vnm_print(2, "Vpmg_fillco:    ymin = %g, ymax = %g\n",
04863                   ymin, ymax);
```

```
04864                 Vnm_print(2, "Vpmg_fillco:    zmin = %g, zmax = %g\n",
04865                   zmin, zmax);
04866             }
04867             fflush(stderr);
04868 
04869         } else if (arad > VPMGSMALL ) { /* if we're on the mesh */
04870 
04871             /* Convert the atom position to grid reference frame */
04872             position[0] = apos[0] - xmin;
04873             position[1] = apos[1] - ymin;
04874             position[2] = apos[2] - zmin;
04875 
04876             /* MARK ION ACCESSIBILITY AND DIELECTRIC VALUES FOR LATER
04877              * ASSIGNMENT (Steps #1-3) */
04878             itot = irad + arad + splineWin;
04879             itot2 = VSQR(itot);
04880             ictot = VMAX2(0, (irad + arad - splineWin));
04881             ictot2 = VSQR(ictot);
04882             stot = arad + splineWin;
04883             stot2 = VSQR(stot);
04884             sctot = VMAX2(0, (arad - splineWin));
04885             sctot2 = VSQR(sctot);
04886 
04887            /* We'll search over grid points which are in the greater of
04888             * these two radii */
04889             rtot = VMAX2(itot, stot);
04890             rtot2 = VMAX2(itot2, stot2);
04891             dx = rtot + 0.5*hx;
04892             dy = rtot + 0.5*hy;
04893             dz = rtot + 0.5*hzed;
04894             imin = VMAX2(0,(int)floor((position[0] - dx)/hx));
04895             imax = VMIN2(nx-1,(int)ceil((position[0] + dx)/hx));
04896             jmin = VMAX2(0,(int)floor((position[1] - dy)/hy));
04897             jmax = VMIN2(ny-1,(int)ceil((position[1] + dy)/hy));
04898             kmin = VMAX2(0,(int)floor((position[2] - dz)/hzed));
04899             kmax = VMIN2(nz-1,(int)ceil((position[2] + dz)/hzed));
04900             for (i=imin; i<=imax; i++) {
04901                 dx2 = VSQR(position[0] - hx*i);
04902                 for (j=jmin; j<=jmax; j++) {
04903                     dy2 = VSQR(position[1] - hy*j);
04904                     for (k=kmin; k<=kmax; k++) {
04905                         dz2 = VSQR(position[2] - k*hzed);
04906 
04907                         /* ASSIGN CCF */
04908                         if (thee->kappa[IJK(i,j,k)] > VPMGSMALL) {
04909                             dist2 = dz2 + dy2 + dx2;
04910                             if (dist2 >= itot2) {
04911                                 ;
04912                             }
04913                             if (dist2 <= ictot2) {
04914                                 thee->kappa[IJK(i,j,k)] = 0.0;
04915                             }
04916                             if ((dist2 < itot2) && (dist2 > ictot2)) {
04917                                 dist = VSQRT(dist2);
04918                                 sm = dist - (arad + irad) + splineWin;
04919                                 sm2 = VSQR(sm);
04920                                 value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
```

```
04921                                   thee->kappa[IJK(i,j,k)] *= value;
04922                               }
04923                           }
04924
04925                           /* ASSIGN A1CF */
04926                           if (thee->epsx[IJK(i,j,k)] > VPMGSMALL) {
04927                               dist2 = dz2+dy2+VSQR(position[0]-(i+0.5)*hx);
04928                               if (dist2 >= stot2) {
04929                                   thee->epsx[IJK(i,j,k)] *= 1.0;
04930                               }
04931                               if (dist2 <= sctot2) {
04932                                   thee->epsx[IJK(i,j,k)] = 0.0;
04933                               }
04934                               if ((dist2 > sctot2) && (dist2 < stot2)) {
04935                                   dist = VSQRT(dist2);
04936                                   sm = dist - arad + splineWin;
04937                                   sm2 = VSQR(sm);
04938                                   value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
04939                                   thee->epsx[IJK(i,j,k)] *= value;
04940                               }
04941                           }
04942
04943                           /* ASSIGN A2CF */
04944                           if (thee->epsy[IJK(i,j,k)] > VPMGSMALL) {
04945                               dist2 = dz2+dx2+VSQR(position[1]-(j+0.5)*hy);
04946                               if (dist2 >= stot2) {
04947                                   thee->epsy[IJK(i,j,k)] *= 1.0;
04948                               }
04949                               if (dist2 <= sctot2) {
04950                                   thee->epsy[IJK(i,j,k)] = 0.0;
04951                               }
04952                               if ((dist2 > sctot2) && (dist2 < stot2)) {
04953                                   dist = VSQRT(dist2);
04954                                   sm = dist - arad + splineWin;
04955                                   sm2 = VSQR(sm);
04956                                   value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
04957                                   thee->epsy[IJK(i,j,k)] *= value;
04958                               }
04959                           }
04960
04961                           /* ASSIGN A3CF */
04962                           if (thee->epsz[IJK(i,j,k)] > VPMGSMALL) {
04963                               dist2 = dy2+dx2+VSQR(position[2]-(k+0.5)*hzed);
04964                               if (dist2 >= stot2) {
04965                                   thee->epsz[IJK(i,j,k)] *= 1.0;
04966                               }
04967                               if (dist2 <= sctot2) {
04968                                   thee->epsz[IJK(i,j,k)] = 0.0;
04969                               }
04970                               if ((dist2 > sctot2) && (dist2 < stot2)) {
04971                                   dist = VSQRT(dist2);
04972                                   sm = dist - arad + splineWin;
04973                                   sm2 = VSQR(sm);
04974                                   value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
04975                                   thee->epsz[IJK(i,j,k)] *= value;
04976                               }
04977                           }
```

```
04978
04979
04980                     } /* k loop */
04981                 } /* j loop */
04982             } /* i loop */
04983         } /* endif (on the mesh) */
04984     } /* endfor (over all atoms) */
04985
04986     Vnm_print(0, "Vpmg_fillco:  filling coefficient arrays\n");
04987     /* Interpret markings and fill the coefficient arrays */
04988     for (k=0; k<nz; k++) {
04989         for (j=0; j<ny; j++) {
04990             for (i=0; i<nx; i++) {
04991
04992                 thee->kappa[IJK(i,j,k)] = ionmask*thee->kappa[IJK(i,j,k)];
04993                 thee->epsx[IJK(i,j,k)] = (epsw-epsp)*thee->epsx[IJK(i,j,k)]
04994                   + epsp;
04995                 thee->epsy[IJK(i,j,k)] = (epsw-epsp)*thee->epsy[IJK(i,j,k)]
04996                   + epsp;
04997                 thee->epsz[IJK(i,j,k)] = (epsw-epsp)*thee->epsz[IJK(i,j,k)]
04998                   + epsp;
04999
05000             } /* i loop */
05001         } /* j loop */
05002     } /* k loop */
05003
05004 }
05005
05006 VPRIVATE void fillcoCoef(Vpmg *thee) {
05007
05008     VASSERT(thee != VNULL);
05009
05010     if (thee->useDielXMap || thee->useDielYMap ||
05011   thee->useDielZMap || thee->useKappaMap) {
05012         fillcoCoefMap(thee);
05013         return;
05014     }
05015
05016     switch(thee->surfMeth) {
05017         case VSM_MOL:
05018             Vnm_print(0, "fillcoCoef:  Calling fillcoCoefMol...\n");
05019             fillcoCoefMol(thee);
05020             break;
05021         case VSM_MOLSMOOTH:
05022             Vnm_print(0, "fillcoCoef:  Calling fillcoCoefMol...\n");
05023             fillcoCoefMol(thee);
05024             break;
05025         case VSM_SPLINE:
05026             Vnm_print(0, "fillcoCoef:  Calling fillcoCoefSpline...\n");
05027             fillcoCoefSpline(thee);
05028             break;
05029   case VSM_SPLINE3:
05030             Vnm_print(0, "fillcoCoef:  Calling fillcoCoefSpline3...\n");
05031             fillcoCoefSpline3(thee);
05032             break;
05033         case VSM_SPLINE4:
05034             Vnm_print(0, "fillcoCoef:  Calling fillcoCoefSpline4...\n");
```

```
05035             fillcoCoefSpline4(thee);
05036             break;
05037         default:
05038             Vnm_print(2, "fillcoCoef:  Invalid surfMeth (%d)!\n",
05039               thee->surfMeth);
05040             VASSERT(0);
05041             break;
05042     }
05043 }
05044
05045
05046 VPRIVATE Vrc_Codes fillcoCharge(Vpmg *thee) {
05047
05048  Vrc_Codes rc;
05049
05050     VASSERT(thee != VNULL);
05051
05052     if (thee->useChargeMap) {
05053         return fillcoChargeMap(thee);
05054     }
05055
05056     switch(thee->chargeMeth) {
05057         case VCM_TRIL:
05058             Vnm_print(0, "fillcoCharge:  Calling fillcoChargeSpline1...\n");
05059             fillcoChargeSpline1(thee);
05060             break;
05061         case VCM_BSPL2:
05062             Vnm_print(0, "fillcoCharge:  Calling fillcoChargeSpline2...\n");
05063             fillcoChargeSpline2(thee);
05064             break;
05065         case VCM_BSPL4:
05066             switch (thee->chargeSrc) {
05067                 case VCM_CHARGE:
05068                     Vnm_print(0, "fillcoCharge: Calling fillcoPermanentMultipole.
      ..\n");
05069                     fillcoPermanentMultipole(thee);
05070                     break;
05071 #if defined(WITH_TINKER)
05072                 case VCM_PERMANENT:
05073                     Vnm_print(0, "fillcoCharge: Calling fillcoPermanentMultipole.
      ..\n");
05074                     fillcoPermanentMultipole(thee);
05075                     break;
05076                 case VCM_INDUCED:
05077                     Vnm_print(0, "fillcoCharge: Calling fillcoInducedDipole...\n"
      );
05078                     fillcoInducedDipole(thee);
05079                     break;
05080                 case VCM_NLINDUCED:
05081                      Vnm_print(0, "fillcoCharge: Calling fillcoNLInducedDipole...
      \n");
05082                      fillcoNLInducedDipole(thee);
05083                      break;
05084 #endif /* if defined(WITH_TINKER) */
05085                 default:
05086                     Vnm_print(2, "fillcoCharge:  Invalid chargeSource (%d)!\n",
05087                       thee->chargeSrc);
```

```
05088                     return VRC_FAILURE;
05089                     break;
05090                 }
05091                 break;
05092             default:
05093                 Vnm_print(2, "fillcoCharge:  Invalid chargeMeth (%d)!\n",
05094                   thee->chargeMeth);
05095                 return VRC_FAILURE;
05096                 break;
05097         }
05098 
05099  return VRC_SUCCESS;
05100 }
05101 
05102 VPRIVATE Vrc_Codes fillcoChargeMap(Vpmg *thee) {
05103 
05104     Vpbe *pbe;
05105     double position[3], charge, zmagic, hx, hy, hzed;
05106     int i, j, k, nx, ny, nz, rc;
05107 
05108 
05109     VASSERT(thee != VNULL);
05110 
05111     /* Get PBE info */
05112     pbe = thee->pbe;
05113     zmagic = Vpbe_getZmagic(pbe);
05114 
05115     /* Mesh info */
05116     nx = thee->pmgp->nx;
05117     ny = thee->pmgp->ny;
05118     nz = thee->pmgp->nz;
05119     hx = thee->pmgp->hx;
05120     hy = thee->pmgp->hy;
05121     hzed = thee->pmgp->hzed;
05122 
05123     /* Reset the charge array */
05124     for (i=0; i<(nx*ny*nz); i++) thee->charge[i] = 0.0;
05125 
05126     /* Fill in the source term (atomic charges) */
05127     Vnm_print(0, "Vpmg_fillco:  filling in source term.\n");
05128     for (k=0; k<nz; k++) {
05129         for (j=0; j<ny; j++) {
05130             for (i=0; i<nx; i++) {
05131                 position[0] = thee->xf[i];
05132                 position[1] = thee->yf[j];
05133                 position[2] = thee->zf[k];
05134     rc = Vgrid_value(thee->chargeMap, position, &charge);
05135     if (!rc) {
05136      Vnm_print(2, "fillcoChargeMap:  Error -- fell off of charge map at (%g, %g,
      %g)!\n",
05137         position[0], position[1], position[2]);
05138      return VRC_FAILURE;
05139     }
05140     /* Scale the charge to internal units */
05141                 charge = charge*zmagic;
05142                 thee->charge[IJK(i,j,k)] = charge;
05143             }
```

```
05144         }
05145     }
05146
05147  return VRC_SUCCESS;
05148 }
05149
05150 VPRIVATE void fillcoChargeSpline1(Vpmg *thee) {
05151
05152     Valist *alist;
05153     Vpbe *pbe;
05154     Vatom *atom;
05155     double xmin, xmax, ymin, ymax, zmin, zmax;
05156     double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;
05157     double charge, dx, dy, dz, zmagic, hx, hy, hzed, *apos;
05158     int i, nx, ny, nz, iatom, ihi, ilo, jhi, jlo, khi, klo;
05159
05160
05161     VASSERT(thee != VNULL);
05162
05163     /* Get PBE info */
05164     pbe = thee->pbe;
05165     alist = pbe->alist;
05166     zmagic = Vpbe_getZmagic(pbe);
05167
05168     /* Mesh info */
05169     nx = thee->pmgp->nx;
05170     ny = thee->pmgp->ny;
05171     nz = thee->pmgp->nz;
05172     hx = thee->pmgp->hx;
05173     hy = thee->pmgp->hy;
05174     hzed = thee->pmgp->hzed;
05175
05176     /* Define the total domain size */
05177     xlen = thee->pmgp->xlen;
05178     ylen = thee->pmgp->ylen;
05179     zlen = thee->pmgp->zlen;
05180
05181     /* Define the min/max dimensions */
05182     xmin = thee->pmgp->xcent - (xlen/2.0);
05183     ymin = thee->pmgp->ycent - (ylen/2.0);
05184     zmin = thee->pmgp->zcent - (zlen/2.0);
05185     xmax = thee->pmgp->xcent + (xlen/2.0);
05186     ymax = thee->pmgp->ycent + (ylen/2.0);
05187     zmax = thee->pmgp->zcent + (zlen/2.0);
05188
05189     /* Reset the charge array */
05190     for (i=0; i<(nx*ny*nz); i++) thee->charge[i] = 0.0;
05191
05192     /* Fill in the source term (atomic charges) */
05193     Vnm_print(0, "Vpmg_fillco:  filling in source term.\n");
05194     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
05195
05196         atom = Valist_getAtom(alist, iatom);
05197         apos = Vatom_getPosition(atom);
05198         charge = Vatom_getCharge(atom);
05199
05200         /* Make sure we're on the grid */
```

```
05201         if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
05202             (apos[1]<=ymin) || (apos[1]>=ymax)  || \
05203             (apos[2]<=zmin) || (apos[2]>=zmax)) {
05204    if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
05205     (thee->pmgp->bcfl != BCFL_MAP)) {
05206                 Vnm_print(2, "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f, \
05207 %4.3f) is off the mesh (ignoring):\n",
05208                   iatom, apos[0], apos[1], apos[2]);
05209                 Vnm_print(2, "Vpmg_fillco:    xmin = %g, xmax = %g\n",
05210                   xmin, xmax);
05211                 Vnm_print(2, "Vpmg_fillco:    ymin = %g, ymax = %g\n",
05212                   ymin, ymax);
05213                 Vnm_print(2, "Vpmg_fillco:    zmin = %g, zmax = %g\n",
05214                   zmin, zmax);
05215             }
05216             fflush(stderr);
05217         } else {
05218 
05219             /* Convert the atom position to grid reference frame */
05220             position[0] = apos[0] - xmin;
05221             position[1] = apos[1] - ymin;
05222             position[2] = apos[2] - zmin;
05223 
05224             /* Scale the charge to be a delta function */
05225             charge = charge*zmagic/(hx*hy*hzed);
05226 
05227             /* Figure out which vertices we're next to */
05228             ifloat = position[0]/hx;
05229             jfloat = position[1]/hy;
05230             kfloat = position[2]/hzed;
05231 
05232             ihi = (int)ceil(ifloat);
05233             ilo = (int)floor(ifloat);
05234             jhi = (int)ceil(jfloat);
05235             jlo = (int)floor(jfloat);
05236             khi = (int)ceil(kfloat);
05237             klo = (int)floor(kfloat);
05238 
05239             /* Now assign fractions of the charge to the nearby verts */
05240             dx = ifloat - (double)(ilo);
05241             dy = jfloat - (double)(jlo);
05242             dz = kfloat - (double)(klo);
05243             thee->charge[IJK(ihi,jhi,khi)] += (dx*dy*dz*charge);
05244             thee->charge[IJK(ihi,jlo,khi)] += (dx*(1.0-dy)*dz*charge);
05245             thee->charge[IJK(ihi,jhi,klo)] += (dx*dy*(1.0-dz)*charge);
05246             thee->charge[IJK(ihi,jlo,klo)] += (dx*(1.0-dy)*(1.0-dz)*charge);
05247             thee->charge[IJK(ilo,jhi,khi)] += ((1.0-dx)*dy*dz *charge);
05248             thee->charge[IJK(ilo,jlo,khi)] += ((1.0-dx)*(1.0-dy)*dz *charge);
05249             thee->charge[IJK(ilo,jhi,klo)] += ((1.0-dx)*dy*(1.0-dz)*charge);
05250             thee->charge[IJK(ilo,jlo,klo)] += ((1.0-dx)*(1.0-dy)*(1.0-dz)*charge)
    ;
05251         } /* endif (on the mesh) */
05252     } /* endfor (each atom) */
05253 }
05254 
05255 VPRIVATE double bspline2(double x) {
05256 
```

```
05257     double m2m, m2, m3;
05258
05259     if ((x >= 0.0) && (x <= 2.0)) m2m = 1.0 - VABS(x - 1.0);
05260     else m2m = 0.0;
05261     if ((x >= 1.0) && (x <= 3.0)) m2 = 1.0 - VABS(x - 2.0);
05262     else m2 = 0.0;
05263
05264     if ((x >= 0.0) && (x <= 3.0)) m3 = 0.5*x*m2m + 0.5*(3.0-x)*m2;
05265     else m3 = 0.0;
05266
05267     return m3;
05268
05269 }
05270
05271 VPRIVATE double dbspline2(double x) {
05272
05273     double m2m, m2, dm3;
05274
05275     if ((x >= 0.0) && (x <= 2.0)) m2m = 1.0 - VABS(x - 1.0);
05276     else m2m = 0.0;
05277     if ((x >= 1.0) && (x <= 3.0)) m2 = 1.0 - VABS(x - 2.0);
05278     else m2 = 0.0;
05279
05280     dm3 = m2m - m2;
05281
05282     return dm3;
05283
05284 }
05285
05286
05287 VPRIVATE void fillcoChargeSpline2(Vpmg *thee) {
05288
05289     Valist *alist;
05290     Vpbe *pbe;
05291     Vatom *atom;
05292     double xmin, xmax, ymin, ymax, zmin, zmax, zmagic;
05293     double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;
05294     double charge, hx, hy, hzed, *apos, mx, my, mz;
05295     int i, ii, jj, kk, nx, ny, nz, iatom;
05296     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
05297
05298
05299     VASSERT(thee != VNULL);
05300
05301     /* Get PBE info */
05302     pbe = thee->pbe;
05303     alist = pbe->alist;
05304     zmagic = Vpbe_getZmagic(pbe);
05305
05306     /* Mesh info */
05307     nx = thee->pmgp->nx;
05308     ny = thee->pmgp->ny;
05309     nz = thee->pmgp->nz;
05310     hx = thee->pmgp->hx;
05311     hy = thee->pmgp->hy;
05312     hzed = thee->pmgp->hzed;
05313
```

```
05314     /* Define the total domain size */
05315     xlen = thee->pmgp->xlen;
05316     ylen = thee->pmgp->ylen;
05317     zlen = thee->pmgp->zlen;
05318
05319     /* Define the min/max dimensions */
05320     xmin = thee->pmgp->xcent - (xlen/2.0);
05321     ymin = thee->pmgp->ycent - (ylen/2.0);
05322     zmin = thee->pmgp->zcent - (zlen/2.0);
05323     xmax = thee->pmgp->xcent + (xlen/2.0);
05324     ymax = thee->pmgp->ycent + (ylen/2.0);
05325     zmax = thee->pmgp->zcent + (zlen/2.0);
05326
05327     /* Reset the charge array */
05328     for (i=0; i<(nx*ny*nz); i++) thee->charge[i] = 0.0;
05329
05330     /* Fill in the source term (atomic charges) */
05331     Vnm_print(0, "Vpmg_fillco:  filling in source term.\n");
05332     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
05333
05334         atom = Valist_getAtom(alist, iatom);
05335         apos = Vatom_getPosition(atom);
05336         charge = Vatom_getCharge(atom);
05337
05338         /* Make sure we're on the grid */
05339         if ((apos[0]<=(xmin-hx)) || (apos[0]>=(xmax+hx))  || \
05340             (apos[1]<=(ymin-hy)) || (apos[1]>=(ymax+hy))  || \
05341             (apos[2]<=(zmin-hzed)) || (apos[2]>=(zmax+hzed))) {
05342    if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
05343     (thee->pmgp->bcfl != BCFL_MAP)) {
05344                 Vnm_print(2, "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f, \
05345 %4.3f) is off the mesh (for cubic splines!!) (ignoring this atom):\n",
05346                   iatom, apos[0], apos[1], apos[2]);
05347                 Vnm_print(2, "Vpmg_fillco:    xmin = %g, xmax = %g\n",
05348                   xmin, xmax);
05349                 Vnm_print(2, "Vpmg_fillco:    ymin = %g, ymax = %g\n",
05350                   ymin, ymax);
05351                 Vnm_print(2, "Vpmg_fillco:    zmin = %g, zmax = %g\n",
05352                   zmin, zmax);
05353             }
05354             fflush(stderr);
05355         } else {
05356
05357             /* Convert the atom position to grid reference frame */
05358             position[0] = apos[0] - xmin;
05359             position[1] = apos[1] - ymin;
05360             position[2] = apos[2] - zmin;
05361
05362             /* Scale the charge to be a delta function */
05363             charge = charge*zmagic/(hx*hy*hzed);
05364
05365             /* Figure out which vertices we're next to */
05366             ifloat = position[0]/hx;
05367             jfloat = position[1]/hy;
05368             kfloat = position[2]/hzed;
05369
05370             ip1   = (int)ceil(ifloat);
```

```
05371             ip2   = ip1 + 1;
05372             im1   = (int)floor(ifloat);
05373             im2   = im1 - 1;
05374             jp1   = (int)ceil(jfloat);
05375             jp2   = jp1 + 1;
05376             jm1   = (int)floor(jfloat);
05377             jm2   = jm1 - 1;
05378             kp1   = (int)ceil(kfloat);
05379             kp2   = kp1 + 1;
05380             km1   = (int)floor(kfloat);
05381             km2   = km1 - 1;
05382
05383             /* This step shouldn't be necessary, but it saves nasty debugging
05384              * later on if something goes wrong */
05385             ip2 = VMIN2(ip2,nx-1);
05386             ip1 = VMIN2(ip1,nx-1);
05387             im1 = VMAX2(im1,0);
05388             im2 = VMAX2(im2,0);
05389             jp2 = VMIN2(jp2,ny-1);
05390             jp1 = VMIN2(jp1,ny-1);
05391             jm1 = VMAX2(jm1,0);
05392             jm2 = VMAX2(jm2,0);
05393             kp2 = VMIN2(kp2,nz-1);
05394             kp1 = VMIN2(kp1,nz-1);
05395             km1 = VMAX2(km1,0);
05396             km2 = VMAX2(km2,0);
05397
05398             /* Now assign fractions of the charge to the nearby verts */
05399             for (ii=im2; ii<=ip2; ii++) {
05400                 mx = bspline2(VFCHI(ii,ifloat));
05401                 for (jj=jm2; jj<=jp2; jj++) {
05402                     my = bspline2(VFCHI(jj,jfloat));
05403                     for (kk=km2; kk<=kp2; kk++) {
05404                         mz = bspline2(VFCHI(kk,kfloat));
05405                         thee->charge[IJK(ii,jj,kk)] += (charge*mx*my*mz);
05406                     }
05407                 }
05408             }
05409
05410         } /* endif (on the mesh) */
05411     } /* endfor (each atom) */
05412 }
05413
05414 VPUBLIC int Vpmg_fillco(Vpmg *thee,
05415       Vsurf_Meth surfMeth, double splineWin, Vchrg_Meth chargeMeth,
05416       int useDielXMap,   Vgrid *dielXMap,
05417       int useDielYMap,   Vgrid *dielYMap,
05418       int useDielZMap,   Vgrid *dielZMap,
05419       int useKappaMap,   Vgrid *kappaMap,
05420       int usePotMap,    Vgrid *potMap,
05421       int useChargeMap,  Vgrid *chargeMap) {
05422
05423     Vpbe *pbe;
05424     double xmin, xmax, ymin, ymax, zmin, zmax;
05425     double xlen, ylen, zlen, hx, hy, hzed;
05426     double epsw, epsp, ionstr;
05427     int i, nx, ny, nz, islap;
```

```
05428  Vrc_Codes rc;
05429
05430     if (thee == VNULL) {
05431         Vnm_print(2, "Vpmg_fillco:  got NULL thee!\n");
05432         return 0;
05433     }
05434
05435     thee->surfMeth = surfMeth;
05436     thee->splineWin = splineWin;
05437     thee->chargeMeth = chargeMeth;
05438     thee->useDielXMap = useDielXMap;
05439     if (thee->useDielXMap) thee->dielXMap = dielXMap;
05440     thee->useDielYMap = useDielYMap;
05441     if (thee->useDielYMap) thee->dielYMap = dielYMap;
05442     thee->useDielZMap = useDielZMap;
05443     if (thee->useDielZMap) thee->dielZMap = dielZMap;
05444     thee->useKappaMap = useKappaMap;
05445     if (thee->useKappaMap) thee->kappaMap = kappaMap;
05446  thee->usePotMap = usePotMap;
05447     if (thee->usePotMap) thee->potMap = potMap;
05448     thee->useChargeMap = useChargeMap;
05449     if (thee->useChargeMap) thee->chargeMap = chargeMap;
05450
05451     /* Get PBE info */
05452     pbe = thee->pbe;
05453     ionstr = Vpbe_getBulkIonicStrength(pbe);
05454     epsw = Vpbe_getSolventDiel(pbe);
05455     epsp = Vpbe_getSoluteDiel(pbe);
05456
05457     /* Mesh info */
05458     nx = thee->pmgp->nx;
05459     ny = thee->pmgp->ny;
05460     nz = thee->pmgp->nz;
05461     hx = thee->pmgp->hx;
05462     hy = thee->pmgp->hy;
05463     hzed = thee->pmgp->hzed;
05464
05465     /* Define the total domain size */
05466     xlen = thee->pmgp->xlen;
05467     ylen = thee->pmgp->ylen;
05468     zlen = thee->pmgp->zlen;
05469
05470     /* Define the min/max dimensions */
05471     xmin = thee->pmgp->xcent - (xlen/2.0);
05472     thee->pmgp->xmin = xmin;
05473     ymin = thee->pmgp->ycent - (ylen/2.0);
05474     thee->pmgp->ymin = ymin;
05475     zmin = thee->pmgp->zcent - (zlen/2.0);
05476     thee->pmgp->zmin = zmin;
05477     xmax = thee->pmgp->xcent + (xlen/2.0);
05478     thee->pmgp->xmax = xmax;
05479     ymax = thee->pmgp->ycent + (ylen/2.0);
05480     thee->pmgp->ymax = ymax;
05481     zmax = thee->pmgp->zcent + (zlen/2.0);
05482     thee->pmgp->zmax = zmax;
05483     thee->rparm[2] = xmin;
05484     thee->rparm[3] = xmax;
```

```
05485     thee->rparm[4] = ymin;
05486     thee->rparm[5] = ymax;
05487     thee->rparm[6] = zmin;
05488     thee->rparm[7] = zmax;
05489
05490     /* This is a flag that gets set if the operator is a simple Laplacian;
05491      * i.e., in the case of a homogenous dielectric and zero ionic strength
05492   * The operator cannot be a simple Laplacian if maps are read in. */
05493   if(thee->useDielXMap || thee->useDielYMap || thee->useDielZMap ||
05494     thee->useKappaMap || thee->usePotMap){
05495    islap = 0;
05496   }else if ( (ionstr < VPMGSMALL) && (VABS(epsp-epsw) < VPMGSMALL) ){
05497    islap = 1;
05498   }else{
05499    islap = 0;
05500   }
05501
05502     /* Fill the mesh point coordinate arrays */
05503     for (i=0; i<nx; i++) thee->xf[i] = xmin + i*hx;
05504     for (i=0; i<ny; i++) thee->yf[i] = ymin + i*hy;
05505     for (i=0; i<nz; i++) thee->zf[i] = zmin + i*hzed;
05506
05507     /* Reset the tcf array */
05508     for (i=0; i<(nx*ny*nz); i++) thee->tcf[i] = 0.0;
05509
05510     /* Fill in the source term (atomic charges) */
05511     Vnm_print(0, "Vpmg_fillco:  filling in source term.\n");
05512     rc = fillcoCharge(thee);
05513   switch(rc) {
05514    case VRC_SUCCESS:
05515     break;
05516    case VRC_WARNING:
05517     Vnm_print(2, "Vpmg_fillco:  non-fatal errors while filling charge map!\n");
05518     break;
05519    case VRC_FAILURE:
05520     Vnm_print(2, "Vpmg_fillco:  fatal errors while filling charge map!\n");
05521     return 0;
05522     break;
05523   }
05524
05525     /* THE FOLLOWING NEEDS TO BE DONE IF WE'RE NOT USING A SIMPLE LAPLACIAN
05526      * OPERATOR */
05527     if (!islap) {
05528         Vnm_print(0, "Vpmg_fillco:  marking ion and solvent accessibility.\n");
05529         fillcoCoef(thee);
05530         Vnm_print(0, "Vpmg_fillco:  done filling coefficient arrays\n");
05531
05532     } else { /* else (!islap) ==> It's a Laplacian operator! */
05533
05534         for (i=0; i<(nx*ny*nz); i++) {
05535             thee->kappa[i] = 0.0;
05536             thee->epsx[i] = epsp;
05537             thee->epsy[i] = epsp;
05538             thee->epsz[i] = epsp;
05539         }
05540
05541     } /* endif (!islap) */
```

```
05542
05543     /* Fill the boundary arrays (except when focusing, bcfl = 4) */
05544     if (thee->pmgp->bcfl != BCFL_FOCUS) {
05545         Vnm_print(0, "Vpmg_fillco:  filling boundary arrays\n");
05546         bcCalc(thee);
05547         Vnm_print(0, "Vpmg_fillco:  done filling boundary arrays\n");
05548     }
05549
05550     thee->filled = 1;
05551
05552     return 1;
05553 }
05554
05555
05556 VPUBLIC int Vpmg_force(Vpmg *thee, double *force, int atomID,
05557   Vsurf_Meth srfm, Vchrg_Meth chgm) {
05558
05559     int rc = 1;
05560     double qfF[3];                  /* Charge-field force */
05561     double dbF[3];                  /* Dielectric boundary force */
05562     double ibF[3];                  /* Ion boundary force */
05563     double npF[3];                  /* Non-polar boundary force */
05564
05565     VASSERT(thee != VNULL);
05566
05567     rc = rc && Vpmg_dbForce(thee, qfF, atomID, srfm);
05568     rc = rc && Vpmg_ibForce(thee, dbF, atomID, srfm);
05569     rc = rc && Vpmg_qfForce(thee, ibF, atomID, chgm);
05570
05571     force[0] = qfF[0] + dbF[0] + ibF[0];
05572     force[1] = qfF[1] + dbF[1] + ibF[1];
05573     force[2] = qfF[2] + dbF[2] + ibF[2];
05574
05575     return rc;
05576
05577 }
05578
05579 VPUBLIC int Vpmg_ibForce(Vpmg *thee, double *force, int atomID,
05580   Vsurf_Meth srfm) {
05581
05582     Valist *alist;
05583     Vacc *acc;
05584     Vpbe *pbe;
05585     Vatom *atom;
05586
05587     double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
05588     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;
05589     double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
05590     double izmagic;
05591     int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
05592
05593  /* For nonlinear forces */
05594  int ichop, nchop, nion, m;
05595     double ionConc[MAXION], ionRadii[MAXION], ionQ[MAXION], ionstr;
05596
05597     VASSERT(thee != VNULL);
05598
```

```
05599     acc = thee->pbe->acc;
05600     atom = Valist_getAtom(thee->pbe->alist, atomID);
05601     apos = Vatom_getPosition(atom);
05602     arad = Vatom_getRadius(atom);
05603
05604     /* Reset force */
05605     force[0] = 0.0;
05606     force[1] = 0.0;
05607     force[2] = 0.0;
05608
05609     /* Check surface definition */
05610     if ((srfm != VSM_SPLINE) && (srfm!=VSM_SPLINE3) && (srfm!=VSM_SPLINE4)) {
05611         Vnm_print(2, "Vpmg_ibForce:  Forces *must* be calculated with \
05612 spline-based surfaces!\n");
05613         Vnm_print(2, "Vpmg_ibForce:  Skipping ionic boundary force \
05614 calculation!\n");
05615         return 0;
05616     }
05617
05618     /* If we aren't in the current position, then we're done */
05619     if (atom->partID == 0) return 1;
05620
05621     /* Get PBE info */
05622     pbe = thee->pbe;
05623     acc = pbe->acc;
05624     alist = pbe->alist;
05625     irad = Vpbe_getMaxIonRadius(pbe);
05626     zkappa2 = Vpbe_getZkappa2(pbe);
05627     izmagic = 1.0/Vpbe_getZmagic(pbe);
05628
05629  ionstr = Vpbe_getBulkIonicStrength(pbe);
05630     Vpbe_getIons(pbe, &nion, ionConc, ionRadii, ionQ);
05631
05632     /* Mesh info */
05633     nx = thee->pmgp->nx;
05634     ny = thee->pmgp->ny;
05635     nz = thee->pmgp->nz;
05636     hx = thee->pmgp->hx;
05637     hy = thee->pmgp->hy;
05638     hzed = thee->pmgp->hzed;
05639     xlen = thee->pmgp->xlen;
05640     ylen = thee->pmgp->ylen;
05641     zlen = thee->pmgp->zlen;
05642     xmin = thee->pmgp->xmin;
05643     ymin = thee->pmgp->ymin;
05644     zmin = thee->pmgp->zmin;
05645     xmax = thee->pmgp->xmax;
05646     ymax = thee->pmgp->ymax;
05647     zmax = thee->pmgp->zmax;
05648
05649     /* Sanity check: there is no force if there is zero ionic strength */
05650     if (zkappa2 < VPMGSMALL) {
05651 #ifndef VAPBSQUIET
05652         Vnm_print(2, "Vpmg_ibForce:  No force for zero ionic strength!\n");
05653 #endif
05654         return 1;
05655     }
```

```
05656
05657     /* Make sure we're on the grid */
05658     if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
05659       (apos[1]<=ymin) || (apos[1]>=ymax)  || \
05660       (apos[2]<=zmin) || (apos[2]>=zmax)) {
05661   if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
05662    (thee->pmgp->bcfl != BCFL_MAP)) {
05663             Vnm_print(2, "Vpmg_ibForce:  Atom #%d at (%4.3f, %4.3f, %4.3f) is off
      the mesh (ignoring):\n",
05664                   atom, apos[0], apos[1], apos[2]);
05665             Vnm_print(2, "Vpmg_ibForce:    xmin = %g, xmax = %g\n",
05666               xmin, xmax);
05667             Vnm_print(2, "Vpmg_ibForce:    ymin = %g, ymax = %g\n",
05668               ymin, ymax);
05669             Vnm_print(2, "Vpmg_ibForce:    zmin = %g, zmax = %g\n",
05670               zmin, zmax);
05671         }
05672         fflush(stderr);
05673     } else {
05674
05675         /* Convert the atom position to grid reference frame */
05676         position[0] = apos[0] - xmin;
05677         position[1] = apos[1] - ymin;
05678         position[2] = apos[2] - zmin;
05679
05680         /* Integrate over points within this atom's (inflated) radius */
05681         rtot = (irad + arad + thee->splineWin);
05682         rtot2 = VSQR(rtot);
05683         dx = rtot + 0.5*hx;
05684         imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
05685         imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
05686         for (i=imin; i<=imax; i++) {
05687             dx2 = VSQR(position[0] - hx*i);
05688             if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
05689             else dy = 0.5*hy;
05690             jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
05691             jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
05692             for (j=jmin; j<=jmax; j++) {
05693                 dy2 = VSQR(position[1] - hy*j);
05694                 if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
05695                 else dz = 0.5*hzed;
05696                 kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
05697                 kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
05698                 for (k=kmin; k<=kmax; k++) {
05699                     dz2 = VSQR(k*hzed - position[2]);
05700                     /* See if grid point is inside ivdw radius and set kappa
05701                      * accordingly (do spline assignment here) */
05702                     if ((dz2 + dy2 + dx2) <= rtot2) {
05703                         gpos[0] = i*hx + xmin;
05704                         gpos[1] = j*hy + ymin;
05705                         gpos[2] = k*hzed + zmin;
05706
05707       /* Select the correct function based on the surface definition
05708        * (now including the 7th order polynomial) */
05709       Vpmg_splineSelect(srfm,acc, gpos,thee->splineWin, irad, atom, tgrad);
05710
05711                         if (thee->pmgp->nonlin) {
```

```
05712                               /* Nonlinear forces */
05713                               fmag = 0.0;
05714                               nchop = 0;
05715                               for (m=0; m<nion; m++) {
05716                                   fmag += (thee->kappa[IJK(i,j,k)])*ionConc[m]*(
      Vcap_exp(-ionQ[m]*thee->u[IJK(i,j,k)], &ichop)-1.0)/ionstr;
05717                                   nchop += ichop;
05718                               }
05719           /*          if (nchop > 0) Vnm_print(2, "Vpmg_ibForece:  Chopped EXP %d ti
      mes!\n", nchop);*/
05720                               force[0] += (zkappa2*fmag*tgrad[0]);
05721                               force[1] += (zkappa2*fmag*tgrad[1]);
05722                               force[2] += (zkappa2*fmag*tgrad[2]);
05723                           } else {
05724                               /* Use of bulk factor (zkappa2) OK here becuase
05725                                * LPBE force approximation */
05726                               /* NAB -- did we forget a kappa factor here??? */
05727                               fmag = VSQR(thee->u[IJK(i,j,k)])*(thee->kappa[IJK(i,j
      ,k)]);
05728                               force[0] += (zkappa2*fmag*tgrad[0]);
05729                               force[1] += (zkappa2*fmag*tgrad[1]);
05730                               force[2] += (zkappa2*fmag*tgrad[2]);
05731                           }
05732                       }
05733                   } /* k loop */
05734               } /* j loop */
05735           } /* i loop */
05736       }
05737       force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
05738       force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
05739       force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
05740
05741       return 1;
05742   }
05743
05744   VPUBLIC int Vpmg_dbForce(Vpmg *thee, double *dbForce, int atomID,
05745           Vsurf_Meth srfm) {
05746
05747       Vacc *acc;
05748       Vpbe *pbe;
05749       Vatom *atom;
05750
05751       double *apos, position[3], arad, srad, hx, hy, hzed, izmagic, deps, depsi;
05752       double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2, epsp;
05753       double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
05754       double *u, Hxijk, Hyijk, Hzijk, Hxim1jk, Hyijm1k, Hzijkm1;
05755       double dHxijk[3], dHyijk[3], dHzijk[3], dHxim1jk[3], dHyijm1k[3];
05756       double dHzijkm1[3];
05757       int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
05758
05759       VASSERT(thee != VNULL);
05760       if (!thee->filled) {
05761           Vnm_print(2, "Vpmg_dbForce:  Need to callVpmg_fillco!\n");
05762           return 0;
05763       }
05764
05765       acc = thee->pbe->acc;
```

```
05766     atom = Valist_getAtom(thee->pbe->alist, atomID);
05767     apos = Vatom_getPosition(atom);
05768     arad = Vatom_getRadius(atom);
05769     srad = Vpbe_getSolventRadius(thee->pbe);
05770 
05771     /* Reset force */
05772     dbForce[0] = 0.0;
05773     dbForce[1] = 0.0;
05774     dbForce[2] = 0.0;
05775 
05776     /* Check surface definition */
05777     if ((srfm != VSM_SPLINE) && (srfm!=VSM_SPLINE3) && (srfm!=VSM_SPLINE4)) {
05778         Vnm_print(2, "Vpmg_dbForce:  Forces *must* be calculated with \
05779 spline-based surfaces!\n");
05780         Vnm_print(2, "Vpmg_dbForce:  Skipping dielectric/apolar boundary \
05781 force calculation!\n");
05782         return 0;
05783     }
05784 
05785 
05786     /* If we aren't in the current position, then we're done */
05787     if (atom->partID == 0) return 1;
05788 
05789     /* Get PBE info */
05790     pbe = thee->pbe;
05791     acc = pbe->acc;
05792     epsp = Vpbe_getSoluteDiel(pbe);
05793     epsw = Vpbe_getSolventDiel(pbe);
05794     kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*Vunit_kb;
05795     izmagic = 1.0/Vpbe_getZmagic(pbe);
05796 
05797     /* Mesh info */
05798     nx = thee->pmgp->nx;
05799     ny = thee->pmgp->ny;
05800     nz = thee->pmgp->nz;
05801     hx = thee->pmgp->hx;
05802     hy = thee->pmgp->hy;
05803     hzed = thee->pmgp->hzed;
05804     xlen = thee->pmgp->xlen;
05805     ylen = thee->pmgp->ylen;
05806     zlen = thee->pmgp->zlen;
05807     xmin = thee->pmgp->xmin;
05808     ymin = thee->pmgp->ymin;
05809     zmin = thee->pmgp->zmin;
05810     xmax = thee->pmgp->xmax;
05811     ymax = thee->pmgp->ymax;
05812     zmax = thee->pmgp->zmax;
05813     u = thee->u;
05814 
05815     /* Sanity check: there is no force if there is zero ionic strength */
05816     if (VABS(epsp-epsw) < VPMGSMALL) {
05817   Vnm_print(0, "Vpmg_dbForce: No force for uniform dielectric!\n");
05818   return 1;
05819     }
05820     deps = (epsw - epsp);
05821     depsi = 1.0/deps;
05822     rtot = (arad + thee->splineWin + srad);
```

```
05823
05824     /* Make sure we're on the grid */
05825     /* Grid checking modified by Matteo Rotter */
05826     if ((apos[0]<=xmin + rtot) || (apos[0]>=xmax - rtot)  || \
05827    (apos[1]<=ymin + rtot) || (apos[1]>=ymax - rtot)  || \
05828    (apos[2]<=zmin + rtot) || (apos[2]>=zmax - rtot)) {
05829    if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
05830     (thee->pmgp->bcfl != BCFL_MAP)) {
05831              Vnm_print(2, "Vpmg_dbForce:  Atom #%d at (%4.3f, %4.3f, %4.3f) is off
      the mesh (ignoring):\n",
05832         atomID, apos[0], apos[1], apos[2]);
05833              Vnm_print(2, "Vpmg_dbForce:    xmin = %g, xmax = %g\n",
05834         xmin, xmax);
05835              Vnm_print(2, "Vpmg_dbForce:    ymin = %g, ymax = %g\n",
05836         ymin, ymax);
05837              Vnm_print(2, "Vpmg_dbForce:    zmin = %g, zmax = %g\n",
05838         zmin, zmax);
05839          }
05840          fflush(stderr);
05841     } else {
05842
05843          /* Convert the atom position to grid reference frame */
05844          position[0] = apos[0] - xmin;
05845          position[1] = apos[1] - ymin;
05846          position[2] = apos[2] - zmin;
05847
05848          /* Integrate over points within this atom's (inflated) radius */
05849          rtot2 = VSQR(rtot);
05850          dx = rtot/hx;
05851          imin = (int)floor((position[0]-rtot)/hx);
05852          if (imin < 1) {
05853              Vnm_print(2, "Vpmg_dbForce:  Atom %d off grid!\n", atomID);
05854              return 0;
05855          }
05856          imax = (int)ceil((position[0]+rtot)/hx);
05857          if (imax > (nx-2)) {
05858              Vnm_print(2, "Vpmg_dbForce:  Atom %d off grid!\n", atomID);
05859              return 0;
05860          }
05861          jmin = (int)floor((position[1]-rtot)/hy);
05862          if (jmin < 1) {
05863              Vnm_print(2, "Vpmg_dbForce:  Atom %d off grid!\n", atomID);
05864              return 0;
05865          }
05866          jmax = (int)ceil((position[1]+rtot)/hy);
05867          if (jmax > (ny-2)) {
05868              Vnm_print(2, "Vpmg_dbForce:  Atom %d off grid!\n", atomID);
05869              return 0;
05870          }
05871          kmin = (int)floor((position[2]-rtot)/hzed);
05872          if (kmin < 1) {
05873              Vnm_print(2, "Vpmg_dbForce:  Atom %d off grid!\n", atomID);
05874              return 0;
05875          }
05876          kmax = (int)ceil((position[2]+rtot)/hzed);
05877          if (kmax > (nz-2)) {
05878              Vnm_print(2, "Vpmg_dbForce:  Atom %d off grid!\n", atomID);
```

```
05879             return 0;
05880         }
05881         for (i=imin; i<=imax; i++) {
05882             for (j=jmin; j<=jmax; j++) {
05883                 for (k=kmin; k<=kmax; k++) {
05884                     /* i,j,k */
05885                     gpos[0] = (i+0.5)*hx + xmin;
05886                     gpos[1] = j*hy + ymin;
05887                     gpos[2] = k*hzed + zmin;
05888                     Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
05889
05890       /* Select the correct function based on the surface definition
05891        * (now including the 7th order polynomial) */
05892       Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin, 0.,atom, dHxijk);
05893       /*
05894        switch (srfm) {
05895         case VSM_SPLINE :
05896          Vacc_splineAccGradAtomNorm(acc, gpos, thee->splineWin, 0.,
05897                atom, dHxijk);
05898          break;
05899         case VSM_SPLINE4 :
05900          Vacc_splineAccGradAtomNorm4(acc, gpos, thee->splineWin, 0.,
05901                atom, dHxijk);
05902          break;
05903         default:
05904          Vnm_print(2, "Vpmg_dbnbForce: Unknown surface method.\n");
05905          return;
05906        }
05907        */
05908                     for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
05909                     gpos[0] = i*hx + xmin;
05910                     gpos[1] = (j+0.5)*hy + ymin;
05911                     gpos[2] = k*hzed + zmin;
05912                     Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
05913
05914       /* Select the correct function based on the surface definition
05915        * (now including the 7th order polynomial) */
05916       Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin, 0.,atom, dHyijk);
05917
05918       for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
05919                     gpos[0] = i*hx + xmin;
05920                     gpos[1] = j*hy + ymin;
05921                     gpos[2] = (k+0.5)*hzed + zmin;
05922                     Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
05923
05924       /* Select the correct function based on the surface definition
05925        * (now including the 7th order polynomial) */
05926       Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin, 0.,atom, dHzijk);
05927
05928       for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
05929                     /* i-1,j,k */
05930                     gpos[0] = (i-0.5)*hx + xmin;
05931                     gpos[1] = j*hy + ymin;
05932                     gpos[2] = k*hzed + zmin;
05933                     Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
05934
05935       /* Select the correct function based on the surface definition
```

```
05936       * (now including the 7th order polynomial) */
05937      Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin, 0.,atom, dHxim1jk);
05938
05939      for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
05940                          /* i,j-1,k */
05941                          gpos[0] = i*hx + xmin;
05942                          gpos[1] = (j-0.5)*hy + ymin;
05943                          gpos[2] = k*hzed + zmin;
05944                          Hyijm1k = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
05945
05946      /* Select the correct function based on the surface definition
05947       * (now including the 7th order polynomial) */
05948      Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin, 0.,atom, dHyijm1k);
05949
05950      for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;
05951                          /* i,j,k-1 */
05952                          gpos[0] = i*hx + xmin;
05953                          gpos[1] = j*hy + ymin;
05954                          gpos[2] = (k-0.5)*hzed + zmin;
05955                          Hzijkm1 = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
05956
05957      /* Select the correct function based on the surface definition
05958       * (now including the 7th order polynomial) */
05959      Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin, 0.,atom, dHzijkm1);
05960
05961      for (l=0; l<3; l++) dHzijkm1[l] *= Hzijkm1;
05962                          /* *** CALCULATE DIELECTRIC BOUNDARY FORCES *** */
05963                          dbFmag = u[IJK(i,j,k)];
05964                          tgrad[0] =
05965       (dHxijk[0]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
05966        +  dHxim1jk[0]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
05967       + (dHyijk[0]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
05968          +  dHyijm1k[0]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
05969       + (dHzijk[0]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
05970          + dHzijkm1[0]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
05971                          tgrad[1] =
05972       (dHxijk[1]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
05973        +  dHxim1jk[1]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
05974       + (dHyijk[1]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
05975          +  dHyijm1k[1]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
05976       + (dHzijk[1]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
05977          + dHzijkm1[1]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
05978                          tgrad[2] =
05979       (dHxijk[2]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
05980        +  dHxim1jk[2]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
05981       + (dHyijk[2]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
05982          +  dHyijm1k[2]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
05983       + (dHzijk[2]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
05984          + dHzijkm1[2]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
05985      dbForce[0] += (dbFmag*tgrad[0]);
05986      dbForce[1] += (dbFmag*tgrad[1]);
05987      dbForce[2] += (dbFmag*tgrad[2]);
05988
05989     } /* k loop */
05990    } /* j loop */
05991   } /* i loop */
05992
```

```
05993         dbForce[0] = -dbForce[0]*hx*hy*hzed*deps*0.5*izmagic;
05994         dbForce[1] = -dbForce[1]*hx*hy*hzed*deps*0.5*izmagic;
05995         dbForce[2] = -dbForce[2]*hx*hy*hzed*deps*0.5*izmagic;
05996  }
05997
05998  return 1;
05999 }
06000
06001 VPUBLIC int Vpmg_qfForce(Vpmg *thee, double *force, int atomID,
06002   Vchrg_Meth chgm) {
06003
06004     double tforce[3];
06005
06006     /* Reset force */
06007     force[0] = 0.0;
06008     force[1] = 0.0;
06009     force[2] = 0.0;
06010
06011     /* Check surface definition */
06012     if (chgm != VCM_BSPL2) {
06013         Vnm_print(2, "Vpmg_qfForce:  It is recommended that forces be \
06014 calculated with the\n");
06015         Vnm_print(2, "Vpmg_qfForce:  cubic spline charge discretization \
06016 scheme\n");
06017  }
06018
06019     switch (chgm) {
06020         case VCM_TRIL:
06021             qfForceSpline1(thee, tforce, atomID);
06022             break;
06023         case VCM_BSPL2:
06024             qfForceSpline2(thee, tforce, atomID);
06025             break;
06026   case VCM_BSPL4:
06027    qfForceSpline4(thee, tforce, atomID);
06028    break;
06029         default:
06030             Vnm_print(2, "Vpmg_qfForce:  Undefined charge discretization \
06031 method (%d)!\n", chgm);
06032             Vnm_print(2, "Vpmg_qfForce:  Forces not calculated!\n");
06033             return 0;
06034     }
06035
06036     /* Assign forces */
06037     force[0] = tforce[0];
06038     force[1] = tforce[1];
06039     force[2] = tforce[2];
06040
06041     return 1;
06042 }
06043
06044
06045 VPRIVATE void qfForceSpline1(Vpmg *thee, double *force, int atomID) {
06046
06047     Vatom *atom;
06048
06049     double *apos, position[3], hx, hy, hzed;
```

```
06050     double xmin, ymin, zmin, xmax, ymax, zmax;
06051     double dx, dy, dz;
06052     double *u, charge, ifloat, jfloat, kfloat;
06053     int nx, ny, nz, ihi, ilo, jhi, jlo, khi, klo;
06054
06055     VASSERT(thee != VNULL);
06056
06057     atom = Valist_getAtom(thee->pbe->alist, atomID);
06058     apos = Vatom_getPosition(atom);
06059     charge = Vatom_getCharge(atom);
06060
06061     /* Reset force */
06062     force[0] = 0.0;
06063     force[1] = 0.0;
06064     force[2] = 0.0;
06065
06066     /* If we aren't in the current position, then we're done */
06067     if (atom->partID == 0) return;
06068
06069     /* Mesh info */
06070     nx = thee->pmgp->nx;
06071     ny = thee->pmgp->ny;
06072     nz = thee->pmgp->nz;
06073     hx = thee->pmgp->hx;
06074     hy = thee->pmgp->hy;
06075     hzed = thee->pmgp->hzed;
06076     xmin = thee->pmgp->xmin;
06077     ymin = thee->pmgp->ymin;
06078     zmin = thee->pmgp->zmin;
06079     xmax = thee->pmgp->xmax;
06080     ymax = thee->pmgp->ymax;
06081     zmax = thee->pmgp->zmax;
06082     u = thee->u;
06083
06084     /* Make sure we're on the grid */
06085     if ((apos[0]<=xmin) || (apos[0]>=xmax) || (apos[1]<=ymin) || \
06086         (apos[1]>=ymax) || (apos[2]<=zmin) || (apos[2]>=zmax)) {
06087   if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
06088    (thee->pmgp->bcfl != BCFL_MAP)) {
06089             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]);
06090             Vnm_print(2, "Vpmg_qfForce:    xmin = %g, xmax = %g\n", xmin, xmax);
06091             Vnm_print(2, "Vpmg_qfForce:    ymin = %g, ymax = %g\n", ymin, ymax);
06092             Vnm_print(2, "Vpmg_qfForce:    zmin = %g, zmax = %g\n", zmin, zmax);
06093         }
06094         fflush(stderr);
06095     } else {
06096
06097         /* Convert the atom position to grid coordinates */
06098         position[0] = apos[0] - xmin;
06099         position[1] = apos[1] - ymin;
06100         position[2] = apos[2] - zmin;
06101         ifloat = position[0]/hx;
06102         jfloat = position[1]/hy;
06103         kfloat = position[2]/hzed;
06104         ihi = (int)ceil(ifloat);
06105         ilo = (int)floor(ifloat);
```

```
06106         jhi = (int)ceil(jfloat);
06107         jlo = (int)floor(jfloat);
06108         khi = (int)ceil(kfloat);
06109         klo = (int)floor(kfloat);
06110         VASSERT((ihi < nx) && (ihi >=0));
06111         VASSERT((ilo < nx) && (ilo >=0));
06112         VASSERT((jhi < ny) && (jhi >=0));
06113         VASSERT((jlo < ny) && (jlo >=0));
06114         VASSERT((khi < nz) && (khi >=0));
06115         VASSERT((klo < nz) && (klo >=0));
06116         dx = ifloat - (double)(ilo);
06117         dy = jfloat - (double)(jlo);
06118         dz = kfloat - (double)(klo);
06119 
06120 
06121 #if 0
06122         Vnm_print(1, "Vpmg_qfForce: (DEBUG) u ~ %g\n",
06123           dx    *dy    *dz    *u[IJK(ihi,jhi,khi)]
06124          +dx    *dy    *(1-dz)*u[IJK(ihi,jhi,klo)]
06125          +dx    *(1-dy)*dz    *u[IJK(ihi,jlo,khi)]
06126          +dx    *(1-dy)*(1-dz)*u[IJK(ihi,jlo,klo)]
06127          +(1-dx)*dy    *dz    *u[IJK(ilo,jhi,khi)]
06128          +(1-dx)*dy    *(1-dz)*u[IJK(ilo,jhi,klo)]
06129          +(1-dx)*(1-dy)*dz    *u[IJK(ilo,jlo,khi)]
06130          +(1-dx)*(1-dy)*(1-dz)*u[IJK(ilo,jlo,klo)]);
06131 #endif
06132 
06133 
06134         if ((dx > VPMGSMALL) && (VABS(1.0-dx) > VPMGSMALL)) {
06135             force[0] =
06136               -charge*(dy    *dz    *u[IJK(ihi,jhi,khi)]
06137                      + dy    *(1-dz)*u[IJK(ihi,jhi,klo)]
06138                      + (1-dy)*dz    *u[IJK(ihi,jlo,khi)]
06139                      + (1-dy)*(1-dz)*u[IJK(ihi,jlo,klo)]
06140                      - dy    *dz    *u[IJK(ilo,jhi,khi)]
06141                      - dy    *(1-dz)*u[IJK(ilo,jhi,klo)]
06142                      - (1-dy)*dz    *u[IJK(ilo,jlo,khi)]
06143                      - (1-dy)*(1-dz)*u[IJK(ilo,jlo,klo)])/hx;
06144         } else {
06145             force[0] = 0;
06146             Vnm_print(0,
06147               "Vpmg_qfForce:  Atom %d on x gridline; zero x-force\n", atomID);
06148         }
06149         if ((dy > VPMGSMALL) && (VABS(1.0-dy) > VPMGSMALL)) {
06150             force[1] =
06151               -charge*(dx    *dz    *u[IJK(ihi,jhi,khi)]
06152                      + dx    *(1-dz)*u[IJK(ihi,jhi,klo)]
06153                      - dx    *dz    *u[IJK(ihi,jlo,khi)]
06154                      - dx    *(1-dz)*u[IJK(ihi,jlo,klo)]
06155                      + (1-dx)*dz    *u[IJK(ilo,jhi,khi)]
06156                      + (1-dx)*(1-dz)*u[IJK(ilo,jhi,klo)]
06157                      - (1-dx)*dz    *u[IJK(ilo,jlo,khi)]
06158                      - (1-dx)*(1-dz)*u[IJK(ilo,jlo,klo)])/hy;
06159         } else {
06160             force[1] = 0;
06161             Vnm_print(0,
06162               "Vpmg_qfForce:  Atom %d on y gridline; zero y-force\n", atomID);
```

```
06163         }
06164         if ((dz > VPMGSMALL) && (VABS(1.0-dz) > VPMGSMALL)) {
06165             force[2] =
06166               -charge*(dy    *dx    *u[IJK(ihi,jhi,khi)]
06167                      - dy    *dx    *u[IJK(ihi,jhi,klo)]
06168                      + (1-dy)*dx    *u[IJK(ihi,jlo,khi)]
06169                      - (1-dy)*dx    *u[IJK(ihi,jlo,klo)]
06170                      + dy    *(1-dx)*u[IJK(ilo,jhi,khi)]
06171                      - dy    *(1-dx)*u[IJK(ilo,jhi,klo)]
06172                      + (1-dy)*(1-dx)*u[IJK(ilo,jlo,khi)]
06173                      - (1-dy)*(1-dx)*u[IJK(ilo,jlo,klo)])/hzed;
06174         } else {
06175             force[2] = 0;
06176             Vnm_print(0,
06177               "Vpmg_qfForce:  Atom %d on z gridline; zero z-force\n", atomID);
06178         }
06179     }
06180 }
06181
06182 VPRIVATE void qfForceSpline2(Vpmg *thee, double *force, int atomID) {
06183
06184     Vatom *atom;
06185
06186     double *apos, position[3], hx, hy, hzed;
06187     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
06188     double mx, my, mz, dmx, dmy, dmz;
06189     double *u, charge, ifloat, jfloat, kfloat;
06190     int nx, ny, nz, im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1;
06191     int kp1, kp2, ii, jj, kk;
06192
06193     VASSERT(thee != VNULL);
06194
06195     atom = Valist_getAtom(thee->pbe->alist, atomID);
06196     apos = Vatom_getPosition(atom);
06197     charge = Vatom_getCharge(atom);
06198
06199     /* Reset force */
06200     force[0] = 0.0;
06201     force[1] = 0.0;
06202     force[2] = 0.0;
06203
06204     /* If we aren't in the current position, then we're done */
06205     if (atom->partID == 0) return;
06206
06207     /* Mesh info */
06208     nx = thee->pmgp->nx;
06209     ny = thee->pmgp->ny;
06210     nz = thee->pmgp->nz;
06211     hx = thee->pmgp->hx;
06212     hy = thee->pmgp->hy;
06213     hzed = thee->pmgp->hzed;
06214     xlen = thee->pmgp->xlen;
06215     ylen = thee->pmgp->ylen;
06216     zlen = thee->pmgp->zlen;
06217     xmin = thee->pmgp->xmin;
06218     ymin = thee->pmgp->ymin;
06219     zmin = thee->pmgp->zmin;
```

```
06220     xmax = thee->pmgp->xmax;
06221     ymax = thee->pmgp->ymax;
06222     zmax = thee->pmgp->zmax;
06223     u = thee->u;
06224
06225     /* Make sure we're on the grid */
06226     if ((apos[0]<=(xmin+hx))    || (apos[0]>=(xmax-hx)) \
06227      || (apos[1]<=(ymin+hy))    || (apos[1]>=(ymax-hy)) \
06228      || (apos[2]<=(zmin+hzed)) || (apos[2]>=(zmax-hzed))) {
06229   if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
06230    (thee->pmgp->bcfl != BCFL_MAP)) {
06231             Vnm_print(2, "qfForceSpline2:  Atom #%d off the mesh \
06232     (ignoring)\n", atomID);
06233         }
06234         fflush(stderr);
06235
06236     } else {
06237
06238         /* Convert the atom position to grid coordinates */
06239         position[0] = apos[0] - xmin;
06240         position[1] = apos[1] - ymin;
06241         position[2] = apos[2] - zmin;
06242         ifloat = position[0]/hx;
06243         jfloat = position[1]/hy;
06244         kfloat = position[2]/hzed;
06245         ip1 = (int)ceil(ifloat);
06246         ip2 = ip1 + 1;
06247         im1 = (int)floor(ifloat);
06248         im2 = im1 - 1;
06249         jp1 = (int)ceil(jfloat);
06250         jp2 = jp1 + 1;
06251         jm1 = (int)floor(jfloat);
06252         jm2 = jm1 - 1;
06253         kp1 = (int)ceil(kfloat);
06254         kp2 = kp1 + 1;
06255         km1 = (int)floor(kfloat);
06256         km2 = km1 - 1;
06257
06258         /* This step shouldn't be necessary, but it saves nasty debugging
06259          * later on if something goes wrong */
06260         ip2 = VMIN2(ip2,nx-1);
06261         ip1 = VMIN2(ip1,nx-1);
06262         im1 = VMAX2(im1,0);
06263         im2 = VMAX2(im2,0);
06264         jp2 = VMIN2(jp2,ny-1);
06265         jp1 = VMIN2(jp1,ny-1);
06266         jm1 = VMAX2(jm1,0);
06267         jm2 = VMAX2(jm2,0);
06268         kp2 = VMIN2(kp2,nz-1);
06269         kp1 = VMIN2(kp1,nz-1);
06270         km1 = VMAX2(km1,0);
06271         km2 = VMAX2(km2,0);
06272
06273
06274         for (ii=im2; ii<=ip2; ii++) {
06275             mx = bspline2(VFCHI(ii,ifloat));
06276             dmx = dbspline2(VFCHI(ii,ifloat));
```

```
06277             for (jj=jm2; jj<=jp2; jj++) {
06278                 my = bspline2(VFCHI(jj,jfloat));
06279                 dmy = dbspline2(VFCHI(jj,jfloat));
06280                 for (kk=km2; kk<=kp2; kk++) {
06281                     mz = bspline2(VFCHI(kk,kfloat));
06282                     dmz = dbspline2(VFCHI(kk,kfloat));
06283 
06284                     force[0] += (charge*dmx*my*mz*u[IJK(ii,jj,kk)])/hx;
06285                     force[1] += (charge*mx*dmy*mz*u[IJK(ii,jj,kk)])/hy;
06286                     force[2] += (charge*mx*my*dmz*u[IJK(ii,jj,kk)])/hzed;
06287 
06288                 }
06289             }
06290         }
06291 
06292     }
06293 }
06294 
06295 VPRIVATE void qfForceSpline4(Vpmg *thee, double *force, int atomID) {
06296 
06297     Vatom *atom;
06298     double f, c, *u, *apos, position[3];
06299 
06300     /* Grid variables */
06301     int nx,ny,nz;
06302     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
06303     double hx, hy, hzed, ifloat, jfloat, kfloat;
06304 
06305     /* B-spline weights */
06306     double mx, my, mz, dmx, dmy, dmz;
06307     double mi, mj, mk;
06308 
06309     /* Loop indeces */
06310     int i, j, k, ii, jj, kk;
06311     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
06312 
06313     /* field */
06314     double e[3];
06315 
06316     VASSERT(thee != VNULL);
06317     VASSERT(thee->filled);
06318 
06319     atom = Valist_getAtom(thee->pbe->alist, atomID);
06320     apos = Vatom_getPosition(atom);
06321     c = Vatom_getCharge(atom);
06322 
06323     for (i=0;i<3;i++){
06324   e[i] = 0.0;
06325     }
06326 
06327     /* Mesh info */
06328     nx = thee->pmgp->nx;
06329     ny = thee->pmgp->ny;
06330     nz = thee->pmgp->nz;
06331     hx = thee->pmgp->hx;
06332     hy = thee->pmgp->hy;
06333     hzed = thee->pmgp->hzed;
```

```
06334     xlen = thee->pmgp->xlen;
06335     ylen = thee->pmgp->ylen;
06336     zlen = thee->pmgp->zlen;
06337     xmin = thee->pmgp->xmin;
06338     ymin = thee->pmgp->ymin;
06339     zmin = thee->pmgp->zmin;
06340     xmax = thee->pmgp->xmax;
06341     ymax = thee->pmgp->ymax;
06342     zmax = thee->pmgp->zmax;
06343     u = thee->u;
06344
06345     /* Make sure we're on the grid */
06346     if ((apos[0]<=(xmin+2*hx))   || (apos[0]>=(xmax-2*hx)) \
06347   || (apos[1]<=(ymin+2*hy))   || (apos[1]>=(ymax-2*hy)) \
06348   || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
06349         Vnm_print(2, "qfForceSpline4:  Atom off the mesh \
06350    (ignoring) %6.3f %6.3f %6.3f\n", apos[0], apos[1], apos[2]);
06351         fflush(stderr);
06352     } else {
06353
06354         /* Convert the atom position to grid coordinates */
06355         position[0] = apos[0] - xmin;
06356         position[1] = apos[1] - ymin;
06357         position[2] = apos[2] - zmin;
06358         ifloat = position[0]/hx;
06359         jfloat = position[1]/hy;
06360         kfloat = position[2]/hzed;
06361         ip1 = (int)ceil(ifloat);
06362         ip2 = ip1 + 2;
06363         im1 = (int)floor(ifloat);
06364         im2 = im1 - 2;
06365         jp1 = (int)ceil(jfloat);
06366         jp2 = jp1 + 2;
06367         jm1 = (int)floor(jfloat);
06368         jm2 = jm1 - 2;
06369         kp1 = (int)ceil(kfloat);
06370         kp2 = kp1 + 2;
06371         km1 = (int)floor(kfloat);
06372         km2 = km1 - 2;
06373
06374         /* This step shouldn't be necessary, but it saves nasty debugging
06375    * later on if something goes wrong */
06376         ip2 = VMIN2(ip2,nx-1);
06377         ip1 = VMIN2(ip1,nx-1);
06378         im1 = VMAX2(im1,0);
06379         im2 = VMAX2(im2,0);
06380         jp2 = VMIN2(jp2,ny-1);
06381         jp1 = VMIN2(jp1,ny-1);
06382         jm1 = VMAX2(jm1,0);
06383         jm2 = VMAX2(jm2,0);
06384         kp2 = VMIN2(kp2,nz-1);
06385         kp1 = VMIN2(kp1,nz-1);
06386         km1 = VMAX2(km1,0);
06387         km2 = VMAX2(km2,0);
06388
06389         for (ii=im2; ii<=ip2; ii++) {
06390             mi = VFCHI4(ii,ifloat);
```

```
06391             mx = bspline4(mi);
06392             dmx = dbspline4(mi);
06393             for (jj=jm2; jj<=jp2; jj++) {
06394                 mj = VFCHI4(jj,jfloat);
06395                 my = bspline4(mj);
06396                 dmy = dbspline4(mj);
06397                 for (kk=km2; kk<=kp2; kk++) {
06398                     mk = VFCHI4(kk,kfloat);
06399                     mz = bspline4(mk);
06400                     dmz = dbspline4(mk);
06401                     f = u[IJK(ii,jj,kk)];
06402                     /* Field */
06403                     e[0] += f*dmx*my*mz/hx;
06404                     e[1] += f*mx*dmy*mz/hy;
06405                     e[2] += f*mx*my*dmz/hzed;
06406                 }
06407             }
06408         }
06409     }
06410
06411     /* Monopole Force */
06412     force[0] = e[0]*c;
06413     force[1] = e[1]*c;
06414     force[2] = e[2]*c;
06415
06416 }
06417
06418 VPRIVATE void markFrac(
06419         double rtot, double *tpos,
06420         int nx, int ny, int nz,
06421         double hx, double hy, double hzed,
06422         double xmin, double ymin, double zmin,
06423         double *xarray, double *yarray, double *zarray) {
06424
06425     int i, j, k, imin, imax, jmin, jmax, kmin, kmax;
06426     double dx, dx2, dy, dy2, dz, dz2, a000, a001, a010, a100, r2;
06427     double x, xp, xm, y, yp, ym, zp, z, zm, xspan, yspan, zspan;
06428     double rtot2, pos[3];
06429
06430     /* Convert to grid reference frame */
06431     pos[0] = tpos[0] - xmin;
06432     pos[1] = tpos[1] - ymin;
06433     pos[2] = tpos[2] - zmin;
06434
06435     rtot2 = VSQR(rtot);
06436
06437     xspan = rtot + 2*hx;
06438     imin = VMAX2(0, (int)ceil((pos[0] - xspan)/hx));
06439     imax = VMIN2(nx-1, (int)floor((pos[0] + xspan)/hx));
06440     for (i=imin; i<=imax; i++) {
06441         x = hx*i;
06442         dx2 = VSQR(pos[0] - x);
06443         if (rtot2 > dx2) {
06444             yspan = VSQRT(rtot2 - dx2) + 2*hy;
06445         } else {
06446             yspan = 2*hy;
06447         }
```

```
06448         jmin = VMAX2(0,(int)ceil((pos[1] - yspan)/hy));
06449         jmax = VMIN2(ny-1,(int)floor((pos[1] + yspan)/hy));
06450         for (j=jmin; j<=jmax; j++) {
06451             y = hy*j;
06452             dy2 = VSQR(pos[1] - y);
06453             if (rtot2 > (dx2+dy2)) {
06454                 zspan = VSQRT(rtot2-dx2-dy2) + 2*hzed;
06455             } else {
06456                 zspan = 2*hzed;
06457             }
06458             kmin = VMAX2(0,(int)ceil((pos[2] - zspan)/hzed));
06459             kmax = VMIN2(nz-1,(int)floor((pos[2] + zspan)/hzed));
06460             for (k=kmin; k<=kmax; k++) {
06461                 z = hzed*k;
06462                 dz2 = VSQR(pos[2] - z);
06463 
06464                 r2 = dx2 + dy2 + dz2;
06465 
06466                 /* We need to determine the inclusion value a000 at (i,j,k) */
06467                 if (r2 < rtot2) a000 = 1.0;
06468                 else a000 = 0.0;
06469 
06470                 /* We need to evaluate the values of x which intersect the
06471                  * sphere and determine if these are in the interval
06472                  * [(i,j,k), (i+1,j,k)] */
06473                 if (r2 < (rtot2 - hx*hx)) a100 = 1.0;
06474                 else if (r2 > (rtot2 + hx*hx)) a100 = 0.0;
06475                 else if (rtot2 > (dy2 + dz2)) {
06476                     dx = VSQRT(rtot2 - dy2 - dz2);
06477                     xm = pos[0] - dx;
06478                     xp = pos[0] + dx;
06479                     if ((xm < x+hx) && (xm > x)) {
06480                         a100 = (xm - x)/hx;
06481                     } else if ((xp < x+hx) && (xp > x)) {
06482                         a100 = (xp - x)/hx;
06483                     }
06484                 } else a100 = 0.0;
06485 
06486                 /* We need to evaluate the values of y which intersect the
06487                  * sphere and determine if these are in the interval
06488                  * [(i,j,k), (i,j+1,k)] */
06489                 if (r2 < (rtot2 - hy*hy)) a010 = 1.0;
06490                 else if (r2 > (rtot2 + hy*hy)) a010 = 0.0;
06491                 else if (rtot2 > (dx2 + dz2)) {
06492                     dy = VSQRT(rtot2 - dx2 - dz2);
06493                     ym = pos[1] - dy;
06494                     yp = pos[1] + dy;
06495                     if ((ym < y+hy) && (ym > y)) {
06496                         a010 = (ym - y)/hy;
06497                     } else if ((yp < y+hy) && (yp > y)) {
06498                         a010 = (yp - y)/hy;
06499                     }
06500                 } else a010 = 0.0;
06501 
06502                 /* We need to evaluate the values of y which intersect the
06503                  * sphere and determine if these are in the interval
06504                  * [(i,j,k), (i,j,k+1)] */
```

```
06505                 if (r2 < (rtot2 - hzed*hzed)) a001 = 1.0;
06506                 else if (r2 > (rtot2 + hzed*hzed)) a001 = 0.0;
06507                 else if (rtot2 > (dx2 + dy2)) {
06508                     dz = VSQRT(rtot2 - dx2 - dy2);
06509                     zm = pos[2] - dz;
06510                     zp = pos[2] + dz;
06511                     if ((zm < z+hzed) && (zm > z)) {
06512                         a001 = (zm - z)/hzed;
06513                     } else if ((zp < z+hzed) && (zp > z)) {
06514                         a001 = (zp - z)/hzed;
06515                     }
06516                 } else a001 = 0.0;
06517
06518                 if (a100 < xarray[IJK(i,j,k)]) xarray[IJK(i,j,k)] = a100;
06519                 if (a010 < yarray[IJK(i,j,k)]) yarray[IJK(i,j,k)] = a010;
06520                 if (a001 < zarray[IJK(i,j,k)]) zarray[IJK(i,j,k)] = a001;
06521
06522             } /* k loop */
06523         } /* j loop */
06524     } /* i loop */
06525 }
06526
06527 /*
06528
06529  NOTE: This is the original version of the markSphere function. It's in here
06530  for reference and in case a reversion to the original code is needed.
06531  D. Gohara (2/14/08)
06532  */
06533 /*
06534 VPRIVATE void markSphere(
06535        double rtot, double *tpos,
06536        int nx, int ny, int nz,
06537        double hx, double hy, double hzed,
06538        double xmin, double ymin, double zmin,
06539        double *array, double markVal) {
06540
06541  int i, j, k, imin, imax, jmin, jmax, kmin, kmax;
06542  double dx, dx2, dy, dy2, dz, dz2;
06543  double rtot2, pos[3];
06544
06545  // Convert to grid reference frame
06546  pos[0] = tpos[0] - xmin;
06547  pos[1] = tpos[1] - ymin;
06548  pos[2] = tpos[2] - zmin;
06549
06550  rtot2 = VSQR(rtot);
06551
06552  dx = rtot + 0.5*hx;
06553  imin = VMAX2(0,(int)ceil((pos[0] - dx)/hx));
06554  imax = VMIN2(nx-1,(int)floor((pos[0] + dx)/hx));
06555  for (i=imin; i<=imax; i++) {
06556   dx2 = VSQR(pos[0] - hx*i);
06557   if (rtot2 > dx2) {
06558    dy = VSQRT(rtot2 - dx2) + 0.5*hy;
06559   } else {
06560    dy = 0.5*hy;
06561   }
```

```
06562   jmin = VMAX2(0,(int)ceil((pos[1] - dy)/hy));
06563   jmax = VMIN2(ny-1,(int)floor((pos[1] + dy)/hy));
06564   for (j=jmin; j<=jmax; j++) {
06565    dy2 = VSQR(pos[1] - hy*j);
06566    if (rtot2 > (dx2+dy2)) {
06567     dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
06568    } else {
06569     dz = 0.5*hzed;
06570    }
06571    kmin = VMAX2(0,(int)ceil((pos[2] - dz)/hzed));
06572    kmax = VMIN2(nz-1,(int)floor((pos[2] + dz)/hzed));
06573    for (k=kmin; k<=kmax; k++) {
06574     dz2 = VSQR(k*hzed - pos[2]);
06575     if ((dz2 + dy2 + dx2) <= rtot2) {
06576      array[IJK(i,j,k)] = markVal;
06577     }
06578    } // k loop
06579   } // j loop
06580  } // i loop
06581 }
06582 */
06583 VPRIVATE void markSphere(double rtot, double *tpos,
06584        int nx, int ny, int nz,
06585        double hx, double hy, double hz,
06586        double xmin, double ymin, double zmin,
06587        double *array, double markVal) {
06588
06589     int i, j, k;
06590  double fi,fj,fk;
06591  int imin, imax;
06592  int jmin, jmax;
06593  int kmin, kmax;
06594     double dx2, dy2, dz2;
06595  double xrange, yrange, zrange;
06596     double rtot2, posx, posy, posz;
06597
06598     /* Convert to grid reference frame */
06599     posx = tpos[0] - xmin;
06600     posy = tpos[1] - ymin;
06601     posz = tpos[2] - zmin;
06602
06603     rtot2 = VSQR(rtot);
06604
06605  xrange = rtot + 0.5 * hx;
06606  yrange = rtot + 0.5 * hy;
06607  zrange = rtot + 0.5 * hz;
06608
06609  imin = VMAX2(0, (int)ceil((posx - xrange)/hx));
06610  jmin = VMAX2(0, (int)ceil((posy - yrange)/hy));
06611  kmin = VMAX2(0, (int)ceil((posz - zrange)/hz));
06612
06613  imax = VMIN2(nx-1, (int)floor((posx + xrange)/hx));
06614  jmax = VMIN2(ny-1, (int)floor((posy + yrange)/hy));
06615  kmax = VMIN2(nz-1, (int)floor((posz + zrange)/hz));
06616
06617  for (i=imin,fi=imin; i<=imax; i++, fi+=1.) {
06618   dx2 = VSQR(posx - hx*fi);
```

```
06619   for (j=jmin,fj=jmin; j<=jmax; j++, fj+=1.) {
06620    dy2 = VSQR(posy - hy*fj);
06621    if((dx2 + dy2) > rtot2) continue;
06622    for (k=kmin, fk=kmin; k<=kmax; k++, fk+=1.) {
06623     dz2 = VSQR(posz - hz*fk);
06624     if ((dz2 + dy2 + dx2) <= rtot2) {
06625      array[IJK(i,j,k)] = markVal;
06626                 }
06627    }
06628   }
06629  }
06630 }
06631
06632 VPRIVATE void zlapSolve(
06633         Vpmg *thee,
06634         double **solution,
06635         double **source,
06636         double **work1
06637         ) {
06638
06639     /* NOTE:  this is an incredibly inefficient algorithm.  The next
06640      * improvement is to focus on only non-zero entries in the source term.
06641      * The best improvement is to use a fast sine transform */
06642
06643     int n, nx, ny, nz, i, j, k, kx, ky, kz;
06644     double hx, hy, hzed, wx, wy, wz, xlen, ylen, zlen;
06645     double phix, phixp1, phixm1, phiy, phiym1, phiyp1, phiz, phizm1, phizp1;
06646     double norm, coef, proj, eigx, eigy, eigz;
06647     double ihx2, ihy2, ihzed2;
06648     double *u, *f, *phi;
06649
06650     /* Snarf grid parameters */
06651     nx = thee->pmgp->nx;
06652     ny = thee->pmgp->ny;
06653     nz = thee->pmgp->nz;
06654     n = nx*ny*nz;
06655     hx = thee->pmgp->hx;
06656     ihx2 = 1.0/hx/hx;
06657     hy = thee->pmgp->hy;
06658     ihy2 = 1.0/hy/hy;
06659     hzed = thee->pmgp->hzed;
06660     ihzed2 = 1.0/hzed/hzed;
06661     xlen = thee->pmgp->xlen;
06662     ylen = thee->pmgp->ylen;
06663     zlen = thee->pmgp->zlen;
06664
06665     /* Set solution and source array pointers */
06666     u = *solution;
06667     f = *source;
06668     phi = *work1;
06669
06670     /* Zero out the solution vector */
06671     for (i=0; i<n; i++) thee->u[i] = 0.0;
06672
06673     /* Iterate through the wavenumbers */
06674     for (kx=1; kx<(nx-1); kx++) {
06675
```

```
06676         wx = (VPI*(double)kx)/((double)nx - 1.0);
06677         eigx = 2.0*ihx2*(1.0 - cos(wx));
06678 
06679         for (ky=1; ky<(ny-1); ky++) {
06680 
06681             wy = (VPI*(double)ky)/((double)ny - 1.0);
06682             eigy = 2.0*ihy2*(1.0 - cos(wy));
06683 
06684             for (kz=1; kz<(nz-1); kz++) {
06685 
06686                 wz = (VPI*(double)kz)/((double)nz - 1.0);
06687                 eigz = 2.0*ihzed2*(1.0 - cos(wz));
06688 
06689                 /* Calculate the basis function.
06690                  * We could calculate each basis function as
06691                  *   phix(i) = sin(wx*i)
06692                  *   phiy(j) = sin(wy*j)
06693                  *   phiz(k) = sin(wz*k)
06694                  * However, this is likely to be very expensive.
06695                  * Therefore, we can use the fact that
06696                  *   phix(i+1) = (2-hx*hx*eigx)*phix(i) - phix(i-1)
06697                  * */
06698                 for (i=1; i<(nx-1); i++) {
06699                     if (i == 1) {
06700                         phix = sin(wx*(double)i);
06701                         phixm1 = 0.0;
06702                     } else {
06703                         phixp1 = (2.0-hx*hx*eigx)*phix - phixm1;
06704                         phixm1 = phix;
06705                         phix = phixp1;
06706                     }
06707                     /* phix = sin(wx*(double)i); */
06708                     for (j=1; j<(ny-1); j++) {
06709                         if (j == 1) {
06710                             phiy = sin(wy*(double)j);
06711                             phiym1 = 0.0;
06712                         } else {
06713                             phiyp1 = (2.0-hy*hy*eigy)*phiy - phiym1;
06714                             phiym1 = phiy;
06715                             phiy = phiyp1;
06716                         }
06717                         /* phiy = sin(wy*(double)j); */
06718                         for (k=1; k<(nz-1); k++) {
06719                             if (k == 1) {
06720                                 phiz = sin(wz*(double)k);
06721                                 phizm1 = 0.0;
06722                             } else {
06723                                 phizp1 = (2.0-hzed*hzed*eigz)*phiz - phizm1;
06724                                 phizm1 = phiz;
06725                                 phiz = phizp1;
06726                             }
06727                             /* phiz = sin(wz*(double)k); */
06728 
06729                             phi[IJK(i,j,k)] = phix*phiy*phiz;
06730 
06731                         }
06732                     }
```

```
06733                 }
06734
06735                 /* Calculate the projection of the source function on this
06736                  * basis function */
06737                 proj = 0.0;
06738                 for (i=1; i<(nx-1); i++) {
06739                     for (j=1; j<(ny-1); j++) {
06740                         for (k=1; k<(nz-1); k++) {
06741
06742                             proj += f[IJK(i,j,k)]*phi[IJK(i,j,k)];
06743
06744                         } /* k loop */
06745                     } /* j loop */
06746                 } /* i loop */
06747
06748                 /* Assemble the coefficient to weight the contribution of this
06749                  * basis function to the solution */
06750                 /* The first contribution is the projection */
06751                 coef = proj;
06752                 /* The second contribution is the eigenvalue */
06753                 coef = coef/(eigx + eigy + eigz);
06754                 /* The third contribution is the normalization factor */
06755                 coef = (8.0/xlen/ylen/zlen)*coef;
06756                 /* The fourth contribution is from scaling the diagonal */
06757                 /* coef = hx*hy*hzed*coef; */
06758
06759                 /* Evaluate the basis function at each grid point */
06760                 for (i=1; i<(nx-1); i++) {
06761                     for (j=1; j<(ny-1); j++) {
06762                         for (k=1; k<(nz-1); k++) {
06763
06764                             u[IJK(i,j,k)] += coef*phi[IJK(i,j,k)];
06765
06766                         } /* k loop */
06767                     } /* j loop */
06768                 } /* i loop */
06769
06770             } /* kz loop */
06771         } /* ky loop */
06772     } /* kx loop */
06773
06774 }
06775
06776 VPUBLIC int Vpmg_solveLaplace(Vpmg *thee) {
06777
06778     int i, j, k, ijk, nx, ny, nz, n, dilo, dihi, djlo, djhi, dklo, dkhi;
06779     double hx, hy, hzed, epsw, iepsw, scal, scalx, scaly, scalz;
06780
06781     nx = thee->pmgp->nx;
06782     ny = thee->pmgp->ny;
06783     nz = thee->pmgp->nz;
06784     n = nx*ny*nz;
06785     hx = thee->pmgp->hx;
06786     hy = thee->pmgp->hy;
06787     hzed = thee->pmgp->hzed;
06788     epsw = Vpbe_getSolventDiel(thee->pbe);
06789     iepsw = 1.0/epsw;
```

```
06790     scal = hx*hy*hzed;
06791     scalx = hx*hy/hzed;
06792     scaly = hx*hzed/hy;
06793     scalz = hx*hy/hzed;
06794
06795     if (!(thee->filled)) {
06796         Vnm_print(2, "Vpmg_solve:  Need to call Vpmg_fillco()!\n");
06797         return 0;
06798     }
06799
06800     /* Load boundary conditions into the RHS array */
06801     for (i=1; i<(nx-1); i++) {
06802
06803         if (i == 1) dilo = 1;
06804         else dilo = 0;
06805         if (i == nx-2) dihi = 1;
06806         else dihi = 0;
06807
06808         for (j=1; j<(ny-1); j++) {
06809
06810             if (j == 1) djlo = 1;
06811             else djlo = 0;
06812             if (j == ny-2) djhi = 1;
06813             else djhi = 0;
06814
06815             for (k=1; k<(nz-1); k++) {
06816
06817                 if (k == 1) dklo = 1;
06818                 else dklo = 0;
06819                 if (k == nz-2) dkhi = 1;
06820                 else dkhi = 0;
06821
06822                 thee->fcf[IJK(i,j,k)] = \
06823                       iepsw*scal*thee->charge[IJK(i,j,k)] \
06824                     + dilo*scalx*thee->gxcf[IJKx(j,k,0)] \
06825                     + dihi*scalx*thee->gxcf[IJKx(j,k,1)] \
06826                     + djlo*scaly*thee->gycf[IJKy(i,k,0)] \
06827                     + djhi*scaly*thee->gycf[IJKy(i,k,1)] \
06828                     + dklo*scalz*thee->gzcf[IJKz(i,j,0)] \
06829                     + dkhi*scalz*thee->gzcf[IJKz(i,j,1)] ;
06830
06831             }
06832         }
06833     }
06834
06835     /* Solve */
06836     zlapSolve( thee, &(thee->u), &(thee->fcf), &(thee->tcf) );
06837
06838     /* Add boundary conditions to solution */
06839     /* i faces */
06840     for (j=0; j<ny; j++) {
06841         for (k=0; k<nz; k++) {
06842             thee->u[IJK(0,j,k)] = thee->gxcf[IJKx(j,k,0)];
06843             thee->u[IJK(nx-1,j,k)] = thee->gycf[IJKx(j,k,1)];
06844         }
06845     }
06846     /* j faces */
```

```
06847     for (i=0; i<nx; i++) {
06848         for (k=0; k<nz; k++) {
06849             thee->u[IJK(i,0,k)] = thee->gycf[IJKy(i,k,0)];
06850             thee->u[IJK(i,ny-1,k)] = thee->gycf[IJKy(i,k,1)];
06851         }
06852     }
06853     /* k faces */
06854     for (i=0; i<nx; i++) {
06855         for (j=0; j<ny; j++) {
06856             thee->u[IJK(i,j,0)] = thee->gzcf[IJKz(i,j,0)];
06857             thee->u[IJK(i,j,nz-1)] = thee->gzcf[IJKz(i,j,1)];
06858         }
06859     }
06860
06861     return 1;
06862
06863 }
06864
06865 VPRIVATE double VFCHI4(int i, double f) {
06866   return (2.5+((double)(i)-(f)));
06867 }
06868
06869 VPRIVATE double bspline4(double x) {
06870
06871     double m, m2;
06872     static double one6 = 1.0/6.0;
06873     static double one8 = 1.0/8.0;
06874     static double one24 = 1.0/24.0;
06875     static double thirteen24 = 13.0/24.0;
06876     static double fourtyseven24 = 47.0/24.0;
06877     static double seventeen24 = 17.0/24.0;
06878
06879     if ((x > 0.0) && (x <= 1.0)){
06880       m = x*x;
06881       return one24*m*m;
06882     } else if ((x > 1.0) && (x <= 2.0)){
06883       m = x - 1.0;
06884       m2 = m*m;
06885       return -one8 + one6*x + m2*(0.25 + one6*m - one6*m2);
06886     } else if ((x > 2.0) && (x <= 3.0)){
06887       m = x - 2.0;
06888       m2 = m*m;
06889       return -thirteen24 + 0.5*x + m2*(-0.25 - 0.5*m + 0.25*m2);
06890     } else if ((x > 3.0) && (x <= 4.0)){
06891       m = x - 3.0;
06892       m2 = m*m;
06893       return fourtyseven24 - 0.5*x + m2*(-0.25 + 0.5*m - one6*m2);
06894     } else if ((x > 4.0) && (x <= 5.0)){
06895       m = x - 4.0;
06896       m2 = m*m;
06897       return seventeen24 - one6*x + m2*(0.25 - one6*m + one24*m2);
06898     } else {
06899       return 0.0;
06900     }
06901 }
06902
06903 VPUBLIC double dbspline4(double x) {
```

```
06904
06905     double m, m2;
06906     static double one6 = 1.0/6.0;
06907     static double one3 = 1.0/3.0;
06908     static double two3 = 2.0/3.0;
06909     static double thirteen6 = 13.0/6.0;
06910
06911     if ((x > 0.0) && (x <= 1.0)){
06912       m2 = x*x;
06913       return one6*x*m2;
06914     } else if ((x > 1.0) && (x <= 2.0)){
06915       m = x - 1.0;
06916       m2 = m*m;
06917       return -one3 + 0.5*x + m2*(0.5 - two3*m);
06918     } else if ((x > 2.0) && (x <= 3.0)){
06919       m = x - 2.0;
06920       m2 = m*m;
06921       return 1.5 - 0.5*x + m2*(-1.5 + m);
06922     } else if ((x > 3.0) && (x <= 4.0)){
06923       m = x - 3.0;
06924       m2 = m*m;
06925       return 1.0 - 0.5*x + m2*(1.5 - two3*m);
06926     } else if ((x > 4.0) && (x <= 5.0)){
06927       m = x - 4.0;
06928       m2 = m*m;
06929       return -thirteen6 + 0.5*x + m2*(-0.5 + one6*m);
06930     } else {
06931       return 0.0;
06932     }
06933 }
06934
06935 VPUBLIC double d2bspline4(double x) {
06936
06937     double m, m2;
06938
06939     if ((x > 0.0) && (x <= 1.0)){
06940       return 0.5*x*x;
06941     } else if ((x > 1.0) && (x <= 2.0)){
06942       m = x - 1.0;
06943       m2 = m*m;
06944       return -0.5 + x - 2.0*m2;
06945     } else if ((x > 2.0) && (x <= 3.0)){
06946       m = x - 2.0;
06947       m2 = m*m;
06948       return 5.5 - 3.0*x + 3.0*m2;
06949     } else if ((x > 3.0) && (x <= 4.0)){
06950       m = x - 3.0;
06951       m2 = m*m;
06952       return -9.5 + 3.0*x - 2.0*m2;
06953     } else if ((x > 4.0) && (x <= 5.0)){
06954       m = x - 4.0;
06955       m2 = m*m;
06956       return 4.5 - x + 0.5*m2;
06957     } else {
06958       return 0.0;
06959     }
06960 }
```

```
06961
06962 VPUBLIC double d3bspline4(double x) {
06963
06964     if      ((x > 0.0) && (x <= 1.0)) return x;
06965     else if ((x > 1.0) && (x <= 2.0)) return 5.0 - 4.0 * x;
06966     else if ((x > 2.0) && (x <= 3.0)) return -15.0 + 6.0 * x;
06967     else if ((x > 3.0) && (x <= 4.0)) return 15.0 - 4.0 * x;
06968     else if ((x > 4.0) && (x <= 5.0)) return x - 5.0;
06969     else                              return 0.0;
06970
06971 }
06972
06973 VPUBLIC void fillcoPermanentMultipole(Vpmg *thee) {
06974
06975     Valist *alist;
06976     Vpbe *pbe;
06977     Vatom *atom;
06978     /* Coversions */
06979     double zmagic, f;
06980     /* Grid */
06981     double xmin, xmax, ymin, ymax, zmin, zmax;
06982     double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;
06983     double hx, hy, hzed, *apos;
06984     /* Multipole */
06985     double charge, *dipole,*quad;
06986     double c,ux,uy,uz,qxx,qyx,qyy,qzx,qzy,qzz,qave;
06987     /* B-spline weights */
06988     double mx,my,mz,dmx,dmy,dmz,d2mx,d2my,d2mz;
06989     double mi,mj,mk;
06990     /* Loop variables */
06991     int i, ii, jj, kk, nx, ny, nz, iatom;
06992     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
06993
06994     /* sanity check */
06995     double mir,mjr,mkr,mr2;
06996     double debye,mc,mux,muy,muz,mqxx,mqyx,mqyy,mqzx,mqzy,mqzz;
06997
06998     VASSERT(thee != VNULL);
06999
07000     /* Get PBE info */
07001     pbe = thee->pbe;
07002     alist = pbe->alist;
07003     zmagic = Vpbe_getZmagic(pbe);
07004
07005     /* Mesh info */
07006     nx = thee->pmgp->nx;
07007     ny = thee->pmgp->ny;
07008     nz = thee->pmgp->nz;
07009     hx = thee->pmgp->hx;
07010     hy = thee->pmgp->hy;
07011     hzed = thee->pmgp->hzed;
07012
07013     /* Conversion */
07014     f = zmagic/(hx*hy*hzed);
07015
07016     /* Define the total domain size */
07017     xlen = thee->pmgp->xlen;
```

```
07018     ylen = thee->pmgp->ylen;
07019     zlen = thee->pmgp->zlen;
07020
07021     /* Define the min/max dimensions */
07022     xmin = thee->pmgp->xcent - (xlen/2.0);
07023     ymin = thee->pmgp->ycent - (ylen/2.0);
07024     zmin = thee->pmgp->zcent - (zlen/2.0);
07025     xmax = thee->pmgp->xcent + (xlen/2.0);
07026     ymax = thee->pmgp->ycent + (ylen/2.0);
07027     zmax = thee->pmgp->zcent + (zlen/2.0);
07028
07029     /* Fill in the source term (permanent atomic multipoles) */
07030     Vnm_print(0, "fillcoPermanentMultipole:  filling in source term.\n");
07031     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
07032
07033         atom = Valist_getAtom(alist, iatom);
07034         apos = Vatom_getPosition(atom);
07035
07036         c = Vatom_getCharge(atom)*f;
07037
07038 #if defined(WITH_TINKER)
07039         dipole = Vatom_getDipole(atom);
07040         ux = dipole[0]/hx*f;
07041         uy = dipole[1]/hy*f;
07042         uz = dipole[2]/hzed*f;
07043         quad = Vatom_getQuadrupole(atom);
07044         qxx = (1.0/3.0)*quad[0]/(hx*hx)*f;
07045         qyx = (2.0/3.0)*quad[3]/(hx*hy)*f;
07046         qyy = (1.0/3.0)*quad[4]/(hy*hy)*f;
07047         qzx = (2.0/3.0)*quad[6]/(hzed*hx)*f;
07048         qzy = (2.0/3.0)*quad[7]/(hzed*hy)*f;
07049         qzz = (1.0/3.0)*quad[8]/(hzed*hzed)*f;
07050 #else
07051         ux = 0.0;
07052         uy = 0.0;
07053         uz = 0.0;
07054         qxx = 0.0;
07055         qyx = 0.0;
07056         qyy = 0.0;
07057         qzx = 0.0;
07058         qzy = 0.0;
07059         qzz = 0.0;
07060 #endif /* if defined(WITH_TINKER) */
07061
07062         /* check
07063         mc = 0.0;
07064         mux = 0.0;
07065         muy = 0.0;
07066         muz = 0.0;
07067         mqxx = 0.0;
07068         mqyx = 0.0;
07069         mqyy = 0.0;
07070         mqzx = 0.0;
07071         mqzy = 0.0;
07072         mqzz = 0.0; */
07073
07074         /* Make sure we're on the grid */
```

```
07075         if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx))  || \
07076             (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy))  || \
07077             (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
07078             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]);

07079             Vnm_print(2, "fillcoPermanentMultipole: xmin = %g, xmax = %g\n", xmin
      , xmax);
07080             Vnm_print(2, "fillcoPermanentMultipole: ymin = %g, ymax = %g\n", ymin
      , ymax);
07081             Vnm_print(2, "fillcoPermanentMultipole: zmin = %g, zmax = %g\n", zmin
      , zmax);
07082             fflush(stderr);
07083         } else {
07084
07085             /* Convert the atom position to grid reference frame */
07086             position[0] = apos[0] - xmin;
07087             position[1] = apos[1] - ymin;
07088             position[2] = apos[2] - zmin;
07089
07090             /* Figure out which vertices we're next to */
07091             ifloat = position[0]/hx;
07092             jfloat = position[1]/hy;
07093             kfloat = position[2]/hzed;
07094
07095             ip1   = (int)ceil(ifloat);
07096             ip2   = ip1 + 2;
07097             im1   = (int)floor(ifloat);
07098             im2   = im1 - 2;
07099             jp1   = (int)ceil(jfloat);
07100             jp2   = jp1 + 2;
07101             jm1   = (int)floor(jfloat);
07102             jm2   = jm1 - 2;
07103             kp1   = (int)ceil(kfloat);
07104             kp2   = kp1 + 2;
07105             km1   = (int)floor(kfloat);
07106             km2   = km1 - 2;
07107
07108             /* This step shouldn't be necessary, but it saves nasty debugging
07109              * later on if something goes wrong */
07110             ip2 = VMIN2(ip2,nx-1);
07111             ip1 = VMIN2(ip1,nx-1);
07112             im1 = VMAX2(im1,0);
07113             im2 = VMAX2(im2,0);
07114             jp2 = VMIN2(jp2,ny-1);
07115             jp1 = VMIN2(jp1,ny-1);
07116             jm1 = VMAX2(jm1,0);
07117             jm2 = VMAX2(jm2,0);
07118             kp2 = VMIN2(kp2,nz-1);
07119             kp1 = VMIN2(kp1,nz-1);
07120             km1 = VMAX2(km1,0);
07121             km2 = VMAX2(km2,0);
07122
07123             /* Now assign fractions of the charge to the nearby verts */
07124             for (ii=im2; ii<=ip2; ii++) {
07125                 mi = VFCHI4(ii,ifloat);
07126                 mx = bspline4(mi);
```

```
07127                 dmx = dbspline4(mi);
07128                 d2mx = d2bspline4(mi);
07129                 for (jj=jm2; jj<=jp2; jj++) {
07130                     mj = VFCHI4(jj,jfloat);
07131                     my = bspline4(mj);
07132                     dmy = dbspline4(mj);
07133                     d2my = d2bspline4(mj);
07134                     for (kk=km2; kk<=kp2; kk++) {
07135                         mk = VFCHI4(kk,kfloat);
07136                         mz = bspline4(mk);
07137                         dmz = dbspline4(mk);
07138                         d2mz = d2bspline4(mk);
07139                         charge = mx*my*mz*c -
07140                          dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz +
07141                          d2mx*my*mz*qxx +
07142                          dmx*dmy*mz*qyx + mx*d2my*mz*qyy +
07143                          dmx*my*dmz*qzx + mx*dmy*dmz*qzy + mx*my*d2mz*qzz;
07144                         thee->charge[IJK(ii,jj,kk)] += charge;
07145 
07146                         /* sanity check - recalculate traceless multipoles
07147                            from the grid charge distribution for this
07148                            site.
07149 
07150                         mir = (mi - 2.5) * hx;
07151                         mjr = (mj - 2.5) * hy;
07152                         mkr = (mk - 2.5) * hzed;
07153                         mr2 = mir*mir+mjr*mjr+mkr*mkr;
07154                         mc += charge;
07155                         mux += mir * charge;
07156                         muy += mjr * charge;
07157                         muz += mkr * charge;
07158                         mqxx += (1.5*mir*mir - 0.5*mr2) * charge;
07159                         mqyx += 1.5*mjr*mir * charge;
07160                         mqyy += (1.5*mjr*mjr - 0.5*mr2) * charge;
07161                         mqzx += 1.5*mkr*mir * charge;
07162                         mqzy += 1.5*mkr*mjr * charge;
07163                         mqzz += (1.5*mkr*mkr - 0.5*mr2) * charge;
07164                          */
07165                     }
07166                 }
07167             }
07168         } /* endif (on the mesh) */
07169 
07170         /* print out the Grid vs. Ideal Point Multipole. */
07171 
07172         /*
07173         debye = 4.8033324;
07174         mc = mc/f;
07175         mux = mux/f*debye;
07176         muy = muy/f*debye;
07177         muz = muz/f*debye;
07178         mqxx = mqxx/f*debye;
07179         mqyy = mqyy/f*debye;
07180         mqzz = mqzz/f*debye;
07181         mqyx = mqyx/f*debye;
07182         mqzx = mqzx/f*debye;
07183         mqzy = mqzy/f*debye;
```

```
07184
07185         printf(" Grid v. Actual Permanent Multipole for Site %i\n",iatom);
07186         printf(" G: %10.6f\n",mc);
07187         printf(" A: %10.6f\n\n",c/f);
07188         printf(" G: %10.6f %10.6f %10.6f\n",mux,muy,muz);
07189         printf(" A: %10.6f %10.6f %10.6f\n\n",
07190                  (ux * hx / f) * debye,
07191                  (uy * hy / f) * debye,
07192                  (uz * hzed /f) * debye);
07193         printf(" G: %10.6f\n",mqxx);
07194         printf(" A: %10.6f\n",quad[0]*debye);
07195         printf(" G: %10.6f %10.6f\n",mqyx,mqyy);
07196         printf(" A: %10.6f %10.6f\n",quad[3]*debye,quad[4]*debye);
07197         printf(" G: %10.6f %10.6f %10.6f\n",mqzx,mqzy,mqzz);
07198         printf(" A: %10.6f %10.6f %10.6f\n\n",
07199                 quad[6]*debye,quad[7]*debye,quad[8]*debye);  */
07200
07201     } /* endfor (each atom) */
07202 }
07203
07204 #if defined(WITH_TINKER)
07205
07206 VPUBLIC void fillcoInducedDipole(Vpmg *thee) {
07207
07208     Valist *alist;
07209     Vpbe *pbe;
07210     Vatom *atom;
07211     /* Conversions */
07212     double zmagic, f;
07213     /* Grid */
07214     double xmin, xmax, ymin, ymax, zmin, zmax;
07215     double xlen, ylen, zlen, ifloat, jfloat, kfloat;
07216     double hx, hy, hzed, *apos, position[3];
07217     /* B-spline weights */
07218     double mx, my, mz, dmx, dmy, dmz;
07219     /* Dipole */
07220     double charge, *dipole, ux,uy,uz;
07221     double mi,mj,mk;
07222     /* Loop indeces */
07223     int i, ii, jj, kk, nx, ny, nz, iatom;
07224     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
07225
07226     double debye;
07227     double mux,muy,muz;
07228     double mir,mjr,mkr;
07229
07230     VASSERT(thee != VNULL);
07231
07232     /* Get PBE info */
07233     pbe = thee->pbe;
07234     alist = pbe->alist;
07235     zmagic = Vpbe_getZmagic(pbe);
07236
07237     /* Mesh info */
07238     nx = thee->pmgp->nx;
07239     ny = thee->pmgp->ny;
07240     nz = thee->pmgp->nz;
```

```
07241     hx = thee->pmgp->hx;
07242     hy = thee->pmgp->hy;
07243     hzed = thee->pmgp->hzed;
07244
07245     /* Conversion */
07246     f = zmagic/(hx*hy*hzed);
07247
07248     /* Define the total domain size */
07249     xlen = thee->pmgp->xlen;
07250     ylen = thee->pmgp->ylen;
07251     zlen = thee->pmgp->zlen;
07252
07253     /* Define the min/max dimensions */
07254     xmin = thee->pmgp->xcent - (xlen/2.0);
07255     ymin = thee->pmgp->ycent - (ylen/2.0);
07256     zmin = thee->pmgp->zcent - (zlen/2.0);
07257     xmax = thee->pmgp->xcent + (xlen/2.0);
07258     ymax = thee->pmgp->ycent + (ylen/2.0);
07259     zmax = thee->pmgp->zcent + (zlen/2.0);
07260
07261     /* Fill in the source term (induced dipoles) */
07262     Vnm_print(0, "fillcoInducedDipole:  filling in the source term.\n");
07263     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
07264
07265         atom = Valist_getAtom(alist, iatom);
07266         apos = Vatom_getPosition(atom);
07267
07268         dipole = Vatom_getInducedDipole(atom);
07269         ux = dipole[0]/hx*f;
07270         uy = dipole[1]/hy*f;
07271         uz = dipole[2]/hzed*f;
07272
07273         mux = 0.0;
07274         muy = 0.0;
07275         muz = 0.0;
07276
07277         /* Make sure we're on the grid */
07278         if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx))  || \
07279             (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy))  || \
07280             (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
07281             Vnm_print(2, "fillcoInducedDipole: Atom #%d at (%4.3f, %4.3f, %4.3f)
      is off the mesh (ignoring this atom):\n", iatom, apos[0], apos[1], apos[2]);
07282             Vnm_print(2, "fillcoInducedDipole: xmin = %g, xmax = %g\n", xmin, xma
      x);
07283             Vnm_print(2, "fillcoInducedDipole: ymin = %g, ymax = %g\n", ymin, yma
      x);
07284             Vnm_print(2, "fillcoInducedDipole: zmin = %g, zmax = %g\n", zmin, zma
      x);
07285             fflush(stderr);
07286         } else {
07287
07288             /* Convert the atom position to grid reference frame */
07289             position[0] = apos[0] - xmin;
07290             position[1] = apos[1] - ymin;
07291             position[2] = apos[2] - zmin;
07292
07293             /* Figure out which vertices we're next to */
```

```
07294             ifloat = position[0]/hx;
07295             jfloat = position[1]/hy;
07296             kfloat = position[2]/hzed;
07297
07298             ip1   = (int)ceil(ifloat);
07299             ip2   = ip1 + 2;
07300             im1   = (int)floor(ifloat);
07301             im2   = im1 - 2;
07302             jp1   = (int)ceil(jfloat);
07303             jp2   = jp1 + 2;
07304             jm1   = (int)floor(jfloat);
07305             jm2   = jm1 - 2;
07306             kp1   = (int)ceil(kfloat);
07307             kp2   = kp1 + 2;
07308             km1   = (int)floor(kfloat);
07309             km2   = km1 - 2;
07310
07311             /* This step shouldn't be necessary, but it saves nasty debugging
07312              * later on if something goes wrong */
07313             ip2 = VMIN2(ip2,nx-1);
07314             ip1 = VMIN2(ip1,nx-1);
07315             im1 = VMAX2(im1,0);
07316             im2 = VMAX2(im2,0);
07317             jp2 = VMIN2(jp2,ny-1);
07318             jp1 = VMIN2(jp1,ny-1);
07319             jm1 = VMAX2(jm1,0);
07320             jm2 = VMAX2(jm2,0);
07321             kp2 = VMIN2(kp2,nz-1);
07322             kp1 = VMIN2(kp1,nz-1);
07323             km1 = VMAX2(km1,0);
07324             km2 = VMAX2(km2,0);
07325
07326             /* Now assign fractions of the dipole to the nearby verts */
07327             for (ii=im2; ii<=ip2; ii++) {
07328                 mi = VFCHI4(ii,ifloat);
07329                 mx = bspline4(mi);
07330                 dmx = dbspline4(mi);
07331                 for (jj=jm2; jj<=jp2; jj++) {
07332                     mj = VFCHI4(jj,jfloat);
07333                     my = bspline4(mj);
07334                     dmy = dbspline4(mj);
07335                     for (kk=km2; kk<=kp2; kk++) {
07336                         mk = VFCHI4(kk,kfloat);
07337                         mz = bspline4(mk);
07338                         dmz = dbspline4(mk);
07339                         charge = -dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz;
07340                         thee->charge[IJK(ii,jj,kk)] += charge;
07341
07342                         /*
07343                         mir = (mi - 2.5) * hx;
07344                         mjr = (mj - 2.5) * hy;
07345                         mkr = (mk - 2.5) * hzed;
07346                         mux += mir * charge;
07347                         muy += mjr * charge;
07348                         muz += mkr * charge;
07349                         */
07350                     }
```

```
07351                 }
07352             }
07353         } /* endif (on the mesh) */
07354 
07355         /* check
07356         debye = 4.8033324;
07357         mux = mux/f*debye;
07358         muy = muy/f*debye;
07359         muz = muz/f*debye;
07360 
07361         printf(" Grid v. Actual Induced Dipole for Site %i\n",iatom);
07362         printf(" G: %10.6f %10.6f %10.6f\n",mux,muy,muz);
07363         printf(" A: %10.6f %10.6f %10.6f\n\n",
07364                  (ux * hx / f) * debye,
07365                  (uy * hy / f) * debye,
07366                  (uz * hzed /f) * debye);
07367          */
07368 
07369     } /* endfor (each atom) */
07370 }
07371 
07372 VPUBLIC void fillcoNLInducedDipole(Vpmg *thee) {
07373 
07374     Valist *alist;
07375     Vpbe *pbe;
07376     Vatom *atom;
07377     /* Conversions */
07378     double zmagic, f;
07379     /* Grid */
07380     double xmin, xmax, ymin, ymax, zmin, zmax;
07381     double xlen, ylen, zlen, ifloat, jfloat, kfloat;
07382     double hx, hy, hzed, *apos, position[3];
07383     /* B-spline weights */
07384     double mx, my, mz, dmx, dmy, dmz;
07385     /* Dipole */
07386     double charge, *dipole, ux,uy,uz;
07387     double mi,mj,mk;
07388     /* Loop indeces */
07389     int i, ii, jj, kk, nx, ny, nz, iatom;
07390     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
07391 
07392     /* sanity check
07393     double debye;
07394     double mux,muy,muz;
07395     double mir,mjr,mkr;
07396      */
07397 
07398     VASSERT(thee != VNULL);
07399 
07400     /* Get PBE info */
07401     pbe = thee->pbe;
07402     alist = pbe->alist;
07403     zmagic = Vpbe_getZmagic(pbe);
07404 
07405     /* Mesh info */
07406     nx = thee->pmgp->nx;
07407     ny = thee->pmgp->ny;
```

```
07408     nz = thee->pmgp->nz;
07409     hx = thee->pmgp->hx;
07410     hy = thee->pmgp->hy;
07411     hzed = thee->pmgp->hzed;
07412
07413     /* Conversion */
07414     f = zmagic/(hx*hy*hzed);
07415
07416     /* Define the total domain size */
07417     xlen = thee->pmgp->xlen;
07418     ylen = thee->pmgp->ylen;
07419     zlen = thee->pmgp->zlen;
07420
07421     /* Define the min/max dimensions */
07422     xmin = thee->pmgp->xcent - (xlen/2.0);
07423     ymin = thee->pmgp->ycent - (ylen/2.0);
07424     zmin = thee->pmgp->zcent - (zlen/2.0);
07425     xmax = thee->pmgp->xcent + (xlen/2.0);
07426     ymax = thee->pmgp->ycent + (ylen/2.0);
07427     zmax = thee->pmgp->zcent + (zlen/2.0);
07428
07429     /* Fill in the source term (non-local induced dipoles) */
07430     Vnm_print(0, "fillcoNLInducedDipole:  filling in source term.\n");
07431     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
07432
07433         atom = Valist_getAtom(alist, iatom);
07434         apos = Vatom_getPosition(atom);
07435
07436         dipole = Vatom_getNLInducedDipole(atom);
07437         ux = dipole[0]/hx*f;
07438         uy = dipole[1]/hy*f;
07439         uz = dipole[2]/hzed*f;
07440
07441         /*
07442         mux = 0.0;
07443         muy = 0.0;
07444         muz = 0.0;
07445          */
07446
07447         /* Make sure we're on the grid */
07448         if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx))  || \
07449             (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy))  || \
07450             (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
07451             Vnm_print(2, "fillcoNLInducedDipole: Atom #%d at (%4.3f, %4.3f,%4.3f)
       is off the mesh (ignoring this atom):\n", iatom, apos[0], apos[1], apos[2]);
07452             Vnm_print(2, "fillcoNLInducedDipole: xmin = %g, xmax = %g\n", xmin, x
      max);
07453             Vnm_print(2, "fillcoNLInducedDipole: ymin = %g, ymax = %g\n", ymin, y
      max);
07454             Vnm_print(2, "fillcoNLInducedDipole: zmin = %g, zmax = %g\n", zmin, z
      max);
07455             fflush(stderr);
07456         } else {
07457
07458             /* Convert the atom position to grid reference frame */
07459             position[0] = apos[0] - xmin;
07460             position[1] = apos[1] - ymin;
```

```
07461             position[2] = apos[2] - zmin;
07462
07463             /* Figure out which vertices we're next to */
07464             ifloat = position[0]/hx;
07465             jfloat = position[1]/hy;
07466             kfloat = position[2]/hzed;
07467
07468             ip1   = (int)ceil(ifloat);
07469             ip2   = ip1 + 2;
07470             im1   = (int)floor(ifloat);
07471             im2   = im1 - 2;
07472             jp1   = (int)ceil(jfloat);
07473             jp2   = jp1 + 2;
07474             jm1   = (int)floor(jfloat);
07475             jm2   = jm1 - 2;
07476             kp1   = (int)ceil(kfloat);
07477             kp2   = kp1 + 2;
07478             km1   = (int)floor(kfloat);
07479             km2   = km1 - 2;
07480
07481             /* This step shouldn't be necessary, but it saves nasty debugging
07482              * later on if something goes wrong */
07483             ip2 = VMIN2(ip2,nx-1);
07484             ip1 = VMIN2(ip1,nx-1);
07485             im1 = VMAX2(im1,0);
07486             im2 = VMAX2(im2,0);
07487             jp2 = VMIN2(jp2,ny-1);
07488             jp1 = VMIN2(jp1,ny-1);
07489             jm1 = VMAX2(jm1,0);
07490             jm2 = VMAX2(jm2,0);
07491             kp2 = VMIN2(kp2,nz-1);
07492             kp1 = VMIN2(kp1,nz-1);
07493             km1 = VMAX2(km1,0);
07494             km2 = VMAX2(km2,0);
07495
07496             /* Now assign fractions of the non local induced dipole
07497                to the nearby verts */
07498             for (ii=im2; ii<=ip2; ii++) {
07499                 mi = VFCHI4(ii,ifloat);
07500                 mx = bspline4(mi);
07501                 dmx = dbspline4(mi);
07502                 for (jj=jm2; jj<=jp2; jj++) {
07503                     mj = VFCHI4(jj,jfloat);
07504                     my = bspline4(mj);
07505                     dmy = dbspline4(mj);
07506                     for (kk=km2; kk<=kp2; kk++) {
07507                         mk = VFCHI4(kk,kfloat);
07508                         mz = bspline4(mk);
07509                         dmz = dbspline4(mk);
07510                         charge = -dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz;
07511                         thee->charge[IJK(ii,jj,kk)] += charge;
07512
07513                         /*
07514                         mir = (mi - 2.5) * hx;
07515                         mjr = (mj - 2.5) * hy;
07516                         mkr = (mk - 2.5) * hzed;
07517                         mux += mir * charge;
```

```
07518                         muy += mjr * charge;
07519                         muz += mkr * charge;
07520                          */
07521                     }
07522                 }
07523             }
07524         } /* endif (on the mesh) */
07525 
07526         /*
07527         debye = 4.8033324;
07528         mux = mux/f*debye;
07529         muy = muy/f*debye;
07530         muz = muz/f*debye;
07531 
07532         printf(" Grid v. Actual Non-Local Induced Dipole for Site %i\n",iatom);
07533         printf(" G: %10.6f %10.6f %10.6f\n",mux,muy,muz);
07534         printf(" A: %10.6f %10.6f %10.6f\n\n",
07535                  (ux * hx / f) * debye,
07536                  (uy * hy / f) * debye,
07537                  (uz * hzed /f) * debye); */
07538 
07539     } /* endfor (each atom) */
07540 }
07541 
07542 VPUBLIC double Vpmg_qfPermanentMultipoleEnergy(Vpmg *thee, int atomID) {
07543 
07544     double *u;
07545     Vatom *atom;
07546     /* Grid variables */
07547     int nx, ny, nz;
07548     double xmax, xmin, ymax, ymin, zmax, zmin;
07549     double hx, hy, hzed, ifloat, jfloat, kfloat;
07550     double mi, mj, mk;
07551     double *position;
07552     /* B-spline weights */
07553     double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz;
07554     /* Loop indeces */
07555     int ip1,ip2,im1,im2,jp1,jp2,jm1,jm2,kp1,kp2,km1,km2;
07556     int i,j,ii,jj,kk;
07557     /* Potential, field, field gradient and multipole components */
07558     double pot, rfe[3], rfde[3][3], energy;
07559     double f, charge, *dipole, *quad;
07560     double qxx, qyx, qyy, qzx, qzy, qzz;
07561 
07562 
07563     VASSERT(thee != VNULL);
07564     VASSERT(thee->filled);
07565 
07566     /* Get the mesh information */
07567     nx = thee->pmgp->nx;
07568     ny = thee->pmgp->ny;
07569     nz = thee->pmgp->nz;
07570     hx = thee->pmgp->hx;
07571     hy = thee->pmgp->hy;
07572     hzed = thee->pmgp->hzed;
07573     xmax = thee->xf[nx-1];
07574     ymax = thee->yf[ny-1];
```

```
07575     zmax = thee->zf[nz-1];
07576     xmin = thee->xf[0];
07577     ymin = thee->yf[0];
07578     zmin = thee->zf[0];
07579
07580     u = thee->u;
07581
07582     atom = Valist_getAtom(thee->pbe->alist, atomID);
07583
07584     /* Currently all atoms must be in the same partition. */
07585
07586     VASSERT(atom->partID != 0);
07587
07588     /* Convert the atom position to grid coordinates */
07589
07590     position = Vatom_getPosition(atom);
07591     ifloat = (position[0] - xmin)/hx;
07592     jfloat = (position[1] - ymin)/hy;
07593     kfloat = (position[2] - zmin)/hzed;
07594     ip1 = (int)ceil(ifloat);
07595     ip2 = ip1 + 2;
07596     im1 = (int)floor(ifloat);
07597     im2 = im1 - 2;
07598     jp1 = (int)ceil(jfloat);
07599     jp2 = jp1 + 2;
07600     jm1 = (int)floor(jfloat);
07601     jm2 = jm1 - 2;
07602     kp1 = (int)ceil(kfloat);
07603     kp2 = kp1 + 2;
07604     km1 = (int)floor(kfloat);
07605     km2 = km1 - 2;
07606
07607     /* This step shouldn't be necessary, but it saves nasty debugging
07608      * later on if something goes wrong */
07609     ip2 = VMIN2(ip2,nx-1);
07610     ip1 = VMIN2(ip1,nx-1);
07611     im1 = VMAX2(im1,0);
07612     im2 = VMAX2(im2,0);
07613     jp2 = VMIN2(jp2,ny-1);
07614     jp1 = VMIN2(jp1,ny-1);
07615     jm1 = VMAX2(jm1,0);
07616     jm2 = VMAX2(jm2,0);
07617     kp2 = VMIN2(kp2,nz-1);
07618     kp1 = VMIN2(kp1,nz-1);
07619     km1 = VMAX2(km1,0);
07620     km2 = VMAX2(km2,0);
07621
07622     /* Initialize observables to zero */
07623     energy = 0.0;
07624     pot = 0.0;
07625     for (i=0;i<3;i++){
07626        rfe[i] = 0.0;
07627        for (j=0;j<3;j++){
07628          rfde[i][j] = 0.0;
07629        }
07630     }
07631
```

```
07632     for (ii=im2; ii<=ip2; ii++) {
07633       mi = VFCHI4(ii,ifloat);
07634       mx = bspline4(mi);
07635       dmx = dbspline4(mi);
07636       d2mx = d2bspline4(mi);
07637       for (jj=jm2; jj<=jp2; jj++) {
07638         mj = VFCHI4(jj,jfloat);
07639         my = bspline4(mj);
07640         dmy = dbspline4(mj);
07641         d2my = d2bspline4(mj);
07642         for (kk=km2; kk<=kp2; kk++) {
07643           mk = VFCHI4(kk,kfloat);
07644           mz = bspline4(mk);
07645           dmz = dbspline4(mk);
07646           d2mz = d2bspline4(mk);
07647           f = u[IJK(ii,jj,kk)];
07648           /* potential */
07649           pot  += f*mx*my*mz;
07650           /* field */
07651           rfe[0] += f*dmx*my*mz/hx;
07652           rfe[1] += f*mx*dmy*mz/hy;
07653           rfe[2] += f*mx*my*dmz/hzed;
07654           /* field gradient */
07655           rfde[0][0] += f*d2mx*my*mz/(hx*hx);
07656           rfde[1][0] += f*dmx*dmy*mz/(hy*hx);
07657           rfde[1][1] += f*mx*d2my*mz/(hy*hy);
07658           rfde[2][0] += f*dmx*my*dmz/(hx*hzed);
07659           rfde[2][1] += f*mx*dmy*dmz/(hy*hzed);
07660           rfde[2][2] += f*mx*my*d2mz/(hzed*hzed);
07661         }
07662       }
07663     }
07664
07665     charge = Vatom_getCharge(atom);
07666     dipole = Vatom_getDipole(atom);
07667     quad = Vatom_getQuadrupole(atom);
07668     qxx = quad[0]/3.0;
07669     qyx = quad[3]/3.0;
07670     qyy = quad[4]/3.0;
07671     qzx = quad[6]/3.0;
07672     qzy = quad[7]/3.0;
07673     qzz = quad[8]/3.0;
07674
07675     energy =   pot * charge
07676              - rfe[0] * dipole[0]
07677              - rfe[1] * dipole[1]
07678              - rfe[2] * dipole[2]
07679              +     rfde[0][0]*qxx
07680              + 2.0*rfde[1][0]*qyx +     rfde[1][1]*qyy
07681              + 2.0*rfde[2][0]*qzx + 2.0*rfde[2][1]*qzy + rfde[2][2]*qzz;
07682
07683     return energy;
07684 }
07685
07686 VPUBLIC void Vpmg_fieldSpline4(Vpmg *thee, int atomID, double field[3]) {
07687
07688     Vatom *atom;
```

```
07689     double *u, f;
07690     /* Grid variables */
07691     int nx, ny, nz;
07692     double xmax, xmin, ymax, ymin, zmax, zmin;
07693     double hx, hy, hzed, ifloat, jfloat, kfloat;
07694     double *apos, position[3];
07695     /* B-Spline weights */
07696     double mx, my, mz, dmx, dmy, dmz;
07697     double mi, mj, mk;
07698     /* Loop indeces */
07699     int ip1,ip2,im1,im2,jp1,jp2,jm1,jm2,kp1,kp2,km1,km2;
07700     int i,j,ii,jj,kk;
07701
07702
07703     VASSERT (thee != VNULL);
07704
07705     /* Get the mesh information */
07706     nx = thee->pmgp->nx;
07707     ny = thee->pmgp->ny;
07708     nz = thee->pmgp->nz;
07709     hx = thee->pmgp->hx;
07710     hy = thee->pmgp->hy;
07711     hzed = thee->pmgp->hzed;
07712     xmax = thee->xf[nx-1];
07713     ymax = thee->yf[ny-1];
07714     zmax = thee->zf[nz-1];
07715     xmin = thee->xf[0];
07716     ymin = thee->yf[0];
07717     zmin = thee->zf[0];
07718
07719     u = thee->u;
07720
07721     atom = Valist_getAtom(thee->pbe->alist, atomID);
07722
07723     /* Currently all atoms must be in the same partition. */
07724
07725     VASSERT (atom->partID != 0);
07726
07727     /* Convert the atom position to grid coordinates */
07728
07729     apos = Vatom_getPosition(atom);
07730     position[0] = apos[0] - xmin;
07731     position[1] = apos[1] - ymin;
07732     position[2] = apos[2] - zmin;
07733     ifloat = position[0]/hx;
07734     jfloat = position[1]/hy;
07735     kfloat = position[2]/hzed;
07736     ip1 = (int)ceil(ifloat);
07737     ip2 = ip1 + 2;
07738     im1 = (int)floor(ifloat);
07739     im2 = im1 - 2;
07740     jp1 = (int)ceil(jfloat);
07741     jp2 = jp1 + 2;
07742     jm1 = (int)floor(jfloat);
07743     jm2 = jm1 - 2;
07744     kp1 = (int)ceil(kfloat);
07745     kp2 = kp1 + 2;
```

```
07746     km1 = (int)floor(kfloat);
07747     km2 = km1 - 2;
07748
07749     /* This step shouldn't be necessary, but it saves nasty debugging
07750      * later on if something goes wrong */
07751     ip2 = VMIN2(ip2,nx-1);
07752     ip1 = VMIN2(ip1,nx-1);
07753     im1 = VMAX2(im1,0);
07754     im2 = VMAX2(im2,0);
07755     jp2 = VMIN2(jp2,ny-1);
07756     jp1 = VMIN2(jp1,ny-1);
07757     jm1 = VMAX2(jm1,0);
07758     jm2 = VMAX2(jm2,0);
07759     kp2 = VMIN2(kp2,nz-1);
07760     kp1 = VMIN2(kp1,nz-1);
07761     km1 = VMAX2(km1,0);
07762     km2 = VMAX2(km2,0);
07763
07764     for (i=0;i<3;i++){
07765        field[i] = 0.0;
07766     }
07767
07768     for (ii=im2; ii<=ip2; ii++) {
07769       mi = VFCHI4(ii,ifloat);
07770       mx = bspline4(mi);
07771       dmx = dbspline4(mi);
07772       for (jj=jm2; jj<=jp2; jj++) {
07773         mj = VFCHI4(jj,jfloat);
07774         my = bspline4(mj);
07775         dmy = dbspline4(mj);
07776         for (kk=km2; kk<=kp2; kk++) {
07777           mk = VFCHI4(kk,kfloat);
07778           mz = bspline4(mk);
07779           dmz = dbspline4(mk);
07780           f = u[IJK(ii,jj,kk)];
07781
07782           field[0] += f*dmx*my*mz/hx;
07783           field[1] += f*mx*dmy*mz/hy;
07784           field[2] += f*mx*my*dmz/hzed;
07785         }
07786       }
07787     }
07788 }
07789
07790 VPUBLIC void Vpmg_qfPermanentMultipoleForce(Vpmg *thee, int atomID,
07791                                             double force[3], double torque[3]) {
07792
07793     Vatom *atom;
07794     double f, *u, *apos, position[3];
07795
07796     /* Grid variables */
07797     int nx,ny,nz;
07798     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
07799     double hx, hy, hzed, ifloat, jfloat, kfloat;
07800
07801     /* B-spline weights */
07802     double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz, d3mx, d3my, d3mz;
```

```
07803     double mi, mj, mk;
07804
07805     /* Loop indeces */
07806     int i, j, k, ii, jj, kk;
07807     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
07808
07809     /* Potential, field, field gradient and 2nd field gradient */
07810     double pot, e[3], de[3][3], d2e[3][3][3];
07811
07812     /* Permanent multipole components */
07813     double *dipole, *quad;
07814     double c, ux, uy, uz, qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
07815
07816     VASSERT(thee != VNULL);
07817     VASSERT(thee->filled);
07818
07819     atom = Valist_getAtom(thee->pbe->alist, atomID);
07820
07821     /* Currently all atoms must be in the same partition. */
07822
07823     VASSERT(atom->partID != 0);
07824
07825     apos = Vatom_getPosition(atom);
07826
07827     c = Vatom_getCharge(atom);
07828     dipole = Vatom_getDipole(atom);
07829     ux = dipole[0];
07830     uy = dipole[1];
07831     uz = dipole[2];
07832     quad = Vatom_getQuadrupole(atom);
07833     qxx = quad[0]/3.0;
07834     qxy = quad[1]/3.0;
07835     qxz = quad[2]/3.0;
07836     qyx = quad[3]/3.0;
07837     qyy = quad[4]/3.0;
07838     qyz = quad[5]/3.0;
07839     qzx = quad[6]/3.0;
07840     qzy = quad[7]/3.0;
07841     qzz = quad[8]/3.0;
07842
07843     /* Initialize observables */
07844     pot = 0.0;
07845     for (i=0;i<3;i++){
07846        e[i] = 0.0;
07847        for (j=0;j<3;j++){
07848           de[i][j] = 0.0;
07849           for (k=0;k<3;k++){
07850              d2e[i][j][k] = 0.0;
07851           }
07852        }
07853     }
07854
07855     /* Mesh info */
07856     nx = thee->pmgp->nx;
07857     ny = thee->pmgp->ny;
07858     nz = thee->pmgp->nz;
07859     hx = thee->pmgp->hx;
```

```
07860     hy = thee->pmgp->hy;
07861     hzed = thee->pmgp->hzed;
07862     xlen = thee->pmgp->xlen;
07863     ylen = thee->pmgp->ylen;
07864     zlen = thee->pmgp->zlen;
07865     xmin = thee->pmgp->xmin;
07866     ymin = thee->pmgp->ymin;
07867     zmin = thee->pmgp->zmin;
07868     xmax = thee->pmgp->xmax;
07869     ymax = thee->pmgp->ymax;
07870     zmax = thee->pmgp->zmax;
07871     u = thee->u;
07872
07873     /* Make sure we're on the grid */
07874     if ((apos[0]<=(xmin+2*hx))   || (apos[0]>=(xmax-2*hx)) \
07875      || (apos[1]<=(ymin+2*hy))   || (apos[1]>=(ymax-2*hy)) \
07876      || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
07877         Vnm_print(2, "qfPermanentMultipoleForce:  Atom off the mesh (ignoring) %6
      .3f %6.3f %6.3f\n", apos[0], apos[1], apos[2]);
07878         fflush(stderr);
07879     } else {
07880
07881         /* Convert the atom position to grid coordinates */
07882         position[0] = apos[0] - xmin;
07883         position[1] = apos[1] - ymin;
07884         position[2] = apos[2] - zmin;
07885         ifloat = position[0]/hx;
07886         jfloat = position[1]/hy;
07887         kfloat = position[2]/hzed;
07888         ip1 = (int)ceil(ifloat);
07889         ip2 = ip1 + 2;
07890         im1 = (int)floor(ifloat);
07891         im2 = im1 - 2;
07892         jp1 = (int)ceil(jfloat);
07893         jp2 = jp1 + 2;
07894         jm1 = (int)floor(jfloat);
07895         jm2 = jm1 - 2;
07896         kp1 = (int)ceil(kfloat);
07897         kp2 = kp1 + 2;
07898         km1 = (int)floor(kfloat);
07899         km2 = km1 - 2;
07900
07901         /* This step shouldn't be necessary, but it saves nasty debugging
07902          * later on if something goes wrong */
07903         ip2 = VMIN2(ip2,nx-1);
07904         ip1 = VMIN2(ip1,nx-1);
07905         im1 = VMAX2(im1,0);
07906         im2 = VMAX2(im2,0);
07907         jp2 = VMIN2(jp2,ny-1);
07908         jp1 = VMIN2(jp1,ny-1);
07909         jm1 = VMAX2(jm1,0);
07910         jm2 = VMAX2(jm2,0);
07911         kp2 = VMIN2(kp2,nz-1);
07912         kp1 = VMIN2(kp1,nz-1);
07913         km1 = VMAX2(km1,0);
07914         km2 = VMAX2(km2,0);
07915
```

```
07916         for (ii=im2; ii<=ip2; ii++) {
07917             mi = VFCHI4(ii,ifloat);
07918             mx = bspline4(mi);
07919             dmx = dbspline4(mi);
07920             d2mx = d2bspline4(mi);
07921             d3mx = d3bspline4(mi);
07922             for (jj=jm2; jj<=jp2; jj++) {
07923                 mj = VFCHI4(jj,jfloat);
07924                 my = bspline4(mj);
07925                 dmy = dbspline4(mj);
07926                 d2my = d2bspline4(mj);
07927                 d3my = d3bspline4(mj);
07928                 for (kk=km2; kk<=kp2; kk++) {
07929                     mk = VFCHI4(kk,kfloat);
07930                     mz = bspline4(mk);
07931                     dmz = dbspline4(mk);
07932                     d2mz = d2bspline4(mk);
07933                     d3mz = d3bspline4(mk);
07934                     f = u[IJK(ii,jj,kk)];
07935                     /* Potential */
07936                     pot  += f*mx*my*mz;
07937                     /* Field */
07938                     e[0] += f*dmx*my*mz/hx;
07939                     e[1] += f*mx*dmy*mz/hy;
07940                     e[2] += f*mx*my*dmz/hzed;
07941                     /* Field gradient */
07942                     de[0][0] += f*d2mx*my*mz/(hx*hx);
07943                     de[1][0] += f*dmx*dmy*mz/(hy*hx);
07944                     de[1][1] += f*mx*d2my*mz/(hy*hy);
07945                     de[2][0] += f*dmx*my*dmz/(hx*hzed);
07946                     de[2][1] += f*mx*dmy*dmz/(hy*hzed);
07947                     de[2][2] += f*mx*my*d2mz/(hzed*hzed);
07948                     /* 2nd Field Gradient
07949                        VxVxVa */
07950                     d2e[0][0][0] += f*d3mx*my*mz /(hx*hx*hx);
07951                     d2e[0][0][1] += f*d2mx*dmy*mz/(hx*hy*hx);
07952                     d2e[0][0][2] += f*d2mx*my*dmz/(hx*hx*hzed);
07953                     /* VyVxVa */
07954                     d2e[1][0][0] += f*d2mx*dmy*mz/(hx*hx*hy);
07955                     d2e[1][0][1] += f*dmx*d2my*mz/(hx*hy*hy);
07956                     d2e[1][0][2] += f*dmx*dmy*dmz/(hx*hy*hzed);
07957                     /* VyVyVa */
07958                     d2e[1][1][0] += f*dmx*d2my*mz/(hx*hy*hy);
07959                     d2e[1][1][1] += f*mx*d3my*mz /(hy*hy*hy);
07960                     d2e[1][1][2] += f*mx*d2my*dmz/(hy*hy*hzed);
07961                     /* VzVxVa */
07962                     d2e[2][0][0] += f*d2mx*my*dmz/(hx*hx*hzed);
07963                     d2e[2][0][1] += f*dmx*dmy*dmz/(hx*hy*hzed);
07964                     d2e[2][0][2] += f*dmx*my*d2mz/(hx*hzed*hzed);
07965                     /* VzVyVa */
07966                     d2e[2][1][0] += f*dmx*dmy*dmz/(hx*hy*hzed);
07967                     d2e[2][1][1] += f*mx*d2my*dmz/(hy*hy*hzed);
07968                     d2e[2][1][2] += f*mx*dmy*d2mz/(hy*hzed*hzed);
07969                     /* VzVzVa */
07970                     d2e[2][2][0] += f*dmx*my*d2mz/(hx*hzed*hzed);
07971                     d2e[2][2][1] += f*mx*dmy*d2mz/(hy*hzed*hzed);
07972                     d2e[2][2][2] += f*mx*my*d3mz /(hzed*hzed*hzed);
```

```
07973                 }
07974             }
07975         }
07976     }
07977
07978     /* Monopole Force */
07979     force[0] = e[0]*c;
07980     force[1] = e[1]*c;
07981     force[2] = e[2]*c;
07982
07983     /* Dipole Force */
07984     force[0] -= de[0][0]*ux+de[1][0]*uy+de[2][0]*uz;
07985     force[1] -= de[1][0]*ux+de[1][1]*uy+de[2][1]*uz;
07986     force[2] -= de[2][0]*ux+de[2][1]*uy+de[2][2]*uz;
07987
07988     /* Quadrupole Force */
07989     force[0] += d2e[0][0][0]*qxx
07990              +  d2e[1][0][0]*qyx*2.0+d2e[1][1][0]*qyy
07991              +  d2e[2][0][0]*qzx*2.0+d2e[2][1][0]*qzy*2.0+d2e[2][2][0]*qzz;
07992     force[1] += d2e[0][0][1]*qxx
07993              +  d2e[1][0][1]*qyx*2.0+d2e[1][1][1]*qyy
07994              +  d2e[2][0][1]*qzx*2.0+d2e[2][1][1]*qzy*2.0+d2e[2][2][1]*qzz;
07995     force[2] += d2e[0][0][2]*qxx
07996              +  d2e[1][0][2]*qyx*2.0+d2e[1][1][2]*qyy
07997              +  d2e[2][0][2]*qzx*2.0+d2e[2][1][2]*qzy*2.0+d2e[2][2][2]*qzz;
07998
07999     /* Dipole Torque */
08000     torque[0] = uy * e[2] - uz * e[1];
08001     torque[1] = uz * e[0] - ux * e[2];
08002     torque[2] = ux * e[1] - uy * e[0];
08003     /* Quadrupole Torque */
08004     de[0][1] = de[1][0];
08005     de[0][2] = de[2][0];
08006     de[1][2] = de[2][1];
08007     torque[0] -= 2.0*(qyx*de[0][2] + qyy*de[1][2] + qyz*de[2][2]
08008                     - qzx*de[0][1] - qzy*de[1][1] - qzz*de[2][1]);
08009     torque[1] -= 2.0*(qzx*de[0][0] + qzy*de[1][0] + qzz*de[2][0]
08010                     - qxx*de[0][2] - qxy*de[1][2] - qxz*de[2][2]);
08011     torque[2] -= 2.0*(qxx*de[0][1] + qxy*de[1][1] + qxz*de[2][1]
08012                     - qyx*de[0][0] - qyy*de[1][0] - qyz*de[2][0]);
08013
08014
08015     /* printf(" qPhi Force %f %f %f\n", force[0], force[1], force[2]);
08016        printf(" qPhi Torque %f %f %f\n", torque[0], torque[1], torque[2]); */
08017 }
08018
08019 VPUBLIC void Vpmg_ibPermanentMultipoleForce(Vpmg *thee, int atomID,
08020                                             double force[3]) {
08021
08022     Valist *alist;
08023     Vacc *acc;
08024     Vpbe *pbe;
08025     Vatom *atom;
08026     Vsurf_Meth srfm;
08027
08028     /* Grid variables */
08029     double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
```

```
08030     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;
08031     double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
08032     double izmagic;
08033     int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
08034
08035     VASSERT(thee != VNULL);
08036
08037     /* Nonlinear PBE is not implemented for AMOEBA */
08038     VASSERT(!thee->pmgp->nonlin);
08039
08040     acc = thee->pbe->acc;
08041     srfm = thee->surfMeth;
08042     atom = Valist_getAtom(thee->pbe->alist, atomID);
08043
08044     /* Currently all atoms must be in the same partition. */
08045
08046     VASSERT(atom->partID != 0);
08047     apos = Vatom_getPosition(atom);
08048     arad = Vatom_getRadius(atom);
08049
08050     /* Reset force */
08051     force[0] = 0.0;
08052     force[1] = 0.0;
08053     force[2] = 0.0;
08054
08055     /* Get PBE info */
08056     pbe = thee->pbe;
08057     acc = pbe->acc;
08058     alist = pbe->alist;
08059     irad = Vpbe_getMaxIonRadius(pbe);
08060     zkappa2 = Vpbe_getZkappa2(pbe);
08061     izmagic = 1.0/Vpbe_getZmagic(pbe);
08062
08063     /* Should be a check for this further up. */
08064     VASSERT (zkappa2 > VPMGSMALL);
08065
08066     /* Mesh info */
08067     nx = thee->pmgp->nx;
08068     ny = thee->pmgp->ny;
08069     nz = thee->pmgp->nz;
08070     hx = thee->pmgp->hx;
08071     hy = thee->pmgp->hy;
08072     hzed = thee->pmgp->hzed;
08073     xlen = thee->pmgp->xlen;
08074     ylen = thee->pmgp->ylen;
08075     zlen = thee->pmgp->zlen;
08076     xmin = thee->pmgp->xmin;
08077     ymin = thee->pmgp->ymin;
08078     zmin = thee->pmgp->zmin;
08079     xmax = thee->pmgp->xmax;
08080     ymax = thee->pmgp->ymax;
08081     zmax = thee->pmgp->zmax;
08082
08083     /* Make sure we're on the grid */
08084     if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
08085       (apos[1]<=ymin) || (apos[1]>=ymax)  || \
08086       (apos[2]<=zmin) || (apos[2]>=zmax)) {
```

```
08087         Vnm_print(2, "ibPermanentMultipoleForce:  Atom %d at (%4.3f, %4.3f, %4.3f
      ) is off the mesh (ignoring):\n", atomID, apos[0], apos[1], apos[2]);
08088         Vnm_print(2, "ibPermanentMultipoleForce:  xmin = %g, xmax = %g\n", xmin,
      xmax);
08089         Vnm_print(2, "ibPermanentMultipoleForce:  ymin = %g, ymax = %g\n", ymin,
      ymax);
08090         Vnm_print(2, "ibPermanentMultipoleForce:  zmin = %g, zmax = %g\n", zmin,
      zmax);
08091         fflush(stderr);
08092     } else {
08093 
08094         /* Convert the atom position to grid reference frame */
08095         position[0] = apos[0] - xmin;
08096         position[1] = apos[1] - ymin;
08097         position[2] = apos[2] - zmin;
08098 
08099         /* Integrate over points within this atom's (inflated) radius */
08100         rtot = (irad + arad + thee->splineWin);
08101         rtot2 = VSQR(rtot);
08102         dx = rtot + 0.5*hx;
08103         imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
08104         imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
08105         for (i=imin; i<=imax; i++) {
08106             dx2 = VSQR(position[0] - hx*i);
08107             if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
08108             else dy = 0.5*hy;
08109             jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
08110             jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
08111             for (j=jmin; j<=jmax; j++) {
08112                 dy2 = VSQR(position[1] - hy*j);
08113                 if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
08114                 else dz = 0.5*hzed;
08115                 kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
08116                 kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
08117                 for (k=kmin; k<=kmax; k++) {
08118                     dz2 = VSQR(k*hzed - position[2]);
08119                     /* See if grid point is inside ivdw radius and set ccf
08120                      * accordingly (do spline assignment here) */
08121                     if ((dz2 + dy2 + dx2) <= rtot2) {
08122                         gpos[0] = i*hx + xmin;
08123                         gpos[1] = j*hy + ymin;
08124                         gpos[2] = k*hzed + zmin;
08125                         Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, irad,
       atom, tgrad);
08126                         fmag = VSQR(thee->u[IJK(i,j,k)])*thee->kappa[IJK(i,j,k)];
08127                         force[0] += (zkappa2*fmag*tgrad[0]);
08128                         force[1] += (zkappa2*fmag*tgrad[1]);
08129                         force[2] += (zkappa2*fmag*tgrad[2]);
08130                     }
08131                 } /* k loop */
08132             } /* j loop */
08133         } /* i loop */
08134     }
08135 
08136     force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
08137     force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
```

```
08138     force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
08139
08140 }
08141
08142 VPUBLIC void Vpmg_dbPermanentMultipoleForce(Vpmg *thee, int atomID,
08143                                             double force[3]) {
08144
08145     Vacc *acc;
08146     Vpbe *pbe;
08147     Vatom *atom;
08148     Vsurf_Meth srfm;
08149
08150     double *apos, position[3], arad, hx, hy, hzed, izmagic, deps, depsi;
08151     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2, epsp;
08152     double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
08153     double *u, Hxijk, Hyijk, Hzijk, Hxim1jk, Hyijm1k, Hzijkm1;
08154     double dHxijk[3], dHyijk[3], dHzijk[3], dHxim1jk[3], dHyijm1k[3];
08155     double dHzijkm1[3];
08156     int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
08157
08158     VASSERT(thee != VNULL);
08159
08160     acc = thee->pbe->acc;
08161     srfm = thee->surfMeth;
08162     atom = Valist_getAtom(thee->pbe->alist, atomID);
08163
08164     /* Currently all atoms must be in the same partition. */
08165
08166     VASSERT(atom->partID != 0);
08167     arad = Vatom_getRadius(atom);
08168     apos = Vatom_getPosition(atom);
08169
08170     /* Reset force */
08171     force[0] = 0.0;
08172     force[1] = 0.0;
08173     force[2] = 0.0;
08174
08175     /* Get PBE info */
08176     pbe = thee->pbe;
08177     acc = pbe->acc;
08178     epsp = Vpbe_getSoluteDiel(pbe);
08179     epsw = Vpbe_getSolventDiel(pbe);
08180     kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*Vunit_kb;
08181     izmagic = 1.0/Vpbe_getZmagic(pbe);
08182
08183
08184     deps = (epsw - epsp);
08185     depsi = 1.0/deps;
08186
08187     VASSERT(VABS(deps) > VPMGSMALL);
08188
08189     /* Mesh info */
08190     nx = thee->pmgp->nx;
08191     ny = thee->pmgp->ny;
08192     nz = thee->pmgp->nz;
08193     hx = thee->pmgp->hx;
08194     hy = thee->pmgp->hy;
```

```
08195     hzed = thee->pmgp->hzed;
08196     xlen = thee->pmgp->xlen;
08197     ylen = thee->pmgp->ylen;
08198     zlen = thee->pmgp->zlen;
08199     xmin = thee->pmgp->xmin;
08200     ymin = thee->pmgp->ymin;
08201     zmin = thee->pmgp->zmin;
08202     xmax = thee->pmgp->xmax;
08203     ymax = thee->pmgp->ymax;
08204     zmax = thee->pmgp->zmax;
08205     u = thee->u;
08206 
08207     /* Make sure we're on the grid */
08208     if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
08209       (apos[1]<=ymin) || (apos[1]>=ymax)  || \
08210       (apos[2]<=zmin) || (apos[2]>=zmax)) {
08211         Vnm_print(2, "dbPermanentMultipoleForce:  Atom at (%4.3f, %4.3f, %4.3f) i
      s off the mesh (ignoring):\n", apos[0], apos[1], apos[2]);
08212         Vnm_print(2, "dbPermanentMultipoleForce:  xmin = %g, xmax = %g\n", xmin,
      xmax);
08213         Vnm_print(2, "dbPermanentMultipoleForce:  ymin = %g, ymax = %g\n", ymin,
      ymax);
08214         Vnm_print(2, "dbPermanentMultipoleForce:  zmin = %g, zmax = %g\n", zmin,
      zmax);
08215         fflush(stderr);
08216     } else {
08217 
08218         /* Convert the atom position to grid reference frame */
08219         position[0] = apos[0] - xmin;
08220         position[1] = apos[1] - ymin;
08221         position[2] = apos[2] - zmin;
08222 
08223         /* Integrate over points within this atom's (inflated) radius */
08224         rtot = (arad + thee->splineWin);
08225         rtot2 = VSQR(rtot);
08226         dx = rtot/hx;
08227         imin = (int)floor((position[0]-rtot)/hx);
08228         if (imin < 1) {
08229             Vnm_print(2, "dbPermanentMultipoleForce:  Atom off grid!\n");
08230             return;
08231         }
08232         imax = (int)ceil((position[0]+rtot)/hx);
08233         if (imax > (nx-2)) {
08234             Vnm_print(2, "dbPermanentMultipoleForce:  Atom off grid!\n");
08235             return;
08236         }
08237         jmin = (int)floor((position[1]-rtot)/hy);
08238         if (jmin < 1) {
08239             Vnm_print(2, "dbPermanentMultipoleForce:  Atom off grid!\n");
08240             return;
08241         }
08242         jmax = (int)ceil((position[1]+rtot)/hy);
08243         if (jmax > (ny-2)) {
08244             Vnm_print(2, "dbPermanentMultipoleForce:  Atom off grid!\n");
08245             return;
08246         }
08247         kmin = (int)floor((position[2]-rtot)/hzed);
```

```
08248         if (kmin < 1) {
08249             Vnm_print(2, "dbPermanentMultipoleForce:  Atom off grid!\n");
08250             return;
08251         }
08252         kmax = (int)ceil((position[2]+rtot)/hzed);
08253         if (kmax > (nz-2)) {
08254             Vnm_print(2, "dbPermanentMultipoleForce:  Atom off grid!\n");
08255             return;
08256         }
08257         for (i=imin; i<=imax; i++) {
08258             for (j=jmin; j<=jmax; j++) {
08259                 for (k=kmin; k<=kmax; k++) {
08260                     /* i,j,k */
08261                     gpos[0] = (i+0.5)*hx + xmin;
08262                     gpos[1] = j*hy + ymin;
08263                     gpos[2] = k*hzed + zmin;
08264                     Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
08265                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
08266                             atom, dHxijk);
08267                     for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
08268                     gpos[0] = i*hx + xmin;
08269                     gpos[1] = (j+0.5)*hy + ymin;
08270                     gpos[2] = k*hzed + zmin;
08271                     Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
08272                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
08273                             atom, dHyijk);
08274                     for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
08275                     gpos[0] = i*hx + xmin;
08276                     gpos[1] = j*hy + ymin;
08277                     gpos[2] = (k+0.5)*hzed + zmin;
08278                     Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
08279                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
08280                             atom, dHzijk);
08281                     for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
08282                     /* i-1,j,k */
08283                     gpos[0] = (i-0.5)*hx + xmin;
08284                     gpos[1] = j*hy + ymin;
08285                     gpos[2] = k*hzed + zmin;
08286                     Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
08287                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
08288                             atom, dHxim1jk);
08289                     for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
08290                     /* i,j-1,k */
08291                     gpos[0] = i*hx + xmin;
08292                     gpos[1] = (j-0.5)*hy + ymin;
08293                     gpos[2] = k*hzed + zmin;
08294                     Hyijm1k = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
08295                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
08296                             atom, dHyijm1k);
08297                     for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;
08298                     /* i,j,k-1 */
08299                     gpos[0] = i*hx + xmin;
08300                     gpos[1] = j*hy + ymin;
08301                     gpos[2] = (k-0.5)*hzed + zmin;
08302                     Hzijkm1 = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
08303                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
08304                             atom, dHzijkm1);
```

```
08305                     for (l=0; l<3; l++) dHzijkm1[l] *= Hzijkm1;
08306                     dbFmag = u[IJK(i,j,k)];
08307                     tgrad[0] =
08308                        (dHxijk[0]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
08309                      +  dHxim1jk[0]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
08310                      + (dHyijk[0]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
08311                      +  dHyijm1k[0]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
08312                      + (dHzijk[0]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
08313                      + dHzijkm1[0]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
08314                     tgrad[1] =
08315                        (dHxijk[1]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
08316                      +  dHxim1jk[1]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
08317                      + (dHyijk[1]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
08318                      +  dHyijm1k[1]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
08319                      + (dHzijk[1]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
08320                      + dHzijkm1[1]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
08321                     tgrad[2] =
08322                        (dHxijk[2]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
08323                      +  dHxim1jk[2]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
08324                      + (dHyijk[2]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
08325                      +  dHyijm1k[2]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
08326                      + (dHzijk[2]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
08327                      + dHzijkm1[2]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
08328                     force[0] += (dbFmag*tgrad[0]);
08329                     force[1] += (dbFmag*tgrad[1]);
08330                     force[2] += (dbFmag*tgrad[2]);
08331                 } /* k loop */
08332             } /* j loop */
08333         } /* i loop */
08334         force[0] = -force[0]*hx*hy*hzed*deps*0.5*izmagic;
08335         force[1] = -force[1]*hx*hy*hzed*deps*0.5*izmagic;
08336         force[2] = -force[2]*hx*hy*hzed*deps*0.5*izmagic;
08337     }
08338 }
08339
08340 VPUBLIC void Vpmg_qfDirectPolForce(Vpmg *thee, Vgrid* perm, Vgrid *induced,
08341                                    int atomID, double force[3], double torque[3])
      {
08342
08343     Vatom *atom;
08344     Vpbe *pbe;
08345     double f, fp, *u, *up, *apos, position[3];
08346
08347     /* Grid variables */
08348     int nx,ny,nz;
08349     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
08350     double hx, hy, hzed, ifloat, jfloat, kfloat;
08351
08352     /* B-spline weights */
08353     double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz, d3mx, d3my, d3mz;
08354     double mi, mj, mk;
08355
08356     /* Loop indeces */
08357     int i, j, k, ii, jj, kk;
08358     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
08359
08360     /* Permanent potential, field, field gradient and 2nd field gradient */
```

```
08361     double pot, e[3], de[3][3], d2e[3][3][3];
08362     /* Induced dipole field */
08363     double dep[3][3];
08364
08365     /* Permanent multipole components */
08366     double *dipole, *quad;
08367     double c, ux, uy, uz, qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
08368     double uix, uiy, uiz;
08369
08370     VASSERT(thee != VNULL);
08371     VASSERT(induced != VNULL); /* the potential due to permanent multipoles.*/
08372     VASSERT(induced != VNULL); /* the potential due to local induced dipoles.*/
08373     VASSERT(thee->pbe != VNULL);
08374     VASSERT(thee->pbe->alist != VNULL);
08375
08376     atom = Valist_getAtom(thee->pbe->alist, atomID);
08377     VASSERT(atom->partID != 0); /* all atoms must be in the same partition.*/
08378     apos = Vatom_getPosition(atom);
08379
08380     c = Vatom_getCharge(atom);
08381     dipole = Vatom_getDipole(atom);
08382     ux = dipole[0];
08383     uy = dipole[1];
08384     uz = dipole[2];
08385     quad = Vatom_getQuadrupole(atom);
08386     qxx = quad[0]/3.0;
08387     qxy = quad[1]/3.0;
08388     qxz = quad[2]/3.0;
08389     qyx = quad[3]/3.0;
08390     qyy = quad[4]/3.0;
08391     qyz = quad[5]/3.0;
08392     qzx = quad[6]/3.0;
08393     qzy = quad[7]/3.0;
08394     qzz = quad[8]/3.0;
08395
08396     dipole = Vatom_getInducedDipole(atom);
08397     uix = dipole[0];
08398     uiy = dipole[1];
08399     uiz = dipole[2];
08400
08401     /* Reset Field Gradients */
08402     pot = 0.0;
08403     for (i=0;i<3;i++){
08404        e[i] = 0.0;
08405        for (j=0;j<3;j++){
08406           de[i][j] = 0.0;
08407           dep[i][j] = 0.0;
08408           for (k=0;k<3;k++){
08409              d2e[i][j][k] = 0.0;
08410           }
08411        }
08412     }
08413
08414     /* Mesh info */
08415     nx = thee->pmgp->nx;
08416     ny = thee->pmgp->ny;
08417     nz = thee->pmgp->nz;
```

```
08418     hx = thee->pmgp->hx;
08419     hy = thee->pmgp->hy;
08420     hzed = thee->pmgp->hzed;
08421     xlen = thee->pmgp->xlen;
08422     ylen = thee->pmgp->ylen;
08423     zlen = thee->pmgp->zlen;
08424     xmin = thee->pmgp->xmin;
08425     ymin = thee->pmgp->ymin;
08426     zmin = thee->pmgp->zmin;
08427     xmax = thee->pmgp->xmax;
08428     ymax = thee->pmgp->ymax;
08429     zmax = thee->pmgp->zmax;
08430     u = induced->data;
08431     up = perm->data;
08432
08433     /* Make sure we're on the grid */
08434     if ((apos[0]<=(xmin+2*hx))   || (apos[0]>=(xmax-2*hx)) \
08435      || (apos[1]<=(ymin+2*hy))   || (apos[1]>=(ymax-2*hy)) \
08436      || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
08437         Vnm_print(2, "qfDirectPolForce:  Atom off the mesh (ignoring) %6.3f %6.3f
      %6.3f\n", apos[0], apos[1], apos[2]);
08438         fflush(stderr);
08439
08440     } else {
08441
08442         /* Convert the atom position to grid coordinates */
08443         position[0] = apos[0] - xmin;
08444         position[1] = apos[1] - ymin;
08445         position[2] = apos[2] - zmin;
08446         ifloat = position[0]/hx;
08447         jfloat = position[1]/hy;
08448         kfloat = position[2]/hzed;
08449         ip1 = (int)ceil(ifloat);
08450         ip2 = ip1 + 2;
08451         im1 = (int)floor(ifloat);
08452         im2 = im1 - 2;
08453         jp1 = (int)ceil(jfloat);
08454         jp2 = jp1 + 2;
08455         jm1 = (int)floor(jfloat);
08456         jm2 = jm1 - 2;
08457         kp1 = (int)ceil(kfloat);
08458         kp2 = kp1 + 2;
08459         km1 = (int)floor(kfloat);
08460         km2 = km1 - 2;
08461
08462         /* This step shouldn't be necessary, but it saves nasty debugging
08463          * later on if something goes wrong */
08464         ip2 = VMIN2(ip2,nx-1);
08465         ip1 = VMIN2(ip1,nx-1);
08466         im1 = VMAX2(im1,0);
08467         im2 = VMAX2(im2,0);
08468         jp2 = VMIN2(jp2,ny-1);
08469         jp1 = VMIN2(jp1,ny-1);
08470         jm1 = VMAX2(jm1,0);
08471         jm2 = VMAX2(jm2,0);
08472         kp2 = VMIN2(kp2,nz-1);
08473         kp1 = VMIN2(kp1,nz-1);
```

```
08474         km1 = VMAX2(km1,0);
08475         km2 = VMAX2(km2,0);
08476
08477         for (ii=im2; ii<=ip2; ii++) {
08478             mi = VFCHI4(ii,ifloat);
08479             mx = bspline4(mi);
08480             dmx = dbspline4(mi);
08481             d2mx = d2bspline4(mi);
08482             d3mx = d3bspline4(mi);
08483             for (jj=jm2; jj<=jp2; jj++) {
08484                 mj = VFCHI4(jj,jfloat);
08485                 my = bspline4(mj);
08486                 dmy = dbspline4(mj);
08487                 d2my = d2bspline4(mj);
08488                 d3my = d3bspline4(mj);
08489                 for (kk=km2; kk<=kp2; kk++) {
08490                     mk = VFCHI4(kk,kfloat);
08491                     mz = bspline4(mk);
08492                     dmz = dbspline4(mk);
08493                     d2mz = d2bspline4(mk);
08494                     d3mz = d3bspline4(mk);
08495                     f = u[IJK(ii,jj,kk)];
08496                     fp = up[IJK(ii,jj,kk)];
08497                     /* The potential */
08498                     pot  += f*mx*my*mz;
08499                     /* The field */
08500                     e[0] += f*dmx*my*mz/hx;
08501                     e[1] += f*mx*dmy*mz/hy;
08502                     e[2] += f*mx*my*dmz/hzed;
08503                     /* The gradient of the field */
08504                     de[0][0] += f*d2mx*my*mz/(hx*hx);
08505                     de[1][0] += f*dmx*dmy*mz/(hy*hx);
08506                     de[1][1] += f*mx*d2my*mz/(hy*hy);
08507                     de[2][0] += f*dmx*my*dmz/(hx*hzed);
08508                     de[2][1] += f*mx*dmy*dmz/(hy*hzed);
08509                     de[2][2] += f*mx*my*d2mz/(hzed*hzed);
08510                     /* The gradient of the (permanent) field */
08511                     dep[0][0] += fp*d2mx*my*mz/(hx*hx);
08512                     dep[1][0] += fp*dmx*dmy*mz/(hy*hx);
08513                     dep[1][1] += fp*mx*d2my*mz/(hy*hy);
08514                     dep[2][0] += fp*dmx*my*dmz/(hx*hzed);
08515                     dep[2][1] += fp*mx*dmy*dmz/(hy*hzed);
08516                     dep[2][2] += fp*mx*my*d2mz/(hzed*hzed);
08517                     /* The 2nd gradient of the field
08518                        VxVxVa */
08519                     d2e[0][0][0] += f*d3mx*my*mz /(hx*hx*hx);
08520                     d2e[0][0][1] += f*d2mx*dmy*mz/(hx*hy*hx);
08521                     d2e[0][0][2] += f*d2mx*my*dmz/(hx*hx*hzed);
08522                     /* VyVxVa */
08523                     d2e[1][0][0] += f*d2mx*dmy*mz/(hx*hx*hy);
08524                     d2e[1][0][1] += f*dmx*d2my*mz/(hx*hy*hy);
08525                     d2e[1][0][2] += f*dmx*dmy*dmz/(hx*hy*hzed);
08526                     /* VyVyVa */
08527                     d2e[1][1][0] += f*dmx*d2my*mz/(hx*hy*hy);
08528                     d2e[1][1][1] += f*mx*d3my*mz /(hy*hy*hy);
08529                     d2e[1][1][2] += f*mx*d2my*dmz/(hy*hy*hzed);
08530                     /* VzVxVa */
```

```
08531                     d2e[2][0][0] += f*d2mx*my*dmz/(hx*hx*hzed);
08532                     d2e[2][0][1] += f*dmx*dmy*dmz/(hx*hy*hzed);
08533                     d2e[2][0][2] += f*dmx*my*d2mz/(hx*hzed*hzed);
08534                     /* VzVyVa */
08535                     d2e[2][1][0] += f*dmx*dmy*dmz/(hx*hy*hzed);
08536                     d2e[2][1][1] += f*mx*d2my*dmz/(hy*hy*hzed);
08537                     d2e[2][1][2] += f*mx*dmy*d2mz/(hy*hzed*hzed);
08538                     /* VzVzVa */
08539                     d2e[2][2][0] += f*dmx*my*d2mz/(hx*hzed*hzed);
08540                     d2e[2][2][1] += f*mx*dmy*d2mz/(hy*hzed*hzed);
08541                     d2e[2][2][2] += f*mx*my*d3mz /(hzed*hzed*hzed);
08542                 }
08543             }
08544         }
08545     }
08546
08547     /* force on permanent multipole due to induced reaction field */
08548
08549     /* Monopole Force */
08550     force[0] = e[0]*c;
08551     force[1] = e[1]*c;
08552     force[2] = e[2]*c;
08553
08554     /* Dipole Force */
08555     force[0] -= de[0][0]*ux+de[1][0]*uy+de[2][0]*uz;
08556     force[1] -= de[1][0]*ux+de[1][1]*uy+de[2][1]*uz;
08557     force[2] -= de[2][0]*ux+de[2][1]*uy+de[2][2]*uz;
08558
08559     /* Quadrupole Force */
08560     force[0] += d2e[0][0][0]*qxx
08561              +  d2e[1][0][0]*qyx*2.0+d2e[1][1][0]*qyy
08562              +  d2e[2][0][0]*qzx*2.0+d2e[2][1][0]*qzy*2.0+d2e[2][2][0]*qzz;
08563     force[1] += d2e[0][0][1]*qxx
08564              +  d2e[1][0][1]*qyx*2.0+d2e[1][1][1]*qyy
08565              +  d2e[2][0][1]*qzx*2.0+d2e[2][1][1]*qzy*2.0+d2e[2][2][1]*qzz;
08566     force[2] += d2e[0][0][2]*qxx
08567              +  d2e[1][0][2]*qyx*2.0+d2e[1][1][2]*qyy
08568              +  d2e[2][0][2]*qzx*2.0+d2e[2][1][2]*qzy*2.0+d2e[2][2][2]*qzz;
08569
08570     /* torque on permanent mulitpole due to induced reaction field */
08571
08572     /* Dipole Torque */
08573     torque[0] = uy * e[2] - uz * e[1];
08574     torque[1] = uz * e[0] - ux * e[2];
08575     torque[2] = ux * e[1] - uy * e[0];
08576
08577     /* Quadrupole Torque */
08578     /* Tx = -2.0*(Sum_a (Qya*dEaz) + Sum_b (Qzb*dEby))
08579        Ty = -2.0*(Sum_a (Qza*dEax) + Sum_b (Qxb*dEbz))
08580        Tz = -2.0*(Sum_a (Qxa*dEay) + Sum_b (Qyb*dEbx))  */
08581     de[0][1] = de[1][0];
08582     de[0][2] = de[2][0];
08583     de[1][2] = de[2][1];
08584     torque[0] -= 2.0*(qyx*de[0][2] + qyy*de[1][2] + qyz*de[2][2]
08585                     - qzx*de[0][1] - qzy*de[1][1] - qzz*de[2][1]);
08586     torque[1] -= 2.0*(qzx*de[0][0] + qzy*de[1][0] + qzz*de[2][0]
08587                     - qxx*de[0][2] - qxy*de[1][2] - qxz*de[2][2]);
```

```
08588     torque[2] -= 2.0*(qxx*de[0][1] + qxy*de[1][1] + qxz*de[2][1]
08589                       - qyx*de[0][0] - qyy*de[1][0] - qyz*de[2][0]);
08590
08591     /* force on induced dipole due to permanent reaction field */
08592
08593     force[0] -= dep[0][0]*uix+dep[1][0]*uiy+dep[2][0]*uiz;
08594     force[1] -= dep[1][0]*uix+dep[1][1]*uiy+dep[2][1]*uiz;
08595     force[2] -= dep[2][0]*uix+dep[2][1]*uiy+dep[2][2]*uiz;
08596
08597     force[0] = 0.5 * force[0];
08598     force[1] = 0.5 * force[1];
08599     force[2] = 0.5 * force[2];
08600     torque[0] = 0.5 * torque[0];
08601     torque[1] = 0.5 * torque[1];
08602     torque[2] = 0.5 * torque[2];
08603
08604     /* printf(" qPhi Force %f %f %f\n", force[0], force[1], force[2]);
08605       printf(" qPhi Torque %f %f %f\n", torque[0], torque[1], torque[2]); */
08606 }
08607
08608 VPUBLIC void Vpmg_qfNLDirectPolForce(Vpmg *thee, Vgrid *perm, Vgrid *nlInduced,
08609                                      int atomID, double force[3], double torque[3
      ]) {
08610
08611     Vatom *atom;
08612     double *apos, *dipole, *quad, position[3], hx, hy, hzed;
08613     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
08614     double pot, e[3],de[3][3],dep[3][3],d2e[3][3][3];
08615     double mx, my, mz, dmx, dmy, dmz, mi, mj, mk;
08616     double d2mx, d2my, d2mz, d3mx, d3my, d3mz;
08617     double *u, *up, charge, ifloat, jfloat, kfloat;
08618     double f, fp, c, ux, uy, uz, qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
08619     double uix, uiy, uiz;
08620     int i,j,k,nx, ny, nz, im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1;
08621     int kp1, kp2, ii, jj, kk;
08622
08623     VASSERT(thee != VNULL);
08624     VASSERT(perm != VNULL);      /* potential due to permanent multipoles. */
08625     VASSERT(nlInduced != VNULL); /* potential due to non-local induced dipoles */

08626     VASSERT(!thee->pmgp->nonlin); /* Nonlinear PBE is not implemented for AMOEBA
      */
08627
08628     atom = Valist_getAtom(thee->pbe->alist, atomID);
08629     VASSERT(atom->partID != 0);   /* Currently all atoms must be in the same part
      ition. */
08630     apos = Vatom_getPosition(atom);
08631
08632     c = Vatom_getCharge(atom);
08633     dipole = Vatom_getDipole(atom);
08634     ux = dipole[0];
08635     uy = dipole[1];
08636     uz = dipole[2];
08637     quad = Vatom_getQuadrupole(atom);
08638     qxx = quad[0]/3.0;
08639     qxy = quad[1]/3.0;
08640     qxz = quad[2]/3.0;
```

```
08641     qyx = quad[3]/3.0;
08642     qyy = quad[4]/3.0;
08643     qyz = quad[5]/3.0;
08644     qzx = quad[6]/3.0;
08645     qzy = quad[7]/3.0;
08646     qzz = quad[8]/3.0;
08647
08648     dipole = Vatom_getNLInducedDipole(atom);
08649     uix = dipole[0];
08650     uiy = dipole[1];
08651     uiz = dipole[2];
08652
08653     /* Reset Field Gradients */
08654     pot = 0.0;
08655     for (i=0;i<3;i++){
08656        e[i] = 0.0;
08657        for (j=0;j<3;j++){
08658           de[i][j] = 0.0;
08659           dep[i][j] = 0.0;
08660           for (k=0;k<3;k++){
08661              d2e[i][j][k] = 0.0;
08662           }
08663        }
08664     }
08665
08666     /* Mesh info */
08667     nx = thee->pmgp->nx;
08668     ny = thee->pmgp->ny;
08669     nz = thee->pmgp->nz;
08670     hx = thee->pmgp->hx;
08671     hy = thee->pmgp->hy;
08672     hzed = thee->pmgp->hzed;
08673     xlen = thee->pmgp->xlen;
08674     ylen = thee->pmgp->ylen;
08675     zlen = thee->pmgp->zlen;
08676     xmin = thee->pmgp->xmin;
08677     ymin = thee->pmgp->ymin;
08678     zmin = thee->pmgp->zmin;
08679     xmax = thee->pmgp->xmax;
08680     ymax = thee->pmgp->ymax;
08681     zmax = thee->pmgp->zmax;
08682     u = nlInduced->data;
08683     up = perm->data;
08684
08685
08686     /* Make sure we're on the grid */
08687     if ((apos[0]<=(xmin+2*hx))   || (apos[0]>=(xmax-2*hx)) \
08688      || (apos[1]<=(ymin+2*hy))   || (apos[1]>=(ymax-2*hy)) \
08689      || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
08690        Vnm_print(2, "qfNLDirectMultipoleForce:  Atom off the mesh (ignoring) %6.
      3f %6.3f %6.3f\n", apos[0], apos[1], apos[2]);
08691     } else {
08692
08693        /* Convert the atom position to grid coordinates */
08694        position[0] = apos[0] - xmin;
08695        position[1] = apos[1] - ymin;
08696        position[2] = apos[2] - zmin;
```

```
08697         ifloat = position[0]/hx;
08698         jfloat = position[1]/hy;
08699         kfloat = position[2]/hzed;
08700         ip1 = (int)ceil(ifloat);
08701         ip2 = ip1 + 2;
08702         im1 = (int)floor(ifloat);
08703         im2 = im1 - 2;
08704         jp1 = (int)ceil(jfloat);
08705         jp2 = jp1 + 2;
08706         jm1 = (int)floor(jfloat);
08707         jm2 = jm1 - 2;
08708         kp1 = (int)ceil(kfloat);
08709         kp2 = kp1 + 2;
08710         km1 = (int)floor(kfloat);
08711         km2 = km1 - 2;
08712 
08713         /* This step shouldn't be necessary, but it saves nasty debugging
08714          * later on if something goes wrong */
08715         ip2 = VMIN2(ip2,nx-1);
08716         ip1 = VMIN2(ip1,nx-1);
08717         im1 = VMAX2(im1,0);
08718         im2 = VMAX2(im2,0);
08719         jp2 = VMIN2(jp2,ny-1);
08720         jp1 = VMIN2(jp1,ny-1);
08721         jm1 = VMAX2(jm1,0);
08722         jm2 = VMAX2(jm2,0);
08723         kp2 = VMIN2(kp2,nz-1);
08724         kp1 = VMIN2(kp1,nz-1);
08725         km1 = VMAX2(km1,0);
08726         km2 = VMAX2(km2,0);
08727 
08728         for (ii=im2; ii<=ip2; ii++) {
08729             mi = VFCHI4(ii,ifloat);
08730             mx = bspline4(mi);
08731             dmx = dbspline4(mi);
08732             d2mx = d2bspline4(mi);
08733             d3mx = d3bspline4(mi);
08734             for (jj=jm2; jj<=jp2; jj++) {
08735                 mj = VFCHI4(jj,jfloat);
08736                 my = bspline4(mj);
08737                 dmy = dbspline4(mj);
08738                 d2my = d2bspline4(mj);
08739                 d3my = d3bspline4(mj);
08740                 for (kk=km2; kk<=kp2; kk++) {
08741                     mk = VFCHI4(kk,kfloat);
08742                     mz = bspline4(mk);
08743                     dmz = dbspline4(mk);
08744                     d2mz = d2bspline4(mk);
08745                     d3mz = d3bspline4(mk);
08746                     f = u[IJK(ii,jj,kk)];
08747                     fp = up[IJK(ii,jj,kk)];
08748                     /* The potential */
08749                     pot  += f*mx*my*mz;
08750                     /* The field */
08751                     e[0] += f*dmx*my*mz/hx;
08752                     e[1] += f*mx*dmy*mz/hy;
08753                     e[2] += f*mx*my*dmz/hzed;
```

```
08754                     /* The gradient of the field */
08755                     de[0][0] += f*d2mx*my*mz/(hx*hx);
08756                     de[1][0] += f*dmx*dmy*mz/(hy*hx);
08757                     de[1][1] += f*mx*d2my*mz/(hy*hy);
08758                     de[2][0] += f*dmx*my*dmz/(hx*hzed);
08759                     de[2][1] += f*mx*dmy*dmz/(hy*hzed);
08760                     de[2][2] += f*mx*my*d2mz/(hzed*hzed);
08761                     /* The gradient of the (permanent) field */
08762                     dep[0][0] += fp*d2mx*my*mz/(hx*hx);
08763                     dep[1][0] += fp*dmx*dmy*mz/(hy*hx);
08764                     dep[1][1] += fp*mx*d2my*mz/(hy*hy);
08765                     dep[2][0] += fp*dmx*my*dmz/(hx*hzed);
08766                     dep[2][1] += fp*mx*dmy*dmz/(hy*hzed);
08767                     dep[2][2] += fp*mx*my*d2mz/(hzed*hzed);
08768                     /* The 2nd gradient of the field */
08769                     /* VxVxVa */
08770                     d2e[0][0][0] += f*d3mx*my*mz /(hx*hx*hx);
08771                     d2e[0][0][1] += f*d2mx*dmy*mz/(hx*hy*hx);
08772                     d2e[0][0][2] += f*d2mx*my*dmz/(hx*hx*hzed);
08773                     /* VyVxVa */
08774                     d2e[1][0][0] += f*d2mx*dmy*mz/(hx*hx*hy);
08775                     d2e[1][0][1] += f*dmx*d2my*mz/(hx*hy*hy);
08776                     d2e[1][0][2] += f*dmx*dmy*dmz/(hx*hy*hzed);
08777                     /* VyVyVa */
08778                     d2e[1][1][0] += f*dmx*d2my*mz/(hx*hy*hy);
08779                     d2e[1][1][1] += f*mx*d3my*mz /(hy*hy*hy);
08780                     d2e[1][1][2] += f*mx*d2my*dmz/(hy*hy*hzed);
08781                     /* VzVxVa */
08782                     d2e[2][0][0] += f*d2mx*my*dmz/(hx*hx*hzed);
08783                     d2e[2][0][1] += f*dmx*dmy*dmz/(hx*hy*hzed);
08784                     d2e[2][0][2] += f*dmx*my*d2mz/(hx*hzed*hzed);
08785                     /* VzVyVa */
08786                     d2e[2][1][0] += f*dmx*dmy*dmz/(hx*hy*hzed);
08787                     d2e[2][1][1] += f*mx*d2my*dmz/(hy*hy*hzed);
08788                     d2e[2][1][2] += f*mx*dmy*d2mz/(hy*hzed*hzed);
08789                     /* VzVzVa */
08790                     d2e[2][2][0] += f*dmx*my*d2mz/(hx*hzed*hzed);
08791                     d2e[2][2][1] += f*mx*dmy*d2mz/(hy*hzed*hzed);
08792                     d2e[2][2][2] += f*mx*my*d3mz /(hzed*hzed*hzed);
08793                 }
08794             }
08795         }
08796     }
08797
08798     /* force on permanent multipole due to non-local induced reaction field */
08799
08800     /* Monopole Force */
08801     force[0] = e[0]*c;
08802     force[1] = e[1]*c;
08803     force[2] = e[2]*c;
08804
08805     /* Dipole Force */
08806     force[0] -= de[0][0]*ux+de[1][0]*uy+de[2][0]*uz;
08807     force[1] -= de[1][0]*ux+de[1][1]*uy+de[2][1]*uz;
08808     force[2] -= de[2][0]*ux+de[2][1]*uy+de[2][2]*uz;
08809
08810     /* Quadrupole Force */
```

```
08811     force[0] += d2e[0][0][0]*qxx
08812               +  d2e[1][0][0]*qyx*2.0+d2e[1][1][0]*qyy
08813               +  d2e[2][0][0]*qzx*2.0+d2e[2][1][0]*qzy*2.0+d2e[2][2][0]*qzz;
08814     force[1] += d2e[0][0][1]*qxx
08815               +  d2e[1][0][1]*qyx*2.0+d2e[1][1][1]*qyy
08816               +  d2e[2][0][1]*qzx*2.0+d2e[2][1][1]*qzy*2.0+d2e[2][2][1]*qzz;
08817     force[2] += d2e[0][0][2]*qxx
08818               +  d2e[1][0][2]*qyx*2.0+d2e[1][1][2]*qyy
08819               +  d2e[2][0][2]*qzx*2.0+d2e[2][1][2]*qzy*2.0+d2e[2][2][2]*qzz;
08820
08821     /* torque on permanent mulitpole due to non-local induced reaction field */
08822
08823     /* Dipole Torque */
08824     torque[0] = uy * e[2] - uz * e[1];
08825     torque[1] = uz * e[0] - ux * e[2];
08826     torque[2] = ux * e[1] - uy * e[0];
08827
08828     /* Quadrupole Torque */
08829     /* Tx = -2.0*(Sum_a (Qya*dEaz) + Sum_b (Qzb*dEby))
08830        Ty = -2.0*(Sum_a (Qza*dEax) + Sum_b (Qxb*dEbz))
08831        Tz = -2.0*(Sum_a (Qxa*dEay) + Sum_b (Qyb*dEbx))  */
08832     de[0][1] = de[1][0];
08833     de[0][2] = de[2][0];
08834     de[1][2] = de[2][1];
08835     torque[0] -= 2.0*(qyx*de[0][2] + qyy*de[1][2] + qyz*de[2][2]
08836                     - qzx*de[0][1] - qzy*de[1][1] - qzz*de[2][1]);
08837     torque[1] -= 2.0*(qzx*de[0][0] + qzy*de[1][0] + qzz*de[2][0]
08838                     - qxx*de[0][2] - qxy*de[1][2] - qxz*de[2][2]);
08839     torque[2] -= 2.0*(qxx*de[0][1] + qxy*de[1][1] + qxz*de[2][1]
08840                     - qyx*de[0][0] - qyy*de[1][0] - qyz*de[2][0]);
08841
08842     /* force on non-local induced dipole due to permanent reaction field */
08843
08844     force[0] -= dep[0][0]*uix+dep[1][0]*uiy+dep[2][0]*uiz;
08845     force[1] -= dep[1][0]*uix+dep[1][1]*uiy+dep[2][1]*uiz;
08846     force[2] -= dep[2][0]*uix+dep[2][1]*uiy+dep[2][2]*uiz;
08847
08848     force[0] = 0.5 * force[0];
08849     force[1] = 0.5 * force[1];
08850     force[2] = 0.5 * force[2];
08851     torque[0] = 0.5 * torque[0];
08852     torque[1] = 0.5 * torque[1];
08853     torque[2] = 0.5 * torque[2];
08854
08855     /* printf(" qPhi Force %f %f %f\n", force[0], force[1], force[2]);
08856        printf(" qPhi Torque %f %f %f\n", torque[0], torque[1], torque[2]); */
08857 }
08858
08859 VPUBLIC void Vpmg_ibDirectPolForce(Vpmg *thee, Vgrid *perm, Vgrid *induced,
08860                                    int atomID, double force[3]) {
08861
08862     Vatom *atom;
08863     Valist *alist;
08864     Vacc *acc;
08865     Vpbe *pbe;
08866     Vsurf_Meth srfm;
08867
```

```
08868     double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
08869     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;
08870     double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
08871     double izmagic;
08872     int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
08873
08874     VASSERT(thee != VNULL);
08875     VASSERT(perm != VNULL);         /* potential due to permanent multipoles.*/
08876     VASSERT(induced != VNULL);      /* potential due to induced dipoles. */
08877     VASSERT (!thee->pmgp->nonlin); /* Nonlinear PBE is not implemented for AMOEBA
      */
08878
08879     acc = thee->pbe->acc;
08880     srfm = thee->surfMeth;
08881     atom = Valist_getAtom(thee->pbe->alist, atomID);
08882     VASSERT(atom->partID != 0);   /* Currently all atoms must be in the same part
     ition. */
08883     apos = Vatom_getPosition(atom);
08884     arad = Vatom_getRadius(atom);
08885
08886     /* Reset force */
08887     force[0] = 0.0;
08888     force[1] = 0.0;
08889     force[2] = 0.0;
08890
08891     /* Get PBE info */
08892     pbe = thee->pbe;
08893     acc = pbe->acc;
08894     alist = pbe->alist;
08895     irad = Vpbe_getMaxIonRadius(pbe);
08896     zkappa2 = Vpbe_getZkappa2(pbe);
08897     izmagic = 1.0/Vpbe_getZmagic(pbe);
08898
08899     VASSERT (zkappa2 > VPMGSMALL); /* It is ok to run AMOEBA with no ions, but th
     is is checked for higher up in the driver. */
08900
08901     /* Mesh info */
08902     nx = induced->nx;
08903     ny = induced->ny;
08904     nz = induced->nz;
08905     hx = induced->hx;
08906     hy = induced->hy;
08907     hzed = induced->hzed;
08908     xmin = induced->xmin;
08909     ymin = induced->ymin;
08910     zmin = induced->zmin;
08911     xmax = induced->xmax;
08912     ymax = induced->ymax;
08913     zmax = induced->zmax;
08914     xlen = xmax-xmin;
08915     ylen = ymax-ymin;
08916     zlen = zmax-zmin;
08917
08918     /* Make sure we're on the grid */
08919     if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
08920       (apos[1]<=ymin) || (apos[1]>=ymax)  || \
08921       (apos[2]<=zmin) || (apos[2]>=zmax)) {
```

```
08922         Vnm_print(2, "Vpmg_ibForce:  Atom at (%4.3f, %4.3f, %4.3f) is off the mes
      h (ignoring):\n",
08923             apos[0], apos[1], apos[2]);
08924         Vnm_print(2, "Vpmg_ibForce:    xmin = %g, xmax = %g\n", xmin, xmax);
08925         Vnm_print(2, "Vpmg_ibForce:    ymin = %g, ymax = %g\n", ymin, ymax);
08926         Vnm_print(2, "Vpmg_ibForce:    zmin = %g, zmax = %g\n", zmin, zmax);
08927         fflush(stderr);
08928     } else {
08929 
08930         /* Convert the atom position to grid reference frame */
08931         position[0] = apos[0] - xmin;
08932         position[1] = apos[1] - ymin;
08933         position[2] = apos[2] - zmin;
08934 
08935         /* Integrate over points within this atom's (inflated) radius */
08936         rtot = (irad + arad + thee->splineWin);
08937         rtot2 = VSQR(rtot);
08938         dx = rtot + 0.5*hx;
08939         imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
08940         imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
08941         for (i=imin; i<=imax; i++) {
08942             dx2 = VSQR(position[0] - hx*i);
08943             if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
08944             else dy = 0.5*hy;
08945             jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
08946             jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
08947             for (j=jmin; j<=jmax; j++) {
08948                 dy2 = VSQR(position[1] - hy*j);
08949                 if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
08950                 else dz = 0.5*hzed;
08951                 kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
08952                 kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
08953                 for (k=kmin; k<=kmax; k++) {
08954                     dz2 = VSQR(k*hzed - position[2]);
08955                     /* See if grid point is inside ivdw radius and set ccf
08956                      * accordingly (do spline assignment here) */
08957                     if ((dz2 + dy2 + dx2) <= rtot2) {
08958                         gpos[0] = i*hx + xmin;
08959                         gpos[1] = j*hy + ymin;
08960                         gpos[2] = k*hzed + zmin;
08961                         Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, irad,
08962                           atom, tgrad);
08963                         fmag = induced->data[IJK(i,j,k)];
08964                         fmag *= perm->data[IJK(i,j,k)];
08965                         fmag *= thee->kappa[IJK(i,j,k)];
08966                         force[0] += (zkappa2*fmag*tgrad[0]);
08967                         force[1] += (zkappa2*fmag*tgrad[1]);
08968                         force[2] += (zkappa2*fmag*tgrad[2]);
08969                     }
08970                 } /* k loop */
08971             } /* j loop */
08972         } /* i loop */
08973     }
08974 
08975     force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
08976     force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
```

```
08977     force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
08978
08979 }
08980
08981 VPUBLIC void Vpmg_ibNLDirectPolForce(Vpmg *thee, Vgrid *perm, Vgrid *nlInduced,
08982                                      int atomID, double force[3]) {
08983      Vpmg_ibDirectPolForce(thee, perm, nlInduced, atomID, force);
08984 }
08985
08986 VPUBLIC void Vpmg_dbDirectPolForce(Vpmg *thee, Vgrid *perm, Vgrid *induced,
08987                                    int atomID, double force[3]) {
08988
08989     Vatom *atom;
08990     Vacc *acc;
08991     Vpbe *pbe;
08992     Vsurf_Meth srfm;
08993
08994     double *apos, position[3], arad, hx, hy, hzed, izmagic, deps, depsi;
08995     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2, epsp;
08996     double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
08997     double *u, *up, Hxijk, Hyijk, Hzijk, Hxim1jk, Hyijm1k, Hzijkm1;
08998     double dHxijk[3], dHyijk[3], dHzijk[3], dHxim1jk[3], dHyijm1k[3];
08999     double dHzijkm1[3];
09000     int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
09001
09002     VASSERT(thee != VNULL);
09003     VASSERT(perm != VNULL);    /* permanent multipole PMG solution. */
09004     VASSERT(induced != VNULL); /* potential due to induced dipoles. */
09005
09006     acc = thee->pbe->acc;
09007     atom = Valist_getAtom(thee->pbe->alist, atomID);
09008     VASSERT (atom->partID != 0);   /* Currently all atoms must be in the same par
      tition. */
09009     apos = Vatom_getPosition(atom);
09010     arad = Vatom_getRadius(atom);
09011
09012     /* Reset force */
09013     force[0] = 0.0;
09014     force[1] = 0.0;
09015     force[2] = 0.0;
09016
09017     /* Get PBE info */
09018     pbe = thee->pbe;
09019     acc = pbe->acc;
09020     srfm = thee->surfMeth;
09021     epsp = Vpbe_getSoluteDiel(pbe);
09022     epsw = Vpbe_getSolventDiel(pbe);
09023     kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*Vunit_kb;
09024     izmagic = 1.0/Vpbe_getZmagic(pbe);
09025
09026     deps = (epsw - epsp);
09027     depsi = 1.0/deps;
09028     VASSERT(VABS(deps) > VPMGSMALL);
09029
09030     /* Mesh info */
09031     nx = thee->pmgp->nx;
09032     ny = thee->pmgp->ny;
```

```
09033     nz = thee->pmgp->nz;
09034     hx = thee->pmgp->hx;
09035     hy = thee->pmgp->hy;
09036     hzed = thee->pmgp->hzed;
09037     xlen = thee->pmgp->xlen;
09038     ylen = thee->pmgp->ylen;
09039     zlen = thee->pmgp->zlen;
09040     xmin = thee->pmgp->xmin;
09041     ymin = thee->pmgp->ymin;
09042     zmin = thee->pmgp->zmin;
09043     xmax = thee->pmgp->xmax;
09044     ymax = thee->pmgp->ymax;
09045     zmax = thee->pmgp->zmax;
09046     /* If the permanent and induced potentials are flipped the
09047        results are exactly the same. */
09048     u = induced->data;
09049     up = perm->data;
09050 
09051     /* Make sure we're on the grid */
09052     if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
09053       (apos[1]<=ymin) || (apos[1]>=ymax)  || \
09054       (apos[2]<=zmin) || (apos[2]>=zmax)) {
09055         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]);
09056         Vnm_print(2, "Vpmg_dbDirectPolForce:    xmin = %g, xmax = %g\n", xmin, x
      max);
09057         Vnm_print(2, "Vpmg_dbDirectPolForce:    ymin = %g, ymax = %g\n", ymin, y
      max);
09058         Vnm_print(2, "Vpmg_dbDirectPolForce:    zmin = %g, zmax = %g\n", zmin, z
      max);
09059         fflush(stderr);
09060     } else {
09061 
09062       /* Convert the atom position to grid reference frame */
09063       position[0] = apos[0] - xmin;
09064       position[1] = apos[1] - ymin;
09065       position[2] = apos[2] - zmin;
09066 
09067       /* Integrate over points within this atom's (inflated) radius */
09068       rtot = (arad + thee->splineWin);
09069       rtot2 = VSQR(rtot);
09070       dx = rtot/hx;
09071       imin = (int)floor((position[0]-rtot)/hx);
09072       if (imin < 1) {
09073           Vnm_print(2, "Vpmg_dbDirectPolForce:  Atom %d off grid!\n", atomID);

09074           return;
09075       }
09076       imax = (int)ceil((position[0]+rtot)/hx);
09077       if (imax > (nx-2)) {
09078           Vnm_print(2, "Vpmg_dbDirectPolForce:  Atom %d off grid!\n", atomID);

09079           return;
09080       }
09081       jmin = (int)floor((position[1]-rtot)/hy);
09082       if (jmin < 1) {
09083           Vnm_print(2, "Vpmg_dbDirectPolForce:  Atom %d off grid!\n", atomID);
```

```
09084             return;
09085         }
09086         jmax = (int)ceil((position[1]+rtot)/hy);
09087         if (jmax > (ny-2)) {
09088             Vnm_print(2, "Vpmg_dbDirectPolForce:  Atom %d off grid!\n", atomID);
09089             return;
09090         }
09091         kmin = (int)floor((position[2]-rtot)/hzed);
09092         if (kmin < 1) {
09093             Vnm_print(2, "Vpmg_dbDirectPolForce:  Atom %d off grid!\n", atomID);
09094             return;
09095         }
09096         kmax = (int)ceil((position[2]+rtot)/hzed);
09097         if (kmax > (nz-2)) {
09098             Vnm_print(2, "Vpmg_dbDirectPolForce:  Atom %d off grid!\n", atomID);
09099             return;
09100         }
09101         for (i=imin; i<=imax; i++) {
09102             for (j=jmin; j<=jmax; j++) {
09103                 for (k=kmin; k<=kmax; k++) {
09104                     /* i,j,k */
09105                     gpos[0] = (i+0.5)*hx + xmin;
09106                     gpos[1] = j*hy + ymin;
09107                     gpos[2] = k*hzed + zmin;
09108                     Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
09109                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09110                             atom, dHxijk);
09111                     for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
09112                     gpos[0] = i*hx + xmin;
09113                     gpos[1] = (j+0.5)*hy + ymin;
09114                     gpos[2] = k*hzed + zmin;
09115                     Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
09116                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09117                             atom, dHyijk);
09118                     for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
09119                     gpos[0] = i*hx + xmin;
09120                     gpos[1] = j*hy + ymin;
09121                     gpos[2] = (k+0.5)*hzed + zmin;
09122                     Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
09123                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09124                             atom, dHzijk);
09125                     for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
09126                     /* i-1,j,k */
09127                     gpos[0] = (i-0.5)*hx + xmin;
09128                     gpos[1] = j*hy + ymin;
09129                     gpos[2] = k*hzed + zmin;
09130                     Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
09131                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09132                             atom, dHxim1jk);
09133                     for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
09134                     /* i,j-1,k */
09135                     gpos[0] = i*hx + xmin;
09136                     gpos[1] = (j-0.5)*hy + ymin;
```

```
09137                     gpos[2] = k*hzed + zmin;
09138                     Hyijm1k = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
09139                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09140                             atom, dHyijm1k);
09141                     for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;
09142                     /* i,j,k-1 */
09143                     gpos[0] = i*hx + xmin;
09144                     gpos[1] = j*hy + ymin;
09145                     gpos[2] = (k-0.5)*hzed + zmin;
09146                     Hzijkm1 = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
09147                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09148                             atom, dHzijkm1);
09149                     for (l=0; l<3; l++) dHzijkm1[l] *= Hzijkm1;
09150
09151                     dbFmag = up[IJK(i,j,k)];
09152                     tgrad[0] =
09153                        (dHxijk[0]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09154                         +  dHxim1jk[0]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09155                         + (dHyijk[0]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09156                         +  dHyijm1k[0]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09157                         + (dHzijk[0]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
09158                         + dHzijkm1[0]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09159                     tgrad[1] =
09160                        (dHxijk[1]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09161                         +  dHxim1jk[1]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09162                         + (dHyijk[1]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09163                         +  dHyijm1k[1]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09164                         + (dHzijk[1]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
09165                         + dHzijkm1[1]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09166                     tgrad[2] =
09167                        (dHxijk[2]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09168                         +  dHxim1jk[2]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09169                         + (dHyijk[2]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09170                         +  dHyijm1k[2]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09171                         + (dHzijk[2]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
09172                         + dHzijkm1[2]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09173                     force[0] += (dbFmag*tgrad[0]);
09174                     force[1] += (dbFmag*tgrad[1]);
09175                     force[2] += (dbFmag*tgrad[2]);
09176
09177                 } /* k loop */
09178             } /* j loop */
09179         } /* i loop */
09180
09181         force[0] = -force[0]*hx*hy*hzed*deps*0.5*izmagic;
09182         force[1] = -force[1]*hx*hy*hzed*deps*0.5*izmagic;
09183         force[2] = -force[2]*hx*hy*hzed*deps*0.5*izmagic;
09184
09185     }
09186 }
09187
09188 VPUBLIC void Vpmg_dbNLDirectPolForce(Vpmg *thee, Vgrid *perm, Vgrid *nlInduced,
09189                                      int atomID, double force[3]) {
09190      Vpmg_dbDirectPolForce(thee, perm, nlInduced, atomID, force);
09191 }
09192
09193 VPUBLIC void Vpmg_qfMutualPolForce(Vpmg *thee, Vgrid *induced,
```

```
09194                               Vgrid *nlinduced, int atomID, double force[3]) {
09195
09196     Vatom *atom;
09197     double *apos, *dipole, position[3], hx, hy, hzed;
09198     double *u, *unl;
09199     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
09200     double de[3][3], denl[3][3];
09201     double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz, mi, mj, mk;
09202     double ifloat, jfloat, kfloat;
09203     double f, fnl, uix, uiy, uiz, uixnl, uiynl, uiznl;
09204     int i,j,k,nx, ny, nz, im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1;
09205     int kp1, kp2, ii, jj, kk;
09206
09207     VASSERT(thee != VNULL);    /* PMG object with PBE info. */
09208     VASSERT(induced != VNULL); /* potential due to induced dipoles. */
09209     VASSERT(nlinduced != VNULL); /* potential due to non-local induced dipoles. *
    /
09210     atom = Valist_getAtom(thee->pbe->alist, atomID);
09211     VASSERT(atom->partID != 0);    /* all atoms must be in the same partition. */

09212     apos = Vatom_getPosition(atom);
09213     dipole = Vatom_getInducedDipole(atom);
09214     uix = dipole[0];
09215     uiy = dipole[1];
09216     uiz = dipole[2];
09217     dipole = Vatom_getNLInducedDipole(atom);
09218     uixnl = dipole[0];
09219     uiynl = dipole[1];
09220     uiznl = dipole[2];
09221     u = induced->data;
09222     unl = nlinduced->data;
09223
09224     for (i=0;i<3;i++){
09225        for (j=0;j<3;j++){
09226           de[i][j] = 0.0;
09227           denl[i][j] = 0.0;
09228        }
09229     }
09230
09231     /* Mesh info */
09232     nx = induced->nx;
09233     ny = induced->ny;
09234     nz = induced->nz;
09235     hx = induced->hx;
09236     hy = induced->hy;
09237     hzed = induced->hzed;
09238     xmin = induced->xmin;
09239     ymin = induced->ymin;
09240     zmin = induced->zmin;
09241     xmax = induced->xmax;
09242     ymax = induced->ymax;
09243     zmax = induced->zmax;
09244     xlen = xmax-xmin;
09245     ylen = ymax-ymin;
09246     zlen = zmax-zmin;
09247
09248     /* If we aren't in the current position, then we're done */
```

```
09249     if (atom->partID == 0) return;
09250
09251     /* Make sure we're on the grid */
09252     if ((apos[0]<=(xmin+2*hx))   || (apos[0]>=(xmax-2*hx)) \
09253      || (apos[1]<=(ymin+2*hy))   || (apos[1]>=(ymax-2*hy)) \
09254      || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
09255         Vnm_print(2, "qfMutualPolForce:  Atom off the mesh (ignoring) %6.3f %6.3f
       %6.3f\n", apos[0], apos[1], apos[2]);
09256         fflush(stderr);
09257     } else {
09258
09259         /* Convert the atom position to grid coordinates */
09260         position[0] = apos[0] - xmin;
09261         position[1] = apos[1] - ymin;
09262         position[2] = apos[2] - zmin;
09263         ifloat = position[0]/hx;
09264         jfloat = position[1]/hy;
09265         kfloat = position[2]/hzed;
09266         ip1 = (int)ceil(ifloat);
09267         ip2 = ip1 + 2;
09268         im1 = (int)floor(ifloat);
09269         im2 = im1 - 2;
09270         jp1 = (int)ceil(jfloat);
09271         jp2 = jp1 + 2;
09272         jm1 = (int)floor(jfloat);
09273         jm2 = jm1 - 2;
09274         kp1 = (int)ceil(kfloat);
09275         kp2 = kp1 + 2;
09276         km1 = (int)floor(kfloat);
09277         km2 = km1 - 2;
09278
09279         /* This step shouldn't be necessary, but it saves nasty debugging
09280          * later on if something goes wrong */
09281         ip2 = VMIN2(ip2,nx-1);
09282         ip1 = VMIN2(ip1,nx-1);
09283         im1 = VMAX2(im1,0);
09284         im2 = VMAX2(im2,0);
09285         jp2 = VMIN2(jp2,ny-1);
09286         jp1 = VMIN2(jp1,ny-1);
09287         jm1 = VMAX2(jm1,0);
09288         jm2 = VMAX2(jm2,0);
09289         kp2 = VMIN2(kp2,nz-1);
09290         kp1 = VMIN2(kp1,nz-1);
09291         km1 = VMAX2(km1,0);
09292         km2 = VMAX2(km2,0);
09293
09294         for (ii=im2; ii<=ip2; ii++) {
09295             mi = VFCHI4(ii,ifloat);
09296             mx = bspline4(mi);
09297             dmx = dbspline4(mi);
09298             d2mx = d2bspline4(mi);
09299             for (jj=jm2; jj<=jp2; jj++) {
09300                 mj = VFCHI4(jj,jfloat);
09301                 my = bspline4(mj);
09302                 dmy = dbspline4(mj);
09303                 d2my = d2bspline4(mj);
09304                 for (kk=km2; kk<=kp2; kk++) {
```

```
09305                     mk = VFCHI4(kk,kfloat);
09306                     mz = bspline4(mk);
09307                     dmz = dbspline4(mk);
09308                     d2mz = d2bspline4(mk);
09309                     f = u[IJK(ii,jj,kk)];
09310                     fnl = unl[IJK(ii,jj,kk)];
09311
09312                     /* The gradient of the reaction field
09313                        due to induced dipoles */
09314                     de[0][0] += f*d2mx*my*mz/(hx*hx);
09315                     de[1][0] += f*dmx*dmy*mz/(hy*hx);
09316                     de[1][1] += f*mx*d2my*mz/(hy*hy);
09317                     de[2][0] += f*dmx*my*dmz/(hx*hzed);
09318                     de[2][1] += f*mx*dmy*dmz/(hy*hzed);
09319                     de[2][2] += f*mx*my*d2mz/(hzed*hzed);
09320
09321                     /* The gradient of the reaction field
09322                        due to non-local induced dipoles */
09323                     denl[0][0] += fnl*d2mx*my*mz/(hx*hx);
09324                     denl[1][0] += fnl*dmx*dmy*mz/(hy*hx);
09325                     denl[1][1] += fnl*mx*d2my*mz/(hy*hy);
09326                     denl[2][0] += fnl*dmx*my*dmz/(hx*hzed);
09327                     denl[2][1] += fnl*mx*dmy*dmz/(hy*hzed);
09328                     denl[2][2] += fnl*mx*my*d2mz/(hzed*hzed);
09329                 }
09330             }
09331         }
09332     }
09333
09334     /* mutual polarization force */
09335     force[0] = -(de[0][0]*uixnl + de[1][0]*uiynl + de[2][0]*uiznl);
09336     force[1] = -(de[1][0]*uixnl + de[1][1]*uiynl + de[2][1]*uiznl);
09337     force[2] = -(de[2][0]*uixnl + de[2][1]*uiynl + de[2][2]*uiznl);
09338     force[0] -=  denl[0][0]*uix + denl[1][0]*uiy + denl[2][0]*uiz;
09339     force[1] -=  denl[1][0]*uix + denl[1][1]*uiy + denl[2][1]*uiz;
09340     force[2] -=  denl[2][0]*uix + denl[2][1]*uiy + denl[2][2]*uiz;
09341
09342     force[0] = 0.5 * force[0];
09343     force[1] = 0.5 * force[1];
09344     force[2] = 0.5 * force[2];
09345
09346 }
09347
09348 VPUBLIC void Vpmg_ibMutualPolForce(Vpmg *thee, Vgrid *induced, Vgrid *nlinduced,
09349                                    int atomID, double force[3]) {
09350
09351     Vatom *atom;
09352     Valist *alist;
09353     Vacc *acc;
09354     Vpbe *pbe;
09355     Vsurf_Meth srfm;
09356
09357     double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
09358     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;
09359     double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
09360     double izmagic;
09361     int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
```

```
09362
09363     VASSERT(thee != VNULL);         /* We need a PMG object with PBE info. */
09364     VASSERT(induced != VNULL);      /* We need the potential due to induced dipole
      s. */
09365     VASSERT(nlinduced != VNULL);    /* We need the potential due to non-local indu
      ced dipoles. */
09366     VASSERT (!thee->pmgp->nonlin); /* Nonlinear PBE is not implemented for AMOEBA
       */
09367
09368     atom = Valist_getAtom(thee->pbe->alist, atomID);
09369  VASSERT (atom->partID != 0);   /* Currently all atoms must be in the same partit
      ion. */
09370
09371     acc = thee->pbe->acc;
09372     srfm = thee->surfMeth;
09373     apos = Vatom_getPosition(atom);
09374     arad = Vatom_getRadius(atom);
09375
09376     /* Reset force */
09377     force[0] = 0.0;
09378     force[1] = 0.0;
09379     force[2] = 0.0;
09380
09381     /* If we aren't in the current position, then we're done */
09382     if (atom->partID == 0) return;
09383
09384     /* Get PBE info */
09385     pbe = thee->pbe;
09386     acc = pbe->acc;
09387     alist = pbe->alist;
09388     irad = Vpbe_getMaxIonRadius(pbe);
09389     zkappa2 = Vpbe_getZkappa2(pbe);
09390     izmagic = 1.0/Vpbe_getZmagic(pbe);
09391
09392     VASSERT (zkappa2 > VPMGSMALL); /* Should be a check for this further up.*/
09393
09394     /* Mesh info */
09395     nx = induced->nx;
09396     ny = induced->ny;
09397     nz = induced->nz;
09398     hx = induced->hx;
09399     hy = induced->hy;
09400     hzed = induced->hzed;
09401     xmin = induced->xmin;
09402     ymin = induced->ymin;
09403     zmin = induced->zmin;
09404     xmax = induced->xmax;
09405     ymax = induced->ymax;
09406     zmax = induced->zmax;
09407     xlen = xmax-xmin;
09408     ylen = ymax-ymin;
09409     zlen = zmax-zmin;
09410
09411     /* Make sure we're on the grid */
09412     if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
09413       (apos[1]<=ymin) || (apos[1]>=ymax)  || \
09414       (apos[2]<=zmin) || (apos[2]>=zmax)) {
```

```
09415         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]);
09416         Vnm_print(2, "Vpmg_ibMutalPolForce:    xmin = %g, xmax = %g\n", xmin, xma
      x);
09417         Vnm_print(2, "Vpmg_ibMutalPolForce:    ymin = %g, ymax = %g\n", ymin, yma
      x);
09418         Vnm_print(2, "Vpmg_ibMutalPolForce:    zmin = %g, zmax = %g\n", zmin, zma
      x);
09419         fflush(stderr);
09420     } else {
09421 
09422         /* Convert the atom position to grid reference frame */
09423         position[0] = apos[0] - xmin;
09424         position[1] = apos[1] - ymin;
09425         position[2] = apos[2] - zmin;
09426 
09427         /* Integrate over points within this atom's (inflated) radius */
09428         rtot = (irad + arad + thee->splineWin);
09429         rtot2 = VSQR(rtot);
09430         dx = rtot + 0.5*hx;
09431         imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
09432         imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
09433         for (i=imin; i<=imax; i++) {
09434             dx2 = VSQR(position[0] - hx*i);
09435             if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
09436             else dy = 0.5*hy;
09437             jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
09438             jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
09439             for (j=jmin; j<=jmax; j++) {
09440                 dy2 = VSQR(position[1] - hy*j);
09441                 if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
09442                 else dz = 0.5*hzed;
09443                 kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
09444                 kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
09445                 for (k=kmin; k<=kmax; k++) {
09446                     dz2 = VSQR(k*hzed - position[2]);
09447                     /* See if grid point is inside ivdw radius and set ccf
09448                      * accordingly (do spline assignment here) */
09449                     if ((dz2 + dy2 + dx2) <= rtot2) {
09450                         gpos[0] = i*hx + xmin;
09451                         gpos[1] = j*hy + ymin;
09452                         gpos[2] = k*hzed + zmin;
09453                         Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, irad,
09454                           atom, tgrad);
09455                         fmag = induced->data[IJK(i,j,k)];
09456                         fmag *= nlinduced->data[IJK(i,j,k)];
09457                         fmag *= thee->kappa[IJK(i,j,k)];
09458                         force[0] += (zkappa2*fmag*tgrad[0]);
09459                         force[1] += (zkappa2*fmag*tgrad[1]);
09460                         force[2] += (zkappa2*fmag*tgrad[2]);
09461                     }
09462                 } /* k loop */
09463             } /* j loop */
09464         } /* i loop */
09465     }
09466 
```

```
09467     force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
09468     force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
09469     force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
09470 }
09471
09472 VPUBLIC void Vpmg_dbMutualPolForce(Vpmg *thee, Vgrid *induced,
09473                                    Vgrid *nlinduced, int atomID,
09474                                    double force[3]) {
09475
09476     Vatom *atom;
09477     Vacc *acc;
09478     Vpbe *pbe;
09479     Vsurf_Meth srfm;
09480
09481     double *apos, position[3], arad, hx, hy, hzed, izmagic, deps, depsi;
09482     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2, epsp;
09483     double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
09484     double *u, *unl, Hxijk, Hyijk, Hzijk, Hxim1jk, Hyijm1k, Hzijkm1;
09485     double dHxijk[3], dHyijk[3], dHzijk[3], dHxim1jk[3], dHyijm1k[3];
09486     double dHzijkm1[3];
09487     int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
09488
09489     VASSERT(thee != VNULL); /* PMG object with PBE info. */
09490     VASSERT(induced != VNULL); /* potential due to induced dipoles.*/
09491     VASSERT(nlinduced != VNULL); /* potential due to non-local induced dipoles.*/

09492
09493     acc = thee->pbe->acc;
09494     srfm = thee->surfMeth;
09495     atom = Valist_getAtom(thee->pbe->alist, atomID);
09496     VASSERT (atom->partID != 0); /* all atoms must be in the same partition.*/
09497     apos = Vatom_getPosition(atom);
09498     arad = Vatom_getRadius(atom);
09499
09500     /* Reset force */
09501     force[0] = 0.0;
09502     force[1] = 0.0;
09503     force[2] = 0.0;
09504
09505     /* Get PBE info */
09506     pbe = thee->pbe;
09507     acc = pbe->acc;
09508     epsp = Vpbe_getSoluteDiel(pbe);
09509     epsw = Vpbe_getSolventDiel(pbe);
09510     kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*Vunit_kb;
09511     izmagic = 1.0/Vpbe_getZmagic(pbe);
09512
09513     deps = (epsw - epsp);
09514     depsi = 1.0/deps;
09515     VASSERT(VABS(deps) > VPMGSMALL);
09516
09517     /* Mesh info */
09518     nx = thee->pmgp->nx;
09519     ny = thee->pmgp->ny;
09520     nz = thee->pmgp->nz;
09521     hx = thee->pmgp->hx;
09522     hy = thee->pmgp->hy;
```

```
09523     hzed = thee->pmgp->hzed;
09524     xlen = thee->pmgp->xlen;
09525     ylen = thee->pmgp->ylen;
09526     zlen = thee->pmgp->zlen;
09527     xmin = thee->pmgp->xmin;
09528     ymin = thee->pmgp->ymin;
09529     zmin = thee->pmgp->zmin;
09530     xmax = thee->pmgp->xmax;
09531     ymax = thee->pmgp->ymax;
09532     zmax = thee->pmgp->zmax;
09533     u = induced->data;
09534     unl = nlinduced->data;
09535
09536     /* Make sure we're on the grid */
09537     if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
09538       (apos[1]<=ymin) || (apos[1]>=ymax)  || \
09539       (apos[2]<=zmin) || (apos[2]>=zmax)) {
09540         Vnm_print(2, "Vpmg_dbMutualPolForce:  Atom at (%4.3f, %4.3f, %4.3f) is of
      f the mesh (ignoring):\n", apos[0], apos[1], apos[2]);
09541         Vnm_print(2, "Vpmg_dbMutualPolForce:    xmin = %g, xmax = %g\n", xmin, xm
      ax);
09542         Vnm_print(2, "Vpmg_dbMutualPolForce:    ymin = %g, ymax = %g\n", ymin, ym
      ax);
09543         Vnm_print(2, "Vpmg_dbMutualPolForce:    zmin = %g, zmax = %g\n", zmin, zm
      ax);
09544         fflush(stderr);
09545     } else {
09546
09547         /* Convert the atom position to grid reference frame */
09548         position[0] = apos[0] - xmin;
09549         position[1] = apos[1] - ymin;
09550         position[2] = apos[2] - zmin;
09551
09552         /* Integrate over points within this atom's (inflated) radius */
09553         rtot = (arad + thee->splineWin);
09554         rtot2 = VSQR(rtot);
09555         dx = rtot/hx;
09556         imin = (int)floor((position[0]-rtot)/hx);
09557         if (imin < 1) {
09558             Vnm_print(2, "Vpmg_dbMutualPolForce:  Atom %d off grid!\n", atomID);

09559             return;
09560         }
09561         imax = (int)ceil((position[0]+rtot)/hx);
09562         if (imax > (nx-2)) {
09563             Vnm_print(2, "Vpmg_dbMutualPolForce:  Atom %d off grid!\n", atomID);

09564             return;
09565         }
09566         jmin = (int)floor((position[1]-rtot)/hy);
09567         if (jmin < 1) {
09568             Vnm_print(2, "Vpmg_dbMutualPolForce:  Atom %d off grid!\n", atomID);

09569             return;
09570         }
09571         jmax = (int)ceil((position[1]+rtot)/hy);
09572         if (jmax > (ny-2)) {
```

```
09573             Vnm_print(2, "Vpmg_dbMutualPolForce:  Atom %d off grid!\n", atomID);

09574             return;
09575         }
09576         kmin = (int)floor((position[2]-rtot)/hzed);
09577         if (kmin < 1) {
09578             Vnm_print(2, "Vpmg_dbMutualPolForce:  Atom %d off grid!\n", atomID);

09579             return;
09580         }
09581         kmax = (int)ceil((position[2]+rtot)/hzed);
09582         if (kmax > (nz-2)) {
09583             Vnm_print(2, "Vpmg_dbMutualPolForce:  Atom %d off grid!\n", atomID);

09584             return;
09585         }
09586         for (i=imin; i<=imax; i++) {
09587             for (j=jmin; j<=jmax; j++) {
09588                 for (k=kmin; k<=kmax; k++) {
09589                     /* i,j,k */
09590                     gpos[0] = (i+0.5)*hx + xmin;
09591                     gpos[1] = j*hy + ymin;
09592                     gpos[2] = k*hzed + zmin;
09593                     Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
09594                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09595                             atom, dHxijk);
09596                     for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
09597                     gpos[0] = i*hx + xmin;
09598                     gpos[1] = (j+0.5)*hy + ymin;
09599                     gpos[2] = k*hzed + zmin;
09600                     Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
09601                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09602                             atom, dHyijk);
09603                     for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
09604                     gpos[0] = i*hx + xmin;
09605                     gpos[1] = j*hy + ymin;
09606                     gpos[2] = (k+0.5)*hzed + zmin;
09607                     Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
09608                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09609                             atom, dHzijk);
09610                     for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
09611                     /* i-1,j,k */
09612                     gpos[0] = (i-0.5)*hx + xmin;
09613                     gpos[1] = j*hy + ymin;
09614                     gpos[2] = k*hzed + zmin;
09615                     Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
09616                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09617                             atom, dHxim1jk);
09618                     for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
09619                     /* i,j-1,k */
09620                     gpos[0] = i*hx + xmin;
09621                     gpos[1] = (j-0.5)*hy + ymin;
09622                     gpos[2] = k*hzed + zmin;
09623                     Hyijm1k = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
09624                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09625                             atom, dHyijm1k);
09626                     for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;
```

```
09627                     /* i,j,k-1 */
09628                     gpos[0] = i*hx + xmin;
09629                     gpos[1] = j*hy + ymin;
09630                     gpos[2] = (k-0.5)*hzed + zmin;
09631                     Hzijkm1 = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
09632                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09633                             atom, dHzijkm1);
09634                     for (l=0; l<3; l++) dHzijkm1[l] *= Hzijkm1;
09635                     dbFmag = unl[IJK(i,j,k)];
09636                     tgrad[0] =
09637                        (dHxijk[0]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09638                      +  dHxim1jk[0]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09639                      + (dHyijk[0]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09640                      +  dHyijm1k[0]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09641                      + (dHzijk[0]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
09642                      + dHzijkm1[0]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09643                     tgrad[1] =
09644                        (dHxijk[1]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09645                      +  dHxim1jk[1]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09646                      + (dHyijk[1]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09647                      +  dHyijm1k[1]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09648                      + (dHzijk[1]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
09649                      + dHzijkm1[1]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09650                     tgrad[2] =
09651                        (dHxijk[2]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09652                      +  dHxim1jk[2]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09653                      + (dHyijk[2]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09654                      +  dHyijm1k[2]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09655                      + (dHzijk[2]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
09656                      + dHzijkm1[2]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09657                      force[0] += (dbFmag*tgrad[0]);
09658                      force[1] += (dbFmag*tgrad[1]);
09659                      force[2] += (dbFmag*tgrad[2]);
09660                  } /* k loop */
09661              } /* j loop */
09662          } /* i loop */
09663
09664          force[0] = -force[0]*hx*hy*hzed*deps*0.5*izmagic;
09665          force[1] = -force[1]*hx*hy*hzed*deps*0.5*izmagic;
09666          force[2] = -force[2]*hx*hy*hzed*deps*0.5*izmagic;
09667      }
09668 }
09669
09670 #endif /* if defined(WITH_TINKER) */
09671
09672 VPRIVATE void fillcoCoefSpline4(Vpmg *thee) {
09673
09674     Valist *alist;
09675     Vpbe *pbe;
09676     Vatom *atom;
09677     double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr, dist2;
09678     double xlen, ylen, zlen, position[3], itot, stot, ictot, ictot2, sctot;
09679     double irad, dx, dy, dz, epsw, epsp, w2i;
09680     double hx, hy, hzed, *apos, arad, sctot2;
09681     double dx2, dy2, dz2, stot2, itot2, rtot, rtot2, splineWin;
09682     double dist, value, denom, sm, sm2, sm3, sm4, sm5, sm6, sm7;
09683     double e, e2, e3, e4, e5, e6, e7;
```

```
09684     double b, b2, b3, b4, b5, b6, b7;
09685     double c0, c1, c2, c3, c4, c5, c6, c7;
09686     double ic0, ic1, ic2, ic3, ic4, ic5, ic6, ic7;
09687     int i, j, k, nx, ny, nz, iatom;
09688     int imin, imax, jmin, jmax, kmin, kmax;
09689 
09690     VASSERT(thee != VNULL);
09691     splineWin = thee->splineWin;
09692 
09693     /* Get PBE info */
09694     pbe = thee->pbe;
09695     alist = pbe->alist;
09696     irad = Vpbe_getMaxIonRadius(pbe);
09697     ionstr = Vpbe_getBulkIonicStrength(pbe);
09698     epsw = Vpbe_getSolventDiel(pbe);
09699     epsp = Vpbe_getSoluteDiel(pbe);
09700 
09701     /* Mesh info */
09702     nx = thee->pmgp->nx;
09703     ny = thee->pmgp->ny;
09704     nz = thee->pmgp->nz;
09705     hx = thee->pmgp->hx;
09706     hy = thee->pmgp->hy;
09707     hzed = thee->pmgp->hzed;
09708 
09709     /* Define the total domain size */
09710     xlen = thee->pmgp->xlen;
09711     ylen = thee->pmgp->ylen;
09712     zlen = thee->pmgp->zlen;
09713 
09714     /* Define the min/max dimensions */
09715     xmin = thee->pmgp->xcent - (xlen/2.0);
09716     ymin = thee->pmgp->ycent - (ylen/2.0);
09717     zmin = thee->pmgp->zcent - (zlen/2.0);
09718     xmax = thee->pmgp->xcent + (xlen/2.0);
09719     ymax = thee->pmgp->ycent + (ylen/2.0);
09720     zmax = thee->pmgp->zcent + (zlen/2.0);
09721 
09722     /* This is a floating point parameter related to the non-zero nature of the
09723      * bulk ionic strength.  If the ionic strength is greater than zero; this
09724      * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
09725      * Otherwise, this parameter is set to 0.0 */
09726     if (ionstr > VPMGSMALL) ionmask = 1.0;
09727     else ionmask = 0.0;
09728 
09729     /* Reset the kappa, epsx, epsy, and epsz arrays */
09730     for (i=0; i<(nx*ny*nz); i++) {
09731         thee->kappa[i] = 1.0;
09732         thee->epsx[i] = 1.0;
09733         thee->epsy[i] = 1.0;
09734         thee->epsz[i] = 1.0;
09735     }
09736 
09737     /* Loop through the atoms and do assign the dielectric */
09738     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
09739 
09740         atom = Valist_getAtom(alist, iatom);
```

```
09741         apos = Vatom_getPosition(atom);
09742         arad = Vatom_getRadius(atom);
09743 
09744         b = arad - splineWin;
09745         e = arad + splineWin;
09746         e2 = e * e;
09747         e3 = e2 * e;
09748         e4 = e3 * e;
09749         e5 = e4 * e;
09750         e6 = e5 * e;
09751         e7 = e6 * e;
09752         b2 = b * b;
09753         b3 = b2 * b;
09754         b4 = b3 * b;
09755         b5 = b4 * b;
09756         b6 = b5 * b;
09757         b7 = b6 * b;
09758         denom = e7  - 7.0*b*e6 + 21.0*b2*e5 - 35.0*e4*b3
09759               + 35.0*e3*b4 - 21.0*b5*e2  + 7.0*e*b6 - b7;
09760         c0 = b4*(35.0*e3 - 21.0*b*e2 + 7*e*b2 - b3)/denom;
09761         c1 = -140.0*b3*e3/denom;
09762         c2 = 210.0*e2*b2*(e + b)/denom;
09763         c3 = -140.0*e*b*(e2 + 3.0*b*e + b2)/denom;
09764         c4 =  35.0*(e3 + 9.0*b*e2 + + 9.0*e*b2 + b3)/denom;
09765         c5 = -84.0*(e2 + 3.0*b*e + b2)/denom;
09766         c6 =  70.0*(e + b)/denom;
09767         c7 = -20.0/denom;
09768 
09769         b = irad + arad - splineWin;
09770         e = irad + arad + splineWin;
09771         e2 = e * e;
09772         e3 = e2 * e;
09773         e4 = e3 * e;
09774         e5 = e4 * e;
09775         e6 = e5 * e;
09776         e7 = e6 * e;
09777         b2 = b * b;
09778         b3 = b2 * b;
09779         b4 = b3 * b;
09780         b5 = b4 * b;
09781         b6 = b5 * b;
09782         b7 = b6 * b;
09783         denom = e7  - 7.0*b*e6 + 21.0*b2*e5 - 35.0*e4*b3
09784               + 35.0*e3*b4 - 21.0*b5*e2  + 7.0*e*b6 - b7;
09785         ic0 = b4*(35.0*e3 - 21.0*b*e2 + 7*e*b2 - b3)/denom;
09786         ic1 = -140.0*b3*e3/denom;
09787         ic2 = 210.0*e2*b2*(e + b)/denom;
09788         ic3 = -140.0*e*b*(e2 + 3.0*b*e + b2)/denom;
09789         ic4 =  35.0*(e3 + 9.0*b*e2 + + 9.0*e*b2 + b3)/denom;
09790         ic5 = -84.0*(e2 + 3.0*b*e + b2)/denom;
09791         ic6 =  70.0*(e + b)/denom;
09792         ic7 = -20.0/denom;
09793 
09794         /* Make sure we're on the grid */
09795         if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
09796             (apos[1]<=ymin) || (apos[1]>=ymax)  || \
09797             (apos[2]<=zmin) || (apos[2]>=zmax)) {
```

```
09798             if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
09799     (thee->pmgp->bcfl != BCFL_MAP)) {
09800                 Vnm_print(2, "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f,\
09801  %4.3f) is off the mesh (ignoring):\n",
09802                   iatom, apos[0], apos[1], apos[2]);
09803                 Vnm_print(2, "Vpmg_fillco:    xmin = %g, xmax = %g\n",
09804                   xmin, xmax);
09805                 Vnm_print(2, "Vpmg_fillco:    ymin = %g, ymax = %g\n",
09806                   ymin, ymax);
09807                 Vnm_print(2, "Vpmg_fillco:    zmin = %g, zmax = %g\n",
09808                   zmin, zmax);
09809             }
09810             fflush(stderr);
09811
09812         } else if (arad > VPMGSMALL ) { /* if we're on the mesh */
09813
09814             /* Convert the atom position to grid reference frame */
09815             position[0] = apos[0] - xmin;
09816             position[1] = apos[1] - ymin;
09817             position[2] = apos[2] - zmin;
09818
09819             /* MARK ION ACCESSIBILITY AND DIELECTRIC VALUES FOR LATER
09820              * ASSIGNMENT (Steps #1-3) */
09821             itot = irad + arad + splineWin;
09822             itot2 = VSQR(itot);
09823             ictot = VMAX2(0, (irad + arad - splineWin));
09824             ictot2 = VSQR(ictot);
09825             stot = arad + splineWin;
09826             stot2 = VSQR(stot);
09827             sctot = VMAX2(0, (arad - splineWin));
09828             sctot2 = VSQR(sctot);
09829
09830            /* We'll search over grid points which are in the greater of
09831             * these two radii */
09832             rtot = VMAX2(itot, stot);
09833             rtot2 = VMAX2(itot2, stot2);
09834             dx = rtot + 0.5*hx;
09835             dy = rtot + 0.5*hy;
09836             dz = rtot + 0.5*hzed;
09837             imin = VMAX2(0,(int)floor((position[0] - dx)/hx));
09838             imax = VMIN2(nx-1,(int)ceil((position[0] + dx)/hx));
09839             jmin = VMAX2(0,(int)floor((position[1] - dy)/hy));
09840             jmax = VMIN2(ny-1,(int)ceil((position[1] + dy)/hy));
09841             kmin = VMAX2(0,(int)floor((position[2] - dz)/hzed));
09842             kmax = VMIN2(nz-1,(int)ceil((position[2] + dz)/hzed));
09843             for (i=imin; i<=imax; i++) {
09844                 dx2 = VSQR(position[0] - hx*i);
09845                 for (j=jmin; j<=jmax; j++) {
09846                     dy2 = VSQR(position[1] - hy*j);
09847                     for (k=kmin; k<=kmax; k++) {
09848                         dz2 = VSQR(position[2] - k*hzed);
09849
09850                         /* ASSIGN CCF */
09851                         if (thee->kappa[IJK(i,j,k)] > VPMGSMALL) {
09852                             dist2 = dz2 + dy2 + dx2;
09853                             if (dist2 >= itot2) {
09854                                 ;
```

```
09855                           }
09856                           if (dist2 <= ictot2) {
09857                               thee->kappa[IJK(i,j,k)] = 0.0;
09858                           }
09859                           if ((dist2 < itot2) && (dist2 > ictot2)) {
09860                               dist = VSQRT(dist2);
09861                               sm = dist;
09862                               sm2 = dist2;
09863                               sm3 = sm2 * sm;
09864                               sm4 = sm3 * sm;
09865                               sm5 = sm4 * sm;
09866                               sm6 = sm5 * sm;
09867                               sm7 = sm6 * sm;
09868                               value = ic0 + ic1*sm + ic2*sm2 + ic3*sm3
09869                                     + ic4*sm4 + ic5*sm5 + ic6*sm6 + ic7*sm7;
09870                               if (value > 1.0) {
09871                                  value = 1.0;
09872                               } else if (value < 0.0){
09873                                  value = 0.0;
09874                               }
09875                               thee->kappa[IJK(i,j,k)] *= value;
09876                           }
09877                       }
09878
09879                       /* ASSIGN A1CF */
09880                       if (thee->epsx[IJK(i,j,k)] > VPMGSMALL) {
09881                           dist2 = dz2+dy2+VSQR(position[0]-(i+0.5)*hx);
09882                           if (dist2 >= stot2) {
09883                               thee->epsx[IJK(i,j,k)] *= 1.0;
09884                           }
09885                           if (dist2 <= sctot2) {
09886                               thee->epsx[IJK(i,j,k)] = 0.0;
09887                           }
09888                           if ((dist2 > sctot2) && (dist2 < stot2)) {
09889                               dist = VSQRT(dist2);
09890                               sm = dist;
09891                               sm2 = VSQR(sm);
09892                               sm3 = sm2 * sm;
09893                               sm4 = sm3 * sm;
09894                               sm5 = sm4 * sm;
09895                               sm6 = sm5 * sm;
09896                               sm7 = sm6 * sm;
09897                               value = c0 + c1*sm + c2*sm2 + c3*sm3
09898                                     + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
09899                               if (value > 1.0) {
09900                                  value = 1.0;
09901                               } else if (value < 0.0){
09902                                  value = 0.0;
09903                               }
09904                               thee->epsx[IJK(i,j,k)] *= value;
09905                           }
09906                       }
09907
09908                       /* ASSIGN A2CF */
09909                       if (thee->epsy[IJK(i,j,k)] > VPMGSMALL) {
09910                           dist2 = dz2+dx2+VSQR(position[1]-(j+0.5)*hy);
09911                           if (dist2 >= stot2) {
```

```
09912                                 thee->epsy[IJK(i,j,k)] *= 1.0;
09913                             }
09914                             if (dist2 <= sctot2) {
09915                                 thee->epsy[IJK(i,j,k)] = 0.0;
09916                             }
09917                             if ((dist2 > sctot2) && (dist2 < stot2)) {
09918                                 dist = VSQRT(dist2);
09919                                 sm = dist;
09920                                 sm2 = VSQR(sm);
09921                                 sm3 = sm2 * sm;
09922                                 sm4 = sm3 * sm;
09923                                 sm5 = sm4 * sm;
09924                                 sm6 = sm5 * sm;
09925                                 sm7 = sm6 * sm;
09926                                 value = c0 + c1*sm + c2*sm2 + c3*sm3
09927                                       + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
09928                                 if (value > 1.0) {
09929                                    value = 1.0;
09930                                 } else if (value < 0.0){
09931                                    value = 0.0;
09932                                 }
09933                                 thee->epsy[IJK(i,j,k)] *= value;
09934                             }
09935                         }
09936 
09937                         /* ASSIGN A3CF */
09938                         if (thee->epsz[IJK(i,j,k)] > VPMGSMALL) {
09939                             dist2 = dy2+dx2+VSQR(position[2]-(k+0.5)*hzed);
09940                             if (dist2 >= stot2) {
09941                                 thee->epsz[IJK(i,j,k)] *= 1.0;
09942                             }
09943                             if (dist2 <= sctot2) {
09944                                 thee->epsz[IJK(i,j,k)] = 0.0;
09945                             }
09946                             if ((dist2 > sctot2) && (dist2 < stot2)) {
09947                                 dist = VSQRT(dist2);
09948                                 sm = dist;
09949                                 sm2 = dist2;
09950                                 sm3 = sm2 * sm;
09951                                 sm4 = sm3 * sm;
09952                                 sm5 = sm4 * sm;
09953                                 sm6 = sm5 * sm;
09954                                 sm7 = sm6 * sm;
09955                                 value = c0 + c1*sm + c2*sm2 + c3*sm3
09956                                       + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
09957                                 if (value > 1.0) {
09958                                    value = 1.0;
09959                                 } else if (value < 0.0){
09960                                    value = 0.0;
09961                                 }
09962                                 thee->epsz[IJK(i,j,k)] *= value;
09963                             }
09964                         }
09965 
09966 
09967                     } /* k loop */
09968                 } /* j loop */
```

```
09969             } /* i loop */
09970         } /* endif (on the mesh) */
09971     } /* endfor (over all atoms) */
09972
09973     Vnm_print(0, "Vpmg_fillco:  filling coefficient arrays\n");
09974     /* Interpret markings and fill the coefficient arrays */
09975     for (k=0; k<nz; k++) {
09976         for (j=0; j<ny; j++) {
09977             for (i=0; i<nx; i++) {
09978
09979                 thee->kappa[IJK(i,j,k)] = ionmask*thee->kappa[IJK(i,j,k)];
09980                 thee->epsx[IJK(i,j,k)] = (epsw-epsp)*thee->epsx[IJK(i,j,k)]
09981                   + epsp;
09982                 thee->epsy[IJK(i,j,k)] = (epsw-epsp)*thee->epsy[IJK(i,j,k)]
09983                   + epsp;
09984                 thee->epsz[IJK(i,j,k)] = (epsw-epsp)*thee->epsz[IJK(i,j,k)]
09985                   + epsp;
09986
09987             } /* i loop */
09988         } /* j loop */
09989     } /* k loop */
09990
09991 }
09992
09993 VPUBLIC void fillcoPermanentInduced(Vpmg *thee) {
09994
09995     Valist *alist;
09996     Vpbe *pbe;
09997     Vatom *atom;
09998     /* Coversions */
09999     double zmagic, f;
10000     /* Grid */
10001     double xmin, xmax, ymin, ymax, zmin, zmax;
10002     double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;
10003     double hx, hy, hzed, *apos;
10004     /* Multipole */
10005     double charge, *dipole,*quad;
10006     double c,ux,uy,uz,qxx,qyx,qyy,qzx,qzy,qzz,qave;
10007     /* B-spline weights */
10008     double mx,my,mz,dmx,dmy,dmz,d2mx,d2my,d2mz;
10009     double mi,mj,mk;
10010     /* Loop variables */
10011     int i, ii, jj, kk, nx, ny, nz, iatom;
10012     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
10013
10014     VASSERT(thee != VNULL);
10015
10016     /* Get PBE info */
10017     pbe = thee->pbe;
10018     alist = pbe->alist;
10019     zmagic = Vpbe_getZmagic(pbe);
10020
10021     /* Mesh info */
10022     nx = thee->pmgp->nx;
10023     ny = thee->pmgp->ny;
10024     nz = thee->pmgp->nz;
10025     hx = thee->pmgp->hx;
```

```
10026     hy = thee->pmgp->hy;
10027     hzed = thee->pmgp->hzed;
10028
10029     /* Conversion */
10030     f = zmagic/(hx*hy*hzed);
10031
10032     /* Define the total domain size */
10033     xlen = thee->pmgp->xlen;
10034     ylen = thee->pmgp->ylen;
10035     zlen = thee->pmgp->zlen;
10036
10037     /* Define the min/max dimensions */
10038     xmin = thee->pmgp->xcent - (xlen/2.0);
10039     ymin = thee->pmgp->ycent - (ylen/2.0);
10040     zmin = thee->pmgp->zcent - (zlen/2.0);
10041     xmax = thee->pmgp->xcent + (xlen/2.0);
10042     ymax = thee->pmgp->ycent + (ylen/2.0);
10043     zmax = thee->pmgp->zcent + (zlen/2.0);
10044
10045     /* Fill in the source term (permanent atomic multipoles
10046        and induced dipoles) */
10047     Vnm_print(0, "fillcoPermanentInduced:  filling in source term.\n");
10048     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
10049
10050         atom = Valist_getAtom(alist, iatom);
10051         apos = Vatom_getPosition(atom);
10052
10053         c = Vatom_getCharge(atom)*f;
10054
10055 #if defined(WITH_TINKER)
10056         dipole = Vatom_getDipole(atom);
10057         ux = dipole[0]/hx*f;
10058         uy = dipole[1]/hy*f;
10059         uz = dipole[2]/hzed*f;
10060         dipole = Vatom_getInducedDipole(atom);
10061         ux = ux + dipole[0]/hx*f;
10062         uy = uy + dipole[1]/hy*f;
10063         uz = uz + dipole[2]/hzed*f;
10064         quad = Vatom_getQuadrupole(atom);
10065         qxx = (1.0/3.0)*quad[0]/(hx*hx)*f;
10066         qyx = (2.0/3.0)*quad[3]/(hx*hy)*f;
10067         qyy = (1.0/3.0)*quad[4]/(hy*hy)*f;
10068         qzx = (2.0/3.0)*quad[6]/(hzed*hx)*f;
10069         qzy = (2.0/3.0)*quad[7]/(hzed*hy)*f;
10070         qzz = (1.0/3.0)*quad[8]/(hzed*hzed)*f;
10071 #else
10072         ux = 0.0;
10073         uy = 0.0;
10074         uz = 0.0;
10075         qxx = 0.0;
10076         qyx = 0.0;
10077         qyy = 0.0;
10078         qzx = 0.0;
10079         qzy = 0.0;
10080         qzz = 0.0;
10081 #endif /* if defined(WITH_TINKER) */
10082
```

```
10083         /* Make sure we're on the grid */
10084         if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx))  || \
10085             (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy))  || \
10086             (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
10087             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]);

10088             Vnm_print(2, "fillcoPermanentMultipole: xmin = %g, xmax = %g\n", xmin
      , xmax);
10089             Vnm_print(2, "fillcoPermanentMultipole: ymin = %g, ymax = %g\n", ymin
      , ymax);
10090             Vnm_print(2, "fillcoPermanentMultipole: zmin = %g, zmax = %g\n", zmin
      , zmax);
10091             fflush(stderr);
10092         } else {
10093
10094             /* Convert the atom position to grid reference frame */
10095             position[0] = apos[0] - xmin;
10096             position[1] = apos[1] - ymin;
10097             position[2] = apos[2] - zmin;
10098
10099             /* Figure out which vertices we're next to */
10100             ifloat = position[0]/hx;
10101             jfloat = position[1]/hy;
10102             kfloat = position[2]/hzed;
10103
10104             ip1   = (int)ceil(ifloat);
10105             ip2   = ip1 + 2;
10106             im1   = (int)floor(ifloat);
10107             im2   = im1 - 2;
10108             jp1   = (int)ceil(jfloat);
10109             jp2   = jp1 + 2;
10110             jm1   = (int)floor(jfloat);
10111             jm2   = jm1 - 2;
10112             kp1   = (int)ceil(kfloat);
10113             kp2   = kp1 + 2;
10114             km1   = (int)floor(kfloat);
10115             km2   = km1 - 2;
10116
10117             /* This step shouldn't be necessary, but it saves nasty debugging
10118              * later on if something goes wrong */
10119             ip2 = VMIN2(ip2,nx-1);
10120             ip1 = VMIN2(ip1,nx-1);
10121             im1 = VMAX2(im1,0);
10122             im2 = VMAX2(im2,0);
10123             jp2 = VMIN2(jp2,ny-1);
10124             jp1 = VMIN2(jp1,ny-1);
10125             jm1 = VMAX2(jm1,0);
10126             jm2 = VMAX2(jm2,0);
10127             kp2 = VMIN2(kp2,nz-1);
10128             kp1 = VMIN2(kp1,nz-1);
10129             km1 = VMAX2(km1,0);
10130             km2 = VMAX2(km2,0);
10131
10132             /* Now assign fractions of the charge to the nearby verts */
10133             for (ii=im2; ii<=ip2; ii++) {
10134                 mi = VFCHI4(ii,ifloat);
```

```
10135                 mx = bspline4(mi);
10136                 dmx = dbspline4(mi);
10137                 d2mx = d2bspline4(mi);
10138                 for (jj=jm2; jj<=jp2; jj++) {
10139                     mj = VFCHI4(jj,jfloat);
10140                     my = bspline4(mj);
10141                     dmy = dbspline4(mj);
10142                     d2my = d2bspline4(mj);
10143                     for (kk=km2; kk<=kp2; kk++) {
10144                         mk = VFCHI4(kk,kfloat);
10145                         mz = bspline4(mk);
10146                         dmz = dbspline4(mk);
10147                         d2mz = d2bspline4(mk);
10148                         charge = mx*my*mz*c -
10149                          dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz +
10150                          d2mx*my*mz*qxx +
10151                          dmx*dmy*mz*qyx + mx*d2my*mz*qyy +
10152                          dmx*my*dmz*qzx + mx*dmy*dmz*qzy + mx*my*d2mz*qzz;
10153                         thee->charge[IJK(ii,jj,kk)] += charge;
10154
10155                     }
10156                 }
10157             }
10158         } /* endif (on the mesh) */
10159
10160     } /* endfor (each atom) */
10161 }
10162
10163 VPRIVATE void fillcoCoefSpline3(Vpmg *thee) {
10164
10165     Valist *alist;
10166     Vpbe *pbe;
10167     Vatom *atom;
10168     double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr, dist2;
10169     double xlen, ylen, zlen, position[3], itot, stot, ictot, ictot2, sctot;
10170     double irad, dx, dy, dz, epsw, epsp, w2i;
10171     double hx, hy, hzed, *apos, arad, sctot2;
10172     double dx2, dy2, dz2, stot2, itot2, rtot, rtot2, splineWin;
10173     double dist, value, denom, sm, sm2, sm3, sm4, sm5;
10174     double e, e2, e3, e4, e5;
10175     double b, b2, b3, b4, b5;
10176     double c0, c1, c2, c3, c4, c5;
10177     double ic0, ic1, ic2, ic3, ic4, ic5;
10178     int i, j, k, nx, ny, nz, iatom;
10179     int imin, imax, jmin, jmax, kmin, kmax;
10180
10181     VASSERT(thee != VNULL);
10182     splineWin = thee->splineWin;
10183
10184     /* Get PBE info */
10185     pbe = thee->pbe;
10186     alist = pbe->alist;
10187     irad = Vpbe_getMaxIonRadius(pbe);
10188     ionstr = Vpbe_getBulkIonicStrength(pbe);
10189     epsw = Vpbe_getSolventDiel(pbe);
10190     epsp = Vpbe_getSoluteDiel(pbe);
10191
```

```
10192     /* Mesh info */
10193     nx = thee->pmgp->nx;
10194     ny = thee->pmgp->ny;
10195     nz = thee->pmgp->nz;
10196     hx = thee->pmgp->hx;
10197     hy = thee->pmgp->hy;
10198     hzed = thee->pmgp->hzed;
10199
10200     /* Define the total domain size */
10201     xlen = thee->pmgp->xlen;
10202     ylen = thee->pmgp->ylen;
10203     zlen = thee->pmgp->zlen;
10204
10205     /* Define the min/max dimensions */
10206     xmin = thee->pmgp->xcent - (xlen/2.0);
10207     ymin = thee->pmgp->ycent - (ylen/2.0);
10208     zmin = thee->pmgp->zcent - (zlen/2.0);
10209     xmax = thee->pmgp->xcent + (xlen/2.0);
10210     ymax = thee->pmgp->ycent + (ylen/2.0);
10211     zmax = thee->pmgp->zcent + (zlen/2.0);
10212
10213     /* This is a floating point parameter related to the non-zero nature of the
10214      * bulk ionic strength.  If the ionic strength is greater than zero; this
10215      * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
10216      * Otherwise, this parameter is set to 0.0 */
10217     if (ionstr > VPMGSMALL) ionmask = 1.0;
10218     else ionmask = 0.0;
10219
10220     /* Reset the kappa, epsx, epsy, and epsz arrays */
10221     for (i=0; i<(nx*ny*nz); i++) {
10222         thee->kappa[i] = 1.0;
10223         thee->epsx[i] = 1.0;
10224         thee->epsy[i] = 1.0;
10225         thee->epsz[i] = 1.0;
10226     }
10227
10228     /* Loop through the atoms and do assign the dielectric */
10229     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
10230
10231         atom = Valist_getAtom(alist, iatom);
10232         apos = Vatom_getPosition(atom);
10233         arad = Vatom_getRadius(atom);
10234
10235         b = arad - splineWin;
10236         e = arad + splineWin;
10237         e2 = e * e;
10238         e3 = e2 * e;
10239         e4 = e3 * e;
10240         e5 = e4 * e;
10241         b2 = b * b;
10242         b3 = b2 * b;
10243         b4 = b3 * b;
10244         b5 = b4 * b;
10245         denom = pow((e - b), 5.0);
10246         c0 = -10.0*e2*b3 + 5.0*e*b4 - b5;
10247         c1 = 30.0*e2*b2;
10248         c2 = -30.0*(e2*b + e*b2);
```

```
10249         c3 = 10.0*(e2 + 4.0*e*b + b2);
10250         c4 = -15.0*(e + b);
10251         c5 = 6;
10252         c0 = c0/denom;
10253         c1 = c1/denom;
10254         c2 = c2/denom;
10255         c3 = c3/denom;
10256         c4 = c4/denom;
10257         c5 = c5/denom;
10258 
10259         b = irad + arad - splineWin;
10260         e = irad + arad + splineWin;
10261         e2 = e * e;
10262         e3 = e2 * e;
10263         e4 = e3 * e;
10264         e5 = e4 * e;
10265         b2 = b * b;
10266         b3 = b2 * b;
10267         b4 = b3 * b;
10268         b5 = b4 * b;
10269         denom = pow((e - b), 5.0);
10270         ic0 = -10.0*e2*b3 + 5.0*e*b4 - b5;
10271         ic1 = 30.0*e2*b2;
10272         ic2 = -30.0*(e2*b + e*b2);
10273         ic3 = 10.0*(e2 + 4.0*e*b + b2);
10274         ic4 = -15.0*(e + b);
10275         ic5 = 6;
10276         ic0 = c0/denom;
10277         ic1 = c1/denom;
10278         ic2 = c2/denom;
10279         ic3 = c3/denom;
10280         ic4 = c4/denom;
10281         ic5 = c5/denom;
10282 
10283         /* Make sure we're on the grid */
10284         if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
10285             (apos[1]<=ymin) || (apos[1]>=ymax)  || \
10286             (apos[2]<=zmin) || (apos[2]>=zmax)) {
10287    if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
10288     (thee->pmgp->bcfl != BCFL_MAP)) {
10289                 Vnm_print(2, "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f,\
10290  %4.3f) is off the mesh (ignoring):\n",
10291                   iatom, apos[0], apos[1], apos[2]);
10292                 Vnm_print(2, "Vpmg_fillco:    xmin = %g, xmax = %g\n",
10293                   xmin, xmax);
10294                 Vnm_print(2, "Vpmg_fillco:    ymin = %g, ymax = %g\n",
10295                   ymin, ymax);
10296                 Vnm_print(2, "Vpmg_fillco:    zmin = %g, zmax = %g\n",
10297                   zmin, zmax);
10298             }
10299             fflush(stderr);
10300 
10301         } else if (arad > VPMGSMALL ) { /* if we're on the mesh */
10302 
10303             /* Convert the atom position to grid reference frame */
10304             position[0] = apos[0] - xmin;
10305             position[1] = apos[1] - ymin;
```

```
10306             position[2] = apos[2] - zmin;
10307
10308             /* MARK ION ACCESSIBILITY AND DIELECTRIC VALUES FOR LATER
10309              * ASSIGNMENT (Steps #1-3) */
10310             itot = irad + arad + splineWin;
10311             itot2 = VSQR(itot);
10312             ictot = VMAX2(0, (irad + arad - splineWin));
10313             ictot2 = VSQR(ictot);
10314             stot = arad + splineWin;
10315             stot2 = VSQR(stot);
10316             sctot = VMAX2(0, (arad - splineWin));
10317             sctot2 = VSQR(sctot);
10318
10319            /* We'll search over grid points which are in the greater of
10320              * these two radii */
10321             rtot = VMAX2(itot, stot);
10322             rtot2 = VMAX2(itot2, stot2);
10323             dx = rtot + 0.5*hx;
10324             dy = rtot + 0.5*hy;
10325             dz = rtot + 0.5*hzed;
10326             imin = VMAX2(0,(int)floor((position[0] - dx)/hx));
10327             imax = VMIN2(nx-1,(int)ceil((position[0] + dx)/hx));
10328             jmin = VMAX2(0,(int)floor((position[1] - dy)/hy));
10329             jmax = VMIN2(ny-1,(int)ceil((position[1] + dy)/hy));
10330             kmin = VMAX2(0,(int)floor((position[2] - dz)/hzed));
10331             kmax = VMIN2(nz-1,(int)ceil((position[2] + dz)/hzed));
10332             for (i=imin; i<=imax; i++) {
10333                 dx2 = VSQR(position[0] - hx*i);
10334                 for (j=jmin; j<=jmax; j++) {
10335                     dy2 = VSQR(position[1] - hy*j);
10336                     for (k=kmin; k<=kmax; k++) {
10337                         dz2 = VSQR(position[2] - k*hzed);
10338
10339                         /* ASSIGN CCF */
10340                         if (thee->kappa[IJK(i,j,k)] > VPMGSMALL) {
10341                             dist2 = dz2 + dy2 + dx2;
10342                             if (dist2 >= itot2) {
10343                                 ;
10344                             }
10345                             if (dist2 <= ictot2) {
10346                                 thee->kappa[IJK(i,j,k)] = 0.0;
10347                             }
10348                             if ((dist2 < itot2) && (dist2 > ictot2)) {
10349                                 dist = VSQRT(dist2);
10350                                 sm = dist;
10351                                 sm2 = dist2;
10352                                 sm3 = sm2 * sm;
10353                                 sm4 = sm3 * sm;
10354                                 sm5 = sm4 * sm;
10355                                 value = ic0 + ic1*sm + ic2*sm2 + ic3*sm3
10356                                       + ic4*sm4 + ic5*sm5;
10357                                 if (value > 1.0) {
10358                                    value = 1.0;
10359                                 } else if (value < 0.0){
10360                                    value = 0.0;
10361                                 }
10362                                 thee->kappa[IJK(i,j,k)] *= value;
```

```
10363                           }
10364                       }
10365
10366                       /* ASSIGN A1CF */
10367                       if (thee->epsx[IJK(i,j,k)] > VPMGSMALL) {
10368                           dist2 = dz2+dy2+VSQR(position[0]-(i+0.5)*hx);
10369                           if (dist2 >= stot2) {
10370                               thee->epsx[IJK(i,j,k)] *= 1.0;
10371                           }
10372                           if (dist2 <= sctot2) {
10373                               thee->epsx[IJK(i,j,k)] = 0.0;
10374                           }
10375                           if ((dist2 > sctot2) && (dist2 < stot2)) {
10376                               dist = VSQRT(dist2);
10377                               sm = dist;
10378                               sm2 = VSQR(sm);
10379                               sm3 = sm2 * sm;
10380                               sm4 = sm3 * sm;
10381                               sm5 = sm4 * sm;
10382                               value = c0 + c1*sm + c2*sm2 + c3*sm3
10383                                     + c4*sm4 + c5*sm5;
10384                               if (value > 1.0) {
10385                                  value = 1.0;
10386                               } else if (value < 0.0){
10387                                  value = 0.0;
10388                               }
10389                               thee->epsx[IJK(i,j,k)] *= value;
10390                           }
10391                       }
10392
10393                       /* ASSIGN A2CF */
10394                       if (thee->epsy[IJK(i,j,k)] > VPMGSMALL) {
10395                           dist2 = dz2+dx2+VSQR(position[1]-(j+0.5)*hy);
10396                           if (dist2 >= stot2) {
10397                               thee->epsy[IJK(i,j,k)] *= 1.0;
10398                           }
10399                           if (dist2 <= sctot2) {
10400                               thee->epsy[IJK(i,j,k)] = 0.0;
10401                           }
10402                           if ((dist2 > sctot2) && (dist2 < stot2)) {
10403                               dist = VSQRT(dist2);
10404                               sm = dist;
10405                               sm2 = VSQR(sm);
10406                               sm3 = sm2 * sm;
10407                               sm4 = sm3 * sm;
10408                               sm5 = sm4 * sm;
10409                               value = c0 + c1*sm + c2*sm2 + c3*sm3
10410                                     + c4*sm4 + c5*sm5;
10411                               if (value > 1.0) {
10412                                  value = 1.0;
10413                               } else if (value < 0.0){
10414                                  value = 0.0;
10415                               }
10416                               thee->epsy[IJK(i,j,k)] *= value;
10417                           }
10418                       }
10419
```

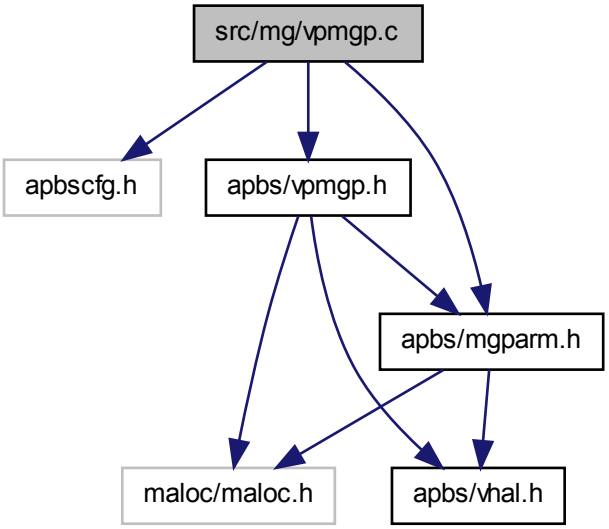
```
10420                         /* ASSIGN A3CF */
10421                         if (thee->epsz[IJK(i,j,k)] > VPMGSMALL) {
10422                             dist2 = dy2+dx2+VSQR(position[2]-(k+0.5)*hzed);
10423                             if (dist2 >= stot2) {
10424                                 thee->epsz[IJK(i,j,k)] *= 1.0;
10425                             }
10426                             if (dist2 <= sctot2) {
10427                                 thee->epsz[IJK(i,j,k)] = 0.0;
10428                             }
10429                             if ((dist2 > sctot2) && (dist2 < stot2)) {
10430                                 dist = VSQRT(dist2);
10431                                 sm = dist;
10432                                 sm2 = dist2;
10433                                 sm3 = sm2 * sm;
10434                                 sm4 = sm3 * sm;
10435                                 sm5 = sm4 * sm;
10436                                 value = c0 + c1*sm + c2*sm2 + c3*sm3
10437                                       + c4*sm4 + c5*sm5;
10438                                 if (value > 1.0) {
10439                                    value = 1.0;
10440                                 } else if (value < 0.0){
10441                                    value = 0.0;
10442                                 }
10443                                 thee->epsz[IJK(i,j,k)] *= value;
10444                             }
10445                         }
10446 
10447 
10448                     } /* k loop */
10449                 } /* j loop */
10450             } /* i loop */
10451         } /* endif (on the mesh) */
10452     } /* endfor (over all atoms) */
10453 
10454     Vnm_print(0, "Vpmg_fillco:  filling coefficient arrays\n");
10455     /* Interpret markings and fill the coefficient arrays */
10456     for (k=0; k<nz; k++) {
10457         for (j=0; j<ny; j++) {
10458             for (i=0; i<nx; i++) {
10459 
10460                 thee->kappa[IJK(i,j,k)] = ionmask*thee->kappa[IJK(i,j,k)];
10461                 thee->epsx[IJK(i,j,k)] = (epsw-epsp)*thee->epsx[IJK(i,j,k)]
10462                   + epsp;
10463                 thee->epsy[IJK(i,j,k)] = (epsw-epsp)*thee->epsy[IJK(i,j,k)]
10464                   + epsp;
10465                 thee->epsz[IJK(i,j,k)] = (epsw-epsp)*thee->epsz[IJK(i,j,k)]
10466                   + epsp;
10467 
10468             } /* i loop */
10469         } /* j loop */
10470     } /* k loop */
10471 
10472 }
10473 
```

10.95 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.

10.95.1 Detailed Description

Class Vpmgp methods.

Author

Nathan Baker

Version

Id:

vpmgp.c 1605 2010-09-13 15:12:09Z yhuang01

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-
* 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 Washington University in St. Louis 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.

10.96 src/mg/vpmgp.c

```
00001
00049 #include "apbscfg.h"
00050 #include "apbs/vpmgp.h"
00051 #include "apbs/mgparm.h"
00052
00053 VEMBED(rcsid="$Id: vpmgp.c 1605 2010-09-13 15:12:09Z yhuang01 $")
00054
00055 /* ///////////////////////////////////////////////////////////////////////////
00056 // Class Vpmgp: Inlineable methods
00058 #if !defined(VINLINE_VACC)
00059 #endif /* if !defined(VINLINE_VACC) */
00060
00061 /* ///////////////////////////////////////////////////////////////////////////
00062 // Class Vpmgp: Non-inlineable methods
00064
00065 /* ///////////////////////////////////////////////////////////////////////////
00066 // Routine:  Vpmgp_ctor
00067 //
00068 // Author:   Nathan Baker
00070 VPUBLIC Vpmgp* Vpmgp_ctor(MGparm *mgparm) {
00071
00072     Vpmgp *thee = VNULL;
00073
00074     /* Set up the structure */
00075     thee = Vmem_malloc(VNULL, 1, sizeof(Vpmgp) );
00076     VASSERT( thee != VNULL);
00077     VASSERT(Vpmgp_ctor2(thee,mgparm));
00078
```

```
00079     return thee;
00080 }
00081
00082 /* ///////////////////////////////////////////////////////////////////////////
00083 // Routine:  Vpmgp_ctor2
00084 //
00085 // Author:   Nathan Baker
00087 VPUBLIC int Vpmgp_ctor2(Vpmgp *thee,MGparm *mgparm) {
00088
00089  /* Specified parameters */
00090     thee->nx = mgparm->dime[0];
00091     thee->ny = mgparm->dime[1];
00092     thee->nz = mgparm->dime[2];
00093     thee->hx = mgparm->grid[0];
00094     thee->hy = mgparm->grid[1];
00095     thee->hzed = mgparm->grid[2];
00096     thee->xlen = ((double)(mgparm->dime[0]-1))*mgparm->grid[0];
00097     thee->ylen = ((double)(mgparm->dime[1]-1))*mgparm->grid[1];
00098     thee->zlen = ((double)(mgparm->dime[2]-1))*mgparm->grid[2];
00099     thee->nlev = mgparm->nlev;
00100
00101     thee->nonlin = mgparm->nonlintype;
00102  thee->meth = mgparm->method;
00103
00104 #ifdef DEBUG_MAC_OSX_OCL
00105 #include "mach_chud.h"
00106  if(kOpenCLAvailable)
00107   thee->meth = 4;
00108 #endif
00109
00110     if (thee->nonlin == NONLIN_LPBE) thee->ipkey = IPKEY_LPBE; /* LPBE case */
00111  else if(thee->nonlin == NONLIN_SMPBE) thee->ipkey = IPKEY_SMPBE; /* SMPBE case *
     /
00112     else thee->ipkey = IPKEY_NPBE; /* NPBE standard case */
00113
00114     /* Default parameters */
00115     if (mgparm->setetol) { /* If etol is set by the user in APBS input file, then
      use this custom-defined etol */
00116         thee->errtol = mgparm->etol;
00117         Vnm_print(1, "  Error tolerance (etol) is now set to user-defined \
00118 value: %g \n", thee->errtol);
00119         Vnm_print(0, "Error tolerance (etol) is now set to user-defined \
00120 value: %g \n", thee->errtol);
00121     } else thee->errtol = 1.0e-6;   /* Here are a few comments.  Mike had this se
     t to
00122   * 1e-9; convential wisdom sets this at 1e-6 for
00123   * the PBE; Ray Luo sets this at 1e-3 for his
00124   * accelerated PBE (for dynamics, etc.) */
00125     thee->itmax = 200;
00126     thee->istop = 1;
00127     thee->iinfo = 1;         /* I'd recommend either 1 (for debugging LPBE) or 2
     (for debugging NPBE), higher values give too much output */
00128
00129     thee->bcfl = BCFL_SDH;
00130     thee->key = 0;
00131     thee->iperf = 0;
00132  thee->mgcoar = 2;
```

```
00133  thee->mgkey = 0;
00134  thee->nu1 = 2;
00135  thee->nu2 = 2;
00136  thee->mgprol = 0;
00137  thee->mgdisc = 0;
00138  thee->omegal = 19.4e-1;
00139  thee->omegan = 9.0e-1;
00140  thee->ipcon = 3;
00141  thee->irite = 8;
00142  thee->xcent = 0.0;
00143  thee->ycent = 0.0;
00144  thee->zcent = 0.0;
00145
00146  /* Default value for all APBS runs */
00147  thee->mgsmoo = 1;
00148     if (thee->nonlin == NONLIN_NPBE || thee->nonlin == NONLIN_SMPBE) {
00149   /* SMPBE Added - SMPBE needs to mimic NPBE */
00150   Vnm_print(0, "Vpmp_ctor2:  Using meth = 1, mgsolv = 0\n");
00151         thee->mgsolv = 0;
00152  } else {
00153   /* Most rigorous (good for testing) */
00154   Vnm_print(0, "Vpmp_ctor2:  Using meth = 2, mgsolv = 1\n");
00155         thee->mgsolv = 1;
00156     }
00157
00158  /* TEMPORARY USEAQUA */
00159  /* If we are using aqua, our solution method is either VSOL_CGMGAqua or VSOL_New
      tonAqua
00160   * so we need to temporarily override the mgsolve value and set it to 0
00161   */
00162  if(mgparm->useAqua == 1) thee->mgsolv = 0;
00163
00164  return 1;
00165 }
00166
00167 /* ///////////////////////////////////////////////////////////////////////////
00168 // Routine:  Vpmgp_dtor
00169 //
00170 // Author:   Nathan Baker
00172 VPUBLIC void Vpmgp_dtor(Vpmgp **thee) {
00173
00174     if ((*thee) != VNULL) {
00175         Vpmgp_dtor2(*thee);
00176         Vmem_free(VNULL, 1, sizeof(Vpmgp), (void **)thee);
00177         (*thee) = VNULL;
00178     }
00179
00180 }
00181
00182 /* ///////////////////////////////////////////////////////////////////////////
00183 // Routine:  Vpmgp_dtor2
00184 //
00185 // Author:   Nathan Baker
00187 VPUBLIC void Vpmgp_dtor2(Vpmgp *thee) { ; }
00188
00189
00190 VPUBLIC void Vpmgp_size(
```

```
00191  Vpmgp *thee
00192  )
00193 {
00194
00195  int num_nf = 0;
00196  int num_narr = 2;
00197  int num_narrc = 27;
00198  int nxf, nyf, nzf, level, num_nf_oper, num_narrc_oper, n_band, nc_band, num_band
       , iretot;
00199
00200  thee->nf = thee->nx * thee->ny * thee->nz;
00201  thee->narr = thee->nf;
00202  nxf = thee->nx;
00203  nyf = thee->ny;
00204  nzf = thee->nz;
00205  thee->nxc = thee->nx;
00206  thee->nyc = thee->ny;
00207  thee->nzc = thee->nz;
00208
00209  for (level=2; level<=thee->nlev; level++) {
00210   Vpmgp_makeCoarse(1, nxf, nyf, nzf, &(thee->nxc), &(thee->nyc), &(thee->nzc)); /
       * NAB TO-DO -- implement this function and check which variables need to be passe
       d by reference... */
00211   nxf = thee->nxc;
00212   nyf = thee->nyc;
00213   nzf = thee->nzc;
00214   thee->narr = thee->narr + (nxf * nyf * nzf);
00215  }
00216
00217  thee->nc = thee->nxc * thee->nyc * thee->nzc;
00218  thee->narrc = thee->narr - thee->nf;
00219
00220  /* Box or FEM discretization on fine grid? */
00221  switch (thee->mgdisc) { /* NAB TO-DO:  This needs to be changed into an enumerat
       ion */
00222  case 0:
00223   num_nf_oper = 4;
00224   break;
00225  case 1:
00226   num_nf_oper = 14;
00227   break;
00228  default:
00229   Vnm_print(2, "Vpmgp_size:  Invalid mgdisc value (%d)!\n", thee->mgdisc);
00230   VASSERT(0);
00231  }
00232
00233  /* Galerkin or standard coarsening? */
00234  switch (thee->mgcoar) { /* NAB TO-DO:  This needs to be changed into an enumerat
       ion */
00235  case 0:
00236   if (thee->mgdisc != 0) {
00237    Vnm_print(2, "Vpmgp_size:  Invalid mgcoar value (%d); must be used with mgdisc
       0!\n", thee->mgcoar);
00238    VASSERT(0);
00239   }
00240   num_narrc_oper = 4;
00241   break;
```

```
00242  case 1:
00243   if (thee->mgdisc != 0) {
00244    Vnm_print(2, "Vpmgp_size:  Invalid mgcoar value (%d); must be used with mgdisc
       0!\n", thee->mgcoar);
00245    VASSERT(0);
00246   }
00247   num_narrc_oper = 14;
00248   break;
00249  case 2:
00250   num_narrc_oper = 14;
00251   break;
00252  default:
00253   Vnm_print(2, "Vpmgp_size:  Invalid mgcoar value (%d)!\n", thee->mgcoar);
00254   VASSERT(0);
00255  }
00256
00257  /* LINPACK storage on coarse grid */
00258  switch (thee->mgsolv) { /* NAB TO-DO:  This needs to be changed into an enumerat
      ion */
00259  case 0:
00260   n_band = 0;
00261   break;
00262  case 1:
00263   if ( ( (thee->mgcoar == 0) || (thee->mgcoar == 1)) && (thee->mgdisc == 0) ) {
00264    num_band = 1 + (thee->nxc-2)*(thee->nyc-2);
00265   } else {
00266    num_band = 1 + (thee->nxc-2)*(thee->nyc-2) + (thee->nxc-2) + 1;
00267   }
00268   nc_band = (thee->nxc-2)*(thee->nyc-2)*(thee->nzc-2);
00269   n_band  = nc_band * num_band;
00270   break;
00271  default:
00272   Vnm_print(2, "Vpmgp_size:  Invalid mgsolv value (%d)!\n", thee->mgsolv);
00273   VASSERT(0);
00274  }
00275
00276  /* Real storage parameters */
00277  thee->n_rpc = 100*(thee->nlev+1);
00278
00279  /* Resulting total required for real storage */
00280  thee->nrwk = num_narr*thee->narr + (num_nf + num_nf_oper)*thee->nf + (num_narrc
      + num_narrc_oper)*thee->narrc + n_band + thee->n_rpc;
00281
00282  /* Integer storage parameters */
00283  thee->n_iz = 50*(thee->nlev+1);
00284  thee->n_ipc = 100*(thee->nlev+1);
00285  thee->niwk = thee->n_iz + thee->n_ipc;
00286 }
00287
00288 VPRIVATE int coarsenThis(nOld) {
00289
00290  int nOut;
00291
00292  nOut = (nOld - 1) / 2 + 1;
00293
00294  if (((nOut-1)*2) != (nOld-1)) {
00295   Vnm_print(2, "Vpmgp_makeCoarse:  Warning!  The grid dimensions you have chosen
```

```
      are not consistent with the nlev you have specified!\n");
00296   Vnm_print(2, "Vpmgp_makeCoarse:  This calculation will only work if you are run
      ning with mg-dummy type.\n");
00297  }
00298  if (nOut < 1) {
00299   Vnm_print(2, "D'oh!  You coarsened the grid below zero!  How did you do that?\n
      ");
00300   VASSERT(0);
00301  }
00302
00303  return nOut;
00304 }
00305
00306 VPUBLIC void Vpmgp_makeCoarse(
00307  int numLevel,
00308  int nxOld,
00309  int nyOld,
00310  int nzOld,
00311  int *nxNew,
00312  int *nyNew,
00313  int *nzNew
00314  )
00315 {
00316  int nxtmp, nytmp, nztmp, iLevel;
00317
00318  for (iLevel=0; iLevel<numLevel; iLevel++) {
00319   nxtmp = *nxNew;
00320   nytmp = *nyNew;
00321   nztmp = *nzNew;
00322   *nxNew = coarsenThis(nxtmp);
00323   *nyNew = coarsenThis(nytmp);
00324   *nzNew = coarsenThis(nztmp);
00325  }
00326
00327
00328 }
```

Index