

APBS

1.4.1-binary

Generated by Doxygen 1.8.7

Wed Jul 16 2014 19:04:41

Contents

1	APBS Programmers Guide	1
1.1	Table of Contents	1
1.2	License	1
1.3	Programming Style	2
1.4	Application programming interface documentation	3
2	Deprecated List	5
3	Bug List	7
4	Module Index	11
4.1	Modules	11
5	Hierarchical Index	13
5.1	Class Hierarchy	13
6	Data Structure Index	15
6.1	Data Structures	15
7	File Index	17
7.1	File List	17
8	Module Documentation	21
8.1	dependencies	21
8.2	Vcsm class	22
8.2.1	Detailed Description	23
8.2.2	Function Documentation	23
8.2.2.1	Gem_setExternalUpdateFunction	23
8.2.2.2	Vcsm_ctor	23
8.2.2.3	Vcsm_ctor2	24
8.2.2.4	Vcsm_dtor	24

8.2.2.5	Vcsm_dtor2	24
8.2.2.6	Vcsm_getAtom	25
8.2.2.7	Vcsm_getAtomIndex	25
8.2.2.8	Vcsm_getNumberAtoms	25
8.2.2.9	Vcsm_getNumberSimplices	26
8.2.2.10	Vcsm_getSimplex	26
8.2.2.11	Vcsm_getSimplexIndex	26
8.2.2.12	Vcsm_getValist	27
8.2.2.13	Vcsm_init	27
8.2.2.14	Vcsm_memChk	27
8.2.2.15	Vcsm_update	28
8.3	Vfetc class	29
8.3.1	Detailed Description	32
8.3.2	Enumeration Type Documentation	33
8.3.2.1	eVfetc_GuessType	33
8.3.2.2	eVfetc_LsolvType	33
8.3.2.3	eVfetc_MeshLoad	33
8.3.2.4	eVfetc_NsolvType	33
8.3.2.5	eVfetc_PrecType	34
8.3.3	Function Documentation	34
8.3.3.1	Bmat_printHB	34
8.3.3.2	Vfetc_ctor	35
8.3.3.3	Vfetc_ctor2	35
8.3.3.4	Vfetc_dqmEnergy	35
8.3.3.5	Vfetc_dtor	36
8.3.3.6	Vfetc_dtor2	36
8.3.3.7	Vfetc_dumpLocalVar	37
8.3.3.8	Vfetc_energy	37
8.3.3.9	Vfetc_externalUpdateFunction	37
8.3.3.10	Vfetc_fillArray	38
8.3.3.11	Vfetc_genCube	38
8.3.3.12	Vfetc_getAM	39
8.3.3.13	Vfetc_getAtomColor	39
8.3.3.14	Vfetc_getGem	39
8.3.3.15	Vfetc_getSolution	40
8.3.3.16	Vfetc_getVcsm	40
8.3.3.17	Vfetc_getVpbe	41

8.3.3.18	Vfetk_loadGem	41
8.3.3.19	Vfetk_loadMesh	41
8.3.3.20	Vfetk_memChk	42
8.3.3.21	Vfetk_PDE_bisectEdge	42
8.3.3.22	Vfetk_PDE_ctor	42
8.3.3.23	Vfetk_PDE_ctor2	43
8.3.3.24	Vfetk_PDE_delta	43
8.3.3.25	Vfetk_PDE_DFu_wv	44
8.3.3.26	Vfetk_PDE_dtor	44
8.3.3.27	Vfetk_PDE_dtor2	44
8.3.3.28	Vfetk_PDE_Fu	45
8.3.3.29	Vfetk_PDE_Fu_v	45
8.3.3.30	Vfetk_PDE_initAssemble	46
8.3.3.31	Vfetk_PDE_initElement	46
8.3.3.32	Vfetk_PDE_initFace	46
8.3.3.33	Vfetk_PDE_initPoint	47
8.3.3.34	Vfetk_PDE_Ju	47
8.3.3.35	Vfetk_PDE_mapBoundary	48
8.3.3.36	Vfetk_PDE_markSimplex	48
8.3.3.37	Vfetk_PDE_oneChart	49
8.3.3.38	Vfetk_PDE_simplexBasisForm	49
8.3.3.39	Vfetk_PDE_simplexBasisInit	50
8.3.3.40	Vfetk_PDE_u_D	51
8.3.3.41	Vfetk_PDE_u_T	51
8.3.3.42	Vfetk_qfEnergy	52
8.3.3.43	Vfetk_readMesh	52
8.3.3.44	Vfetk_setAtomColors	53
8.3.3.45	Vfetk_setParameters	53
8.3.3.46	Vfetk_write	53
8.4	Vpee class	55
8.4.1	Detailed Description	55
8.4.2	Function Documentation	56
8.4.2.1	Vpee_ctor	56
8.4.2.2	Vpee_ctor2	56
8.4.2.3	Vpee_dtor	57
8.4.2.4	Vpee_dtor2	57
8.4.2.5	Vpee_markRefine	57

8.4.2.6	Vpee_numSS	58
8.5	APOLparm class	60
8.5.1	Detailed Description	61
8.5.2	Enumeration Type Documentation	61
8.5.2.1	eAPOLparm_calcEnergy	61
8.5.2.2	eAPOLparm_calcForce	61
8.5.2.3	eAPOLparm_doCalc	61
8.5.3	Function Documentation	62
8.5.3.1	APOLparm_check	62
8.5.3.2	APOLparm_copy	62
8.5.3.3	APOLparm_ctor	62
8.5.3.4	APOLparm_ctor2	62
8.5.3.5	APOLparm_dtor	63
8.5.3.6	APOLparm_dtor2	63
8.6	BEMparm class	64
8.6.1	Detailed Description	64
8.6.2	Typedef Documentation	65
8.6.2.1	BEMparm	65
8.6.3	Enumeration Type Documentation	65
8.6.3.1	eBEMparm_CalcType	65
8.6.4	Function Documentation	65
8.6.4.1	BEMparm_check	65
8.6.4.2	BEMparm_ctor	65
8.6.4.3	BEMparm_ctor2	66
8.6.4.4	BEMparm_dtor	66
8.6.4.5	BEMparm_dtor2	66
8.6.4.6	BEMparm_parseToken	67
8.7	FEMparm class	68
8.7.1	Detailed Description	69
8.7.2	Typedef Documentation	69
8.7.2.1	FEMparm_EtolType	69
8.7.3	Enumeration Type Documentation	69
8.7.3.1	eFEMparm_CalcType	69
8.7.3.2	eFEMparm_EstType	69
8.7.3.3	eFEMparm_EtolType	70
8.7.4	Function Documentation	70
8.7.4.1	FEMparm_check	70

8.7.4.2	FEMparm_copy	71
8.7.4.3	FEMparm_ctor	71
8.7.4.4	FEMparm_ctor2	71
8.7.4.5	FEMparm_dtor	72
8.7.4.6	FEMparm_dtor2	72
8.8	GEOFLOWparm class	73
8.8.1	Detailed Description	74
8.8.2	Typedef Documentation	74
8.8.2.1	GEOFLOWparm	74
8.8.3	Enumeration Type Documentation	74
8.8.3.1	eGEOFLOWparm_CalcType	74
8.8.4	Function Documentation	74
8.8.4.1	GEOFLOWparm_check	74
8.8.4.2	GEOFLOWparm_ctor	75
8.8.4.3	GEOFLOWparm_ctor2	75
8.8.4.4	GEOFLOWparm_dtor	75
8.8.4.5	GEOFLOWparm_dtor2	76
8.8.4.6	GEOFLOWparm_parseToken	76
8.9	MGparm class	77
8.9.1	Detailed Description	78
8.9.2	Enumeration Type Documentation	78
8.9.2.1	eMGparm_CalcType	78
8.9.2.2	eMGparm_CentMeth	79
8.9.3	Function Documentation	79
8.9.3.1	APOLparm_parseToken	79
8.9.3.2	FEMparm_parseToken	79
8.9.3.3	MGparm_check	80
8.9.3.4	MGparm_copy	80
8.9.3.5	MGparm_ctor	80
8.9.3.6	MGparm_ctor2	81
8.9.3.7	MGparm_dtor	81
8.9.3.8	MGparm_dtor2	81
8.9.3.9	MGparm_getCenterX	82
8.9.3.10	MGparm_getCenterY	82
8.9.3.11	MGparm_getCenterZ	82
8.9.3.12	MGparm_getHx	83
8.9.3.13	MGparm_getHy	83

8.9.3.14	MGparm_getHz	83
8.9.3.15	MGparm_getNx	84
8.9.3.16	MGparm_getNy	84
8.9.3.17	MGparm_getNz	84
8.9.3.18	MGparm_parseToken	85
8.9.3.19	MGparm_setCenterX	85
8.9.3.20	MGparm_setCenterY	85
8.9.3.21	MGparm_setCenterZ	86
8.10	NOsh class	87
8.10.1	Detailed Description	89
8.10.2	Enumeration Type Documentation	89
8.10.2.1	eNOsh_CalcType	89
8.10.2.2	eNOsh_MolFormat	90
8.10.2.3	eNOsh_ParmFormat	90
8.10.2.4	eNOsh_PrintType	90
8.10.3	Function Documentation	90
8.10.3.1	NOsh_apol2calc	90
8.10.3.2	NOsh_calc_copy	91
8.10.3.3	NOsh_calc_ctor	91
8.10.3.4	NOsh_calc_dtor	91
8.10.3.5	NOsh_ctor	92
8.10.3.6	NOsh_ctor2	92
8.10.3.7	NOsh_dtor	92
8.10.3.8	NOsh_dtor2	93
8.10.3.9	NOsh_elec2calc	93
8.10.3.10	NOsh_elecname	93
8.10.3.11	NOsh_getCalc	94
8.10.3.12	NOsh_getChargefmt	94
8.10.3.13	NOsh_getChargepath	95
8.10.3.14	NOsh_getDielfmt	95
8.10.3.15	NOsh_getDielXpath	95
8.10.3.16	NOsh_getDielYpath	96
8.10.3.17	NOsh_getDielZpath	96
8.10.3.18	NOsh_getKappafmt	96
8.10.3.19	NOsh_getKappapath	97
8.10.3.20	NOsh_getMolpath	97
8.10.3.21	NOsh_getPotfmt	97

8.10.3.22 NOsh_getPotpath	98
8.10.3.23 NOsh_parseInput	98
8.10.3.24 NOsh_parseInputFile	99
8.10.3.25 NOsh_printCalc	99
8.10.3.26 NOsh_printNarg	100
8.10.3.27 NOsh_printOp	100
8.10.3.28 NOsh_printWhat	101
8.10.3.29 NOsh_setupApolCalc	101
8.10.3.30 NOsh_setupElecCalc	102
8.11 PBEparm class	103
8.11.1 Detailed Description	104
8.11.2 Enumeration Type Documentation	104
8.11.2.1 ePBEparm_calcEnergy	104
8.11.2.2 ePBEparm_calcForce	104
8.11.3 Function Documentation	105
8.11.3.1 PBEparm_check	105
8.11.3.2 PBEparm_copy	105
8.11.3.3 PBEparm_ctor	105
8.11.3.4 PBEparm_ctor2	105
8.11.3.5 PBEparm_dtor	106
8.11.3.6 PBEparm_dtor2	106
8.11.3.7 PBEparm_getIonCharge	106
8.11.3.8 PBEparm_getIonConc	107
8.11.3.9 PBEparm_getIonRadius	107
8.11.3.10 PBEparm_parseToken	107
8.12 Vacc class	109
8.12.1 Detailed Description	111
8.12.2 Function Documentation	111
8.12.2.1 Vacc_atomdSASA	111
8.12.2.2 Vacc_atomdSAV	111
8.12.2.3 Vacc_atomSASA	112
8.12.2.4 Vacc_atomSASPoints	112
8.12.2.5 Vacc_atomSurf	113
8.12.2.6 Vacc_ctor	113
8.12.2.7 Vacc_ctor2	114
8.12.2.8 Vacc_dtor	114
8.12.2.9 Vacc_dtor2	114

8.12.2.10	Vacc_fastMolAcc	114
8.12.2.11	Vacc_ivdwAcc	115
8.12.2.12	Vacc_memChk	115
8.12.2.13	Vacc_molAcc	116
8.12.2.14	Vacc_SASA	116
8.12.2.15	Vacc_splineAcc	117
8.12.2.16	Vacc_splineAccAtom	117
8.12.2.17	Vacc_splineAccGrad	117
8.12.2.18	Vacc_splineAccGradAtomNorm	118
8.12.2.19	Vacc_splineAccGradAtomNorm3	118
8.12.2.20	Vacc_splineAccGradAtomNorm4	119
8.12.2.21	Vacc_splineAccGradAtomUnnorm	119
8.12.2.22	Vacc_totalAtomdSASA	119
8.12.2.23	Vacc_totalAtomdSAV	120
8.12.2.24	Vacc_totalSASA	120
8.12.2.25	Vacc_totalSAV	121
8.12.2.26	Vacc_vdwAcc	121
8.12.2.27	Vacc_wcaEnergy	121
8.12.2.28	Vacc_wcaEnergyAtom	122
8.12.2.29	Vacc_wcaForceAtom	122
8.12.2.30	VaccSurf_ctor	123
8.12.2.31	VaccSurf_ctor2	123
8.12.2.32	VaccSurf_dtor	123
8.12.2.33	VaccSurf_dtor2	124
8.12.2.34	VaccSurf_refSphere	124
8.13	Valist class	125
8.13.1	Detailed Description	126
8.13.2	Function Documentation	126
8.13.2.1	Valist_ctor	126
8.13.2.2	Valist_ctor2	126
8.13.2.3	Valist_dtor	126
8.13.2.4	Valist_dtor2	127
8.13.2.5	Valist_getAtom	127
8.13.2.6	Valist_getAtomList	127
8.13.2.7	Valist_getCenterX	128
8.13.2.8	Valist_getCenterY	128
8.13.2.9	Valist_getCenterZ	128

8.13.2.10	Valist_getNumberAtoms	129
8.13.2.11	Valist_getStatistics	129
8.13.2.12	Valist_memChk	129
8.13.2.13	Valist_readPDB	130
8.13.2.14	Valist_readPQR	130
8.13.2.15	Valist_readXML	131
8.14	Vatom class	132
8.14.1	Detailed Description	133
8.14.2	Macro Definition Documentation	133
8.14.2.1	VMAX_RECLEN	133
8.14.3	Function Documentation	134
8.14.3.1	Vatom_copyFrom	134
8.14.3.2	Vatom_copyTo	134
8.14.3.3	Vatom_ctor	134
8.14.3.4	Vatom_ctor2	134
8.14.3.5	Vatom_dtor	135
8.14.3.6	Vatom_dtor2	135
8.14.3.7	Vatom_getAtomID	135
8.14.3.8	Vatom_getAtomName	136
8.14.3.9	Vatom_getCharge	136
8.14.3.10	Vatom_getEpsilon	136
8.14.3.11	Vatom_getPartID	137
8.14.3.12	Vatom_getPosition	137
8.14.3.13	Vatom_getRadius	137
8.14.3.14	Vatom_getResName	138
8.14.3.15	Vatom_memChk	138
8.14.3.16	Vatom_setAtomID	138
8.14.3.17	Vatom_setAtomName	138
8.14.3.18	Vatom_setCharge	139
8.14.3.19	Vatom_setEpsilon	139
8.14.3.20	Vatom_setPartID	139
8.14.3.21	Vatom_setPosition	139
8.14.3.22	Vatom_setRadius	140
8.14.3.23	Vatom_setResName	140
8.15	Vcap class	141
8.15.1	Detailed Description	141
8.15.2	Function Documentation	141

8.15.2.1	Vcap_cosh	141
8.15.2.2	Vcap_exp	142
8.15.2.3	Vcap_sinh	142
8.16	Vclist class	144
8.16.1	Detailed Description	145
8.16.2	Enumeration Type Documentation	145
8.16.2.1	eVclist_DomainMode	145
8.16.3	Function Documentation	145
8.16.3.1	Vclist_ctor	145
8.16.3.2	Vclist_ctor2	146
8.16.3.3	Vclist_dtor	146
8.16.3.4	Vclist_dtor2	146
8.16.3.5	Vclist_getCell	147
8.16.3.6	Vclist_maxRadius	147
8.16.3.7	Vclist_memChk	147
8.16.3.8	VclistCell_ctor	148
8.16.3.9	VclistCell_ctor2	148
8.16.3.10	VclistCell_dtor	148
8.16.3.11	VclistCell_dtor2	149
8.17	Vgreen class	150
8.17.1	Detailed Description	151
8.17.2	Function Documentation	152
8.17.2.1	Vgreen_coulomb	152
8.17.2.2	Vgreen_coulomb_direct	152
8.17.2.3	Vgreen_coulombD	153
8.17.2.4	Vgreen_coulombD_direct	153
8.17.2.5	Vgreen_ctor	154
8.17.2.6	Vgreen_ctor2	154
8.17.2.7	Vgreen_dtor	155
8.17.2.8	Vgreen_dtor2	155
8.17.2.9	Vgreen_getValist	155
8.17.2.10	Vgreen_helmholtz	156
8.17.2.11	Vgreen_helmholtzD	156
8.17.2.12	Vgreen_memChk	157
8.18	Vhal class	158
8.18.1	Detailed Description	160
8.18.2	Macro Definition Documentation	161

8.18.2.1	MAX_SPHERE_PTS	161
8.18.2.2	VAPBS_BACK	161
8.18.2.3	VAPBS_DOWN	161
8.18.2.4	VAPBS_FRONT	161
8.18.2.5	VAPBS_LEFT	161
8.18.2.6	VAPBS_RIGHT	162
8.18.2.7	VAPBS_UP	162
8.18.2.8	VEMBED	162
8.18.2.9	VFLOOR	162
8.18.3	Enumeration Type Documentation	162
8.18.3.1	eVbcfl	162
8.18.3.2	eVchrg_Meth	163
8.18.3.3	eVchrg_Src	163
8.18.3.4	eVdata_Format	164
8.18.3.5	eVdata_Type	164
8.18.3.6	eVhal_IPKEYType	165
8.18.3.7	eVhal_PBEType	165
8.18.3.8	eVoutput_Format	165
8.18.3.9	eVrc_Codes	165
8.18.3.10	eVsol_Meth	166
8.18.3.11	eVsurf_Meth	166
8.19	Matrix wrapper class	167
8.19.1	Detailed Description	167
8.20	Vparam class	168
8.20.1	Detailed Description	169
8.20.2	Function Documentation	170
8.20.2.1	readFlatFileLine	170
8.20.2.2	readXMLFileAtom	170
8.20.2.3	Vparam_AtomData_copyFrom	170
8.20.2.4	Vparam_AtomData_copyTo	171
8.20.2.5	Vparam_AtomData_ctor	171
8.20.2.6	Vparam_AtomData_ctor2	171
8.20.2.7	Vparam_AtomData_dtor	172
8.20.2.8	Vparam_AtomData_dtor2	172
8.20.2.9	Vparam_ctor	172
8.20.2.10	Vparam_ctor2	172
8.20.2.11	Vparam_dtor	173

8.20.2.12	Vparam_dtor2	173
8.20.2.13	Vparam_getAtomData	173
8.20.2.14	Vparam_getResData	174
8.20.2.15	Vparam_memChk	174
8.20.2.16	Vparam_readFlatFile	175
8.20.2.17	Vparam_readXMLFile	175
8.20.2.18	Vparam_ResData_copyTo	176
8.20.2.19	Vparam_ResData_ctor	176
8.20.2.20	Vparam_ResData_ctor2	176
8.20.2.21	Vparam_ResData_dtor	177
8.20.2.22	Vparam_ResData_dtor2	177
8.21	Vpbe class	178
8.21.1	Detailed Description	179
8.21.2	Function Documentation	180
8.21.2.1	Vpbe_ctor	180
8.21.2.2	Vpbe_ctor2	181
8.21.2.3	Vpbe_dtor	182
8.21.2.4	Vpbe_dtor2	182
8.21.2.5	Vpbe_getBulkIonicStrength	182
8.21.2.6	Vpbe_getCoulombEnergy1	183
8.21.2.7	Vpbe_getDeblen	183
8.21.2.8	Vpbe_getGamma	183
8.21.2.9	Vpbe_getIons	184
8.21.2.10	Vpbe_getLmem	184
8.21.2.11	Vpbe_getMaxIonRadius	184
8.21.2.12	Vpbe_getmembraneDiel	185
8.21.2.13	Vpbe_getmemv	185
8.21.2.14	Vpbe_getSoluteCenter	185
8.21.2.15	Vpbe_getSoluteCharge	186
8.21.2.16	Vpbe_getSoluteDiel	186
8.21.2.17	Vpbe_getSoluteRadius	186
8.21.2.18	Vpbe_getSoluteXlen	187
8.21.2.19	Vpbe_getSoluteYlen	187
8.21.2.20	Vpbe_getSoluteZlen	187
8.21.2.21	Vpbe_getSolventDiel	188
8.21.2.22	Vpbe_getSolventRadius	188
8.21.2.23	Vpbe_getTemperature	188

8.21.2.24	Vpbe_getVacc	189
8.21.2.25	Vpbe_getValist	189
8.21.2.26	Vpbe_getXkappa	189
8.21.2.27	Vpbe_getZkappa2	190
8.21.2.28	Vpbe_getZmagic	190
8.21.2.29	Vpbe_getzmem	190
8.21.2.30	Vpbe_memChk	191
8.22	Vstring class	192
8.22.1	Detailed Description	192
8.22.2	Function Documentation	192
8.22.2.1	Vstring_isdigit	192
8.22.2.2	Vstring_strcasecmp	192
8.22.2.3	Vstring_wrappedtext	193
8.23	Vunit class	195
8.23.1	Detailed Description	195
8.24	Vgrid class	196
8.24.1	Detailed Description	197
8.24.2	Function Documentation	198
8.24.2.1	Vgrid_ctor	198
8.24.2.2	Vgrid_ctor2	198
8.24.2.3	Vgrid_curvature	199
8.24.2.4	Vgrid_dtor	199
8.24.2.5	Vgrid_dtor2	200
8.24.2.6	Vgrid_gradient	200
8.24.2.7	Vgrid_integrate	200
8.24.2.8	Vgrid_memChk	201
8.24.2.9	Vgrid_normH1	201
8.24.2.10	Vgrid_normL1	201
8.24.2.11	Vgrid_normL2	202
8.24.2.12	Vgrid_normLinf	202
8.24.2.13	Vgrid_readDX	203
8.24.2.14	Vgrid_readGZ	203
8.24.2.15	Vgrid_seminormH1	204
8.24.2.16	Vgrid_value	204
8.24.2.17	Vgrid_writeDX	204
8.24.2.18	Vgrid_writeUHBD	205
8.25	Vmgrid class	206

8.25.1	Detailed Description	207
8.25.2	Function Documentation	207
8.25.2.1	Vmgrid_addGrid	207
8.25.2.2	Vmgrid_ctor	207
8.25.2.3	Vmgrid_ctor2	207
8.25.2.4	Vmgrid_curvature	208
8.25.2.5	Vmgrid_dtor	208
8.25.2.6	Vmgrid_dtor2	209
8.25.2.7	Vmgrid_getGridByNum	209
8.25.2.8	Vmgrid_getGridByPoint	209
8.25.2.9	Vmgrid_gradient	209
8.25.2.10	Vmgrid_value	210
8.26	Vopot class	211
8.26.1	Detailed Description	211
8.26.2	Function Documentation	212
8.26.2.1	Vopot_ctor	212
8.26.2.2	Vopot_ctor2	212
8.26.2.3	Vopot_curvature	212
8.26.2.4	Vopot_dtor	213
8.26.2.5	Vopot_dtor2	213
8.26.2.6	Vopot_gradient	213
8.26.2.7	Vopot_pot	214
8.27	Vpmg class	215
8.27.1	Detailed Description	217
8.27.2	Function Documentation	218
8.27.2.1	bcolcomp	218
8.27.2.2	bcolcomp2	218
8.27.2.3	bcolcomp3	219
8.27.2.4	bcolcomp4	219
8.27.2.5	pcolcomp	220
8.27.2.6	Vpackmg	220
8.27.2.7	Vpmg_ctor	221
8.27.2.8	Vpmg_ctor2	221
8.27.2.9	Vpmg_dbDirectPolForce	222
8.27.2.10	Vpmg_dbForce	222
8.27.2.11	Vpmg_dbMutualPolForce	223
8.27.2.12	Vpmg_dbNLDirectPolForce	223

8.27.2.13 Vpmg_dbPermanentMultipoleForce	223
8.27.2.14 Vpmg_dielEnergy	224
8.27.2.15 Vpmg_dielGradNorm	224
8.27.2.16 Vpmg_dtor	225
8.27.2.17 Vpmg_dtor2	225
8.27.2.18 Vpmg_energy	225
8.27.2.19 Vpmg_fieldSpline4	226
8.27.2.20 Vpmg_fillArray	226
8.27.2.21 Vpmg_fillco	226
8.27.2.22 Vpmg_force	227
8.27.2.23 Vpmg_ibDirectPolForce	228
8.27.2.24 Vpmg_ibForce	228
8.27.2.25 Vpmg_ibMutualPolForce	229
8.27.2.26 Vpmg_ibNLDirectPolForce	229
8.27.2.27 Vpmg_ibPermanentMultipoleForce	229
8.27.2.28 Vpmg_memChk	230
8.27.2.29 Vpmg_printColComp	230
8.27.2.30 Vpmg_qfAtomEnergy	231
8.27.2.31 Vpmg_qfDirectPolForce	231
8.27.2.32 Vpmg_qfEnergy	232
8.27.2.33 Vpmg_qfForce	232
8.27.2.34 Vpmg_qfMutualPolForce	233
8.27.2.35 Vpmg_qfNLDirectPolForce	233
8.27.2.36 Vpmg_qfPermanentMultipoleEnergy	233
8.27.2.37 Vpmg_qfPermanentMultipoleForce	234
8.27.2.38 Vpmg_qmEnergy	234
8.27.2.39 Vpmg_setPart	235
8.27.2.40 Vpmg_solve	235
8.27.2.41 Vpmg_solveLaplace	235
8.27.2.42 Vpmg_unsetPart	236
8.28 Vpmgp class	237
8.28.1 Detailed Description	237
8.28.2 Function Documentation	238
8.28.2.1 Vpmgp_ctor	238
8.28.2.2 Vpmgp_ctor2	238
8.28.2.3 Vpmgp_dtor	238
8.28.2.4 Vpmgp_dtor2	239

8.28.2.5	Vpmgp_makeCoarse	239
8.28.2.6	Vpmgp_size	239
8.29	C translation of Holst group PMG code	240
8.29.1	Detailed Description	244
8.29.2	Macro Definition Documentation	244
8.29.2.1	HARMO2	244
8.29.2.2	MAXIONS	245
8.29.3	Function Documentation	246
8.29.3.1	Vaxrand	246
8.29.3.2	Vazeros	246
8.29.3.3	VbuildA	247
8.29.3.4	Vbuildband	249
8.29.3.5	Vbuildband1_27	251
8.29.3.6	Vbuildband1_7	252
8.29.3.7	VbuildG	253
8.29.3.8	VbuildG_1	255
8.29.3.9	VbuildG_27	257
8.29.3.10	VbuildG_7	259
8.29.3.11	Vbuildgaler0	261
8.29.3.12	Vbuildops	262
8.29.3.13	VbuildP	264
8.29.3.14	Vbuildstr	265
8.29.3.15	Vc_vec	266
8.29.3.16	Vc_vecpmg	266
8.29.3.17	Vc_vecsmpbe	267
8.29.3.18	Vcghs	267
8.29.3.19	Vdc_vec	269
8.29.3.20	Vdpbsl	269
8.29.3.21	Vextrac	271
8.29.3.22	VfboundPMG	272
8.29.3.23	VfboundPMG00	272
8.29.3.24	Vfmvfas	273
8.29.3.25	Vfnewton	275
8.29.3.26	Vgetjac	278
8.29.3.27	Vgsrb	279
8.29.3.28	VinterpPMG	281
8.29.3.29	Vipower	282

8.29.3.30 Vmatvec	283
8.29.3.31 Vmgdriv	285
8.29.3.32 Vmgdriv2	287
8.29.3.33 Vmgsz	288
8.29.3.34 Vmresid	290
8.29.3.35 Vmvcs	291
8.29.3.36 Vmvfas	294
8.29.3.37 Vmypdefinitlpbe	295
8.29.3.38 Vmypdefinitnpbe	296
8.29.3.39 Vmypdefinitmpbe	297
8.29.3.40 Vnewdriv	297
8.29.3.41 Vnewdriv2	299
8.29.3.42 Vnewton	301
8.29.3.43 Vnmatvec	302
8.29.3.44 Vnmresid	303
8.29.3.45 Vnsmooth	303
8.29.3.46 Vpower	304
8.29.3.47 Vprtmatd	306
8.29.3.48 Vrestrc	307
8.29.3.49 Vsmooth	307
8.29.3.50 Vxaxpy	310
8.29.3.51 Vxcopy	310
8.29.3.52 Vxcopy_large	312
8.29.3.53 Vxcopy_small	313
8.29.3.54 Vxdot	313
8.29.3.55 Vxnrm1	314
8.29.3.56 Vxnrm2	314
8.29.3.57 Vxscal	314
8.30 High-level front-end routines	316
8.30.1 Detailed Description	319
8.30.2 Function Documentation	319
8.30.2.1 energyAPOL	319
8.30.2.2 energyFE	320
8.30.2.3 energyGEOFLOW	321
8.30.2.4 energyMG	321
8.30.2.5 forceAPOL	322
8.30.2.6 forceGEOFLOW	322

8.30.2.7	forceMG	323
8.30.2.8	initAPOL	323
8.30.2.9	initFE	325
8.30.2.10	initGEOFLOW	326
8.30.2.11	initMG	326
8.30.2.12	killChargeMaps	327
8.30.2.13	killDielMaps	327
8.30.2.14	killEnergy	328
8.30.2.15	killFE	328
8.30.2.16	killForce	328
8.30.2.17	killGEOFLOW	328
8.30.2.18	killKappaMaps	329
8.30.2.19	killMeshes	329
8.30.2.20	killMG	329
8.30.2.21	killMolecules	329
8.30.2.22	killPotMaps	330
8.30.2.23	loadChargeMaps	330
8.30.2.24	loadDielMaps	330
8.30.2.25	loadKappaMaps	331
8.30.2.26	loadMeshes	331
8.30.2.27	loadMolecules	332
8.30.2.28	loadParameter	332
8.30.2.29	loadPotMaps	332
8.30.2.30	main	333
8.30.2.31	partFE	333
8.30.2.32	postRefineFE	334
8.30.2.33	preRefineFE	334
8.30.2.34	printApolEnergy	335
8.30.2.35	printApolForce	335
8.30.2.36	printElecEnergy	336
8.30.2.37	printElecForce	336
8.30.2.38	printEnergy	336
8.30.2.39	printFEPARM	337
8.30.2.40	printForce	337
8.30.2.41	printGEOFLOWPARM	337
8.30.2.42	printMGPARM	338
8.30.2.43	printPBEPARM	338

8.30.2.44	returnEnergy	338
8.30.2.45	setPartGEOFLOW	338
8.30.2.46	setPartMG	339
8.30.2.47	solveFE	339
8.30.2.48	solveGEOFLOW	340
8.30.2.49	solveMG	340
8.30.2.50	startVio	340
8.30.2.51	storeAtomEnergy	341
8.30.2.52	writedataFE	341
8.30.2.53	writedataFlat	342
8.30.2.54	writedataGEOFLOW	342
8.30.2.55	writedataMG	343
8.30.2.56	writedataXML	343
8.30.2.57	writematGEOFLOW	344
8.30.2.58	writematMG	344
9	Data Structure Documentation	345
9.1	_GeoflowInput Struct Reference	345
9.1.1	Detailed Description	346
9.2	_GeoflowOutput Struct Reference	346
9.2.1	Detailed Description	346
9.3	AtomForce Struct Reference	346
9.3.1	Detailed Description	346
9.3.2	Field Documentation	347
9.3.2.1	dbForce	347
9.3.2.2	ibForce	347
9.3.2.3	qfForce	347
9.3.2.4	sasaForce	347
9.3.2.5	savForce	347
9.3.2.6	wcaForce	347
9.4	Comdata Struct Reference	347
9.4.1	Detailed Description	348
9.5	LJ Struct Reference	348
9.5.1	Detailed Description	349
9.6	Mat< T > Class Template Reference	349
9.6.1	Detailed Description	350
9.7	sAPOLparm Struct Reference	350

9.7.1	Detailed Description	351
9.7.2	Field Documentation	351
9.7.2.1	bconc	351
9.7.2.2	calcenergy	351
9.7.2.3	calcforce	351
9.7.2.4	dpos	351
9.7.2.5	gamma	351
9.7.2.6	grid	351
9.7.2.7	molid	351
9.7.2.8	parsed	352
9.7.2.9	press	352
9.7.2.10	sasa	352
9.7.2.11	sav	352
9.7.2.12	sdens	352
9.7.2.13	setbconc	352
9.7.2.14	setcalcenergy	352
9.7.2.15	setcalcforce	353
9.7.2.16	setdpos	353
9.7.2.17	setgamma	353
9.7.2.18	setgrid	353
9.7.2.19	setmolid	353
9.7.2.20	setpress	354
9.7.2.21	setsdens	354
9.7.2.22	setsrad	354
9.7.2.23	setsrfm	354
9.7.2.24	setswin	354
9.7.2.25	settemp	355
9.7.2.26	setwat	355
9.7.2.27	srad	355
9.7.2.28	srfm	355
9.7.2.29	swin	355
9.7.2.30	temp	355
9.7.2.31	totForce	355
9.7.2.32	watepsilon	355
9.7.2.33	watsigma	356
9.7.2.34	wcaEnergy	356
9.8	sBEMparm Struct Reference	356

9.8.1	Detailed Description	356
9.8.2	Field Documentation	357
9.8.2.1	chgs	357
9.8.2.2	mac	357
9.8.2.3	nonlotype	357
9.8.2.4	parsed	357
9.8.2.5	setmac	357
9.8.2.6	setnonlotype	357
9.8.2.7	settree_n0	357
9.8.2.8	settree_order	358
9.8.2.9	tree_n0	358
9.8.2.10	tree_order	358
9.8.2.11	type	358
9.9	sFEMparm Struct Reference	358
9.9.1	Detailed Description	359
9.9.2	Field Documentation	359
9.9.2.1	akeyPRE	359
9.9.2.2	akeySOLVE	359
9.9.2.3	ekey	359
9.9.2.4	etol	360
9.9.2.5	glen	360
9.9.2.6	maxsolve	360
9.9.2.7	maxvert	360
9.9.2.8	meshID	360
9.9.2.9	parsed	360
9.9.2.10	pkey	360
9.9.2.11	setakeyPRE	360
9.9.2.12	setakeySOLVE	360
9.9.2.13	setekey	361
9.9.2.14	setetol	361
9.9.2.15	setglen	361
9.9.2.16	setmaxsolve	361
9.9.2.17	setmaxvert	361
9.9.2.18	settargetNum	361
9.9.2.19	settargetRes	361
9.9.2.20	settype	361
9.9.2.21	targetNum	362

9.9.2.22	targetRes	362
9.9.2.23	type	362
9.9.2.24	useMesh	362
9.10	sGEOFLOWparm Struct Reference	362
9.10.1	Detailed Description	362
9.10.2	Field Documentation	363
9.10.2.1	parsed	363
9.10.2.2	type	363
9.11	sMGparm Struct Reference	363
9.11.1	Detailed Description	364
9.11.2	Field Documentation	365
9.11.2.1	async	365
9.11.2.2	ccenter	365
9.11.2.3	ccentmol	365
9.11.2.4	ccmeth	365
9.11.2.5	center	365
9.11.2.6	centmol	365
9.11.2.7	cglen	365
9.11.2.8	chgm	365
9.11.2.9	chgs	366
9.11.2.10	cmeth	366
9.11.2.11	dime	366
9.11.2.12	etol	366
9.11.2.13	fcenter	366
9.11.2.14	fcentmol	366
9.11.2.15	fcmeth	366
9.11.2.16	fglen	366
9.11.2.17	glen	366
9.11.2.18	grid	367
9.11.2.19	method	367
9.11.2.20	nlev	367
9.11.2.21	nonlintype	367
9.11.2.22	ofrac	367
9.11.2.23	parsed	367
9.11.2.24	partDisjCenter	367
9.11.2.25	partDisjLength	367
9.11.2.26	partDisjOwnSide	368

9.11.2.27 pdime	368
9.11.2.28 proc_rank	368
9.11.2.29 proc_size	368
9.11.2.30 setasync	368
9.11.2.31 setcgcent	368
9.11.2.32 setcglen	368
9.11.2.33 setchgm	369
9.11.2.34 setdime	369
9.11.2.35 setetol	369
9.11.2.36 setfgcent	369
9.11.2.37 setfglen	369
9.11.2.38 setgcent	370
9.11.2.39 setglen	370
9.11.2.40 setgrid	370
9.11.2.41 setmethod	370
9.11.2.42 setnlev	370
9.11.2.43 setnonlotype	371
9.11.2.44 setofrac	371
9.11.2.45 setpdime	371
9.11.2.46 setrank	371
9.11.2.47 setsize	371
9.11.2.48 setUseAqua	372
9.11.2.49 type	372
9.11.2.50 useAqua	372
9.12 sNOsh Struct Reference	372
9.12.1 Detailed Description	373
9.12.2 Field Documentation	373
9.12.2.1 alist	373
9.12.2.2 apol	373
9.12.2.3 apol2calc	374
9.12.2.4 apolname	374
9.12.2.5 bogus	374
9.12.2.6 calc	374
9.12.2.7 chargefmt	374
9.12.2.8 chargepath	374
9.12.2.9 dielfmt	374
9.12.2.10 dielXpath	374

9.12.2.11 dielYpath	374
9.12.2.12 dielZpath	375
9.12.2.13 elec	375
9.12.2.14 elec2calc	375
9.12.2.15 elecname	375
9.12.2.16 gotparm	375
9.12.2.17 ispara	375
9.12.2.18 kappafmt	375
9.12.2.19 kappapath	375
9.12.2.20 meshfmt	376
9.12.2.21 meshpath	376
9.12.2.22 molfmt	376
9.12.2.23 molpath	376
9.12.2.24 napol	376
9.12.2.25 ncalc	376
9.12.2.26 ncharge	376
9.12.2.27 ndiel	376
9.12.2.28 nelec	376
9.12.2.29 nkappa	377
9.12.2.30 nmesh	377
9.12.2.31 nmol	377
9.12.2.32 npot	377
9.12.2.33 nprint	377
9.12.2.34 parmfmt	377
9.12.2.35 parmpath	377
9.12.2.36 parsed	377
9.12.2.37 potfmt	377
9.12.2.38 potpath	378
9.12.2.39 printcalc	378
9.12.2.40 printnarg	378
9.12.2.41 printop	378
9.12.2.42 printwhat	378
9.12.2.43 proc_rank	378
9.12.2.44 proc_size	378
9.13 sNOsh_calc Struct Reference	379
9.13.1 Detailed Description	379
9.13.2 Field Documentation	379

9.13.2.1	apolparm	379
9.13.2.2	bemparm	379
9.13.2.3	calctype	379
9.13.2.4	femparm	379
9.13.2.5	mgparm	380
9.13.2.6	pbeparm	380
9.14	sPBEparm Struct Reference	380
9.14.1	Detailed Description	381
9.14.2	Field Documentation	382
9.14.2.1	bcfl	382
9.14.2.2	calcenergy	382
9.14.2.3	calcforce	382
9.14.2.4	chargeMapID	382
9.14.2.5	dielMapID	382
9.14.2.6	ionc	382
9.14.2.7	ionq	382
9.14.2.8	ionr	382
9.14.2.9	kappaMapID	383
9.14.2.10	Lmem	383
9.14.2.11	mdie	383
9.14.2.12	memv	383
9.14.2.13	molid	383
9.14.2.14	nion	383
9.14.2.15	numwrite	383
9.14.2.16	parsed	383
9.14.2.17	pbetype	383
9.14.2.18	pdie	384
9.14.2.19	potMapID	384
9.14.2.20	sdens	384
9.14.2.21	sdie	384
9.14.2.22	setbcfl	384
9.14.2.23	setcalcenergy	384
9.14.2.24	setcalcforce	384
9.14.2.25	setion	385
9.14.2.26	setLmem	385
9.14.2.27	setmdie	385
9.14.2.28	setmemv	385

9.14.2.29 setmolid	385
9.14.2.30 setnion	385
9.14.2.31 setpbetype	385
9.14.2.32 setpdie	386
9.14.2.33 setsdens	386
9.14.2.34 setsdie	386
9.14.2.35 setsmsize	386
9.14.2.36 setsmvolume	386
9.14.2.37 setsrad	387
9.14.2.38 setsrfm	387
9.14.2.39 setswin	387
9.14.2.40 settemp	387
9.14.2.41 setwritemat	387
9.14.2.42 setzmem	388
9.14.2.43 smsize	388
9.14.2.44 smvolume	388
9.14.2.45 srad	388
9.14.2.46 srfm	388
9.14.2.47 swin	388
9.14.2.48 temp	388
9.14.2.49 useChargeMap	388
9.14.2.50 useDielMap	388
9.14.2.51 useKappaMap	389
9.14.2.52 usePotMap	389
9.14.2.53 writefmt	389
9.14.2.54 writemat	389
9.14.2.55 writematflag	389
9.14.2.56 writematstem	389
9.14.2.57 writestem	389
9.14.2.58 writetype	390
9.14.2.59 zmem	390
9.15 Stencil< T > Struct Template Reference	390
9.15.1 Detailed Description	391
9.16 sVacc Struct Reference	391
9.16.1 Detailed Description	392
9.16.2 Field Documentation	392
9.16.2.1 acc	392

9.16.2.2	alist	392
9.16.2.3	atomFlags	392
9.16.2.4	clist	392
9.16.2.5	mem	392
9.16.2.6	refSphere	393
9.16.2.7	surf	393
9.16.2.8	surf_density	393
9.17	sVaccSurf Struct Reference	393
9.17.1	Detailed Description	393
9.17.2	Field Documentation	394
9.17.2.1	area	394
9.17.2.2	bpts	394
9.17.2.3	mem	394
9.17.2.4	npts	394
9.17.2.5	probe_radius	394
9.17.2.6	xpts	394
9.17.2.7	ypts	394
9.17.2.8	zpts	394
9.18	sValist Struct Reference	395
9.18.1	Detailed Description	395
9.18.2	Field Documentation	395
9.18.2.1	atoms	395
9.18.2.2	center	395
9.18.2.3	charge	395
9.18.2.4	maxcrd	395
9.18.2.5	maxrad	396
9.18.2.6	mincrd	396
9.18.2.7	number	396
9.18.2.8	vmem	396
9.19	sVatom Struct Reference	396
9.19.1	Detailed Description	396
9.19.2	Field Documentation	397
9.19.2.1	atomName	397
9.19.2.2	charge	397
9.19.2.3	epsilon	397
9.19.2.4	id	397
9.19.2.5	partID	397

9.19.2.6	position	397
9.19.2.7	radius	397
9.19.2.8	resName	398
9.20	sVclist Struct Reference	398
9.20.1	Detailed Description	398
9.20.2	Field Documentation	398
9.20.2.1	alist	398
9.20.2.2	cells	398
9.20.2.3	lower_corner	399
9.20.2.4	max_radius	399
9.20.2.5	mode	399
9.20.2.6	n	399
9.20.2.7	npts	399
9.20.2.8	spacs	399
9.20.2.9	upper_corner	399
9.20.2.10	vmem	399
9.21	sVclistCell Struct Reference	400
9.21.1	Detailed Description	400
9.21.2	Field Documentation	400
9.21.2.1	atoms	400
9.21.2.2	natoms	400
9.22	sVcsm Struct Reference	400
9.22.1	Detailed Description	401
9.22.2	Field Documentation	401
9.22.2.1	alist	401
9.22.2.2	gm	401
9.22.2.3	initFlag	401
9.22.2.4	msimp	401
9.22.2.5	natom	401
9.22.2.6	nqsm	402
9.22.2.7	nsimp	402
9.22.2.8	nsqm	402
9.22.2.9	qsm	402
9.22.2.10	sqm	402
9.22.2.11	vmem	402
9.23	sVfetk Struct Reference	402
9.23.1	Detailed Description	403

9.23.2	Field Documentation	403
9.23.2.1	am	403
9.23.2.2	aprx	403
9.23.2.3	csm	403
9.23.2.4	feparm	404
9.23.2.5	gm	404
9.23.2.6	gues	404
9.23.2.7	level	404
9.23.2.8	lkey	404
9.23.2.9	lmax	404
9.23.2.10	lprec	404
9.23.2.11	ltol	404
9.23.2.12	nkey	404
9.23.2.13	nmax	405
9.23.2.14	ntol	405
9.23.2.15	pbe	405
9.23.2.16	pbeparm	405
9.23.2.17	pde	405
9.23.2.18	pjac	405
9.23.2.19	type	405
9.23.2.20	vmem	405
9.24	sVfetk_LocalVar Struct Reference	406
9.24.1	Detailed Description	406
9.24.2	Field Documentation	407
9.24.2.1	A	407
9.24.2.2	B	407
9.24.2.3	d2W	407
9.24.2.4	DB	407
9.24.2.5	delta	407
9.24.2.6	DFu_wv	407
9.24.2.7	dU	407
9.24.2.8	dW	407
9.24.2.9	F	408
9.24.2.10	fetk	408
9.24.2.11	fType	408
9.24.2.12	Fu_v	408
9.24.2.13	green	408

9.24.2.14	initGreen	408
9.24.2.15	ionConc	408
9.24.2.16	ionQ	408
9.24.2.17	ionRadii	408
9.24.2.18	ionstr	409
9.24.2.19	jumpDiel	409
9.24.2.20	nion	409
9.24.2.21	nvec	409
9.24.2.22	nverts	409
9.24.2.23	simp	409
9.24.2.24	sType	409
9.24.2.25	U	409
9.24.2.26	u_D	409
9.24.2.27	u_T	410
9.24.2.28	verts	410
9.24.2.29	vx	410
9.24.2.30	W	410
9.24.2.31	xq	410
9.24.2.32	zkappa2	410
9.24.2.33	zks2	410
9.25	sVgreen Struct Reference	410
9.25.1	Detailed Description	411
9.25.2	Field Documentation	411
9.25.2.1	alist	411
9.25.2.2	np	411
9.25.2.3	qp	411
9.25.2.4	vmem	411
9.25.2.5	xp	411
9.25.2.6	yp	412
9.25.2.7	zp	412
9.26	sVgrid Struct Reference	412
9.26.1	Detailed Description	412
9.26.2	Field Documentation	413
9.26.2.1	ctordata	413
9.26.2.2	data	413
9.26.2.3	hx	413
9.26.2.4	hy	413

9.26.2.5	hzed	413
9.26.2.6	mem	413
9.26.2.7	nx	413
9.26.2.8	ny	413
9.26.2.9	nz	414
9.26.2.10	readdata	414
9.26.2.11	xmax	414
9.26.2.12	xmin	414
9.26.2.13	ymax	414
9.26.2.14	ymin	414
9.26.2.15	zmax	414
9.26.2.16	zmin	414
9.27	sVmgrid Struct Reference	415
9.27.1	Detailed Description	415
9.27.2	Field Documentation	415
9.27.2.1	grids	415
9.27.2.2	ngrids	415
9.28	sVopot Struct Reference	415
9.28.1	Detailed Description	416
9.28.2	Field Documentation	416
9.28.2.1	bcfl	416
9.28.2.2	mgrid	416
9.28.2.3	pbe	416
9.29	sVparam_AtomData Struct Reference	416
9.29.1	Detailed Description	417
9.29.2	Field Documentation	417
9.29.2.1	atomName	417
9.29.2.2	charge	417
9.29.2.3	epsilon	417
9.29.2.4	radius	417
9.29.2.5	resName	418
9.30	sVpbe Struct Reference	418
9.30.1	Detailed Description	419
9.30.2	Field Documentation	419
9.30.2.1	acc	419
9.30.2.2	alist	419
9.30.2.3	bulkIonicStrength	419

9.30.2.4	clist	419
9.30.2.5	deblen	419
9.30.2.6	ionConc	419
9.30.2.7	ionQ	419
9.30.2.8	ionRadii	420
9.30.2.9	ipkey	420
9.30.2.10	L	420
9.30.2.11	maxIonRadius	420
9.30.2.12	membraneDiel	420
9.30.2.13	numlon	420
9.30.2.14	param2Flag	420
9.30.2.15	paramFlag	420
9.30.2.16	smsize	420
9.30.2.17	smvolume	421
9.30.2.18	soluteCenter	421
9.30.2.19	soluteCharge	421
9.30.2.20	soluteDiel	421
9.30.2.21	soluteRadius	421
9.30.2.22	soluteXlen	421
9.30.2.23	soluteYlen	421
9.30.2.24	soluteZlen	421
9.30.2.25	solventDiel	421
9.30.2.26	solventRadius	422
9.30.2.27	T	422
9.30.2.28	V	422
9.30.2.29	vmem	422
9.30.2.30	xkappa	422
9.30.2.31	z_mem	422
9.30.2.32	zkappa2	422
9.30.2.33	zmagic	422
9.31	sVpee Struct Reference	423
9.31.1	Detailed Description	423
9.31.2	Field Documentation	423
9.31.2.1	gm	423
9.31.2.2	killFlag	423
9.31.2.3	killParam	423
9.31.2.4	localPartCenter	423

9.31.2.5	localPartID	424
9.31.2.6	localPartRadius	424
9.31.2.7	mem	424
9.32	sVpmg Struct Reference	424
9.32.1	Detailed Description	425
9.32.2	Field Documentation	425
9.32.2.1	a1cf	425
9.32.2.2	a2cf	426
9.32.2.3	a3cf	426
9.32.2.4	ccf	426
9.32.2.5	charge	426
9.32.2.6	chargeMap	426
9.32.2.7	chargeMeth	426
9.32.2.8	chargeSrc	426
9.32.2.9	dielXMap	426
9.32.2.10	dielYMap	426
9.32.2.11	dielZMap	427
9.32.2.12	epsx	427
9.32.2.13	epsy	427
9.32.2.14	epsz	427
9.32.2.15	extDiEnergy	427
9.32.2.16	extNpEnergy	427
9.32.2.17	extQfEnergy	427
9.32.2.18	extQmEnergy	427
9.32.2.19	fcf	427
9.32.2.20	filled	428
9.32.2.21	gxcf	428
9.32.2.22	gycf	428
9.32.2.23	gzcf	428
9.32.2.24	iparm	428
9.32.2.25	iwork	428
9.32.2.26	kappa	428
9.32.2.27	kappaMap	428
9.32.2.28	pbe	428
9.32.2.29	pmgp	429
9.32.2.30	pot	429
9.32.2.31	potMap	429

9.32.2.32 pvec	429
9.32.2.33 rparm	429
9.32.2.34 rwork	429
9.32.2.35 splineWin	429
9.32.2.36 surfMeth	429
9.32.2.37 tcf	429
9.32.2.38 u	430
9.32.2.39 useChargeMap	430
9.32.2.40 useDielXMap	430
9.32.2.41 useDielYMap	430
9.32.2.42 useDielZMap	430
9.32.2.43 useKappaMap	430
9.32.2.44 usePotMap	430
9.32.2.45 vmem	430
9.32.2.46 xf	430
9.32.2.47 yf	431
9.32.2.48 zf	431
9.33 sVpmgp Struct Reference	431
9.33.1 Detailed Description	432
9.33.2 Field Documentation	432
9.33.2.1 bcfl	432
9.33.2.2 errtol	433
9.33.2.3 hx	433
9.33.2.4 hy	433
9.33.2.5 hzed	433
9.33.2.6 iinfo	433
9.33.2.7 ipcon	433
9.33.2.8 iperf	434
9.33.2.9 ipkey	434
9.33.2.10 irite	434
9.33.2.11 istop	434
9.33.2.12 itmax	434
9.33.2.13 key	435
9.33.2.14 meth	435
9.33.2.15 mgcoar	435
9.33.2.16 mgdisc	435
9.33.2.17 mgkey	436

9.33.2.18 mgprol	436
9.33.2.19 mgsmoo	436
9.33.2.20 mgsolv	436
9.33.2.21 n_ipc	436
9.33.2.22 n_iz	437
9.33.2.23 n_rpc	437
9.33.2.24 narr	437
9.33.2.25 narrc	437
9.33.2.26 nc	437
9.33.2.27 nf	437
9.33.2.28 niwk	437
9.33.2.29 nlev	437
9.33.2.30 nonlin	437
9.33.2.31 nrwk	438
9.33.2.32 nu1	438
9.33.2.33 nu2	438
9.33.2.34 nx	438
9.33.2.35 nxc	438
9.33.2.36 ny	438
9.33.2.37 nyc	438
9.33.2.38 nz	439
9.33.2.39 nzc	439
9.33.2.40 omegal	439
9.33.2.41 omegan	439
9.33.2.42 xcent	439
9.33.2.43 xlen	439
9.33.2.44 xmax	439
9.33.2.45 xmin	439
9.33.2.46 ycent	439
9.33.2.47 ylen	440
9.33.2.48 ymax	440
9.33.2.49 ymin	440
9.33.2.50 zcent	440
9.33.2.51 zlen	440
9.33.2.52 zmax	440
9.33.2.53 zmin	440
9.34 Vparam Struct Reference	440

9.34.1 Detailed Description	441
9.34.2 Field Documentation	441
9.34.2.1 nResData	441
9.34.2.2 resData	441
9.34.2.3 vmem	441
9.35 Vparam_ResData Struct Reference	441
9.35.1 Detailed Description	442
9.35.2 Field Documentation	442
9.35.2.1 atomData	442
9.35.2.2 name	442
9.35.2.3 nAtomData	442
9.35.2.4 vmem	442
10 File Documentation	443
10.1 src/apbs.h File Reference	443
10.1.1 Detailed Description	444
10.2 apbs.h	445
10.3 src/fem/vcsm.c File Reference	446
10.3.1 Detailed Description	447
10.4 vcsm.c	448
10.5 src/fem/vcsm.h File Reference	454
10.5.1 Detailed Description	456
10.6 vcsm.h	457
10.7 src/fem/vfetc.c File Reference	459
10.7.1 Detailed Description	462
10.7.2 Variable Documentation	463
10.7.2.1 lgr_2DP1	463
10.7.2.2 lgr_2DP1x	463
10.7.2.3 lgr_2DP1y	463
10.7.2.4 lgr_2DP1z	463
10.7.2.5 lgr_3DP1	464
10.7.2.6 lgr_3DP1x	464
10.7.2.7 lgr_3DP1y	464
10.7.2.8 lgr_3DP1z	464
10.8 vfetc.c	465
10.9 src/fem/vfetc.h File Reference	495
10.9.1 Detailed Description	500

10.10vfetk.h	501
10.11src/fem/vpee.c File Reference	506
10.11.1 Detailed Description	507
10.12vpee.c	508
10.13src/fem/vpee.h File Reference	514
10.13.1 Detailed Description	516
10.14vpee.h	517
10.15src/generic/apolparm.c File Reference	518
10.15.1 Detailed Description	519
10.16apolparm.c	520
10.17src/generic/bemparm.c File Reference	527
10.17.1 Detailed Description	529
10.18bemparm.c	530
10.19src/generic/femparm.c File Reference	532
10.19.1 Detailed Description	534
10.20femparm.c	535
10.21src/generic/femparm.h File Reference	540
10.21.1 Detailed Description	542
10.22femparm.h	543
10.23src/generic/geoflowparm.c File Reference	544
10.23.1 Detailed Description	546
10.24geoflowparm.c	547
10.25src/generic/geoflowparm.h File Reference	549
10.25.1 Detailed Description	551
10.26geoflowparm.h	552
10.27src/generic/mgparm.c File Reference	553
10.27.1 Detailed Description	554
10.28mgparm.c	555
10.29src/generic/mgparm.h File Reference	567
10.29.1 Detailed Description	569
10.30mgparm.h	570
10.31src/generic/nosh.c File Reference	572
10.31.1 Detailed Description	574
10.32nosh.c	575
10.33src/generic/nosh.h File Reference	610
10.33.1 Detailed Description	614
10.34nosh.h	615

10.35src/generic/pbeparm.c File Reference	617
10.35.1 Detailed Description	619
10.36pbeparm.c	620
10.37src/generic/pbeparm.h File Reference	636
10.37.1 Detailed Description	638
10.38pbeparm.h	639
10.39src/generic/vacc.c File Reference	641
10.39.1 Detailed Description	643
10.39.2 Function Documentation	644
10.39.2.1 ivdwAccExclus	644
10.39.2.2 splineAcc	644
10.39.2.3 Vacc_allocate	645
10.39.2.4 Vacc_storeParms	645
10.40vacc.c	645
10.41src/generic/vacc.h File Reference	668
10.41.1 Detailed Description	671
10.42vacc.h	672
10.43src/generic/valist.c File Reference	676
10.43.1 Detailed Description	678
10.44valist.c	679
10.45src/generic/valist.h File Reference	689
10.45.1 Detailed Description	691
10.46valist.h	692
10.47src/generic/vatom.c File Reference	694
10.47.1 Detailed Description	695
10.48vatom.c	696
10.49src/generic/vatom.h File Reference	699
10.49.1 Detailed Description	702
10.50vatom.h	703
10.51src/generic/vcap.c File Reference	704
10.51.1 Detailed Description	705
10.52vcap.c	706
10.53src/generic/vcap.h File Reference	707
10.53.1 Detailed Description	708
10.54vcap.h	709
10.55src/generic/vclist.c File Reference	710
10.55.1 Detailed Description	711

10.56vclist.c	712
10.57src/generic/vclist.h File Reference	718
10.57.1 Detailed Description	720
10.58vclist.h	721
10.59src/generic/vgreen.c File Reference	723
10.59.1 Detailed Description	724
10.60vgreen.c	725
10.61src/generic/vgreen.h File Reference	731
10.61.1 Detailed Description	733
10.62vgreen.h	733
10.63src/generic/vhal.h File Reference	734
10.63.1 Detailed Description	738
10.63.2 Macro Definition Documentation	739
10.63.2.1 PRINT_FUNC	739
10.63.2.2 VABORT_MSG0	739
10.63.2.3 VABORT_MSG1	739
10.63.2.4 VABORT_MSG2	740
10.63.2.5 VASSERT_MSG0	740
10.63.2.6 VASSERT_MSG1	740
10.63.2.7 VASSERT_MSG2	740
10.63.2.8 VCHANNELEDMESSAGE0	741
10.63.2.9 VCHANNELEDMESSAGE1	741
10.63.2.10VCHANNELEDMESSAGE2	741
10.63.2.11VCHANNELEDMESSAGE3	741
10.63.2.12VCOPY	742
10.63.2.13VFILL	742
10.63.2.14VWARN_MSG0	742
10.63.2.15VWARN_MSG1	742
10.63.2.16VWARN_MSG2	743
10.64vhal.h	743
10.65src/generic/vmatrix.h File Reference	751
10.65.1 Detailed Description	752
10.65.2 Macro Definition Documentation	753
10.65.2.1 MAT2	753
10.65.2.2 MAT3	753
10.65.2.3 VAT3	753
10.66vmatrix.h	753

10.67src/generic/vparam.c File Reference	754
10.67.1 Detailed Description	756
10.68vparam.c	757
10.69src/generic/vparam.h File Reference	765
10.69.1 Detailed Description	767
10.70vparam.h	768
10.71src/generic/vpbe.c File Reference	770
10.71.1 Detailed Description	772
10.72vpbe.c	773
10.73src/generic/vpbe.h File Reference	779
10.73.1 Detailed Description	782
10.74vpbe.h	783
10.75src/generic/vstring.c File Reference	785
10.75.1 Detailed Description	786
10.76vstring.c	787
10.77src/generic/vstring.h File Reference	790
10.77.1 Detailed Description	790
10.78vstring.h	792
10.79src/generic/vunit.h File Reference	792
10.79.1 Detailed Description	793
10.80vunit.h	794
10.81src/geoflow/cpbconcz2.h File Reference	795
10.81.1 Detailed Description	796
10.82cpbconcz2.h	797
10.83src/geoflow/Mat.h File Reference	798
10.83.1 Detailed Description	799
10.84Mat.h	800
10.85src/geoflow/modules.h File Reference	804
10.85.1 Detailed Description	805
10.86modules.h	806
10.87src/main.c File Reference	806
10.87.1 Detailed Description	807
10.88main.c	808
10.89src/mg/vgrid.c File Reference	818
10.89.1 Detailed Description	821
10.89.2 Function Documentation	822
10.89.2.1 Vgrid_writeGZ	822

10.90vgrid.c	822
10.91src/mg/vgrid.h File Reference	840
10.91.1 Detailed Description	844
10.91.2 Function Documentation	845
10.91.2.1 Vgrid_writeGZ	845
10.92vgrid.h	845
10.93src/mg/vmgrid.c File Reference	847
10.93.1 Detailed Description	848
10.94vmgrid.c	849
10.95src/mg/vmgrid.h File Reference	851
10.95.1 Detailed Description	853
10.96vmgrid.h	854
10.97src/mg/vopot.c File Reference	855
10.97.1 Detailed Description	855
10.98vopot.c	856
10.99src/mg/vopot.h File Reference	861
10.99.1 Detailed Description	862
10.100opot.h	863
10.101src/mg/vpmg.c File Reference	864
10.101.1 Detailed Description	867
10.101.2 Function Documentation	869
10.101.2.1bcCalc	869
10.101.2.2bcfl1	869
10.101.2.3bspline2	869
10.101.2.4bspline4	870
10.101.2.5d2bspline4	870
10.101.2.6d3bspline4	870
10.101.2.7dbspline2	871
10.101.2.8dbspline4	871
10.101.2.9fillCoCharge	871
10.101.2.10lCoChargeMap	872
10.101.2.11lCoChargeSpline1	872
10.101.2.12lCoChargeSpline2	872
10.101.2.13lCoCoef	872
10.101.2.14lCoCoefMap	872
10.101.2.15lCoCoefMol	873
10.101.2.16lCoCoefMolDiel	873

10.101.2.17	fillCoefMolDielNoSmooth	. 873
10.101.2.18	fillCoefMolDielSmooth	. 873
10.101.2.19	fillCoefMollon	. 874
10.101.2.20	fillCoefSpline	. 874
10.101.2.21	fillCoefSpline3	. 874
10.101.2.22	fillCoefSpline4	. 874
10.101.2.23	fillCoPermanentMultipole	. 874
10.101.2.24	markSphere	. 875
10.101.2.25	multipolebc	. 875
10.101.2.26	ForceSpline1	. 875
10.101.2.27	ForceSpline2	. 876
10.101.2.28	ForceSpline4	. 876
10.101.2.29	FCHI4	. 876
10.101.2.30	pmg_polarizEnergy	. 876
10.101.2.31	pmg_qfEnergyPoint	. 877
10.101.2.32	pmg_qfEnergyVolume	. 877
10.101.2.33	pmg_qmEnergySMPBE	. 877
10.101.2.34	pmg_splineSelect	. 878
10.101.2.35	apSolve	. 878
10.102	pmg.c	. 878
10.103	src/mg/vpmg.h File Reference	. 1015
10.103.1	Detailed Description	. 1021
10.103.2	Function Documentation	. 1022
10.103.2.1	bcCalc	. 1022
10.103.2.2	bcf1	. 1023
10.103.2.3	bspline2	. 1023
10.103.2.4	bspline4	. 1023
10.103.2.5	d2bspline4	. 1024
10.103.2.6	d3bspline4	. 1024
10.103.2.7	db spline2	. 1024
10.103.2.8	db spline4	. 1025
10.103.2.9	fillCoCharge	. 1025
10.103.2.10	fillCoChargeMap	. 1025
10.103.2.11	fillCoChargeSpline1	. 1026
10.103.2.12	fillCoChargeSpline2	. 1026
10.103.2.13	fillCoCoef	. 1026
10.103.2.14	fillCoCoefMap	. 1026

10.103.2.16	lcoCoefMol	1026
10.103.2.16	lcoCoefMolDiel	1027
10.103.2.17	lcoCoefMolDielNoSmooth	1027
10.103.2.18	lcoCoefMolDielSmooth	1027
10.103.2.19	lcoCoefMollon	1027
10.103.2.20	lcoCoefSpline	1028
10.103.2.21	lcoCoefSpline3	1028
10.103.2.22	lcoCoefSpline4	1028
10.103.2.23	lcoInducedDipole	1028
10.103.2.24	lcoNLInducedDipole	1028
10.103.2.25	lcoPermanentMultipole	1028
10.103.2.26	arkSphere	1029
10.103.2.27	ultipolebc	1029
10.103.2.28	ForceSpline1	1029
10.103.2.29	ForceSpline2	1030
10.103.2.30	ForceSpline4	1030
10.103.2.31	VFCHI4	1030
10.103.2.32	pmg_polarizEnergy	1030
10.103.2.33	pmg_qfEnergyPoint	1031
10.103.2.34	pmg_qfEnergyVolume	1031
10.103.2.35	pmg_qmEnergySMPBE	1031
10.103.2.36	pmg_splineSelect	1032
10.103.2.37	lapSolve	1032
10.104	pmg.h	1032
10.105	rc/mg/vpmgp.c File Reference	1041
10.105	Detailed Description	1042
10.106	pmgp.c	1043
10.107	rc/mg/vpmgp.h File Reference	1046
10.107	Detailed Description	1049
10.108	pmgp.h	1050
10.109	rc/routines.c File Reference	1051
10.109	Detailed Description	1054
10.110	outines.c	1055
10.111	rc/routines.h File Reference	1118
10.111	Detailed Description	1122
10.112	outines.h	1123

[Index](#)

1129

Chapter 1

APBS Programmers Guide

APBS was written by Nathan A. Baker.
Additional contributing authors listed in the code documentation.

1.1 Table of Contents

- [Programming Style](#)
- [Application programming interface documentation](#)
 - [Modules](#)
 - [Class list](#)
 - [Class members](#)
 - [Class methods](#)

1.2 License

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

Pacific Northwest National Laboratory

Additional contributing authors are listed in the code documentation.

Copyright (c) 2010-2014 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific Northwest Division for the U.S. Department of Energy.

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

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

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

Portions Copyright (c) 1995, Michael Holst.

All rights reserved.

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

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

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

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

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

1.3 Programming Style

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

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

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

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

would be declared as

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

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

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

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


would be declared as

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

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

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

in C++ would be

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

in Clean OO C.

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

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

1.4 Application programming interface documentation

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

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

Chapter 2

Deprecated List

Global **smGparm::nlev**

Just ignored now

Chapter 3

Bug List

Global **Bmat_printHB** (Bmat *thee, char *fname)

Hardwired to only handle the single block symmetric case.

Global **energyFE** (NOSH *nosh, int icalc, Vfetk *fetk[NOSH_MAXCALC], int *nenergy, double *totEnergy, double *qfEnergy, double *qmEnergy, double *dielEnergy)

"calcenergy 2" does not work

Returns

1 if successful, 0 otherwise

Global **initFE** (int icalc, NOSH *nosh, FEMparm *feparm, PBEparm *pbeparm, Vpbe *pbe[NOSH_MAXCALC], Valist *alist[NOSH_MAXMOL], Vfetk *fetk[NOSH_MAXCALC])

THIS FUNCTION IS HARD-CODED TO SOLVE LRPBE

THIS FUNCTION IS HARD-CODED TO SOLVE LRPBE

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

Returns

1 if successful, 0 otherwise

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 tvec[])

This function is not thread-safe

Global **Vfetk_PDE_initPoint** (PDE *thee, int pointType, int chart, double txq[], double tU[], double tdU[][VAPBS_DIM])

This function is not thread-safe

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

Global **Vfetk_PDE_Ju** (PDE *thee, int key)

This function is not thread-safe.

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

This function is not thread-safe

Global **Vfetk_PDE_u_D** (PDE *thee, int type, int chart, double txq[], double F[])

This function is hard-coded to call only multiple-sphere Debye-Hückel functions.

This function is not thread-safe.

Global **Vfetk_PDE_u_T** (PDE *thee, int type, int chart, double txq[], double F[])

This function is not thread-safe.

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

Some values of format are not implemented

Returns

1 if successful, 0 otherwise

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

Not implemented yet

Note

Not implemented yet

Parameters

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

Returns

1 if successful, 0 otherwise

Global **Vgreen_helmholtzD** (Vgreen *thee, int npos, double *x, double *y, double *z, double *gradx, double *grady, double *gradz, double kappa)

Not implemented yet

Note

Not implemented yet

Parameters

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

Returns

int 1 if successful, 0 otherwise

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 **Vpackmg** (int *iparm, double *rparm, size_t *nrwk, int *niwk, int *nx, int *ny, int *nz, int *nlev, int *nu1, int *nu2, int *mgkey, int *itmax, int *istop, int *ipcon, int *nonlin, int *mgsmoo, int *mgprol, int *mgcoar, int *mgsolv, int *mgdisc, int *iinfo, double *errtol, int *ipkey, double *omegal, double *omegan, int *irite, int *iperf)

Can this path variable be replaced with a Vio socket?

Global **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)

The focusing flag is currently not used!!

Returns

1 if successful, 0 otherwise

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_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)

useDielMap could only be passed once, not three times, to this function - why not just once? that's what the call in [routines.c](#) ends up doing - just passing useDielMap three times. - P. Ellis 11/3/11

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 4

Module Index

4.1 Modules

Here is a list of all modules:

dependencies	21
Vcsm class	22
Vfetc class	29
Vpee class	55
APOLparm class	60
BEMparm class	64
FEMparm class	68
GEOFLOWparm class	73
MGparm class	77
NOsh class	87
PBEparm class	103
Vacc class	109
Valist class	125
Vatom class	132
Vcap class	141
Vclist class	144
Vgreen class	150
Vhal class	158
Matrix wrapper class	167
Vparam class	168
Vpbe class	178
Vstring class	192
Vunit class	195
Vgrid class	196
Vmgrid class	206
Vopot class	211
Vpmg class	215
Vpmgp class	237
C translation of Holst group PMG code	240
High-level front-end routines	316

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

_GeoflowInput	345
_GeoflowOutput	346
AtomForce	346
Comdata	347
iterator	
Stencil< T >	390
LJ	348
Mat< T >	349
sAPOLparm	350
sBEMparm	356
sFEMparm	358
sGEOFLOWparm	362
sMGparm	363
sNOsh	372
sNOsh_calc	379
sPBEparm	380
sVacc	391
sVaccSurf	393
sValist	395
sVatom	396
sVclist	398
sVclistCell	400
sVcsm	400
sVfetk	402
sVfetk_LocalVar	406
sVgreen	410
sVgrid	412
sVmgrid	415
sVopot	415
sVparam_AtomData	416
sVpbe	418
sVpee	423
sVpmg	424
sVpmgp	431

Vparam	440
Vparam_ResData	441

Chapter 6

Data Structure Index

6.1 Data Structures

Here are the data structures with brief descriptions:

_GeoflowInput	345
_GeoflowOutput	346
AtomForce	
Structure to hold atomic forces	346
Comdata	347
LJ	348
Mat< T >	349
sAPOLparm	
Parameter structure for APOL-specific variables from input files	350
sBEMparm	
Parameter structure for BEM-specific variables from input files	356
sFEMparm	
Parameter structure for FEM-specific variables from input files	358
sGEOFLOWparm	
Parameter structure for GEOFLOW-specific variables from input files	362
sMGparm	
Parameter structure for MG-specific variables from input files	363
sNOsh	
Class for parsing fixed format input files	372
sNOsh_calc	
Calculation class for use when parsing fixed format input files	379
sPBEparm	
Parameter structure for PBE variables from input files	380
Stencil< T >	
!!	390
sVacc	
Oracle for solvent- and ion-accessibility around a biomolecule	391
sVaccSurf	
Surface object list of per-atom surface points	393
sValist	
Container class for list of atom objects	395
sVatom	
Contains public data members for Vatom class/module	396

sVclist	Atom cell list	398
sVclistCell	Atom cell list cell	400
sVcsm	Charge-simplex map class	400
sVfetc	Contains public data members for Vfetc class/module	402
sVfetc_LocalVar	Vfetc LocalVar subclass	406
sVgreen	Contains public data members for Vgreen class/module	410
sVgrid	Electrostatic potential oracle for Cartesian mesh data	412
sVmgrid	Multiresolution oracle for Cartesian mesh data	415
sVopot	Electrostatic potential oracle for Cartesian mesh data	415
sVparam_AtomData	AtomData sub-class; stores atom data	416
sVpbe	Contains public data members for Vpbe class/module	418
sVpee	Contains public data members for Vpee class/module	423
sVpmg	Contains public data members for Vpmg class/module	424
sVpmgp	Contains public data members for Vpmgp class/module	431
Vparam	Reads and assigns charge/radii parameters	440
Vparam_ResData	ResData sub-class; stores residue data	441

Chapter 7

File Index

7.1 File List

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

doc/programmer/ mainpage.h	??
src/ apbs.h	
Header file for header dependencies	443
src/ apbscfg.h	??
src/ main.c	
APBS "front end" program using formatted input files	806
src/ routines.c	
Supporting routines for APBS front end	1051
src/ routines.h	
Header file for front end auxiliary routines	1118
src/fem/ vcsn.c	
Class Vcsn methods	446
src/fem/ vcsn.h	
Contains declarations for the Vcsn class	454
src/fem/ vfetk.c	
Class Vfetk methods	459
src/fem/ vfetk.h	
Contains declarations for class Vfetk	495
src/fem/ vpee.c	
Class Vpee methods	506
src/fem/ vpee.h	
Contains declarations for class Vpee	514
src/generic/ apolparm.c	
Class APOLparm methods	518
src/generic/ apolparm.h	??
src/generic/ bemparm.c	
Class BEMparm methods	527
src/generic/ bemparm.h	??
src/generic/ femparm.c	
Class FEMparm methods	532
src/generic/ femparm.h	
Contains declarations for class APOLparm	540
src/generic/ geoflowparm.c	
Class GEOFLOWparm methods	544

src/generic/geoflowparm.h	
Contains declarations for class GEOFLOWparm	549
src/generic/mgparm.c	
Class MGparm methods	553
src/generic/mgparm.h	
Contains declarations for class MGparm	567
src/generic/nosh.c	
Class NOsh methods	572
src/generic/nosh.h	
Contains declarations for class NOsh	610
src/generic/pbeparm.c	
Class PBEparm methods	617
src/generic/pbeparm.h	
Contains declarations for class PBEparm	636
src/generic/vacc.c	
Class Vacc methods	641
src/generic/vacc.h	
Contains declarations for class Vacc	668
src/generic/valist.c	
Class Valist methods	676
src/generic/valist.h	
Contains declarations for class Valist	689
src/generic/vatom.c	
Class Vatom methods	694
src/generic/vatom.h	
Contains declarations for class Vatom	699
src/generic/vcap.c	
Class Vcap methods	704
src/generic/vcap.h	
Contains declarations for class Vcap	707
src/generic/vclist.c	
Class Vclist methods	710
src/generic/vclist.h	
Contains declarations for class Vclist	718
src/generic/vgreen.c	
Class Vgreen methods	723
src/generic/vgreen.h	
Contains declarations for class Vgreen	731
src/generic/vhal.h	
Contains generic macro definitions for APBS	734
src/generic/vmatrix.h	
Contains inclusions for matrix data wrappers	751
src/generic/vparam.c	
Class Vparam methods	754
src/generic/vparam.h	
Contains declarations for class Vparam	765
src/generic/vpbe.c	
Class Vpbe methods	770
src/generic/vpbe.h	
Contains declarations for class Vpbe	779
src/generic/vstring.c	
Class Vstring methods	785
src/generic/vstring.h	
Contains declarations for class Vstring	790

src/generic/vunit.h	
Contains a collection of useful constants and conversion factors	792
src/geoflow/cpbconcz2.h	
Some definitons for APBS	795
src/geoflow/Mat.h	
Wrapper class for 2d and 3d arrays	798
src/geoflow/modules.h	
FORTRAN globals and function headers	804
src/mg/vgrid.c	
Class Vgrid methods	818
src/mg/vgrid.h	
Potential oracle for Cartesian mesh data	840
src/mg/vmgrid.c	
Class Vmgrid methods	847
src/mg/vmgrid.h	
Multiresolution oracle for Cartesian mesh data	851
src/mg/vopot.c	
Class Vopot methods	855
src/mg/vopot.h	
Potential oracle for Cartesian mesh data	861
src/mg/vpmg.c	
Class Vpmg methods	864
src/mg/vpmg.h	
Contains declarations for class Vpmg	1015
src/mg/vpmgp.c	
Class Vpmgp methods	1041
src/mg/vpmgp.h	
Contains declarations for class Vpmgp	1046
src/pmgc/buildAd.c	??
src/pmgc/buildAd.h	??
src/pmgc/buildBd.c	??
src/pmgc/buildBd.h	??
src/pmgc/buildGd.c	??
src/pmgc/buildGd.h	??
src/pmgc/buildPd.c	??
src/pmgc/buildPd.h	??
src/pmgc/cgd.c	??
src/pmgc/cgd.h	??
src/pmgc/gsd.c	??
src/pmgc/gsd.h	??
src/pmgc/matvecd.c	??
src/pmgc/matvecd.h	??
src/pmgc/mgcsd.c	??
src/pmgc/mgcsd.h	??
src/pmgc/mgdrvd.c	??
src/pmgc/mgdrvd.h	??
src/pmgc/mgfasd.c	??
src/pmgc/mgfasd.h	??
src/pmgc/mgsubd.c	??
src/pmgc/mgsubd.h	??
src/pmgc/mikpckd.c	??
src/pmgc/mikpckd.h	??
src/pmgc/mlinpckd.c	??
src/pmgc/mlinpckd.h	??

src/pmgc/ mypdec.c	??
src/pmgc/ mypdec.h	??
src/pmgc/ newdrv.c	??
src/pmgc/ newdrv.h	??
src/pmgc/ newton.c	??
src/pmgc/ newton.h	??
src/pmgc/ power.c	??
src/pmgc/ power.h	??
src/pmgc/ smooth.c	??
src/pmgc/ smooth.h	??

Chapter 8

Module Documentation

8.1 dependencies

8.2 Vcsm class

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

Files

- file [vcsm.c](#)
Class Vcsm methods.
- file [vcsm.h](#)
Contains declarations for the Vcsm class.

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_getNumberSimplexes](#) ([Vcsm](#) *thee, int iatom)
Get number of simplexes 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)
FORTTRAN stub to construct Vcsm object.
- VEXTERNC void [Vcsm_dtor](#) ([Vcsm](#) **thee)

Destroy Vcsm object.

- VEXTERNC void `Vcsm_dtor2` (`Vcsm *thee`)

FORTTRAN 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.

8.2.1 Detailed Description

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

8.2.2 Function Documentation

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

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

Author

Nathan Baker

Parameters

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

8.2.2.2 VEXTERNC `Vcsm*` `Vcsm_ctor` (`Valist * alist`, `Gem * gm`)

Construct Vcsm object.

Author

Nathan Baker

Note

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

Returns

Pointer to newly allocated Vcsm object

Parameters

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

Definition at line 136 of file [vscm.c](#).

8.2.2.3 VEXTERNC int Vscm_ctor2 (Vscm * *thee*, Valist * *alist*, Gem * *gm*)

FORTTRAN stub to construct Vscm 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 Vscm_init is called

Returns

1 if successful, 0 otherwise

Parameters

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

Definition at line 147 of file [vscm.c](#).

8.2.2.4 VEXTERNC void Vscm_dtor (Vscm ** *thee*)

Destroy Vscm object.

Author

Nathan Baker

Parameters

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

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

8.2.2.5 VEXTERNC void Vscm_dtor2 (Vscm * *thee*)

FORTTRAN stub to destroy Vscm object.

Author

Nathan Baker

Parameters

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

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

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

Get particular atom associated with a simplex.

Author

Nathan Baker

Returns

Array of atoms associated with a simplex

Parameters

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

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

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

Get ID of particular atom in a simplex.

Author

Nathan Baker

Returns

Index of atom in Valist object

Parameters

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

Definition at line 88 of file [vcsm.c](#).

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

Get number of atoms associated with a simplex.

Author

Nathan Baker

Returns

Number of atoms associated with a simplex

Parameters

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

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

8.2.2.9 VEXTERNC int Vcsm_getNumberSimplexes (Vcsm * *thee*, int *iatom*)

Get number of simplices associated with an atom.

Author

Nathan Baker

Returns

Number of simplices associated with an atom

Parameters

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

Definition at line 99 of file [vcsm.c](#).

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

Get particular simplex associated with an atom.

Author

Nathan Baker

Returns

Pointer to simplex object

Parameters

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

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

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

Get index particular simplex associated with an atom.

Author

Nathan Baker

Returns

Gem index of specified simplex

Parameters

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

Definition at line 119 of file [vcsm.c](#).

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

Get atom list.

Author

Nathan Baker

Returns

Pointer to Valist atom list

Parameters

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

Definition at line 62 of file [vcsm.c](#).

8.2.2.13 VEXTERNC void Vcsm_init (Vcsm * *thee*)

Initialize charge-simplex map with mesh and atom data.

Author

Nathan Baker

Note

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

Parameters

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

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

8.2.2.14 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

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

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

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

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

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Parameters

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

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

8.3 Vfetk class

FETk master class (interface between FETk and APBS)

Files

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

Data Structures

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

Macros

- #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` { `VG_T_ZERO` =0, `VG_T_DIRI` =1, `VG_T_PREV` =2 }
Initial guess type.
- enum `eVfetk_PrecType` { `VPT_IDEN` =0, `VPT_DIAG` =1, `VPT_MG` =2 }
Preconditioner type.

Functions

- VPUBLIC double `Vfetk_energy` (`Vfetk *thee`, int color, int nonlin)
Return the total electrostatic energy.
- 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)
FORTTRAN stub constructor for Vfetk object.
- VEXTERNC void `Vfetk_dtor` (`Vfetk **thee`)
Object destructor.
- VEXTERNC void `Vfetk_dtor2` (`Vfetk *thee`)
FORTTRAN stub object destructor.
- VEXTERNC double * `Vfetk_getSolution` (`Vfetk *thee`, int *length)
Create an array containing the solution (electrostatic potential in units of $k_B T / e$) at the finest mesh level.
- VEXTERNC void `Vfetk_setParameters` (`Vfetk *thee`, `PBEparm *pbeparm`, `FEMparm *feparm`)
Set the parameter objects.
- VEXTERNC double `Vfetk_dqmEnergy` (`Vfetk *thee`, int color)
Get the "mobile charge" and "polarization" contributions to the electrostatic energy.
- VEXTERNC double `Vfetk_qfEnergy` (`Vfetk *thee`, int color)
Get the "fixed charge" contribution to the electrostatic energy.
- VEXTERNC unsigned long int `Vfetk_memChk` (`Vfetk *thee`)
Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC void `Vfetk_setAtomColors` (`Vfetk *thee`)
Transfer color (partition ID) information from a partitioned mesh to the atoms.

- VEXTERNC void **Bmat_printHB** (Bmat *thee, char *fname)
Writes a Bmat to disk in Harwell-Boeing sparse matrix format.
- VEXTERNC Vrc_Codes **Vfetk_genCube** (Vfetk *thee, double center[3], double length[3], Vfetk_MeshLoad meshType)
Construct a rectangular mesh (in the current Vfetk object)
- VEXTERNC Vrc_Codes **Vfetk_loadMesh** (Vfetk *thee, double center[3], double length[3], Vfetk_MeshLoad meshType, Vio *sock)
Loads a mesh into the Vfetk (and associated) object(s).
- VEXTERNC PDE * **Vfetk_PDE_ctor** (Vfetk *fetk)
Constructs the FEtk PDE object.
- VEXTERNC int **Vfetk_PDE_ctor2** (PDE *thee, Vfetk *fetk)
Intializes the FEtk PDE object.
- VEXTERNC void **Vfetk_PDE_dtor** (PDE **thee)
Destroys FEtk PDE object.
- VEXTERNC void **Vfetk_PDE_dtor2** (PDE *thee)
FORTTRAN stub: destroys FEtk PDE object.
- VEXTERNC void **Vfetk_PDE_initAssemble** (PDE *thee, int ip[], double rp[])
Do once-per-assembly initialization.
- VEXTERNC void **Vfetk_PDE_initElement** (PDE *thee, int elementType, int chart, double tvx[][VAPBS_DIM], void *data)
Do once-per-element initialization.
- VEXTERNC void **Vfetk_PDE_initFace** (PDE *thee, int faceType, int chart, double tvec[])
Do once-per-face initialization.
- VEXTERNC void **Vfetk_PDE_initPoint** (PDE *thee, int pointType, int chart, double txq[], double tU[], double td←U[][VAPBS_DIM])
Do once-per-point initialization.
- VEXTERNC void **Vfetk_PDE_Fu** (PDE *thee, int key, double F[])
Evaluate strong form of PBE. For interior points, this is:

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

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

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

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

- VEXTERNC double **Vfetk_PDE_Fu_v** (PDE *thee, int key, double V[], double dV[][VAPBS_DIM])
This is the weak form of the PBE; i.e. the strong form integrated with a test function to give:

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

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

- VEXTERNC double **Vfetk_PDE_DFu_wv** (PDE *thee, int key, double W[], double dW[][VAPBS_DIM], double V[], double dV[][VAPBS_DIM])
This is the linearization of the weak form of the PBE; e.g., for use in a Newton iteration. This is the functional linearization of the strong form integrated with a test function to give:

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

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

- VEXTERNC void [Vfetk_PDE_delta](#) (PDE *thee, int type, int chart, double txq[], void *user, double F[])
Evaluate a (discretized) delta function source term at the given point.
- VEXTERNC void [Vfetk_PDE_u_D](#) (PDE *thee, int type, int chart, double txq[], double F[])
Evaluate the Dirichlet boundary condition at the given point.
- VEXTERNC void [Vfetk_PDE_u_T](#) (PDE *thee, int type, int chart, double txq[], double F[])
Evaluate the "true solution" at the given point for comparison with the numerical solution.
- VEXTERNC void [Vfetk_PDE_bisectEdge](#) (int dim, int dimII, int edgeType, int chart[], double vx[][VAPBS_DIM])
Define the way manifold edges are bisected.
- VEXTERNC void [Vfetk_PDE_mapBoundary](#) (int dim, int dimII, int vertexType, int chart, double vx[VAPBS_DIM])
Map a boundary point to some pre-defined shape.
- VEXTERNC int [Vfetk_PDE_markSimplex](#) (int dim, int dimII, int simplexType, int faceType[VAPBS_NVS], int vertexType[VAPBS_NVS], int chart[], double vx[][VAPBS_DIM], void *simplex)
User-defined error estimator – in our case, a geometry-based refinement method; forcing simplex refinement at the dielectric boundary and (for non-regularized PBE) the charges.
- VEXTERNC void [Vfetk_PDE_oneChart](#) (int dim, int dimII, int objType, int chart[], double vx[][VAPBS_DIM], int dimV)
Unify the chart for different coordinate systems – a no-op for us.
- VEXTERNC double [Vfetk_PDE_Ju](#) (PDE *thee, int key)
Energy functional. This returns the energy (less delta function terms) in the form:
$$c^{-1}/2 \int (\epsilon(\nabla u)^2 + \kappa^2(\cosh u - 1)) dx$$

for a 1:1 electrolyte where c is the output from `Vpbe_getZmagic`.
- VEXTERNC void [Vfetk_externalUpdateFunction](#) (SS **simps, int num)
External hook to simplex subdivision routines in Gem. Called each time a simplex is subdivided (we use it to update the charge-simplex map)
- VEXTERNC int [Vfetk_PDE_simplexBasisInit](#) (int key, int dim, int comp, int *ndof, int dof[])
Initialize the bases for the trial or the test space, for a particular component of the system, at all quadrature points on the master simplex element.
- VEXTERNC void [Vfetk_PDE_simplexBasisForm](#) (int key, int dim, int comp, int pdkey, double xq[], double basis[])
Evaluate the bases for the trial or test space, for a particular component of the system, at all quadrature points on the master simplex element.
- VEXTERNC void [Vfetk_readMesh](#) (Vfetk *thee, int skey, Vio *sock)
Read in mesh and initialize associated internal structures.
- VEXTERNC void [Vfetk_dumpLocalVar](#) ()
Debugging routine to print out local variables used by PDE object.
- VEXTERNC int [Vfetk_fillArray](#) (Vfetk *thee, Bvec *vec, [Vdata_Type](#) type)
Fill an array with the specified data.
- VEXTERNC int [Vfetk_write](#) (Vfetk *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, Bvec *vec, [Vdata_Format](#) format)
Write out data.
- VEXTERNC Vrc_Codes [Vfetk_loadGem](#) (Vfetk *thee, Gem *gm)
Load a Gem geometry manager object into Vfetk.

8.3.1 Detailed Description

FEtk master class (interface between FEtk and APBS)

8.3.2 Enumeration Type Documentation

8.3.2.1 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 138 of file [vfetk.h](#).

8.3.2.2 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 86 of file [vfetk.h](#).

8.3.2.3 enum eVfetk_MeshLoad

Mesh loading operation.

Enumerator

VML_DIRICUBE Dirichlet cube

VML_NEUMCUBE Neumann cube

VML_EXTERNAL External mesh (from socket)

Definition at line 104 of file [vfetk.h](#).

8.3.2.4 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 121 of file [vfetk.h](#).

8.3.2.5 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 155 of file [vfetk.h](#).

8.3.3 Function Documentation

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

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

Author

Stephen Bond

Note

This is a friend function of Bmat

Bug Hardwired to only handle the single block symmetric case.

Parameters

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

Definition at line 1026 of file [vfetk.c](#).

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

Constructor for Vfetk object.

Author

Nathan Baker

Returns

Pointer to newly allocated Vfetk object

Note

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

Parameters

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

Definition at line [532](#) of file [vfetk.c](#).

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

FORTTRAN stub constructor for Vfetk object.

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Note

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

Parameters

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

Definition at line [545](#) of file [vfetk.c](#).

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

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

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

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

for the NPBE and

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

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

Author

Nathan Baker

Parameters

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

Returns

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

Parameters

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

Definition at line 842 of file [vfetc.c](#).

8.3.3.5 VEXTERNC void Vfetc_dtor (Vfetc ** *thee*)

Object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 625 of file [vfetc.c](#).

8.3.3.6 VEXTERNC void Vfetc_dtor2 (Vfetc * *thee*)

FORTTRAN stub object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 633 of file [vfetc.c](#).

8.3.3.7 VEXTERNC void Vfetc_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 2255 of file [vfetc.c](#).

8.3.3.8 VEXTERNC double Vfetc_energy (Vfetc * *thee*, int *color*, int *nonlin*)

Return the total electrostatic energy.

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

Author

Nathan Baker

Returns

Total electrostatic energy in units of $k_B T$.

< Total energy

<

<

Parameters

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

Definition at line 693 of file [vfetc.c](#).

8.3.3.9 VEXTERNC void Vfetc_externalUpdateFunction (SS ** *simps*, int *num*)

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

Author

Nathan Baker

Bug This function is not thread-safe.**Parameters**

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

Definition at line 2078 of file [vfetk.c](#).**8.3.3.10 VEXTERNC int Vfetk_fillArray (Vfetk * *thee*, Bvec * *vec*, Vdata_Type *type*)**

Fill an array with the specified data.

Author

Nathan Baker

Note

This function is thread-safe

Bug Several values of type are not implemented**Returns**

1 if successful, 0 otherwise

Parameters

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

Definition at line 2299 of file [vfetk.c](#).**8.3.3.11 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

Generates a new cube mesh within the provided Vfetk object based on the specified mesh type. Creates a new copy of the mesh based on the global variables at the top of the file and the mesh type, then recenters the mesh based on the center and length variables provided to the function.

Parameters

<i>thee</i>	Vfetk object
<i>center</i>	Center for mesh, which the new mesh will adjust to
<i>length</i>	Mesh lengths, which the new mesh will adjust to
<i>meshType</i>	Mesh boundary conditions

Definition at line 885 of file [vfetk.c](#).

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

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

Author

Nathan Baker

Returns

Pointer to the AM (algebra manager) object

Parameters

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

Definition at line 497 of file [vfetk.c](#).

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

Get the partition information for a particular atom.

Author

Nathan Baker

Note

Friend function of Vatom

Returns

Partition ID

Parameters

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

Definition at line 517 of file [vfetk.c](#).

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

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

Author

Nathan Baker

Returns

Pointer to the Gem (grid manager) object

Parameters

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

Definition at line 490 of file [vfetk.c](#).

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

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

Author

Nathan Baker and Michael Holst

Note

The user is responsible for destroying the newly created array

Returns

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

Parameters

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

Definition at line 641 of file [vfetk.c](#).

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

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

Author

Nathan Baker

Returns

Pointer to the Vcsm (charge-simplex map) object

Parameters

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

Definition at line 510 of file [vfetk.c](#).

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

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

Author

Nathan Baker

Returns

Pointer to the Vpbe (PBE manager) object

Parameters

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

Definition at line 503 of file [vfetk.c](#).

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

Load a Gem geometry manager object into Vfetk.

Author

Nathan Baker

Parameters

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

8.3.3.19 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

If we have an external mesh, load that external mesh from the provided socket. If we specify a non-external mesh type, we generate a new mesh cube based on templates. We then create and store a new Vcsm object in our Vfetk structure, which will carry the mesh data.

Parameters

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

Definition at line 980 of file [vfetc.c](#).

8.3.3.20 VEXTERNC unsigned long int Vfetc_memChk (Vfetc * *thee*)

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

Author

Nathan Baker

Returns

The memory used by this structure and its contents in bytes

Parameters

<i>thee</i>	THE Vfetc object
-------------	------------------

Definition at line 867 of file [vfetc.c](#).

8.3.3.21 VEXTERNC void Vfetc_PDE_bisectEdge (int *dim*, int *dimll*, int *edgeType*, int *chart*[], double *vx*[][VAPBS_DIM])

Define the way manifold edges are bisected.

Author

Nathan Baker and Mike Holst

Note

This function is thread-safe.

Parameters

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

8.3.3.22 VEXTERNC PDE* Vfetc_PDE_ctor (Vfetc * *fetc*)

Constructs the FEtk PDE object.

Author

Nathan Baker

Returns

Newly-allocated PDE object

Bug Not thread-safe

Parameters

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

Definition at line 1176 of file [vfetk.c](#).

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

Initializes the FEtk PDE object.

Author

Nathan Baker (with code by Mike Holst)

Returns

1 if successful, 0 otherwise

Bug Not thread-safe

Parameters

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

Definition at line 1187 of file [vfetk.c](#).

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

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

Author

Nathan Baker

Bug This function is not thread-safe

Parameters

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

Definition at line 1780 of file [vfetk.c](#).

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

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

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

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

Author

Nathan Baker and Mike Holst

Returns

Integrand value

Bug This function is not thread-safe

Parameters

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

8.3.3.26 `VEXTERNC void Vfetk_PDE_dtor (PDE ** thee)`

Destroys FEtk PDE object.

Author

Nathan Baker

Note

Thread-safe

Parameters

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

Definition at line 1231 of file [vfetc.c](#).

8.3.3.27 VEXTERNC void Vfetc_PDE_dtor2 (PDE * *thee*)

FORTTRAN stub: destroys FETk PDE object.

Author

Nathan Baker

Note

Thread-safe

Parameters

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

Definition at line 1246 of file [vfetc.c](#).

8.3.3.28 VEXTERNC void Vfetc_PDE_Fu (PDE * *thee*, int *key*, double *F*[])

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

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

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

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

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

Author

Nathan Baker

Bug This function is not thread-safe

This function is not implemented (sets error to zero)

Parameters

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

Definition at line 1695 of file [vfetc.c](#).

8.3.3.29 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:

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

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

Author

Nathan Baker and Mike Holst

Returns

Integrand value

Bug This function is not thread-safe

Parameters

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

Definition at line 1703 of file [vfetk.c](#).

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

Do once-per-assembly initialization.

Author

Nathan Baker and Mike Holst

Note

Thread-safe

Parameters

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

Definition at line 1508 of file [vfetk.c](#).

8.3.3.31 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

Bug This function is not thread-safe

Parameters

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

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

Do once-per-face initialization.

Author

Nathan Baker and Mike Holst

Bug This function is not thread-safe

Parameters

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

Definition at line 1559 of file [vfetk.c](#).

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

Do once-per-point initialization.

Author

Nathan Baker

Bug This function is not thread-safe

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

Parameters

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

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

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

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

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

Author

Nathan Baker

Returns

Energy value (in kT)

Bug This function is not thread-safe.

Parameters

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

Definition at line 2003 of file [vfetk.c](#).

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

Map a boundary point to some pre-defined shape.

Author

Nathan Baker and Mike Holst

Note

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

Parameters

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

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

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

Author

Nathan Baker

Returns

1 if mark simplex for refinement, 0 otherwise

Bug This function is not thread-safe

Parameters

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

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

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

Author

Nathan Baker

Note

Thread-safe; a no-op

Parameters

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

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

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

Author

Mike Holst

Parameters

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

Definition at line 2203 of file [vfetk.c](#).

8.3.3.39 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_1 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

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

Definition at line 2140 of file [vfetk.c](#).

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

Evaluate the Dirichlet boundary condition at the given point.

Author

Nathan Baker

Bug This function is hard-coded to call only multiple-sphere Debye-Hü functions.

This function is not thread-safe.

Parameters

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

Definition at line 1867 of file [vfetk.c](#).

8.3.3.41 VEXTERNC void Vfetc_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.

The signature here doesn't match what's in mc's src/pde/mc/pde.h, which g++ seems to dislike for GAMer integration. Trying a change of function signature to match to see if that makes g++ happy. Also see [vfetc.h](#) for similar signature change. - P. Ellis 11-8-2011

Parameters

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

Definition at line 1886 of file [vfetc.c](#).

8.3.3.42 VEXTERNC double Vfetc_qfEnergy (Vfetc * *thee*, int *color*)

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

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

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

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

Author

Nathan Baker

Parameters

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

Returns

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

Parameters

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

Definition at line 732 of file [vfetk.c](#).

8.3.3.43 VEXTERNC void Vfetk_readMesh (Vfetk * *thee*, int *skey*, Vio * *sock*)

Read in mesh and initialize associated internal structures.

Author

Nathan Baker

Note

See also

[Vfetk_genCube](#)

Parameters

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

8.3.3.44 VEXTERNC void Vfetk_setAtomColors (Vfetk * *thee*)

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

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

Warning

This function should only be used immediately after mesh partitioning

Author

Nathan Baker

Note

This is a friend function of Vcsm

Parameters

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

Definition at line 849 of file [vfetk.c](#).

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

Set the parameter objects.

Author

Nathan Baker

Parameters

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

Definition at line 615 of file [vfetk.c](#).

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

Write out data.

Author

Nathan Baker

Parameters

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

Note

This function is thread-safe

Bug Some values of format are not implemented

Returns

1 if successful, 0 otherwise

Parameters

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

Definition at line [2464](#) of file [vfetk.c](#).

8.4 Vpee class

This class provides some functionality for error esimation in parallel.

Files

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

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)
FORTTRAN stub to construct the Vpee object.
- VEXTERNC void [Vpee_dtor](#) ([Vpee](#) **thee)
Object destructor.
- VEXTERNC void [Vpee_dtor2](#) ([Vpee](#) *thee)
FORTTRAN 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.

8.4.1 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.

8.4.2 Function Documentation

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

Construct the Vpee object.

Author

Nathan Baker

Returns

Newly constructed Vpee object

Parameters

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

See also

killFlag for usage

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

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

FORTTRAN stub to construct the Vpee object.

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Parameters

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

See also

killFlag for usage

Definition at line 114 of file [vpee.c](#).

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

Object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 225 of file [vpee.c](#).

8.4.2.4 VEXTERNC void Vpee_dtor2 (Vpee * *thee*)

FORTTRAN stub object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 240 of file [vpee.c](#).

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

Mark simplices for refinement based on attenuated error estimates.

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

Author

Nathan Baker and Mike Holst

Note

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

Returns

The number of simplices marked for refinement.

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

Parameters

<i>thee</i>	The Vpee object
<i>am</i>	The FEtk algebra manager currently used to solve the PB
<i>level</i>	The current level of the multigrid hierarchy
<i>akey</i>	The marking method: <ul style="list-style-type: none"> • -1: Reset markings → killFlag has no effect. • 0: Uniform. • 1: User defined (geometry-based). • >1: A numerical estimate for the error has already been set in am and should be attenuated according to killFlag and used, in conjunction with etol, to mark simplices for refinement.
<i>rcol</i>	The ID of the main partition on which to mark (or -1 if all partitions should be marked). NOte that we shouldhave (rcol == thee->localPartID) for (thee->killFlag == 2 or 3)
<i>etol</i>	The error tolerance criterion for marking
<i>bkey</i>	How the error tolerance is interpreted: <ul style="list-style-type: none"> • 0: Simplex marked if error > etol. • 1: Simplex marked if error > sqrt(etol²/L) where L\$ is the number of simplices

Definition at line 250 of file [vpee.c](#).

8.4.2.6 VEXTERNC int Vpee_numSS (Vpee * *thee*)

Returns the number of simplices in the local partition.

Author

Nathan Baker

Returns

Number of simplices in the local partition

Parameters

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

Definition at line [479](#) of file [vpee.c](#).

8.5 APOLparm class

Parameter structure for APOL-specific variables from input files.

Files

- file [apolparm.c](#)
Class APOLparm methods.
- file [femparm.h](#)
Contains declarations for class APOLparm.

Data Structures

- struct [sAPOLparm](#)
Parameter structure for APOL-specific variables from input files.

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)
FORTTRAN stub to construct APOLparm.
- VEXTERNC void [APOLparm_dtor](#) ([APOLparm](#) **thee)
Object destructor.
- VEXTERNC void [APOLparm_dtor2](#) ([APOLparm](#) *thee)
FORTTRAN 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.

8.5.1 Detailed Description

Parameter structure for APOL-specific variables from input files.

8.5.2 Enumeration Type Documentation

8.5.2.1 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 [79](#) of file [apolparm.h](#).

8.5.2.2 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 [95](#) of file [apolparm.h](#).

8.5.2.3 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 [111](#) of file [apolparm.h](#).

8.5.3 Function Documentation

8.5.3.1 VEXTERNC Vrc_Codes APOLparm_check (APOLparm * *thee*)

Consistency check for parameter values stored in object.

Author

David Gohara, Yong Huang

Parameters

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

Returns

Success enumeration

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

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

Copy target object into *thee*.

Author

Nathan Baker

Parameters

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

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

8.5.3.3 VEXTERNC APOLparm* APOLparm_ctor ()

Construct APOLparm.

Author

David Gohara

Returns

Newly allocated and initialized Vpmgp object

Definition at line 65 of file [apolparm.c](#).

8.5.3.4 VEXTERNC Vrc_Codes APOLparm_ctor2 (APOLparm * *thee*)

FORTTRAN stub to construct APOLparm.

Author

David Gohara, Yong Huang

Parameters

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

Returns

Success enumeration

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

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

Object destructor.

Author

David Gohara

Parameters

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

Definition at line 167 of file [apolparm.c](#).

8.5.3.6 VEXTERNC void APOLparm_dtor2 (APOLparm * *thee*)

FORTTRAN stub for object destructor.

Author

David Gohara

Parameters

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

Definition at line 177 of file [apolparm.c](#).

8.6 BEMparm class

Parameter which holds useful parameters for generic multigrid calculations.

Files

- file [bemparm.c](#)
Class BEMparm methods.

Data Structures

- struct [sBEMparm](#)
Parameter structure for BEM-specific variables from input files.

Typedefs

- typedef enum [eBEMparm_CalcType](#) [BEMparm_CalcType](#)
Declare BEMparm_CalcType type.
- typedef struct [sBEMparm](#) [BEMparm](#)
Parameter structure for BEM-specific variables from input files.

Enumerations

- enum [eBEMparm_CalcType](#) { [BCT_MANUAL](#) =0, [BCT_NONE](#) =1 }
Calculation type.

Functions

- VEXTERNC [BEMparm](#) * [BEMparm_ctor](#) ([BEMparm_CalcType](#) type)
Construct BEMparm object.
- VEXTERNC Vrc_Codes [BEMparm_ctor2](#) ([BEMparm](#) *thee, [BEMparm_CalcType](#) type)
FORTTRAN stub to construct BEMparm object.
- VEXTERNC void [BEMparm_dtor](#) ([BEMparm](#) **thee)
Object destructor.
- VEXTERNC void [BEMparm_dtor2](#) ([BEMparm](#) *thee)
FORTTRAN stub for object destructor.
- VEXTERNC Vrc_Codes [BEMparm_check](#) ([BEMparm](#) *thee)
Consistency check for parameter values stored in object.
- VEXTERNC Vrc_Codes [BEMparm_parseToken](#) ([BEMparm](#) *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 Typedef Documentation

8.6.2.1 typedef struct sBEMparm BEMparm

Parameter structure for BEM-specific variables from input files.

Author

Nathan Baker and Todd Dolinsky and Weihua Geng

Note

If you add/delete/change something in this class, the member functions – especially BEMparm_copy – must be modified accordingly

8.6.3 Enumeration Type Documentation

8.6.3.1 enum eBEMparm_CalcType

Calculation type.

Enumerator

BCT_MANUAL bem-manual

BCT_NONE not defined

Definition at line 77 of file [bemparm.h](#).

8.6.4 Function Documentation

8.6.4.1 VEXTERNC Vrc_Codes BEMparm_check (BEMparm * *thee*)

Consistency check for parameter values stored in object.

Author

Nathan Baker

Parameters

<i>thee</i>	BEMparm object
-------------	----------------

Returns

Success enumeration

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

8.6.4.2 VEXTERNC BEMparm* BEMparm_ctor (BEMparm_CalcType *type*)

Construct BEMparm object.

Author

Nathan Baker

Parameters

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

Returns

Newly allocated and initialized BEMparm object

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

8.6.4.3 VEXTERNC Vrc_Codes BEMparm_ctor2 (BEMparm * *thee*, BEMparm_CalcType *type*)

FORTTRAN stub to construct BEMparm object.

Author

Nathan Baker and Todd Dolinsky

Parameters

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

Returns

Success enumeration

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

8.6.4.4 VEXTERNC void BEMparm_dtor (BEMparm ** *thee*)

Object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 108 of file [bemparm.c](#).

8.6.4.5 VEXTERNC void BEMparm_dtor2 (BEMparm * *thee*)

FORTTRAN stub for object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 116 of file [bemparm.c](#).

8.6.4.6 VEXTERNC Vrc_Codes BEMparm_parseToken (BEMparm * *thee*, char *tok*[VMAX_BUFSIZE], Vio * *sock*)

Parse an MG keyword from an input file.

Author

Nathan Baker and Todd Dolinsky

Parameters

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

Returns

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

Definition at line 257 of file [bemparm.c](#).

8.7 FEMparm class

Parameter structure for FEM-specific variables from input files.

Files

- file [femparm.c](#)
Class FEMparm methods.
- file [femparm.h](#)
Contains declarations for class APOLparm.

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 refinement error estimate tolerance key.
- enum [eFEMparm_EstType](#) {
FRT_UNIF =0, FRT_GEOM =1, FRT_RESI =2, FRT_DUAL =3,
FRT_LOCA =4 }
Adaptive refinement error estimator method.
- enum [eFEMparm_CalcType](#) { FCT_MANUAL, FCT_NONE }
Calculation type.

Functions

- VEXTERNC [FEMparm](#) * [FEMparm_ctor](#) (FEMparm_CalcType type)
Construct FEMparm.
- VEXTERNC int [FEMparm_ctor2](#) (FEMparm *thee, FEMparm_CalcType type)
FORTTRAN stub to construct FEMparm.
- VEXTERNC void [FEMparm_dtor](#) (FEMparm **thee)
Object destructor.

- VEXTERN void [FEMparm_dtor2](#) ([FEMparm](#) *thee)
FORTTRAN stub for object destructor.
- VEXTERN int [FEMparm_check](#) ([FEMparm](#) *thee)
Consistency check for parameter values stored in object.
- VEXTERN void [FEMparm_copy](#) ([FEMparm](#) *thee, [FEMparm](#) *source)
Copy target object into thee.

8.7.1 Detailed Description

Parameter structure for FEM-specific variables from input files.

8.7.2 Typedef Documentation

8.7.2.1 typedef enum eFEMparm_EtolType FEMparm_EtolType

Declare FEparm_EtolType type.

Author

Nathan Baker

Definition at line 90 of file [femparm.h](#).

8.7.3 Enumeration Type Documentation

8.7.3.1 enum eFEMparm_CalcType

Calculation type.

Enumerator

FCT_MANUAL fe-manual

FCT_NONE unspecified

Definition at line 117 of file [femparm.h](#).

8.7.3.2 enum eFEMparm_EstType

Adaptive refinement error estimator method.

Note

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

Author

Nathan Baker

Enumerator

- FRT_UNIF** Uniform refinement
- FRT_GEOM** Geometry-based (i.e. surfaces and charges) refinement
- FRT_RESI** Nonlinear residual estimate-based refinement
- FRT_DUAL** Dual-solution weight nonlinear residual estimate-based refinement
- FRT_LOCA** Local problem error estimate-based refinement

Definition at line 98 of file [femparm.h](#).

8.7.3.3 enum eFEMparm_EtolType

Adaptive refinement error estimate tolerance key.

Author

Nathan Baker

Enumerator

- FET_SIMP** per-simplex error tolerance
- FET_GLOB** global error tolerance
- FET_FRAC** fraction of simplices we want to have refined

Definition at line 79 of file [femparm.h](#).

8.7.4 Function Documentation

8.7.4.1 VEXTERNC int FEMparm_check (FEMparm * *thee*)

Consistency check for parameter values stored in object.

Author

Nathan Baker

Parameters

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

Returns

1 if OK, 0 otherwise

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

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

Copy target object into thee.

Author

Nathan Baker

Parameters

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

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

8.7.4.3 VEXTERNC FEMparm* FEMparm_ctor (FEMparm_CalcType *type*)

Construct FEMparm.

Author

Nathan Baker

Parameters

<i>type</i>	FEM calculation type
-------------	----------------------

Returns

Newly allocated and initialized Vpmgp object

Definition at line 65 of file [femparm.c](#).

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

FORTTRAN stub to construct FEMparm.

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

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

8.7.4.5 VEXTERN void FEMparm_dtor (FEMparm ** *thee*)

Object destructor.

Author

Nathan Baker

Parameters

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

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

8.7.4.6 VEXTERN void FEMparm_dtor2 (FEMparm * *thee*)

FORTTRAN stub for object destructor.

Author

Nathan Baker

Parameters

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

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

8.8 GEOFLOWparm class

Parameter which holds useful parameters for GEOFLOWeric multigrid calculations.

Files

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

Data Structures

- struct [sGEOFLOWparm](#)
Parameter structure for GEOFLOW-specific variables from input files.

Typedefs

- typedef enum [eGEOFLOWparm_CalcType](#) [GEOFLOWparm_CalcType](#)
Declare GEOFLOWparm_CalcType type.
- typedef struct [sGEOFLOWparm](#) [GEOFLOWparm](#)
Parameter structure for GEOFLOW-specific variables from input files.

Enumerations

- enum [eGEOFLOWparm_CalcType](#) { [GFCT_MANUAL](#) =0, [GFCT_AUTO](#) =1, [GFCT_NONE](#) =2 }
Calculation type.

Functions

- VEXTERNC [GEOFLOWparm](#) * [GEOFLOWparm_ctor](#) ([GEOFLOWparm_CalcType](#) type)
Construct GEOFLOWparm object.
- VEXTERNC Vrc_Codes [GEOFLOWparm_ctor2](#) ([GEOFLOWparm](#) *thee, [GEOFLOWparm_CalcType](#) type)
FORTTRAN stub to construct GEOFLOWparm object ?????????!!!!!!!
- VEXTERNC void [GEOFLOWparm_dtor](#) ([GEOFLOWparm](#) **thee)
Object destructor.
- VEXTERNC void [GEOFLOWparm_dtor2](#) ([GEOFLOWparm](#) *thee)
FORTTRAN stub for object destructor ?????????!!!!!!!
- VEXTERNC Vrc_Codes [GEOFLOWparm_check](#) ([GEOFLOWparm](#) *thee)
Consistency check for parameter values stored in object.
- VEXTERNC Vrc_Codes [GEOFLOWparm_parseToken](#) ([GEOFLOWparm](#) *thee, char tok[VMAX_BUFSIZE], Vio *sock)
Parse an MG keyword from an input file.

8.8.1 Detailed Description

Parameter which holds useful parameters for GEOFLOWeric multigrid calculations.

8.8.2 Typedef Documentation

8.8.2.1 typedef struct sGEOFLOWparm GEOFLOWparm

Parameter structure for GEOFLOW-specific variables from input files.

Author

Andrew Stevens, Kyle Monson

Note

If you add/delete/change something in this class, the member functions – especially GEOFLOWparm_copy – must be modified accordingly

8.8.3 Enumeration Type Documentation

8.8.3.1 enum eGEOFLOWparm_CalcType

Calculation type.

Enumerator

GFCT_MANUAL GEOFLOW-manual

GFCT_AUTO GEOFLOW-auto

GFCT_NONE not defined

Definition at line 77 of file [geoflowparm.h](#).

8.8.4 Function Documentation

8.8.4.1 VEXTERNC Vrc_Codes GEOFLOWparm_check (GEOFLOWparm * thee)

Consistency check for parameter values stored in object.

Author

Andrew Stevens, Kyle Monson

Parameters

<i>thee</i>	GEOFLOWparm object
-------------	--------------------

Returns

Success enumeration

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

8.8.4.2 VEXTERNC GEOFLOWparm* GEOFLOWparm_ctor (GEOFLOWparm_CalcType *type*)

Construct GEOFLOWparm object.

Author

Andrew Stevens, Kyle Monson

Parameters

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

Returns

Newly allocated and initialized GEOFLOWparm object

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

8.8.4.3 VEXTERNC Vrc_Codes GEOFLOWparm_ctor2 (GEOFLOWparm * *thee*, GEOFLOWparm_CalcType *type*)

FORTTRAN stub to construct GEOFLOWparm object ?????????!!!!!!

Author

Andrew Stevens, Kyle Monson

Parameters

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

Returns

Success enumeration

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

8.8.4.4 VEXTERNC void GEOFLOWparm_dtor (GEOFLOWparm ** *thee*)

Object destructor.

Author

Andrew Stevens, Kyle Monson

Parameters

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

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

8.8.4.5 VEXTERNC void GEOFLOWparm_dtor2 (GEOFLOWparm * *thee*)

FORTRAN stub for object destructor ??????????!!!!!!!!!!!!

Author

Andrew Stevens, Kyle Monson

Parameters

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

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

8.8.4.6 VEXTERNC Vrc_Codes GEOFLOWparm_parseToken (GEOFLOWparm * *thee*, char *tok*[VMAX_BUFSIZE], Vio * *sock*)

Parse an MG keyword from an input file.

Author

Andrew Stevens, Kyle Monson

Parameters

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

Returns

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

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

8.9 MGparm class

Parameter which holds useful parameters for generic multigrid calculations.

Files

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

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 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)
FORTTRAN stub to construct MGparm object.
- VEXTERNC void `MGparm_dtor` (`MGparm **thee`)
Object destructor.
- VEXTERNC void `MGparm_dtor2` (`MGparm *thee`)
FORTTRAN 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.

8.9.1 Detailed Description

Parameter which holds useful parameters for generic multigrid calculations.

8.9.2 Enumeration Type Documentation

8.9.2.1 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 77 of file [mgparm.h](#).

8.9.2.2 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 95 of file [mgparm.h](#).

8.9.3 Function Documentation

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

Parse an MG keyword from an input file.

Author

David Gohara

Parameters

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

Returns

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

Definition at line 577 of file [apolparm.c](#).

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

Parse an MG keyword from an input file.

Author

Nathan Baker

Parameters

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

Returns

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

Definition at line 431 of file [femparm.c](#).

8.9.3.3 VEXTERNC Vrc_Codes MGparm_check (MGparm * *thee*)

Consistency check for parameter values stored in object.

Author

Nathan Baker

Parameters

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

Returns

Success enumeration

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

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

Copy MGparm object into thee.

Author

Nathan Baker and Todd Dolinsky

Parameters

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

Definition at line 341 of file [mgparm.c](#).

8.9.3.5 VEXTERNC MGparm* MGparm_ctor (MGparm_CalcType *type*)

Construct MGparm object.

Author

Nathan Baker

Parameters

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

Returns

Newly allocated and initialized MGparm object

Definition at line 114 of file [mgparm.c](#).

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

FORTTRAN stub to construct MGparm object.

Author

Nathan Baker and Todd Dolinsky

Parameters

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

Returns

Success enumeration

Definition at line 125 of file [mgparm.c](#).

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

Object destructor.

Author

Nathan Baker

Parameters

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

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

8.9.3.8 VEXTERNC void MGparm_dtor2 (MGparm * *thee*)

FORTTRAN stub for object destructor.

Author

Nathan Baker

Parameters

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

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

8.9.3.9 VEXTERNC double MGparm_getCenterX (MGparm * *thee*)

Get center x-coordinate.

Author

Nathan Baker

Parameters

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

Returns

x-coordinate

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

8.9.3.10 VEXTERNC double MGparm_getCenterY (MGparm * *thee*)

Get center y-coordinate.

Author

Nathan Baker

Parameters

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

Returns

y-coordinate

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

8.9.3.11 VEXTERNC double MGparm_getCenterZ (MGparm * *thee*)

Get center z-coordinate.

Author

Nathan Baker

Parameters

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

Returns

z-coordinate

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

8.9.3.12 VEXTERNC double MGparm_getHx (MGparm * *thee*)

Get grid spacing in x direction (Å)

Author

Nathan Baker

Parameters

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

Returns

Grid spacing in the x direction

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

8.9.3.13 VEXTERNC double MGparm_getHy (MGparm * *thee*)

Get grid spacing in y direction (Å)

Author

Nathan Baker

Parameters

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

Returns

Grid spacing in the y direction

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

8.9.3.14 VEXTERNC double MGparm_getHz (MGparm * *thee*)

Get grid spacing in z direction (Å)

Author

Nathan Baker

Parameters

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

Returns

Grid spacing in the z direction

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

8.9.3.15 VEXTERNC int MGparm_getNx (MGparm * *thee*)

Get number of grid points in x direction.

Author

Nathan Baker

Parameters

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

Returns

Number of grid points in the x direction

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

8.9.3.16 VEXTERNC int MGparm_getNy (MGparm * *thee*)

Get number of grid points in y direction.

Author

Nathan Baker

Parameters

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

Returns

Number of grid points in the y direction

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

8.9.3.17 VEXTERNC int MGparm_getNz (MGparm * *thee*)

Get number of grid points in z direction.

Author

Nathan Baker

Parameters

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

Returns

Number of grid points in the z direction

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

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

Parse an MG keyword from an input file.

Author

Nathan Baker and Todd Dolinsky

Parameters

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

Returns

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

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

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

Set center x-coordinate.

Author

Nathan Baker

Parameters

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

Definition at line 65 of file [mgparm.c](#).

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

Set center y-coordinate.

Author

Nathan Baker

Parameters

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

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

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

Set center z-coordinate.

Author

Nathan Baker

Parameters

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

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

8.10 NOsh class

Class for parsing for fixed format input files.

Files

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

Data Structures

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

Macros

- `#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, `NCT_BEM` =3,
`NCT_GEOFLOW` =4 }
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)
FORTTRAN stub to construct NOsh.
- VEXTERNC void [NOsh_dtor](#) ([NOsh](#) **thee)
Object destructor.
- VEXTERNC void [NOsh_dtor2](#) ([NOsh](#) *thee)
FORTTRAN 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.

8.10.1 Detailed Description

Class for parsing for fixed format input files.

8.10.2 Enumeration Type Documentation

8.10.2.1 enum [eNOsh_CalcType](#)

NOsh calculation types.

Enumerator

[NCT_MG](#) Multigrid
[NCT_FEM](#) Finite element
[NCT_APOL](#) non-polar
[NCT_BEM](#) Boundary element (TABI)
[NCT_GEOFLOW](#) Geometric flow

Definition at line 115 of file [nosh.h](#).

8.10.2.2 enum eNOsh_MolFormat

Molecule file format types.

Enumerator

NMF_PQR PQR format

NMF_PDB PDB format

NMF_XML XML format

Definition at line 99 of file [nosh.h](#).

8.10.2.3 enum eNOsh_ParmFormat

Parameter file format types.

Enumerator

NPF_FLAT Flat-file format

NPF_XML XML format

Definition at line 133 of file [nosh.h](#).

8.10.2.4 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 148 of file [nosh.h](#).

8.10.3 Function Documentation

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

Return the name of an apol statement.

Author

David Gohara

Parameters

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

Returns

The name (if present) of an APOL statement

Definition at line 250 of file [nosh.c](#).

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

Copy NOsh_calc object into thee.

Author

Nathan Baker

Parameters

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

Definition at line 419 of file [nosh.c](#).

8.10.3.3 VEXTERNC NOsh_calc* NOsh_calc_ctor (NOsh_CalcType *calcType*)

Construct NOsh_calc.

Author

Nathan Baker

Parameters

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

Returns

Newly allocated and initialized NOsh object

Definition at line 342 of file [nosh.c](#).

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

Object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 382 of file [nosh.c](#).

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

Construct NOsh.

Author

Nathan Baker

Parameters

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

Returns

Newly allocated and initialized NOsh object

Definition at line 276 of file [nosh.c](#).

8.10.3.6 VEXTERNC int NOsh_ctor2 (NOsh * *thee*, int *rank*, int *size*)

FORTTRAN stub to construct NOsh.

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 287 of file [nosh.c](#).

8.10.3.7 VEXTERNC void NOsh_dtor (NOsh ** *thee*)

Object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 322 of file [nosh.c](#).

8.10.3.8 VEXTERNC void NOsh_dtor2 (NOsh * *thee*)

FORTTRAN stub for object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 330 of file [nosh.c](#).

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

Return the name of an elec statement.

Author

Todd Dolinsky

Parameters

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

Returns

The name (if present) of an ELEC statement

Definition at line 244 of file [nosh.c](#).

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

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

See also

[elec2calc](#))

Author

Nathan Baker

Parameters

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

Returns

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

See also

`elec2calc`)

Definition at line [256](#) of file [nosh.c](#).

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

Returns specified calculation object.

Author

Nathan Baker

Parameters

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

Returns

Pointer to specified calculation object

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

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

Returns format of specified charge map.

Author

Nathan Baker

Parameters

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

Returns

Format of charge map

Definition at line [223](#) of file [nosh.c](#).

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

Returns path to specified charge distribution map.

Author

Nathan Baker

Parameters

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

Returns

Path string

Definition at line 198 of file [nosh.c](#).

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

Returns format of specified dielectric map.

Author

Nathan Baker

Parameters

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

Returns

Format of dielectric map

Definition at line 208 of file [nosh.c](#).

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

Returns path to specified x-shifted dielectric map.

Author

Nathan Baker

Parameters

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

<i>imap</i>	Map ID of interest
-------------	--------------------

Returns

Path string

Definition at line 173 of file [nosh.c](#).

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

Returns path to specified y-shifted dielectric map.

Author

Nathan Baker

Parameters

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

Returns

Path string

Definition at line 178 of file [nosh.c](#).

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

Returns path to specified z-shifted dielectric map.

Author

Nathan Baker

Parameters

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

Returns

Path string

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

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

Returns format of specified kappa map.

Author

Nathan Baker

Parameters

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

Returns

Format of kappa map

Definition at line 213 of file [nosh.c](#).

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

Returns path to specified kappa map.

Author

Nathan Baker

Parameters

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

Returns

Path string

Definition at line 188 of file [nosh.c](#).

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

Returns path to specified molecule.

Author

Nathan Baker

Parameters

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

Returns

Path string

Definition at line 168 of file [nosh.c](#).

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

Returns format of specified potential map.

Author

Nathan Baker

Parameters

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

Returns

Format of potential map

Definition at line 218 of file [nosh.c](#).

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

Returns path to specified potential map.

Author

David Gohara

Parameters

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

Returns

Path string

Definition at line 193 of file [nosh.c](#).

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

Parse an input file from a socket.

Note

Should be called before NOsh_setupCalc

Author

Nathan Baker and Todd Dolinsky

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 457 of file [nosh.c](#).

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

Parse an input file only from a file.

Note

Included for SWIG wrapper compatibility
Should be called before NOsh_setupCalc

Author

Nathan Baker and Todd Dolinsky

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line [442](#) of file [nosh.c](#).

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

Return calculation ID for specified PRINT statement (.

See also

printcalc)

Author

Nathan Baker

Parameters

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

Returns

Calculation ID for specified PRINT statement (

See also

printcalc)

Definition at line [269](#) of file [nosh.c](#).

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

Return number of arguments to PRINT statement (.

See also

printrarg)

Author

Nathan Baker

Parameters

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

Returns

Number of arguments to PRINT statement (

See also

printrarg)

Definition at line [238](#) of file [nosh.c](#).

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

Return integer ID for specified operation (.

See also

printop)

Author

Nathan Baker

Parameters

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

Returns

Integer ID for specified operation (

See also

printop)

Definition at line [262](#) of file [nosh.c](#).

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

Return an integer ID of the observable to print (.

See also

printwhat)

Author

Nathan Baker

Parameters

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

Returns

An integer ID of the observable to print (

See also

printwhat)

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

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

Setup the series of non-polar calculations.

Note

Should be called after NOsh_parseInput*

Author

Nathan Baker and Todd Dolinsky

Parameters

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

Returns

1 if successful, 0 otherwise

Parameters

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

Definition at line 1371 of file [nosh.c](#).

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

Setup the series of electrostatics calculations.

Note

Should be called after NOsh_parseInput*

Author

Nathan Baker and Todd Dolinsky

Parameters

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

Returns

1 if successful, 0 otherwise

Parameters

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

Definition at line 1282 of file [nosh.c](#).

8.11 PBEparm class

Parameter structure for PBE variables independent of solver.

Files

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

Data Structures

- struct [sPBEparm](#)
Parameter structure for PBE variables from input files.

Macros

- `#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 (Å) of specified ion species.
- VEXTERNC [PBEparm](#) * [PBEparm_ctor](#) ()

Construct PBEparm object.

- VEXTERNC int [PBEparm_ctor2](#) ([PBEparm](#) *thee)

FORTTRAN stub to construct PBEparm object.

- VEXTERNC void [PBEparm_dtor](#) ([PBEparm](#) **thee)

Object destructor.

- VEXTERNC void [PBEparm_dtor2](#) ([PBEparm](#) *thee)

FORTTRAN 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.

8.11.1 Detailed Description

Parameter structure for PBE variables independent of solver.

8.11.2 Enumeration Type Documentation

8.11.2.1 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 81 of file [pbeparm.h](#).

8.11.2.2 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 97 of file [pbeparm.h](#).

8.11.3 Function Documentation

8.11.3.1 VEXTERNC int PBEparm_check (PBEparm * *thee*)

Consistency check for parameter values stored in object.

Author

Nathan Baker

Returns

1 if OK, 0 otherwise

Parameters

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

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

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

Copy PBEparm object into thee.

Author

Nathan Baker

Parameters

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

Definition at line 283 of file [pbeparm.c](#).

8.11.3.3 VEXTERNC PBEparm* PBEparm_ctor ()

Construct PBEparm object.

Author

Nathan Baker

Returns

Newly allocated and initialized PBEparm object

Definition at line 104 of file [pbeparm.c](#).

8.11.3.4 VEXTERNC int PBEparm_ctor2 (PBEparm * *thee*)

FORTTRAN stub to construct PBEparm object.

Author

Nathan Baker

Returns

1 if succesful, 0 otherwise

Parameters

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

Definition at line 115 of file [pbeparm.c](#).**8.11.3.5 VEXTERNC void PBEparm_dtor (PBEparm ** *thee*)**

Object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 173 of file [pbeparm.c](#).**8.11.3.6 VEXTERNC void PBEparm_dtor2 (PBEparm * *thee*)**

FORTRAN stub for object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 181 of file [pbeparm.c](#).**8.11.3.7 VEXTERNC double PBEparm_getIonCharge (PBEparm * *thee*, int *iion*)**

Get charge (e) of specified ion species.

Author

Nathan Baker

Returns

Charge of ion species (e)

Parameters

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

Definition at line 65 of file [pbeparm.c](#).

8.11.3.8 VEXTERNC double PBEparm_getIonConc (PBEparm * *thee*, int *iion*)

Get concentration (M) of specified ion species.

Author

Nathan Baker

Returns

Concentration of ion species (M)

Parameters

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

Definition at line 71 of file [pbeparm.c](#).

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

Get radius (A) of specified ion species.

Author

Nathan Baker

Returns

Radius of ion species (A)

Parameters

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

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

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

Parse a keyword from an input file.

Author

Nathan Baker

Returns

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

Parameters

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

Definition at line [1206](#) of file [pbeparm.c](#).

8.12 Vacc class

Solvent- and ion-accessibility oracle.

Files

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

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)
FORTTRAN stub to construct the accessibility object.
- VEXTERNC void [Vacc_dtor](#) ([Vacc](#) **thee)

- Destroy object.*
- VEXTERNC void `Vacc_dtor2` (`Vacc *thee`)
 - FORTTRAN stub to destroy object.*
- VEXTERNC double `Vacc_vdwAcc` (`Vacc *thee`, double center[`VAPBS_DIM`])
 - Report van der Waals accessibility.*
- VEXTERNC double `Vacc_ivdwAcc` (`Vacc *thee`, double center[`VAPBS_DIM`], double radius)
 - Report inflated van der Waals accessibility.*
- VEXTERNC double `Vacc_molAcc` (`Vacc *thee`, double center[`VAPBS_DIM`], double radius)
 - Report molecular accessibility.*
- VEXTERNC double `Vacc_fastMolAcc` (`Vacc *thee`, double center[`VAPBS_DIM`], double radius)
 - Report molecular accessibility quickly.*
- VEXTERNC double `Vacc_splineAcc` (`Vacc *thee`, double center[`VAPBS_DIM`], double win, double infrad)
 - Report spline-based accessibility.*
- VEXTERNC void `Vacc_splineAccGrad` (`Vacc *thee`, double center[`VAPBS_DIM`], double win, double infrad, double *grad)
 - Report gradient of spline-based accessibility.*
- VEXTERNC double `Vacc_splineAccAtom` (`Vacc *thee`, double center[`VAPBS_DIM`], double win, double infrad, `Vatom *atom`)
 - Report spline-based accessibility for a given atom.*
- VEXTERNC void `Vacc_splineAccGradAtomUnnorm` (`Vacc *thee`, double center[`VAPBS_DIM`], double win, double infrad, `Vatom *atom`, double *force)
 - Report gradient of spline-based accessibility with respect to a particular atom (see `Vpmg_splineAccAtom`)*
- VEXTERNC void `Vacc_splineAccGradAtomNorm` (`Vacc *thee`, double center[`VAPBS_DIM`], double win, double infrad, `Vatom *atom`, double *force)
 - Report gradient of spline-based accessibility with respect to a particular atom normalized by the accessibility value due to that atom at that point (see `Vpmg_splineAccAtom`)*
- VEXTERNC void `Vacc_splineAccGradAtomNorm4` (`Vacc *thee`, double center[`VAPBS_DIM`], double win, double infrad, `Vatom *atom`, double *force)
 - Report gradient of spline-based accessibility with respect to a particular atom normalized by a 4th order accessibility value due to that atom at that point (see `Vpmg_splineAccAtom`)*
- VEXTERNC void `Vacc_splineAccGradAtomNorm3` (`Vacc *thee`, double center[`VAPBS_DIM`], double win, double infrad, `Vatom *atom`, double *force)
 - Report gradient of spline-based accessibility with respect to a particular atom normalized by a 3rd order accessibility value due to that atom at that point (see `Vpmg_splineAccAtom`)*
- VEXTERNC double `Vacc_SASA` (`Vacc *thee`, double radius)
 - Build the solvent accessible surface (SAS) and calculate the solvent accessible surface area.*
- VEXTERNC double `Vacc_totalSASA` (`Vacc *thee`, double radius)
 - Return the total solvent accessible surface area (SASA)*
- VEXTERNC double `Vacc_atomSASA` (`Vacc *thee`, double radius, `Vatom *atom`)
 - Return the atomic solvent accessible surface area (SASA)*
- VEXTERNC `VaccSurf * Vacc_atomSASPoints` (`Vacc *thee`, double radius, `Vatom *atom`)
 - Get the set of points for this atom's solvent-accessible surface.*
- VEXTERNC void `Vacc_atomdSAV` (`Vacc *thee`, double radius, `Vatom *atom`, double *dSA)
 - Get the derivative of solvent accessible volume.*
- VEXTERNC void `Vacc_atomdSASA` (`Vacc *thee`, double dpos, double radius, `Vatom *atom`, double *dSA)
 - Get the derivative of solvent accessible area.*
- VEXTERNC void `Vacc_totalAtomdSASA` (`Vacc *thee`, double dpos, double radius, `Vatom *atom`, double *dSA)
 - Testing purposes only.*

- VEXTERNC void [Vacc_totalAtomdSAV](#) ([Vacc](#) *thee, double dpos, double radius, [Vatom](#) *atom, double *dSA, [Vclist](#) *clist)
Total solvent accessible volume.
- VEXTERNC double [Vacc_totalSAV](#) ([Vacc](#) *thee, [Vclist](#) *clist, [APOLparm](#) *apolparm, double radius)
Return the total solvent accessible volume (SAV)
- VEXTERNC int [Vacc_wcaEnergy](#) ([Vacc](#) *thee, [APOLparm](#) *apolparm, [Valist](#) *alist, [Vclist](#) *clist)
Return the WCA integral energy.
- VEXTERNC int [Vacc_wcaForceAtom](#) ([Vacc](#) *thee, [APOLparm](#) *apolparm, [Vclist](#) *clist, [Vatom](#) *atom, double *force)
Return the WCA integral force.
- VEXTERNC int [Vacc_wcaEnergyAtom](#) ([Vacc](#) *thee, [APOLparm](#) *apolparm, [Valist](#) *alist, [Vclist](#) *clist, int iatom, double *value)
Calculate the WCA energy for an atom.

8.12.1 Detailed Description

Solvent- and ion-accessibility oracle.

8.12.2 Function Documentation

8.12.2.1 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

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

Definition at line [1332](#) of file [vacc.c](#).

8.12.2.2 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

<i>thee</i>	Accessibility object
<i>radius</i>	Probe radius (Å)
<i>atom</i>	Atom of interest
<i>dSA</i>	Array holding answers of calc

Definition at line 1212 of file [vacc.c](#).

8.12.2.3 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 (Å²)

Parameters

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

Definition at line 780 of file [vacc.c](#).

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

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

Author

Nathan Baker

Returns

Pointer to VaccSurf object for this atom

Parameters

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

<i>atom</i>	Atom of interest
-------------	------------------

Definition at line 994 of file [vacc.c](#).

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

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

Author

Nathan Baker

Returns

Atom sphere surface object

Parameters

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

See also

[VaccSurf_refSphere](#)

Parameters

<i>probe_radius</i>	Probe radius (in Å)
---------------------	---------------------

Definition at line 873 of file [vacc.c](#).

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

Construct the accessibility object.

Author

Nathan Baker

Returns

Newly allocated Vacc object

Parameters

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

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

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

FORTTRAN stub to construct the accessibility object.

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Parameters

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

Definition at line 213 of file [vacc.c](#).

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

Destroy object.

Author

Nathan Baker

Parameters

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

Definition at line 245 of file [vacc.c](#).

8.12.2.9 VEXTERNC void Vacc_dtor2 (Vacc * *thee*)

FORTTRAN stub to destroy object.

Author

Nathan Baker

Parameters

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

Definition at line 255 of file [vacc.c](#).

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

Report molecular accessibility quickly.

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

Note

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

Author

Nathan Baker

Returns

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

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

Parameters

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

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

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

Report inflated van der Waals accessibility.

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

Author

Nathan Baker

Returns

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

Parameters

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

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

Get number of bytes in this object and its members.

Author

Nathan Baker

Returns

Number of bytes allocated for object

Parameters

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

Definition at line 63 of file [vacc.c](#).

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

Report molecular accessibility.

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

Author

Nathan Baker

Returns

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

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

Parameters

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

Definition at line 608 of file [vacc.c](#).

8.12.2.14 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 (Å²)

Parameters

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

Definition at line 713 of file [vacc.c](#).

8.12.2.15 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

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

Definition at line 528 of file [vacc.c](#).

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

Report spline-based accessibility for a given atom.

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

Author

Nathan Baker

Returns

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

Parameters

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

Definition at line 438 of file [vacc.c](#).

8.12.2.17 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

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

Definition at line 561 of file [vacc.c](#).

8.12.2.18 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 evaluation; basically a cubic spline.

Author

Nathan Baker

Parameters

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

Definition at line 316 of file [vacc.c](#).

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

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

Author

Michael Schnieders

Parameters

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

Definition at line 1111 of file [vacc.c](#).

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

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

Author

Michael Schnieders

Parameters

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

Definition at line 1018 of file [vacc.c](#).

8.12.2.21 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

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

Definition at line 377 of file [vacc.c](#).

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

Testing purposes only.

Author

David Gohara, Nathan Baker

Parameters

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

Definition at line 1401 of file [vacc.c](#).

8.12.2.23 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

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

Definition at line 1460 of file [vacc.c](#).

8.12.2.24 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 (Å²)

Parameters

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

Definition at line 774 of file [vacc.c](#).

8.12.2.25 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 (Å³)

Parameters

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

Definition at line 1515 of file [vacc.c](#).

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

Report van der Waals accessibility.

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

Author

Nathan Baker

Returns

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

Parameters

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

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

Return the WCA integral energy.

Author

David Gohara

Returns

Success flag

Parameters

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

Definition at line 1733 of file [vacc.c](#).

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

Calculate the WCA energy for an atom.

Author

Dave Gohara and Nathan Baker

Returns

Success flag

Parameters

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

Definition at line 1592 of file [vacc.c](#).

8.12.2.29 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/Å)

Parameters

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

Definition at line 1768 of file [vacc.c](#).

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

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

Author

Nathan Baker

Returns

Newly allocated and constructed surface object

Parameters

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

Definition at line 803 of file [vacc.c](#).

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

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

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Parameters

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

Definition at line 818 of file [vacc.c](#).

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

Destroy the surface object and free its memory.

Author

Nathan Baker

Parameters

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

Definition at line 844 of file [vacc.c](#).

8.12.2.33 VEXTERNC void VaccSurf_dtor2 (VaccSurf * *thee*)

Destroy the surface object.

Author

Nathan Baker

Parameters

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

Definition at line 858 of file [vacc.c](#).

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

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

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

Note

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

Author

Nathan Baker (original FORTRAN code by Mike Gilson)

Returns

Reference sphere surface object

Parameters

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

Definition at line 938 of file [vacc.c](#).

8.13 Valist class

Container class for list of atom objects.

Files

- file [valist.h](#)
Contains declarations for class Valist.

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)
FORTTRAN stub to construct the atom list object.
- VEXTERNC void [Valist_dtor](#) ([Valist](#) **thee)
Destroys atom list object.
- VEXTERNC void [Valist_dtor2](#) ([Valist](#) *thee)
FORTTRAN 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.

8.13.1 Detailed Description

Container class for list of atom objects.

8.13.2 Function Documentation

8.13.2.1 VEXTERNC Valist* Valist_ctor ()

Construct the atom list object.

Author

Nathan Baker

Returns

Pointer to newly allocated (empty) atom list

Definition at line [138](#) of file [valist.c](#).

8.13.2.2 VEXTERNC Vrc_Codes Valist_ctor2 (Valist * thee)

FORTTRAN stub to construct the atom list object.

Author

Nathan Baker, Yong Huang

Returns

Success enumeration

Parameters

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

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

8.13.2.3 VEXTERNC void Valist_dtor (Valist ** thee)

Destroys atom list object.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to storage for atom list
-------------	----------------------------------

Definition at line 167 of file [valist.c](#).

8.13.2.4 VEXTERNC void Valist_dtor2 (Valist * *thee*)

FORTTRAN stub to destroy atom list object.

Author

Nathan Baker

Parameters

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

Definition at line 176 of file [valist.c](#).

8.13.2.5 VEXTERNC Vatom* Valist_getAtom (Valist * *thee*, int *i*)

Get pointer to particular atom in list.

Author

Nathan Baker

Returns

Pointer to atom object *i*

Parameters

<i>thee</i>	Atom list object
<i>i</i>	Index of atom in list

Definition at line 115 of file [valist.c](#).

8.13.2.6 VEXTERNC Vatom* Valist_getAtomList (Valist * *thee*)

Get actual array of atom objects from the list.

Author

Nathan Baker

Returns

Array of atom objects

Parameters

<i>thee</i>	Atom list object
-------------	------------------

Definition at line 95 of file [valist.c](#).

8.13.2.7 VEXTERNC double Valist_getCenterX (Valist * *thee*)

Get x-coordinate of molecule center.

Author

Nathan Baker

Returns

X-coordinate of molecule center

Parameters

<i>thee</i>	Atom list object
-------------	------------------

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

8.13.2.8 VEXTERNC double Valist_getCenterY (Valist * *thee*)

Get y-coordinate of molecule center.

Author

Nathan Baker

Returns

Y-coordinate of molecule center

Parameters

<i>thee</i>	Atom list object
-------------	------------------

Definition at line 76 of file [valist.c](#).

8.13.2.9 VEXTERNC double Valist_getCenterZ (Valist * *thee*)

Get z-coordinate of molecule center.

Author

Nathan Baker

Returns

Z-coordinate of molecule center

Parameters

<i>thee</i>	Atom list object
-------------	------------------

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

8.13.2.10 VEXTERNC int Valist_getNumberAtoms (Valist * *thee*)

Get number of atoms in the list.

Author

Nathan Baker

Returns

Number of atoms in list

Parameters

<i>thee</i>	Atom list object
-------------	------------------

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

8.13.2.11 VEXTERNC Vrc_Codes Valist_getStatistics (Valist * *thee*)

Load up Valist with various statistics.

Author

Nathan Baker, Yong Huang

Returns

Success enumeration

Definition at line 869 of file [valist.c](#).

8.13.2.12 VEXTERNC unsigned long int Valist_memChk (Valist * *thee*)

Get total memory allocated for this object and its members.

Author

Nathan Baker

Returns

Total memory in bytes

Parameters

<i>thee</i>	Atom list object
-------------	------------------

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

8.13.2.13 VEXTERNC Vrc_Codes Valist_readPDB (Valist * *thee*, Vparam * *param*, Vio * *sock*)

Fill atom list with information from a PDB file.

Author

Nathan Baker, Todd Dolinsky, Yong Huang

Returns

Success enumeration

Note

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

Parameters

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

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

8.13.2.14 VEXTERNC Vrc_Codes Valist_readPQR (Valist * *thee*, Vparam * *param*, Vio * *sock*)

Fill atom list with information from a PQR file.

Author

Nathan Baker, Yong Huang

Returns

Success enumeration

Note

- A PQR file has PDB structure with charge and radius in the last two columns instead of weight and occupancy
- We don't actually respect PDB format; instead recognize whitespace- or tab-delimited fields which allows us to deal with structures with coordinates > 999 or < -999.

Parameters

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

Definition at line 606 of file [valist.c](#).

8.13.2.15 VEXTERNC Vrc_Codes Valist_readXML (Valist * *thee*, Vparam * *param*, Vio * *sock*)

Fill atom list with information from an XML file.

Author

Todd Dolinsky, Yong Huang

Returns

Success enumeration

Note

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

Parameters

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

Definition at line 725 of file [valist.c](#).

8.14 Vatom class

Atom class for interfacing APBS with PDB files.

Files

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

Data Structures

- struct [sVatom](#)
Contains public data members for Vatom class/module.

Macros

- `#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.

8.14.1 Detailed Description

Atom class for interfacing APBS with PDB files.

8.14.2 Macro Definition Documentation

8.14.2.1 #define VMAX_RECLEN 64

Residue name length.

Author

Nathan Baker, David Gohara, Mike Schneiders

Definition at line 77 of file [vatom.h](#).

8.14.3 Function Documentation

8.14.3.1 VEXTERNC void Vatom_copyFrom (Vatom * *thee*, Vatom * *src*)

Copy information to another atom.

Author

Nathan Baker

Parameters

<i>thee</i>	Destination for atom information
<i>src</i>	Source for atom information

Definition at line 186 of file [vatom.c](#).

8.14.3.2 VEXTERNC void Vatom_copyTo (Vatom * *thee*, Vatom * *dest*)

Copy information to another atom.

Author

Nathan Baker

Parameters

<i>thee</i>	Source for atom information
<i>dest</i>	Destination for atom information

Definition at line 177 of file [vatom.c](#).

8.14.3.3 VEXTERNC Vatom* Vatom_ctor ()

Constructor for the Vatom class.

Author

Nathan Baker

Returns

Pointer to newly allocated Vatom object

Definition at line 142 of file [vatom.c](#).

8.14.3.4 VEXTERNC int Vatom_ctor2 (Vatom * *thee*)

FORTTRAN stub constructor for the Vatom class.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to Vatom allocated memory location
-------------	--

Returns

1 if succesful, 0 otherwise

Definition at line 153 of file [vatom.c](#).

8.14.3.5 VEXTERNC void Vatom_dtor (Vatom ** *thee*)

Object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 158 of file [vatom.c](#).

8.14.3.6 VEXTERNC void Vatom_dtor2 (Vatom * *thee*)

FORTTRAN stub object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 166 of file [vatom.c](#).

8.14.3.7 VEXTERNC double Vatom_getAtomID (Vatom * *thee*)

Get atom ID.

Author

Nathan Baker

Parameters

<i>thee</i>	Vatom object
-------------	--------------

Returns

Unique non-negative number

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

8.14.3.8 VEXTERNC void Vatom_getAtomName (Vatom * *thee*, char *atomName*[VMAX_RECLEN])

Retrieve atom name.

Author

Jason Wagoner

Parameters

<i>thee</i>	Vatom object
<i>atomName</i>	Atom name

Definition at line [214](#) of file [vatom.c](#).

8.14.3.9 VEXTERNC double Vatom_getCharge (Vatom * *thee*)

Get atomic charge.

Author

Nathan Baker

Parameters

<i>thee</i>	Vatom object
-------------	--------------

Returns

Atom partial charge (in e)

Definition at line [119](#) of file [vatom.c](#).

8.14.3.10 VEXTERNC double Vatom_getEpsilon (Vatom * *thee*)

Get atomic epsilon.

Author

David Gohara

Parameters

<i>thee</i>	Vatom object
-------------	--------------

Returns

Atomic epsilon (in Å)

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

8.14.3.11 VEXTERNC double Vatom_getPartID (Vatom * *thee*)

Get partition ID.

Author

Nathan Baker

Parameters

<i>thee</i>	Vatom object
-------------	--------------

Returns

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

Definition at line 70 of file [vatom.c](#).

8.14.3.12 VEXTERNC double* Vatom_getPosition (Vatom * *thee*)

Get atomic position.

Author

Nathan Baker

Parameters

<i>thee</i>	Vatom object
-------------	--------------

Returns

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

Definition at line 63 of file [vatom.c](#).

8.14.3.13 VEXTERNC double Vatom_getRadius (Vatom * *thee*)

Get atomic position.

Author

Nathan Baker

Parameters

<i>thee</i>	Vatom object
-------------	--------------

Returns

Atomic radius (in Å)

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

8.14.3.14 VEXTERNC void Vatom_getResName (Vatom * *thee*, char *resName*[VMAX_RECLEN])

Retrieve residue name.

Author

Jason Wagoner

Parameters

<i>thee</i>	Vatom object
<i>resName</i>	Residue Name

Definition at line 199 of file [vatom.c](#).

8.14.3.15 VEXTERNC unsigned long int Vatom_memChk (Vatom * *thee*)

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

Author

Nathan Baker

Parameters

<i>thee</i>	Vpmg object
-------------	-------------

Returns

The memory used by this structure and its contents in bytes

Definition at line 138 of file [vatom.c](#).

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

Set atom ID.

Author

Nathan Baker

Parameters

<i>thee</i>	Vatom object
<i>id</i>	Unique non-negative number

Definition at line 91 of file [vatom.c](#).

8.14.3.17 VEXTERNC void Vatom_setAtomName (Vatom * *thee*, char *atomName*[VMAX_RECLEN])

Set atom name.

Author

Jason Wagoner

Parameters

<i>thee</i>	Vatom object
<i>atomName</i>	Atom name

Definition at line 207 of file [vatom.c](#).

8.14.3.18 VEXTERNC void Vatom_setCharge (Vatom * *thee*, double *charge*)

Set atomic charge.

Author

Nathan Baker

Parameters

<i>thee</i>	Vatom object
<i>charge</i>	Atom partial charge (in e)

Definition at line 112 of file [vatom.c](#).

8.14.3.19 VEXTERNC void Vatom_setEpsilon (Vatom * *thee*, double *epsilon*)

Set atomic epsilon.

Author

David Gohara

Parameters

<i>thee</i>	Vatom object
<i>epsilon</i>	Atomic epsilon (in Å)

Definition at line 126 of file [vatom.c](#).

8.14.3.20 VEXTERNC void Vatom_setPartID (Vatom * *thee*, int *partID*)

Set partition ID.

Author

Nathan Baker

Parameters

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

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

8.14.3.21 VEXTERNC void Vatom_setPosition (Vatom * *thee*, double *position*[3])

Set the atomic position.

Author

Nathan Baker

Parameters

<i>thee</i>	Vatom object to be modified
<i>position</i>	Coordinates (in Å)

Definition at line 168 of file [vatom.c](#).8.14.3.22 VEXTERN void Vatom_setRadius (Vatom * *thee*, double *radius*)

Set atomic radius.

Author

Nathan Baker

Parameters

<i>thee</i>	Vatom object
<i>radius</i>	Atomic radius (in Å)

Definition at line 98 of file [vatom.c](#).8.14.3.23 VEXTERN void Vatom_setResName (Vatom * *thee*, char *resName*[VMAX_RECLEN])

Set residue name.

Author

Jason Wagoner

Parameters

<i>thee</i>	Vatom object
<i>resName</i>	Residue Name

Definition at line 192 of file [vatom.c](#).

8.15 Vcap class

Collection of routines which cap certain exponential and hyperbolic functions.

Files

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

Macros

- `#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.

8.15.1 Detailed Description

Collection of routines which cap certain exponential and hyperbolic functions.

Note

These routines are based on FORTRAN code by Mike Holst

8.15.2 Function Documentation

8.15.2.1 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()
<i>ichop</i>	Set to 1 if function capped, 0 otherwise

Definition at line 91 of file [vcap.c](#).

8.15.2.2 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()
<i>ichop</i>	Set to 1 if function capped, 0 otherwise

Definition at line 59 of file [vcap.c](#).

8.15.2.3 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

<i>x</i>	Argument to sinh()
<i>ichop</i>	Set to 1 if function capped, 0 otherwise

Definition at line 75 of file [vcap.c](#).

8.16 Vclist class

Atom cell list.

Files

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

Data Structures

- struct [sVclistCell](#)
Atom cell list cell.
- struct [sVclist](#)
Atom cell list.

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 Å) 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](#)])
FORTTRAN stub to construct the cell list object.
- VEXTERNC void [Vclist_dtor](#) ([Vclist](#) **thee)
Destroy object.

- VEXTERNC void [Vclist_dtor2](#) ([Vclist](#) *thee)
FORTTRAN 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)
FORTTRAN stub to destroy object.

8.16.1 Detailed Description

Atom cell list.

8.16.2 Enumeration Type Documentation

8.16.2.1 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 [82](#) of file [vclist.h](#).

8.16.3 Function Documentation

- 8.16.3.1 VEXTERNC [Vclist*](#) [Vclist_ctor](#) ([Valist](#) * *alist*, double *max_radius*, int *npts*[[VAPBS_DIM](#)], [Vclist_DomainMode](#) *mode*, double *lower_corner*[[VAPBS_DIM](#)], double *upper_corner*[[VAPBS_DIM](#)])

Construct the cell list object.

Author

Nathan Baker

Returns

Newly allocated [Vclist](#) object

Parameters

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

Definition at line 75 of file [vclist.c](#).

8.16.3.2 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])

FORTTRAN stub to construct the cell list object.

Author

Nathan Baker, Yong Huang

Returns

Success enumeration

Parameters

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

Definition at line 343 of file [vclist.c](#).

8.16.3.3 VEXTERNC void Vclist_dtor (Vclist ** *thee*)

Destroy object.

Author

Nathan Baker

Parameters

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

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

8.16.3.4 VEXTERNC void Vclist_dtor2 (Vclist * *thee*)

FORTTRAN stub to destroy object.

Author

Nathan Baker

Parameters

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

Definition at line 408 of file [vclist.c](#).**8.16.3.5 VEXTERNC VclistCell* Vclist_getCell (Vclist * *thee*, double *position*[VAPBS_DIM])**

Return cell corresponding to specified position or return VNULL.

Author

Nathan Baker

Returns

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

Parameters

<i>thee</i>	Pointer to Vclist cell list
<i>position</i>	Position to evaluate

Definition at line 423 of file [vclist.c](#).**8.16.3.6 VEXTERNC double Vclist_maxRadius (Vclist * *thee*)**

Get the max probe radius value (in A) the cell list was constructed with.

Author

Nathan Baker

Returns

Max probe radius (in A)

Parameters

<i>thee</i>	Cell list object
-------------	------------------

Definition at line 68 of file [vclist.c](#).**8.16.3.7 VEXTERNC unsigned long int Vclist_memChk (Vclist * *thee*)**

Get number of bytes in this object and its members.

Author

Nathan Baker

Returns

Number of bytes allocated for object

Parameters

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

Definition at line 63 of file [vclist.c](#).

8.16.3.8 VEXTERNC VclistCell* VclistCell_ctor (int *natoms*)

Allocate and construct a cell list cell object.

Author

Nathan Baker

Returns

Pointer to newly-allocated and constructed object.

Parameters

<i>natoms</i>	Number of atoms associated with this cell
---------------	---

Definition at line 452 of file [vclist.c](#).

8.16.3.9 VEXTERNC Vrc_Codes VclistCell_ctor2 (VclistCell * *thee*, int *natoms*)

Construct a cell list object.

Author

Nathan Baker, Yong Huang

Returns

Success enumeration

Parameters

<i>thee</i>	Memory location for object
<i>natoms</i>	Number of atoms associated with this cell

Definition at line 464 of file [vclist.c](#).

8.16.3.10 VEXTERNC void VclistCell_dtor (VclistCell ** *thee*)

Destroy object.

Author

Nathan Baker

Parameters

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

Definition at line [486](#) of file [vclist.c](#).

8.16.3.11 VEXTERNC void VclistCell_dtor2 (VclistCell * *thee*)

FORTTRAN stub to destroy object.

Author

Nathan Baker

Parameters

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

Definition at line [497](#) of file [vclist.c](#).

8.17 Vgreen class

Provides capabilities for pointwise evaluation of free space Green's function for point charges in a uniform dielectric.

Files

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

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)
FORTTRAN 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)
FORTTRAN 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)

8.17.1 Detailed Description

Provides capabilities for pointwise evaluation of free space Green's function for point charges in a uniform dielectric.

Note

Right now, these are very slow methods without any fast multipole acceleration.

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
```

```
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

8.17.2 Function Documentation

8.17.2.1 VEXTERNC int Vgreen_coulomb (Vgreen * *thee*, int *npos*, double * *x*, double * *y*, double * *z*, double * *val*)

Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation or H. E. Johnston, R. Krasny FMM library (if available)

Returns the potential ϕ defined by

$$\phi(r) = \sum_i \frac{q_i}{r_i}$$

where q_i is the atomic charge (in e) and r_i is the distance to the observation point r . The potential is scaled to units of V.

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 258 of file [vgreen.c](#).

8.17.2.2 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

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

Returns

1 if successful, 0 otherwise

Definition at line 224 of file [vgreen.c](#).

8.17.2.3 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

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

Returns

1 if successful, 0 otherwise

Definition at line 362 of file [vgreen.c](#).

8.17.2.4 VEXTERNC int Vgreen_coulombD_direct (Vgreen * *thee*, int *npos*, double * *x*, double * *y*, double * *z*, double * *pot*, double * *gradx*, double * *grady*, double * *gradz*)

Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.

Returns the field $\nabla\phi$ defined by

$$\nabla\phi(r) = \sum_i \frac{q_i}{r_i}$$

where q_i is the atomic charge (in e) and r_i is the distance to the observation point r . The field is scaled to units of V/Å.

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 310 of file [vgreen.c](#).

8.17.2.5 VEXTERNC Vgreen* Vgreen_ctor (Valist * alist)

Construct the Green's function oracle.

Author

Nathan Baker

Parameters

<i>alist</i>	Atom (charge) list associated with object
--------------	---

Returns

Pointer to newly allocated Green's function oracle

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

8.17.2.6 VEXTERNC int Vgreen_ctor2 (Vgreen * thee, Valist * alist)

FORTTRAN stub to construct the Green's function oracle.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to memory allocated for object
<i>alist</i>	Atom (charge) list associated with object

Returns

1 if successful, 0 otherwise

Definition at line 167 of file [vgreen.c](#).

8.17.2.7 VEXTERNC void Vgreen_dtor (Vgreen ** *thee*)

Destruct the Green's function oracle.

Author

Nathan Baker

Parameters

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

Definition at line 192 of file [vgreen.c](#).

8.17.2.8 VEXTERNC void Vgreen_dtor2 (Vgreen * *thee*)

FORTTRAN stub to destruct the Green's function oracle.

Author

Nathan Baker

Parameters

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

Definition at line 200 of file [vgreen.c](#).

8.17.2.9 VEXTERNC Valist* Vgreen_getValist (Vgreen * *thee*)

Get the atom list associated with this Green's function object.

Author

Nathan Baker

Parameters

<i>thee</i>	Vgreen object
-------------	---------------

Returns

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

Definition at line 142 of file [vgreen.c](#).

8.17.2.10 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 \AA^{-1} ;

q_i is the atomic charge (in e), and r_i is the distance from atom i to the observation point r . The potential is scaled to units of V.

Author

Nathan Baker

Bug Not implemented yet

Note

Not implemented yet

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 209 of file [vgreen.c](#).

8.17.2.11 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 \AA^{-1}).

q_i is the atomic charge (in e), and r_i is the distance from atom i to the observation point r . The potential is scaled to units of V/ \AA .

Author

Nathan Baker

Bug Not implemented yet

Note

Not implemented yet

Parameters

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

Returns

int 1 if successful, 0 otherwise

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

8.17.2.12 VEXTERNC unsigned long int Vgreen_memChk (Vgreen * *thee*)

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

Author

Nathan Baker

Parameters

<i>thee</i>	Vgreen object
-------------	---------------

Returns

The memory used by this structure and its contents in bytes

Definition at line 149 of file [vgreen.c](#).

8.18 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.

Macros

- `#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-zero (etc.)
- #define `SINH_MIN` -85.0

Used to set the min values acceptable for sinh chopping.
- #define `SINH_MAX` 85.0

Used to set the max values acceptable for sinh chopping.
- #define `VF77_MANGLE`(name, NAME) name

Name-mangling macro for using FORTRAN functions in C code.
- #define `VFLOOR`(value) floor(value)

Wrapped floor to fix floating point issues in the Intel compiler.
- #define `VEMBED`(rctag)

Allows embedding of RCS ID tags in object files.

Typedefs

- typedef enum `eVhal_PBEType` `Vhal_PBEType`

Declaration of the Vhal_PBEType type as the Vhal_PBEType enum.
- typedef enum `eVhal_IPKEYType` `Vhal_IPKEYType`

Declaration of the Vhal_IPKEYType type as the Vhal_IPKEYType enum.
- typedef enum `eVhal_NONLINType` `Vhal_NONLINType`

Declaration of the Vhal_NONLINType type as the Vhal_NONLINType enum.
- typedef enum `eVoutput_Format` `Voutput_Format`

Declaration of the Voutput_Format type as the VOutput_Format enum.
- typedef enum `eVbcfl` `Vbcfl`

Declare Vbcfl type.
- typedef enum `eVsurf_Meth` `Vsurf_Meth`

Declaration of the Vsurf_Meth type as the Vsurf_Meth enum.
- typedef enum `eVchrg_Meth` `Vchrg_Meth`

Declaration of the Vchrg_Meth type as the Vchrg_Meth enum.
- typedef enum `eVchrg_Src` `Vchrg_Src`

Declaration of the Vchrg_Src type as the Vchrg_Meth enum.
- typedef enum `eVdata_Type` `Vdata_Type`

Declaration of the Vdata_Type type as the Vdata_Type enum.
- typedef enum `eVdata_Format` `Vdata_Format`

Declaration of the Vdata_Format type as the Vdata_Format enum.

Enumerations

- enum `eVrc_Codes` { `VRC_WARNING` = -1, `VRC_FAILURE` = 0, `VRC_SUCCESS` = 1 }
Return code enumerations.
- enum `eVsol_Meth` {
`VSOL_CGMG`, `VSOL_Newton`, `VSOL_MG`, `VSOL_CG`,
`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.

8.18.1 Detailed Description

A "class" which consists solely of macro definitions which are used by several other classes.

8.18.2 Macro Definition Documentation

8.18.2.1 `#define MAX_SPHERE_PTS 50000`

Maximum number of points on a sphere.

Note

Used by VaccSurf

Definition at line [415](#) of file [vhal.h](#).

8.18.2.2 `#define VAPBS_BACK 4`

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line [439](#) of file [vhal.h](#).

8.18.2.3 `#define VAPBS_DOWN 5`

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line [445](#) of file [vhal.h](#).

8.18.2.4 `#define VAPBS_FRONT 1`

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line [421](#) of file [vhal.h](#).

8.18.2.5 `#define VAPBS_LEFT 3`

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line [433](#) of file [vhal.h](#).

8.18.2.6 `#define VAPBS_RIGHT 0`

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line 409 of file [vhal.h](#).

8.18.2.7 `#define VAPBS_UP 2`

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line 427 of file [vhal.h](#).

8.18.2.8 `#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 563 of file [vhal.h](#).

8.18.2.9 `#define VFLOOR(value) floor(value)`

Wrapped floor to fix floating point issues in the Intel compiler.

Author

Todd Dolinsky

Definition at line 554 of file [vhal.h](#).

8.18.3 Enumeration Type Documentation

8.18.3.1 `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 209 of file [vhal.h](#).

8.18.3.2 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 232 of file [vhal.h](#).

8.18.3.3 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 253 of file [vhal.h](#).

8.18.3.4 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 311 of file [vhal.h](#).

8.18.3.5 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/infated van der Waals (1 = accessible, 0 = inaccessible)
VDT_LAP Laplacian of potential (kT/e/A²)
VDT_EDENS Energy density $\epsilon(\nabla u)^2$, where u is potential (kT/e/A)²
VDT_NDENS Ion number density $\sum c_i \exp(-q_i u)^2$, where u is potential (output in M)
VDT_QDENS Ion charge density $\sum q_i c_i \exp(-q_i u)^2$, where u is potential (output in $e_c M$)
VDT_DIELX Dielectric x-shifted map as calculated with the currently specified scheme (dimensionless)
VDT_DIELY Dielectric y-shifted map as calculated with the currently specified scheme (dimensionless)
VDT_DIELZ Dielectric 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 271 of file [vhal.h](#).

8.18.3.6 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 159 of file [vhal.h](#).

8.18.3.7 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 141 of file [vhal.h](#).

8.18.3.8 enum eVoutput_Format

Output file format.

Enumerator

OUTPUT_NULL No output

OUTPUT_FLAT Output in flat-file format

Definition at line 193 of file [vhal.h](#).

8.18.3.9 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 68 of file [vhal.h](#).

8.18.3.10 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 83 of file [vhal.h](#).

8.18.3.11 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 parameters.

VSM_SPLINE4 A 7th order polynomial spline is used to create a smoothly varying characteristic function (continuity through 3rd derivatives) for surface based parameters.

Definition at line 104 of file [vhal.h](#).

8.19 Matrix wrapper class

A header for including data wrapping matrices.

Files

- file [vmatrix.h](#)

Contains inclusions for matrix data wrappers.

8.19.1 Detailed Description

A header for including data wrapping matrices.

8.20 Vparam class

Reads and assigns charge/radii parameters.

Files

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

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

- 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.
- 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)
FORTTRAN stub to construct the object.
- VEXTERNC void [Vparam_AtomData_dtor](#) ([Vparam_AtomData](#) **thee)
Destroy object.
- VEXTERNC void [Vparam_AtomData_dtor2](#) ([Vparam_AtomData](#) *thee)
FORTTRAN 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`)
- FORTTRAN stub to construct the object.*

 - VEXTERNC void `Vparam_ResData_dtor` (`Vparam_ResData **thee`)
- Destroy object.*

 - VEXTERNC void `Vparam_ResData_dtor2` (`Vparam_ResData *thee`)
- FORTTRAN stub to destroy object.*

 - VEXTERNC `Vparam * Vparam_ctor` ()
- Construct the object.*

 - VEXTERNC int `Vparam_ctor2` (`Vparam *thee`)
- FORTTRAN stub to construct the object.*

 - VEXTERNC void `Vparam_dtor` (`Vparam **thee`)
- Destroy object.*

 - VEXTERNC void `Vparam_dtor2` (`Vparam *thee`)
- FORTTRAN 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.*

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.

8.20.1 Detailed Description

Reads and assigns charge/radii parameters.

8.20.2 Function Documentation

8.20.2.1 VPRIVATE int readFlatFileLine (Vio * *sock*, Vparam_AtomData * *atom*)

Read a single line of the flat file database.

Author

Nathan Baker

Parameters

<i>sock</i>	Socket ready for reading
<i>atom</i>	Atom to hold parsed data

Returns

1 if successful, 0 otherwise

Definition at line 691 of file [vparam.c](#).

8.20.2.2 VPRIVATE int readXMLFileAtom (Vio * *sock*, Vparam_AtomData * *atom*)

Read atom information from an XML file.

Author

Todd Dolinsky

Parameters

<i>sock</i>	Socket ready for reading
<i>atom</i>	Atom to hold parsed data

Returns

1 if successful, 0 otherwise

Definition at line 610 of file [vparam.c](#).

8.20.2.3 VEXTERNC void Vparam_AtomData_copyFrom (Vparam_AtomData * *thee*, Vparam_AtomData * *src*)

Copy current atom object from another.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to destination object
<i>src</i>	Pointer to source object

Definition at line 607 of file [vparam.c](#).

8.20.2.4 VEXTERNC void Vparam_AtomData_copyTo (Vparam_AtomData * *thee*, Vparam_AtomData * *dest*)

Copy current atom object to destination.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to source object
<i>dest</i>	Pointer to destination object

Definition at line 571 of file [vparam.c](#).

8.20.2.5 VEXTERNC Vparam_AtomData* Vparam_AtomData_ctor ()

Construct the object.

Author

Nathan Baker

Returns

Newly allocated object

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

8.20.2.6 VEXTERNC int Vparam_AtomData_ctor2 (Vparam_AtomData * *thee*)

FORTTRAN stub to construct the object.

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 121 of file [vparam.c](#).

8.20.2.7 VEXTERNC void Vparam_AtomData_dtor (Vparam_AtomData ** *thee*)

Destroy object.

Author

Nathan Baker

Parameters

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

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

8.20.2.8 VEXTERNC void Vparam_AtomData_dtor2 (Vparam_AtomData * *thee*)

FORTTRAN stub to destroy object.

Author

Nathan Baker

Parameters

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

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

8.20.2.9 VEXTERNC Vparam* Vparam_ctor ()

Construct the object.

Author

Nathan Baker

Returns

Newly allocated [Vparam](#) object

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

8.20.2.10 VEXTERNC int Vparam_ctor2 (Vparam * *thee*)

FORTTRAN stub to construct the object.

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line [193](#) of file [vparam.c](#).

8.20.2.11 VEXTERNC void Vparam_dtor (Vparam ** *thee*)

Destroy object.

Author

Nathan Baker

Parameters

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

Definition at line [213](#) of file [vparam.c](#).

8.20.2.12 VEXTERNC void Vparam_dtor2 (Vparam * *thee*)

FORTTRAN stub to destroy object.

Author

Nathan Baker

Parameters

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

Definition at line [223](#) of file [vparam.c](#).

8.20.2.13 VEXTERNC Vparam_AtomData* Vparam_getAtomData (Vparam * *thee*, char *resName*[VMAX_ARGLEN], char *atomName*[VMAX_ARGLEN])

Get atom data.

Author

Nathan Baker

Parameters

<i>thee</i>	Vparam object
-------------	-------------------------------

<i>resName</i>	Residue name
<i>atomName</i>	Atom name

Returns

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

Note

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

See also

[Vparam_readFlatFile](#))

Definition at line 267 of file [vparam.c](#).

8.20.2.14 VEXTERNC Vparam_ResData* Vparam_getResData (Vparam * *thee*, char *resName*[VMAX_ARGLEN])

Get residue data.

Author

Nathan Baker

Parameters

<i>thee</i>	Vparam object
<i>resName</i>	Residue name

Returns

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

Note

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

See also

[Vparam_readFlatFile](#))

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

8.20.2.15 VEXTERNC unsigned long int Vparam_memChk (Vparam * *thee*)

Get number of bytes in this object and its members.

Author

Nathan Baker

Parameters

<i>thee</i>	Vparam object
-------------	-------------------------------

Returns

Number of bytes allocated for object

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

8.20.2.16 VEXTERNC int Vparam_readFlatFile (Vparam * *thee*, const char * *idev*, const char * *iofmt*, const char * *thost*, const char * *fname*)

Read a flat-file format parameter database.

Author

Nathan Baker

Parameters

<i>thee</i>	Vparam object
<i>idev</i>	Input device type (FILE/BUFF/UNIX/INET)
<i>iofmt</i>	Input device format (ASCII/XDR)
<i>thost</i>	Input hostname (for sockets)
<i>fname</i>	Input FILE/BUFF/UNIX/INET name (see note below for format)

Returns

1 if successful, 0 otherwise

Note

The database file should have the following format:

```
RESIDUE ATOM CHARGE RADIUS EPSILON
```

where RESIDUE is the residue name string, ATOM is the atom name string, CHARGE is the charge in e, RADIUS is the van der Waals radius (σ_i) in Å, and EPSILON is the van der Waals well-depth (ϵ_i) in kJ/mol. See the [Vparam](#) structure documentation for the precise definitions of σ_i and ϵ_i .

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

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

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

Definition at line 445 of file [vparam.c](#).

8.20.2.17 VEXTERNC int Vparam_readXMLFile (Vparam * *thee*, const char * *idev*, const char * *iofmt*, const char * *thost*, const char * *fname*)

Read an XML format parameter database.

Author

Todd Dolinsky

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 306 of file [vparam.c](#).

8.20.2.18 VEXTERNC void Vparam_ResData_copyTo (Vparam_ResData * *thee*, Vparam_ResData * *dest*)

Copy current residue object to destination.

Author

Todd Dolinsky

Parameters

<i>thee</i>	Pointer to source object
<i>dest</i>	Pointer to destination object

Definition at line 585 of file [vparam.c](#).

8.20.2.19 VEXTERNC Vparam_ResData* Vparam_ResData_ctor (Vmem * *mem*)

Construct the object.

Author

Nathan Baker

Parameters

<i>mem</i>	Memory object of Vparam master class
------------	--

Returns

Newly allocated object

Definition at line 135 of file [vparam.c](#).

8.20.2.20 VEXTERNC int Vparam_ResData_ctor2 (Vparam_ResData * *thee*, Vmem * *mem*)

FORTTRAN stub to construct the object.

Author

Nathan Baker

Parameters

<i>thee</i>	Allocated memory
<i>mem</i>	Memory object of Vparam master class

Returns

1 if successful, 0 otherwise

Definition at line [147](#) of file [vparam.c](#).

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

Destroy object.

Author

Nathan Baker

Parameters

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

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

8.20.2.22 VEXTERNC void Vparam_ResData_dtor2 (Vparam_ResData * *thee*)

FORTTRAN stub to destroy object.

Author

Nathan Baker

Parameters

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

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

8.21 Vpbe class

The Poisson-Boltzmann master class.

Files

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

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)*
aaauthor 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_getlons](#) ([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.*

8.21.1 Detailed Description

The Poisson-Boltzmann master class.

Contains objects and parameters used in every PBE calculation, regardless of method.

8.21.2 Function Documentation

8.21.2.1 **VEXTERNC Vpbe*** *Vpbe_ctor* (*Valist ** *alist*, *int ionNum*, *double ** *ionConc*, *double ** *ionRadii*, *double ** *ionQ*, *double T*, *double soluteDiel*, *double solventDiel*, *double solventRadius*, *int focusFlag*, *double sdens*, *double z_mem*, *double L*, *double membraneDiel*, *double V*)

Construct Vpbe object.

Author

Nathan Baker and Mike Holst and Michael Grabe

Note

This is partially based on some of Mike Holst's PMG code. Here are a few of the original function comments: kappa is defined as follows:

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

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

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

and the factor of 10^{-16} results from converting cm^2 to \AA^2 , noting that the 1000 in the denominator has converted m^3 to cm^3 , since the ionic strength I_s is assumed to have been provided in moles per liter, which is moles per 1000 cm^3 .

Returns

Pointer to newly allocated Vpbe object

Parameters

<i>alist</i>	Atom list
<i>ionNum</i>	Number of counterion species
<i>ionConc</i>	Array containing counterion concentrations (M)
<i>ionRadii</i>	Array containing counterion radii (A)
<i>ionQ</i>	Array containing counterion charges (e)
<i>T</i>	Temperature for Boltzmann distribution (K)
<i>soluteDiel</i>	Solute internal dielectric constant
<i>solventDiel</i>	Solvent dielectric constant
<i>solventRadius</i>	Solvent probe radius for surfaces that use it (A)
<i>focusFlag</i>	1 if focusing operation, 0 otherwise
<i>sdens</i>	Vacc sphere density
<i>z_mem</i>	Membrane location (A)
<i>L</i>	Membrane thickness (A)

<i>membraneDiel</i>	Membrane dielectric constant
<i>V</i>	Transmembrane potential (V)

Definition at line 246 of file [vpbe.c](#).

8.21.2.2 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*)

FORTTRAN stub to construct Vpbe object.

Author

Nathan Baker and Mike Holst and Michael Grabe

Note

This is partially based on some of Mike Holst's PMG code. Here are a few of the original function comments: kappa is defined as follows:

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

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

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

and the factor of 10^{-16} results from converting cm^2 to angstroms^2 , noting that the 1000 in the denominator has converted m^3 to cm^3 , since the ionic strength I_s is assumed to have been provided in moles per liter, which is moles per 1000 cm^3 .

Bug The focusing flag is currently not used!!

Returns

1 if successful, 0 otherwise

Parameters

<i>thee</i>	Pointer to memory allocated for Vpbe object
<i>alist</i>	Atom list
<i>ionNum</i>	Number of counterion species
<i>ionConc</i>	Array containing counterion concentrations (M)
<i>ionRadii</i>	Array containing counterion radii (A)
<i>ionQ</i>	Array containing counterion charges (e)
<i>T</i>	Temperature for Boltzmann distribution (K)
<i>soluteDiel</i>	Solute internal dielectric constant
<i>solventDiel</i>	Solvent dielectric constant

<i>solventRadius</i>	Solvent probe radius for surfaces that use it (A)
<i>focusFlag</i>	1 if focusing operation, 0 otherwise
<i>sdens</i>	Vacc sphere density
<i>z_mem</i>	Membrane location (A)
<i>L</i>	Membrane thickness (A)
<i>membraneDiel</i>	Membrane dielectric constant
<i>V</i>	Transmembrane potential (V)

Definition at line 264 of file [vpbe.c](#).

8.21.2.3 VEXTERNC void Vpbe_dtor (Vpbe ** *thee*)

Object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 467 of file [vpbe.c](#).

8.21.2.4 VEXTERNC void Vpbe_dtor2 (Vpbe * *thee*)

FORTTRAN stub object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 475 of file [vpbe.c](#).

8.21.2.5 VEXTERNC double Vpbe_getBulkIonicStrength (Vpbe * *thee*)

Get bulk ionic strength.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Bulk ionic strength (M)

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

8.21.2.6 VEXTERNC double Vpbe_getCoulombEnergy1 (Vpbe * thee)

Calculate coulombic energy of set of charges.

Perform an inefficient double sum to calculate the Coulombic energy of a set of charges in a homogeneous dielectric (with permittivity equal to the protein interior) and zero ionic strength. Result is returned in units of $k_B T$. The sum can be restriction to charges present in simplices of specified color (pcolor); if (color == -1) no restrictions are used.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Coulombic energy in units of $k_B T$.

Definition at line 481 of file [vpbe.c](#).

8.21.2.7 VEXTERNC double Vpbe_getDeblen (Vpbe * thee)

Get Debye-Huckel screening length.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Debye-Huckel screening length (Å)

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

8.21.2.8 VEXTERNC double Vpbe_getGamma (Vpbe * thee)

Get apolar coefficient.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Apolar coefficient (kJ/mol/Å²)

8.21.2.9 VEXTERNC int Vpbe_getIons (Vpbe * *thee*, int * *nion*, double *ionConc*[MAXION], double *ionRadii*[MAXION], double *ionQ*[MAXION])

Get information about the counterion species present.

Author

Nathan Baker

Parameters

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

Returns

Number of ions

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

8.21.2.10 VEXTERNC double Vpbe_getLmem (Vpbe * *thee*)

Get length of the membrane (Å)

aaauthor Michael Grabe.

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Length of the membrane (Å)

Definition at line 209 of file [vpbe.c](#).

8.21.2.11 VEXTERNC double Vpbe_getMaxIonRadius (Vpbe * *thee*)

Get maximum radius of ion species.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Maximum radius (A)

Definition at line 127 of file [vpbe.c](#).

8.21.2.12 VEXTERNC double Vpbe_getmembraneDiel (Vpbe * *thee*)

Get membrane dielectric constant.

Author

Michael Grabe

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Membrane dielectric constant

Definition at line 221 of file [vpbe.c](#).

8.21.2.13 VEXTERNC double Vpbe_getmemv (Vpbe * *thee*)

Get membrane potential (kT)

Author

Michael Grabe

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Definition at line 233 of file [vpbe.c](#).

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

Get coordinates of solute center.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

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

Definition at line 107 of file [vpbe.c](#).

8.21.2.15 VEXTERNC double Vpbe_getSoluteCharge (Vpbe * *thee*)

Get total solute charge.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Total solute charge (e)

Definition at line 186 of file [vpbe.c](#).

8.21.2.16 VEXTERNC double Vpbe_getSoluteDiel (Vpbe * *thee*)

Get solute dielectric constant.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Solute dielectric constant

Definition at line 99 of file [vpbe.c](#).

8.21.2.17 VEXTERNC double Vpbe_getSoluteRadius (Vpbe * *thee*)

Get sphere radius which bounds biomolecule.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Sphere radius which bounds biomolecule (A)

Definition at line 162 of file [vpbe.c](#).

8.21.2.18 VEXTERNC double Vpbe_getSoluteXlen (Vpbe * *thee*)

Get length of solute in x dimension.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Length of solute in x dimension (A)

Definition at line 168 of file [vpbe.c](#).

8.21.2.19 VEXTERNC double Vpbe_getSoluteYlen (Vpbe * *thee*)

Get length of solute in y dimension.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Length of solute in y dimension (A)

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

8.21.2.20 VEXTERNC double Vpbe_getSoluteZlen (Vpbe * *thee*)

Get length of solute in z dimension.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Length of solute in z dimension (A)

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

8.21.2.21 VEXTERNC double Vpbe_getSolventDiel (Vpbe * *thee*)

Get solvent dielectric constant.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Solvent dielectric constant

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

8.21.2.22 VEXTERNC double Vpbe_getSolventRadius (Vpbe * *thee*)

Get solvent molecule radius.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Solvent molecule radius (A)

Definition at line 120 of file [vpbe.c](#).

8.21.2.23 VEXTERNC double Vpbe_getTemperature (Vpbe * *thee*)

Get temperature.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Temperature (K)

Definition at line 91 of file [vpbe.c](#).

8.21.2.24 VEXTERNC Vacc* Vpbe_getVacc (Vpbe * *thee*)

Get accessibility oracle.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Pointer to internal Vacc object

Definition at line 76 of file [vpbe.c](#).

8.21.2.25 VEXTERNC Valist* Vpbe_getValist (Vpbe * *thee*)

Get atom list.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Pointer to internal Valist object

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

8.21.2.26 VEXTERNC double Vpbe_getXkappa (Vpbe * *thee*)

Get Debye-Huckel parameter.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

Bulk Debye-Huckel parameter (Å)

Definition at line 134 of file [vpbe.c](#).

8.21.2.27 VEXTERNC double Vpbe_getZkappa2 (Vpbe * *thee*)

Get modified squared Debye-Huckel parameter.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

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

Definition at line 148 of file [vpbe.c](#).

8.21.2.28 VEXTERNC double Vpbe_getZmagic (Vpbe * *thee*)

Get charge scaling factor.

Author

Nathan Baker and Mike Holst

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

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

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

8.21.2.29 VEXTERNC double Vpbe_getzmem (Vpbe * *thee*)

Get z position of the membrane bottom.

Author

Michael Grabe

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

z value of membrane (A)

Definition at line [197](#) of file [vpbe.c](#).

8.21.2.30 VEXTERNC unsigned long int Vpbe_memChk (Vpbe * *thee*)

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

Author

Nathan Baker

Parameters

<i>thee</i>	Vpbe object
-------------	-------------

Returns

The memory used by this structure and its contents in bytes

Definition at line [523](#) of file [vpbe.c](#).

8.22 Vstring class

Provides a collection of useful non-ANSI string functions.

Files

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

Functions

- char * [Vstring_wrappedtext](#) (const char *str, int right_margin, int left_padding)
- 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.

8.22.1 Detailed Description

Provides a collection of useful non-ANSI string functions.

8.22.2 Function Documentation

8.22.2.1 VEXTERNC int Vstring_isdigit (const char * tok)

A modified sscanf that examines the complete string.

Author

Todd Dolinsky

Parameters

<i>tok</i>	The string to examine
------------	-----------------------

Returns

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

Definition at line [130](#) of file [vstring.c](#).

8.22.2.2 VEXTERNC int Vstring_strcasecmp (const char * s1, const char * s2)

Case-insensitive string comparison (BSD standard)

Author

Copyright (c) 1988-1993 The Regents of the University of California. Copyright (c) 1995-1996 Sun Microsystems, Inc.

Note

Copyright (c) 1988-1993 The Regents of the University of California. Copyright (c) 1995-1996 Sun Microsystems, Inc.

Parameters

<i>s1</i>	First string for comparison
<i>s2</i>	Second string for comparison

Returns

An integer less than, equal to, or greater than zero if *s1* is found, respectively, to be less than, to match, or be greater than *s2*. (Source: Linux man pages)

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

8.22.2.3 VEXTERNC char * Vstring_wrappedtext (const char * *str*, int *right_margin*, int *left_padding*)

Creates a wrapped and indented string from an input string

Author

Tucker Beck

Note

This function allocates a new string, so be sure to free it!

Creates a wrapped and indented string from an input string

Author

Tucker Beck

Note

This function allocates a new string, so be sure to free it!

Note

: The +2 is for the newline character and the null-terminating character;

Parameters

<i>str</i>	The input string to wrap and indent
<i>right_margin</i>	The number of characters to the right margin
<i>left_padding</i>	The number of characters in the left indent

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

8.23 Vunit class

Collection of constants and conversion factors.

Files

- file [vunit.h](#)

Contains a collection of useful constants and conversion factors.

Macros

- `#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

8.23.1 Detailed Description

Collection of constants and conversion factors.

8.24 Vgrid class

Oracle for Cartesian mesh data.

Files

- file [vgrid.c](#)
Class Vgrid methods.
- file [vgrid.h](#)
Potential oracle for Cartesian mesh data.

Data Structures

- struct [sVgrid](#)
Electrostatic potential oracle for Cartesian mesh data.

Macros

- #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)
FORTTRAN 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:

$$\|u\|_{H_1} = \left(\int_{\Omega} |\nabla u(x)|^2 dx + \int_{\Omega} |u(x)|^2 dx \right)^{1/2}$$

8.24.1 Detailed Description

Oracle for Cartesian mesh data.

8.24.2 Function Documentation

8.24.2.1 **VEXTERNC** `Vgrid* Vgrid_ctor (int nx, int ny, int nz, double hx, double hy, double hz, double xmin, double ymin, double zmin, double * data)`

Construct Vgrid object with values obtained from `Vpmg_readDX` (for example)

Author

Nathan Baker

Parameters

<i>nx</i>	Number grid points in x direction
<i>ny</i>	Number grid points in y direction
<i>nz</i>	Number grid points in z direction
<i>hx</i>	Grid spacing in x direction
<i>hy</i>	Grid spacing in y direction
<i>hz</i>	Grid spacing in z direction
<i>xmin</i>	x coordinate of lower grid corner
<i>ymin</i>	y coordinate of lower grid corner
<i>zmin</i>	z coordinate of lower grid corner
<i>data</i>	<i>nx</i> * <i>ny</i> * <i>nz</i> array of data. This can be VNULL if you are planning to read in data later with one of the read routines

Returns

Newly allocated and initialized Vgrid object

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

8.24.2.2 **VEXTERNC** `int Vgrid_ctor2 (Vgrid * thee, int nx, int ny, int nz, double hx, double hy, double hz, double xmin, double ymin, double zmin, double * data)`

Initialize Vgrid object with values obtained from `Vpmg_readDX` (for example)

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to newly allocated Vgrid object
<i>nx</i>	Number grid points in x direction
<i>ny</i>	Number grid points in y direction
<i>nz</i>	Number grid points in z direction
<i>hx</i>	Grid spacing in x direction
<i>hy</i>	Grid spacing in y direction
<i>hz</i>	Grid spacing in z direction

<i>xmin</i>	x coordinate of lower grid corner
<i>ymin</i>	y coordinate of lower grid corner
<i>zmin</i>	z coordinate of lower grid corner
<i>data</i>	nx*ny*nz array of data. This can be VNULL if you are planning to read in data later with one of the read routines

Returns

Newly allocated and initialized Vgrid object

Definition at line 111 of file [vgrid.c](#).

8.24.2.3 VEXTERNC int Vgrid_curvature (Vgrid * *thee*, double *pt*[3], int *cflag*, double * *curv*)

Get second derivative values at a point.

Author

Steve Bond and Nathan Baker

Parameters

<i>thee</i>	Pointer to Vgrid object
<i>pt</i>	Location to evaluate second derivative
<i>cflag</i>	<ul style="list-style-type: none"> • 0: Reduced Maximal Curvature • 1: Mean Curvature (Laplace) • 2: Gauss Curvature • 3: True Maximal Curvature
<i>curv</i>	Specified curvature value

Returns

1 if successful, 0 if off grid

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

8.24.2.4 VEXTERNC void Vgrid_dtor (Vgrid ** *thee*)

Object destructor.

Author

Nathan Baker

Parameters

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

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

8.24.2.5 VEXTERNC void Vgrid_dtor2 (Vgrid * *thee*)

FORTTRAN stub object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 164 of file [vgrid.c](#).

8.24.2.6 VEXTERNC int Vgrid_gradient (Vgrid * *thee*, double *pt*[3], double *grad*[3])

Get first derivative values at a point.

Author

Nathan Baker and Steve Bond

Parameters

<i>thee</i>	Pointer to Vgrid object
<i>pt</i>	Location to evaluate gradient
<i>grad</i>	Gradient

Returns

1 if successful, 0 if off grid

Definition at line 377 of file [vgrid.c](#).

8.24.2.7 VEXTERNC double Vgrid_integrate (Vgrid * *thee*)

Get the integral of the data.

Author

Nathan Baker

Parameters

<i>thee</i>	Vgrid object
-------------	--------------

Returns

Integral of data

Definition at line [1356](#) of file [vgrid.c](#).

8.24.2.8 VEXTERNC unsigned long int Vgrid_memChk (Vgrid * *thee*)

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

Author

Nathan Baker

Parameters

<i>thee</i>	Vgrid object
-------------	--------------

Returns

The memory used by this structure and its contents in bytes

Definition at line [62](#) of file [vgrid.c](#).

8.24.2.9 VEXTERNC double Vgrid_normH1 (Vgrid * *thee*)

Get the H_1 norm (or energy norm) of the data. This returns the integral:

$$\|u\|_{H_1} = \left(\int_{\Omega} |\nabla u(x)|^2 dx + \int_{\Omega} |u(x)|^2 dx \right)^{1/2}$$

.

Author

Nathan Baker

Parameters

<i>thee</i>	Vgrid object
-------------	--------------

Returns

Integral of data

Definition at line [1497](#) of file [vgrid.c](#).

8.24.2.10 VEXTERNC double Vgrid_normL1 (Vgrid * *thee*)

Get the L_1 norm of the data. This returns the integral:

$$\|u\|_{L_1} = \int_{\Omega} |u(x)| dx$$

.

Author

Nathan Baker

Parameters

<i>thee</i>	Vgrid object
-------------	--------------

Returns

 L_1 norm of dataDefinition at line 1394 of file [vgrid.c](#).8.24.2.11 VEXTERNC double Vgrid_normL2 (Vgrid * *thee*)Get the L_2 norm of the data. This returns the integral:

$$\|u\|_{L_2} = \left(\int_{\Omega} |u(x)|^2 dx \right)^{1/2}$$

.

Author

Nathan Baker

Parameters

<i>thee</i>	Vgrid object
-------------	--------------

Returns

 L_2 norm of dataDefinition at line 1424 of file [vgrid.c](#).8.24.2.12 VEXTERNC double Vgrid_normLinf (Vgrid * *thee*)Get the L_{∞} norm of the data. This returns the integral:

$$\|u\|_{L_{\infty}} = \sup_{x \in \Omega} |u(x)|$$

.

Author

Nathan Baker

Parameters

<i>thee</i>	Vgrid object
-------------	--------------

Returns

L_∞ norm of data

Definition at line 1512 of file [vgrid.c](#).

8.24.2.13 VEXTERNC int Vgrid_readDX (Vgrid * *thee*, const char * *iodev*, const char * *iofmt*, const char * *thost*, const char * *fname*)

Read in data in OpenDX grid format.

Note

All dimension information is given in order: z, y, x

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Load grid from an input file using sockets.

Author

Nathan Baker

Definition at line 584 of file [vgrid.c](#).

8.24.2.14 VEXTERNC int Vgrid_readGZ (Vgrid * *thee*, const char * *fname*)

Read in OpenDX data in GZIP format.

Author

Dave Gohara

Returns

1 if successful, 0 otherwise

Parameters

<i>thee</i>	Object with grid data to write
<i>fname</i>	Path to write to

Definition at line 460 of file [vgrid.c](#).

8.24.2.15 VEXTERNC double Vgrid_seminormH1 (Vgrid * *thee*)

Get the H_1 semi-norm of the data. This returns the integral:

$$|u|_{H_1} = \left(\int_{\Omega} |\nabla u(x)|^2 dx \right)^{1/2}$$

.

Author

Nathan Baker

Parameters

<i>thee</i>	Vgrid object
-------------	--------------

Returns

Integral of data

Definition at line 1454 of file [vgrid.c](#).

8.24.2.16 VEXTERNC int Vgrid_value (Vgrid * *thee*, double *x*[3], double * *value*)

Get potential value (from mesh or approximation) at a point.

Author

Nathan Baker

Parameters

<i>thee</i>	Vgrid obejct
<i>x</i>	Point at which to evaluate potential
<i>value</i>	Value of data at point x

Returns

1 if successful, 0 if off grid

Definition at line 178 of file [vgrid.c](#).

8.24.2.17 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

<i>thee</i>	Grid object
<i>idev</i>	Output device type (FILE/BUFF/UNIX/INET)
<i>iofmt</i>	Output device format (ASCII/XDR)
<i>thost</i>	Output hostname (for sockets)
<i>fname</i>	Output FILE/BUFF/UNIX/INET name
<i>title</i>	Title to be inserted in grid file
<i>pvec</i>	Partition weight (if 1: point in current partition, if 0 point not in current partition if $> 0 \ \&\& \ < 1$ point on/near boundary)

Definition at line 1007 of file [vgrid.c](#).

8.24.2.18 VEXTERNC void Vgrid_writeUHBD (Vgrid * *thee*, const char * *idev*, 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

<i>thee</i>	Grid object
<i>idev</i>	Output device type (FILE/BUFF/UNIX/INET)
<i>iofmt</i>	Output device format (ASCII/XDR)
<i>thost</i>	Output hostname (for sockets)
<i>fname</i>	Output FILE/BUFF/UNIX/INET name
<i>title</i>	Title to be inserted in grid file
<i>pvec</i>	Partition weight (if 1: point in current partition, if 0 point not in current partition if $> 0 \ \&\& \ < 1$ point on/near boundary)

Bug This routine does not respect partition information

Definition at line 1258 of file [vgrid.c](#).

8.25 Vmgrid class

Oracle for Cartesian mesh data.

Files

- file [vmgrid.c](#)
Class Vmgrid methods.
- file [vmgrid.h](#)
Multiresolution oracle for Cartesian mesh data.

Data Structures

- struct [sVmgrid](#)
Multiresolution oracle for Cartesian mesh data.

Macros

- `#define VMGRIDMAX 20`
The maximum number of levels in the grid hierarchy.

Typedefs

- typedef struct [sVmgrid](#) Vmgrid
Declaration of the Vmgrid class as the Vmgrid 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)
FORTTRAN stub object destructor.
- VEXTERNC int [Vmgrid_addGrid](#) (Vmgrid *thee, Vgrid *grid)
Add a grid to the hierarchy.
- VEXTERNC int [Vmgrid_curvature](#) (Vmgrid *thee, double pt[3], int cflag, double *curv)
Get second derivative values at a point.
- VEXTERNC int [Vmgrid_gradient](#) (Vmgrid *thee, double pt[3], double grad[3])
Get first derivative values at a point.
- VEXTERNC Vgrid * [Vmgrid_getGridByNum](#) (Vmgrid *thee, int num)
Get specific grid in hierarchy.
- VEXTERNC Vgrid * [Vmgrid_getGridByPoint](#) (Vmgrid *thee, double pt[3])
Get grid in hierarchy which contains specified point or VNULL.

8.25.1 Detailed Description

Oracle for Cartesian mesh data.

8.25.2 Function Documentation

8.25.2.1 VEXTERNC int Vmgrid_addGrid (Vmgrid * *thee*, Vgrid * *grid*)

Add a grid to the hierarchy.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to object to be destroyed
<i>grid</i>	Grid to be added. As mentioned above, we would prefer to have the finest grid added first, next-finest second, ..., coarsest last – this is how the grid will be searched when looking up values for points. However, this is not enforced to provide flexibility for cases where the dataset is decomposed into disjoint partitions, etc.

Returns

1 if successful, 0 otherwise

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

8.25.2.2 VEXTERNC Vmgrid* Vmgrid_ctor ()

Construct Vmgrid object.

Author

Nathan Baker

Returns

Newly allocated and initialized Vmgrid object

Definition at line 57 of file [vmgrid.c](#).

8.25.2.3 VEXTERNC int Vmgrid_ctor2 (Vmgrid * *thee*)

Initialize Vmgrid object.

Author

Nathan Baker

Parameters

<i>thee</i>	Newly allocated Vmgrid object
-------------	-------------------------------

Returns

Newly allocated and initialized Vmgrid object

Definition at line 72 of file [vmgrid.c](#).

8.25.2.4 VEXTERNC int Vmgrid_curvature (Vmgrid * *thee*, double *pt*[3], int *cflag*, double * *curv*)

Get second derivative values at a point.

Author

Nathan Baker (wrapper for Vgrid routine by Steve Bond)

Parameters

<i>thee</i>	Pointer to Vmgrid object
<i>pt</i>	Location to evaluate second derivative
<i>cflag</i>	<ul style="list-style-type: none"> • 0: Reduced Maximal Curvature • 1: Mean Curvature (Laplace) • 2: Gauss Curvature • 3: True Maximal Curvature
<i>curv</i>	Specified curvature value

Returns

1 if successful, 0 if off grid

Definition at line 138 of file [vmgrid.c](#).

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

Object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 88 of file [vmgrid.c](#).

8.25.2.6 VEXTERNC void Vmgrid_dtor2 (Vmgrid * *thee*)

FORTTRAN stub object destructor.

Author

Nathan Baker

Parameters

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

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

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

Get specific grid in hierarchy.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to Vmgrid object
<i>num</i>	Number of grid in hierarchy

Returns

Pointer to specified grid

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

Get grid in hierarchy which contains specified point or VNULL.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to Vmgrid object
<i>pt</i>	Point to check

Returns

Pointer to specified grid

8.25.2.9 VEXTERNC int Vmgrid_gradient (Vmgrid * *thee*, double *pt*[3], double *grad*[3])

Get first derivative values at a point.

Author

Nathan Baker and Steve Bond

Parameters

<i>thee</i>	Pointer to Vmgrid object
<i>pt</i>	Location to evaluate gradient
<i>grad</i>	Gradient

Returns

1 if successful, 0 if off grid

Definition at line 167 of file [vmgrid.c](#).

8.25.2.10 `VEXTERNC int Vmgrid_value (Vmgrid * thee, double x[3], double * value)`

Get potential value (from mesh or approximation) at a point.

Author

Nathan Baker

Parameters

<i>thee</i>	Vmgrid obejct
<i>x</i>	Point at which to evaluate potential
<i>value</i>	Value of data at point x

Returns

1 if successful, 0 if off grid

Definition at line 107 of file [vmgrid.c](#).

8.26 Vopot class

Potential oracle for Cartesian mesh data.

Files

- file [vopot.c](#)
Class Vopot methods.
- file [vopot.h](#)
Potential oracle for Cartesian mesh data.

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)
FORTTRAN 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.

8.26.1 Detailed Description

Potential oracle for Cartesian mesh data.

8.26.2 Function Documentation

8.26.2.1 VEXTERNC Vopot* Vopot_ctor (Vmgrid * *mgrid*, Vpbe * *pbe*, Vbcfl *bcfl*)

Construct Vopot object with values obtained from Vpmg_readDX (for example)

Author

Nathan Baker

Parameters

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

Returns

Newly allocated and initialized Vopot object

Definition at line 65 of file [vopot.c](#).

8.26.2.2 VEXTERNC int Vopot_ctor2 (Vopot * *thee*, Vmgrid * *mgrid*, Vpbe * *pbe*, Vbcfl *bcfl*)

Initialize Vopot object with values obtained from Vpmg_readDX (for example)

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 80 of file [vopot.c](#).

8.26.2.3 VEXTERNC int Vopot_curvature (Vopot * *thee*, double *pt*[3], int *cflag*, double * *curv*)

Get second derivative values at a point.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to Vopot object
<i>pt</i>	Location to evaluate second derivative
<i>cflag</i>	<ul style="list-style-type: none"> • 0: Reduced Maximal Curvature • 1: Mean Curvature (Laplace) • 2: Gauss Curvature • 3: True Maximal Curvature
<i>curv</i>	Set to specified curvature value

Returns

1 if successful, 0 otherwise

Definition at line [214](#) of file [vopot.c](#).

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

Object destructor.

Author

Nathan Baker

Parameters

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

Definition at line [94](#) of file [vopot.c](#).

8.26.2.5 VEXTERNC void Vopot_dtor2 (Vopot * *thee*)

FORTTRAN stub object destructor.

Author

Nathan Baker

Parameters

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

Definition at line [107](#) of file [vopot.c](#).

8.26.2.6 VEXTERNC int Vopot_gradient (Vopot * *thee*, double *pt*[3], double *grad*[3])

Get first derivative values at a point.

Author

Nathan Baker

Parameters

<i>thee</i>	Pointer to Vopot object
<i>pt</i>	Location to evaluate gradient
<i>grad</i>	Gradient

Returns

1 if successful, 0 otherwise

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

8.26.2.7 VEXTERNC int Vopot_pot (Vopot * *thee*, double *x*[3], double * *pot*)

Get potential value (from mesh or approximation) at a point.

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 114 of file [vopot.c](#).

8.27 Vpmg class

A wrapper for Mike Holst's PMG multigrid code.

Files

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

Data Structures

- struct [sVpmg](#)
Contains public data members for Vpmg class/module.

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)
FORTTRAN stub constructor for the Vpmg class (uses previously-allocated memory)
- VEXTERNC void [Vpmg_dtor](#) ([Vpmg](#) **thee)
Object destructor.
- VEXTERNC void [Vpmg_dtor2](#) ([Vpmg](#) *thee)
FORTTRAN 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` srfrm, `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` srfrm)
- 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` srfrm)
- 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` pbe-type, `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 dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.*

 - VEXTERNC void `Vpmg_ibDirectPolForce` (`Vpmg *thee`, `Vgrid` *perm, `Vgrid` *induced, int atomID, double force[3])
- Ionic boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.*

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

- VEXTERNC void `Vpmg_dbDirectPolForce` (`Vpmg *thee`, `Vgrid *perm`, `Vgrid *induced`, `int atomID`, `double force[3]`)
Dielectric boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.
- VEXTERNC void `Vpmg_dbNLDirectPolForce` (`Vpmg *thee`, `Vgrid *perm`, `Vgrid *nlInduced`, `int atomID`, `double force[3]`)
Dielectric boundary direct polarization force between permanent multipoles and non-local induced dipoles. Keep in mind that the "non-local" induced dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.
- VEXTERNC void `Vpmg_qfMutualPolForce` (`Vpmg *thee`, `Vgrid *induced`, `Vgrid *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.
- VPRIVATE void `bcolcomp` (`int *iparm`, `double *rparm`, `int *iwork`, `double *rwork`, `double *values`, `int *rowind`, `int *colptr`, `int *flag`)
Build a column-compressed matrix in Harwell-Boeing format.
- VPRIVATE void `bcolcomp2` (`int *iparm`, `double *rparm`, `int *nx`, `int *ny`, `int *nz`, `int *iz`, `int *ipc`, `double *rpc`, `double *ac`, `double *cc`, `double *values`, `int *rowind`, `int *colptr`, `int *flag`)
Build a column-compressed matrix in Harwell-Boeing format.
- VPRIVATE void `bcolcomp3` (`int *nx`, `int *ny`, `int *nz`, `int *ipc`, `double *rpc`, `double *ac`, `double *cc`, `double *values`, `int *rowind`, `int *colptr`, `int *flag`)
Build a column-compressed matrix in Harwell-Boeing format.
- VPRIVATE void `bcolcomp4` (`int *nx`, `int *ny`, `int *nz`, `int *ipc`, `double *rpc`, `double *oC`, `double *cc`, `double *oE`, `double *oN`, `double *uC`, `double *values`, `int *rowind`, `int *colptr`, `int *flag`)
Build a column-compressed matrix in Harwell-Boeing format.
- VPRIVATE void `pcolcomp` (`int *nrow`, `int *ncol`, `int *nnzero`, `double *values`, `int *rowind`, `int *colptr`, `char *path`, `char *title`, `char *mxtype`)
Print a column-compressed matrix in Harwell-Boeing format.
- VEXTERNC void `Vpackmg` (`int *iparm`, `double *rparm`, `size_t *nrwk`, `int *niwk`, `int *nx`, `int *ny`, `int *nz`, `int *nlev`, `int *nu1`, `int *nu2`, `int *mgkey`, `int *itmax`, `int *istop`, `int *ipcon`, `int *nonlin`, `int *mgsmoo`, `int *mgprol`, `int *mgcoar`, `int *mgsolv`, `int *mgdisc`, `int *iinfo`, `double *errtol`, `int *ipkey`, `double *omegal`, `double *omegan`, `int *irite`, `int *iperf`)
Print out a column-compressed sparse matrix in Harwell-Boeing format.

8.27.1 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.

8.27.2 Function Documentation

8.27.2.1 `VPRIVATE void bcolcomp (int * iparm, double * rparm, int * iwork, double * rwork, double * values, int * rowind, int * colptr, int * flag)`

Build a column-compressed matrix in Harwell-Boeing format.

Author

Tucker Beck [C Translation] Nathan Baker [Original] (mostly ripped off from Harwell-Boeing format documentation)Michael Schnieders)

Parameters

<i>iparm</i>	
<i>rparm</i>	
<i>iwork</i>	
<i>rwork</i>	
<i>values</i>	
<i>rowind</i>	
<i>colptr</i>	
<i>flag</i>	Operation selection parameter 0 = Use Poisson operator only 1 = Use linearization of full operation around current solution.

Definition at line [10726](#) of file [vpmg.c](#).

8.27.2.2 `VPRIVATE void bcolcomp2 (int * iparm, double * rparm, int * nx, int * ny, int * nz, int * iz, int * ipc, double * rpc, double * ac, double * cc, double * values, int * rowind, int * colptr, int * flag)`

Build a column-compressed matrix in Harwell-Boeing format.

Author

Tucker Beck [C Translation] Nathan Baker [Original] (mostly ripped off from Harwell-Boeing format documentation)Michael Schnieders)

Parameters

<i>iparm</i>	
<i>rparm</i>	
<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>iz</i>	

<i>ipc</i>	
<i>rpc</i>	
<i>ac</i>	
<i>cc</i>	
<i>values</i>	
<i>rowind</i>	
<i>colptr</i>	
<i>flag</i>	Operation selection parameter 0 = Use Poisson operator only 1 = Use linearization of full operation around current solution.

Definition at line 10780 of file [vpmg.c](#).

8.27.2.3 VPRIVATE void bcolcomp3 (int * *nx*, int * *ny*, int * *nz*, int * *ipc*, double * *rpc*, double * *ac*, double * *cc*, double * *values*, int * *rowind*, int * *colptr*, int * *flag*)

Build a column-compressed matrix in Harwell-Boeing format.

Author

Tucker Beck [C Translation] Nathan Baker [Original] (mostly ripped off from Harwell-Boeing format documentation)Michael Schnieders)

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>ac</i>	
<i>cc</i>	
<i>values</i>	
<i>rowind</i>	
<i>colptr</i>	
<i>flag</i>	

Definition at line 10816 of file [vpmg.c](#).

8.27.2.4 VPRIVATE void bcolcomp4 (int * *nx*, int * *ny*, int * *nz*, int * *ipc*, double * *rpc*, double * *oC*, double * *cc*, double * *oE*, double * *oN*, double * *uC*, double * *values*, int * *rowind*, int * *colptr*, int * *flag*)

Build a column-compressed matrix in Harwell-Boeing format.

Author

Tucker Beck [C Translation] Nathan Baker [Original] (mostly ripped off from Harwell-Boeing format documentation)Michael Schnieders)

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>oC</i>	
<i>cc</i>	
<i>oE</i>	
<i>oN</i>	
<i>uC</i>	
<i>values</i>	
<i>rowind</i>	
<i>colptr</i>	
<i>flag</i>	

Definition at line 10843 of file [vpmg.c](#).

8.27.2.5 VPRIVATE void pcolcomp (int * *nrow*, int * *ncol*, int * *nnzero*, double * *values*, int * *rowind*, int * *colptr*, char * *path*, char * *title*, char * *mxtype*)

Print a column-compressed matrix in Harwell-Boeing format.

Author

Tucker Beck [C Translation] Nathan Baker [Original] (mostly ripped off from Harwell-Boeing format documentation)Michael Schnieders)

Parameters

<i>nrow</i>	
<i>ncol</i>	
<i>nnzero</i>	
<i>values</i>	
<i>rowind</i>	
<i>colptr</i>	
<i>path</i>	
<i>title</i>	
<i>mxtype</i>	

Definition at line 11008 of file [vpmg.c](#).

8.27.2.6 VEXTERNC void Vpackmg (int * *iparm*, double * *rparm*, size_t * *nrwk*, int * *niwk*, int * *nx*, int * *ny*, int * *nz*, int * *nlev*, int * *nu1*, int * *nu2*, int * *mgkey*, int * *itmax*, int * *istop*, int * *ipcon*, int * *nonlin*, int * *mgsmoo*, int * *mgprol*, int * *mgcoar*, int * *mgsolv*, int * *mgdisc*, int * *iinfo*, double * *errtol*, int * *ipkey*, double * *omegal*, double * *omegan*, int * *irite*, int * *iperf*)

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?

Definition at line 550 of file [mgsubd.c](#).

8.27.2.7 `VEXTERNC Vpmg* Vpmg_ctor (Vpmgp * parms, Vpbe * pbe, int focusFlag, Vpmg * pmgOLD, MGparm * mgparm, PBEparm_calcEnergy energyFlag)`

Constructor for the Vpmg class (allocates new memory)

Author

Nathan Baker

Returns

Pointer to newly allocated Vpmg object

Parameters

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

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

8.27.2.8 `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

Note

this is common to both replace/noreplace options

The fortran replacement functions are run along side the old fortran functions. This is due to the use of common variables in the fortran sub-routines. Once the fortran code has been successfully excised, these functions will no longer need to be called in tandem, and the fortran version may be dropped

Parameters

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

Definition at line 153 of file [vpmg.c](#).

8.27.2.9 VEXTERNC void Vpmg_dbDirectPolForce (Vpmg * *thee*, Vgrid * *perm*, Vgrid * *induced*, int *atomID*, double *force*[3])

Dielectric boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>perm</i>	Permanent multipole potential
<i>induced</i>	Induced dipole potential
<i>atomID</i>	Atom index
<i>force</i>	(returned) force

8.27.2.10 VEXTERNC int Vpmg_dbForce (Vpmg * *thee*, double * *dbForce*, int *atomID*, Vsurf_Meth *srfm*)

Calculate the dielectric boundary forces on the specified atom in units of $k_B T/AA$.

Author

Nathan Baker

Note

- Using the force evaluation methods of Im et al (Roux group), Comput Phys Commun, 111, 59–75 (1998). However, this gives the whole (self-interactions included) force – reaction field forces will have to be calculated at higher level.
- No contributions are made from higher levels of focusing.

Returns

1 if successful, 0 otherwise

Parameters

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

Definition at line 5995 of file [vpmg.c](#).

8.27.2.11 VEXTERNC void Vpmg_dbMutualPolForce (Vpmg * *thee*, Vgrid * *induced*, Vgrid * *nllInduced*, int *atomID*, double *force*[3])

Dielectric boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>induced</i>	Induced dipole potential
<i>nllInduced</i>	Non-local induced dipole potential
<i>atomID</i>	Atom index
<i>force</i>	(returned) force

8.27.2.12 VEXTERNC void Vpmg_dbNLDirectPolForce (Vpmg * *thee*, Vgrid * *perm*, Vgrid * *nllInduced*, int *atomID*, double *force*[3])

Dielectric bounday direct polarization force between permanent multipoles and non-local induced dipoles. Keep in mind that the "non-local" induced dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>perm</i>	Permanent multipole potential
<i>nllInduced</i>	Non-local induced dipole potential
<i>atomID</i>	Atom index
<i>force</i>	(returned) force

8.27.2.13 VEXTERNC void Vpmg_dbPermanentMultipoleForce (Vpmg * *thee*, int *atomID*, double *force*[3])

Compute the dielectric boundary force for permanent multipoles.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>atomID</i>	Atom index
<i>force</i>	(returned) force

8.27.2.14 VEXTERNC double Vpmg_dielEnergy (Vpmg * *thee*, int *extFlag*)

Get the "polarization" contribution to the electrostatic energy.

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

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

where epsilon is the dielectric parameter and u(x) is the dimensionless electrostatic potential. The energy is scaled to units of k_B T.

Author

Nathan Baker

Note

The value of this observable may be modified by setting restrictions on the subdomain over which it is calculated. Such limits can be set via Vpmg_setPart and are generally useful for parallel runs.

Returns

The polarization electrostatic energy in units of k_B T.

Parameters

<i>thee</i>	Vpmg object
<i>extFlag</i>	If this was a focused calculation, include (1 – for serial calculations) or ignore (0 – for parallel calculations) energy contributions from outside the focusing domain

Definition at line 1279 of file [vpmg.c](#).

8.27.2.15 VEXTERNC double Vpmg_dielGradNorm (Vpmg * *thee*)

Get the integral of the gradient of the dielectric function.

Using the dielectric map at the finest mesh level, calculate the integral of the norm of the dielectric function gradient routines of Im et al (see Vpmg_dbForce for reference):

$$\int \|\nabla \epsilon\| dx$$

where epsilon is the dielectric parameter. The integral is returned in units of A².

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 A².

Parameters

<i>thee</i>	Vpmg object
-------------	-------------

Definition at line 1342 of file [vpmg.c](#).

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

Object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 561 of file [vpmg.c](#).

8.27.2.17 VEXTERNC void Vpmg_dtor2 (Vpmg * *thee*)

FORTTRAN stub object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 571 of file [vpmg.c](#).

8.27.2.18 VEXTERNC double Vpmg_energy (Vpmg * *thee*, int *extFlag*)

Get the total electrostatic energy.

Author

Nathan Baker

Note

The value of this observable may be modified by setting restrictions on the subdomain over which it is calculated. Such limits can be set via `Vpmg_setPart` and are generally useful for parallel runs.

Returns

The electrostatic energy in units of `k_B T`.

Parameters

<i>thee</i>	Vpmg object
<i>extFlag</i>	If this was a focused calculation, include (1 – for serial calculations) or ignore (0 – for parallel calculations) energy contributions from outside the focusing domain

Definition at line 1248 of file [vpmg.c](#).

8.27.2.19 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

<i>thee</i>	Vpmg object
<i>atomID</i>	Atom index
<i>field</i>	The (returned) electric field

8.27.2.20 VEXTERNC int Vpmg_fillArray (Vpmg * *thee*, double * *vec*, Vdata_Type *type*, double *parm*, Vhal_PBEType *pbetype*, PBEparm * *pbeparm*)

Fill the specified array with accessibility values.

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Parameters

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

Definition at line 892 of file [vpmg.c](#).

8.27.2.21 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

Bug useDielMap could only be passed once, not three times, to this function - why not just once? that's what the call in [routines.c](#) ends up doing - just passing useDielMap three times. - P. Ellis 11/3/11

Parameters

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

Definition at line 5640 of file [vpmg.c](#).

8.27.2.22 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/Å.

Author

Nathan Baker

Note

- Using the force evaluation methods of Im et al (Roux group), Comput Phys Commun, 111, 59–75 (1998). However, this gives the whole (self-interactions included) force – reaction field forces will have to be calculated at higher level.
- No contributions are made from higher levels of focusing.

Returns

1 if successful, 0 otherwise

Parameters

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

Definition at line 5807 of file [vpmg.c](#).

8.27.2.23 VEXTERNC void Vpmg_ibDirectPolForce (Vpmg * *thee*, Vgrid * *perm*, Vgrid * *induced*, int *atomID*, double *force*[3])

Ionic boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>perm</i>	Permanent multipole potential
<i>induced</i>	Induced dipole potential
<i>atomID</i>	Atom index
<i>force</i>	(returned) force

8.27.2.24 VEXTERNC int Vpmg_ibForce (Vpmg * *thee*, double * *force*, int *atomID*, Vsurf_Meth *srfm*)

Calculate the osmotic pressure on the specified atom in units of $k_B T/AA$.

Author

Nathan Baker

Note

- Using the force evaluation methods of Im et al (Roux group), Comput Phys Commun, 111, 59–75 (1998). However, this gives the whole (self-interactions included) force – reaction field forces will have to be calculated at higher level.
- No contributions are made from higher levels of focusing.

Returns

1 if successful, 0 otherwise

Parameters

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

Definition at line 5830 of file [vpmg.c](#).

8.27.2.25 VEXTERNC void Vpmg_ibMutualPolForce (Vpmg * *thee*, Vgrid * *induced*, Vgrid * *nllInduced*, int *atomID*, double *force*[3])

Ionic boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>induced</i>	Induced dipole potential
<i>nllInduced</i>	Non-local induced dipole potential
<i>atomID</i>	Atom index
<i>force</i>	(returned) force

8.27.2.26 VEXTERNC void Vpmg_ibNLDirectPolForce (Vpmg * *thee*, Vgrid * *perm*, Vgrid * *nllInduced*, int *atomID*, double *force*[3])

Ionic boundary direct polarization force between permanent multipoles and non-local induced dipoles based on 5th order Keep in mind that the "non-local" induced dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>perm</i>	Permanent multipole potential
<i>nllInduced</i>	Induced dipole potential
<i>atomID</i>	Atom index
<i>force</i>	(returned) force

8.27.2.27 VEXTERNC void Vpmg_ibPermanentMultipoleForce (Vpmg * *thee*, int *atomID*, double *force*[3])

Compute the ionic boundary force for permanent multipoles.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>atomID</i>	Atom index
<i>force</i>	(returned) force

8.27.2.28 VEXTERNC unsigned long int Vpmg_memChk (Vpmg * *thee*)

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

Author

Nathan Baker

Returns

The memory used by this structure and its contents in bytes

Parameters

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

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

8.27.2.29 VEXTERNC void Vpmg_printColComp (Vpmg * *thee*, char *path*[72], char *title*[72], char *mxttype*[3], int *flag*)

Print out a column-compressed sparse matrix in Harwell-Boeing format.

Author

Nathan Baker

Bug Can this path variable be replaced with a Vio socket?

Parameters

<i>thee</i>	Vpmg object
<i>path</i>	The file to which the matrix is to be written
<i>title</i>	The title of the matrix
<i>mxttype</i>	The type of REAL-valued matrix, a 3-character string of the form "R_A" where the '_' can be one of: <ul style="list-style-type: none"> • S: symmetric matrix • U: unsymmetric matrix • H: Hermitian matrix • Z: skew-symmetric matrix • R: rectangular matrix
<i>flag</i>	The operator to compress: <ul style="list-style-type: none"> • 0: Poisson operator • 1: Linearization of the full Poisson-Boltzmann operator around the current solution

Definition at line 87 of file [vpmg.c](#).

8.27.2.30 VEXTERNC double Vpmg_qfAtomEnergy (Vpmg * *thee*, Vatom * *atom*)

Get the per-atom "fixed charge" contribution to the electrostatic energy.

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

$$G = qu(r),$$

where q is the charge and r is the location of the atom of interest. The result is returned in units of $k_B T$. Clearly, no self-interaction terms are removed. A factor a 1/2 has to be included to convert this to a real energy.

Author

Nathan Baker

Note

The value of this observable may be modified by setting restrictions on the subdomain over which it is calculated. Such limits can be set via `Vpmg_setPart` and are generally useful for parallel runs.

Returns

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

Parameters

<i>thee</i>	The Vpmg object
<i>atom</i>	The atom for energy calculations

Definition at line 1791 of file [vpmg.c](#).

8.27.2.31 VEXTERNC void Vpmg_qfDirectPolForce (Vpmg * *thee*, Vgrid * *perm*, Vgrid * *induced*, int *atomID*, double *force*[3], double *torque*[3])

q-Phi direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>perm</i>	Permanent multipole potential
<i>induced</i>	Induced dipole potential
<i>atomID</i>	Atom index

<i>force</i>	(returned) force
<i>torque</i>	(returned) torque

8.27.2.32 VEXTERNC double Vpmg_qfEnergy (Vpmg * *thee*, int *extFlag*)

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

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

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

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

Author

Nathan Baker

Note

The value of this observable may be modified by setting restrictions on the subdomain over which it is calculated. Such limits can be set via Vpmg_setPart and are generally useful for parallel runs.

Returns

The fixed charge electrostatic energy in units of k_B T.

Parameters

<i>thee</i>	Vpmg object
<i>extFlag</i>	If this was a focused calculation, include (1 – for serial calculations) or ignore (0 – for parallel calculations) energy contributions from outside the focusing domain

Definition at line 1687 of file [vpmg.c](#).

8.27.2.33 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 successful, 0 otherwise

Parameters

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

Definition at line 6252 of file [vpmg.c](#).

8.27.2.34 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

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

8.27.2.35 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 dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

Author

Michael Schnieders

Parameters

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

8.27.2.36 VEXTERNC double Vpmg_qfPermanentMultipoleEnergy (Vpmg * *thee*, int *atomID*)

Computes the permanent multipole electrostatic hydration energy (the polarization component of the hydration energy currently computed in TINKER).

Author

Michael Schnieders

Returns

The permanent multipole electrostatic hydration energy

Parameters

<i>thee</i>	Vpmg object
<i>atomID</i>	Atom index

8.27.2.37 VEXTERNC void Vpmg_qfPermanentMultipoleForce (Vpmg * *thee*, int *atomID*, double *force*[3], double *torque*[3])

Computes the q-Phi Force for permanent multipoles based on 5th order B-splines.

Author

Michael Schnieders

Parameters

<i>thee</i>	Vpmg object
<i>atomID</i>	Atom index
<i>force</i>	(returned) force
<i>torque</i>	(returned) torque

8.27.2.38 VEXTERNC double Vpmg_qmEnergy (Vpmg * *thee*, int *extFlag*)

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

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

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

for the NPBE and

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

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

Author

Nathan Baker

Note

The value of this observable may be modified by setting restrictions on the subdomain over which it is calculated. Such limits can be set via Vpmg_setPart and are generally useful for parallel runs.

Returns

The mobile charge electrostatic energy in units of *k_B* T.

Parameters

<i>thee</i>	Vpmg object
<i>extFlag</i>	If this was a focused calculation, include (1 – for serial calculations) or ignore (0 – for parallel calculations) energy contributions from outside the focusing domain

Definition at line 1386 of file [vpmg.c](#).

8.27.2.39 VEXTERNC void Vpmg_setPart (Vpmg * *thee*, double *lowerCorner*[3], double *upperCorner*[3], int *bflags*[6])

Set partition information which restricts the calculation of observables to a (rectangular) subset of the problem domain.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpmg object
<i>lowerCorner</i>	Partition lower corner
<i>upperCorner</i>	Partition upper corner
<i>bflags</i>	Booleans indicating whether a particular processor is on the boundary with another partition. 0 if the face is not bounded (next to) another partition, and 1 otherwise.

Definition at line 627 of file [vpmg.c](#).

8.27.2.40 VEXTERNC int Vpmg_solve (Vpmg * *thee*)

Solve the PBE using PMG.

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Parameters

<i>thee</i>	Vpmg object
-------------	-------------

Definition at line 401 of file [vpmg.c](#).

8.27.2.41 VEXTERNC int Vpmg_solveLaplace (Vpmg * *thee*)

Solve Poisson's equation with a homogeneous Laplacian operator using the solvent dielectric constant. This solution is performed by a sine wave decomposition.

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Note

This function is really only for testing purposes as the PMG multigrid solver can solve the homogeneous system much more quickly. Perhaps we should implement an FFT version at some point...

Parameters

<i>thee</i>	Vpmg object
-------------	-------------

Definition at line [7027](#) of file [vpmg.c](#).

8.27.2.42 VEXTERNC void Vpmg_unsetPart (Vpmg * *thee*)

Remove partition restrictions.

Author

Nathan Baker

Parameters

<i>thee</i>	Vpmg object
-------------	-------------

Definition at line [872](#) of file [vpmg.c](#).

8.28 Vpmgp class

Parameter structure for Mike Holst's PMGP code.

Files

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

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)
FORTTRAN 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)
FORTTRAN 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.

8.28.1 Detailed Description

Parameter structure for Mike Holst's PMGP code.

Note

Variables and many default values taken directly from PMG

8.28.2 Function Documentation

8.28.2.1 VEXTERNC Vpmgp* Vpmgp_ctor (MGparm * mgparm)

Construct PMG parameter object and initialize to default values.

Author

Nathan Baker

Parameters

<i>mgparm</i>	MGParm object containing parameters to be used in setup
---------------	---

Returns

Newly allocated and initialized Vpmgp object

Definition at line 76 of file [vpmgp.c](#).

8.28.2.2 VEXTERNC int Vpmgp_ctor2 (Vpmgp * thee, MGparm * mgparm)

FORTTRAN stub to construct PMG parameter object and initialize to default values.

Author

Nathan Baker

Parameters

<i>thee</i>	Newly allocated PMG object
<i>mgparm</i>	MGParm object containing parameters to be used in setup

Returns

1 if successful, 0 otherwise

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

8.28.2.3 VEXTERNC void Vpmgp_dtor (Vpmgp ** thee)

Object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 178 of file [vpmgp.c](#).

8.28.2.4 VEXTERN void Vpmgp_dtor2 (Vpmgp * *thee*)

FORTRAN stub for object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 193 of file [vpmgp.c](#).

8.28.2.5 VEXTERN void Vpmgp_makeCoarse (int *numLevel*, int *nxOld*, int *nyOld*, int *nzOld*, int * *nxNew*, int * *nyNew*, int * *nzNew*)

Coarsen the grid by the desired number of levels and determine the resulting numbers of grid points.

Author

Mike Holst and Nathan Baker

Parameters

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

Definition at line 312 of file [vpmgp.c](#).

8.28.2.6 VEXTERN void Vpmgp_size (Vpmgp * *thee*)

Determine array sizes and parameters for multigrid solver.

Author

Mike Holst and Nathan Baker

Parameters

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

Definition at line 196 of file [vpmgp.c](#).

8.29 C translation of Holst group PMG code

C translation of Holst group PMG code.

Macros

- #define **HARMO2**(a, b) $(2.0 * (a) * (b) / ((a) + (b)))$
Multigrid subroutines.
- #define **MAXIONS** 50
Specifies the PDE definition for PMG to solve.

Functions

- VPUBLIC void **VbuildA** (int *nx,int *ny,int *nz,int *ipkey,int *mgdisc,int *numdia,int *ipc,double *rpc,double *ac,double *cc,double *fc,double *xf,double *yf,double *zf,double *gxcf,double *gycf,double *gzcf,double *a1cf,double *a2cf,double *a3cf,double *ccf,double *fcf)
Build the Laplacian.
- VPUBLIC void **Vbuildband** (int *key,int *nx,int *ny,int *nz,int *ipc,double *rpc,double *ac,int *ipcB,double *rpcB,double *acB)
Banded matrix builder.
- VEXTERN void **Vbuildband1_7** (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *oC, double *oE, double *oN, double *uC, int *ipcB, double *rpcB, double *acB, int *n, int *m, int *lda)
Build the operator in banded form given the 7-diagonal form.
- VEXTERN void **Vbuildband1_27** (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *oC, double *oE, double *oN, double *uC, double *oNE, double *oNW, double *uE, double *uW, double *uN, double *uS, double *uNE, double *uNW, double *uSE, double *uSW, int *ipcB, double *rpcB, double *acB, int *n, int *m, int *lda)
Build the operator in banded form given the 27-diagonal form.
- VPUBLIC void **VbuildG** (int *nxf,int *nyf,int *nzf,int *nxc,int *nyc,int *nzc,int *numdia,double *pcFF,double *acFF,double *ac)
Build Galerkin matrix structures.
- VEXTERN void **VbuildG_1** (int *nxf, int *nyf, int *nzf, int *nx, int *ny, int *nz, double *oPC, double *oPN, double *oPS, double *oPE, double *oPW, double *oPNE, double *oPNW, double *oPSE, double *oPSW, double *uPC, double *uPN, double *uPS, double *uPE, double *uPW, double *uPNE, double *uPNW, double *uPSE, double *uPSW, double *dPC, double *dPN, double *dPS, double *dPE, double *dPW, double *dPNE, double *dPNW, double *dPSE, double *dPSW, double *oC, double *XoC, double *XoE, double *XoN, double *XuC, double *XoNE, double *XoNW, double *XuE, double *XuW, double *XuN, double *XuS, double *XuNE, double *XuNW, double *XuSE, double *XuSW)
Computes a 27-point galerkin coarse grid matrix from a 1-point (i.e., diagonal) fine grid matrix.
- VEXTERN void **VbuildG_7** (int *nxf, int *nyf, int *nzf, int *nx, int *ny, int *nz, double *oPC, double *oPN, double *oPS, double *oPE, double *oPW, double *oPNE, double *oPNW, double *oPSE, double *oPSW, double *uPC, double *uPN, double *uPS, double *uPE, double *uPW, double *uPNE, double *uPNW, double *uPSE, double *uPSW, double *dPC, double *dPN, double *dPS, double *dPE, double *dPW, double *dPNE, double *dPNW, double *dPSE, double *dPSW, double *oC, double *oE, double *oN, double *uC, double *XoC, double *XoE, double *XoN, double *XuC, double *XoNE, double *XoNW, double *XuE, double *XuW, double *XuN, double *XuS, double *XuNE, double *XuNW, double *XuSE, double *XuSW)
Computes a 27-point galerkin coarse grid matrix from a 7-point fine grid matrix.
- VEXTERN void **VbuildG_27** (int *nxf, int *nyf, int *nzf, int *nx, int *ny, int *nz, double *oPC, double *oPN, double *oPS, double *oPE, double *oPW, double *oPNE, double *oPNW, double *oPSE, double *oPSW, double *uPC, double *uPN, double *uPS, double *uPE, double *uPW, double *uPNE, double *uPNW, double

*uPSE, double *uPSW, double *dPC, double *dPN, double *dPS, double *dPE, double *dPW, double *dPNE, double *dPNW, double *dPSE, double *dPSW, double *oC, double *oE, double *oN, double *uC, double *oNE, double *oNW, double *uE, double *uW, double *uN, double *uS, double *uNE, double *uNW, double *uSE, double *uSW, double *XoC, double *XoE, double *XoN, double *XuC, double *XoNE, double *XoNW, double *XuE, double *XuW, double *XuN, double *XuS, double *XuNE, double *XuNW, double *XuSE, double *XuSW)

Compute a 27-point galerkin coarse grid matrix from a 27-point fine grid matrix.

- VPUBLIC void **VbuildP** (int *nxf,int *nyf,int *nzf,int *nxc,int *nyc,int *nzc,int *mgprol,int *ipc,double *rpc,double *pc,double *ac,double *xf,double *yf,double *zf)

Builds prolongation matrix.

- VPUBLIC void **Vcghs** (int *nx,int *ny,int *nz,int *ipc,double *rpc,double *ac,double *cc,double *fc,double *x,double *p,double *ap,double *r,int *itmax,int *iters,double *errtol,double *omega,int *iresid,int *iadjoint)

A collection of useful low-level routines (timing, etc).

- VPUBLIC void **Vgsrb** (int *nx,int *ny,int *nz,int *ipc,double *rpc,double *ac,double *cc,double *fc,double *x,double *w1,double *w2,double *r,int *itmax,int *iters,double *errtol,double *omega,int *iresid,int *iadjoint)

Guass-Seidel solver.

- VPUBLIC void **Vmatvec** (int *nx,int *ny,int *nz,int *ipc,double *rpc,double *ac,double *cc,double *x,double *y)

Matrix-vector multiplication routines.

- VEXTERNC void **Vnmatvec** (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *ac, double *cc, double *x, double *y, double *w1)

Break the matrix data-structure into diagonals and then call the matrix-vector routine.

- VEXTERNC void **Vmresid** (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *ac, double *cc, double *fc, double *x, double *r)

Break the matrix data-structure into diagonals and then call the residual routine.

- VEXTERNC void **Vnmresid** (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *ac, double *cc, double *fc, double *x, double *r, double *w1)

Break the matrix data-structure into diagonals and then call the residual routine.

- VEXTERNC void **Vrestrc** (int *nxf, int *nyf, int *nzf, int *nxc, int *nyc, int *nzc, double *xin, double *xout, double *pc)

Apply the restriction operator.

- VEXTERNC void **VinterpPMG** (int *nxc, int *nyc, int *nzc, int *nxf, int *nyf, int *nzf, double *xin, double *xout, double *pc)

Apply the prolongation operator.

- VEXTERNC void **Vextrac** (int *nxf, int *nyf, int *nzf, int *nxc, int *nyc, int *nzc, double *xin, double *xout)

Simple injection of a fine grid function into coarse grid.

- VEXTERNC void **Vmvcs** (int *nx,int *ny,int *nz,double *x,int *iz,double *w0,double *w1,double *w2,double *w3,int *istop,int *itmax,int *iters,int *ierror,int *nlev,int *ilev,int *nlev_real,int *mgsolv,int *iok,int *info,double *epsiln,double *errtol,double *omega,int *nu1,int *nu2,int *mgsmoo,int *ipc,double *rpc,double *pc,double *ac,double *cc,double *fc,double *tru)

MG helper functions.

- VPUBLIC void **Vmgdriv** (int *iparm,double *rparm,int *iwork,double *rwork,double *u,double *xf,double *yf,double *zf,double *gxcf,double *gycf,double *gzcf,double *a1cf,double *a2cf,double *a3cf,double *ccf,double *fcf,double *tcf)

Multilevel solver driver.

- VEXTERNC void **Vmgdriv2** (int *iparm, double *rparm, int *nx, int *ny, int *nz, double *u, int *iz, int *ipc, double *rpc, double *pc, double *ac, double *cc, double *fc, double *xf, double *yf, double *zf, double *gxcf, double *gycf, double *gzcf, double *a1cf, double *a2cf, double *a3cf, double *ccf, double *fcf, double *tcf)

Solves the pde using the multi-grid method.

- VEXTERNC void **Vmgsz** (int *mgcoar, int *mgdisc, int *mgsolv, int *nx, int *ny, int *nz, int *nlev, int *nxc, int *nyc, int *nzc, int *nf, int *nc, int *narr, int *narrc, int *n_rpc, int *n_iz, int *n_ipc, int *iretot, int *iintot)

This routine computes the required sizes of the real and integer work arrays for the multigrid code. these two sizes are a (complicated) function of input parameters.

- VPUBLIC void **Vmvmf** (int *nx,int *ny,int *nz,double *x,int *iz,double *w0,double *w1,double *w2,double *w3,double *w4,int *istop,int *itmax,int *iters,int *ierror,int *nlev,int *ilev,int *nlev_real,int *mgsolv,int *iok,int *iinfo,double *epsiln,double *errtol,double *omega,int *nu1,int *nu2,int *mgsmoo,int *ipc,double *rpc,double *pc,double *ac,double *cc,double *fc,double *tru)

Multigrid nonlinear solve iteration routine.

- VEXTERNC void **Vmvmf** (int *nx, int *ny, int *nz, double *x, int *iz, double *w0, double *w1, double *w2, double *w3, double *w4, int *istop, int *itmax, int *iters, int *ierror, int *nlev, int *ilev, int *nlev_real, int *mgsolv, int *iok, int *iinfo, double *epsiln, double *errtol, double *omega, int *nu1, int *nu2, int *mgsmoo, int *ipc, double *rpc, double *pc, double *ac, double *cc, double *fc, double *tru)

Nonlinear multilevel method.

- VPUBLIC void **Vbuildops** (int *nx,int *ny,int *nz,int *nlev,int *ipkey,int *iinfo,int *ido,int *iz,int *mgprol,int *mgcoar,int *mgsolv,int *mgdisc,int *ipc,double *rpc,double *pc,double *ac,double *cc,double *fc,double *xf,double *yf,double *zf,double *gxcf,double *gycf,double *gzcfc,double *a1cf,double *a2cf,double *a3cf,double *ccf,double *fcf,double *tcf)

Build operators, boundary arrays, modify affine vectors ido==0: do only fine level ido==1: do only coarse levels (including second op at coarsest) ido==2: do all levels ido==3: rebuild the second operator at the coarsest level.

- VEXTERNC void **Vbuildstr** (int *nx, int *ny, int *nz, int *nlev, int *iz)

Build the nexted operator framework in the array iz.

- VEXTERNC void **Vbuildgaler0** (int *nxf, int *nyf, int *nzf, int *nxc, int *nyc, int *nzc, int *ipkey, int *numdia, double *pcFF, int *ipcFF, double *rpcFF, double *acFF, double *ccFF, double *fcFF, int *ipc, double *rpc, double *ac, double *cc, double *fc)

Form the Galerkin coarse grid system.

- VPUBLIC void **Vxcopy** (int *nx,int *ny,int *nz,double *x,double *y)

A collection of useful low-level routines (timing, etc).

- VEXTERNC void **Vxcopy_small** (int *nx, int *ny, int *nz, double *x, double *y)

Copy operation for a grid function with boundary values. Quite simply copies one 3d matrix to another.

- VEXTERNC void **Vxcopy_large** (int *nx, int *ny, int *nz, double *x, double *y)

Copy operation for a grid function with boundary values. Quite simply copies one 3d matrix to another.

- VEXTERNC void **Vxaxpy** (int *nx, int *ny, int *nz, double *alpha, double *x, double *y)

saxpy operation for a grid function with boundary values.

- VEXTERNC double **Vxnorm1** (int *nx, int *ny, int *nz, double *x)

Norm operation for a grid function with boundary values.

- VEXTERNC double **Vxnorm2** (int *nx, int *ny, int *nz, double *x)

Norm operation for a grid function with boundary values.

- VEXTERNC double **Vxdot** (int *nx, int *ny, int *nz, double *x, double *y)

Inner product operation for a grid function with boundary values.

- VEXTERNC void **Vazeros** (int *nx, int *ny, int *nz, double *x)

Zero out operation for a grid function, including boundary values.

- VEXTERNC void **VfboundPMG** (int *ibound, int *nx, int *ny, int *nz, double *x, double *gxc, double *gyc, double *gzc)

Initialize a grid function to have a certain boundary value.

- VEXTERNC void **VfboundPMG00** (int *nx, int *ny, int *nz, double *x)

Initialize a grid function to have a zero boundary value.

- VEXTERNC void **Vaxrand** (int *nx, int *ny, int *nz, double *x)

Fill grid function with random values, including boundary values.

- VEXTERNC void **Vxscal** (int *nx, int *ny, int *nz, double *fac, double *x)

Scale operation for a grid function with boundary values.

- VEXTERNC void [Vprtmtd](#) (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *ac)
- VPUBLIC void [Vdpbsl](#) (double *abd,int *lda,int *n,int *m,double *b)
LINPACK interface.
- VPUBLIC void [Vmydefinltpbe](#) (int *tnion,double *tcharge,double *tsconc)
Set up the ionic species to be used in later calculations. This must be called before any other of the routines in this file.
- VEXTERNC void [Vmydefinltnpbe](#) (int *tnion, double *tcharge, double *tsconc)
Set up the ionic species to be used in later calculations. This must be called before any other of the routines in this file.
- VEXTERNC void [Vmydefinitsmpbe](#) (int *tnion, double *tcharge, double *tsconc, double *smvolume, double *smsize)
Set up the ionic species to be used in later calculations. This must be called before any other of the routines in this file.
- VEXTERNC void [Vc_vec](#) (double *coef, double *uin, double *uout, int *nx, int *ny, int *nz, int *ipkey)
Define the nonlinearity (vector version)
- VEXTERNC void [Vdc_vec](#) (double *coef, double *uin, double *uout, int *nx, int *ny, int *nz, int *ipkey)
Define the derivative of the nonlinearity (vector version)
- VEXTERNC void [Vc_vecpmg](#) (double *coef, double *uin, double *uout, int *nx, int *ny, int *nz, int *ipkey)
Define the nonlinearity (vector version)
- VEXTERNC void [Vc_vecsmpbe](#) (double *coef, double *uin, double *uout, int *nx, int *ny, int *nz, int *ipkey)
Define the nonlinearity (vector version)
- VEXTERNC void [Vnewdriv](#) (int *iparm,double *rparm,int *iwork,double *rwork,double *u,double *xf,double *yf,double *zf,double *gxcf,double *gycf,double *gzcf,double *a1cf,double *a2cf,double *a3cf,double *ccf,double *fcf,double *tcf)
Driver for the Newton Solver.
- VEXTERNC void [Vnewdriv2](#) (int *iparm, double *rparm, int *nx, int *ny, int *nz, double *u, int *iz, double *w1, double *w2, int *ipc, double *rpc, double *pc, double *ac, double *cc, double *fc, double *xf, double *yf, double *zf, double *gxcf, double *gycf, double *gzcf, double *a1cf, double *a2cf, double *a3cf, double *ccf, double *fcf, double *tcf)
Solves using Newton's Method.
- VPUBLIC void [Vfnewton](#) (int *nx,int *ny,int *nz,double *x,int *iz,double *w0,double *w1,double *w2,double *w3,int *istop,int *itmax,int *iters,int *ierror,int *nlev,int *ilev,int *nlev_real,int *mgsolv,int *iok,int *iinfo,double *epsiln,double *errtol,double *omega,int *nu1,int *nu2,int *mgsmoo,double *cprime,double *rhs,double *xtmp,int *ipc,double *rpc,double *pc,double *ac,double *cc,double *fc,double *tru)
Driver routines for the Newton method.
- VEXTERNC void [Vnewton](#) (int *nx, int *ny, int *nz, double *x, int *iz, double *w0, double *w1, double *w2, double *w3, int *istop, int *itmax, int *iters, int *ierror, int *nlev, int *ilev, int *nlev_real, int *mgsolv, int *iok, int *iinfo, double *epsiln, double *errtol, double *omega, int *nu1, int *nu2, int *mgsmoo, double *cprime, double *rhs, double *xtmp, int *ipc, double *rpc, double *pc, double *ac, double *cc, double *fc, double *tru)
Inexact-newton-multilevel method.
- VEXTERNC void [Vgetjac](#) (int *nx, int *ny, int *nz, int *nlev_real, int *iz, int *lev, int *ipkey, double *x, double *r, double *cprime, double *rhs, double *cc, double *pc)
Form the jacobian system.
- VPUBLIC void [Vpower](#) (int *nx,int *ny,int *nz,int *iz,int *ilev,int *ipc,double *rpc,double *ac,double *cc,double *w1,double *w2,double *w3,double *w4,double *eigmax,double *eigmax_model,double *tol,int *itmax,int *iters,int *iinfo)
Power methods for eigenvalue estimation.
- VEXTERNC void [Vipower](#) (int *nx, int *ny, int *nz, double *u, int *iz, double *w0, double *w1, double *w2, double *w3, double *w4, double *eigmin, double *eigmin_model, double *tol, int *itmax, int *iters, int *nlev, int *ilev, int *nlev_real, int *mgsolv, int *iok, int *iinfo, double *epsiln, double *errtol, double *omega, int *nu1, int *nu2, int *mgsmoo, int *ipc, double *rpc, double *pc, double *ac, double *cc, double *tru)
Standard inverse power method for minimum eigenvalue estimation.

- VEXTERNC void [Vsmooth](#) (int *nx,int *ny,int *nz,int *ipc,double *rpc,double *ac,double *cc,double *fc,double *x,double *w1,double *w2,double *r,int *itmax,int *iters,double *errtol,double *omega,int *iresid,int *iadjoint,int *meth)

Multigrid smoothing functions.

- VEXTERNC void [Vnsmooth](#) (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *ac, double *cc, double *fc, double *x, double *w1, double *w2, double *r, int *itmax, int *iters, double *errtol, double *omega, int *iresid, int *iadjoint, int *meth)

call the appropriate non-linear smoothing routine.

8.29.1 Detailed Description

C translation of Holst group PMG code.

8.29.2 Macro Definition Documentation

8.29.2.1 `#define HARMO2(a, b) (2.0 * (a) * (b) / ((a) + (b)))`

Multigrid subroutines.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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 at line 65 of file [mgsubd.h](#).

8.29.2.2 #define MAXIONS 50

Specifies the PDE definition for PMG to solve.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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 at line 62 of file [mypdec.h](#).

8.29.3 Function Documentation

8.29.3.1 VEXTERNC void Vaxrand (int * nx, int * ny, int * nz, double * x)

Fill grid function with random values, including boundary values.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces axrand from mikpckd.f

Parameters

<i>nx</i>	The size of the x dimension of the 3d matrix
<i>ny</i>	The size of the y dimension of the 3d matrix
<i>nz</i>	The size of the z dimension of the 3d matrix
<i>x</i>	The 3d matrix to fill

Definition at line 286 of file [mikpckd.c](#).

8.29.3.2 VEXTERNC void Vazeros (int * nx, int * ny, int * nz, double * x)

Zero out operation for a grid function, including boundary values.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces azeros from mikpckd.f

Parameters

<i>nx</i>	The size of the x dimension of the 3d matrix
<i>ny</i>	The size of the x dimension of the 3d matrix
<i>nz</i>	The size of the x dimension of the 3d matrix

x	The matrix to zero out
---	------------------------

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

```
8.29.3.3 VEXTERNC void VbuildA ( int * nx, int * ny, int * nz, int * ipkey, int * mgdisc, int * numdia, int * ipc, double * rpc,
double * ac, double * cc, double * fc, double * xf, double * yf, double * zf, double * gxcf, double * gycf, double * gzcf,
double * a1cf, double * a2cf, double * a3cf, double * ccf, double * fcf )
```

Build the Laplacian.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*
```

Author

Mike Holst [original], Tucker Beck [translation]

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*
```

Break the matrix data-structure into diagonals and then call the matrix build routine

Author

Tucker Beck [C Translation], Michael Holst [Original]

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>ipkey</i>	

<i>mgdisc</i>	
<i>numdia</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>ac</i>	
<i>cc</i>	
<i>fc</i>	
<i>xf</i>	
<i>yf</i>	
<i>zf</i>	
<i>gxcf</i>	
<i>gycf</i>	
<i>gzcf</i>	
<i>a1cf</i>	
<i>a2cf</i>	
<i>a3cf</i>	
<i>ccf</i>	
<i>fcf</i>	

Definition at line 52 of file [buildAd.c](#).

8.29.3.4 VEXTERNC void Vbuildband (int * *key*, int * *nx*, int * *ny*, int * *nz*, int * *ipc*, double * *rpc*, double * *ac*, int * *ipcB*, double * *rpcB*, double * *acB*)

Banded matrix builder.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*

```

Author

Mike Holst and Steve Bond [original], Tucker Beck [translation]

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*

```

Build and factor a banded matrix given a matrix in diagonal form.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces buildband from buildBd.f

Parameters

<i>key</i>	
<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>ac</i>	
<i>ipcB</i>	
<i>rpcB</i>	
<i>acB</i>	

Definition at line 52 of file [buildBd.c](#).

8.29.3.5 VEXTERNC void Vbuildband1_27 (int * *nx*, int * *ny*, int * *nz*, int * *ipc*, double * *rpc*, double * *oC*, double * *oE*, double * *oN*, double * *uC*, double * *oNE*, double * *oNW*, double * *uE*, double * *uW*, double * *uN*, double * *uS*, double * *uNE*, double * *uNW*, double * *uSE*, double * *uSW*, int * *ipcB*, double * *rpcB*, double * *acB*, int * *n*, int * *m*, int * *lda*)

Build the operator in banded form given the 27-diagonal form.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces buildband1_7 from buildBd.f

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>oC</i>	
<i>oE</i>	
<i>oN</i>	
<i>uC</i>	

<i>oNE</i>	
<i>oNW</i>	
<i>uE</i>	
<i>uW</i>	
<i>uN</i>	
<i>uS</i>	
<i>uNE</i>	
<i>uNW</i>	
<i>uSE</i>	
<i>uSW</i>	
<i>ipcB</i>	
<i>rpcB</i>	
<i>acB</i>	
<i>n</i>	
<i>m</i>	
<i>lda</i>	

Definition at line 178 of file [buildBd.c](#).

8.29.3.6 VEXTERNC void Vbuildband1_7 (int * *nx*, int * *ny*, int * *nz*, int * *ipc*, double * *rpc*, double * *oC*, double * *oE*, double * *oN*, double * *uC*, int * *ipcB*, double * *rpcB*, double * *acB*, int * *n*, int * *m*, int * *lda*)

Build the operator in banded form given the 7-diagonal form.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces buildband1_7 from buildBd.f

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>oC</i>	
<i>oE</i>	
<i>oN</i>	
<i>uC</i>	
<i>ipcB</i>	
<i>rpcB</i>	
<i>acB</i>	
<i>n</i>	

<i>m</i>	
<i>lda</i>	

Definition at line 114 of file [buildBd.c](#).

8.29.3.7 VEXTERNC void VbuildG (int * *nxf*, int * *nyf*, int * *nzf*, int * *nxc*, int * *nyc*, int * *nzc*, int * *numdia*, double * *pcFF*, double * *acFF*, double * *ac*)

Build Galerkin matrix structures.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*
```

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*

```

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>nxc</i>	
<i>nyc</i>	
<i>nzc</i>	
<i>numdia</i>	
<i>pcFF</i>	
<i>acFF</i>	

<i>ac</i>	
-----------	--

Definition at line 52 of file [buildGd.c](#).

8.29.3.8 VEXTERNC void VbuildG_1 (int * *nx*, int * *ny*, int * *nz*, int * *nx*, int * *ny*, int * *nz*, double * *oPC*, double * *oPN*, double * *oPS*, double * *oPE*, double * *oPW*, double * *oPNE*, double * *oPNW*, double * *oPSE*, double * *oPSW*, double * *uPC*, double * *uPN*, double * *uPS*, double * *uPE*, double * *uPW*, double * *uPNE*, double * *uPNW*, double * *uPSE*, double * *uPSW*, double * *dPC*, double * *dPN*, double * *dPS*, double * *dPE*, double * *dPW*, double * *dPNE*, double * *dPNW*, double * *dPSE*, double * *dPSW*, double * *oC*, double * *XoC*, double * *XoE*, double * *XoN*, double * *XuC*, double * *XoNE*, double * *XoNW*, double * *XuE*, double * *XuW*, double * *XuN*, double * *XuS*, double * *XuNE*, double * *XuNW*, double * *XuSE*, double * *XuSW*)

Computes a 27-point galerkin coarse grid matrix from a 1-point (i.e., diagonal) fine grid matrix.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces buildG_1 from buildGd.f

Expressions for the galerkin coarse grid stencil XA in terms of the fine grid matrix stencil A and the interpolation operator stencil P. these stencils have the form:

```
XA := array([
matrix([ [ -XdNW(i,j,k), -XdN(i,j,k), -XdNE(i,j,k) ], [ -XdW(i,j,k), -XdC(i,j,k), -XdE(i,j,k) ], [ -XdSW(i,j,k), -XdS(i,j,k), -XdSE(i,j,k) ] ]),
matrix([ [ -XoNW(i,j,k), -XoN(i,j,k), -XoNE(i,j,k) ], [ -XoW(i,j,k), XoC(i,j,k), -XoE(i,j,k) ], [ -XoSW(i,j,k), -XoS(i,j,k), -XoSE(i,j,k) ] ]),
matrix([ [ -XuNW(i,j,k), -XuN(i,j,k), -XuNE(i,j,k) ], [ -XuW(i,j,k), -XuC(i,j,k), -XuE(i,j,k) ], [ -XuSW(i,j,k), -XuS(i,j,k), -XuSE(i,j,k) ] ] )]:
```

```
A := array([
matrix([ [ 0, 0, 0 ], [ 0, 0, 0 ], [ 0, 0, 0 ] ]),
matrix([ [ 0, 0, 0 ], [ 0, oC(i,j,k), 0 ], [ 0, 0, 0 ] ]),
matrix([ [ 0, 0, 0 ], [ 0, 0, 0 ], [ 0, 0, 0 ] ])
```

```
P := array([
matrix([ [ dPNW(i,j,k), dPN(i,j,k), dPNE(i,j,k) ], [ dPW(i,j,k), dPC(i,j,k), dPE(i,j,k) ], [ dPSW(i,j,k), dPS(i,j,k), dPSE(i,j,k) ] ]),
matrix([ [ oPNW(i,j,k), oPN(i,j,k), oPNE(i,j,k) ], [ oPW(i,j,k), oPC(i,j,k), oPE(i,j,k) ], [ oPSW(i,j,k), oPS(i,j,k), oPSE(i,j,k) ] ]),
matrix([ [ uPNW(i,j,k), uPN(i,j,k), uPNE(i,j,k) ], [ uPW(i,j,k), uPC(i,j,k), uPE(i,j,k) ], [ uPSW(i,j,k), uPS(i,j,k), uPSE(i,j,k) ] ] )]:
```

Parameters

<i>nx</i>	
-----------	--

<i>nyf</i>	
<i>nzf</i>	
<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>oPC</i>	
<i>oPN</i>	
<i>oPS</i>	
<i>oPE</i>	
<i>oPW</i>	
<i>oPNE</i>	
<i>oPNW</i>	
<i>oPSE</i>	
<i>oPSW</i>	
<i>uPC</i>	
<i>uPN</i>	
<i>uPS</i>	
<i>uPE</i>	
<i>uPW</i>	
<i>uPNE</i>	
<i>uPNW</i>	
<i>uPSE</i>	
<i>uPSW</i>	
<i>dPC</i>	
<i>dPN</i>	
<i>dPS</i>	
<i>dPE</i>	
<i>dPW</i>	
<i>dPNE</i>	
<i>dPNW</i>	
<i>dPSE</i>	
<i>dPSW</i>	
<i>oC</i>	
<i>XoC</i>	
<i>XoE</i>	
<i>XoN</i>	
<i>XuC</i>	
<i>XoNE</i>	
<i>XoNW</i>	
<i>XuE</i>	
<i>XuW</i>	
<i>XuN</i>	
<i>XuS</i>	
<i>XuNE</i>	

<i>XuNW</i>	
<i>XuSE</i>	
<i>XuSW</i>	

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

```
8.29.3.9 VEXTERNC void VbuildG_27 ( int * nxf, int * nyf, int * nzf, int * nx, int * ny, int * nz, double * oPC, double * oPN,
double * oPS, double * oPE, double * oPW, double * oPNE, double * oPNW, double * oPSE, double * oPSW, double *
uPC, double * uPN, double * uPS, double * uPE, double * uPW, double * uPNE, double * uPNW, double * uPSE,
double * uPSW, double * dPC, double * dPN, double * dPS, double * dPE, double * dPW, double * dPNE, double *
dPNW, double * dPSE, double * dPSW, double * oC, double * oE, double * oN, double * uC, double * oNE, double *
oNW, double * uE, double * uW, double * uN, double * uS, double * uNE, double * uNW, double * uSE, double * uSW,
double * XoC, double * XoE, double * XoN, double * XuC, double * XoNE, double * XoNW, double * XuE, double *
XuW, double * XuN, double * XuS, double * XuNE, double * XuNW, double * XuSE, double * XuSW )
```

Compute a 27-point galerkin coarse grid matrix from a 27-point fine grid matrix.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces buildG_27 from buildGd.f

Expressions for the galerkin coarse grid stencil XA in terms of the fine grid matrix stencil A and the interpolation operator stencil P. these stencils have the form:

```
XA := array([
matrix([ [ -XdNW(i,j,k), -XdN(i,j,k), -XdNE(i,j,k) ], [ -XdW(i,j,k), -XdC(i,j,k), -XdE(i,j,k) ], [ -XdSW(i,j,k), -XdS(i,j,k), -XdSE(i,j,k) ] ]),
matrix([ [ -XoNW(i,j,k), -XoN(i,j,k), -XoNE(i,j,k) ], [ -XoW(i,j,k), XoC(i,j,k), -XoE(i,j,k) ], [ -XoSW(i,j,k), -XoS(i,j,k), -XoSE(i,j,k) ] ]),
matrix([ [ -XuNW(i,j,k), -XuN(i,j,k), -XuNE(i,j,k) ], [ -XuW(i,j,k), -XuC(i,j,k), -XuE(i,j,k) ], [ -XuSW(i,j,k), -XuS(i,j,k), -XuSE(i,j,k) ] ] )]:
A := array([
matrix([ [ -dNW(i,j,k), -dN(i,j,k), -dNE(i,j,k) ], [ -dW(i,j,k), -dC(i,j,k), -dE(i,j,k) ], [ -dSW(i,j,k), -dS(i,j,k), -dSE(i,j,k) ] ]),
matrix([ [ -oNW(i,j,k), -oN(i,j,k), -oNE(i,j,k) ], [ -oW(i,j,k), oC(i,j,k), -oE(i,j,k) ], [ -oSW(i,j,k), -oS(i,j,k), -oSE(i,j,k) ] ]),
matrix([ [ -uNW(i,j,k), -uN(i,j,k), -uNE(i,j,k) ], [ -uW(i,j,k), -uC(i,j,k), -uE(i,j,k) ], [ -uSW(i,j,k), -uS(i,j,k), -uSE(i,j,k) ] ] )]:
P := array([
matrix([ [ dPNW(i,j,k), dPN(i,j,k), dPNE(i,j,k) ], [ dPW(i,j,k), dPC(i,j,k), dPE(i,j,k) ], [ dPSW(i,j,k), dPS(i,j,k), dPSE(i,j,k) ] ]),
matrix([ [ oPNW(i,j,k), oPN(i,j,k), oPNE(i,j,k) ], [ oPW(i,j,k), oPC(i,j,k), oPE(i,j,k) ], [ oPSW(i,j,k), oPS(i,j,k), oPSE(i,j,k) ] ]),
matrix([ [ uPNW(i,j,k), uPN(i,j,k), uPNE(i,j,k) ], [ uPW(i,j,k), uPC(i,j,k), uPE(i,j,k) ], [ uPSW(i,j,k), uPS(i,j,k), uPSE(i,j,k) ] ] )]:
```

in addition, A is assumed to be symmetric so that:

```
oS := proc(x,y,z) RETURN( oN(x,y-1,z) ): end: oW := proc(x,y,z) RETURN( oE(x-1,y,z) ): end: oSE := proc(x,y,z) RETURN( oNE(x-1,y-1,z) ): end:
oS := proc(x,y,z) RETURN( oN(x,y-1,z) ): end: oW := proc(x,y,z) RETURN( oE(x-1,y,z) ): end: oSE := proc(x,y,z) RETURN( oNE(x-1,y-1,z) ): end:
```



```
dC := proc(x,y,z) RETURN( uC(x,y,z-1) ): end: dW := proc(x,y,z) RETURN( uE(x-1,y,z-1) ): end: dE := proc(x,y,z)
RETURN( uW(x+1,y,z-1) ): end:
```

```
dN := proc(x,y,z) RETURN( uS(x,y+1,z-1) ): end: dNW := proc(x,y,z) RETURN( uSE(x-1,y+1,z-1) ): end: dNE :=
proc(x,y,z) RETURN( uSW(x+1,y+1,z-1) ): end:
```

```
dS := proc(x,y,z) RETURN( uN(x,y-1,z-1) ): end: dSW := proc(x,y,z) RETURN( uNE(x-1,y-1,z-1) ): end: dSE :=
proc(x,y,z) RETURN( uNW(x+1,y-1,z-1) ): end:
```

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>oPC</i>	
<i>oPN</i>	
<i>oPS</i>	
<i>oPE</i>	
<i>oPW</i>	
<i>oPNE</i>	
<i>oPNW</i>	
<i>oPSE</i>	
<i>oPSW</i>	
<i>uPC</i>	
<i>uPN</i>	
<i>uPS</i>	
<i>uPE</i>	
<i>uPW</i>	
<i>uPNE</i>	
<i>uPNW</i>	
<i>uPSE</i>	
<i>uPSW</i>	
<i>dPC</i>	
<i>dPN</i>	
<i>dPS</i>	
<i>dPE</i>	
<i>dPW</i>	
<i>dPNE</i>	
<i>dPNW</i>	
<i>dPSE</i>	
<i>dPSW</i>	
<i>oC</i>	
<i>oE</i>	

<i>oN</i>	
<i>uC</i>	
<i>oNE</i>	
<i>oNW</i>	
<i>uE</i>	
<i>uW</i>	
<i>uN</i>	
<i>uS</i>	
<i>uNE</i>	
<i>uNW</i>	
<i>uSE</i>	
<i>uSW</i>	
<i>XoC</i>	
<i>XoE</i>	
<i>XoN</i>	
<i>XuC</i>	
<i>XoNE</i>	
<i>XoNW</i>	
<i>XuE</i>	
<i>XuW</i>	
<i>XuN</i>	
<i>XuS</i>	
<i>XuNE</i>	
<i>XuNW</i>	
<i>XuSE</i>	
<i>XuSW</i>	

Definition at line 1247 of file [buildGd.c](#).

8.29.3.10 VEXTERNC void VbuildG_7 (int * *nx*, int * *ny*, int * *nz*, int * *nx*, int * *ny*, int * *nz*, double * *oPC*, double * *oPN*, double * *oPS*, double * *oPE*, double * *oPW*, double * *oPNE*, double * *oPNW*, double * *oPSE*, double * *oPSW*, double * *uPC*, double * *uPN*, double * *uPS*, double * *uPE*, double * *uPW*, double * *uPNE*, double * *uPNW*, double * *uPSE*, double * *uPSW*, double * *dPC*, double * *dPN*, double * *dPS*, double * *dPE*, double * *dPW*, double * *dPNE*, double * *dPNW*, double * *dPSE*, double * *dPSW*, double * *oC*, double * *oE*, double * *oN*, double * *uC*, double * *XoC*, double * *XoE*, double * *XoN*, double * *XuC*, double * *XoNE*, double * *XoNW*, double * *XuE*, double * *XuW*, double * *XuN*, double * *XuS*, double * *XuNE*, double * *XuNW*, double * *XuSE*, double * *XuSW*)

Computes a 27-point galerkin coarse grid matrix from a 7-point fine grid matrix.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces buildG_7 from buildGd.f

Expressions for the galerkin coarse grid stencil XA in terms of the fine grid matrix stencil A and the interpolation operator stencil P. these stencils have the form:

XA := array([
matrix([[-XdNW(i,j,k), -XdN(i,j,k), -XdNE(i,j,k)], [-XdW(i,j,k), -XdC(i,j,k), -XdE(i,j,k)], [-XdSW(i,j,k), -XdS(i,j,k), -XdS↵
E(i,j,k)]]),

```
matrix([ [ -XoNW(i,j,k), -XoN(i,j,k), -XoNE(i,j,k) ], [ -XoW(i,j,k), XoC(i,j,k), -XoE(i,j,k) ], [ -XoSW(i,j,k), -XoS(i,j,k), -XoS↵
E(i,j,k) ] ]),
```

```
matrix([ [ -XuNW(i,j,k), -XuN(i,j,k), -XuNE(i,j,k) ], [ -XuW(i,j,k), -XuC(i,j,k), -XuE(i,j,k) ], [ -XuSW(i,j,k), -XuS(i,j,k), -XuS↵
E(i,j,k) ] ] )):
```

```
A := array([
```

```
matrix([ [ 0, 0, 0 ], [ 0, -dC(i,j,k), 0 ], [ 0, 0, 0 ] ]),
```

```
matrix([ [ 0, -oN(i,j,k), 0 ], [ -oW(i,j,k), oC(i,j,k), -oE(i,j,k) ], [ 0, -oS(i,j,k), 0 ] ]),
```

```
matrix([ [ 0, 0, 0 ], [ 0, -uC(i,j,k), 0 ], [ 0, 0, 0 ] ])
```

```
P := array([
```

```
matrix([ [ dPNW(i,j,k), dPN(i,j,k), dPNE(i,j,k) ], [ dPW(i,j,k), dPC(i,j,k), dPE(i,j,k) ], [ dPSW(i,j,k), dPS(i,j,k), dPSE(i,j,k) ] ]),
```

```
matrix([ [ oPNW(i,j,k), oPN(i,j,k), oPNE(i,j,k) ], [ oPW(i,j,k), oPC(i,j,k), oPE(i,j,k) ], [ oPSW(i,j,k), oPS(i,j,k), oPSE(i,j,k) ] ]),
```

```
matrix([ [ uPNW(i,j,k), uPN(i,j,k), uPNE(i,j,k) ], [ uPW(i,j,k), uPC(i,j,k), uPE(i,j,k) ], [ uPSW(i,j,k), uPS(i,j,k), uPSE(i,j,k) ] ] )
]):
```

in addition, A is assumed to be symmetric so that:

```
oS := proc(x,y,z) RETURN( oN(x,y-1,z) ): end: oW := proc(x,y,z) RETURN( oE(x-1,y,z) ): end: dC := proc(x,y,z) RETU↵
RN( uC(x,y,z-1) ): end:
```

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>oPC</i>	
<i>oPN</i>	
<i>oPS</i>	
<i>oPE</i>	
<i>oPW</i>	
<i>oPNE</i>	
<i>oPNW</i>	
<i>oPSE</i>	
<i>oPSW</i>	
<i>uPC</i>	
<i>uPN</i>	
<i>uPS</i>	
<i>uPE</i>	
<i>uPW</i>	
<i>uPNE</i>	
<i>uPNW</i>	
<i>uPSE</i>	

<i>uPSW</i>	
<i>dPC</i>	
<i>dPN</i>	
<i>dPS</i>	
<i>dPE</i>	
<i>dPW</i>	
<i>dPNE</i>	
<i>dPNW</i>	
<i>dPSE</i>	
<i>dPSW</i>	
<i>oC</i>	
<i>oE</i>	
<i>oN</i>	
<i>uC</i>	
<i>XoC</i>	
<i>XoE</i>	
<i>XoN</i>	
<i>XuC</i>	
<i>XoNE</i>	
<i>XoNW</i>	
<i>XuE</i>	
<i>XuW</i>	
<i>XuN</i>	
<i>XuS</i>	
<i>XuNE</i>	
<i>XuNW</i>	
<i>XuSE</i>	
<i>XuSW</i>	

Definition at line 445 of file [buildGd.c](#).

8.29.3.11 VEXTERNC void Vbuildgaler0 (int * *nxl*, int * *nyl*, int * *nzl*, int * *nxc*, int * *nyc*, int * *nzc*, int * *ipkey*, int * *numdia*, double * *pcFF*, int * *ipcFF*, double * *rpcFF*, double * *acFF*, double * *ccFF*, double * *fcFF*, int * *ipc*, double * *rpc*, double * *ac*, double * *cc*, double * *fc*)

Form the Galerkin coarse grid system.

Note

Although the fine grid matrix may be 7 or 27 diagonal, the coarse grid matrix is always 27 diagonal. (only 14 stored due to symmetry.)

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces buildgaler0 from mgsbnd.f

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>nxc</i>	
<i>nyc</i>	
<i>nzc</i>	
<i>ipkey</i>	
<i>numdia</i>	
<i>pcFF</i>	
<i>ipcFF</i>	
<i>rpcFF</i>	
<i>acFF</i>	
<i>ccFF</i>	
<i>fcFF</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>ac</i>	
<i>cc</i>	
<i>fc</i>	

Definition at line 336 of file [mgsubd.c](#).

8.29.3.12 VEXTERNC void Vbuildops (int * *nx*, int * *ny*, int * *nz*, int * *nlev*, int * *ipkey*, int * *iinfo*, int * *ido*, int * *iz*, int * *mgprol*, int * *mgcoar*, int * *mgsoiv*, int * *mgdisc*, int * *ipc*, double * *rpc*, double * *pc*, double * *ac*, double * *cc*, double * *fc*, double * *xf*, double * *yf*, double * *zf*, double * *gxcf*, double * *gycf*, double * *gzcf*, double * *a1cf*, double * *a2cf*, double * *a3cf*, double * *ccf*, double * *fcf*, double * *tcf*)

Build operators, boundary arrays, modify affine vectors ido==0: do only fine level ido==1: do only coarse levels (including second op at coarsest) ido==2: do all levels ido==3: rebuild the second operator at the coarsest level.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*

```

Note

The fine level must be build before any coarse levels.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Replaces buildops from mgsubd.f

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>nlev</i>	
<i>ipkey</i>	
<i>iinfo</i>	
<i>ido</i>	
<i>iz</i>	
<i>mgprol</i>	
<i>mgcoar</i>	
<i>mgsovl</i>	
<i>mgdisc</i>	
<i>ipc</i>	
<i>rpc</i>	

<i>pc</i>	
<i>ac</i>	
<i>cc</i>	
<i>fc</i>	
<i>xf</i>	
<i>yf</i>	
<i>zf</i>	
<i>gxcf</i>	
<i>gycf</i>	
<i>gzcf</i>	
<i>a1cf</i>	
<i>a2cf</i>	
<i>a3cf</i>	
<i>ccf</i>	
<i>fcf</i>	
<i>tcf</i>	

Definition at line 52 of file [mgsubd.c](#).

8.29.3.13 `VEXTERNC void VbuildP (int * nx, int * ny, int * nz, int * nxc, int * nyc, int * nzc, int * mgprol, int * ipc, double * rpc, double * pc, double * ac, double * xf, double * yf, double * zf)`

Builds prolongation matrix.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*

```

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>nxc</i>	
<i>nyc</i>	
<i>nzc</i>	
<i>mgprol</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>pc</i>	
<i>ac</i>	
<i>xf</i>	
<i>yf</i>	
<i>zf</i>	

Definition at line 52 of file [buildPd.c](#).

8.29.3.14 VEXTERNC void Vbuildstr (int * nx, int * ny, int * nz, int * nlev, int * iz)

Build the nexted operator framework in the array iz.

Note

iz(50,i) indexes into the gridfcn arrays for each level i=(1,...,nlev+1) as follows:

```

fun(i) = fun (iz(1,i)) bndx(i) = bndx(iz(2,i)) bndy(i) = bndy(iz(3,i)) bndz(i) = bndz(iz(4,i)) ipc(i) = ipc(iz(5,i)) rpc(i) =
rpc(iz(6,i)) oper(i) = oper(iz(7,i)) grdx(i) = brdx(iz(8,i)) grdy(i) = brdy(iz(9,i)) grdz(i) = brdz(iz(10,i))

```

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces buildstr from mgsubd.f

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>nlev</i>	
<i>iz</i>	

Definition at line 252 of file [mgsubd.c](#).

8.29.3.15 VEXTERNC void Vc_vec (double * *coef*, double * *uin*, double * *uout*, int * *nx*, int * *ny*, int * *nz*, int * *ipkey*)

Define the nonlinearity (vector version)

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces c_vec from mypde.f

Parameters

<i>coef</i>	
<i>uin</i>	
<i>uout</i>	
<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>ipkey</i>	

Definition at line 116 of file [mypdec.c](#).

8.29.3.16 VEXTERNC void Vc_vecpmg (double * *coef*, double * *uin*, double * *uout*, int * *nx*, int * *ny*, int * *nz*, int * *ipkey*)

Define the nonlinearity (vector version)

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces c_vecpmg from mypde.f

Parameters

<i>coef</i>	
-------------	--

<i>uin</i>	
<i>uout</i>	
<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>ipkey</i>	

Definition at line 128 of file [mypdec.c](#).

8.29.3.17 VEXTERNC void Vc_vecsmpbe (double * *coef*, double * *uin*, double * *uout*, int * *nx*, int * *ny*, int * *nz*, int * *ipkey*)

Define the nonlinearity (vector version)

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces c_vecpmg from mypde.f

Parameters

<i>coef</i>	
<i>uin</i>	
<i>uout</i>	
<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>ipkey</i>	

Definition at line 208 of file [mypdec.c](#).

8.29.3.18 VEXTERNC void Vcghs (int * *nx*, int * *ny*, int * *nz*, int * *ipc*, double * *rpc*, double * *ac*, double * *cc*, double * *fc*, double * *x*, double * *p*, double * *ap*, double * *r*, int * *itmax*, int * *iters*, double * *errtol*, double * *omega*, int * *iresid*, int * *iadjoint*)

A collection of useful low-level routines (timing, etc).

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*

```

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>ac</i>	
<i>cc</i>	
<i>fc</i>	
<i>x</i>	
<i>p</i>	
<i>ap</i>	
<i>r</i>	
<i>itmax</i>	

<i>iters</i>	
<i>errtol</i>	
<i>omega</i>	
<i>iresid</i>	
<i>iadjoint</i>	

Definition at line 52 of file [cgd.c](#).

8.29.3.19 VEXTERNC void Vdc_vec (double * *coef*, double * *uin*, double * *uout*, int * *nx*, int * *ny*, int * *nz*, int * *ipkey*)

Define the derivative of the nonlinearity (vector version)

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces dc_vec from mypde.f

Parameters

<i>coef</i>	
<i>uin</i>	
<i>uout</i>	
<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>ipkey</i>	

Definition at line 336 of file [mypdec.c](#).

8.29.3.20 VEXTERNC void Vdpbsl (double * *abd*, int * *lda*, int * *n*, int * *m*, double * *b*)

LINPACK interface.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
```

```

* 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.
*
*

```

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.

Solves the double precision symmetric positive definite band system $A \cdot X = B$ using the factors computed by `dpbco` or `dpbfa`

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

A division by zero will occur if the input factor contains a zero on the diagonal. Technically this indicates singularity, but it is usually caused by improper subroutine arguments. It will not occur if the subroutines are called correctly and `info == 0`

Replaces `dpbsl` from `mgsbdf.f`

Parameters

<i>abd</i>	The output from <code>dpbco</code> or <code>dpbfa</code>
<i>lda</i>	The leading dimension of the array <code>abd</code>
<i>n</i>	The order of the matrix <code>a</code>
<i>m</i>	The number of diagonals above the main diagonal
<i>b</i>	The right hand side vector

Definition at line 52 of file [mlinpkd.c](#).

8.29.3.21 **VEXTERNC** void Vextrac (int * *nxf*, int * *nyf*, int * *nzf*, int * *nxc*, int * *ny*, int * *nzc*, double * *xin*, double * *xout*)

Simple injection of a fine grid function into coarse grid.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces `extrac` from `matvecd.f`

Parameters

<i>nxf</i>	
<i>nyf</i>	

<i>nzf</i>	
<i>nxc</i>	
<i>ny</i>	
<i>nzc</i>	
<i>xin</i>	
<i>xout</i>	

Definition at line 1073 of file [matvecd.c](#).

8.29.3.22 VEXTERNC void VfboundPMG (int * *ibound*, int * *nx*, int * *ny*, int * *nz*, double * *x*, double * *gxc*, double * *gyc*, double * *gzc*)

Initialize a grid function to have a certain boundary value,.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces fboundPMG from mikpckd.f

Parameters

<i>ibound</i>	
<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>x</i>	
<i>gxc</i>	
<i>gyc</i>	
<i>gzc</i>	

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

8.29.3.23 VEXTERNC void VfboundPMG00 (int * *nx*, int * *ny*, int * *nz*, double * *x*)

Initialize a grid function to have a zero boundary value.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces fboundPMG00 from mikpckd.f

Parameters

<i>nx</i>	The size of the x dimension of the 3d matrix
<i>ny</i>	The size of the y dimension of the 3d matrix
<i>nz</i>	The size of the z dimension of the 3d matrix
<i>x</i>	The 3d matrix to initialize

Definition at line 253 of file [mikpckd.c](#).

8.29.3.24 `VEXTERNC void Vfmvfas (int * nx, int * ny, int * nz, double * x, int * iz, double * w0, double * w1, double * w2, double * w3, double * w4, int * istop, int * itmax, int * iters, int * ierror, int * nlev, int * ilev, int * nlev_real, int * mgsolv, int * iok, int * iinfo, double * epsiln, double * errtol, double * omega, int * nu1, int * nu2, int * mgs moo, int * ipc, double * rpc, double * pc, double * ac, double * cc, double * fc, double * tru)`

Multigrid nonlinear solve iteration routine.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*
```


Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*

```

Nested iteration for a nonlinear multilevel method. Algorithm: nonlinear multigrid iteration (fas)

this routine is the full multigrid front-end for a multigrid v-cycle solver. in other words, at repeatedly calls the v-cycle multigrid solver on successively finer and finer grids.

Note**Author**

Tucker Beck [C Translation], Michael Holst [Original]

Replaces fmvfas from mgfasd.f

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>x</i>	
<i>iz</i>	
<i>w0</i>	
<i>w1</i>	
<i>w2</i>	
<i>w3</i>	
<i>w4</i>	
<i>istop</i>	
<i>itmax</i>	
<i>iters</i>	
<i>ierror</i>	
<i>nlev</i>	
<i>ilev</i>	
<i>nlev_real</i>	
<i>mgstolv</i>	
<i>iok</i>	
<i>iinfo</i>	
<i>epsiln</i>	
<i>errtol</i>	
<i>omega</i>	
<i>nu1</i>	
<i>nu2</i>	
<i>mgsmoo</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>pc</i>	
<i>ac</i>	
<i>cc</i>	
<i>fc</i>	
<i>tru</i>	

Definition at line 52 of file [mgfasd.c](#).

```
8.29.3.25  VPUBLIC void Vfnewton ( int * nx, int * ny, int * nz, double * x, int * iz, double * w0, double * w1, double * w2,
double * w3, int * istop, int * itmax, int * iters, int * ierror, int * nlev, int * ilev, int * nlev_real, int * mgstolv, int * iok,
int * iinfo, double * epsiln, double * errtol, double * omega, int * nu1, int * nu2, int * mgsmoo, double * cprime,
double * rhs, double * xtmp, int * ipc, double * rpc, double * pc, double * ac, double * cc, double * fc, double * tru )
```

Driver routines for the Newton method.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific
* Northwest National Laboratory, operated by Battelle Memorial Institute,
* Pacific Northwest Division for the U.S. Department Energy. Portions
* Copyright (c) 2002-2010, Washington University in St. Louis. Portions
* Copyright (c) 2002-2010, Nathan A. Baker. Portions Copyright (c) 1999-2002,
* The Regents of the University of California. Portions Copyright (c) 1995,
* Michael Holst.
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* - Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of 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.
*
*

```

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*

```

Nested iteration for an inexact-newton-multilevel method.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces `fnewton` from `newtond.f`

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>x</i>	
<i>iz</i>	
<i>w0</i>	
<i>w1</i>	
<i>w2</i>	
<i>w3</i>	
<i>istop</i>	
<i>itmax</i>	
<i>iters</i>	
<i>ierror</i>	

<i>nlev</i>	
<i>ilev</i>	
<i>nlev_real</i>	
<i>mgsolv</i>	
<i>iok</i>	
<i>iinfo</i>	
<i>epsiln</i>	
<i>errtol</i>	
<i>omega</i>	
<i>nu1</i>	
<i>nu2</i>	
<i>mgsmoo</i>	
<i>cprime</i>	
<i>rhs</i>	
<i>xtmp</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>pc</i>	
<i>ac</i>	
<i>cc</i>	
<i>fc</i>	
<i>tru</i>	

Definition at line 58 of file [newtond.c](#).

8.29.3.26 VEXTERNC void Vgetjac (int * *nx*, int * *ny*, int * *nz*, int * *nlev_real*, int * *iz*, int * *lev*, int * *ipkey*, double * *x*, double * *r*, double * *cprime*, double * *rhs*, double * *cc*, double * *pc*)

Form the jacobian system.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces getjac from newtond.f

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>nlev_real</i>	
<i>iz</i>	
<i>lev</i>	
<i>ipkey</i>	

<i>x</i>	
<i>r</i>	
<i>cprime</i>	
<i>rhs</i>	
<i>cc</i>	
<i>pc</i>	

Definition at line 550 of file [newtond.c](#).

8.29.3.27 VEXTERNC void Vgsrb (int * *nx*, int * *ny*, int * *nz*, int * *ipc*, double * *rpc*, double * *ac*, double * *cc*, double * *fc*, double * *x*, double * *w1*, double * *w2*, double * *r*, int * *itmax*, int * *iters*, double * *errtol*, double * *omega*, int * *iresid*, int * *iadjoint*)

Guass-Seidel solver.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*
```

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*
* Call the fast diagonal iterative method.
```

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces gsrp from gsd.f

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>ac</i>	
<i>cc</i>	
<i>fc</i>	
<i>x</i>	
<i>w1</i>	
<i>w2</i>	
<i>r</i>	
<i>itmax</i>	
<i>iters</i>	
<i>errtol</i>	
<i>omega</i>	
<i>iresid</i>	
<i>iadjoint</i>	

Definition at line 52 of file [gsd.c](#).

8.29.3.28 VEXTERNC void VinterpPMG (int * *nxc*, int * *nyc*, int * *nzc*, int * *nx**f*, int * *ny**f*, int * *nz**f*, double * *xin*, double * *xout*, double * *pc*)

Apply the prolongation operator.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces interpPMG from matvecd.f

Parameters

<i>nxc</i>	
<i>nyc</i>	
<i>nzc</i>	
<i>nx</i> <i>f</i>	
<i>ny</i> <i>f</i>	
<i>nz</i> <i>f</i>	
<i>xin</i>	
<i>xout</i>	
<i>pc</i>	

Definition at line 910 of file [matvecd.c](#).

8.29.3.29 VEXTERNC void Vipower (int * nx, int * ny, int * nz, double * u, int * iz, double * w0, double * w1, double * w2, double * w3, double * w4, double * eigmin, double * eigmin_model, double * tol, int * itmax, int * iters, int * nlev, int * ilev, int * nlev_real, int * mgsolv, int * iok, int * iinfo, double * epsiln, double * errtol, double * omega, int * nu1, int * nu2, int * mgs moo, int * ipc, double * rpc, double * pc, double * ac, double * cc, double * tru)

Standard inverse power method for minimum eigenvalue estimation.

Note

To test, note that the 3d laplacean has min/max eigenvalues:

```
lambda_min = 6 - 2*dcos(pi/(nx-1))
              - 2*dcos(pi/(ny-1))
              - 2*dcos(pi/(nz-1))

lambda_max = 6 - 2*dcos((nx-2)*pi/(nx-1))
              - 2*dcos((ny-2)*pi/(ny-1))
              - 2*dcos((nz-2)*pi/(nz-1))
```

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces ipower from powerd.f

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>u</i>	
<i>iz</i>	
<i>w0</i>	
<i>w1</i>	
<i>w2</i>	
<i>w3</i>	
<i>w4</i>	
<i>eigmin</i>	
<i>eigmin_model</i>	
<i>tol</i>	
<i>itmax</i>	
<i>iters</i>	
<i>nlev</i>	
<i>ilev</i>	
<i>nlev_real</i>	
<i>mgsolv</i>	

<i>iok</i>	
<i>iinfo</i>	
<i>epsiln</i>	
<i>errtol</i>	
<i>omega</i>	
<i>nu1</i>	
<i>nu2</i>	
<i>mgsmoo</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>pc</i>	
<i>ac</i>	
<i>cc</i>	
<i>tru</i>	

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

8.29.3.30 VEXTERNC void Vmatvec (int * *nx*, int * *ny*, int * *nz*, int * *ipc*, double * *rpc*, double * *ac*, double * *cc*, double * *x*, double * *y*)

Matrix-vector multiplication routines.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*

```

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*

```

Break the matrix data-structure into diagonals and then call the matrix-vector routine.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces matvec from matvecd.f

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>ac</i>	
<i>cc</i>	
<i>x</i>	
<i>y</i>	

Definition at line 52 of file [matvecd.c](#).

8.29.331 `VEXTERNC void Vmgdriv (int * iparm, double * rparm, int * iwork, double * rwork, double * u, double * xf, double * yf, double * zf, double * gxcf, double * gycf, double * gzcf, double * a1cf, double * a2cf, double * a3cf, double * ccf, double * fcf, double * tcf)`

Multilevel solver driver.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*

```

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*

```

Author

Tucker Beck [C Translation], Michael Holst [Original]

Replaces mgdriv from mgdrvd.f

Parameters

<i>iparm</i>	
<i>rparm</i>	
<i>iwork</i>	
<i>rwork</i>	
<i>u</i>	
<i>xf</i>	
<i>yf</i>	
<i>zf</i>	
<i>gxcf</i>	
<i>gycf</i>	
<i>gzcf</i>	
<i>a1cf</i>	
<i>a2cf</i>	
<i>a3cf</i>	
<i>ccf</i>	
<i>fcf</i>	
<i>tcf</i>	

Definition at line 52 of file [mgdrvd.c](#).

8.29.3.32 VEXTERNC void Vmgdriv2 (int * *iparm*, double * *rparm*, int * *nx*, int * *ny*, int * *nz*, double * *u*, int * *iz*, int * *ipc*, double * *rpc*, double * *pc*, double * *ac*, double * *cc*, double * *fc*, double * *xf*, double * *yf*, double * *zf*, double * *gxcf*, double * *gycf*, double * *gzcf*, double * *a1cf*, double * *a2cf*, double * *a3cf*, double * *ccf*, double * *fcf*, double * *tcf*)

Solves the pde using the multi-grid method.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Replaces mgdriv2 from mgdrvd.f

This routine uses a multigrid method to solve the following three-dimensional, 2nd order elliptic partial differential equation:

```
lu = f, u in omega
u = g, u on boundary of omega
```

where

```
omega = [xmin,xmax]x[ymin,ymax]x[zmin,zmax]
```

the multigrid code requires the operator in the form:

```
- \nabla \cdot (a \nabla u) + c(u) = f
```

with

$a(x,y,z), f(x,y,z)$, scalar functions (possibly discontinuous) on ω . (discontinuities must be along fine grid lines). boundary function $g(x,y,z)$ is smooth on boundary of ω .

the function $c(u)$ is a possibly nonlinear function of the unknown u , and varies (possibly discontinuously) with the spatial position also.

user inputs:

the user must provide the coefficients of the differential operator, some initial parameter settings in an integer and a real parameter array, and various work arrays.

Parameters

<i>iparm</i>	
<i>rparm</i>	
<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>u</i>	
<i>iz</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>pc</i>	
<i>ac</i>	
<i>cc</i>	
<i>fc</i>	
<i>xf</i>	
<i>yf</i>	
<i>zf</i>	
<i>gxcf</i>	
<i>gycf</i>	
<i>gzcf</i>	
<i>a1cf</i>	
<i>a2cf</i>	
<i>a3cf</i>	
<i>ccf</i>	
<i>fcf</i>	
<i>tcf</i>	

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

8.29.3.33 VEXTERNC void *Vmgsz* (int * *mgcoar*, int * *mgdisc*, int * *mgsolv*, int * *nx*, int * *ny*, int * *nz*, int * *nlev*, int * *nxc*, int * *nyc*, int * *nzc*, int * *nf*, int * *nc*, int * *narr*, int * *narrc*, int * *n_rpc*, int * *n_iz*, int * *n_ipc*, int * *iretot*, int * *iintot*)

This routine computes the required sizes of the real and integer work arrays for the multigrid code. these two sizes are a (complicated) function of input parameters.

The work arrays must have been declared in the calling program as:

```
double precision rwork(iretot)
integer          iwork(iintot)
```

where:

```

iretot   = function_of(mgcoar,mgdisc,mgstlv,nx,ny,nz,nlev)
iintot   = function_of(mgcoar,mgdisc,mgstlv,nx,ny,nz,nlev)

mgcoar   = coarsening technique:
           0=standard discretization
           1=averaged coefficient + standard discretization
           2=algebraic galerkin coarsening

mgdisc   = discretization technique:
           0=box method
           1=fem method

mgstlv   = coarse grid solver:
           0=conjugate gradients
           1=symmetric banded linpack solver

nx,ny,nz = grid dimensions in each direction,
           including boundary points

nlev     = the number of multigrid levels desired for the
           method.

```

other parameters:

```

nf       = number of unknowns on the finest mesh
          = nx * ny * nz

nc       = number of unknowns on the coarsest mesh

narr     = storage for one vector on all the meshes

narrc    = storage for one vector on all the meshes but the finest

```

the work arrays rwork and iwork will be chopped into smaller pieces according to:

```

double precision ac(STORE)          (system operators on all levels)
double precision pc(27*narrc)       (prol. ops for coarse levels)
double precision cc(narr),fc(narr)  (helmholtz term, rhs -- all levels)
double precision rpc(100*(nlev+1))  (real info for all levels)
integer          ipc(100*(nlev+1))  (integer info for all levels)
integer          iz(50,nlev+1),     (pointers into ac,pc,cc,fc,etc.)

```

where STORE depends on the discretization, coarsening, and coarse grid solver:

```

STORE = 4*nf + 4*narrc + NBAND*nc (mgdisc=box, mgcoar=stan/harm)
       or = 4*nf + 14*narrc + NBAND*nc (mgdisc=box, mgcoar=gal)
       or = 14*nf + 14*narrc + NBAND*nc (mgdisc=fem, mgcoar=stan/harm/gal)

NBAND = 0 (mgsolv=iterative)
       or = 1+(nxc-2)*(nyc-2) (mgsolv=7-pt banded linpack)
       or = 1+(nxc-2)*(nyc-2)+(nxc-2)+1 (mgsolv=27-pt banded linpack)

```

Author

Tucker Beck [C Translation], Michael Holst [Original]

Replaces mgsz from mgdrvd.f

Parameters

<i>mgcoar</i>	
<i>mgdisc</i>	
<i>mgsovl</i>	
<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>nlev</i>	
<i>nxc</i>	
<i>nyc</i>	
<i>nzc</i>	
<i>nf</i>	
<i>nc</i>	
<i>narr</i>	
<i>narrc</i>	
<i>n_rpc</i>	
<i>n_iz</i>	
<i>n_ipc</i>	
<i>iretot</i>	
<i>iintot</i>	

Definition at line 557 of file [mgdrvd.c](#).

8.29.34 VEXTERNC void Vmresid (int * *nx*, int * *ny*, int * *nz*, int * *ipc*, double * *rpc*, double * *ac*, double * *cc*, double * *fc*, double * *x*, double * *r*)

Break the matrix data-structure into diagonals and then call the residual routine.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces mresid from matvecd.f

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>ac</i>	
<i>cc</i>	
<i>fc</i>	
<i>x</i>	

<i>r</i>	
----------	--

Definition at line 421 of file [matvecd.c](#).

```
8.29.3.35  VEXTERNC void Vmvs ( int * nx, int * ny, int * nz, double * x, int * iz, double * w0, double * w1, double * w2, double
* w3, int * istop, int * itmax, int * iters, int * ierror, int * nlev, int * ilev, int * nlev_real, int * mgsolv, int * iok, int *
iinfo, double * epsiln, double * ertol, double * omega, int * nu1, int * nu2, int * mgsmoo, int * ipc, double * rpc,
double * pc, double * ac, double * cc, double * fc, double * tru )
```

MG helper functions.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*
```

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*
Screaming linear multilevel method.

```

algorithm: linear multigrid iteration (cs)

multigrid v-cycle solver.

input: (1) fine and coarse grid discrete linear operators: L_h, L_H (2) fine grid source function: f_h (3) fine grid approximate solution: u_h

output: (1) fine grid improved solution: u_h

the two-grid algorithm is: (1) pre-smooth: $u1_h = \text{smooth}(L_h, f_h, u_h)$ (2) restrict defect: $d_H = r(L_h(u1_h) - f_h)$ (3) solve for correction: $c_H = L_H^{-1}(d_H)$ (4) prolongate and correct: $u2_h = u1_h - p(c_H)$ (5) post-smooth: $u_h = \text{smooth}(L_h, f_h, u2_h)$

(of course, c_H is determined with another two-grid algorithm)

implementation notes: (0) "u1_h" must be kept on each level until "c_H" is computed, and then both are used to compute "u2_h". (1) "u_h" (and then "u1_h") on all levels is stored in the "x" array. (2) "d_H" is identically "f_h" for f_h on the next coarser grid. (3) "c_h" is identically "u_h" for u_h on the next coarser grid. (4) "d_H" is stored in the "r" array (must be kept for post-smooth). (5) "f_h" is stored in the "fc" array. (6) "L_h" on all levels is stored in the "ac" array. (7) signs may be reversed; i.e., residual is used in place of the defect in places, etc.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces mvcs from mgcsd.f

New grid size

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>x</i>	
<i>iz</i>	
<i>w0</i>	
<i>w1</i>	
<i>w2</i>	
<i>w3</i>	
<i>istop</i>	
<i>itmax</i>	
<i>iters</i>	
<i>ierror</i>	
<i>nlev</i>	
<i>ilev</i>	
<i>nlev_real</i>	
<i>mgssolv</i>	
<i>iok</i>	
<i>iinfo</i>	
<i>epsiln</i>	
<i>errtol</i>	
<i>omega</i>	
<i>nu1</i>	
<i>nu2</i>	
<i>mgsmoo</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>pc</i>	

<i>ac</i>	
<i>cc</i>	
<i>fc</i>	
<i>tru</i>	

Definition at line 52 of file [mgcsd.c](#).

8.29.3.36 VEXTERNC void Vmvas (int * *nx*, int * *ny*, int * *nz*, double * *x*, int * *iz*, double * *w0*, double * *w1*, double * *w2*, double * *w3*, double * *w4*, int * *istop*, int * *itmax*, int * *iters*, int * *ierror*, int * *nlev*, int * *ilev*, int * *nlev_real*, int * *mgsovl*, int * *iok*, int * *iinfo*, double * *epsiln*, double * *errtol*, double * *omega*, int * *nu1*, int * *nu2*, int * *mgsmoo*, int * *ipc*, double * *rpc*, double * *pc*, double * *ac*, double * *cc*, double * *fc*, double * *tru*)

Nonlinear multilevel method.

Note

Replaces mvfas from mgfasd.f

Author

Tucker Beck [C Translation], Michael Holst [Original]

Algorithm: nonlinear multigrid iteration (fas)

multigrid v-cycle solver.

input: (1) fine and coarse grid discrete nonlinear operators: *L_h*, *L_H* (2) fine grid source function: *f_h* (3) fine grid approximate solution: *u_h*

output: (1) fine grid improved solution: *u_h*

the two-grid algorithm is: (1) pre-smooth: $u1_h = \text{smooth}(L_h, f_h, u_h)$ (2) restrict defect: $d_H = r(L_h(u1_h) - f_h)$ restrict solution: $u_H = r(u1_h)$ (3) form coarse grid rhs: $f_H = L_H(u_H) - d_H$ solve for correction: $c_H = L_H^{-1}(f_H)$ (4) prolongate and correct: $u2_h = u1_h - p(c_H - u_H)$ (5) post-smooth: $u_h = \text{smooth}(L_h, f_h, u2_h)$

(of course, *c_H* is determined with another two-grid algorithm)

implementation notes: (0) "*u1_h*" and "*u_H*" must be kept on each level until "*c_H*" is computed, and then all three are used to compute "*u2_h*". (1) "*u_h*" (and then "*u1_h*") on all levels is stored in the "*x*" array. (2) "*u_H*" on all levels is stored in the "*e*" array. (3) "*c_h*" is identically "*u_h*" for *u_h* on the next coarser grid. (4) "*d_H*" is stored in the "*r*" array. (5) "*f_h*" and "*f_H*" are stored in the "*fc*" array. (6) "*L_h*" on all levels is stored in the "*ac*" array. (7) signs may be reversed; i.e., residual is used in place of the defect in places, etc.

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>x</i>	
<i>iz</i>	
<i>w0</i>	
<i>w1</i>	

<i>w2</i>	
<i>w3</i>	
<i>w4</i>	
<i>istop</i>	
<i>itmax</i>	
<i>iters</i>	
<i>ierror</i>	
<i>nlev</i>	
<i>ilev</i>	
<i>nlev_real</i>	
<i>mgsovl</i>	
<i>iok</i>	
<i>iinfo</i>	
<i>epsiln</i>	
<i>errtol</i>	
<i>omega</i>	
<i>nu1</i>	
<i>nu2</i>	
<i>mgsmoo</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>pc</i>	
<i>ac</i>	
<i>cc</i>	
<i>fc</i>	
<i>tru</i>	

Definition at line 152 of file [mgfasd.c](#).

8.29.3.37 VEXTERNC void Vmypdefinitlpbe (int * *tnion*, double * *tcharge*, double * *tsconc*)

Set up the ionic species to be used in later calculations. This must be called before any other of the routines in this file.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*

```

Author

Tucker Beck [C Translation], Nathan Baker [Original]

Note

Replaces mypdefinitlpbe from mypde.f

Parameters

<i>tnion</i>	The number if ionic species
<i>tcharge</i>	The charge in electrons
<i>tsconc</i>	Prefactor for conterion Boltzmann distribution terms. Basically a scaled concentration $-(ion\ concentration/bulkIonicStrength)/2$

Definition at line 52 of file [mypdec.c](#).

8.29.3.38 VEXTERNC void Vmypdefinitnpbe (int * *tnion*, double * *tcharge*, double * *tsconc*)

Set up the ionic species to be used in later calculations. This must be called before any other of the routines in this file.

Author

Tucker Beck [C Translation], Nathan Baker [Original]

Note

Replaces mypdefinitnpbe from mypde.f

Parameters

<i>tnion</i>	The number if ionic species
<i>tcharge</i>	The charge in electrons
<i>tsconc</i>	Prefactor for conterion Boltzmann distribution terms. Basically a scaled concentration -(ion concentration/bulkIonicStrength)/2

Definition at line 70 of file [mypdec.c](#).

8.29.3.39 VEXTERNC void Vmypdefinitmpbe (int * *tnion*, double * *tcharge*, double * *tsconc*, double * *smvolume*, double * *smsize*)

Set up the ionic species to be used in later calculations. This must be called before any other of the routines in this file.

Author

Tucker Beck [C Translation], Nathan Baker [Original]

Note

Replaces mypdefinitmpbe from mypde.f

Parameters

<i>tnion</i>	The number if ionic species
<i>tcharge</i>	The charge in electrons
<i>tsconc</i>	Prefactor for conterion Boltzmann distribution terms. Basically a scaled concentration -(ion concentration/bulkIonicStrength)/2
<i>smvolume</i>	
<i>smsize</i>	

Definition at line 88 of file [mypdec.c](#).

8.29.3.40 VEXTERNC void Vnewdriv (int * *iparm*, double * *rparm*, int * *iwork*, double * *rwork*, double * *u*, double * *xf*, double * *yf*, double * *zf*, double * *gxcf*, double * *gycf*, double * *gzcf*, double * *a1cf*, double * *a2cf*, double * *a3cf*, double * *ccf*, double * *fcf*, double * *tcf*)

Driver for the Newton Solver.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*

```

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*

```

Driver for a screaming inexact-newton-multilevel solver.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces newdriv from newdrvd.f

Parameters

<i>iparm</i>	
<i>rparm</i>	
<i>iwork</i>	
<i>rwork</i>	
<i>u</i>	
<i>xf</i>	
<i>yf</i>	
<i>zf</i>	
<i>gxcf</i>	
<i>gycf</i>	
<i>gzcf</i>	
<i>a1cf</i>	
<i>a2cf</i>	
<i>a3cf</i>	
<i>ccf</i>	
<i>fcf</i>	
<i>tcf</i>	

Definition at line 52 of file [newdrvd.c](#).

8.29.3.41 VEXTERNC void Vnewdriv2 (int * *iparm*, double * *rparm*, int * *nx*, int * *ny*, int * *nz*, double * *u*, int * *iz*, double * *w1*, double * *w2*, int * *ipc*, double * *rpc*, double * *pc*, double * *ac*, double * *cc*, double * *fc*, double * *xf*, double * *yf*, double * *zf*, double * *gxcf*, double * *gycf*, double * *gzcf*, double * *a1cf*, double * *a2cf*, double * *a3cf*, double * *ccf*, double * *fcf*, double * *tcf*)

Solves using Newton's Method.

Author

Tucker Beck [C Translation], Michael Holst [Original]

This routine uses a newton's method, combined with a linear multigrid iteration, to solve the following three-dimensional, 2nd order elliptic partial differential equation:

$$\begin{aligned} \Delta u &= f, \quad u \text{ in } \omega \\ u &= g, \quad u \text{ on boundary of } \omega \end{aligned}$$

where

$$\omega = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [z_{\min}, z_{\max}]$$

the multigrid code requires the operator in the form:

$$-\nabla \cdot (a \nabla u) + c(u) = f$$

with

$a(x,y,z), f(x,y,z)$, scalar functions (possibly discontinuous) on ω . (discontinuities must be along fine grid lines).
boundary function $g(x,y,z)$ is smooth on boundary of ω .

the function $c(u)$ is a possibly nonlinear function of the unknown u , and varies (possibly discontinuously) with the spatial position also.

User inputs:

the user must provide the coefficients of the differential operator, some initial parameter settings in an integer and a real parameter array, and various work arrays.

Note

Replaces newdriv2 from newdrvd.f

Parameters

<i>iparm</i>	
<i>rparm</i>	
<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>u</i>	
<i>iz</i>	
<i>w1</i>	
<i>w2</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>pc</i>	

<i>ac</i>	
<i>cc</i>	
<i>fc</i>	
<i>xf</i>	
<i>yf</i>	
<i>zf</i>	
<i>gxcf</i>	
<i>gycf</i>	
<i>gzcf</i>	
<i>a1cf</i>	
<i>a2cf</i>	
<i>a3cf</i>	
<i>ccf</i>	
<i>fcf</i>	
<i>tcf</i>	

Definition at line 164 of file [newdrvd.c](#).

8.29.3.42 VEXTERNC void Vnewton (int * *nx*, int * *ny*, int * *nz*, double * *x*, int * *iz*, double * *w0*, double * *w1*, double * *w2*, double * *w3*, int * *istop*, int * *itmax*, int * *iters*, int * *ierror*, int * *nlev*, int * *ilev*, int * *nlev_real*, int * *mgsovl*, int * *iok*, int * *iinfo*, double * *epsiln*, double * *errtol*, double * *omega*, int * *nu1*, int * *nu2*, int * *mgsmoo*, double * *cprime*, double * *rhs*, double * *xtmp*, int * *ipc*, double * *rpc*, double * *pc*, double * *ac*, double * *cc*, double * *fc*, double * *tru*)

Inexact-newton-multilevel method.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces newton from newtond.f

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>x</i>	
<i>iz</i>	
<i>w0</i>	
<i>w1</i>	
<i>w2</i>	
<i>w3</i>	
<i>istop</i>	
<i>itmax</i>	
<i>iters</i>	

<i>ierror</i>	
<i>nlev</i>	
<i>ilev</i>	
<i>nlev_real</i>	
<i>mgsolv</i>	
<i>iok</i>	
<i>iinfo</i>	
<i>epsiln</i>	
<i>errtol</i>	
<i>omega</i>	
<i>nu1</i>	
<i>nu2</i>	
<i>mgsmoo</i>	
<i>cprime</i>	
<i>rhs</i>	
<i>xtmp</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>pc</i>	
<i>ac</i>	
<i>cc</i>	
<i>fc</i>	
<i>tru</i>	

Definition at line 162 of file [newtond.c](#).

8.29.3.43 VEXTERNC void Vnmatvec (int * *nx*, int * *ny*, int * *nz*, int * *ipc*, double * *rpc*, double * *ac*, double * *cc*, double * *x*, double * *y*, double * *w1*)

Break the matrix data-structure into diagonals and then call the matrix-vector routine.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces nmatvec from matvecd.f

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>ac</i>	

<i>cc</i>	
<i>x</i>	
<i>y</i>	
<i>w1</i>	

Definition at line 227 of file [matvecd.c](#).

8.29.3.44 VEXTERNC void Vnmresid (int * *nx*, int * *ny*, int * *nz*, int * *ipc*, double * *rpc*, double * *ac*, double * *cc*, double * *fc*, double * *x*, double * *r*, double * *w1*)

Break the matrix data-structure into diagonals and then call the residual routine.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces nmresid from matvecd.f

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>ac</i>	
<i>cc</i>	
<i>fc</i>	
<i>x</i>	
<i>r</i>	
<i>w1</i>	

Definition at line 593 of file [matvecd.c](#).

8.29.3.45 VEXTERNC void Vnsmooth (int * *nx*, int * *ny*, int * *nz*, int * *ipc*, double * *rpc*, double * *ac*, double * *cc*, double * *fc*, double * *x*, double * *w1*, double * *w2*, double * *r*, int * *imax*, int * *iters*, double * *errtol*, double * *omega*, int * *iresid*, int * *iadjoint*, int * *meth*)

call the appropriate non-linear smoothing routine.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces nsmooth from nsmoothd.f

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>ac</i>	
<i>cc</i>	
<i>fc</i>	
<i>x</i>	
<i>w1</i>	
<i>w2</i>	
<i>r</i>	
<i>itmax</i>	
<i>iters</i>	
<i>errtol</i>	
<i>omega</i>	
<i>iresid</i>	
<i>iadjoint</i>	
<i>meth</i>	

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

8.29.3.46 VEXTERNC void Vpower (int * *nx*, int * *ny*, int * *nz*, int * *iz*, int * *ilev*, int * *ipc*, double * *rpc*, double * *ac*, double * *cc*, double * *w1*, double * *w2*, double * *w3*, double * *w4*, double * *eigmax*, double * *eigmax_model*, double * *tol*, int * *itmax*, int * *iters*, int * *iinfo*)

Power methods for eigenvalue estimation.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*

```

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.

```



```

*
*
Standard power method for maximum eigenvalue estimation of a matrix c* c*

```

Note

To test, note that the 3d laplacean has min/max eigenvalues: $c* c* \lambda_{\min} = 6 - 2*\text{dcos}(\pi/(nx-1)) c* - 2*\text{dcos}(\pi/(ny-1)) c* - 2*\text{dcos}(\pi/(nz-1)) c*$ $c* c* \lambda_{\max} = 6 - 2*\text{dcos}((nx-2)*\pi/(nx-1)) c* - 2*\text{dcos}((ny-2)*\pi/(ny-1)) c* - 2*\text{dcos}((nz-2)*\pi/(nz-1))$

```
@author Tucker Beck [C Translation], Michael Holst [Original]
```

```
@note Replaces power from powerd.f
```

```
@note Vpower is yet untested as a call stack including it hasn't been found
```

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>iz</i>	
<i>ilev</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>ac</i>	
<i>cc</i>	
<i>w1</i>	
<i>w2</i>	
<i>w3</i>	
<i>w4</i>	
<i>eigmax</i>	
<i>eigmax_model</i>	
<i>tol</i>	
<i>itmax</i>	
<i>iters</i>	
<i>iinfo</i>	

Definition at line 52 of file [powerd.c](#).

8.29.3.47 VEXTERNC void Vprtmtd (int * *nx*, int * *ny*, int * *nz*, int * *ipc*, double * *rpc*, double * *ac*)

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces prtmtd from mikpckd.f

Parameters

<i>nx</i>	The size of the x dimension of the 3d matrix
<i>ny</i>	The size of the y dimension of the 3d matrix
<i>nz</i>	The size of the z dimension of the 3d matrix
<i>ipc</i>	Integer parameters
<i>rpc</i>	Double parameters
<i>ac</i>	

Definition at line 327 of file [mikpckd.c](#).

8.29.3.48 VEXTERNC void Vrestrc (int * *nx*, int * *ny*, int * *nz*, int * *nxc*, int * *nyc*, int * *nzc*, double * *xin*, double * *xout*, double * *pc*)

Apply the restriction operator.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces restrc from matvecd.f

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>nxc</i>	
<i>nyc</i>	
<i>nzc</i>	
<i>xin</i>	
<i>xout</i>	
<i>pc</i>	

Definition at line 777 of file [matvecd.c](#).

8.29.3.49 VEXTERNC void Vsmooth (int * *nx*, int * *ny*, int * *nz*, int * *ipc*, double * *rpc*, double * *ac*, double * *cc*, double * *fc*, double * *x*, double * *w1*, double * *w2*, double * *r*, int * *itmax*, int * *iters*, double * *errtol*, double * *omega*, int * *iresid*, int * *iadjoin*, int * *meth*)

Multigrid smoothing functions.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*
```

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*

```

call the appropriate linear smoothing routine.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces smooth from smoothd.f

Parameters

<i>nx</i>	
<i>ny</i>	
<i>nz</i>	
<i>ipc</i>	
<i>rpc</i>	
<i>ac</i>	
<i>cc</i>	
<i>fc</i>	
<i>x</i>	
<i>w1</i>	
<i>w2</i>	
<i>r</i>	
<i>itmax</i>	
<i>iters</i>	
<i>errtol</i>	
<i>omega</i>	
<i>iresid</i>	

<i>iadjoint</i>	
<i>meth</i>	

Definition at line 52 of file [smoothd.c](#).

8.29.3.50 VEXTERNC void Vxaxpy (int * *nx*, int * *ny*, int * *nz*, double * *alpha*, double * *x*, double * *y*)

saxpy operation for a grid function with boundary values.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces xaxpy from mikpckd.f

Parameters

<i>nx</i>	The size of the x dimension of the 3d matrix
<i>ny</i>	The size of the y dimension of the 3d matrix
<i>nz</i>	The size of the z dimension of the 3d matrix
<i>alpha</i>	
<i>x</i>	The source matrix from which to copy data
<i>y</i>	The destination matrix to receive copied data

Definition at line 107 of file [mikpckd.c](#).

8.29.3.51 VEXTERNC void Vxcopy (int * *nx*, int * *ny*, int * *nz*, double * *x*, double * *y*)

A collection of useful low-level routines (timing, etc).

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*

```

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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.
*
*

```

Copy operation for a grid function with boundary values. Quite simply copies one 3d matrix to another

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces xcopy from mikpckd.f

Parameters

<i>nx</i>	The size of the x dimension of the 3d matrix
<i>ny</i>	The size of the y dimension of the 3d matrix
<i>nz</i>	The size of the z dimension of the 3d matrix
<i>x</i>	The source matrix from which to copy data
<i>y</i>	The destination matrix to receive copied data

Definition at line 52 of file [mikpckd.c](#).

8.29.3.52 VEXTERNC void Vxcopy_large (int * *nx*, int * *ny*, int * *nz*, double * *x*, double * *y*)

Copy operation for a grid function with boundary values. Quite simply copies one 3d matrix to another.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces xcopy_large from mikpckd.f

Note

This function is exactly equivalent to calling xcopy_small with the matrix arguments reversed.

Parameters

<i>nx</i>	The size of the x dimension of the 3d matrix
-----------	--

<i>ny</i>	The size of the y dimension of the 3d matrix
<i>nz</i>	The size of the z dimension of the 3d matrix
<i>x</i>	The source matrix from which to copy data
<i>y</i>	The destination matrix to receive copied data

Definition at line 86 of file [mikpckd.c](#).

8.29.3.53 VEXTERNC void Vxcopy_small (int * nx, int * ny, int * nz, double * x, double * y)

Copy operation for a grid function with boundary values. Quite simply copies one 3d matrix to another.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces xcopy_small from mikpckd.f

Parameters

<i>nx</i>	The size of the x dimension of the 3d matrix
<i>ny</i>	The size of the y dimension of the 3d matrix
<i>nz</i>	The size of the z dimension of the 3d matrix
<i>x</i>	The source matrix from which to copy data
<i>y</i>	The destination matrix to receive copied data

Definition at line 70 of file [mikpckd.c](#).

8.29.3.54 VEXTERNC double Vxdot (int * nx, int * ny, int * nz, double * x, double * y)

Inner product operation for a grid function with boundary values.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces xdot from mikpckd.f

Parameters

<i>nx</i>	The size of the x dimension of the 3d matrix
<i>ny</i>	The size of the y dimension of the 3d matrix
<i>nz</i>	The size of the z dimension of the 3d matrix
<i>x</i>	The first vector
<i>y</i>	The second vector

Definition at line 168 of file [mikpckd.c](#).

8.29.3.55 VEXTERNC double Vxnm1 (int * nx, int * ny, int * nz, double * x)

Norm operation for a grid function with boundary values.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces xnm1 from mikpckd.f

< Accumulates the calculated normal value

Parameters

<i>nx</i>	The size of the x dimension of the 3d matrix
<i>ny</i>	The size of the y dimension of the 3d matrix
<i>nz</i>	The size of the z dimension of the 3d matrix
<i>x</i>	The matrix to normalize

Definition at line 126 of file [mikpckd.c](#).

8.29.3.56 VEXTERNC double Vxnm2 (int * nx, int * ny, int * nz, double * x)

Norm operation for a grid function with boundary values.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces xnm2 from mikpckd.f

< Accumulates the calculated normal value

Parameters

<i>nx</i>	The size of the x dimension of the 3d matrix
<i>ny</i>	The size of the y dimension of the 3d matrix
<i>nz</i>	The size of the z dimension of the 3d matrix
<i>x</i>	The matrix to normalize

Definition at line 147 of file [mikpckd.c](#).

8.29.3.57 VEXTERNC void Vxscal (int * nx, int * ny, int * nz, double * fac, double * x)

Scale operation for a grid function with boundary values.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces xscal from mikpckd.f

Parameters

<i>nx</i>	The size of the x dimension of the 3d matrix
<i>ny</i>	The size of the y dimension of the 3d matrix
<i>nz</i>	The size of the z dimension of the 3d matrix
<i>fac</i>	The scaling factor
<i>x</i>	The 3d matrix to scale

Definition at line 313 of file [mikpckd.c](#).

8.30 High-level front-end routines

Files

- file [apbs.h](#)
Header file for header dependencies.
- file [main.c](#)
APBS "front end" program using formatted input files.
- file [routines.h](#)
Header file for front end auxiliary routines.

Data Structures

- struct [AtomForce](#)
Structure to hold atomic forces.

Macros

- `#define APBSRC 13`
Return code for APBS during failure.

Typedefs

- typedef struct [AtomForce](#) [AtomForce](#)
Define [AtomForce](#) type.

Functions

- int [main](#) (int argc, char **argv)
The main APBS function.
- VPUBLIC Vrc_Codes [initFE](#) (int icalc, [NOsh](#) *nosh, [FEMparm](#) *feparm, [PBEparm](#) *pbeparm, [Vpbe](#) *pbe[[NOSH_MAXCALC](#)], [Valist](#) *alist[[NOSH_MAXMOL](#)], [Vfetk](#) *fetk[[NOSH_MAXCALC](#)])
Initialize FE solver objects.
- VEXTERNC [Vparam](#) * [loadParameter](#) ([NOsh](#) *nosh)
Loads and returns parameter object.
- VEXTERNC int [loadMolecules](#) ([NOsh](#) *nosh, [Vparam](#) *param, [Valist](#) *alist[[NOSH_MAXMOL](#)])
Load the molecules given in NOsh into atom lists.
- VEXTERNC void [killMolecules](#) ([NOsh](#) *nosh, [Valist](#) *alist[[NOSH_MAXMOL](#)])
Destroy the loaded molecules.
- VEXTERNC int [loadDielMaps](#) ([NOsh](#) *nosh, [Vgrid](#) *dielXMap[[NOSH_MAXMOL](#)], [Vgrid](#) *dielYMap[[NOSH_MAXMOL](#)], [Vgrid](#) *dielZMap[[NOSH_MAXMOL](#)])
Load the dielectric maps given in NOsh into grid objects.
- VEXTERNC void [killDielMaps](#) ([NOsh](#) *nosh, [Vgrid](#) *dielXMap[[NOSH_MAXMOL](#)], [Vgrid](#) *dielYMap[[NOSH_MAXMOL](#)], [Vgrid](#) *dielZMap[[NOSH_MAXMOL](#)])
Destroy the loaded dielectric.
- VEXTERNC int [loadKappaMaps](#) ([NOsh](#) *nosh, [Vgrid](#) *kappa[[NOSH_MAXMOL](#)])

- Load the kappa maps given in NOsh into grid objects.*

 - VEXTERNC void `killKappaMaps` (NOsh *nosh, Vgrid *kappa[NOSH_MAXMOL])

Destroy the loaded kappa maps.

 - VEXTERNC int `loadPotMaps` (NOsh *nosh, Vgrid *pot[NOSH_MAXMOL])

Load the potential maps given in NOsh into grid objects.

 - VEXTERNC void `killPotMaps` (NOsh *nosh, Vgrid *pot[NOSH_MAXMOL])

Destroy the loaded potential maps.

 - VEXTERNC int `loadChargeMaps` (NOsh *nosh, Vgrid *charge[NOSH_MAXMOL])

Load the charge maps given in NOsh into grid objects.

 - VEXTERNC void `killChargeMaps` (NOsh *nosh, Vgrid *charge[NOSH_MAXMOL])

Destroy the loaded charge maps.

 - VEXTERNC void `printPBEPARM` (PBEParm *pbeparm)

Print out generic PBE params loaded from input.

 - VEXTERNC void `printMGPARAM` (MGparm *mgparm, double realCenter[3])

Print out MG-specific params loaded from input.

 - VEXTERNC int `initMG` (int icalc, NOsh *nosh, MGparm *mgparm, PBEParm *pbeparm, double realCenter[3], Vpbe *pbe[NOSH_MAXCALC], Valist *alist[NOSH_MAXMOL], Vgrid *dielXMap[NOSH_MAXMOL], Vgrid *dielYMap[NOSH_MAXMOL], Vgrid *dielZMap[NOSH_MAXMOL], Vgrid *kappaMap[NOSH_MAXMOL], Vgrid *chargeMap[NOSH_MAXMOL], Vpmgp *pmgp[NOSH_MAXCALC], Vpmg *pmg[NOSH_MAXCALC], Vgrid *potMap[NOSH_MAXMOL])

Initialize an MG calculation.

 - VEXTERNC void `killMG` (NOsh *nosh, Vpbe *pbe[NOSH_MAXCALC], Vpmgp *pmgp[NOSH_MAXCALC], Vpmg *pmg[NOSH_MAXCALC])

Kill structures initialized during an MG calculation.

 - VEXTERNC int `solveMG` (NOsh *nosh, Vpmg *pmg, MGparm_CalcType type)

Solve the PBE with MG.

 - VEXTERNC int `setPartMG` (NOsh *nosh, MGparm *mgparm, Vpmg *pmg)

Set MG partitions for calculating observables and performing I/O.

 - VEXTERNC int `energyMG` (NOsh *nosh, int icalc, Vpmg *pmg, int *nenergy, double *totEnergy, double *qfEnergy, double *qmEnergy, double *dielEnergy)

Calculate electrostatic energies from MG solution.

 - VEXTERNC void `killEnergy` ()

Kill arrays allocated for energies.

 - VEXTERNC int `forceMG` (Vmem *mem, NOsh *nosh, PBEParm *pbeparm, MGparm *mgparm, Vpmg *pmg, int *nforce, AtomForce **atomForce, Valist *alist[NOSH_MAXMOL])

Calculate forces from MG solution.

 - VEXTERNC void `killForce` (Vmem *mem, NOsh *nosh, int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC])

Free memory from MG force calculation.

 - VEXTERNC void `storeAtomEnergy` (Vpmg *pmg, int icalc, double **atomEnergy, int *nenergy)

Store energy in arrays for future use.

 - VEXTERNC int `writedataFlat` (NOsh *nosh, Vcom *com, const char *fname, double totEnergy[NOSH_MAXCALC], double qfEnergy[NOSH_MAXCALC], double qmEnergy[NOSH_MAXCALC], double dielEnergy[NOSH_MAXCALC], int nenergy[NOSH_MAXCALC], double *atomEnergy[NOSH_MAXCALC], int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC])

Write out information to a flat file.

 - VEXTERNC int `writedataXML` (NOsh *nosh, Vcom *com, const char *fname, double totEnergy[NOSH_MAXCALC], double qfEnergy[NOSH_MAXCALC], double qmEnergy[NOSH_MAXCALC], double dielEnergy[NOSH_MAXCALC], int nenergy[NOSH_MAXCALC], double *atomEnergy[NOSH_MAXCALC], int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC])

- Write out information to an XML file.*

 - VEXTERNC int `writedataMG` (int rank, `NOsh` *nosh, `PBEparm` *pbeparm, `Vpmg` *pmg)
- Write out observables from MG calculation to file.*

 - VEXTERNC int `writematMG` (int rank, `NOsh` *nosh, `PBEparm` *pbeparm, `Vpmg` *pmg)
- Write out operator matrix from MG calculation to file.*

 - VEXTERNC double `returnEnergy` (Vcom *com, `NOsh` *nosh, double totEnergy[NOSH_MAXCALC], int iprint)
- Access net local energy.*

 - VEXTERNC int `printEnergy` (Vcom *com, `NOsh` *nosh, double totEnergy[NOSH_MAXCALC], int iprint)
- Combine and pretty-print energy data (deprecated...see printElecEnergy)*

 - VEXTERNC int `printElecEnergy` (Vcom *com, `NOsh` *nosh, double totEnergy[NOSH_MAXCALC], int iprint)
- Combine and pretty-print energy data.*

 - VEXTERNC int `printApolEnergy` (`NOsh` *nosh, int iprint)
- Combine and pretty-print energy data.*

 - VEXTERNC int `printForce` (Vcom *com, `NOsh` *nosh, int nforce[NOSH_MAXCALC], `AtomForce` *atomForce[NOSH_MAXCALC], int i)
- Combine and pretty-print force data (deprecated...see printElecForce)*

 - VEXTERNC int `printElecForce` (Vcom *com, `NOsh` *nosh, int nforce[NOSH_MAXCALC], `AtomForce` *atomForce[NOSH_MAXCALC], int i)
- Combine and pretty-print force data.*

 - VEXTERNC int `printApolForce` (Vcom *com, `NOsh` *nosh, int nforce[NOSH_MAXCALC], `AtomForce` *atomForce[NOSH_MAXCALC], int i)
- Combine and pretty-print force data.*

 - VEXTERNC void `startVio` ()
- Wrapper to start MALOC Vio layer.*

 - VEXTERNC int `energyAPOL` (`APOLparm` *apolparm, double sasa, double sav, double atomsasa[], double atomwcaEnergy[], int numatoms)
- Calculate non-polar energies.*

 - VEXTERNC int `forceAPOL` (Vacc *acc, Vmem *mem, `APOLparm` *apolparm, int *nforce, `AtomForce` **atomForce, `Valist` *alist, `Vclist` *clist)
- Calculate non-polar forces.*

 - VEXTERNC int `initAPOL` (`NOsh` *nosh, Vmem *mem, `Vparam` *param, `APOLparm` *apolparm, int *nforce, `AtomForce` **atomForce, `Valist` *alist)
- Upperlevel routine to the non-polar energy and force routines.*

 - VEXTERNC void `printFEPARM` (int icalc, `NOsh` *nosh, `FEMparm` *feparm, `Vfetk` *fetk[NOSH_MAXCALC])
- Print out FE-specific params loaded from input.*

 - VEXTERNC int `energyFE` (`NOsh` *nosh, int icalc, `Vfetk` *fetk[NOSH_MAXCALC], int *nenergy, double *totEnergy, double *qfEnergy, double *qmEnergy, double *dielEnergy)
- Calculate electrostatic energies from FE solution.*

 - VEXTERNC void `killFE` (`NOsh` *nosh, `Vpbe` *pbe[NOSH_MAXCALC], `Vfetk` *fetk[NOSH_MAXCALC], `Gem` *gem[NOSH_MAXMOL])
- Kill structures initialized during an FE calculation.*

 - VEXTERNC int `preRefineFE` (int i, `FEMparm` *feparm, `Vfetk` *fetk[NOSH_MAXCALC])
- Pre-refine mesh before solve.*

 - VEXTERNC int `partFE` (int i, `NOsh` *nosh, `FEMparm` *feparm, `Vfetk` *fetk[NOSH_MAXCALC])
- Partition mesh (if applicable)*

 - VEXTERNC int `solveFE` (int i, `PBEparm` *pbeparm, `FEMparm` *feparm, `Vfetk` *fetk[NOSH_MAXCALC])
- Solve-estimate-refine.*

 - VEXTERNC int `postRefineFE` (int icalc, `FEMparm` *feparm, `Vfetk` *fetk[NOSH_MAXCALC])

- Estimate error, mark mesh, and refine mesh after solve.*
- VEXTERNC int [writedataFE](#) (int rank, [NOsh](#) *nosh, [PBEparm](#) *pbeparm, [Vfetk](#) *fetk)
- Write FEM data to files.*
- VEXTERNC Vrc_Codes [loadMeshes](#) ([NOsh](#) *nosh, Gem *gm[[NOSH_MAXMOL](#)])
- Load the meshes given in NOsh into geometry objects.*
- VEXTERNC void [killMeshes](#) ([NOsh](#) *nosh, Gem *alist[[NOSH_MAXMOL](#)])
- Destroy the loaded meshes.*
- VEXTERNC int [initGEOFLOW](#) (int icalc, [NOsh](#) *nosh, [GEOFLOWparm](#) *bemparm, [PBEparm](#) *pbeparm, [Vpbe](#) *pbe[[NOSH_MAXCALC](#)])
- Initialize an GEOFLOW calculation.*
- VEXTERNC void [killGEOFLOW](#) ([NOsh](#) *nosh, [Vpbe](#) *pbe[[NOSH_MAXCALC](#)])
- Kill structures initialized during an GEOFLOW calculation.*
- VEXTERNC int [solveGEOFLOW](#) ([Valist](#) *molecules[[NOSH_MAXMOL](#)], [NOsh](#) *nosh, [PBEparm](#) *pbeparm, [APOLparm](#) *apolparm, [GEOFLOWparm](#) *parm, [GEOFLOWparm_CalcType](#) type)
- Solve the PBE with GEOFLOW.*
- VEXTERNC int [setPartGEOFLOW](#) ([NOsh](#) *nosh, [GEOFLOWparm](#) *parm)
- Set GEOFLOW partitions for calculating observables and performing I/O.*
- VEXTERNC int [energyGEOFLOW](#) ([NOsh](#) *nosh, int icalc, int *nenergy, double *totEnergy, double *qfEnergy, double *qmEnergy, double *dielEnergy)
- Calculate electrostatic energies from GEOFLOW solution.*
- VEXTERNC int [forceGEOFLOW](#) ([NOsh](#) *nosh, [PBEparm](#) *pbeparm, [GEOFLOWparm](#) *parm, int *nforce, [AtomForce](#) **atomForce, [Valist](#) *alist[[NOSH_MAXMOL](#)])
- Calculate forces from GEOFLOW solution.*
- VEXTERNC void [printGEOFLOWPARAM](#) ([GEOFLOWparm](#) *parm)
- Print out GEOFLOW-specific params loaded from input.*
- VEXTERNC int [writedataGEOFLOW](#) (int rank, [NOsh](#) *nosh, [PBEparm](#) *pbeparm)
- Write out observables from GEOFLOW calculation to file.*
- VEXTERNC int [writematGEOFLOW](#) (int rank, [NOsh](#) *nosh, [PBEparm](#) *pbeparm)
- Write out operator matrix from GEOFLOW calculation to file.*

8.30.1 Detailed Description

8.30.2 Function Documentation

- 8.30.2.1 VEXTERNC int [energyAPOL](#) ([APOLparm](#) * [apolparm](#), double [sasa](#), double [sav](#), double [atomsasa](#)[], double [atomwcaEnergy](#)[], int [numatoms](#))

Calculate non-polar energies.

Author

David Gohara

Returns

1 if successful, 0 otherwise

Parameters

<i>sasa</i>	APOLparm object
<i>sav</i>	Solvent accessible surface area
<i>atomsasa</i>	Solvent accessible volume
<i>atomwcaEnergy</i>	Array for SASA per atom *
<i>numatoms</i>	Array for WCA energy per atom * Number of atoms (or size of the above arrays) *

Definition at line 4507 of file [routines.c](#).

8.30.2.2 VEXTERNC int energyFE (NOsh * *nosh*, int *icalc*, Vfetc * *fetc*[NOSH_MAXCALC], int * *nenergy*, double * *totEnergy*, double * *qfEnergy*, double * *qmEnergy*, double * *dielEnergy*)

Calculate electrostatic energies from FE solution.

Author

Nathan Baker

Parameters

<i>nosh</i>	Object with parsed input file parameters
<i>icalc</i>	Index of calculation
<i>fetc</i>	FE object array
<i>nenergy</i>	Set to number of entries in energy arrays
<i>totEnergy</i>	Set to total energy (in kT)
<i>qfEnergy</i>	Set to charge-potential energy (in kT)
<i>qmEnergy</i>	Set to mobile ion energy (in kT)
<i>dielEnergy</i>	Set to polarization energy (in kT)

Bug "calcenergy 2" does not work

Returns

1 if successful, 0 otherwise

Calculates the electrostatic energies from an FE calculation. < FE-specific parameters

< PBE-specific parameters

If we're not ignoring this particular NOsh object because it has been rendered invalid, call the Vfetc object's energy calculation function. The flag differences specified have to do with setting specific calculation restrictions (see color variable documentation in function code).

Parameters

<i>nosh</i>	Object with parsed input file parameters
<i>icalc</i>	Calculation index
<i>fetc</i>	FE object array
<i>nenergy</i>	Set to number of entries in energy arrays
<i>totEnergy</i>	Set to total energy (in kT)
<i>qfEnergy</i>	Set to charge-potential energy (in kT)

<i>qmEnergy</i>	Set to mobile ion energy (in kT)
<i>dielEnergy</i>	Set to polarization energy (in kT)

Definition at line 4019 of file [routines.c](#).

8.30.2.3 VEXTERNC int energyGEOFLOW (NOsh * *nosh*, int *icalc*, int * *nenergy*, double * *totEnergy*, double * *qfEnergy*, double * *qmEnergy*, double * *dielEnergy*)

Calculate electrostatic energies from GEOFLOW solution.

Author

Andrew Stevens

Parameters

<i>nosh</i>	Object with parsed input file parameters
<i>icalc</i>	Index of calculation
<i>nenergy</i>	Set to number of entries in energy arrays
<i>totEnergy</i>	Set to total energy (in kT)
<i>qfEnergy</i>	Set to charge-potential energy (in kT)
<i>qmEnergy</i>	Set to mobile ion energy (in kT)
<i>dielEnergy</i>	Set to polarization energy (in kT)

Returns

1 if successful, 0 otherwise

Definition at line 5051 of file [routines.c](#).

8.30.2.4 VEXTERNC int energyMG (NOsh * *nosh*, int *icalc*, Vpmg * *pmg*, int * *nenergy*, double * *totEnergy*, double * *qfEnergy*, double * *qmEnergy*, double * *dielEnergy*)

Calculate electrostatic energies from MG solution.

Author

Nathan Baker

Parameters

<i>nosh</i>	Object with parsed input file parameters
<i>icalc</i>	Index of calculation
<i>pmg</i>	MG object
<i>nenergy</i>	Set to number of entries in energy arrays
<i>totEnergy</i>	Set to total energy (in kT)
<i>qfEnergy</i>	Set to charge-potential energy (in kT)
<i>qmEnergy</i>	Set to mobile ion energy (in kT)

<i>dielEnergy</i>	Set to polarization energy (in kT)
-------------------	------------------------------------

Returns

1 if successful, 0 otherwise

Definition at line 1423 of file [routines.c](#).

8.30.2.5 VEXTERNC int forceAPOL (Vacc * *acc*, Vmem * *mem*, APOLparm * *apolparm*, int * *nforce*, AtomForce ** *atomForce*, Valist * *alist*, Vclist * *clist*)

Calculate non-polar forces.

Author

David Gohara

Returns

1 if successful, 0 otherwise

Parameters

<i>acc</i>	Accessibility object
<i>mem</i>	Memory manager
<i>apolparm</i>	Apolar calculation parameter object
<i>nforce</i>	Number of atomic forces to calculate statements for
<i>atomForce</i>	Object for storing atom forces
<i>alist</i>	Atom list
<i>clist</i>	Cell list for accessibility object

Definition at line 4562 of file [routines.c](#).

8.30.2.6 VEXTERNC int forceGEOFLOW (NOsh * *nosh*, PBeparm * *pbeparm*, GEOFLOWparm * *parm*, int * *nforce*, AtomForce ** *atomForce*, Valist * *alist*[NOSH_MAXMOL])

Calculate forces from GEOFLOW solution.

Author

Andrew Stevens

Parameters

<i>nosh</i>	Parameters from input file
<i>pbeparm</i>	Generic PBE parameters
<i>parm</i>	GEOFLOW-specific parameters
<i>nforce</i>	Set to number of forces in arrays

<i>atomForce</i>	List of atom forces
<i>alist</i>	List of atom lists

Returns

1 if successful, 0 otherwise

Definition at line 5077 of file [routines.c](#).

8.30.2.7 VEXTERNC int forceMG (Vmem * *mem*, NOsh * *nosh*, PBEparm * *pbeparm*, MGparm * *mgparm*, Vpmg * *pmg*, int * *nforce*, AtomForce ** *atomForce*, Valist * *alist*[NOSH_MAXMOL])

Calculate forces from MG solution.

Author

Nathan Baker

Parameters

<i>mem</i>	Memory management object
<i>nosh</i>	Parameters from input file
<i>pbeparm</i>	Generic PBE parameters
<i>mgparm</i>	MG-specific parameters
<i>pmg</i>	MG object
<i>nforce</i>	Set to number of forces in arrays
<i>atomForce</i>	List of atom forces
<i>alist</i>	List of atom lists

Returns

1 if successful, 0 otherwise

Definition at line 1490 of file [routines.c](#).

8.30.2.8 VEXTERNC int initAPOL (NOsh * *nosh*, Vmem * *mem*, Vparam * *param*, APOLparm * *apolparm*, int * *nforce*, AtomForce ** *atomForce*, Valist * *alist*)

Upperlevel routine to the non-polar energy and force routines.

Author

David Gohara

Returns

1 if successful, 0 otherwise

<

< Number of atoms

< Used to capture length of loops to prevent multiple calls in counters

<

<

< Temporary timing variable for debugging (PCE)

<

<

<

<

<

<

<

<

<

<

<

<

<

<

<

< WCA energy per atom

<

<

<

<

<

<

<

<

<

<

<

<

<

Parameters

<i>nosh</i>	Input parameter object
-------------	------------------------

<i>mem</i>	Memory manager
<i>param</i>	Atom parameters
<i>apolparam</i>	Apolar calculation parameters
<i>nforce</i>	Number of force calculations
<i>atomForce</i>	Atom force storage object
<i>alist</i>	Atom list

Definition at line 4308 of file [routines.c](#).

8.30.2.9 VEXTERNC Vrc_Codes initFE (int *icalc*, NOsh * *nosh*, FEMparm * *feparm*, PBEparm * *pbeparm*, Vpbe * *pbe*[NOSH_MAXCALC], Valist * *alist*[NOSH_MAXMOL], Vfetk * *fetk*[NOSH_MAXCALC])

Initialize FE solver objects.

Author

Nathan Baker

Bug THIS FUNCTION IS HARD-CODED TO SOLVE LRPBE

Author

Nathan Baker

Bug THIS FUNCTION IS HARD-CODED TO SOLVE LRPBE

< Loop counter
 < Mesh ID
 < Loop counter
 < Loop counter
 < Molecule ID
 <
 < I/O socket for reading MCSF mesh data
 < Total bytes used by this operation
 < High-water memory usage for this operation
 < The type of mesh being used (see struct for enum values)
 <
 <
 <
 <
 <
 < Return codes for function calls (see struct for enum value)
 < List of atoms being operated on
 < Atom/molecule being operated on

Parameters

<i>icalc</i>	Index in pb, fetk to initialize (calculation index)
<i>nosh</i>	Master parameter object
<i>feparm</i>	FE-specific parameters
<i>pbeparm</i>	Generic PBE parameters
<i>pbe</i>	Array of PBE objects
<i>alist</i>	Array of atom lists
<i>fetk</i>	Array of finite element objects

Definition at line 3554 of file [routines.c](#).

8.30.2.10 VEXTERNC int initGEOFLOW (int *icalc*, NOsh * *nosh*, GEOFLOWparm * *beparm*, PBEparm * *pbeparm*, Vpbe * *pbe*[NOSH_MAXCALC])

Initialize an GEOFLOW calculation.

Author

Andrew Stevens

Returns

1 if succesful, 0 otherwise

Initialize a geometric flow calculation.

Parameters

<i>icalc</i>	Index of calculation in pbem/pmbem arrays
<i>nosh</i>	Object with parsed input file parameters
<i>beparm</i>	Object with GEOFLOW -specific parameters
<i>pbeparm</i>	Object with generic PBE parameters
<i>pbe</i>	Array of Vpbe objects (one for each calc)

Definition at line 4934 of file [routines.c](#).

8.30.2.11 VEXTERNC int initMG (int *icalc*, NOsh * *nosh*, MGparm * *mgparm*, PBEparm * *pbeparm*, double *realCenter*[3], Vpbe * *pbe*[NOSH_MAXCALC], Valist * *alist*[NOSH_MAXMOL], Vgrid * *dielXMap*[NOSH_MAXMOL], Vgrid * *dielYMap*[NOSH_MAXMOL], Vgrid * *dielZMap*[NOSH_MAXMOL], Vgrid * *kappaMap*[NOSH_MAXMOL], Vgrid * *chargeMap*[NOSH_MAXMOL], Vpmgp * *pmgp*[NOSH_MAXCALC], Vpmg * *pmg*[NOSH_MAXCALC], Vgrid * *potMap*[NOSH_MAXMOL])

Initialize an MG calculation.

Author

Nathan Baker

Returns

1 if succesful, 0 otherwise

Initialize a multigrid calculation.

Parameters

<i>icalc</i>	Index of calculation in pmg/pmpg arrays
<i>nosh</i>	Object with parsed input file parameters
<i>mgparm</i>	Object with MG-specific parameters
<i>pbeparm</i>	Object with generic PBE parameters
<i>realCenter</i>	The actual center of the current mesh
<i>pbe</i>	Array of Vpbe objects (one for each calc)
<i>alist</i>	Array of atom lists
<i>dielXMap</i>	Array of x-shifted dielectric maps
<i>dielYMap</i>	Array of y-shifted dielectric maps
<i>dielZMap</i>	Array of z-shifted dielectric maps
<i>kappaMap</i>	Array of kappa maps
<i>chargeMap</i>	Array of charge maps
<i>pmpg</i>	Array of MG parameter objects (one for each calc)
<i>pmg</i>	Array of MG objects (one for each calc)
<i>potMap</i>	Array of potential maps

Definition at line 1074 of file [routines.c](#).

8.30.2.12 VEXTERN void killChargeMaps (NOsh * *nosh*, Vgrid * *charge*[NOSH_MAXMOL])

Destroy the loaded charge maps.

Author

Nathan Baker

Parameters

<i>nosh</i>	NOsh object with input file information
<i>charge</i>	List of charge maps

Definition at line 849 of file [routines.c](#).

8.30.2.13 VEXTERN void killDielMaps (NOsh * *nosh*, Vgrid * *dielXMap*[NOSH_MAXMOL], Vgrid * *dielYMap*[NOSH_MAXMOL], Vgrid * *dielZMap*[NOSH_MAXMOL])

Destroy the loaded dielectric.

Author

Nathan Baker

Parameters

<i>nosh</i>	NOsh object with input file information
<i>dielXMap</i>	List of x-shifted dielectric maps
<i>dielYMap</i>	List of y-shifted dielectric maps
<i>dielZMap</i>	List of z-shifted dielectric maps

Definition at line 550 of file [routines.c](#).

8.30.2.14 VEXTERNC void killEnergy ()

Kill arrays allocated for energies.

Author

Nathan Baker

Definition at line 1628 of file [routines.c](#).

8.30.2.15 VEXTERNC void killFE (NOsh * *nosh*, Vpbe * *pbe*[NOSH_MAXCALC], Vfetk * *fetk*[NOSH_MAXCALC], Gem * *gem*[NOSH_MAXMOL])

Kill structures initialized during an FE calculation.

Author

Nathan Baker

Parameters

<i>pbe</i>	Object with parsed input file parameters
<i>fetk</i>	Array of Vpbe objects for each calc
<i>gem</i>	Array of FEtk objects for each calc Array of geometry manager objects for each calc

Definition at line 3526 of file [routines.c](#).

8.30.2.16 VEXTERNC void killForce (Vmem * *mem*, NOsh * *nosh*, int *nforce*[NOSH_MAXCALC], AtomForce * *atomForce*[NOSH_MAXCALC])

Free memory from MG force calculation.

Author

Nathan Baker

Parameters

<i>mem</i>	Memory management object
<i>nosh</i>	Parameters from input file
<i>nforce</i>	Number of forces in arrays
<i>atomForce</i>	List of atom forces

Definition at line 1636 of file [routines.c](#).

8.30.2.17 VEXTERNC void killGEOFLOW (NOsh * *nosh*, Vpbe * *pbe*[NOSH_MAXCALC])

Kill structures initialized during an GEOFLOW calculation.

Author

Andrew Stevens

Parameters

<i>pbe</i>	Object with parsed input file parameters Array of Vpbe objects for each calc
------------	--

Definition at line 4950 of file [routines.c](#).

8.30.2.18 VEXTERNC void killKappaMaps (NOsh * *nosh*, Vgrid * *kappa*[NOSH_MAXMOL])

Destroy the loaded kappa maps.

Author

Nathan Baker

Parameters

<i>nosh</i>	NOsh object with input file information
<i>kappa</i>	List of kappa maps

Definition at line 662 of file [routines.c](#).

8.30.2.19 VEXTERNC void killMeshes (NOsh * *nosh*, Gem * *alist*[NOSH_MAXMOL])

Destroy the loaded meshes.

Author

Nathan Baker

Parameters

<i>nosh</i>	NOsh object with input file information
<i>alist</i>	Populated list of geometry objects to be destroyed

8.30.2.20 VEXTERNC void killMG (NOsh * *nosh*, Vpbe * *pbe*[NOSH_MAXCALC], Vpmgp * *pmgp*[NOSH_MAXCALC], Vpmg * *pmg*[NOSH_MAXCALC])

Kill structures initialized during an MG calculation.

Author

Nathan Baker

Parameters

<i>pbe</i>	Object with parsed input file parameters
<i>pmgp</i>	Array of Vpbe objects for each calc
<i>pmg</i>	Array of MG parameter objects for each calc Array of MG objects for each calc

Definition at line 1315 of file [routines.c](#).

8.30.2.21 VEXTERNC void killMolecules (NOsh * *nosh*, Valist * *alist*[NOSH_MAXMOL])

Destroy the loaded molecules.

Author

Nathan Baker

Parameters

<i>nosh</i>	NOsh object with input file information
<i>alist</i>	List of atom list objects

Definition at line 233 of file [routines.c](#).8.30.2.22 VEXTERNC void killPotMaps (NOsh * *nosh*, Vgrid * *pot*[NOSH_MAXMOL])

Destroy the loaded potential maps.

Author

David Gohara

Parameters

<i>nosh</i>	NOsh object with input file information
<i>pot</i>	List of potential maps

Definition at line 751 of file [routines.c](#).8.30.2.23 VEXTERNC int loadChargeMaps (NOsh * *nosh*, Vgrid * *map*[NOSH_MAXMOL])

Load the charge maps given in NOsh into grid objects.

Author

Nathan Baker

Parameters

<i>nosh</i>	NOsh object with input file information
<i>charge</i>	List of kappa maps

Returns

1 if successful, 0 otherwise
 0 on failure, 1 on success

Definition at line 770 of file [routines.c](#).8.30.2.24 VEXTERNC int loadDielMaps (NOsh * *nosh*, Vgrid * *dielXMap*[NOSH_MAXMOL], Vgrid * *dielYMap*[NOSH_MAXMOL], Vgrid * *dielZMap*[NOSH_MAXMOL])

Load the dielectric maps given in NOsh into grid objects.

Author

Nathan Baker

Parameters

<i>nosh</i>	NOsh object with input file information
<i>dielXMap</i>	List of x-shifted dielectric maps
<i>dielYMap</i>	List of y-shifted dielectric maps
<i>dielZMap</i>	List of x-shifted dielectric maps

Returns

1 if successful, 0 otherwise

Loads dielectric map path data into NOsh object

Returns

1 on success, 0 on error

Definition at line [250](#) of file [routines.c](#).

8.30.2.25 VEXTERNC int loadKappaMaps (NOsh * *nosh*, Vgrid * *map*[NOSH_MAXMOL])

Load the kappa maps given in NOsh into grid objects.

Author

Nathan Baker

Parameters

<i>nosh</i>	NOsh object with input file information
<i>kappa</i>	List of kappa maps

Returns

1 if successful, 0 otherwise
0 on failure, 1 on success

Definition at line [575](#) of file [routines.c](#).

8.30.2.26 VEXTERNC Vrc_Codes loadMeshes (NOsh * *nosh*, Gem * *gm*[NOSH_MAXMOL])

Load the meshes given in NOsh into geometry objects.

Author

Nathan Baker

Returns

Error code on success/failure

Parameters

<i>nosh</i>	NOsh object with input file information
<i>gm</i>	List of geometry objects (to be populated)

8.30.2.27 VEXTERNC int loadMolecules (NOsh * *nosh*, Vparam * *param*, Valist * *alist*[NOSH_MAXMOL])

Load the molecules given in NOsh into atom lists.

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Parameters

<i>nosh</i>	NOsh object with input file information
<i>param</i>	NULL (if PQR files only) or pointer to parameter object
<i>alist</i>	List of atom list objects (to be populated)

Definition at line 95 of file [routines.c](#).

8.30.2.28 VEXTERNC Vparam* loadParameter (NOsh * *nosh*)

Loads and returns parameter object.

Author

Nathan Baker

Returns

Pointer to parameter object or NULL

Parameters

<i>nosh</i>	Pointer to NOsh object with input file information
-------------	--

Definition at line 60 of file [routines.c](#).

8.30.2.29 VEXTERNC int loadPotMaps (NOsh * *nosh*, Vgrid * *map*[NOSH_MAXMOL])

Load the potential maps given in NOsh into grid objects.

Author

David Gohara

Parameters

<i>nosh</i>	NOsh object with input file information
<i>pot</i>	List of potential maps

Returns

1 if successful, 0 otherwise
0 on failure, 1 on success

Definition at line 679 of file [routines.c](#).

8.30.2.30 int main (int *argc*, char ** *argv*)

The main APBS function.

Author

Nathan Baker, Dave Gohara, Todd Dolinsky

Returns

Status code (0 for success)

Parameters

<i>argc</i>	Number of arguments
<i>argv</i>	Argument strings

Definition at line 80 of file [main.c](#).

8.30.2.31 VEXTERNC int partFE (int *i*, NOsh * *nosh*, FEMparm * *feparm*, Vfetk * *fetk*[NOSH_MAXCALC])

Partition mesh (if applicable)

Author

Nathan Baker

Parameters

<i>i</i>	Calculation index
<i>nosh</i>	Master parameter object
<i>feparm</i>	FE-specific parameters
<i>fetk</i>	Array of FE solver objects

Returns

1 if successful, 0 otherwise

Definition at line 3879 of file [routines.c](#).

8.30.2.32 VEXTERNC int postRefineFE (int *icalc*, FEMparm * *feparm*, Vfetk * *fetk*[NOSH_MAXCALC])

Estimate error, mark mesh, and refine mesh after solve.

Author

Nathan Baker

Parameters

<i>icalc</i>	Calculation index
<i>feparm</i>	FE-specific parameters
<i>fetk</i>	Array of FE solver objects

Returns

1 if successful, 0 otherwise – note that a 0 will likely imply that either the max number of vertices have been met or no vertices were marked for refinement. In either case, this should not be treated as a fatal error.

Estimates the error, marks the mesh, and refines the mesh after solving.

Returns

1 if successful, 0 otherwise – note that a 0 will likely imply that either the max number of vertices have been met or no vertices were marked for refinement. In either case, this should not be treated as a fatal error.

< Number of vertices in the molecular geometry

< Whether vertices are marked for refinement

Parameters

<i>icalc</i>	Calculation index
<i>feparm</i>	FE-specific parameters
<i>fetk</i>	Array of FE solver objects

Definition at line 4079 of file [routines.c](#).

8.30.2.33 VEXTERNC int preRefineFE (int *i*, FEMparm * *feparm*, Vfetk * *fetk*[NOSH_MAXCALC])

Pre-refine mesh before solve.

Author

Nathan Baker

Parameters

<i>i</i>	Calculation index
<i>nosh</i>	Master parameter object
<i>feparm</i>	FE-specific parameters

<i>fetk</i>	Array of FE solver objects
-------------	----------------------------

Returns

1 if successful, 0 otherwise

< Number of vertices in the mesh geometry

< Essentially a boolean; indicates whether further refinement is required after running MC's refinement algorithm.

TODO: could this be optimized by moving nverts out of the loop to just above this initial printout? This depends heavily on whether the number of vertices can change during the calculation. - PCE

Definition at line 3886 of file [routines.c](#).

8.30.2.34 VEXTERNC int printApolEnergy (NOsh * *nosh*, int *iprint*)

Combine and pretty-print energy data.

Author

David Gohara

Returns

1 if successful, 0 otherwise

Parameters

<i>nosh</i>	Parameters from input file
<i>iprint</i>	Index of energy statement to print

Definition at line 2761 of file [routines.c](#).

8.30.2.35 VEXTERNC int printApolForce (Vcom * *com*, NOsh * *nosh*, int *nforce*[NOSH_MAXCALC], AtomForce * *atomForce*[NOSH_MAXCALC], int *i*)

Combine and pretty-print force data.

Author

David Gohara

Returns

1 if successful, 0 otherwise

Parameters

<i>nosh</i>	Communications object
<i>nforce</i>	Parameters from input file
<i>atomForce</i>	Number of forces calculated
<i>i</i>	Array of force structures Index of force statement to print

Definition at line 3314 of file [routines.c](#).

8.30.2.36 VEXTERNC int printElecEnergy (Vcom * *com*, NOsh * *nosh*, double *totEnergy*[NOSH_MAXCALC], int *iprint*)

Combine and pretty-print energy data.

Author

David Gohara

Returns

1 if successful, 0 otherwise

Parameters

<i>nosh</i>	Communications object
<i>totEnergy</i>	Parameters from input file
<i>iprint</i>	Array of energies from different calculations Index of energy statement to print

Definition at line 2696 of file [routines.c](#).

8.30.2.37 VEXTERNC int printElecForce (Vcom * *com*, NOsh * *nosh*, int *nforce*[NOSH_MAXCALC], AtomForce * *atomForce*[NOSH_MAXCALC], int *i*)

Combine and pretty-print force data.

Author

David Gohara

Returns

1 if successful, 0 otherwise

Parameters

<i>nosh</i>	Communications object
<i>nforce</i>	Parameters from input file
<i>atomForce</i>	Number of forces calculated
<i>i</i>	Array of force structures Index of force statement to print

Definition at line 3071 of file [routines.c](#).

8.30.2.38 VEXTERNC int printEnergy (Vcom * *com*, NOsh * *nosh*, double *totEnergy*[NOSH_MAXCALC], int *iprint*)

Combine and pretty-print energy data (deprecated...see printElecEnergy)

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Parameters

<i>nosh</i>	Communications object
<i>totEnergy</i>	Parameters from input file
<i>iprint</i>	Array of energies from different calculations Index of energy statement to print

Definition at line 2628 of file [routines.c](#).

8.30.2.39 VEXTERNC void printFEPARM (int *icalc*, NOsh * *nosh*, FEMparm * *feparm*, Vfetk * *fetk*[NOSH_MAXCALC])

Print out FE-specific params loaded from input.

Author

Nathan Baker

Parameters

<i>icalc</i>	Calculation index
<i>nosh</i>	Master parameter object
<i>feparm</i>	FE-specific parameters
<i>fetk</i>	Array of FE solver objects

Definition at line 3743 of file [routines.c](#).

8.30.2.40 VEXTERNC int printForce (Vcom * *com*, NOsh * *nosh*, int *nforce*[NOSH_MAXCALC], AtomForce * *atomForce*[NOSH_MAXCALC], int *i*)

Combine and pretty-print force data (deprecated...see printElecForce)

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Parameters

<i>nosh</i>	Communications object
<i>nforce</i>	Parameters from input file
<i>atomForce</i>	Number of forces calculated
<i>i</i>	Array of force structures Index of force statement to print

Definition at line 2823 of file [routines.c](#).

8.30.2.41 VEXTERNC void printGEOFLOWPARAM (GEOFLOWparm * *parm*)

Print out GEOFLOW-specific params loaded from input.

Author

Andrew Stevens

Definition at line 5103 of file [routines.c](#).

8.30.2.42 VEXTERNC void printMGPARM (MGparm * mgparm, double realCenter[3])

Print out MG-specific params loaded from input.

Author

Nathan Baker

Parameters

<i>realCenter</i>	Center of mesh for actual calculation
<i>mgparm</i>	MGparm object

Definition at line 1041 of file [routines.c](#).

8.30.2.43 VEXTERNC void printPBEPARM (PBEparm * pbeparm)

Print out generic PBE params loaded from input.

Author

Nathan Baker

Parameters

<i>pbeparm</i>	PBEparm object
----------------	----------------

Definition at line 867 of file [routines.c](#).

8.30.2.44 VEXTERNC double returnEnergy (Vcom * com, NOsh * nosh, double totEnergy[NOSH_MAXCALC], int iprint)

Access net local energy.

Author

Justin Xiang

Parameters

<i>com</i>	Communications object
<i>nosh</i>	Parameters from input file
<i>totEnergy</i>	Array of energies from different calculations
<i>iprint</i>	Index of energy statement to print

Returns

Net local energy

Definition at line 2596 of file [routines.c](#).

8.30.2.45 VEXTERNC int setPartGEOFLOW (NOsh * nosh, GEOFLOWparm * parm)

Set GEOFLOW partitions for calculating observables and performing I/O.

Author

Andrew Stevens

Parameters

<i>nosh</i>	Object with parsed input file parameters
<i>bemparm</i>	GEOFLOW parameters from input file

Returns

1 if successful, 0 otherwise

Definition at line 5037 of file [routines.c](#).

8.30.2.46 VEXTERNC int setPartMG (NOsh * *nosh*, MGparm * *mgparm*, Vpmg * *pmg*)

Set MG partitions for calculating observables and performing I/O.

Author

Nathan Baker

Parameters

<i>nosh</i>	Object with parsed input file parameters
<i>mgparm</i>	MG parameters from input file
<i>pmg</i>	MG object

Returns

1 if successful, 0 otherwise

Definition at line 1377 of file [routines.c](#).

8.30.2.47 VEXTERNC int solveFE (int *icalc*, PBEparm * *pbeparm*, FEMparm * *feparm*, Vfetk * *fetk*[NOSH_MAXCALC])

Solve-estimate-refine.

Author

Nathan Baker

Parameters

<i>i</i>	Calculation index
<i>feparm</i>	FE-specific parameters
<i>pbeparm</i>	Generic PBE parameters
<i>fetk</i>	Array of FE solver objects

Returns

1 if successful, 0 otherwise

Call MC's mesh solving equations depending upon the type of PBE we're dealing with. < AM_hPcg

< Coarse-grid solver; 0 = SLU, 1 = MG, 2 = CG, 3 = BCG, 4 = PCG, 5 = PBCG

< Primal problem

< Preconditioner; 0 = identity.

Parameters

<i>icalc</i>	Calculation index
<i>pbeparm</i>	PBE-specific parameters
<i>feparm</i>	FE-specific parameters
<i>fetk</i>	Array of FE solver objects

Definition at line 3956 of file [routines.c](#).

8.30.2.48 VEXTERNC int solveGEOFLOW (Valist * *molecules*[*NOSH_MAXMOL*], NOsh * *nosh*, PBEparm * *pbeparm*, APOLparm * *apolparm*, GEOFLOWparm * *parm*, GEOFLOWparm_CalcType *type*)

Solve the PBE with GEOFLOW.

Author

Andrew Stevens

Parameters

<i>nosh</i>	Object with parsed input file parameters
<i>pbem</i>	GEOFLOW objects for this calculation
<i>type</i>	Type of GEOFLOW calculation

Returns

1 if successful, 0 otherwise

Definition at line 4962 of file [routines.c](#).

8.30.2.49 VEXTERNC int solveMG (NOsh * *nosh*, Vpmg * *pmg*, MGparm_CalcType *type*)

Solve the PBE with MG.

Author

Nathan Baker

Parameters

<i>nosh</i>	Object with parsed input file parameters
<i>pmg</i>	MG objects for this calculation
<i>type</i>	Type of MG calculation

Returns

1 if successful, 0 otherwise

Definition at line 1341 of file [routines.c](#).

8.30.2.50 VEXTERNC void startVio ()

Wrapper to start MALOC Vio layer.

Author

Nathan Baker and Robert Konecny

Definition at line 58 of file [routines.c](#).

8.30.2.51 VEXTERNC void storeAtomEnergy (Vpmg * *pmg*, int *icalc*, double ** *atomEnergy*, int * *nenergy*)

Store energy in arrays for future use.

Author

Todd Dolinsky

Parameters

<i>pmg</i>	MG object
<i>icalc</i>	Calculation number
<i>atomEnergy</i>	Pointer to storage array of doubles
<i>nenergy</i>	Stores number of atoms per calc

Definition at line 1724 of file [routines.c](#).

8.30.2.52 VEXTERNC int writedataFE (int *rank*, NOsh * *nosh*, PBEparm * *pbeparm*, Vfetk * *fetk*)

Write FEM data to files.

Author

Nathan Baker

Parameters

<i>rank</i>	Rank of processor (for parallel runs)
<i>nosh</i>	NOsh object
<i>pbeparm</i>	PBEparm object
<i>fetk</i>	FETk object (with solution)

Returns

1 if successful, 0 otherwise

Write FEM data to file. <

<

< Loop counter

< Flag indicating whether data can be written to output

<

<

Parameters

<i>rank</i>	Rank of processor (for parallel runs)
<i>nosh</i>	NOsh object
<i>pbeparm</i>	PBE-specific parameters
<i>fetk</i>	FEtk object (with solution)

Definition at line 4140 of file [routines.c](#).

8.30.2.53 VEXTERNC int writedataFlat (NOsh * *nosh*, Vcom * *com*, const char * *fname*, double *totEnergy*[NOSH_MAXCALC], double *qfEnergy*[NOSH_MAXCALC], double *qmEnergy*[NOSH_MAXCALC], double *dielEnergy*[NOSH_MAXCALC], int *nenergy*[NOSH_MAXCALC], double * *atomEnergy*[NOSH_MAXCALC], int *nforce*[NOSH_MAXCALC], AtomForce * *atomForce*[NOSH_MAXCALC])

Write out information to a flat file.

Author

Todd Dolinsky

Parameters

<i>nosh</i>	Parameters from input file
<i>com</i>	The communications object
<i>fname</i>	The target XML file name
<i>totEnergy</i>	An array with per-calc total energies (in kT)
<i>qfEnergy</i>	An array with per-calc charge-potential energies (in kT)
<i>qmEnergy</i>	An array with per-calc mobile energies (in kT)
<i>dielEnergy</i>	An array with per-calc polarization energies (in kT)
<i>nenergy</i>	An array containing the number of atoms per-calc
<i>atomEnergy</i>	An array containing per-atom energies (in kT) per calc
<i>nforce</i>	An array containing the number of forces calculated per-calc
<i>atomForce</i>	An array containing per-atom forces per calc

Returns

1 if successful, 0 otherwise

Definition at line 1741 of file [routines.c](#).

8.30.2.54 VEXTERNC int writedataGEOFLOW (int *rank*, NOsh * *nosh*, PBEparm * *pbeparm*)

Write out observables from GEOFLOW calculation to file.

Author

Andrew Stevens

Parameters

<i>rank</i>	Processor rank (if parallel calculation)
<i>nosh</i>	Parameters from input file
<i>pbeparm</i>	Generic PBE parameters

Returns

1 if successful, 0 otherwise

Definition at line 5108 of file [routines.c](#).

8.30.2.55 VEXTERNC int writedataMG (int *rank*, NOsh * *nosh*, PBEparm * *pbeparm*, Vpmg * *pmg*)

Write out observables from MG calculation to file.

Author

Nathan Baker

Parameters

<i>rank</i>	Processor rank (if parallel calculation)
<i>nosh</i>	Parameters from input file
<i>pbeparm</i>	Generic PBE parameters
<i>pmg</i>	MG object

Returns

1 if successful, 0 otherwise

Definition at line 2237 of file [routines.c](#).

8.30.2.56 VEXTERNC int writedataXML (NOsh * *nosh*, Vcom * *com*, const char * *fname*, double *totEnergy*[NOSH_MAXCALC], double *qfEnergy*[NOSH_MAXCALC], double *qmEnergy*[NOSH_MAXCALC], double *dielEnergy*[NOSH_MAXCALC], int *nenergy*[NOSH_MAXCALC], double * *atomEnergy*[NOSH_MAXCALC], int *nforce*[NOSH_MAXCALC], AtomForce * *atomForce*[NOSH_MAXCALC])

Write out information to an XML file.

Author

Todd Dolinsky

Parameters

<i>nosh</i>	Parameters from input file
<i>com</i>	The communications object
<i>fname</i>	The target XML file name
<i>totEnergy</i>	An array with per-calc total energies (in kT)

<i>qfEnergy</i>	An array with per-calc charge-potential energies (in kT)
<i>qmEnergy</i>	An array with per-calc mobile energies (in kT)
<i>dielEnergy</i>	An array with per-calc polarization energies (in kT)
<i>nenergy</i>	An array containing the number of atoms per-calc
<i>atomEnergy</i>	An array containing per-atom energies (in KT) per calc
<i>nforce</i>	An array containing the number of forces calculated per-calc
<i>atomForce</i>	An array containing per-atom forces per calc

Returns

1 if successful, 0 otherwise

Definition at line 1977 of file [routines.c](#).

8.30.2.57 VEXTERNC int writematGEOFLOW (int *rank*, NOsh * *nosh*, PBEparm * *pbeparm*)

Write out operator matrix from GEOFLOW calculation to file.

Author

Andrew Stevens

Parameters

<i>rank</i>	Processor rank (if parallel calculation)
<i>nosh</i>	Parameters from input file
<i>pbeparm</i>	Generic PBE parameters

Returns

1 if successful, 0 otherwise

Definition at line 5117 of file [routines.c](#).

8.30.2.58 VEXTERNC int writematMG (int *rank*, NOsh * *nosh*, PBEparm * *pbeparm*, Vpmg * *pmg*)

Write out operator matrix from MG calculation to file.

Author

Nathan Baker

Parameters

<i>rank</i>	Processor rank (if parallel calculation)
<i>nosh</i>	Parameters from input file
<i>pbeparm</i>	Generic PBE parameters
<i>pmg</i>	MG object

Returns

1 if successful, 0 otherwise

Definition at line 1653 of file [routines.c](#).

Chapter 9

Data Structure Documentation

9.1 _GeoflowInput Struct Reference

Data Fields

- double * **dcel**
- int **ffmodel**
- double **extvalue**
- double * **pqr**
- int **maxstep**
- double **crevalue**
- int **iadi**
- double **tottf**
- double * **ljepsilon**
- double **alpha**
- int **igfin**
- double **epsilons**
- double **epsilonp**
- int **idacsl**
- double **tol**
- int **iterf**
- double **tpb**
- int **itert**
- double **pres**
- double **gama**
- double **tauval**
- double **prob**
- int **vdwdispersion**
- double **sigmas**
- double **density**
- double **epsilonw**

9.1.1 Detailed Description

Definition at line 76 of file [cpbconcz2.h](#).

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

- [src/geoflow/cpbconcz2.h](#)

9.2 _GeoflowOutput Struct Reference

Data Fields

- double **area**
- double **volume**
- double **attint**
- double **sumpot**
- double **totalSolvation**
- double **nonpolarSolvation**
- double **elecSolvation**

9.2.1 Detailed Description

Definition at line 65 of file [cpbconcz2.h](#).

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

- [src/geoflow/cpbconcz2.h](#)

9.3 AtomForce Struct Reference

Structure to hold atomic forces.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/routines.h>
```

Data Fields

- double [ibForce](#) [3]
- double [qfForce](#) [3]
- double [dbForce](#) [3]
- double [sasaForce](#) [3]
- double [savForce](#) [3]
- double [wcaForce](#) [3]

9.3.1 Detailed Description

Structure to hold atomic forces.

Author

Nathan Baker

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

9.3.2 Field Documentation**9.3.2.1 double dbForce[3]**

Dielectric boundary force

Definition at line 86 of file [routines.h](#).

9.3.2.2 double ibForce[3]

Ion-boundary force

Definition at line 84 of file [routines.h](#).

9.3.2.3 double qfForce[3]

Charge-field force

Definition at line 85 of file [routines.h](#).

9.3.2.4 double sasaForce[3]

SASA force (coupled to gamma)

Definition at line 87 of file [routines.h](#).

9.3.2.5 double savForce[3]

SAV force (coupled to press)

Definition at line 88 of file [routines.h](#).

9.3.2.6 double wcaForce[3]

WCA integral force (coupled to bconc)

Definition at line 89 of file [routines.h](#).

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

- [src/routines.h](#)

9.4 Comdata Struct Reference

Data Fields

- char **fname** [100]
- size_t **nx**
- size_t **ny**
- size_t **nz**
- double **xleft**
- double **xright**
- double **yleft**
- double **yright**
- double **zleft**
- double **zright**
- double **deltax**
- double **deltay**
- double **deltaz**
- double **dcel**
- double **pi**
- std::vector< double > **xc**
- std::vector< double > **yc**
- std::vector< double > **zc**

9.4.1 Detailed Description

Definition at line 62 of file [modules.h](#).

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

- [src/geoflow/modules.h](#)

9.5 LJ Struct Reference

Data Fields

- double **tauval**
- double **prob**
- double **vdwdispersion**
- double **sigmas**
- double **roro**
- double **conms**
- double **density**
- double **epsilonw**
- int **ffmodel**

Static Public Attributes

- static const int **iosetar** = 1
- static const int **iosetaa** = 1
- static const int **iwca** = 1

9.5.1 Detailed Description

Definition at line 75 of file [modules.h](#).

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

- [src/geoflow/modules.h](#)

9.6 Mat< T > Class Template Reference

Public Member Functions

- `size_t nx () const`
- `size_t ny () const`
- `size_t nz () const`
- `double hx () const`
- `double hy () const`
- `double hz () const`
- `Mat (size_t nx, size_t ny, size_t nz=1, T a=0)`
- `Mat (size_t nx, size_t ny, size_t nz, double hx, double hy, double hz, T a=0)`
- `Eigen::Matrix< T, Eigen::Dynamic, 1 > & baseInterface ()`
- `const Eigen::Matrix< T, Eigen::Dynamic, 1 > & baseInterface () const`
- `bool equalSize (const Mat< T > &rhs) const`
- `bool equalSpacing (const Mat< T > &rhs) const`
- `bool operator== (const Mat< T > &rhs) const`
- `T & operator() (size_t x, size_t y, size_t z=1)`
- `T operator() (size_t x, size_t y, size_t z=1) const`
- `T & operator[] (size_t i)`
- `T operator[] (size_t i) const`
- `size_t index (size_t x, size_t y, size_t z) const`
- `T * data ()`
- `const T * data () const`
- `size_t size ()`
- `T * end ()`
- `Mat< T > & operator= (T a)`
- `Mat< T > & operator= (Mat a)`
- `Stencil< T > stencilBegin ()`
- `Stencil< T > stencilEnd ()`

Friends

- `struct Stencil< T >`
- `void swap (Mat< T > &a, Mat< T > &b) throw ()`

9.6.1 Detailed Description

```
template<typename T = double>class Mat< T >
```

Definition at line 78 of file [Mat.h](#).

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

- [src/geoflow/Mat.h](#)

9.7 sAPOLparm Struct Reference

Parameter structure for APOL-specific variables from input files.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/generic/apolparm.h>
```

Data Fields

- int [parsed](#)
- double [grid](#) [3]
- int [setgrid](#)
- int [molid](#)
- int [setmolid](#)
- double [bconc](#)
- int [setbconc](#)
- double [sdens](#)
- int [setsdens](#)
- double [dpos](#)
- int [setdpos](#)
- double [press](#)
- int [setpress](#)
- [Vsurf_Meth](#) [srfm](#)
- int [setsrfm](#)
- double [srad](#)
- int [setsrad](#)
- double [swin](#)
- int [setswin](#)
- double [temp](#)
- int [settemp](#)
- double [gamma](#)
- int [setgamma](#)
- [APOLparm_calcEnergy](#) [calcenergy](#)
- int [setcalcenergy](#)
- [APOLparm_calcForce](#) [calcforce](#)
- int [setcalcforce](#)
- double [watsigma](#)
- double [watepsilon](#)
- double [sasa](#)
- double [sav](#)
- double [wcaEnergy](#)
- double [totForce](#) [3]
- int [setwat](#)

9.7.1 Detailed Description

Parameter structure for APOL-specific variables from input files.

Author

David Gohara

Definition at line 129 of file [apolparm.h](#).

9.7.2 Field Documentation

9.7.2.1 double bconc

Vacc sphere density

Definition at line 139 of file [apolparm.h](#).

9.7.2.2 APOLparm_calcEnergy calcenergy

Energy calculation flag

Definition at line 167 of file [apolparm.h](#).

9.7.2.3 APOLparm_calcForce calcforce

Atomic forces calculation

Definition at line 170 of file [apolparm.h](#).

9.7.2.4 double dpos

Atom position offset

Definition at line 145 of file [apolparm.h](#).

9.7.2.5 double gamma

Surface tension for apolar energies/forces (in kJ/mol/A²)

Definition at line 163 of file [apolparm.h](#).

9.7.2.6 double grid[3]

Grid spacing

Definition at line 133 of file [apolparm.h](#).

9.7.2.7 int molid

Molecule ID to perform calculation on

Definition at line 136 of file [apolparm.h](#).

9.7.2.8 int parsed

Flag: Has this structure been filled with anything other than the default values? (0 = no, 1 = yes)

Definition at line 131 of file [apolparm.h](#).

9.7.2.9 double press

Solvent pressure

Definition at line 148 of file [apolparm.h](#).

9.7.2.10 double sasa

Solvent accessible surface area for this calculation

Definition at line 175 of file [apolparm.h](#).

9.7.2.11 double sav

Solvent accessible volume for this calculation

Definition at line 176 of file [apolparm.h](#).

9.7.2.12 double sdens

Vacc sphere density

Definition at line 142 of file [apolparm.h](#).

9.7.2.13 int setbconc

Flag,

See also

[bconc](#)

Definition at line 140 of file [apolparm.h](#).

9.7.2.14 int setcalcenergy

Flag,

See also

[calcenergy](#)

Definition at line 168 of file [apolparm.h](#).

9.7.2.15 int setcalcforce

Flag,

See also

[calcforce](#)

Definition at line 171 of file [apolparm.h](#).

9.7.2.16 int setdpos

Flag,

See also

[dpos](#)

Definition at line 146 of file [apolparm.h](#).

9.7.2.17 int setgamma

Flag,

See also

[gamma](#)

Definition at line 165 of file [apolparm.h](#).

9.7.2.18 int setgrid

Flag,

See also

[grid](#)

Definition at line 134 of file [apolparm.h](#).

9.7.2.19 int setmolid

Flag,

See also

[molid](#)

Definition at line 137 of file [apolparm.h](#).

9.7.2.20 int setpress

Flag,

See also

[press](#)

Definition at line 149 of file [apolparm.h](#).

9.7.2.21 int setsdens

Flag,

See also

[sdens](#)

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

9.7.2.22 int setsrad

Flag,

See also

[srad](#)

Definition at line 155 of file [apolparm.h](#).

9.7.2.23 int setsrfm

Flag,

See also

[srfm](#)

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

9.7.2.24 int setswin

Flag,

See also

[swin](#)

Definition at line 158 of file [apolparm.h](#).

9.7.2.25 int settemp

Flag,

See also

[temp](#)

Definition at line 161 of file [apolparm.h](#).

9.7.2.26 int setwat

Boolean for determining if a water parameter is supplied. Yes = 1, No = 0

Definition at line 180 of file [apolparm.h](#).

9.7.2.27 double srاد

Solvent radius

Definition at line 154 of file [apolparm.h](#).

9.7.2.28 Vsurf_Meth srfm

Surface calculation method

Definition at line 151 of file [apolparm.h](#).

9.7.2.29 double swin

Cubic spline window

Definition at line 157 of file [apolparm.h](#).

9.7.2.30 double temp

Temperature (in K)

Definition at line 160 of file [apolparm.h](#).

9.7.2.31 double totForce[3]

Total forces on x, y, z

Definition at line 178 of file [apolparm.h](#).

9.7.2.32 double watepsilon

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

Definition at line 174 of file [apolparm.h](#).

9.7.2.33 double watsigma

Water oxygen Lennard-Jones radius (A)

Definition at line 173 of file [apolparm.h](#).

9.7.2.34 double wcaEnergy

wcaEnergy

Definition at line 177 of file [apolparm.h](#).

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

- [src/generic/apolparm.h](#)

9.8 sBEMparm Struct Reference

Parameter structure for BEM-specific variables from input files.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/generic/bemparm.h>
```

Data Fields

- [BEMparm_CalcType](#) type
- int [parsed](#)
- [Vchrg_Src](#) chgs
- int [tree_order](#)
- int [settree_order](#)
- int [tree_n0](#)
- int [settree_n0](#)
- double [mac](#)
- int [setmac](#)
- int [nonlintype](#)
- int [setnonlintype](#)

9.8.1 Detailed Description

Parameter structure for BEM-specific variables from input files.

Author

Nathan Baker and Todd Dolinsky and Weihua Geng

Note

If you add/delete/change something in this class, the member functions – especially `BEMparm_copy` – must be modified accordingly

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

9.8.2 Field Documentation

9.8.2.1 Vchrg_Src chgs

Charge source (Charge, Multipole, Induced Dipole, NL Induced. Not currently implemented but should be relatively easy to add in the future (cf Pengyu Ren)

Definition at line 102 of file [bemparm.h](#).

9.8.2.2 double mac

Multipole acceptance criterion (should be between 0 and 1)

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

9.8.2.3 int nonlotype

Linearity Type Method to be used

Definition at line 110 of file [bemparm.h](#).

9.8.2.4 int parsed

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

Definition at line 99 of file [bemparm.h](#).

9.8.2.5 int setmac

Flag,

See also

[mac](#)

Definition at line 109 of file [bemparm.h](#).

9.8.2.6 int setnonlotype

Flag,

See also

[nonlotype](#)

Definition at line 111 of file [bemparm.h](#).

9.8.2.7 int settree_n0

Flag,

See also

[tree_npart](#)

Definition at line 107 of file [bemparm.h](#).

9.8.2.8 int settree_order

Flag,

See also

[tree_order](#)

Definition at line 105 of file [bemparm.h](#).

9.8.2.9 int tree_n0

Number of particles per leaf of the tree

Definition at line 106 of file [bemparm.h](#).

9.8.2.10 int tree_order

User-defined order for the treecode expansion

Definition at line 104 of file [bemparm.h](#).

9.8.2.11 BEMparm_CalcType type

What type of BEM calculation?

Definition at line 98 of file [bemparm.h](#).

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

- [src/generic/bemparm.h](#)

9.9 sFEMparm Struct Reference

Parameter structure for FEM-specific variables from input files.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/generic/femparm.h>
```

Data Fields

- int [parsed](#)
- [FEMparm_CalcType](#) type
- int [settype](#)
- double [glen](#) [3]

- int [setglen](#)
- double [etol](#)
- int [setetol](#)
- [FEMparm_EtolType](#) [ekey](#)
- int [setekey](#)
- [FEMparm_EstType](#) [akeyPRE](#)
- int [setakeyPRE](#)
- [FEMparm_EstType](#) [akeySOLVE](#)
- int [setakeySOLVE](#)
- int [targetNum](#)
- int [settargetNum](#)
- double [targetRes](#)
- int [settargetRes](#)
- int [maxsolve](#)
- int [setmaxsolve](#)
- int [maxvert](#)
- int [setmaxvert](#)
- int [pkey](#)
- int [useMesh](#)
- int [meshID](#)

9.9.1 Detailed Description

Parameter structure for FEM-specific variables from input files.

Author

Nathan Baker

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

9.9.2 Field Documentation

9.9.2.1 [FEMparm_EstType](#) [akeyPRE](#)

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

Definition at line [148](#) of file [femparm.h](#).

9.9.2.2 [FEMparm_EstType](#) [akeySOLVE](#)

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

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

9.9.2.3 [FEMparm_EtolType](#) [ekey](#)

Adaptive refinement interpretation of error tolerance

Definition at line [145](#) of file [femparm.h](#).

9.9.2.4 double etol

Error tolerance for refinement; interpretation depends on the adaptive refinement method chosen

Definition at line 142 of file [femparm.h](#).

9.9.2.5 double glen[3]

Domain side lengths (in Å)

Definition at line 140 of file [femparm.h](#).

9.9.2.6 int maxsolve

Maximum number of solve-estimate-refine cycles

Definition at line 165 of file [femparm.h](#).

9.9.2.7 int maxvert

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

Definition at line 167 of file [femparm.h](#).

9.9.2.8 int meshID

External finite element mesh ID (if used)

Definition at line 174 of file [femparm.h](#).

9.9.2.9 int parsed

Flag: Has this structure been filled with anything other than * the default values? (0 = no, 1 = yes)

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

9.9.2.10 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 170 of file [femparm.h](#).

9.9.2.11 int setakeyPRE

Boolean

Definition at line 151 of file [femparm.h](#).

9.9.2.12 int setakeySOLVE

Boolean

Definition at line 154 of file [femparm.h](#).

9.9.2.13 int setekey

Boolean

Definition at line 147 of file [femparm.h](#).

9.9.2.14 int setetol

Boolean

Definition at line 144 of file [femparm.h](#).

9.9.2.15 int setglen

Boolean

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

9.9.2.16 int setmaxsolve

Boolean

Definition at line 166 of file [femparm.h](#).

9.9.2.17 int setmaxvert

Boolean

Definition at line 169 of file [femparm.h](#).

9.9.2.18 int settargetNum

Boolean

Definition at line 159 of file [femparm.h](#).

9.9.2.19 int settargetRes

Boolean

Definition at line 164 of file [femparm.h](#).

9.9.2.20 int settype

Boolean

Definition at line 139 of file [femparm.h](#).

9.9.2.21 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 155 of file [femparm.h](#).

9.9.2.22 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 160 of file [femparm.h](#).

9.9.2.23 FEMparm_CalcType type

Calculation type

Definition at line 138 of file [femparm.h](#).

9.9.2.24 int useMesh

Indicates whether we use external finite element mesh

Definition at line 173 of file [femparm.h](#).

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

- [src/generic/femparm.h](#)

9.10 sGEOFLOWparm Struct Reference

Parameter structure for GEOFLOW-specific variables from input files.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/generic/geoflowparm.h>
```

Data Fields

- [GEOFLOWparm_CalcType](#) type
- int [parsed](#)
- int **vdw**
- int **setvdw**

9.10.1 Detailed Description

Parameter structure for GEOFLOW-specific variables from input files.

Author

Andrew Stevens, Kyle Monson

Note

If you add/delete/change something in this class, the member functions – especially GEOFLOWparm_copy – must be modified accordingly

Definition at line 97 of file [geoflowparm.h](#).

9.10.2 Field Documentation**9.10.2.1 int parsed**

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

Definition at line 100 of file [geoflowparm.h](#).

9.10.2.2 GEOFLOWparm_CalcType type

What type of GEOFLOW calculation?

Definition at line 99 of file [geoflowparm.h](#).

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

- [src/generic/geoflowparm.h](#)

9.11 sMGparm Struct Reference

Parameter structure for MG-specific variables from input files.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/generic/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 [setgcent](#)
- [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 [nonlotype](#)
- int [setnonlotype](#)
- int [method](#)
- int [setmethod](#)
- int [useAqua](#)
- int [setUseAqua](#)

9.11.1 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 114 of file [mgparm.h](#).

9.11.2 Field Documentation

9.11.2.1 int async

Processor ID for asynchronous calculation

Definition at line 186 of file [mgparm.h](#).

9.11.2.2 double ccenter[3]

Coarse grid center.

Definition at line 157 of file [mgparm.h](#).

9.11.2.3 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 158 of file [mgparm.h](#).

9.11.2.4 MGparm_CentMeth cmeth

Coarse grid centering method

Definition at line 156 of file [mgparm.h](#).

9.11.2.5 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 138 of file [mgparm.h](#).

9.11.2.6 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 146 of file [mgparm.h](#).

9.11.2.7 double cglen[3]

Coarse grid side lengths

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

9.11.2.8 Vchrg_Meth chgm

Charge discretization method

Definition at line 122 of file [mgparm.h](#).

9.11.2.9 Vchrg_Src chgs

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

Definition at line [124](#) of file [mgparm.h](#).

9.11.2.10 MGparm_CentMeth cmeth

Centering method

Definition at line [137](#) of file [mgparm.h](#).

9.11.2.11 int dime[3]

Grid dimensions

Definition at line [120](#) of file [mgparm.h](#).

9.11.2.12 double etol

User-defined error tolerance

Definition at line [131](#) of file [mgparm.h](#).

9.11.2.13 double fcenter[3]

Fine grid center.

Definition at line [163](#) of file [mgparm.h](#).

9.11.2.14 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 [164](#) of file [mgparm.h](#).

9.11.2.15 MGparm_CentMeth fcmeth

Fine grid centering method

Definition at line [162](#) of file [mgparm.h](#).

9.11.2.16 double fglen[3]

Fine grid side lengths

Definition at line [154](#) of file [mgparm.h](#).

9.11.2.17 double glen[3]

Grid side lengths.

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

9.11.2.18 double grid[3]

Grid spacings

Definition at line 133 of file [mgparm.h](#).

9.11.2.19 int method

Solver Method

Definition at line 192 of file [mgparm.h](#).

9.11.2.20 int nlev

Levels in multigrid hierarchy

Deprecated Just ignored now

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

9.11.2.21 int nonlotype

Linearity Type Method to be used

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

9.11.2.22 double ofrac

Overlap fraction between procs

Definition at line 184 of file [mgparm.h](#).

9.11.2.23 int parsed

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

Definition at line 117 of file [mgparm.h](#).

9.11.2.24 double partDisjCenter[3]

This gives the center of the disjoint partitions

Definition at line 171 of file [mgparm.h](#).

9.11.2.25 double partDisjLength[3]

This gives the lengths of the disjoint partitions

Definition at line 173 of file [mgparm.h](#).

9.11.2.26 int partDisjOwnSide[6]

Tells whether the boundary points are ours (1) or not (0)

Definition at line [175](#) of file [mgparm.h](#).

9.11.2.27 int pdime[3]

Grid of processors to be used in calculation

Definition at line [178](#) of file [mgparm.h](#).

9.11.2.28 int proc_rank

Rank of this processor

Definition at line [180](#) of file [mgparm.h](#).

9.11.2.29 int proc_size

Total number of processors

Definition at line [182](#) of file [mgparm.h](#).

9.11.2.30 int setasynch

Flag,

See also

[asynch](#)

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

9.11.2.31 int setcgcent

Flag,

See also

[ccmeth](#)

Definition at line [161](#) of file [mgparm.h](#).

9.11.2.32 int setcglen

Flag,

See also

[cglen](#)

Definition at line [153](#) of file [mgparm.h](#).

9.11.2.33 int setchgm

Flag,

See also

[chgm](#)

Definition at line 123 of file [mgparm.h](#).

9.11.2.34 int setdime

Flag,

See also

[dime](#)

Definition at line 121 of file [mgparm.h](#).

9.11.2.35 int setetol

Flag,

See also

[etol](#)

Definition at line 132 of file [mgparm.h](#).

9.11.2.36 int setfgcent

Flag,

See also

[fcmeth](#)

Definition at line 167 of file [mgparm.h](#).

9.11.2.37 int setfglen

Flag,

See also

[fglen](#)

Definition at line 155 of file [mgparm.h](#).

9.11.2.38 int setgcent

Flag,

See also

[cmeth](#)

Definition at line [149](#) of file [mgparm.h](#).

9.11.2.39 int setglen

Flag,

See also

[glen](#)

Definition at line [136](#) of file [mgparm.h](#).

9.11.2.40 int setgrid

Flag,

See also

[grid](#)

Definition at line [134](#) of file [mgparm.h](#).

9.11.2.41 int setmethod

Flag,

See also

[method](#)

Definition at line [193](#) of file [mgparm.h](#).

9.11.2.42 int setnlev

Flag,

See also

[nlev](#)

Definition at line [130](#) of file [mgparm.h](#).

9.11.2.43 int setnonlotype

Flag,

See also

[nonlotype](#)

Definition at line 190 of file [mgparm.h](#).

9.11.2.44 int setofrac

Flag,

See also

[ofrac](#)

Definition at line 185 of file [mgparm.h](#).

9.11.2.45 int setpdime

Flag,

See also

[pdime](#)

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

9.11.2.46 int setrank

Flag,

See also

[proc_rank](#)

Definition at line 181 of file [mgparm.h](#).

9.11.2.47 int setsize

Flag,

See also

[proc_size](#)

Definition at line 183 of file [mgparm.h](#).

9.11.2.48 int setUseAqua

Flag,

See also

[useAqua](#)

Definition at line 196 of file [mgparm.h](#).

9.11.2.49 MGparm_CalcType type

What type of MG calculation?

Definition at line 116 of file [mgparm.h](#).

9.11.2.50 int useAqua

Enable use of lpbe/aqua

Definition at line 195 of file [mgparm.h](#).

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

- [src/generic/mgparm.h](#)

9.12 sNOsh Struct Reference

Class for parsing fixed format input files.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/generic/nosh.h>
```

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]

9.12.1 Detailed Description

Class for parsing fixed format input files.

Author

Nathan Baker

Definition at line 189 of file [nosh.h](#).

9.12.2 Field Documentation

9.12.2.1 [Valist*](#) [alist](#)[NOSH_MAXMOL]

Molecules for calculation (can be used in setting mesh centers)

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

9.12.2.2 [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 202 of file [nosh.h](#).

9.12.2.3 int apol2calc[NOSH_MAXCALC]

(see elec2calc)

Definition at line 223 of file [nosh.h](#).

9.12.2.4 char apolname[NOSH_MAXCALC][VMAX_ARGLEN]

Optional user-specified name for APOLAR statement

Definition at line 263 of file [nosh.h](#).

9.12.2.5 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 211 of file [nosh.h](#).

9.12.2.6 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 191 of file [nosh.h](#).

9.12.2.7 Vdata_Format chargefmt[NOSH_MAXMOL]

Charge maps fileformats

Definition at line 249 of file [nosh.h](#).

9.12.2.8 char chargepath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to charge map files

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

9.12.2.9 Vdata_Format dielfmt[NOSH_MAXMOL]

Dielectric maps file formats

Definition at line 240 of file [nosh.h](#).

9.12.2.10 char dielXpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to x-shifted dielectric map files

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

9.12.2.11 char dielYpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to y-shifted dielectric map files

Definition at line 236 of file [nosh.h](#).

9.12.2.12 char dielZpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to z-shifted dielectric map files

Definition at line 238 of file [nosh.h](#).

9.12.2.13 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 196 of file [nosh.h](#).

9.12.2.14 int elec2calc[NOSH_MAXCALC]

A mapping between ELEC statements which appear in the input file and calc objects stored above. Since we allow both normal and focused multigrid, there isn't a 1-to-1 correspondence between ELEC statements and actual calculations. This can really confuse operations which work on specific calculations further down the road (like PRINT). Therefore this array is the initial point of entry for any calculation-specific operation. It points to a specific entry in the calc array.

Definition at line 215 of file [nosh.h](#).

9.12.2.15 char elecname[NOSH_MAXCALC][VMAX_ARGLEN]

Optional user-specified name for ELEC statement

Definition at line 261 of file [nosh.h](#).

9.12.2.16 int gotparm

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

Definition at line 230 of file [nosh.h](#).

9.12.2.17 int ispara

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

Definition at line 208 of file [nosh.h](#).

9.12.2.18 Vdata_Format kappafmt[NOSH_MAXMOL]

Kappa maps file formats

Definition at line 243 of file [nosh.h](#).

9.12.2.19 char kappaopath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to kappa map files

Definition at line 242 of file [nosh.h](#).

9.12.2.20 Vdata_Format meshfmt[NOSH_MAXMOL]

Mesh fileformats

Definition at line [252](#) of file [nosh.h](#).

9.12.2.21 char meshpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to mesh files

Definition at line [251](#) of file [nosh.h](#).

9.12.2.22 NOsh_MolFormat molfmt[NOSH_MAXMOL]

Mol files formats

Definition at line [227](#) of file [nosh.h](#).

9.12.2.23 char molpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to mol files

Definition at line [226](#) of file [nosh.h](#).

9.12.2.24 int napol

The number of apolar statements in the input file and in the apolar array

Definition at line [205](#) of file [nosh.h](#).

9.12.2.25 int ncalc

The number of calculations in the calc array

Definition at line [194](#) of file [nosh.h](#).

9.12.2.26 int ncharge

Number of charge maps

Definition at line [247](#) of file [nosh.h](#).

9.12.2.27 int ndiel

Number of dielectric maps

Definition at line [233](#) of file [nosh.h](#).

9.12.2.28 int nelec

The number of elec statements in the input file and in the elec array

Definition at line [199](#) of file [nosh.h](#).

9.12.2.29 int nkappa

Number of kappa maps

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

9.12.2.30 int nmesh

Number of meshes

Definition at line 250 of file [nosh.h](#).

9.12.2.31 int nmol

Number of molecules

Definition at line 225 of file [nosh.h](#).

9.12.2.32 int npot

Number of potential maps

Definition at line 244 of file [nosh.h](#).

9.12.2.33 int nprint

How many print sections?

Definition at line 253 of file [nosh.h](#).

9.12.2.34 NOsh_ParmFormat parmfmt

Parm file format

Definition at line 232 of file [nosh.h](#).

9.12.2.35 char parmpath[VMAX_ARGLEN]

Paths to parm file

Definition at line 231 of file [nosh.h](#).

9.12.2.36 int parsed

Have we parsed an input file yet?

Definition at line 260 of file [nosh.h](#).

9.12.2.37 Vdata_Format potfmt[NOSH_MAXMOL]

Potential maps file formats

Definition at line 246 of file [nosh.h](#).

9.12.2.38 char potpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to potential map files

Definition at line 245 of file [nosh.h](#).

9.12.2.39 int printcalc[NOSH_MAXPRINT][NOSH_MAXPOP]

ELEC id (see elec2calc)

Definition at line 257 of file [nosh.h](#).

9.12.2.40 int printnarg[NOSH_MAXPRINT]

How many arguments in energy list

Definition at line 256 of file [nosh.h](#).

9.12.2.41 int printop[NOSH_MAXPRINT][NOSH_MAXPOP]

Operation id (0 = add, 1 = subtract)

Definition at line 258 of file [nosh.h](#).

9.12.2.42 NOsh_PrintType printwhat[NOSH_MAXPRINT]

What do we print:

- 0 = energy,
- 1 = force

Definition at line 254 of file [nosh.h](#).

9.12.2.43 int proc_rank

Processor rank in parallel calculation

Definition at line 209 of file [nosh.h](#).

9.12.2.44 int proc_size

Number of processors in parallel calculation

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

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

- [src/generic/nosh.h](#)

9.13 sNOsh_calc Struct Reference

Calculation class for use when parsing fixed format input files.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/generic/nosh.h>
```

Data Fields

- [MGparm](#) * [mgparm](#)
- [FEMparm](#) * [femparm](#)
- [BEMparm](#) * [bemparm](#)
- [GEOFLOWparm](#) * [geoflowparm](#)
- [PBEparm](#) * [pbeparm](#)
- [APOLparm](#) * [apolparm](#)
- [NOsh_CalcType](#) [calctype](#)

9.13.1 Detailed Description

Calculation class for use when parsing fixed format input files.

Author

Nathan Baker

Definition at line 168 of file [nosh.h](#).

9.13.2 Field Documentation

9.13.2.1 [APOLparm](#)* [apolparm](#)

Non-polar parameters

Definition at line 174 of file [nosh.h](#).

9.13.2.2 [BEMparm](#)* [bemparm](#)

boundary element (tabi) parameters

Definition at line 171 of file [nosh.h](#).

9.13.2.3 [NOsh_CalcType](#) [calctype](#)

Calculation type

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

9.13.2.4 [FEMparm](#)* [femparm](#)

Finite element parameters

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

9.13.2.5 MGparm* mgparm

Multigrid parameters

Definition at line 169 of file [nosh.h](#).

9.13.2.6 PBEparm* pbeparm

Generic PBE parameters

Definition at line 173 of file [nosh.h](#).

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

- [src/generic/nosh.h](#)

9.14 sPBEparm Struct Reference

Parameter structure for PBE variables from input files.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/generic/pbeparm.h>
```

Data Fields

- int [molid](#)
- int [setmolid](#)
- int [useDielMap](#)
- int [dielMapID](#)
- int [useKappaMap](#)
- int [kappaMapID](#)
- int [usePotMap](#)
- int [potMapID](#)
- int [useChargeMap](#)
- int [chargeMapID](#)
- [Vhal_PBEType](#) [pbetype](#)
- int [setpbetype](#)
- [Vbcfl](#) [bcfl](#)
- int [setbcfl](#)
- int [nion](#)
- int [setnion](#)
- double [ionq](#) [MAXION]
- double [ionc](#) [MAXION]
- double [ionr](#) [MAXION]
- int [setion](#) [MAXION]
- double [pdie](#)
- int [setpdie](#)
- double [sdens](#)
- int [setsdens](#)
- double [sdie](#)
- int [setsdie](#)

- [Vsurf_Meth](#) srfm
- int [setsrfm](#)
- double [srad](#)
- int [setsrad](#)
- double [swin](#)
- int [setswin](#)
- double [temp](#)
- int [settemp](#)
- double [smsize](#)
- int [setsmsize](#)
- double [smvolume](#)
- int [setsmvolume](#)
- [PBEparm_calcEnergy](#) calcenergy
- int [setcalcenergy](#)
- [PBEparm_calcForce](#) calcforce
- int [setcalcforce](#)
- double [zmem](#)
- int [setzmem](#)
- double [Lmem](#)
- int [setLmem](#)
- double [mdie](#)
- int [setmdie](#)
- double [memv](#)
- int [setmemv](#)
- int [numwrite](#)
- char [writestem](#) [PBEPARM_MAXWRITE][VMAX_ARGLEN]
- [Vdata_Type](#) writetype [PBEPARM_MAXWRITE]
- [Vdata_Format](#) writefmt [PBEPARM_MAXWRITE]
- int [writemat](#)
- int [setwritemat](#)
- char [writematstem](#) [VMAX_ARGLEN]
- int [writematflag](#)
- int [parsed](#)

9.14.1 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 117 of file [pbeparm.h](#).

9.14.2 Field Documentation

9.14.2.1 Vbcfl bcfl

Boundary condition method

Definition at line 136 of file [pbeparm.h](#).

9.14.2.2 PBEparm_calcEnergy calcenergy

Energy calculation flag

Definition at line 165 of file [pbeparm.h](#).

9.14.2.3 PBEparm_calcForce calcforce

Atomic forces calculation

Definition at line 167 of file [pbeparm.h](#).

9.14.2.4 int chargeMapID

Charge distribution map ID (if used)

Definition at line 133 of file [pbeparm.h](#).

9.14.2.5 int dielMapID

Dielectric map ID (if used)

Definition at line 123 of file [pbeparm.h](#).

9.14.2.6 double ionc[MAXION]

Counterion concentrations (in M)

Definition at line 141 of file [pbeparm.h](#).

9.14.2.7 double ionq[MAXION]

Counterion charges (in e)

Definition at line 140 of file [pbeparm.h](#).

9.14.2.8 double ionr[MAXION]

Counterion radii (in Å)

Definition at line 142 of file [pbeparm.h](#).

9.14.2.9 int kappaMapID

Kappa map ID (if used)

Definition at line 126 of file [pbeparm.h](#).

9.14.2.10 double Lmem

membrane width

Definition at line 176 of file [pbeparm.h](#).

9.14.2.11 double mdie

membrane dielectric constant

Definition at line 178 of file [pbeparm.h](#).

9.14.2.12 double memv

Membrane potential

Definition at line 180 of file [pbeparm.h](#).

9.14.2.13 int molid

Molecule ID to perform calculation on

Definition at line 119 of file [pbeparm.h](#).

9.14.2.14 int nion

Number of counterion species

Definition at line 138 of file [pbeparm.h](#).

9.14.2.15 int numwrite

Number of write statements encountered

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

9.14.2.16 int parsed

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

Definition at line 201 of file [pbeparm.h](#).

9.14.2.17 Vhal_PBEType pbetype

Which version of the PBE are we solving?

Definition at line 134 of file [pbeparm.h](#).

9.14.2.18 double pdie

Solute dielectric

Definition at line [144](#) of file [pbeparm.h](#).

9.14.2.19 int potMapID

Kappa map ID (if used)

Definition at line [129](#) of file [pbeparm.h](#).

9.14.2.20 double sdens

Vacc sphere density

Definition at line [146](#) of file [pbeparm.h](#).

9.14.2.21 double sdie

Solvent dielectric

Definition at line [148](#) of file [pbeparm.h](#).

9.14.2.22 int setbcfl

Flag,

See also

[bcfl](#)

Definition at line [137](#) of file [pbeparm.h](#).

9.14.2.23 int setcalcenergy

Flag,

See also

[calcenergy](#)

Definition at line [166](#) of file [pbeparm.h](#).

9.14.2.24 int setcalcforce

Flag,

See also

[calcforce](#)

Definition at line [168](#) of file [pbeparm.h](#).

9.14.2.25 int setion[MAXION]

Flag,

See also

[ionq](#)

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

9.14.2.26 int setLmem

Flag

Definition at line 177 of file [pbeparm.h](#).

9.14.2.27 int setmdie

Flag

Definition at line 179 of file [pbeparm.h](#).

9.14.2.28 int setmemv

Flag

Definition at line 181 of file [pbeparm.h](#).

9.14.2.29 int setmolid

Flag,

See also

[molid](#)

Definition at line 120 of file [pbeparm.h](#).

9.14.2.30 int setnion

Flag,

See also

[nion](#)

Definition at line 139 of file [pbeparm.h](#).

9.14.2.31 int setpbetype

Flag,

See also

[pbetype](#)

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

9.14.2.32 int setpdie

Flag,

See also

[pdie](#)

Definition at line [145](#) of file [pbeparm.h](#).

9.14.2.33 int setsdens

Flag,

See also

[sdens](#)

Definition at line [147](#) of file [pbeparm.h](#).

9.14.2.34 int setsdie

Flag,

See also

[sdie](#)

Definition at line [149](#) of file [pbeparm.h](#).

9.14.2.35 int setsmsize

Flag,

See also

[temp](#)

Definition at line [160](#) of file [pbeparm.h](#).

9.14.2.36 int setsmvolume

Flag,

See also

[temp](#)

Definition at line [163](#) of file [pbeparm.h](#).

9.14.2.37 int setsrad

Flag,

See also

[srad](#)

Definition at line 153 of file [pbeparm.h](#).

9.14.2.38 int setsrfm

Flag,

See also

[srfm](#)

Definition at line 151 of file [pbeparm.h](#).

9.14.2.39 int setswin

Flag,

See also

[swin](#)

Definition at line 155 of file [pbeparm.h](#).

9.14.2.40 int settemp

Flag,

See also

[temp](#)

Definition at line 157 of file [pbeparm.h](#).

9.14.2.41 int setwritemat

Flag,

See also

[writemat](#)

Definition at line 194 of file [pbeparm.h](#).

9.14.2.42 int setzmem

Flag

Definition at line 175 of file [pbeparm.h](#).

9.14.2.43 double smsize

SMPBE size

Definition at line 159 of file [pbeparm.h](#).

9.14.2.44 double smvolume

SMPBE size

Definition at line 162 of file [pbeparm.h](#).

9.14.2.45 double srad

Solvent radius

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

9.14.2.46 Vsurf_Meth srfm

Surface calculation method

Definition at line 150 of file [pbeparm.h](#).

9.14.2.47 double swin

Cubic spline window

Definition at line 154 of file [pbeparm.h](#).

9.14.2.48 double temp

Temperature (in K)

Definition at line 156 of file [pbeparm.h](#).

9.14.2.49 int useChargeMap

Indicates whether we use an external charge distribution map

Definition at line 131 of file [pbeparm.h](#).

9.14.2.50 int useDielMap

Indicates whether we use external dielectric maps (note plural)

Definition at line 121 of file [pbeparm.h](#).

9.14.2.51 int useKappaMap

Indicates whether we use an external kappa map

Definition at line 124 of file [pbeparm.h](#).

9.14.2.52 int usePotMap

Indicates whether we use an external kappa map

Definition at line 127 of file [pbeparm.h](#).

9.14.2.53 Vdata_Format writefmt[PBEPARM_MAXWRITE]

File format to write data in

Definition at line 189 of file [pbeparm.h](#).

9.14.2.54 int writemat

Write out the operator matrix?

- 0 => no
- 1 => yes

Definition at line 191 of file [pbeparm.h](#).

9.14.2.55 int writematflag

What matrix should we write:

- 0 => Poisson (differential operator)
- 1 => Poisson-Boltzmann operator linearized around solution (if applicable)

Definition at line 196 of file [pbeparm.h](#).

9.14.2.56 char writematstem[VMAX_ARGLEN]

File stem to write mat

Definition at line 195 of file [pbeparm.h](#).

9.14.2.57 char writestem[PBEPARM_MAXWRITE][VMAX_ARGLEN]

File stem to write data to

Definition at line 186 of file [pbeparm.h](#).

- ## Data Fields

- ### 9.15.1 Detailed Description

Definition at line 76 of file Mat.h.

- `src/geoflow/Mat.h`

9.16 sVacc Struct Reference

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/generic/vacc.h>
```

Data Fields

- Vmem * [mem](#)
- Valist * [alist](#)
- Vclist * [clist](#)
- int * [atomFlags](#)
- VaccSurf * [refSphere](#)
- VaccSurf ** [surf](#)
- Vset [acc](#)
- double [surf_density](#)

9.16.1 Detailed Description

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

Author

Nathan Baker

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

9.16.2 Field Documentation

9.16.2.1 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 [120](#) of file [vacc.h](#).

9.16.2.2 Valist* alist

Valist structure for list of atoms

Definition at line [111](#) of file [vacc.h](#).

9.16.2.3 int* atomFlags

Array of boolean flags of length Valist_getNumberAtoms(thee->alist) to prevent double-counting atoms during calculations

Definition at line [113](#) of file [vacc.h](#).

9.16.2.4 Vclist* clist

Vclist structure for atom cell list

Definition at line [112](#) of file [vacc.h](#).

9.16.2.5 Vmem* mem

Memory management object for this class

Definition at line [110](#) of file [vacc.h](#).

9.16.2.6 VaccSurf* refSphere

Reference sphere for SASA calculations

Definition at line 116 of file [vacc.h](#).

9.16.2.7 VaccSurf** surf

Array of surface points for each atom; is not initialized until needed (test against VNULL to determine initialization state)

Definition at line 117 of file [vacc.h](#).

9.16.2.8 double surf_density

Minimum solvent accessible surface point density (in pts/A²)

Definition at line 122 of file [vacc.h](#).

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

- [src/generic/vacc.h](#)

9.17 sVaccSurf Struct Reference

Surface object list of per-atom surface points.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/generic/vacc.h>
```

Data Fields

- Vmem * [mem](#)
- double * [xpts](#)
- double * [ypts](#)
- double * [zpts](#)
- char * [bpts](#)
- double [area](#)
- int [npts](#)
- double [probe_radius](#)

9.17.1 Detailed Description

Surface object list of per-atom surface points.

Author

Nathan Baker

Definition at line 84 of file [vacc.h](#).

9.17.2 Field Documentation

9.17.2.1 double area

Area spanned by these points

Definition at line 91 of file [vacc.h](#).

9.17.2.2 char* bpts

Array of booleans indicating whether a point is (1) or is not (0) part of the surface

Definition at line 89 of file [vacc.h](#).

9.17.2.3 Vmem* mem

Memory object

Definition at line 85 of file [vacc.h](#).

9.17.2.4 int npts

Length of thee->xpts, ypts, zpts arrays

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

9.17.2.5 double probe_radius

Probe radius (A) with which this surface was constructed

Definition at line 93 of file [vacc.h](#).

9.17.2.6 double* xpts

Array of point x-locations

Definition at line 86 of file [vacc.h](#).

9.17.2.7 double* ypts

Array of point y-locations

Definition at line 87 of file [vacc.h](#).

9.17.2.8 double* zpts

Array of point z-locations

Definition at line 88 of file [vacc.h](#).

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

- [src/generic/vacc.h](#)

9.18 sValist Struct Reference

Container class for list of atom objects.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/generic/valist.h>
```

Data Fields

- int [number](#)
- double [center](#) [3]
- double [mincrd](#) [3]
- double [maxcrd](#) [3]
- double [maxrad](#)
- double [charge](#)
- [Vatom](#) * [atoms](#)
- [Vmem](#) * [vmem](#)

9.18.1 Detailed Description

Container class for list of atom objects.

Author

Nathan Baker

Definition at line 78 of file [valist.h](#).

9.18.2 Field Documentation

9.18.2.1 [Vatom](#)* [atoms](#)

Atom list

Definition at line 86 of file [valist.h](#).

9.18.2.2 double [center](#)[3]

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

Definition at line 81 of file [valist.h](#).

9.18.2.3 double [charge](#)

Net charge

Definition at line 85 of file [valist.h](#).

9.18.2.4 double [maxcrd](#)[3]

Maximum coordinates

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

9.18.2.5 double maxrad

Maximum radius

Definition at line 84 of file [valist.h](#).

9.18.2.6 double mincrd[3]

Minimum coordinates

Definition at line 82 of file [valist.h](#).

9.18.2.7 int number

Number of atoms in list

Definition at line 80 of file [valist.h](#).

9.18.2.8 Vmem* vmem

Memory management object

Definition at line 87 of file [valist.h](#).

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

- [src/generic/valist.h](#)

9.19 sVatom Struct Reference

Contains public data members for Vatom class/module.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/generic/vatom.h>
```

Data Fields

- double [position](#) [3]
- double [radius](#)
- double [charge](#)
- double [partID](#)
- double [epsilon](#)
- int [id](#)
- char [resName](#) [VMAX_RECLEN]
- char [atomName](#) [VMAX_RECLEN]

9.19.1 Detailed Description

Contains public data members for Vatom class/module.

Author

Nathan Baker, David Gohara, Mike Schneiders

Definition at line 84 of file [vatom.h](#).

9.19.2 Field Documentation**9.19.2.1 char atomName[VMAX_RECLEN]**

Atom name from PDB/PDR file

Definition at line 98 of file [vatom.h](#).

9.19.2.2 double charge

Atomic charge

Definition at line 88 of file [vatom.h](#).

9.19.2.3 double epsilon

Epsilon value for WCA calculations

Definition at line 91 of file [vatom.h](#).

9.19.2.4 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 93 of file [vatom.h](#).

9.19.2.5 double partID

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

Definition at line 89 of file [vatom.h](#).

9.19.2.6 double position[3]

Atomic position

Definition at line 86 of file [vatom.h](#).

9.19.2.7 double radius

Atomic radius

Definition at line 87 of file [vatom.h](#).

9.19.2.8 char resName[VMAX_RECLEN]

Residue name from PDB/PQR file

Definition at line 97 of file [vatom.h](#).

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

- [src/generic/vatom.h](#)

9.20 sVclist Struct Reference

Atom cell list.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/generic/vclist.h>
```

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]

9.20.1 Detailed Description

Atom cell list.

Author

Nathan Baker

Definition at line 117 of file [vclist.h](#).

9.20.2 Field Documentation

9.20.2.1 Valist* alist

Original Valist structure for list of atoms

Definition at line 120 of file [vclist.h](#).

9.20.2.2 VclistCell* cells

Cell array of length thee->n

Definition at line 125 of file [vclist.h](#).

9.20.2.3 double lower_corner[VAPBS_DIM]

Hash table grid corner

Definition at line 126 of file [vclist.h](#).

9.20.2.4 double max_radius

Maximum probe radius

Definition at line 124 of file [vclist.h](#).

9.20.2.5 Vclist_DomainMode mode

How the cell list was constructed

Definition at line 121 of file [vclist.h](#).

9.20.2.6 int n

$n = n_x * n_z * n_y$

Definition at line 123 of file [vclist.h](#).

9.20.2.7 int npts[VAPBS_DIM]

Hash table grid dimensions

Definition at line 122 of file [vclist.h](#).

9.20.2.8 double spacs[VAPBS_DIM]

Hash table grid spacings

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

9.20.2.9 double upper_corner[VAPBS_DIM]

Hash table grid corner

Definition at line 127 of file [vclist.h](#).

9.20.2.10 Vmem* vmem

Memory management object for this class

Definition at line 119 of file [vclist.h](#).

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

- [src/generic/vclist.h](#)

9.21 sVclistCell Struct Reference

Atom cell list cell.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/generic/vclist.h>
```

Data Fields

- [Vatom](#) ** [atoms](#)
- int [natoms](#)

9.21.1 Detailed Description

Atom cell list cell.

Author

Nathan Baker

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

9.21.2 Field Documentation

9.21.2.1 Vatom** atoms

Array of atom objects associated with this cell

Definition at line [102](#) of file [vclist.h](#).

9.21.2.2 int natoms

Length of the->atoms array

Definition at line [103](#) of file [vclist.h](#).

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

- [src/generic/vclist.h](#)

9.22 sVcsm Struct Reference

Charge-simplex map class.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/fem/vcsm.h>
```

Data Fields

- [Valist](#) * [alist](#)

- int [natom](#)
- Gem * [gm](#)
- int ** [sqm](#)
- int * [nsqm](#)
- int [nsimp](#)
- int [msimp](#)
- int ** [qsm](#)
- int * [nqsm](#)
- int [initFlag](#)
- Vmem * [vmem](#)

9.22.1 Detailed Description

Charge-simplex map class.

Author

Nathan Baker

Definition at line [89](#) of file [vcsn.h](#).

9.22.2 Field Documentation

9.22.2.1 Valist* alist

Atom (charge) list

Definition at line [91](#) of file [vcsn.h](#).

9.22.2.2 Gem* gm

Grid manager (container class for master vertex and simplex lists as well as prolongation operator for updating after refinement)

Definition at line [94](#) of file [vcsn.h](#).

9.22.2.3 int initFlag

Indicates whether the maps have been initialized yet

Definition at line [112](#) of file [vcsn.h](#).

9.22.2.4 int msimp

The maximum number of entries that can be accomodated by sqm or nsqm – saves on realloc's

Definition at line [107](#) of file [vcsn.h](#).

9.22.2.5 int natom

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

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

9.22.2.6 int* nqsm

The length of the simplex lists in thee->qsm

Definition at line 111 of file [vcsn.h](#).

9.22.2.7 int nsimp

The _currently used) length of sqm, nsqm – may not always be up-to-date with Gem

Definition at line 105 of file [vcsn.h](#).

9.22.2.8 int* nsqm

The length of the charge lists in thee->sqm

Definition at line 104 of file [vcsn.h](#).

9.22.2.9 int qsm**

The inverse of sqm; the list of simplices associated with a given charge

Definition at line 109 of file [vcsn.h](#).

9.22.2.10 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 indices in thee->alist with lengths given in thee->nsqm

Definition at line 97 of file [vcsn.h](#).

9.22.2.11 Vmem* vmem

Memory management object

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

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

- [src/fem/vcsn.h](#)

9.23 sVfetk Struct Reference

Contains public data members for Vfetk class/module.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/fem/vfetk.h>
```

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](#)

9.23.1 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 [176](#) of file [vfetk.h](#).

9.23.2 Field Documentation

9.23.2.1 AM* [am](#)

Multilevel algebra manager.

Definition at line [182](#) of file [vfetk.h](#).

9.23.2.2 Aprx* [aprx](#)

Approximation manager.

Definition at line [183](#) of file [vfetk.h](#).

9.23.2.3 Vcsm* [csm](#)

Charge-simplex map

Definition at line [186](#) of file [vfetk.h](#).

9.23.2.4 **FEMparm*** feparm

FEM-specific parameters

Definition at line 198 of file [vfetk.h](#).

9.23.2.5 **Gem*** gm

Grid manager (container class for master vertex and simplex lists as well as prolongation operator for updating after refinement).

Definition at line 179 of file [vfetk.h](#).

9.23.2.6 **Vfetk_GuessType** gues

Initial guess method

Definition at line 193 of file [vfetk.h](#).

9.23.2.7 **int** level

Refinement level (starts at 0)

Definition at line 200 of file [vfetk.h](#).

9.23.2.8 **Vfetk_LsolvType** lkey

Linear solver method

Definition at line 187 of file [vfetk.h](#).

9.23.2.9 **int** lmax

Maximum number of linear solver iterations

Definition at line 188 of file [vfetk.h](#).

9.23.2.10 **Vfetk_PrecType** lprec

Linear preconditioner

Definition at line 194 of file [vfetk.h](#).

9.23.2.11 **double** ltol

Residual tolerance for linear solver

Definition at line 189 of file [vfetk.h](#).

9.23.2.12 **Vfetk_NsolvType** nkey

Nonlinear solver method

Definition at line 190 of file [vfetk.h](#).

9.23.2.13 int nmax

Maximum number of nonlinear solver iterations

Definition at line 191 of file [vfetk.h](#).

9.23.2.14 double ntol

Residual tolerance for nonlinear solver

Definition at line 192 of file [vfetk.h](#).

9.23.2.15 Vpbe* pbe

Poisson-Boltzmann object

Definition at line 185 of file [vfetk.h](#).

9.23.2.16 PBEparm* pbeparm

Generic PB parameters

Definition at line 197 of file [vfetk.h](#).

9.23.2.17 PDE* pde

FEtk PDE object

Definition at line 184 of file [vfetk.h](#).

9.23.2.18 int pjac

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

Definition at line 195 of file [vfetk.h](#).

9.23.2.19 Vhal_PBEType type

Version of PBE to solve

Definition at line 199 of file [vfetk.h](#).

9.23.2.20 Vmem* vmem

Memory management object

Definition at line 178 of file [vfetk.h](#).

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

- [src/fem/vfetk.h](#)

9.24 sVfetk_LocalVar Struct Reference

Vfetk LocalVar subclass.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/fem/vfetk.h>
```

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 [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](#)

9.24.1 Detailed Description

Vfetk LocalVar subclass.

Author

Nathan Baker Contains variables used when solving the PDE with FEtk

Definition at line [215](#) of file [vfetk.h](#).

9.24.2 Field Documentation

9.24.2.1 double A

Second-order differential term

Definition at line 228 of file [vfetk.h](#).

9.24.2.2 double B

Entire ionic strength term

Definition at line 230 of file [vfetk.h](#).

9.24.2.3 double d2W

Coulomb regularization term Laplacian

Definition at line 223 of file [vfetk.h](#).

9.24.2.4 double DB

Entire ionic strength term derivative

Definition at line 231 of file [vfetk.h](#).

9.24.2.5 double delta

Store delta value

Definition at line 250 of file [vfetk.h](#).

9.24.2.6 double DFu_wv

Store DFu_wv value

Definition at line 249 of file [vfetk.h](#).

9.24.2.7 double dU[MAXV][VAPBS_DIM]

Solution gradient

Definition at line 220 of file [vfetk.h](#).

9.24.2.8 double dW[VAPBS_DIM]

Coulomb regularization term gradient

Definition at line 222 of file [vfetk.h](#).

9.24.2.9 double F

RHS characteristic function value

Definition at line [229](#) of file [vfetk.h](#).

9.24.2.10 Vfetk* fetk

Pointer to the VFETK object

Definition at line [233](#) of file [vfetk.h](#).

9.24.2.11 int fType

Face type

Definition at line [225](#) of file [vfetk.h](#).

9.24.2.12 double Fu_v

Store Fu_v value

Definition at line [248](#) of file [vfetk.h](#).

9.24.2.13 Vgreen* green

Pointer to a Green's function object

Definition at line [234](#) of file [vfetk.h](#).

9.24.2.14 int initGreen

Boolean to designate whether Green's function has been initialized

Definition at line [235](#) of file [vfetk.h](#).

9.24.2.15 double ionConc[MAXION]

Counterion species' concentrations

Definition at line [241](#) of file [vfetk.h](#).

9.24.2.16 double ionQ[MAXION]

Counterion species' valencies

Definition at line [242](#) of file [vfetk.h](#).

9.24.2.17 double ionRadii[MAXION]

Counterion species' radii

Definition at line [243](#) of file [vfetk.h](#).

9.24.2.18 double ionstr

Ionic strength parameters (M)

Definition at line 246 of file [vfetk.h](#).

9.24.2.19 double jumpDiel

Dielectric value on one side of a simplex face

Definition at line 232 of file [vfetk.h](#).

9.24.2.20 int nion

Number of ion species

Definition at line 247 of file [vfetk.h](#).

9.24.2.21 double nvec[VAPBS_DIM]

Normal vector for a simplex face

Definition at line 216 of file [vfetk.h](#).

9.24.2.22 int nverts

number of vertices in the simplex

Definition at line 240 of file [vfetk.h](#).

9.24.2.23 SS* simp

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

Definition at line 237 of file [vfetk.h](#).

9.24.2.24 int sType

Simplex type

Definition at line 224 of file [vfetk.h](#).

9.24.2.25 double U[MAXV]

Solution value

Definition at line 219 of file [vfetk.h](#).

9.24.2.26 double u_D

Store Dirichlet value

Definition at line 251 of file [vfetk.h](#).

9.24.2.27 double u_T

Store true value

Definition at line 252 of file [vfetk.h](#).

9.24.2.28 VV* verts[4]

Pointer to the latest vertices; set in initElement

Definition at line 239 of file [vfetk.h](#).

9.24.2.29 double vx[4][VAPBS_DIM]

Vertex coordinates

Definition at line 217 of file [vfetk.h](#).

9.24.2.30 double W

Coulomb regularization term scalar value

Definition at line 221 of file [vfetk.h](#).

9.24.2.31 double xq[VAPBS_DIM]

Quadrature pt

Definition at line 218 of file [vfetk.h](#).

9.24.2.32 double zkappa2

Ionic strength parameters

Definition at line 244 of file [vfetk.h](#).

9.24.2.33 double zks2

Ionic strength parameters

Definition at line 245 of file [vfetk.h](#).

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

- [src/fem/vfetk.h](#)

9.25 sVgreen Struct Reference

Contains public data members for Vgreen class/module.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/generic/vgreen.h>
```

Data Fields

- Valist * [alist](#)
- Vmem * [vmem](#)
- double * [xp](#)
- double * [yp](#)
- double * [zp](#)
- double * [qp](#)
- int [np](#)

9.25.1 Detailed Description

Contains public data members for Vgreen class/module.

Author

Nathan Baker

Definition at line [82](#) of file [vgreen.h](#).

9.25.2 Field Documentation

9.25.2.1 Valist* alist

Atom (charge) list for Green's function

Definition at line [84](#) of file [vgreen.h](#).

9.25.2.2 int np

Set to size of above arrays

Definition at line [94](#) of file [vgreen.h](#).

9.25.2.3 double* qp

Array of particle charges for use with treecode routines

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

9.25.2.4 Vmem* vmem

Memory management object

Definition at line [85](#) of file [vgreen.h](#).

9.25.2.5 double* xp

Array of particle x-coordinates for use with treecode routines

Definition at line [86](#) of file [vgreen.h](#).

9.25.2.6 `double* yp`

Array of particle y-coordinates for use with treecode routines

Definition at line 88 of file [vgreen.h](#).

9.25.2.7 `double* zp`

Array of particle z-coordinates for use with treecode routines

Definition at line 90 of file [vgreen.h](#).

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

- [src/generic/vgreen.h](#)

9.26 `sVgrid` Struct Reference

Electrostatic potential oracle for Cartesian mesh data.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/mg/vgrid.h>
```

Data Fields

- int `nx`
- int `ny`
- int `nz`
- double `hx`
- double `hy`
- double `hz`
- double `xmin`
- double `ymin`
- double `zmin`
- double `xmax`
- double `ymax`
- double `zmax`
- double * `data`
- int `readdata`
- int `ctordata`
- Vmem * `mem`

9.26.1 Detailed Description

Electrostatic potential oracle for Cartesian mesh data.

Author

Nathan Baker

Definition at line 81 of file [vgrid.h](#).

9.26.2 Field Documentation

9.26.2.1 int ctordata

flag indicating whether data was included at construction

Definition at line 97 of file [vgrid.h](#).

9.26.2.2 double* data

nx*ny*nz array of data

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

9.26.2.3 double hx

Grid spacing in x direction

Definition at line 86 of file [vgrid.h](#).

9.26.2.4 double hy

Grid spacing in y direction

Definition at line 87 of file [vgrid.h](#).

9.26.2.5 double hzed

Grid spacing in z direction

Definition at line 88 of file [vgrid.h](#).

9.26.2.6 Vmem* mem

Memory manager object

Definition at line 99 of file [vgrid.h](#).

9.26.2.7 int nx

Number grid points in x direction

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

9.26.2.8 int ny

Number grid points in y direction

Definition at line 84 of file [vgrid.h](#).

9.26.2.9 int nz

Number grid points in z direction

Definition at line 85 of file [vgrid.h](#).

9.26.2.10 int readdata

flag indicating whether data was read from file

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

9.26.2.11 double xmax

x coordinate of upper grid corner

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

9.26.2.12 double xmin

x coordinate of lower grid corner

Definition at line 89 of file [vgrid.h](#).

9.26.2.13 double ymax

y coordinate of upper grid corner

Definition at line 93 of file [vgrid.h](#).

9.26.2.14 double ymin

y coordinate of lower grid corner

Definition at line 90 of file [vgrid.h](#).

9.26.2.15 double zmax

z coordinate of upper grid corner

Definition at line 94 of file [vgrid.h](#).

9.26.2.16 double zmin

z coordinate of lower grid corner

Definition at line 91 of file [vgrid.h](#).

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

- [src/mg/vgrid.h](#)

9.27 sVmgrid Struct Reference

Multiresolution oracle for Cartesian mesh data.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/mg/vmgrid.h>
```

Data Fields

- int [ngrid](#)s
- [Vgrid](#) * [grid](#)s [[VMGRIDMAX](#)]

9.27.1 Detailed Description

Multiresolution oracle for Cartesian mesh data.

Author

Nathan Baker

Definition at line [84](#) of file [vmgrid.h](#).

9.27.2 Field Documentation

9.27.2.1 [Vgrid](#)* [grid](#)s[[VMGRIDMAX](#)]

Grids in hierarchy. Our convention will be to have the finest grid first, however, this will not be enforced as it may be useful to search multiple grids for parallel datasets, etc.

Definition at line [87](#) of file [vmgrid.h](#).

9.27.2.2 int [ngrid](#)s

Number of grids in hierarchy

Definition at line [86](#) of file [vmgrid.h](#).

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

- [src/mg/vmgrid.h](#)

9.28 sVopot Struct Reference

Electrostatic potential oracle for Cartesian mesh data.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/mg/vopot.h>
```

Data Fields

- [Vmgrid](#) * [mgrid](#)
- [Vpbe](#) * [pbe](#)
- [Vbcfl](#) [bcfl](#)

9.28.1 Detailed Description

Electrostatic potential oracle for Cartesian mesh data.

Author

Nathan Baker

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

9.28.2 Field Documentation

9.28.2.1 [Vbcfl](#) [bcfl](#)

Boundary condition flag for returning potential values at points off the grid.

Definition at line [88](#) of file [vopot.h](#).

9.28.2.2 [Vmgrid](#)* [mgrid](#)

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

Definition at line [85](#) of file [vopot.h](#).

9.28.2.3 [Vpbe](#)* [pbe](#)

Pointer to PBE object

Definition at line [87](#) of file [vopot.h](#).

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

- [src/mg/vopot.h](#)

9.29 [sVparam_AtomData](#) Struct Reference

AtomData sub-class; stores atom data.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/generic/vparam.h>
```

Data Fields

- char [atomName](#) [VMAX_ARGLEN]
- char [resName](#) [VMAX_ARGLEN]

- double [charge](#)
- double [radius](#)
- double [epsilon](#)

9.29.1 Detailed Description

AtomData sub-class; stores atom data.

Author

Nathan Baker

Note

The epsilon and radius members of this class refer use the following formula for calculating the van der Waals energy of atom i interacting with atom j :

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

where $\epsilon_{ij} = \sqrt{\epsilon_i \epsilon_j}$ is the well-depth (in the desired energy units), r_{ij} is the distance between atoms i and j , and $\sigma_{ij} = \sigma_i + \sigma_j$ is the sum of the van der Waals radii.

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

9.29.2 Field Documentation

9.29.2.1 char atomName[VMAX_ARGLEN]

Atom name

Definition at line 93 of file [vparam.h](#).

9.29.2.2 double charge

Atom charge (in e)

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

9.29.2.3 double epsilon

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

Definition at line 97 of file [vparam.h](#).

9.29.2.4 double radius

Atom VdW radius (σ_i above; in Å)

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

9.29.2.5 char resName[VMAX_ARGLEN]

Residue name

Definition at line 94 of file [vparam.h](#).

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

- [src/generic/vparam.h](#)

9.30 sVpbe Struct Reference

Contains public data members for Vpbe class/module.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/generic/vpbe.h>
```

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](#)

9.30.1 Detailed Description

Contains public data members for Vpbe class/module.

Author

Nathan Baker

Definition at line 84 of file [vpbe.h](#).

9.30.2 Field Documentation

9.30.2.1 **Vacc*** acc

Accessibility object

Definition at line 90 of file [vpbe.h](#).

9.30.2.2 **Valist*** alist

Atom (charge) list

Definition at line 88 of file [vpbe.h](#).

9.30.2.3 **double** bulkIonicStrength

Bulk ionic strength (M)

Definition at line 99 of file [vpbe.h](#).

9.30.2.4 **Vclist*** clist

Atom location cell list

Definition at line 89 of file [vpbe.h](#).

9.30.2.5 **double** deblen

Debye length (bulk)

Definition at line 109 of file [vpbe.h](#).

9.30.2.6 **double** ionConc[**MAXION**]

Concentration (M) of each species

Definition at line 104 of file [vpbe.h](#).

9.30.2.7 **double** ionQ[**MAXION**]

Charge (e) of each species

Definition at line 106 of file [vpbe.h](#).

9.30.2.8 double ionRadii[MAXION]

Ionic radius (A) of each species

Definition at line 105 of file [vpbe.h](#).

9.30.2.9 int ipkey

PBE calculation type (this is a cached copy it should not be used directly in code)

Definition at line 122 of file [vpbe.h](#).

9.30.2.10 double L

Length of the membrane (A)

Definition at line 132 of file [vpbe.h](#).

9.30.2.11 double maxIonRadius

Max ion radius (A; used for calculating accessibility and defining volumes for ionic strength coefficients)

Definition at line 100 of file [vpbe.h](#).

9.30.2.12 double membraneDiel

Membrane dielectric constant

Definition at line 133 of file [vpbe.h](#).

9.30.2.13 int numIon

Total number of ion species

Definition at line 103 of file [vpbe.h](#).

9.30.2.14 int param2Flag

Check to see if bcf=3 parms have been set

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

9.30.2.15 int paramFlag

Check to see if the parameters have been set

Definition at line 125 of file [vpbe.h](#).

9.30.2.16 double smsize

Size-Modified PBE size

Definition at line 121 of file [vpbe.h](#).

9.30.2.17 double smvolume

Size-Modified PBE relative volume

Definition at line 120 of file [vpbe.h](#).

9.30.2.18 double soluteCenter[3]

Center of solute molecule (A)

Definition at line 113 of file [vpbe.h](#).

9.30.2.19 double soluteCharge

Charge of solute molecule (e)

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

9.30.2.20 double soluteDiel

Solute dielectric constant (unitless)

Definition at line 93 of file [vpbe.h](#).

9.30.2.21 double soluteRadius

Radius of solute molecule (A)

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

9.30.2.22 double soluteXlen

Solute length in x-direction

Definition at line 115 of file [vpbe.h](#).

9.30.2.23 double soluteYlen

Solute length in y-direction

Definition at line 116 of file [vpbe.h](#).

9.30.2.24 double soluteZlen

Solute length in z-direction

Definition at line 117 of file [vpbe.h](#).

9.30.2.25 double solventDiel

Solvent dielectric constant (unitless)

Definition at line 94 of file [vpbe.h](#).

9.30.2.26 double solventRadius

Solvent probe radius (angstroms) for accessibility; determining defining volumes for the dielectric coefficient

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

9.30.2.27 double T

Temperature (K)

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

9.30.2.28 double V

Membrane potential

Definition at line 134 of file [vpbe.h](#).

9.30.2.29 Vmem* vmem

Memory management object

Definition at line 86 of file [vpbe.h](#).

9.30.2.30 double xkappa

Debye-Huckel parameter (bulk)

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

9.30.2.31 double z_mem

Z value of the bottom of the membrane (A)

Definition at line 131 of file [vpbe.h](#).

9.30.2.32 double zkappa2

Square of modified Debye-Huckel parameter (bulk)

Definition at line 110 of file [vpbe.h](#).

9.30.2.33 double zmagic

Delta function scaling parameter

Definition at line 111 of file [vpbe.h](#).

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

- [src/generic/vpbe.h](#)

9.31 sVpee Struct Reference

Contains public data members for Vpee class/module.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/fem/vpee.h>
```

Data Fields

- Gem * [gm](#)
- int [localPartID](#)
- double [localPartCenter](#) [3]
- double [localPartRadius](#)
- int [killFlag](#)
- double [killParam](#)
- Vmem * [mem](#)

9.31.1 Detailed Description

Contains public data members for Vpee class/module.

Author

Nathan Baker

Definition at line 89 of file [vpee.h](#).

9.31.2 Field Documentation

9.31.2.1 Gem* gm

Grid manager

Definition at line 91 of file [vpee.h](#).

9.31.2.2 int killFlag

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

Definition at line 99 of file [vpee.h](#).

9.31.2.3 double killParam

A parameter for the error estimate attenuation method

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

9.31.2.4 double localPartCenter[3]

The coordinates of the center of the local partition

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

9.31.2.5 int localPartID

The local partition ID: i.e. the partition whose boundary simplices we're keeping track of

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

9.31.2.6 double localPartRadius

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

Definition at line 97 of file [vpee.h](#).

9.31.2.7 Vmem* mem

Memory manager

Definition at line 104 of file [vpee.h](#).

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

- [src/fem/vpee.h](#)

9.32 sVpmg Struct Reference

Contains public data members for Vpmg class/module.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/mg/vpmg.h>
```

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](#)

9.32.1 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 [116](#) of file [vpmg.h](#).

9.32.2 Field Documentation

9.32.2.1 double* [a1cf](#)

Operator coefficient values ([a11](#)) – this array can be overwritten

Definition at line [138](#) of file [vpmg.h](#).

9.32.2.2 double* a2cf

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

Definition at line 140 of file [vpmg.h](#).

9.32.2.3 double* a3cf

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

Definition at line 142 of file [vpmg.h](#).

9.32.2.4 double* ccf

Helmholtz term – this array can be overwritten

Definition at line 144 of file [vpmg.h](#).

9.32.2.5 double* charge

Charge map

Definition at line 132 of file [vpmg.h](#).

9.32.2.6 Vgrid* chargeMap

External charge distribution map

Definition at line 188 of file [vpmg.h](#).

9.32.2.7 Vchrg_Meth chargeMeth

Charge discretization method

Definition at line 165 of file [vpmg.h](#).

9.32.2.8 Vchrg_Src chargeSrc

Charge source

Definition at line 166 of file [vpmg.h](#).

9.32.2.9 Vgrid* dielXMap

External x-shifted dielectric map

Definition at line 172 of file [vpmg.h](#).

9.32.2.10 Vgrid* dielYMap

External y-shifted dielectric map

Definition at line 175 of file [vpmg.h](#).

9.32.2.11 Vgrid* dielZMap

External z-shifted dielectric map

Definition at line 178 of file [vpmpg.h](#).

9.32.2.12 double* epsx

X-shifted dielectric map

Definition at line 127 of file [vpmpg.h](#).

9.32.2.13 double* epsy

Y-shifted dielectric map

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

9.32.2.14 double* epsz

Y-shifted dielectric map

Definition at line 129 of file [vpmpg.h](#).

9.32.2.15 double extDiEnergy

Stores contributions to the dielectric energy from regions outside the problem domain

Definition at line 155 of file [vpmpg.h](#).

9.32.2.16 double extNpEnergy

Stores contributions to the apolar energy from regions outside the problem domain

Definition at line 161 of file [vpmpg.h](#).

9.32.2.17 double extQfEnergy

Stores contributions to the fixed charge energy from regions outside the problem domain

Definition at line 159 of file [vpmpg.h](#).

9.32.2.18 double extQmEnergy

Stores contributions to the mobile ion energy from regions outside the problem domain

Definition at line 157 of file [vpmpg.h](#).

9.32.2.19 double* fcf

Right-hand side – this array can be overwritten

Definition at line 145 of file [vpmpg.h](#).

9.32.2.20 int filled

Indicates whether Vpmg_fillco has been called

Definition at line 168 of file [vpmg.h](#).

9.32.2.21 double* gxcf

Boundary conditions for x faces

Definition at line 151 of file [vpmg.h](#).

9.32.2.22 double* gyxf

Boundary conditions for y faces

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

9.32.2.23 double* gzcf

Boundary conditions for z faces

Definition at line 153 of file [vpmg.h](#).

9.32.2.24 int* iparm

Passing int parameters to FORTRAN

Definition at line 134 of file [vpmg.h](#).

9.32.2.25 int* iwork

Work array

Definition at line 136 of file [vpmg.h](#).

9.32.2.26 double* kappa

Ion accessibility map ($0 \leq \text{kappa}(x) \leq 1$)

Definition at line 130 of file [vpmg.h](#).

9.32.2.27 Vgrid* kappaMap

External kappa map

Definition at line 181 of file [vpmg.h](#).

9.32.2.28 Vpbe* pbe

Information about the PBE system

Definition at line 120 of file [vpmg.h](#).

9.32.2.29 Vpmgp* pmgp

Parameters

Definition at line 119 of file [vpmg.h](#).

9.32.2.30 double* pot

Potential map

Definition at line 131 of file [vpmg.h](#).

9.32.2.31 Vgrid* potMap

External potential map

Definition at line 184 of file [vpmg.h](#).

9.32.2.32 double* pvec

Partition mask array

Definition at line 154 of file [vpmg.h](#).

9.32.2.33 double* rparm

Passing real parameters to FORTRAN

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

9.32.2.34 double* rwork

Work array

Definition at line 137 of file [vpmg.h](#).

9.32.2.35 double splineWin

Spline window parm for surf defs

Definition at line 164 of file [vpmg.h](#).

9.32.2.36 Vsurf_Meth surfMeth

Surface definition method

Definition at line 163 of file [vpmg.h](#).

9.32.2.37 double* tcf

True solution

Definition at line 146 of file [vpmg.h](#).

9.32.2.38 double* u

Solution

Definition at line 147 of file [vpmg.h](#).

9.32.2.39 int useChargeMap

Indicates whether Vpmg_fillco was called with an external charge distribution map

Definition at line 186 of file [vpmg.h](#).

9.32.2.40 int useDielXMap

Indicates whether Vpmg_fillco was called with an external x-shifted dielectric map

Definition at line 170 of file [vpmg.h](#).

9.32.2.41 int useDielYMap

Indicates whether Vpmg_fillco was called with an external y-shifted dielectric map

Definition at line 173 of file [vpmg.h](#).

9.32.2.42 int useDielZMap

Indicates whether Vpmg_fillco was called with an external z-shifted dielectric map

Definition at line 176 of file [vpmg.h](#).

9.32.2.43 int useKappaMap

Indicates whether Vpmg_fillco was called with an external kappa map

Definition at line 179 of file [vpmg.h](#).

9.32.2.44 int usePotMap

Indicates whether Vpmg_fillco was called with an external potential map

Definition at line 182 of file [vpmg.h](#).

9.32.2.45 Vmem* vmem

Memory management object for this class

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

9.32.2.46 double* xf

Mesh point x coordinates

Definition at line 148 of file [vpmg.h](#).

9.32.2.47 double* yf

Mesh point y coordinates

Definition at line 149 of file [vpmg.h](#).

9.32.2.48 double* zf

Mesh point z coordinates

Definition at line 150 of file [vpmg.h](#).

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

- [src/mg/vpmg.h](#)

9.33 sVpmgp Struct Reference

Contains public data members for Vpmgp class/module.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/mg/vpmgp.h>
```

Data Fields

- int [nx](#)
- int [ny](#)
- int [nz](#)
- int [nlev](#)
- double [hx](#)
- double [hy](#)
- double [hz](#)
- int [nonlin](#)
- int [nxc](#)
- int [nyc](#)
- int [nzc](#)
- int [nf](#)
- int [nc](#)
- int [narrc](#)
- int [n_rpc](#)
- int [n_iz](#)
- int [n_ipc](#)
- size_t [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](#)

9.33.1 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 [80](#) of file [vpmgp.h](#).

9.33.2 Field Documentation

9.33.2.1 [Vbcfl](#) [bcfl](#)

Boundary condition method [default = BCFL_SDH]

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

9.33.2.2 double errtol

Desired error tolerance [default = 1e-9]

Definition at line 121 of file [vpmgp.h](#).

9.33.2.3 double hx

Grid x spacings [no default]

Definition at line 87 of file [vpmgp.h](#).

9.33.2.4 double hy

Grid y spacings [no default]

Definition at line 88 of file [vpmgp.h](#).

9.33.2.5 double hzed

Grid z spacings [no default]

Definition at line 89 of file [vpmgp.h](#).

9.33.2.6 int iinfo

Runtime status messages [default = 1]

- 0: none
- 1: some
- 2: lots
- 3: more

Definition at line 130 of file [vpmgp.h](#).

9.33.2.7 int ipcon

Preconditioning method [default = 3]

- 0: diagonal
- 1: ICCG
- 2: ICCGDW
- 3: MICCGDW
- 4: none

Definition at line 183 of file [vpmgp.h](#).

9.33.2.8 int iperf

Analysis of the operator [default = 0]

- 0: no
- 1: condition number
- 2: spectral radius
- 3: cond. number & spectral radius

Definition at line 139 of file [vpmgp.h](#).

9.33.2.9 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 109 of file [vpmgp.h](#).

9.33.2.10 int irite

FORTTRAN output unit [default = 8]

Definition at line 182 of file [vpmgp.h](#).

9.33.2.11 int istop

Stopping criterion [default = 1]

- 0: residual
- 1: relative residual
- 2: diff
- 3: errc
- 4: errd
- 5: aerrd

Definition at line 123 of file [vpmgp.h](#).

9.33.2.12 int itmax

Maximum number of iters [default = 100]

Definition at line 122 of file [vpmgp.h](#).

9.33.2.13 int key

Print solution to file [default = 0]

- 0: no
- 1: yes

Definition at line 136 of file [vpmgp.h](#).

9.33.2.14 int meth

Solution method [default = 2]

- 0: conjugate gradient multigrid
- 1: newton
- 2: multigrid
- 3: conjugate gradient
- 4: successive overrelaxation
- 5: red-black gauss-seidel
- 6: weighted jacobi
- 7: richardson
- 8: conjugate gradient multigrid aqua
- 9: newton aqua

Definition at line 144 of file [vpmgp.h](#).

9.33.2.15 int mgcoar

Coarsening method [default = 2]

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

Definition at line 170 of file [vpmgp.h](#).

9.33.2.16 int mgdisc

Discretization method [default = 0]

- 0: finite volume
- 1: finite element

Definition at line 177 of file [vpmgp.h](#).

9.33.2.17 int mgkey

Multigrid method [default = 0]

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

Definition at line 155 of file [vpmgp.h](#).

9.33.2.18 int mgprol

Prolongation method [default = 0]

- 0: trilinear
- 1: operator-based
- 2: mod. operator-based

Definition at line 166 of file [vpmgp.h](#).

9.33.2.19 int mgsmoo

Smoothing method [default = 1]

- 0: weighted jacobi
- 1: gauss-seidel
- 2: SOR
- 3: richardson
- 4: cghs

Definition at line 160 of file [vpmgp.h](#).

9.33.2.20 int mgsolv

Coarse equation solve method [default = 1]

- 0: cghs
- 1: banded linpack

Definition at line 174 of file [vpmgp.h](#).

9.33.2.21 int n_ipc

Integer info work array required storage

Definition at line 104 of file [vpmgp.h](#).

9.33.2.22 int n_iz

Integer storage parameter (index max)

Definition at line 103 of file [vpmgp.h](#).

9.33.2.23 int n_rpc

Real info work array required storage

Definition at line 102 of file [vpmgp.h](#).

9.33.2.24 int narr

Array work storage

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

9.33.2.25 int narrc

Size of vector on coarse level

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

9.33.2.26 int nc

Number of coarse grid unknowns

Definition at line 100 of file [vpmgp.h](#).

9.33.2.27 int nf

Number of fine grid unknowns

Definition at line 99 of file [vpmgp.h](#).

9.33.2.28 int niwk

Integer work storage

Definition at line 107 of file [vpmgp.h](#).

9.33.2.29 int nlev

Number of mesh levels [no default]

Definition at line 86 of file [vpmgp.h](#).

9.33.2.30 int nonlin

Problem type [no default]

- 0: linear
- 1: nonlinear
- 2: linear then nonlinear

Definition at line 90 of file [vpmgp.h](#).

9.33.2.31 `size_t nrw`

Real work storage

Definition at line 106 of file [vpmgp.h](#).

9.33.2.32 `int nu1`

Number of pre-smoothings [default = 2]

Definition at line 158 of file [vpmgp.h](#).

9.33.2.33 `int nu2`

Number of post-smoothings [default = 2]

Definition at line 159 of file [vpmgp.h](#).

9.33.2.34 `int nx`

Grid x dimensions [no default]

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

9.33.2.35 `int nxc`

Coarse level grid x dimensions

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

9.33.2.36 `int ny`

Grid y dimensions [no default]

Definition at line 84 of file [vpmgp.h](#).

9.33.2.37 `int nyc`

Coarse level grid y dimensions

Definition at line 97 of file [vpmgp.h](#).

9.33.2.38 int nz

Grid z dimensions [no default]

Definition at line 85 of file [vpmgp.h](#).

9.33.2.39 int nzc

Coarse level grid z dimensions

Definition at line 98 of file [vpmgp.h](#).

9.33.2.40 double omegal

Linear relax parameter [default = 8e-1]

Definition at line 180 of file [vpmgp.h](#).

9.33.2.41 double omegan

Nonlin relax parameter [default = 9e-1]

Definition at line 181 of file [vpmgp.h](#).

9.33.2.42 double xcent

Grid x center [0]

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

9.33.2.43 double xlen

Domain x length

Definition at line 189 of file [vpmgp.h](#).

9.33.2.44 double xmax

Domain upper x corner

Definition at line 195 of file [vpmgp.h](#).

9.33.2.45 double xmin

Domain lower x corner

Definition at line 192 of file [vpmgp.h](#).

9.33.2.46 double ycent

Grid y center [0]

Definition at line 119 of file [vpmgp.h](#).

9.33.2.47 double ylen

Domain y length

Definition at line 190 of file [vpmgp.h](#).

9.33.2.48 double ymax

Domain upper y corner

Definition at line 196 of file [vpmgp.h](#).

9.33.2.49 double ymin

Domain lower y corner

Definition at line 193 of file [vpmgp.h](#).

9.33.2.50 double zcent

Grid z center [0]

Definition at line 120 of file [vpmgp.h](#).

9.33.2.51 double zlen

Domain z length

Definition at line 191 of file [vpmgp.h](#).

9.33.2.52 double zmax

Domain upper z corner

Definition at line 197 of file [vpmgp.h](#).

9.33.2.53 double zmin

Domain lower z corner

Definition at line 194 of file [vpmgp.h](#).

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

- [src/mg/vpmgp.h](#)

9.34 Vparam Struct Reference

Reads and assigns charge/radii parameters.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/generic/vparam.h>
```


Data Fields

- Vmem * [vmem](#)
- int [nResData](#)
- [Vparam_ResData](#) * [resData](#)

9.34.1 Detailed Description

Reads and assigns charge/radii parameters.

Author

Nathan Baker

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

9.34.2 Field Documentation

9.34.2.1 int nResData

Number of [Vparam_ResData](#) objects associated with this object

Definition at line [138](#) of file [vparam.h](#).

9.34.2.2 Vparam_ResData* resData

Array of nResData [Vparam_ResData](#) objects

Definition at line [140](#) of file [vparam.h](#).

9.34.2.3 Vmem* vmem

Memory management object for this class

Definition at line [137](#) of file [vparam.h](#).

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

- [src/generic/vparam.h](#)

9.35 Vparam_ResData Struct Reference

ResData sub-class; stores residue data.

```
#include </Users/d3x923/Documents/projects/APBS/apbs-pdb2pqr/apbs/src/generic/vparam.h>
```

Data Fields

- Vmem * [vmem](#)
- char [name](#) [VMAX_ARGLEN]

- int [nAtomData](#)
- [Vparam_AtomData](#) * [atomData](#)

9.35.1 Detailed Description

ResData sub-class; stores residue data.

Author

Nathan Baker

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

9.35.2 Field Documentation

9.35.2.1 [Vparam_AtomData](#)* [atomData](#)

Array of [Vparam_AtomData](#) natom objects

Definition at line [119](#) of file [vparam.h](#).

9.35.2.2 char name[VMAX_ARGLEN]

Residue name

Definition at line [116](#) of file [vparam.h](#).

9.35.2.3 int nAtomData

Number of [Vparam_AtomData](#) objects associated with this object

Definition at line [117](#) of file [vparam.h](#).

9.35.2.4 [Vmem](#)* [vmem](#)

Pointer to memory manager from [Vparam](#) master class

Definition at line [115](#) of file [vparam.h](#).

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

- [src/generic/vparam.h](#)

Chapter 10

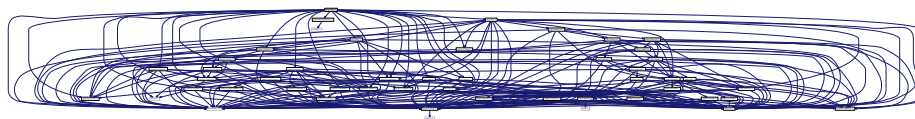
File Documentation

10.1 src/apbs.h File Reference

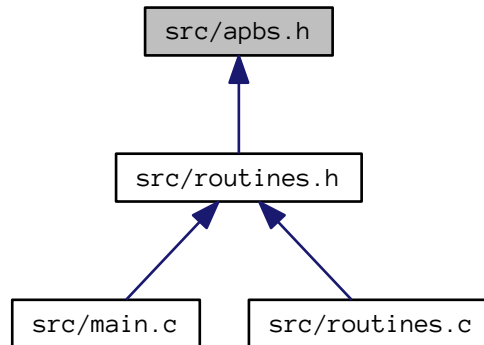
Header file for header dependencies.

```
#include "apbscfg.h"
#include "malloc/malloc.h"
#include "generic/nosh.h"
#include "generic/mgparm.h"
#include "generic/pbeparm.h"
#include "generic/femparm.h"
#include "generic/bemparm.h"
#include "generic/geoflowparm.h"
#include "generic/vacc.h"
#include "generic/valist.h"
#include "generic/vatom.h"
#include "generic/vcap.h"
#include "generic/vhal.h"
#include "generic/vpbe.h"
#include "generic/vstring.h"
#include "generic/vunit.h"
#include "generic/vparam.h"
#include "generic/vgreen.h"
#include "geoflow/cpbconcz2.h"
#include "mg/vgrid.h"
#include "mg/vmgrid.h"
#include "mg/vopot.h"
#include "mg/vpmg.h"
#include "mg/vpmgp.h"
```

Include dependency graph for apbs.h:



This graph shows which files directly or indirectly include this file:



10.1.1 Detailed Description

Header file for header dependencies.

Author

Nathan Baker

Version

\$Id\$

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*

```

```

* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [apbs.h](#).

10.2 apbs.h

```

00001
00061 #ifndef _APBSHEADERS_H_
00062 #define _APBSHEADERS_H_
00063
00064 #include "apbscfg.h"
00065
00066 /* MALLOC headers */
00067 #include "malloc/malloc.h"
00068
00069 /* Generic headers */
00070 #include "generic/nosh.h"
00071 #include "generic/mgparm.h"
00072 #include "generic/pbeparm.h"
00073 #include "generic/femparm.h"
00074 #include "generic/bemparm.h"
00075 #include "generic/geoflowparm.h"
00076 #include "generic/vacc.h"
00077 #include "generic/valist.h"
00078 #include "generic/vatom.h"
00079 #include "generic/vcap.h"
00080 #include "generic/vhal.h"
00081 #include "generic/vpbe.h"
00082 #include "generic/vstring.h"
00083 #include "generic/vunit.h"
00084 #include "generic/vparam.h"
00085 #include "generic/vgreen.h"
00086
00087 #include "geoflow/cpbconcz2.h"
00088
00089 /* MG headers */
00090 #include "mg/vgrid.h"
00091 #include "mg/vmgrid.h"
00092 #include "mg/vopot.h"
00093 #include "mg/vpmpg.h"
00094 #include "mg/vpmgp.h"
00095
00096 /* FEM headers */
00097 #if defined(FETK_ENABLED)
00098     #include "fem/vfetc.h"
00099     #include "fem/vpee.h"
00100 #endif
00101
00102 #endif /* _APBSHEADERS_H_ */

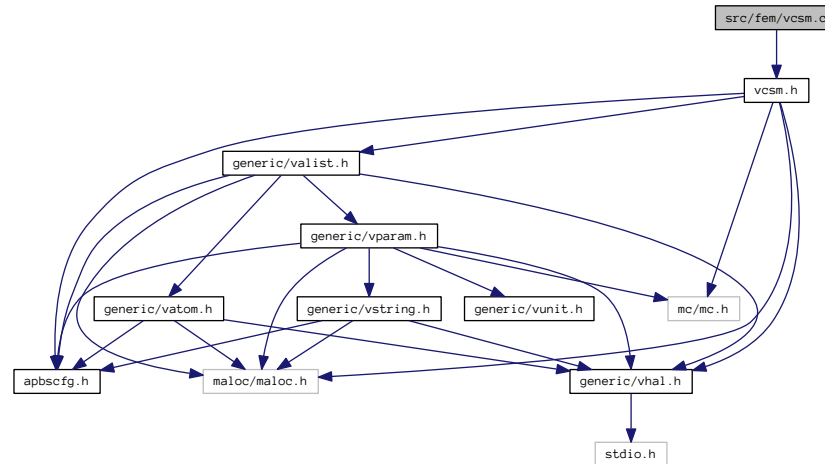
```

10.3 src/fem/vcsm.c File Reference

Class Vcsm methods.

```
#include "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_getNumberSimplexes (Vcsm *thee, int iatom)
Get number of simplexes 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)
FORTTRAN 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`)

FORTTRAN 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.

10.3.1 Detailed Description

Class Vcsm methods.

Author

Nathan Baker

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
```

```

* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [vcsn.c](#).

10.4 vcsn.c

```

00001
00057 #include "vcsn.h"
00058
00059 /* Inlineable methods */
00060 #if !defined(VINLINE_VCSN)
00061
00062 VPUBLIC Valist* Vcsn_getValist(Vcsn *thee) {
00063
00064     VASSERT(thee != VNULL);
00065     return thee->alist;
00066 }
00067
00068
00069 VPUBLIC int Vcsn_getNumberAtoms(Vcsn *thee, int isimp) {
00070
00071     VASSERT(thee != VNULL);
00072     VASSERT(thee->initFlag);
00073     return thee->nsqm[isimp];
00074 }
00075
00076
00077 VPUBLIC Vatom* Vcsn_getAtom(Vcsn *thee, int iatom, int isimp) {
00078
00079     VASSERT(thee != VNULL);
00080     VASSERT(thee->initFlag);
00081
00082     VASSERT(iatom < (thee->nsqm)[isimp]);
00083     return Valist_getAtom(thee->alist, (thee->sqm)[isimp][iatom]);
00084 }
00085
00086
00087
00088 VPUBLIC int Vcsn_getAtomIndex(Vcsn *thee, int iatom, int isimp) {
00089
00090     VASSERT(thee != VNULL);
00091     VASSERT(thee->initFlag);
00092
00093     VASSERT(iatom < (thee->nsqm)[isimp]);
00094     return (thee->sqm)[isimp][iatom];
00095 }
00096
00097
00098
00099 VPUBLIC int Vcsn_getNumberSimplices(Vcsn *thee, int iatom) {
00100
00101     VASSERT(thee != VNULL);
00102     VASSERT(thee->initFlag);
00103
00104     return (thee->nqsm)[iatom];
00105 }
00106
00107
00108
00109 VPUBLIC SS* Vcsn_getSimplex(Vcsn *thee, int isimp, int iatom) {
00110
00111     VASSERT(thee != VNULL);
00112     VASSERT(thee->initFlag);
00113
00114     return Gem_SS(thee->gm, (thee->qsm)[iatom][isimp]);
00115 }
00116
```



```

00117 }
00118
00119 VPUBLIC int Vcsn_getSimplexIndex(Vcsn *thee, int isimp, int iatom) {
00120
00121     VASSERT(thee != VNULL);
00122     VASSERT(thee->initFlag);
00123
00124     return (thee->qsm)[iatom][isimp];
00125 }
00126
00127 }
00128
00129 VPUBLIC unsigned long int Vcsn_memChk(Vcsn *thee) {
00130     if (thee == VNULL) return 0;
00131     return Vmem_bytes(thee->vmem);
00132 }
00133
00134 #endif /* if !defined(VINLINE_VCSM) */
00135
00136 VPUBLIC Vcsn* Vcsn_ctor(Valist *alist, Gem *gm) {
00137
00138     /* Set up the structure */
00139     Vcsn *thee = VNULL;
00140     thee = (Vcsn*)Vmem_malloc(VNULL, 1, sizeof(Vcsn) );
00141     VASSERT( thee != VNULL);
00142     VASSERT( Vcsn_ctor2(thee, alist, gm));
00143
00144     return thee;
00145 }
00146
00147 VPUBLIC int Vcsn_ctor2(Vcsn *thee, Valist *alist, Gem *gm) {
00148
00149     VASSERT( thee != VNULL );
00150
00151     /* Memory management object */
00152     thee->vmem = Vmem_ctor("APBS:VCSM");
00153
00154     /* Set up the atom list and grid manager */
00155     if( alist == VNULL) {
00156         Vnm_print(2, "Vcsn_ctor2: got null pointer to Valist object!\n");
00157         return 0;
00158     }
00159     thee->alist = alist;
00160     if( gm == VNULL) {
00161         Vnm_print(2, "Vcsn_ctor2: got a null pointer to the Gem object!\n");
00162         return 0;
00163     }
00164     thee->gm = gm;
00165
00166     thee->initFlag = 0;
00167     return 1;
00168 }
00169
00170 VPUBLIC void Vcsn_init(Vcsn *thee) {
00171
00172     /* Counters */
00173     int iatom,
00174         jatom,
00175         isimp,
00176         jsimp,
00177         gotSimp;
00178
00179     /* Atomic information */
00180     Vatom *atom;
00181     double *position;
00182     /* Simplex/Vertex information */
00183     SS *simplex;
00184     /* Basis function values */
00185
00186     if (thee == VNULL) {
00187         Vnm_print(2, "Vcsn_init: Error! Got NULL thee!\n");
00188         VASSERT(0);
00189     }
00190     if (thee->gm == VNULL) {
00191         Vnm_print(2, "Vcsn_init: Error! Got NULL thee->gm!\n");
00192         VASSERT(0);
00193     }
00194     thee->nsimp = Gem_numSS(thee->gm);
00195     if (thee->nsimp <= 0) {
00196         Vnm_print(2, "Vcsn_init: Error! Got %d simplices!\n", thee->nsimp);
00197         VASSERT(0);
00198     }
00199 }

```

```

00198     thee->natom = Valist_getNumberAtoms(thee->alist);
00199
00200     /* Allocate and initialize space for the first dimensions of the
00201      * simplex-charge map, the simplex array, and the counters */
00202     thee->sgm = (int**)Vmem_malloc(thee->vmem, thee->nsimp, sizeof(int *));
00203     VASSERT(thee->sgm != VNULL);
00204     thee->nsqm = (int*)Vmem_malloc(thee->vmem, thee->nsimp, sizeof(int));
00205     VASSERT(thee->nsqm != VNULL);
00206     for (isimp=0; isimp<thee->nsimp; isimp++) (thee->nsqm)[isimp] = 0;
00207
00208     /* Count the number of charges per simplex. */
00209     for (iatom=0; iatom<thee->natom; iatom++) {
00210         atom = Valist_getAtom(thee->alist, iatom);
00211         position = Vatom_getPosition(atom);
00212         gotSimp = 0;
00213         for (isimp=0; isimp<thee->nsimp; isimp++) {
00214             simplex = Gem_SS(thee->gm, isimp);
00215             if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00216                 (thee->nsqm)[isimp]++;
00217                 gotSimp = 1;
00218             }
00219         }
00220     }
00221
00222     /* @todo Combine the following two loops? - PCE */
00223     /* Allocate the space for the simplex-charge map */
00224     for (isimp=0; isimp<thee->nsimp; isimp++) {
00225         if ((thee->nsqm)[isimp] > 0) {
00226             thee->sgm[isimp] = (int*)Vmem_malloc(thee->vmem, (thee->nsqm)[isimp],
00227                 sizeof(int));
00228             VASSERT(thee->sgm[isimp] != VNULL);
00229         }
00230     }
00231
00232     /* Finally, set up the map */
00233     for (isimp=0; isimp<thee->nsimp; isimp++) {
00234         jsimp = 0;
00235         simplex = Gem_SS(thee->gm, isimp);
00236         for (iatom=0; iatom<thee->natom; iatom++) {
00237             atom = Valist_getAtom(thee->alist, iatom);
00238             position = Vatom_getPosition(atom);
00239             /* Check to see if the atom's in this simplex */
00240             if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00241                 /* Assign the entries in the next vacant spot */
00242                 (thee->sgm[isimp])[jsimp] = iatom;
00243                 jsimp++;
00244             }
00245         }
00246     }
00247
00248     thee->msimp = thee->nsimp;
00249
00250     /* Allocate space for the charge-simplex map */
00251     thee->qsm = (int**)Vmem_malloc(thee->vmem, thee->natom, sizeof(int *));
00252     VASSERT(thee->qsm != VNULL);
00253     thee->nqsm = (int*)Vmem_malloc(thee->vmem, thee->natom, sizeof(int));
00254     VASSERT(thee->nqsm != VNULL);
00255     for (iatom=0; iatom<thee->natom; iatom++) (thee->nqsm)[iatom] = 0;
00256     /* Loop through the list of simplices and count the number of times
00257      * each atom appears */
00258     for (isimp=0; isimp<thee->nsimp; isimp++) {
00259         for (iatom=0; iatom<thee->nsqm[isimp]; iatom++) {
00260             jatom = thee->sgm[isimp][iatom];
00261             thee->nqsm[jatom]++;
00262         }
00263     }
00264     /* Do a TIME-CONSUMING SANITY CHECK to make sure that each atom was
00265      * placed in at simplex */
00266     for (iatom=0; iatom<thee->natom; iatom++) {
00267         if (thee->nqsm[iatom] == 0) {
00268             Vnm_print(2, "Vcsm_init: Atom %d not placed in simplex!\n", iatom);
00269             VASSERT(0);
00270         }
00271     }
00272     /* Allocate the appropriate amount of space for each entry in the
00273      * charge-simplex map and clear the counter for re-use in assignment */
00274     for (iatom=0; iatom<thee->natom; iatom++) {
00275         thee->qsm[iatom] = (int*)Vmem_malloc(thee->vmem, (thee->nqsm)[iatom],
00276             sizeof(int));
00277         VASSERT(thee->qsm[iatom] != VNULL);
00278         thee->nqsm[iatom] = 0;

```

```

00279     }
00280     /* Assign the simplices to atoms */
00281     for (isimp=0; isimp<thee->nsimp; isimp++) {
00282         for (iatom=0; iatom<thee->nsqm[isimp]; iatom++) {
00283             jatom = thee->sqm[isimp][iatom];
00284             thee->qsm[jatom][thee->nqsm[jatom]] = isimp;
00285             thee->nqsm[jatom]++;
00286         }
00287     }
00288     thee->initFlag = 1;
00289 }
00290
00291
00292 VPUBLIC void Vcsn_dtor(Vcsn **thee) {
00293     if ((*thee) != VNULL) {
00294         Vcsn_dtor2(*thee);
00295         Vmem_free(VNULL, 1, sizeof(Vcsn), (void **)thee);
00296         (*thee) = VNULL;
00297     }
00298 }
00299
00300 VPUBLIC void Vcsn_dtor2(Vcsn *thee) {
00301     int i;
00302
00303     if ((thee != VNULL) && thee->initFlag) {
00304         for (i=0; i<thee->msimp; i++) {
00305             if (thee->nsqm[i] > 0) Vmem_free(thee->vmem, thee->nsqm[i],
00306                 sizeof(int), (void **)&(thee->sqm[i]));
00307         }
00308         for (i=0; i<thee->natom; i++) {
00309             if (thee->nqsm[i] > 0) Vmem_free(thee->vmem, thee->nqsm[i],
00310                 sizeof(int), (void **)&(thee->qsm[i]));
00311         }
00312         Vmem_free(thee->vmem, thee->msimp, sizeof(int *),
00313             (void **)&(thee->sqm));
00314         Vmem_free(thee->vmem, thee->msimp, sizeof(int),
00315             (void **)&(thee->nsqm));
00316         Vmem_free(thee->vmem, thee->natom, sizeof(int *),
00317             (void **)&(thee->qsm));
00318         Vmem_free(thee->vmem, thee->natom, sizeof(int),
00319             (void **)&(thee->nqsm));
00320     }
00321     Vmem_dtor(&(thee->vmem));
00322 }
00323
00324 }
00325
00326 VPUBLIC int Vcsn_update(Vcsn *thee, SS **sims, int num) {
00327     /* Counters */
00328     int isimp, jsimp, iatom, jatom, atomID, simpID;
00329     int nsimps, gotMem;
00330     /* Object info */
00331     Vatom *atom;
00332     SS *simplex;
00333     double *position;
00334     /* Lists */
00335     int *qParent, nqParent;
00336     int **sqmNew, *nsqmNew;
00337     int *affAtoms, nAffAtoms;
00338     int *dnqsm, *nqsmNew, **qsmNew;
00339
00340     VASSERT(thee != VNULL);
00341     VASSERT(thee->initFlag);
00342
00343     /* If we don't have enough memory to accommodate the new entries,
00344      * add more by doubling the existing amount */
00345     isimp = thee->nsimp + num - 1;
00346     gotMem = 0;
00347     while (!gotMem) {
00348         if (isimp > thee->msimp) {
00349             isimp = 2 * isimp;
00350             thee->nsqm = (int*)Vmem_realloc(thee->vmem, thee->msimp, sizeof(int),
00351                 (void **)&(thee->nsqm), isimp);
00352             VASSERT(thee->nsqm != VNULL);
00353             thee->sqm = (int**)Vmem_realloc(thee->vmem, thee->msimp, sizeof(int *),
00354                 (void **)&(thee->sqm), isimp);
00355             VASSERT(thee->sqm != VNULL);
00356             thee->msimp = isimp;
00357         } else gotMem = 1;
00358     }
00359 }

```

```

00360      /* Initialize the nsqm entires we just allocated */
00361      for (isimp = thee->nsimp; isimp<thee->nsimp+num-1 ; isimp++) {
00362          thee->nsqm[isimp] = 0;
00363      }
00364
00365      thee->nsimp = thee->nsimp + num - 1;
00366
00367      /* There's a simple case to deal with: if simps[0] didn't have a
00368       * charge in the first place */
00369      isimp = SS_id(simps[0]);
00370      if (thee->nsqm[isimp] == 0) {
00371          for (isimp=1; isimp<num; isimp++) {
00372              thee->nsqm[SS_id(simps[isimp])] = 0;
00373          }
00374          return 1;
00375      }
00376
00377      /* The more complicated case has occured; the parent simplex had one or
00378       * more charges. First, generate the list of affected charges. */
00379      isimp = SS_id(simps[0]);
00380      nqParent = thee->nsqm[isimp];
00381      qParent = thee->sqm[isimp];
00382
00383      sqmNew = (int**)Vmem_malloc(thee->vmem, num, sizeof(int *));
00384      VASSERT(sqmNew != VNULL);
00385      nsqmNew = (int**)Vmem_malloc(thee->vmem, num, sizeof(int));
00386      VASSERT(nsqmNew != VNULL);
00387      for (isimp=0; isimp<num; isimp++) nsqmNew[isimp] = 0;
00388
00389      /* Loop throught the affected atoms to determine how many atoms each
00390       * simplex will get. */
00391      for (iatom=0; iatom<nqParent; iatom++) {
00392
00393          atomID = qParent[iatom];
00394          atom = Valist_getAtom(thee->alist, atomID);
00395          position = Vatom_getPosition(atom);
00396          nsimps = 0;
00397
00398          jsimp = 0;
00399
00400          for (isimp=0; isimp<num; isimp++) {
00401              simplex = simps[isimp];
00402              if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00403                  nsqmNew[isimp]++;
00404                  jsimp = 1;
00405              }
00406          }
00407
00408          VASSERT(jsimp != 0);
00409      }
00410
00411      /* Sanity check that we didn't lose any atoms... */
00412      iatom = 0;
00413      for (isimp=0; isimp<num; isimp++) iatom += nsqmNew[isimp];
00414      if (iatom < nqParent) {
00415          Vnm_print(2,"Vcsm_update: Lost %d (of %d) atoms!\n",
00416                  nqParent - iatom, nqParent);
00417          VASSERT(0);
00418      }
00419
00420      /* Allocate the storage */
00421      for (isimp=0; isimp<num; isimp++) {
00422          if (nsqmNew[isimp] > 0) {
00423              sqmNew[isimp] = (int**)Vmem_malloc(thee->vmem, nsqmNew[isimp],
00424                                                  sizeof(int));
00425              VASSERT(sqmNew[isimp] != VNULL);
00426          }
00427      }
00428
00429      /* Assign charges to simplices */
00430      for (isimp=0; isimp<num; isimp++) {
00431
00432          jsimp = 0;
00433          simplex = simps[isimp];
00434
00435          /* Loop over the atoms associated with the parent simplex */
00436          for (iatom=0; iatom<nqParent; iatom++) {
00437
00438              atomID = qParent[iatom];
00439              atom = Valist_getAtom(thee->alist, atomID);
00440              position = Vatom_getPosition(atom);

```

```

00441         if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00442             sqmNew[isimp][jsimp] = atomID;
00443             jsimp++;
00444         }
00445     }
00446 }
00447
00448 /* Update the QSM map using the old and new SQM lists */
00449 /* The affected atoms are those contained in the parent simplex; i.e.
00450  * thee->sqm[SS_id(simps[0])] */
00451 affAtoms = thee->sqm[SS_id(simps[0])];
00452 nAffAtoms = thee->nsgm[SS_id(simps[0])];
00453 /* Each of these atoms will go somewhere else; i.e., the entries in
00454  * thee->qsm are never destroyed and thee->nqsm never decreases.
00455  * However, it is possible that a subdivision could cause an atom to be
00456  * shared by two child simplices. Here we record the change, if any,
00457  * in the number of simplices associated with each atom. */
00458 dnqsm = (int*)Vmem_malloc(thee->vmem, nAffAtoms, sizeof(int));
00459 VASSERT(dnqsm != VNULL);
00460 nqsmNew = (int*)Vmem_malloc(thee->vmem, nAffAtoms, sizeof(int));
00461 VASSERT(nqsmNew != VNULL);
00462 qsmNew = (int**)Vmem_malloc(thee->vmem, nAffAtoms, sizeof(int*));
00463 VASSERT(qsmNew != VNULL);
00464 for (iatom=0; iatom<nAffAtoms; iatom++) {
00465     dnqsm[iatom] = -1;
00466     atomID = affAtoms[iatom];
00467     for (isimp=0; isimp<num; isimp++) {
00468         for (jatom=0; jatom<nsgmNew[isimp]; jatom++) {
00469             if (sqmNew[isimp][jatom] == atomID) dnqsm[iatom]++;
00470         }
00471     }
00472     VASSERT(dnqsm[iatom] > -1);
00473 }
00474 /* Setup the new entries in the array */
00475 for (iatom=0; iatom<nAffAtoms; iatom++) {
00476     atomID = affAtoms[iatom];
00477     qsmNew[iatom] = (int*)Vmem_malloc(thee->vmem,
00478         (dnqsm[iatom] + thee->nqsm[atomID]),
00479         sizeof(int));
00480     nqsmNew[iatom] = 0;
00481     VASSERT(qsmNew[iatom] != VNULL);
00482 }
00483 /* Fill the new entries in the array */
00484 /* First, do the modified entries */
00485 for (isimp=0; isimp<num; isimp++) {
00486     simpID = SS_id(simps[isimp]);
00487     for (iatom=0; iatom<nsgmNew[isimp]; iatom++) {
00488         atomID = sqmNew[isimp][iatom];
00489         for (jatom=0; jatom<nAffAtoms; jatom++) {
00490             if (atomID == affAtoms[jatom]) break;
00491         }
00492         if (jatom < nAffAtoms) {
00493             qsmNew[jatom][nqsmNew[jatom]] = simpID;
00494             nqsmNew[jatom]++;
00495         }
00496     }
00497 }
00498 /* Now do the unmodified entries */
00499 for (iatom=0; iatom<nAffAtoms; iatom++) {
00500     atomID = affAtoms[iatom];
00501     for (isimp=0; isimp<thee->nqsm[atomID]; isimp++) {
00502         for (jsimp=0; jsimp<num; jsimp++) {
00503             simpID = SS_id(simps[jsimp]);
00504             if (thee->qsm[atomID][isimp] == simpID) break;
00505         }
00506         if (jsimp == num) {
00507             qsmNew[iatom][nqsmNew[iatom]] = thee->qsm[atomID][isimp];
00508             nqsmNew[iatom]++;
00509         }
00510     }
00511 }
00512
00513 /* Replace the existing entries in the table. Do the QSM entires
00514  * first, since they require affAtoms = thee->sqm[simps[0]] */
00515 for (iatom=0; iatom<nAffAtoms; iatom++) {
00516     atomID = affAtoms[iatom];
00517     Vmem_free(thee->vmem, thee->nqsm[atomID], sizeof(int),
00518         (void *)&(thee->qsm[atomID]));
00519     thee->qsm[atomID] = qsmNew[iatom];
00520     thee->nqsm[atomID] = nqsmNew[iatom];
00521 }

```

```

00522     for (isimp=0; isimp<num; isimp++) {
00523         simpID = SS_id(simps[isimp]);
00524         if (thee->nsqm[simpID] > 0) Vmem_free(thee->vmem, thee->nsqm[simpID],
00525             sizeof(int), (void **)&(thee->sqm[simpID]));
00526         thee->sqm[simpID] = sqmNew[isimp];
00527         thee->nsqm[simpID] = nsqmNew[isimp];
00528     }
00529
00530     Vmem_free(thee->vmem, num, sizeof(int *), (void **)&sqmNew);
00531     Vmem_free(thee->vmem, num, sizeof(int), (void **)&nsqmNew);
00532     Vmem_free(thee->vmem, nAffAtoms, sizeof(int *), (void **)&qsmNew);
00533     Vmem_free(thee->vmem, nAffAtoms, sizeof(int), (void **)&nqsmNew);
00534     Vmem_free(thee->vmem, nAffAtoms, sizeof(int), (void **)&dnqsm);
00535
00536
00537     return 1;
00538
00539
00540 }

```

10.5 src/fem/vcsm.h File Reference

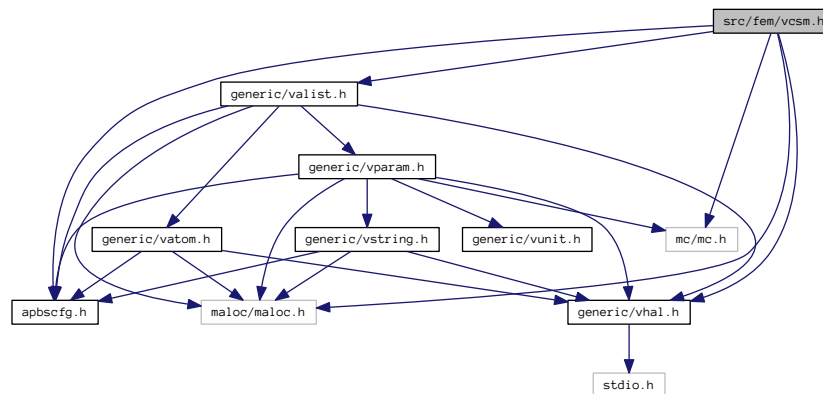
Contains declarations for the Vcsm class.

```

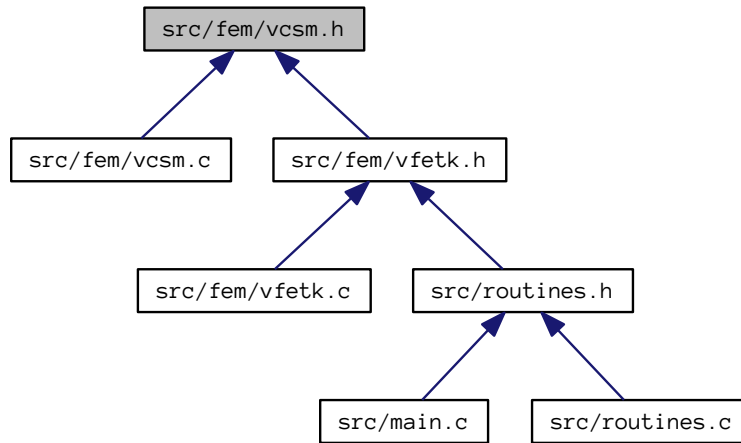
#include "apbscfg.h"
#include "malloc/malloc.h"
#include "mc/mc.h"
#include "generic/vhal.h"
#include "generic/valist.h"

```

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)
FORTTRAN stub to construct Vcsm object.
- VEXTERNC void [Vcsm_dtor](#) ([Vcsm](#) **thee)
Destroy Vcsm object.
- VEXTERNC void [Vcsm_dtor2](#) ([Vcsm](#) *thee)
FORTTRAN 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.

10.5.1 Detailed Description

Contains declarations for the Vcsm class.

Version

\$Id\$

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
```



```

*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vcsn.h](#).

10.6 vcsn.h

```

00001
00063 #ifndef _VCSN_H_
00064 #define _VCSN_H_
00065
00066 #include "apbscfg.h"
00067
00068 #include "malloc/malloc.h"
00069 #include "mc/mc.h"
00070
00071 #include "generic/vhal.h"
00072 #include "generic/valist.h"
00073
00078 VEXTERNC void Gem_setExternalUpdateFunction(
00079     Gem *thee,
00080     void (*externalUpdate)(SS **sims, int num)
00083 );
00084
00089 struct sVcsn {
00090
00091     Valist *alist;
00092     int natom;
00094     Gem *gm;
00097     int **sqm;
00104     int *nsqm;
00105     int nsimp;
00107     int msimp;
00109     int **qsm;
00111     int *nqsm;
00112     int initFlag;
00114     Vmem *vmem;
00116 };
00117
00122 typedef struct sVcsn Vcsn;
00123
00124 /* ////////////////////////////////////////////////////
00125 // Class Vcsn: Inlineable methods (vcsn.c)
00127
00128 #if !defined(VINLINE_VCSN)
00129
00135     VEXTERNC Valist* Vcsn_getValist(
00136         Vcsn *thee
00137     );
00138
00144     VEXTERNC int Vcsn_getNumberAtoms(
00145         Vcsn *thee,

```

```

00146         int isimp
00147     );
00148
00154     VEXTERNC Vatom* Vcsm_getAtom(
00155         Vcsm *thee,
00156         int iatom,
00157         int isimp
00158     );
00159
00165     VEXTERNC int Vcsm_getAtomIndex(
00166         Vcsm *thee,
00167         int iatom,
00168         int isimp
00169     );
00170
00176     VEXTERNC int Vcsm_getNumberSimplices(
00177         Vcsm *thee,
00178         int iatom
00179     );
00180
00186     VEXTERNC SS* Vcsm_getSimplex(
00187         Vcsm *thee,
00188         int isimp,
00189         int iatom
00190     );
00191
00197     VEXTERNC int Vcsm_getSimplexIndex(
00198         Vcsm *thee,
00199         int isimp,
00200         int iatom
00201     );
00202
00209     VEXTERNC unsigned long int Vcsm_memChk(
00210         Vcsm *thee
00211     );
00212
00213 #else /* if defined(VINLINE_VCSM) */
00214 #   define Vcsm_getValist(thee) ((thee)->alist)
00215 #   define Vcsm_getNumberAtoms(thee, isimp) ((thee)->nsqm[isimp])
00216 #   define Vcsm_getAtom(thee, iatom, isimp) (Valist_getAtom((thee)->alist, ((thee)->sqm)[isimp][iatom]))
00217 #   define Vcsm_getAtomIndex(thee, iatom, isimp) (((thee)->sqm)[isimp][iatom])
00218 #   define Vcsm_getNumberSimplices(thee, iatom) (((thee)->nqsm)[iatom])
00219 #   define Vcsm_getSimplex(thee, isimp, iatom) (Gem_SS((thee)->gm, ((thee)->qsm)[iatom][isimp]))
00220 #   define Vcsm_getSimplexIndex(thee, isimp, iatom) (((thee)->qsm)[iatom][isimp])
00221 #   define Vcsm_memChk(thee) (Vmem_bytes((thee)->vmem))
00222 #endif /* if !defined(VINLINE_VCSM) */
00223
00224 /* ////////////////////////////////////////
00225 // Class Vcsm: Non-Inlineable methods (vcsm.c)
00226
00236 VEXTERNC Vcsm* Vcsm_ctor(
00237     Valist *alist,
00238     Gem *gm
00239 );
00240
00249 VEXTERNC int Vcsm_ctor2(
00250     Vcsm *thee,
00251     Valist *alist,
00252     Gem *gm
00253 );
00254
00259 VEXTERNC void Vcsm_dtor(
00260     Vcsm **thee
00261 );
00262
00267 VEXTERNC void Vcsm_dtor2(
00268     Vcsm *thee
00269 );
00270
00277 VEXTERNC void Vcsm_init(
00278     Vcsm *thee
00279 );
00280
00287 VEXTERNC int Vcsm_update(
00288     Vcsm *thee,
00289     SS **simps,
00290     int num
00291 );
00292
00297 #endif /* ifndef _VCSM_H_ */

```


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

 - VPUBLIC double [Vfetk_energy](#) ([Vfetk](#) *thee, int color, int nonlin)

Return the total electrostatic energy.

 - VPUBLIC double [Vfetk_qfEnergy](#) ([Vfetk](#) *thee, int color)

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

 - VPUBLIC double [Vfetk_dqmEnergy](#) ([Vfetk](#) *thee, int color)

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

 - VPUBLIC void [Vfetk_setAtomColors](#) ([Vfetk](#) *thee)

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

 - VPUBLIC unsigned long int [Vfetk_memChk](#) ([Vfetk](#) *thee)

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

 - VPUBLIC Vrc_Codes [Vfetk_genCube](#) ([Vfetk](#) *thee, double center[3], double length[3], [Vfetk_MeshLoad](#) mesh↔ Type)

Construct a rectangular mesh (in the current [Vfetk](#) object)

 - VPUBLIC Vrc_Codes [Vfetk_loadMesh](#) ([Vfetk](#) *thee, double center[3], double length[3], [Vfetk_MeshLoad](#) mesh↔ Type, Vio *sock)

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

 - VPUBLIC void [Bmat_printHB](#) (Bmat *thee, char *fname)

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

 - VPUBLIC PDE * [Vfetk_PDE_ctor](#) ([Vfetk](#) *fetk)

Constructs the FEtk PDE object.

 - VPUBLIC int [Vfetk_PDE_ctor2](#) (PDE *thee, [Vfetk](#) *fetk)

Initializes the FEtk PDE object.

 - VPUBLIC void [Vfetk_PDE_dtor](#) (PDE **thee)

Destroys FEtk PDE object.

 - VPUBLIC void [Vfetk_PDE_dtor2](#) (PDE *thee)

FORTTRAN 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 td↔ U[] [3])
 - VPUBLIC void [Vfetk_PDE_Fu](#) (PDE *thee, int key, double F[])

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

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

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

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

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

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

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

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

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

- VPUBLIC double **Vfetk_PDE_DFu_wv** (PDE *thee, int key, double W[], double dW[][VAPBS_DIM], double V[], double dV[][3])
- VPUBLIC void **Vfetk_PDE_delta** (PDE *thee, int type, int chart, double txq[], void *user, double F[])
Evaluate a (discretized) delta function source term at the given point.
- VPUBLIC void **Vfetk_PDE_u_D** (PDE *thee, int type, int chart, double txq[], double F[])
Evaluate the Dirichlet boundary condition at the given point.
- VPUBLIC void **Vfetk_PDE_u_T** (PDE *thee, int type, int chart, double txq[], double F[])
Evaluate the "true solution" at the given point for comparison with the numerical solution.
- VPUBLIC void **Vfetk_PDE_bisectEdge** (int dim, int 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:

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

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

- VPUBLIC void **Vfetk_externalUpdateFunction** (SS **simps, int num)
External hook to simplex subdivision routines in Gem. Called each time a simplex is subdivided (we use it to update the charge-simplex map)
- VPUBLIC int **Vfetk_PDE_simplexBasisInit** (int key, int dim, int comp, int *ndof, int dof[])
Initialize the bases for the trial or the test space, for a particular component of the system, at all quadrature points on the master simplex element.
- VPUBLIC void **Vfetk_PDE_simplexBasisForm** (int key, int dim, int comp, int pdkey, double xq[], double basis[])
Evaluate the bases for the trial or test space, for a particular component of the system, at all quadrature points on the master simplex element.
- VPUBLIC void **Vfetk_dumpLocalVar** ()
Debugging routine to print out local variables used by PDE object.
- VPUBLIC int **Vfetk_fillArray** (Vfetk *thee, Bvec *vec, Vdata_Type type)
Fill an array with the specified data.
- VPUBLIC int **Vfetk_write** (Vfetk *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, Bvec *vec, Vdata_Format format)
Write out data.

Variables

- VPRIVATE **Vfetk_LocalVar** var
- VPRIVATE char * **diriCubeString**
- VPRIVATE char * **neumCubeString**
- VPRIVATE int **dim_2DP1** = 3
- VPRIVATE int **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]

10.7.1 Detailed Description

Class Vfetk methods.

Author

Nathan Baker

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
```

```

* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [vfetk.c](#).

10.7.2 Variable Documentation

10.7.2.1 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 [386](#) of file [vfetk.c](#).

10.7.2.2 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 },
{ 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 }
}
```

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

10.7.2.3 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 },
{ 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
}
```

Definition at line [402](#) of file [vfetk.c](#).

10.7.2.4 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 409 of file [vfetk.c](#).

10.7.2.5 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 438 of file [vfetk.c](#).

10.7.2.6 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 446 of file [vfetk.c](#).

10.7.2.7 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 454 of file [vfetk.c](#).

10.7.2.8 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 462 of file [vfetk.c](#).

10.8 vfetk.c

```
00001
00057 #include "vfetk.h"
00058
00059 /* Define the macro DONEUMANN to run with all-Neumann boundary conditions.
00060  * Set this macro at your own risk! */
00061 /* #define DONEUMANN 1 */
00062
00063 /*
00064  * @brief Calculate the contribution to the charge-potential energy from one
00065  * atom
00066  * @ingroup Vfetk
00067  * @author Nathan Baker
00068  * @param thee current Vfetk object
00069  * @param iatom current atom index
00070  * @param color simplex subset (partition) under consideration
00071  * @param sol current solution
00072  * @returns Per-atom energy
00073  */
00074 VPRIVATE double Vfetk_qfEnergyAtom(
00075     Vfetk *thee,
00076     int iatom,
00077     int color,
00078     double *sol
00079 );
00080
00081 /*
00082  * @brief Container for local variables
00083  * @ingroup Vfetk
00084  * @bug Not thread-safe
00085  */
00086 VPRIVATE Vfetk_LocalVar var;
00087
00088 /*
00089  * @brief MCSF-format cube mesh (all Dirichlet)
00090  * @ingroup Vfetk
00091  * @author Based on mesh by Mike Holst
00092  */
00093 VPRIVATE char *diriCubeString =
00094     "mcsf_begin=1;\n\
00095     \n\
00096     dim=3;\n\
00097     dimii=3;\n\
00098     vertices=8;\n\
00099     simplices=6;\n\
00100     \n\
00101     vert=[\n\
00102     0 0 -0.5 -0.5 -0.5\n\
00103     1 0 0.5 -0.5 -0.5\n\
00104     2 0 -0.5 0.5 -0.5\n\
00105     3 0 0.5 0.5 -0.5\n\
00106     4 0 -0.5 -0.5 0.5\n\
00107     5 0 0.5 -0.5 0.5\n\
00108     6 0 -0.5 0.5 0.5\n\
00109     7 0 0.5 0.5 0.5\n\
00110     ];\n\
00111     \n\
00112     simp=[\n\
00113     0 0 0 0 1 0 1 0 5 1 2\n\
00114     1 0 0 0 1 1 0 0 5 2 4\n\
00115     2 0 0 0 1 0 1 1 5 3 2\n\
00116     3 0 0 0 1 0 1 3 5 7 2\n\
00117     4 0 0 1 1 0 0 2 5 7 6\n\
00118     5 0 0 1 1 0 0 2 5 6 4\n\
```

```

00119 ];\n\
00120 \n\
00121 mcsf_end=1;\n\
00122 \n\
00123 ";
00124
00125 /*
00126  * @brief MCSF-format cube mesh (all Neumann)
00127  * @ingroup Vfetk
00128  * @author Based on mesh by Mike Holst
00129  */
00130 VPRIVATE char *neumCubeString =
00131 "mcsf_begin=1;\n\
00132 \n\
00133 dim=3;\n\
00134 dimii=3;\n\
00135 vertices=8;\n\
00136 simplices=6;\n\
00137 \n\
00138 vert=[\n\
00139 0 0 -0.5 -0.5 -0.5\n\
00140 1 0 0.5 -0.5 -0.5\n\
00141 2 0 -0.5 0.5 -0.5\n\
00142 3 0 0.5 0.5 -0.5\n\
00143 4 0 -0.5 -0.5 0.5\n\
00144 5 0 0.5 -0.5 0.5\n\
00145 6 0 -0.5 0.5 0.5\n\
00146 7 0 0.5 0.5 0.5\n\
00147 ];\n\
00148 \n\
00149 simp=[\n\
00150 0 0 0 0 2 0 2 0 5 1 2\n\
00151 1 0 0 0 2 2 0 0 5 2 4\n\
00152 2 0 0 0 2 0 2 1 5 3 2\n\
00153 3 0 0 0 2 0 2 3 5 7 2\n\
00154 4 0 0 2 2 0 0 2 5 7 6\n\
00155 5 0 0 2 2 0 0 2 5 6 4\n\
00156 ];\n\
00157 \n\
00158 mcsf_end=1;\n\
00159 \n\
00160 ";
00161
00162 /*
00163  * @brief Return the smoothed value of the dielectric coefficient at the
00164  * current point using a fast, chart-based method
00165  * @ingroup Vfetk
00166  * @author Nathan Baker
00167  * @returns Value of dielectric coefficient
00168  * @bug Not thread-safe
00169  */
00170 VPRIVATE double diel();
00171
00172 /*
00173  * @brief Return the smoothed value of the ion accessibility at the
00174  * current point using a fast, chart-based method
00175  * @ingroup Vfetk
00176  * @author Nathan Baker
00177  * @returns Value of mobile ion coefficient
00178  * @bug Not thread-safe
00179  */
00180 VPRIVATE double ionacc();
00181
00182 /*
00183  * @brief Smooths a mesh-based coefficient with a simple harmonic function
00184  * @ingroup Vfetk
00185  * @author Nathan Baker
00186  * @param meth Method for smoothing
00187  * \li 0 ==> arithmetic mean (gives bad results)
00188  * \li 1 ==> geometric mean
00189  * @param nverts Number of vertices
00190  * @param dist distance from point to each vertex
00191  * @param coeff coefficient value at each vertex
00192  * @note Thread-safe
00193  * @return smoothed value of coefficient at point of interest */
00194 VPRIVATE double smooth(
00195     int nverts,
00196     double dist[VAPBS_NVS],
00197     double coeff[VAPBS_NVS],
00198     int meth
00199 );

```

```

00200
00201
00202 /*
00203  * @brief Return the analytical multi-sphere Debye-Huckel approximation (in
00204  * kT/e) at the specified point
00205  * @ingroup Vftk
00206  * @author Nathan Baker
00207  * @param pbe Vpbe object
00208  * @param d Dimension of x
00209  * @param x Coordinates of point of interest (in &Aring;)
00210  * @note Thread-safe
00211  * @returns Multi-sphere Debye-Huckel potential in kT/e
00212  */
00213 VPRIVATE double debye_U(
00214     Vpbe *pbe,
00215     int d,
00216     double x[]
00217 );
00218
00219 /*
00220  * @brief Return the difference between the analytical multi-sphere
00221  * Debye-Huckel approximation and Coulomb's law (in kT/e) at the specified
00222  * point
00223  * @ingroup Vftk
00224  * @author Nathan Baker
00225  * @param pbe Vpbe object
00226  * @param d Dimension of x
00227  * @param x Coordinates of point of interest (in &Aring;)
00228  * @note Thread-safe
00229  * @returns Multi-sphere Debye-Huckel potential in kT/e */
00230 VPRIVATE double debye_Udiff(
00231     Vpbe *pbe,
00232     int d,
00233     double x[]
00234 );
00235
00236 /*
00237  * @brief Calculate the Coulomb's
00238  * Debye-Huckel approximation and Coulomb's law (in kT/e) at the specified
00239  * point
00240  * @ingroup Vftk
00241  * @author Nathan Baker
00242  * @param pbe Vpbe object
00243  * @param d Dimension of x
00244  * @param x Coordinates of point of interest (in &Aring;)
00245  * @param eps Dielectric constant
00246  * @param U Set to potential (in kT/e)
00247  * @param dU Set to potential gradient (in kT/e/&Aring;)
00248  * @param d2U Set to Laplacian of potential (in  $\frac{kT}{e} e^{-1} \AA^{-2}$ )
00249  * @returns Multi-sphere Debye-Huckel potential in kT/e */
00250 VPRIVATE void coulomb(
00251     Vpbe *pbe,
00252     int d,
00253     double x[],
00254     double eps,
00255     double *U,
00256     double dU[],
00257     double *d2U
00258 );
00259
00260 /*
00261  * @brief 2D linear master simplex information generator
00262  * @ingroup Vftk
00263  * @author Mike Holst
00264  * @param dimIS dunno
00265  * @param ndof dunno
00266  * @param dof dunno
00267  * @param c dunno
00268  * @param cx dunno
00269  * @note Trust in Mike */
00270 VPRIVATE void init_2DP1(
00271     int dimIS[],
00272     int *ndof,
00273     int dof[],
00274     double c[][VMAXP],
00275     double cx[][VMAXP],
00276     double cy[][VMAXP],
00277     double cz[][VMAXP]
00278 );
00279
00280 /*

```

```

00281 * @brief 3D linear master simplex information generator
00282 * @ingroup Vfetk
00283 * @author Mike Holst
00284 * @param dimIS dunno
00285 * @param ndof dunno
00286 * @param dof dunno
00287 * @param c dunno
00288 * @param cx dunno
00289 * @param cy dunno
00290 * @param cz dunno
00291 * @note Trust in Mike */
00292 VPRIVATE void init_3DP1(
00293     int dimIS[],
00294     int *ndof,
00295     int dof[],
00296     double c[][VMAXP],
00297     double cx[][VMAXP],
00298     double cy[][VMAXP],
00299     double cz[][VMAXP]
00300 );
00301
00302 /*
00303 * @brief Setup coefficients of polynomials from integer table data
00304 * @ingroup Vfetk
00305 * @author Mike Holst
00306 * @param numP dunno
00307 * @param c dunno
00308 * @param cx dunno
00309 * @param cy dunno
00310 * @param cz dunno
00311 * @param ic dunno
00312 * @param icx dunno
00313 * @param icy dunno
00314 * @param icz dunno
00315 * @note Trust in Mike */
00316 VPRIVATE void setCoef(
00317     int numP,
00318     double c[][VMAXP],
00319     double cx[][VMAXP],
00320     double cy[][VMAXP],
00321     double cz[][VMAXP],
00322     int ic[][VMAXP],
00323     int icx[][VMAXP],
00324     int icy[][VMAXP],
00325     int icz[][VMAXP]
00326 );
00327
00328 /*
00329 * @brief Evaluate a collection of at most cubic polynomials at a
00330 * specified point in at most R^3.
00331 * @ingroup Vfetk
00332 * @author Mike Holst
00333 * @param numP the number of polynomials to evaluate
00334 * @param p the results of the evaluation
00335 * @param c the coefficients of each polynomial
00336 * @param xv the point (x,y,z) to evaluate the polynomials.
00337 * @note Mike says:
00338 * <pre>
00339 * Note that "VMAXP" must be >= 19 for cubic polynomials.
00340 * The polynomials are build from the coefficients c[][] as
00341 * follows. To build polynomial "k", fix k and set:
00342 *
00343 * c0=c[k][0], c1=c[k][1], ... , cp=c[k][p]
00344 *
00345 * Then evaluate as:
00346 *
00347 * p3(x,y,z) = c0 + c1*x + c2*y + c3*z
00348 *             + c4*x*x + c5*y*y + c6*z*z + c7*x*y + c8*x*z + c9*y*z
00349 *             + c10*x*x*x + c11*y*y*y + c12*z*z*z
00350 *             + c13*x*x*y + c14*x*x*z + c15*x*y*y
00351 *             + c16*y*y*z + c17*x*z*z + c18*y*z*z
00352 * </pre>
00353 */
00354 VPRIVATE void polyEval(
00355     int numP,
00356     double p[],
00357     double c[][VMAXP],
00358     double xv[]
00359 );
00360
00361 /*

```

```

00362 * @brief I have no clue what this variable does, but we need it to initialize
00363 * the simplices
00364 * @ingroup Vfetk
00365 * @author Mike Holst */
00366 VPRIVATE int dim_2DP1 = 3;
00367
00368 /*
00369 * @brief I have no clue what these variable do, but we need it to initialize
00370 * the simplices
00371 * @ingroup Vfetk
00372 * @author Mike Holst
00373 * @note Mike says:
00374 * <pre>
00375 * 2D-P1 Basis:
00376 *
00377 *  $p_1(x,y) = c_0 + c_1x + c_2y$ 
00378 *
00379 * Lagrange Point      Lagrange Basis Function Definition
00380 * -----
00381 * (0, 0)               $p[0](x,y) = 1 - x - y$ 
00382 * (1, 0)               $p[1](x,y) = x$ 
00383 * (0, 1)               $p[2](x,y) = y$ 
00384 * </pre>
00385 */
00386 VPRIVATE int lgr_2DP1[3][VMAXP] = {
00387 /*c0 c1 c2 c3
00388 * ----- */
00389 /* 1 x y z
00390 * ----- */
00391 { 2, -2, -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00392 { 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00393 { 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
00394 };
00395 VPRIVATE int lgr_2DP1x[3][VMAXP] = {
00396 /*c0 ----- */
00397 /* 1 ----- */
00398 { -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00399 { 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00400 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
00401 };
00402 VPRIVATE int lgr_2DP1y[3][VMAXP] = {
00403 /*c0 ----- */
00404 /* 1 ----- */
00405 { -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00406 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00407 { 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
00408 };
00409 VPRIVATE int lgr_2DP1z[3][VMAXP] = {
00410 /*c0 ----- */
00411 /* 1 ----- */
00412 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00413 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00414 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
00415 };
00416
00417 /*
00418 * @brief I have no clue what these variable do, but we need it to initialize
00419 * the simplices
00420 * @ingroup Vfetk
00421 * @author Mike Holst
00422 * @note Mike says:
00423 * <pre>
00424 * 3D-P1 Basis:
00425 *
00426 *  $p_1(x,y,z) = c_0 + c_1x + c_2y + c_3z$ 
00427 *
00428 * Lagrange Point      Lagrange Basis Function Definition
00429 * -----
00430 * (0, 0, 0)            $p[0](x,y,z) = 1 - x - y - z$ 
00431 * (1, 0, 0)            $p[1](x,y,z) = x$ 
00432 * (0, 1, 0)            $p[2](x,y,z) = y$ 
00433 * (0, 0, 1)            $p[3](x,y,z) = z$ 
00434 * </pre>
00435 */
00436
00437 VPRIVATE int dim_3DP1 = VAPBS_NVS;
00438 VPRIVATE int lgr_3DP1[VAPBS_NVS][VMAXP] = {
00439 /*c0 c1 c2 c3 ----- */
00440 /* 1 x y z ----- */
00441 { 2, -2, -2, -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00442 { 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },

```

```

00443 { 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00444 { 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
00445 };
00446 VPRIVATE int lgr_3DP1x[VAPBS_NVS][VMAXP] = {
00447 /*c0 ----- */
00448 /* 1 ----- */
00449 { -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00450 { 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00451 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00452 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
00453 };
00454 VPRIVATE int lgr_3DP1y[VAPBS_NVS][VMAXP] = {
00455 /*c0 ----- */
00456 /* 1 ----- */
00457 { -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00458 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00459 { 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00460 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
00461 };
00462 VPRIVATE int lgr_3DP1z[VAPBS_NVS][VMAXP] = {
00463 /*c0 ----- */
00464 /* 1 ----- */
00465 { -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00466 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00467 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00468 { 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
00469 };
00470
00471 /*
00472  * @brief Another Holst variable
00473  * @ingroup Vfetk
00474  * @author Mike Holst
00475  * @note Mike says: 1 = linear, 2 = quadratic */
00476 VPRIVATE const int P_DEG=1;
00477
00478 /*
00479  * @brief Another Holst variable
00480  * @ingroup Vfetk
00481  * @author Mike Holst */
00482 VPRIVATE int numP;
00483 VPRIVATE double c[VMAXP][VMAXP];
00484 VPRIVATE double cx[VMAXP][VMAXP];
00485 VPRIVATE double cy[VMAXP][VMAXP];
00486 VPRIVATE double cz[VMAXP][VMAXP];
00487
00488 #if !defined(VINLINE_VFETK)
00489
00490 VPUBLIC Gem* Vfetk_getGem(Vfetk *thee) {
00491     VASSERT(thee != VNULL);
00492     return thee->gm;
00493 }
00494
00495
00496
00497 VPUBLIC AM* Vfetk_getAM(Vfetk *thee) {
00498     VASSERT(thee != VNULL);
00499     return thee->am;
00500 }
00501
00502
00503 VPUBLIC Vpbe* Vfetk_getVpbe(Vfetk *thee) {
00504     VASSERT(thee != VNULL);
00505     return thee->pbe;
00506 }
00507
00508
00509
00510 VPUBLIC Vcsm* Vfetk_getVcsm(Vfetk *thee) {
00511     VASSERT(thee != VNULL);
00512     return thee->csm;
00513 }
00514
00515
00516
00517 VPUBLIC int Vfetk_getAtomColor(Vfetk *thee,
                                int iatom
                                ) {
00518     int natoms;
00519     VASSERT(thee != VNULL);

```

```

00524
00525     natoms = Valist_getNumberAtoms(Vpbe_getValist(thee->
00526     pbe));
00527     VASSERT(iatom < natoms);
00528     return Vatom_getPartID(Valist_getAtom(
00529     Vpbe_getValist(thee->pbe), iatom));
00530 }
00531 #endif /* if !defined(VINLINE_VFETK) */
00532 VPUBLIC Vfetk* Vfetk_ctor(Vpbe *pbe,
00533     Vhal_PBEType type
00534     ) {
00535
00536     /* Set up the structure */
00537     Vfetk *thee = VNULL;
00538     thee = (Vfetk*)Vmem_malloc(VNULL, 1, sizeof(Vfetk) );
00539     VASSERT(thee != VNULL);
00540     VASSERT(Vfetk_ctor2(thee, pbe, type));
00541
00542     return thee;
00543 }
00544
00545 VPUBLIC int Vfetk_ctor2(Vfetk *thee,
00546     Vpbe *pbe,
00547     Vhal_PBEType type
00548     ) {
00549
00550     int i;
00551     double center[VAPBS_DIM];
00552
00553     /* Make sure things have been properly initialized & store them */
00554     VASSERT(pbe != VNULL);
00555     thee->pbe = pbe;
00556     VASSERT(pbe->alist != VNULL);
00557     VASSERT(pbe->acc != VNULL);
00558
00559     /* Store PBE type */
00560     thee->type = type;
00561
00562     /* Set up memory management object */
00563     thee->vmem = Vmem_ctor("APBS::VFETK");
00564
00565     /* Set up FETk objects */
00566     Vnm_print(0, "Vfetk_ctor2: Constructing PDE...\n");
00567     thee->pde = Vfetk_PDE_ctor(thee);
00568     Vnm_print(0, "Vfetk_ctor2: Constructing Gem...\n");
00569     thee->gm = Gem_ctor(thee->vmem, thee->pde);
00570     Vnm_print(0, "Vfetk_ctor2: Constructing Aprx...\n");
00571     thee->aprx = Aprx_ctor(thee->vmem, thee->gm, thee->pde);
00572     Vnm_print(0, "Vfetk_ctor2: Constructing Aprx...\n");
00573     thee->am = AM_ctor(thee->vmem, thee->aprx);
00574
00575     /* Reset refinement level */
00576     thee->level = 0;
00577
00578     /* Set default solver variables */
00579     thee->lkey = VLT_MG;
00580     thee->lmax = 1000000;
00581     thee->ltol = 1e-5;
00582     thee->lprec = VPT_MG;
00583     thee->nkey = VNT_NEW;
00584     thee->nmax = 1000000;
00585     thee->ntol = 1e-5;
00586     thee->gues = VGT_ZERO;
00587     thee->pjac = -1;
00588
00589     /* Store local copy of myself */
00590     var.fetk = thee;
00591     var.initGreen = 0;
00592
00593     /* Set up the external Gem subdivision hook */
00594     Gem_setExternalUpdateFunction(thee->gm,
00595     Vfetk_externalUpdateFunction);
00596
00597     /* Set up ion-related variables */
00598     var.zkappa2 = Vpbe_getZkappa2(var.fetk->pbe);
00599     var.ionstr = Vpbe_getBulkIonicStrength(var.
00600     fetk->pbe);
00599     if (var.ionstr > 0.0) var.zks2 = 0.5*var.zkappa2/var.ionstr;
00600     else var.zks2 = 0.0;

```

```

00601     Vpbe_getIons(var.fetk->pbe, &(var.nion), var.ionConc, var.
ionRadii,
00602     var.ionQ);
00603     for (i=0; i<var.nion; i++) {
00604         var.ionConc[i] = var.zks2 * var.ionConc[i] * var.ionQ[i];
00605     }
00606
00607     /* Set uninitialized objects to NULL */
00608     thee->pbeparm = VNULL;
00609     thee->feparm = VNULL;
00610     thee->csm = VNULL;
00611
00612     return 1;
00613 }
00614
00615 VPUBLIC void Vfetk_setParameters(Vfetk *thee,
                                PBEparm *pbeparm,
                                FEMparm *feparm
                                ) {
00616
00620     VASSERT(thee != VNULL);
00621     thee->feparm = feparm;
00622     thee->pbeparm = pbeparm;
00623 }
00624
00625 VPUBLIC void Vfetk_dtor(Vfetk **thee) {
00626     if ((*thee) != VNULL) {
00627         Vfetk_dtor2(*thee);
00628         //Vmem_free(VNULL, 1, sizeof(Vfetk), (void **)thee);
00629         (*thee) = VNULL;
00630     }
00631 }
00632
00633 VPUBLIC void Vfetk_dtor2(Vfetk *thee) {
00634     Vcsm_dtor(&(thee->csm));
00635     AM_dtor(&(thee->am));
00636     Aprx_dtor(&(thee->apr));
00637     Vfetk_PDE_dtor(&(thee->pde));
00638     Vmem_dtor(&(thee->vmem));
00639 }
00640
00641 VPUBLIC double* Vfetk_getSolution(Vfetk *thee,
                                int *length
                                ) {
00642
00643     int i;
00644     double *solution,
00645           *theAnswer;
00646     AM *am;
00647
00650     VASSERT(thee != VNULL);
00651
00652     /* Get the AM object */
00653     am = thee->am;
00654     /* Copy the solution into the w0 vector */
00655     Bvec_copy(am->w0, am->u);
00656     /* Add the Dirichlet conditions */
00657     Bvec_axpy(am->w0, am->ud, 1.);
00658     /* Get the data from the Bvec */
00659     solution = Bvec_addr(am->w0);
00660     /* Get the length of the data from the Bvec */
00661     *length = Bvec_numRT(am->w0);
00662     /* Make sure that we got scalar data (only one block) for the solution
00663      * to the FETK */
00664     VASSERT(1 == Bvec_numB(am->w0));
00665     /* Allocate space for the returned vector and copy the solution into it */
00666     theAnswer = VNULL;
00667     theAnswer = (double*)Vmem_malloc(VNULL, *length, sizeof(double));
00668     VASSERT(theAnswer != VNULL);
00669     for (i=0; i<(*length); i++) theAnswer[i] = solution[i];
00670
00671     return theAnswer;
00672 }
00673
00674
00693 VPUBLIC double Vfetk_energy(Vfetk *thee,
                                int color,
                                int nonlin
                                ) {
00694
00701     double totEnergy = 0.0,

```



```

00703         qfEnergy = 0.0,
00704         dqmEnergy = 0.0;
00706     VASSERT(thee != VNULL);
00707
00708     if (nonlin && (Vpbe_getBulkIonicStrength(thee->pbe) > 0.)) {
00709         Vnm_print(0, "Vfetk_energy: calculating full PBE energy\n");
00710         Vnm_print(0, "Vfetk_energy: bulk ionic strength = %g M\n",
00711             Vpbe_getBulkIonicStrength(thee->pbe));
00712         dqmEnergy = Vfetk_dqmEnergy(thee, color);
00713         Vnm_print(0, "Vfetk_energy: dqmEnergy = %g kT\n", dqmEnergy);
00714         qfEnergy = Vfetk_qfEnergy(thee, color);
00715         Vnm_print(0, "Vfetk_energy: qfEnergy = %g kT\n", qfEnergy);
00716
00717         totEnergy = qfEnergy - dqmEnergy;
00718     } else {
00719         Vnm_print(0, "Vfetk_energy: calculating only q-phi energy\n");
00720         dqmEnergy = Vfetk_dqmEnergy(thee, color);
00721         Vnm_print(0, "Vfetk_energy: dqmEnergy = %g kT (NOT USED)\n", dqmEnergy);
00722         qfEnergy = Vfetk_qfEnergy(thee, color);
00723         Vnm_print(0, "Vfetk_energy: qfEnergy = %g kT\n", qfEnergy);
00724         totEnergy = 0.5*qfEnergy;
00725     }
00726
00727     return totEnergy;
00728 }
00729 }
00730
00731
00732 VPUBLIC double Vfetk_qfEnergy(Vfetk *thee,
00733     int color
00734 ) {
00735
00736     double *sol,
00737         energy = 0.0;
00738     int nsol,
00739         iatom,
00740         natoms;
00741     AM *am;
00742
00743     VASSERT(thee != VNULL);
00744     am = thee->am;
00745
00746     /* Get the finest level solution */
00747     sol = VNULL;
00748     sol = Vfetk_getSolution(thee, &nsol);
00749     VASSERT(sol != VNULL);
00750
00751     /* Make sure the number of entries in the solution array matches the
00752      * number of vertices currently in the mesh */
00753     if (nsol != Gem_numVV(thee->gm)) {
00754         Vnm_print(2, "Vfetk_qfEnergy: Number of unknowns in solution does not match\n");
00755         Vnm_print(2, "Vfetk_qfEnergy: number of vertices in mesh!!! Bailing out!\n");
00756         VASSERT(0);
00757     }
00758
00759     /* Now we do the sum over atoms... */
00760     natoms = Valist_getNumberAtoms(thee->pbe->alist);
00761     for (iatom=0; iatom<natoms; iatom++) {
00762
00763         energy = energy + Vfetk_qfEnergyAtom(thee, iatom, color, sol);
00764
00765     } /* end for iatom */
00766
00767     /* Destroy the finest level solution */
00768     Vmem_free(VNULL, nsol, sizeof(double), (void **)&sol);
00769
00770     /* Return the energy */
00771     return energy;
00772 }
00773
00774 VPRIVATE double Vfetk_qfEnergyAtom(
00775     Vfetk *thee,
00776     int iatom,
00777     int color,
00778     double *sol) {
00779
00780     Vatom *atom;
00781     double charge,
00782         phi[VAPBS_NVs],
00783         phix[VAPBS_NVs][3],
00784         *position,

```

```

00785         uval,
00786         energy = 0.0;
00787     int isimp,
00788         nsimps,
00789         icolor,
00790         ivert,
00791         usingColor;
00792     SS *simp;
00793
00794
00795     /* Get atom information */
00796     atom = Valist_getAtom(thee->pbe->alist, iatom);
00797     icolor = Vfetc_getAtomColor(thee, iatom);
00798     charge = Vatom_getCharge(atom);
00799     position = Vatom_getPosition(atom);
00800
00801     /* Find out if we're using colors */
00802     usingColor = (color >= 0);
00803
00804     if (usingColor && (icolor<0)) {
00805         Vnm_print(2, "Vfetc_qfEnergy: Atom colors not set!\n");
00806         VASSERT(0);
00807     }
00808
00809     /* Check if this atom belongs to the specified partition */
00810     if ((icolor==color) || (!usingColor)) {
00811         /* Loop over the simps associated with this atom */
00812         nsimps = Vcsm_getNumberSimplices(thee->csm, iatom);
00813
00814         /* Get the first simp of the correct color; we can use just one
00815          * simplex for energy evaluations, but not for force
00816          * evaluations */
00817         for (isimp=0; isimp<nsimps; isimp++) {
00818
00819             simp = Vcsm_getSimplex(thee->csm, isimp, iatom);
00820
00821             /* If we've asked for a particular partition AND if the atom
00822              * is our partition, then compute the energy */
00823             if ((SS_chart(simp)==color) || (color<0)) {
00824                 /* Get the value of each basis function evaluated at this
00825                  * point */
00826                 Gem_pointInSimplexVal(thee->gm, simp, position, phi, phix);
00827                 for (ivert=0; ivert<SS_dimVV(simp); ivert++) {
00828                     uval = sol[VV_id(SS_vertex(simp,ivert))];
00829                     energy += (charge*phi[ivert]*uval);
00830                 } /* end for ivert */
00831                 /* We only use one simplex of the appropriate color for
00832                  * energy calculations, so break here */
00833                 break;
00834             } /* endif (color) */
00835         } /* end for isimp */
00836     }
00837
00838     return energy;
00839 }
00840
00841
00842 VPUBLIC double Vfetc_dqmEnergy(Vfetc *thee,
00843                               int color) {
00844
00845     return AM_evalJ(thee->am);
00846 }
00847
00848
00849 VPUBLIC void Vfetc_setAtomColors(Vfetc *thee) {
00850
00851     SS *simp;
00852     Vatom *atom;
00853     int i,
00854         natoms;
00855
00856     VASSERT(thee != VNULL);
00857
00858     natoms = Valist_getNumberAtoms(thee->pbe->alist);
00859     for (i=0; i<natoms; i++) {
00860         atom = Valist_getAtom(thee->pbe->alist, i);
00861         simp = Vcsm_getSimplex(thee->csm, 0, i);
00862         Vatom_setPartID(atom, SS_chart(simp));
00863     }
00864
00865 }

```

```

00866
00867 VPUBLIC unsigned long int Vfetk_memChk(Vfetk *thee) {
00868
00869     int memUse = 0;
00870
00871     if (thee == VNULL) return 0;
00872
00873     memUse = memUse + sizeof(Vfetk);
00874     memUse = memUse + Vcsm_memChk(thee->csm);
00875
00876     return memUse;
00877 }
00878
00885 VPUBLIC Vrc_Codes Vfetk_genCube(Vfetk *thee,
00886                                  double center[3],
00887                                  double length[3],
00888                                  Vfetk_MeshLoad meshType
00889                                  ) {
00890
00891     VASSERT(thee != VNULL);
00892
00893     AM *am = VNULL; /* @todo - no idea what this is */
00894     Gem *gm = VNULL; /* Geometry manager */
00895
00896     int skey = 0, /* Simplex format */
00897         bufsize = 0, /* Buffer size */
00898         i, /* Loop counter */
00899         j; /* Loop counter */
00900     char *key = "r", /* Read */
00901         *iodev = "BUFF", /* Buffer */
00902         *iofmt = "ASC", /* ASCII */
00903         *iohost = "localhost", /* localhost (dummy) */
00904         *iofile = "0", /*< socket 0 (dummy) */
00905         buf[VMAX_BUFSIZE]; /* Socket buffer */
00906     Vio *sock = VNULL; /* Socket object */
00907     VV *vx = VNULL; /* @todo - no idea what this is */
00908     double x;
00909
00910     am = thee->am;
00911     VASSERT(am != VNULL);
00912     gm = thee->gm;
00913     VASSERT(gm != VNULL);
00914
00915     /* @note This code is based on Gem_makeCube by Mike Holst */
00916     /* Write mesh string to buffer and read back */
00917     switch (meshType) {
00918     case VML_DIRICUBE:
00919         /* Create a new copy of the DIRICUBE mesh (see globals higher in this file) */
00920         bufsize = strlen(diriCubeString);
00921         VASSERT( bufsize <= VMAX_BUFSIZE );
00922         strncpy(buf, diriCubeString, VMAX_BUFSIZE);
00923         break;
00924     case VML_NEUMCUBE:
00925         /* Create a new copy of the NEUMCUBE mesh (see globals higher in this file) */
00926         bufsize = strlen(neumCubeString);
00927         Vnm_print(2, "Vfetk_genCube: WARNING! USING EXPERIMENTAL NEUMANN BOUNDARY CONDITIONS!\n");
00928         VASSERT( bufsize <= VMAX_BUFSIZE );
00929         strncpy(buf, neumCubeString, VMAX_BUFSIZE);
00930         break;
00931     case VML_EXTERNAL:
00932         Vnm_print(2, "Vfetk_genCube: Got request for external mesh!\n");
00933         Vnm_print(2, "Vfetk_genCube: How did we get here?\n");
00934         return VRC_FAILURE;
00935     default:
00936         Vnm_print(2, "Vfetk_genCube: Unknown mesh type (%d)\n", meshType);
00937         return VRC_FAILURE;
00938     }
00939
00940     VASSERT( VNULL != (sock=Vio_socketOpen(key,iodev,iofmt,iohost,iofile)) ); /* Open socket */
00941     Vio_bufTake(sock, buf, bufsize); /* Initialize internal buffer for socket */
00942     AM_read(am, skey, sock); /* Take the initial mesh from the socket and load
00943                               into internal AM data structure with simplex
00944                               format */
00945     Vio_connectFree(sock); /* Purge output buffers */
00946     Vio_bufGive(sock); /* Get pointer to output buffer? No assignment of return value... */
00947     Vio_dtor(&sock); /* Destroy output buffer */
00948
00949     /* @todo - could the following be done in a single pass? - PCE */
00950     /* Scale (unit) cube - for each vertex, set the new coordinates of that
00951        vertex based on the vertex length */
00952     for (i=0; i<Gem_numVV(gm); i++) {

```

```

00953     vx = Gem_VV(gm, i);
00954     for (j=0; j<3; j++) {
00955         x = VV_coord(vx, j);
00956         x *= length[j];
00957         VV_setCoord(vx, j, x);
00958     }
00959 }
00960
00961 /* Add new center - for each vertex, set a new center for the vertex */
00962 for (i=0; i<Gem_numVV(gm); i++) {
00963     vx = Gem_VV(gm, i);
00964     for (j=0; j<3; j++) {
00965         x = VV_coord(vx, j);
00966         x += center[j];
00967         VV_setCoord(vx, j, x);
00968     }
00969 }
00970
00971 return VRC_SUCCESS;
00972 }
00973
00980 VPUBLIC Vrc_Codes Vfetc_loadMesh(Vfetc *thee, /* Vfetc object to load into */
00981                                 double center[3], /* Center for mesh (if constructed) */
00982                                 double length[3], /* Mesh lengths (if constructed) */
00983                                 Vfetc_MeshLoad meshType, /* Type of mesh to load */
00984                                 Vio *sock /* Socket for external mesh data (NULL otherwise) */
00985                                 ) {
00986
00987     Vrc_Codes vrc; /* Function return codes - see vhal.h for enum */
00988     int skey = 0; /* Simplex format */
00989
00990     /* Load mesh from socket if external mesh, otherwise generate mesh */
00991     switch (meshType) {
00992     case VML_EXTERNAL:
00993         if (sock == VNULL) {
00994             Vnm_print(2, "Vfetc_loadMesh: Got NULL socket!\n");
00995             return VRC_FAILURE;
00996         }
00997         AM_read(thee->am, skey, sock);
00998         Vio_connectFree(sock);
00999         Vio_bufGive(sock);
01000         Vio_dtor(&sock);
01001         break;
01002     case VML_DIRICUBE:
01003     case VML_NEUMCUBE:
01004         /* Create new mesh and store in thee */
01005         vrc = Vfetc_genCube(thee, center, length, meshType);
01006         if (vrc == VRC_FAILURE) return VRC_FAILURE;
01007         break;
01008     default:
01009         Vnm_print(2, "Vfetc_loadMesh: unrecognized mesh type (%d)!\n",
01010                 meshType);
01011         return VRC_FAILURE;
01012     };
01013
01014     /* Setup charge-simplex map */
01015     Vnm_print(0, "Vfetc_ctor2: Constructing Vcsm...\n");
01016     thee->csm = VNULL;
01017     /* Construct a new Vcsm with the atom list and gem data */
01018     thee->csm = Vcsm_ctor(Vpbe_getValist(thee->pbe), thee->
gm);
01019     VASSERT(thee->csm != VNULL);
01020     Vcsm_init(thee->csm);
01021
01022     return VRC_SUCCESS;
01023 }
01024
01025
01026 VPUBLIC void Bmat_printHB(Bmat *thee,
01027                          char *fname
01028                          ) {
01029
01030     Mat *Ablock;
01031     MATsym pqsym;
01032     int i, j, jj;
01033     int *IA, *JA;
01034     double *D, *L, *U;
01035     FILE *fp;
01036
01037     char mmttitle[72];
01038     char mmkey[] = {"8charkey"};

```

```

01039     int totc = 0, ptrc = 0, indc = 0, valc = 0;
01040     char mxtyp[] = {"RUA"}; /* Real Unsymmetric Assembled */
01041     int nrow = 0, ncol = 0, numZ = 0;
01042     int numZdigits = 0, nrowdigits = 0;
01043     int nptrline = 8, nindline = 8, nvalline = 5;
01044     char ptrfmt[] = {"(8I10)"}; ptrfmtstr[] = {"%10d"};
01045     char indfmt[] = {"(8I10)"}; indfmtstr[] = {"%10d"};
01046     char valfmt[] = {"(5E16.8)"}; valfmtstr[] = {"%16.8E"};
01047
01048     VASSERT( thee->numB == 1 ); /* HARDWARE FOR NOW */
01049     Ablock = thee->AD[0][0];
01050
01051     VASSERT( Mat_format( Ablock ) == DRC_FORMAT ); /* HARDWARE FOR NOW */
01052
01053     pqsym = Mat_sym( Ablock );
01054
01055     if ( pqsym == IS_SYM ) {
01056         mxtyp[1] = 'S';
01057     } else if ( pqsym == ISNOT_SYM ) {
01058         mxtyp[1] = 'U';
01059     } else {
01060         VASSERT( 0 ); /* NOT VALID */
01061     }
01062
01063     nrow = Bmat_numRT( thee ); /* Number of rows */
01064     ncol = Bmat_numCT( thee ); /* Number of cols */
01065     numZ = Bmat_numZT( thee ); /* Number of entries */
01066
01067     nrowdigits = (int) (log( nrow )/log( 10 )) + 1;
01068     numZdigits = (int) (log( numZ )/log( 10 )) + 1;
01069
01070     nptrline = (int) ( 80 / (numZdigits + 1) );
01071     nindline = (int) ( 80 / (nrowdigits + 1) );
01072
01073     sprintf(ptrfmt,"%dI%d",nptrline,numZdigits+1);
01074     sprintf(ptrfmtstr,"%%dd",numZdigits+1);
01075     sprintf(indfmt,"%dI%d",nindline,nrowdigits+1);
01076     sprintf(indfmtstr,"%%dd",nrowdigits+1);
01077
01078     ptrc = (int) ( ( ncol + 1 ) - 1 ) / nptrline ) + 1;
01079     indc = (int) ( ( numZ - 1 ) / nindline ) + 1;
01080     valc = (int) ( ( numZ - 1 ) / nvalline ) + 1;
01081
01082     totc = ptrc + indc + valc;
01083
01084     sprintf( mmtitle, "Sparse '%s' Matrix - Harwell-Boeing Format - '%s'",
01085             thee->name, fname );
01086
01087     /* Step 0: Open the file for writing */
01088
01089     fp = fopen( fname, "w" );
01090     if (fp == VNULL) {
01091         Vnm_print(2,"Bmat_printHB: Ouch couldn't open file <%s>\n",fname);
01092         return;
01093     }
01094
01095     /* Step 1: Print the header information */
01096
01097     fprintf( fp, "%-72s%-8s\n", mmtitle, mmkey );
01098     fprintf( fp, "%14d%14d%14d%14d\n", totc, ptrc, indc, valc, 0 );
01099     fprintf( fp, "%3s%11s%14d%14d%14d\n", mxtyp, " ", nrow, ncol, numZ );
01100     fprintf( fp, "%-16s%-16s%-20s%-20s\n", ptrfmt, indfmt, valfmt, "6E13.5" );
01101
01102     IA = Ablock->IA;
01103     JA = Ablock->JA;
01104     D = Ablock->diag;
01105     L = Ablock->offL;
01106     U = Ablock->offU;
01107
01108     if ( pqsym == IS_SYM ) {
01109
01110         /* Step 2: Print the pointer information */
01111
01112         for (i=0; i<(ncol+1); i++) {
01113             fprintf( fp, ptrfmtstr, Ablock->IA[i] + (i+1) );
01114             if ( ( i+1 ) % nptrline ) == 0 ) {
01115                 fprintf( fp, "\n" );
01116             }
01117         }
01118
01119         if ( ( ( ncol+1 ) % nptrline ) != 0 ) {

```

```

01120         fprintf( fp, "\n" );
01121     }
01122
01123     /* Step 3: Print the index information */
01124
01125     j = 0;
01126     for (i=0; i<ncol; i++) {
01127         fprintf( fp, indfmtstr, i+1); /* diagonal */
01128         if ( ( (j+1) % nindline ) == 0 ) {
01129             fprintf( fp, "\n" );
01130         }
01131         j++;
01132         for (jj=IA[i]; jj<IA[i+1]; jj++) {
01133             fprintf( fp, indfmtstr, JA[jj] + 1 ); /* lower triangle */
01134             if ( ( (j+1) % nindline ) == 0 ) {
01135                 fprintf( fp, "\n" );
01136             }
01137             j++;
01138         }
01139     }
01140
01141     if ( ( j % nindline ) != 0 ) {
01142         fprintf( fp, "\n" );
01143     }
01144
01145     /* Step 4: Print the value information */
01146
01147     j = 0;
01148     for (i=0; i<ncol; i++) {
01149         fprintf( fp, valfmtstr, D[i] );
01150         if ( ( (j+1) % nvalline ) == 0 ) {
01151             fprintf( fp, "\n" );
01152         }
01153         j++;
01154         for (jj=IA[i]; jj<IA[i+1]; jj++) {
01155             fprintf( fp, valfmtstr, L[jj] );
01156             if ( ( (j+1) % nvalline ) == 0 ) {
01157                 fprintf( fp, "\n" );
01158             }
01159             j++;
01160         }
01161     }
01162
01163     if ( ( j % nvalline ) != 0 ) {
01164         fprintf( fp, "\n" );
01165     }
01166
01167     } else { /* ISNOT_SYM */
01168
01169         VASSERT( 0 ); /* NOT CODED YET */
01170     }
01171
01172     /* Step 5: Close the file */
01173     fclose( fp );
01174 }
01175
01176 VPUBLIC PDE* Vfetc_PDE_ctor(Vfetc *fetc) {
01177
01178     PDE *thee = VNULL;
01179
01180     thee = (PDE*)Vmem_malloc(fetc->vmem, 1, sizeof(PDE));
01181     VASSERT(thee != VNULL);
01182     VASSERT(Vfetc_PDE_ctor2(thee, fetc));
01183
01184     return thee;
01185 }
01186
01187 VPUBLIC int Vfetc_PDE_ctor2(PDE *thee,
01188                             Vfetc *fetc
01189                             ) {
01190
01191     int i;
01192
01193     if (thee == VNULL) {
01194         Vnm_print(2, "Vfetc_PDE_ctor2: Got NULL thee!\n");
01195         return 0;
01196     }
01197
01198     /* Store a local copy of the Vfetc class */
01199     var.fetc = fetc;
01200

```

```

01201     /* PDE-specific parameters and function pointers */
01202     thee->initAssemble = Vfetk_PDE_initAssemble;
01203     thee->initElement  = Vfetk_PDE_initElement;
01204     thee->initFace     = Vfetk_PDE_initFace;
01205     thee->initPoint    = Vfetk_PDE_initPoint;
01206     thee->Fu           = Vfetk_PDE_Fu;
01207     thee->Fu_v         = Vfetk_PDE_Fu_v;
01208     thee->DFu_wv       = Vfetk_PDE_DF_uwv;
01209     thee->delta        = Vfetk_PDE_delta;
01210     thee->u_D          = Vfetk_PDE_u_D;
01211     thee->u_T          = Vfetk_PDE_u_T;
01212     thee->Ju           = Vfetk_PDE_Ju;
01213     thee->vec          = 1; /* FIX! */
01214     thee->sym[0][0]    = 1;
01215     thee->est[0]       = 1.0;
01216     for (i=0; i<VMAX_BDTYPE; i++) thee->bmap[0][i] = i;
01217
01218     /* Manifold-specific function pointers */
01219     thee->bisectEdge    = Vfetk_PDE_bisectEdge;
01220     thee->mapBoundary   = Vfetk_PDE_mapBoundary;
01221     thee->markSimplex   = Vfetk_PDE_markSimplex;
01222     thee->oneChart      = Vfetk_PDE_oneChart;
01223
01224     /* Element-specific function pointers */
01225     thee->simplexBasisInit = Vfetk_PDE_simplexBasisInit;
01226     thee->simplexBasisForm = Vfetk_PDE_simplexBasisForm;
01227
01228     return 1;
01229 }
01230
01231 VPUBLIC void Vfetk_PDE_dtor(PDE **thee) {
01232
01233     if ((*thee) != VNULL) {
01234         Vfetk_PDE_dtor2(*thee);
01235         /* TODO: The following line is commented out because at the moment,
01236            there is a seg fault when deallocating at the end of a run. Since
01237            this routine is called only once at the very end, we'll leave it
01238            commented out. However, this could be a memory leak.
01239            */
01240         /* Vmem_free(var.fetk->vmem, 1, sizeof(PDE), (void **)thee); */
01241         (*thee) = VNULL;
01242     }
01243 }
01244
01245
01246 VPUBLIC void Vfetk_PDE_dtor2(PDE *thee) {
01247     var.fetk = VNULL;
01248 }
01249
01250 VPRIVATE double smooth(int nverts, double dist[VAPBS_NVS], double coeff[VAPBS_NVS], int meth) {
01251
01252     int i;
01253     double weight;
01254     double num = 0.0;
01255     double den = 0.0;
01256
01257     for (i=0; i<nverts; i++) {
01258         if (dist[i] < VSMALL) return coeff[i];
01259         weight = 1.0/dist[i];
01260         if (meth == 0) {
01261             num += (weight * coeff[i]);
01262             den += weight;
01263         } else if (meth == 1) {
01264             /* Small coefficients reset the average to 0; we need to break out
01265                * of the loop */
01266             if (coeff[i] < VSMALL) {
01267                 num = 0.0;
01268                 break;
01269             } else {
01270                 num += weight; den += (weight/coeff[i]);
01271             }
01272         } else VASSERT(0);
01273     }
01274
01275     return (num/den);
01276 }
01277
01278
01279 VPRIVATE double diel() {
01280
01281     int i, j;

```

```

01282     double eps, epsp, epsw, dist[5], coeff[5], srاد, swin, *vx;
01283     Vsurf_Meth srاد;
01284     Vacc *acc;
01285     PBEParm *pbeparm;
01286
01287     epsp = Vpbe_getSoluteDiel(var.fetk->pbe);
01288     epsw = Vpbe_getSolventDiel(var.fetk->pbe);
01289     VASSERT(var.fetk->pbeparm != VNULL);
01290     pbeparm = var.fetk->pbeparm;
01291     srاد = pbeparm->srاد;
01292     srاد = pbeparm->srاد;
01293     swin = pbeparm->swin;
01294     acc = var.fetk->pbe->acc;
01295
01296     eps = 0;
01297
01298     if (VABS(epsp - epsw) < VSMALL) return epsp;
01299     switch (srاد) {
01300     case VSM_MOL:
01301         eps = ((epsp-epsp)*Vacc_molAcc(acc, var.xq, srاد) + epsp);
01302         break;
01303     case VSM_MOLSMOOTH:
01304         for (i=0; i<var.nverts; i++) {
01305             dist[i] = 0;
01306             vx = var.vx[i];
01307             for (j=0; j<3; j++) {
01308                 dist[i] += VSQR(var.xq[j] - vx[j]);
01309             }
01310             dist[i] = VSQRT(dist[i]);
01311             coeff[i] = (epsp-epsp)*Vacc_molAcc(acc, var.xq, srاد) + epsp;
01312         }
01313         eps = smooth(var.nverts, dist, coeff, 1);
01314         break;
01315     case VSM_SPLINE:
01316         eps = ((epsp-epsp)*Vacc_splineAcc(acc, var.xq, swin, 0.0) + epsp);
01317         break;
01318     default:
01319         Vnm_print(2, "Undefined surface method (%d)!\n", srاد);
01320         VASSERT(0);
01321     }
01322
01323     return eps;
01324 }
01325
01326 VPRIVATE double ionacc() {
01327
01328     int i, j;
01329     double dist[5], coeff[5], irاد, swin, *vx, accval;
01330     Vsurf_Meth srاد;
01331     Vacc *acc = VNULL;
01332     PBEParm *pbeparm = VNULL;
01333
01334     VASSERT(var.fetk->pbeparm != VNULL);
01335     pbeparm = var.fetk->pbeparm;
01336     srاد = pbeparm->srاد;
01337     irاد = Vpbe_getMaxIonRadius(var.fetk->pbe);
01338     swin = pbeparm->swin;
01339     acc = var.fetk->pbe->acc;
01340
01341     if (var.zks2 < VSMALL) return 0.0;
01342     switch (srاد) {
01343     case VSM_MOL:
01344         accval = Vacc_ivdwAcc(acc, var.xq, irاد);
01345         break;
01346     case VSM_MOLSMOOTH:
01347         for (i=0; i<var.nverts; i++) {
01348             dist[i] = 0;
01349             vx = var.vx[i];
01350             for (j=0; j<3; j++) {
01351                 dist[i] += VSQR(var.xq[j] - vx[j]);
01352             }
01353             dist[i] = VSQRT(dist[i]);
01354             coeff[i] = Vacc_ivdwAcc(acc, var.xq, irاد);
01355         }
01356         accval = smooth(var.nverts, dist, coeff, 1);
01357         break;
01358     case VSM_SPLINE:
01359         accval = Vacc_splineAcc(acc, var.xq, swin, irاد);
01360         break;
01361     default:
01362         Vnm_print(2, "Undefined surface method (%d)!\n", srاد);

```



```

01363         VASSERT(0);
01364     }
01365
01366     return accval;
01367 }
01368
01369 VPRIVATE double debye_U(Vpbe *pbe, int d, double x[]) {
01370
01371     double size, *position, charge, xkappa, eps_w, dist, T, pot, val;
01372     int iatom, i;
01373     Valist *alist;
01374     Vatom *atom;
01375
01376     eps_w = Vpbe_getSolventDiel(pbe);
01377     xkappa = (1.0e10)*Vpbe_getXkappa(pbe);
01378     T = Vpbe_getTemperature(pbe);
01379     alist = Vpbe_getValist(pbe);
01380     val = 0;
01381     pot = 0;
01382
01383     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
01384         atom = Valist_getAtom(alist, iatom);
01385         position = Vatom_getPosition(atom);
01386         charge = Vunit_ec*Vatom_getCharge(atom);
01387         size = (1e-10)*Vatom_getRadius(atom);
01388         dist = 0;
01389         for (i=0; i<d; i++) {
01390             dist += VSQR(position[i] - x[i]);
01391         }
01392         dist = (1.0e-10)*VSQRT(dist);
01393         val = (charge)/(4*VPI*Vunit_eps0*eps_w*dist);
01394         if (xkappa != 0.0) {
01395             val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
01396         }
01397         pot = pot + val;
01398     }
01399     pot = pot*Vunit_ec/(Vunit_kb*T);
01400
01401     return pot;
01402 }
01403
01404 VPRIVATE double debye_Udiff(Vpbe *pbe, int d, double x[]) {
01405
01406     double size, *position, charge, eps_p, dist, T, pot, val;
01407     double Ufull;
01408     int iatom, i;
01409     Valist *alist;
01410     Vatom *atom;
01411
01412     Ufull = debye_U(pbe, d, x);
01413
01414     eps_p = Vpbe_getSoluteDiel(pbe);
01415     T = Vpbe_getTemperature(pbe);
01416     alist = Vpbe_getValist(pbe);
01417     val = 0;
01418     pot = 0;
01419
01420     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
01421         atom = Valist_getAtom(alist, iatom);
01422         position = Vatom_getPosition(atom);
01423         charge = Vunit_ec*Vatom_getCharge(atom);
01424         size = (1e-10)*Vatom_getRadius(atom);
01425         dist = 0;
01426         for (i=0; i<d; i++) {
01427             dist += VSQR(position[i] - x[i]);
01428         }
01429         dist = (1.0e-10)*VSQRT(dist);
01430         val = (charge)/(4*VPI*Vunit_eps0*eps_p*dist);
01431         pot = pot + val;
01432     }
01433     pot = pot*Vunit_ec/(Vunit_kb*T);
01434
01435     pot = Ufull - pot;
01436
01437     return pot;
01438 }
01439
01440 VPRIVATE void coulomb(Vpbe *pbe, int d, double pt[], double eps, double *U,
01441     double dU[], double *d2U) {
01442
01443     int iatom, i;

```

```

01444     double T, pot, fx, fy, fz, x, y, z, scale;
01445     double *position, charge, dist, dist2, val, vec[3], dUold[3], Uold;
01446     Valist *alist;
01447     Vatom *atom;
01448
01449     /* Initialize variables */
01450     T = Vpbe_getTemperature(pbe);
01451     alist = Vpbe_getValist(pbe);
01452     pot = 0; fx = 0; fy = 0; fz = 0;
01453     x = pt[0]; y = pt[1]; z = pt[2];
01454
01455     /* Calculate */
01456     if (!Vgreen_coulombD(var.green, 1, &x, &y, &z, &pot, &fx, &fy, &fz)) {
01457         Vnm_print(2, "Error calculating Green's function!\n");
01458         VASSERT(0);
01459     }
01460
01461     /* Scale the results */
01462     scale = Vunit_ec/(eps*Vunit_kb*T);
01463     *U = pot*scale;
01464     *d2U = 0.0;
01465     dU[0] = -fx*scale;
01466     dU[1] = -fy*scale;
01467     dU[2] = -fz*scale;
01468
01469 #if 0
01470     /* Compare with old results */
01471     val = 0.0;
01472     Uold = 0.0; dUold[0] = 0.0; dUold[1] = 0.0; dUold[2] = 0.0;
01473     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
01474         atom = Valist_getAtom(alist, iatom);
01475         position = Vatom_getPosition(atom);
01476         charge = Vatom_getCharge(atom);
01477         dist2 = 0;
01478         for (i=0; i<d; i++) {
01479             vec[i] = (position[i] - pt[i]);
01480             dist2 += VSQR(vec[i]);
01481         }
01482         dist = VSQRT(dist2);
01483
01484         /* POTENTIAL */
01485         Uold = Uold + charge/dist;
01486
01487         /* GRADIENT */
01488         for (i=0; i<d; i++) dUold[i] = dUold[i] + vec[i]*charge/(dist2*dist);
01489     }
01490     Uold = Uold*VSQR(Vunit_ec)*(1.0e10)/(4*VPI*Vunit_eps0*eps*
01491     Vunit_kb*T);
01492     for (i=0; i<d; i++) {
01493         dUold[i] = dUold[i]*VSQR(Vunit_ec)*(1.0e10)/(4*VPI*Vunit_eps0*eps*
01494         Vunit_kb*T);
01495     }
01496
01497     printf("Unew - Uold = %g - %g = %g\n", *U, Uold, (*U - Uold));
01498     printf("||dUnew - dUold||^2 = %g\n", (VSQR(dU[0] - dUold[0])
01499     + VSQR(dU[1] - dUold[1]) + VSQR(dU[2] - dUold[2])));
01500     printf("dUnew[0] = %g, dUold[0] = %g\n", dU[0], dUold[0]);
01501     printf("dUnew[1] = %g, dUold[1] = %g\n", dU[1], dUold[1]);
01502     printf("dUnew[2] = %g, dUold[2] = %g\n", dU[2], dUold[2]);
01503
01504 #endif
01505 }
01506
01507
01508 VPUBLIC void Vfetk_PDE_initAssemble(PDE *thee, int ip[], double rp[]) {
01509
01510 #if 1
01511     /* Re-initialize the Green's function oracle in case the atom list has
01512     * changed */
01513     if (var.initGreen) {
01514         Vgreen_dtor(&(var.green));
01515         var.initGreen = 0;
01516     }
01517     var.green = Vgreen_ctor(var.fetk->pbe->alist);
01518     var.initGreen = 1;
01519 #else
01520     if (!var.initGreen) {
01521         var.green = Vgreen_ctor(var.fetk->pbe->alist);
01522         var.initGreen = 1;
01523     }
01524 #endif

```

```

01523     }
01524 #endif
01525
01526 }
01527
01528 VPUBLIC void Vfetk_PDE_initElement(PDE *thee, int elementType, int chart,
01529     double tvx[][3], void *data) {
01530
01531     int i, j;
01532     double epsp, epsw;
01533
01534     /* We assume that the simplex has been passed in as the void *data *
01535      * argument. Store it */
01536     VASSERT(data != NULL);
01537     var.simp = (SS *)data;
01538
01539     /* save the element type */
01540     var.sType = elementType;
01541
01542     /* Grab the vertices from this simplex */
01543     var.nverts = thee->dim+1;
01544     for (i=0; i<thee->dim+1; i++) var.verts[i] = SS_vertex(var.simp, i);
01545
01546     /* Vertex locations of this simplex */
01547     for (i=0; i<thee->dim+1; i++) {
01548         for (j=0; j<thee->dim; j++) {
01549             var.vx[i][j] = tvx[i][j];
01550         }
01551     }
01552
01553     /* Set the dielectric constant for this element for use in the jump term *
01554      * of the residual-based error estimator. The value is set to the average
01555      * * value of the vertices */
01556     var.jumpDiel = 0; /* NOT IMPLEMENTED YET! */
01557 }
01558
01559 VPUBLIC void Vfetk_PDE_initFace(PDE *thee, int faceType, int chart,
01560     double tnvec[]) {
01561
01562     int i;
01563
01564     /* unit normal vector of this face */
01565     for (i=0; i<thee->dim; i++) var.nvec[i] = tnvec[i];
01566
01567     /* save the face type */
01568     var.fType = faceType;
01569 }
01570
01571 VPUBLIC void Vfetk_PDE_initPoint(PDE *thee, int pointType, int chart,
01572     double txq[], double tU[], double tdU[][3]) {
01573
01574     int i, j, ichop;
01575     double u2, coef2, eps_p;
01576     Vhal_PBEType pdetype;
01577     Vpbe *pbe = VNULL;
01578
01579     eps_p = Vpbe_getSoluteDiel(var.fetk->pbe);
01580     pdetype = var.fetk->type;
01581     pbe = var.fetk->pbe;
01582
01583     /* the point, the solution value and gradient, and the Coulomb value and *
01584      * gradient at the point */
01585     if ((pdetype == PBE_LRPBE) || (pdetype == PBE_NRPBE)) {
01586         coulomb(pbe, thee->dim, txq, eps_p, &(var.W), var.dW, &(var.d2W));
01587     }
01588     for (i=0; i<thee->vec; i++) {
01589         var.U[i] = tU[i];
01590         for (j=0; j<thee->dim; j++) {
01591             var.xq[j] = txq[j];
01592             var.dU[i][j] = tdU[i][j];
01593         }
01594     }
01595
01596     /* interior form case */
01597     if (pointType == 0) {
01598
01599         /* Get the dielectric values */
01600         var.diel = diel();
01601         var.ionacc = ionacc();
01602         var.A = var.diel;
01603         var.F = (var.diel - eps_p);

```

```

01604
01605     switch (pdetype) {
01606
01607         case PBE_LPBE:
01608             var.DB = var.ionacc*var.zkappa2*var.ionstr;
01609             var.B = var.DB*var.U[0];
01610             break;
01611
01612         case PBE_NPBE:
01613
01614             var.B = 0;
01615             var.DB = 0;
01616             if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01617                 for (i=0; i<var.nion; i++) {
01618                     u2 = -1.0 * var.U[0] * var.ionQ[i];
01619
01620                     /* NONLINEAR TERM */
01621                     coef2 = -1.0 * var.ionacc * var.zks2 * var.ionConc[i];
01622                     var.B += (coef2 * Vcap_exp(u2, &ichop));
01623                     /* LINEARIZED TERM */
01624                     coef2 = -1.0 * var.ionQ[i] * coef2;
01625                     var.DB += (coef2 * Vcap_exp(u2, &ichop));
01626                 }
01627             }
01628             break;
01629
01630         case PBE_LRPBE:
01631             var.DB = var.ionacc*var.zkappa2*var.ionstr;
01632             var.B = var.DB*(var.U[0]+var.W);
01633             break;
01634
01635         case PBE_NRPBE:
01636
01637             var.B = 0;
01638             var.DB = 0;
01639             if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01640                 for (i=0; i<var.nion; i++) {
01641                     u2 = -1.0 * (var.U[0] + var.W) * var.ionQ[i];
01642
01643                     /* NONLINEAR TERM */
01644                     coef2 = -1.0 * var.ionacc * var.zks2 * var.ionConc[i];
01645                     var.B += (coef2 * Vcap_exp(u2, &ichop));
01646
01647                     /* LINEARIZED TERM */
01648                     coef2 = -1.0 * var.ionQ[i] * coef2;
01649                     var.DB += (coef2 * Vcap_exp(u2, &ichop));
01650                 }
01651             }
01652             break;
01653
01654         case PBE_SMPBE: /* SMPBE Temp */
01655
01656             var.B = 0;
01657             var.DB = 0;
01658             if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01659                 for (i=0; i<var.nion; i++) {
01660                     u2 = -1.0 * var.U[0] * var.ionQ[i];
01661
01662                     /* NONLINEAR TERM */
01663                     coef2 = -1.0 * var.ionacc * var.zks2 * var.ionConc[i];
01664                     var.B += (coef2 * Vcap_exp(u2, &ichop));
01665                     /* LINEARIZED TERM */
01666                     coef2 = -1.0 * var.ionQ[i] * coef2;
01667                     var.DB += (coef2 * Vcap_exp(u2, &ichop));
01668                 }
01669             }
01670             break;
01671         default:
01672             Vnm_print(2, "Vfetc_PDE_initPoint: Unknown PBE type (%d)!\n",
01673                 pdetype);
01674             VASSERT(0);
01675             break;
01676     }
01677
01678     /* boundary form case */
01679 } else {
01680 #ifdef DONEUMANN
01681     ;
01682 #else
01683     Vnm_print(2, "Vfetc: Whoa! I just got a boundary point to evaluate (%d)!\n", pointType);
01684 
```

```

01685         Vnm_print(2, "Vfetk: Did you do that on purpose?\n");
01686 #endif
01687     }
01688
01689     #if 0 /* THIS IS VERY NOISY! */
01690         Vfetk_dumpLocalVar();
01691 #endif
01692
01693 }
01694
01695 VPUBLIC void Vfetk_PDE_Fu(PDE *thee, int key, double F[]) {
01696
01697     //Vnm_print(2, "Vfetk_PDE_Fu: Setting error to zero!\n");
01698
01699     F[0] = 0.;
01700
01701 }
01702
01703 VPUBLIC double Vfetk_PDE_Fu_v(
01704     PDE *thee,
01705     int key,
01706     double V[],
01707     double dV[][VAPBS_DIM]
01708 ) {
01709
01710     Vhal_PBEType type;
01711     int i;
01712     double value = 0.;
01713
01714     type = var.fetk->type;
01715
01716     /* interior form case */
01717     if (key == 0) {
01718
01719         for (i=0; i<thee->dim; i++) value += ( var.A * var.dU[0][i] * dV[0][i] );
01720         value += var.B * V[0];
01721
01722         if ((type == PBE_LRPBE) || (type == PBE_NRPBE)) {
01723             for (i=0; i<thee->dim; i++) {
01724                 if (var.F > VSMALL) value += (var.F * var.dW[i] * dV[0][i]);
01725             }
01726         }
01727
01728         /* boundary form case */
01729     } else {
01730 #ifdef DONEUMANN
01731         value = 0.0;
01732 #else
01733         Vnm_print(2, "Vfetk: Whoa! I was just asked to evaluate a boundary weak form for point type %d!\n"
01734             , key);
01735 #endif
01736     }
01737
01738     var.Fu_v = value;
01739     return value;
01740 }
01741
01742 VPUBLIC double Vfetk_PDE_DFu_wv(
01743     PDE *thee,
01744     int key,
01745     double W[],
01746     double dW[][VAPBS_DIM],
01747     double V[],
01748     double dV[][3]
01749 ) {
01750
01751     Vhal_PBEType type;
01752     int i;
01753     double value = 0.;
01754
01755     type = var.fetk->type;
01756
01757     /* Interior form */
01758     if (key == 0) {
01759         value += var.DB * W[0] * V[0];
01760         for (i=0; i<thee->dim; i++) value += ( var.A * dW[0][i] * dV[0][i] );
01761
01762         /* boundary form case */
01763     } else {
01764 #ifdef DONEUMANN
01765         value = 0.0;
01766 #endif
01767     }
01768 }

```

```

01765 #else
01766     Vnm_print(2, "Vfetk:  Whoa! I was just asked to evaluate a boundary weak form for point type %d!\n"
, key);
01767 #endif
01768 }
01769
01770     var.DFu_wv = value;
01771     return value;
01772 }
01773
01776 #define VRINGMAX 1000
01777
01779 #define VATOMMAX 1000000
01780 VPUBLIC void Vfetk_PDE_delta(PDE *thee, int type, int chart, double txq[],
01781     void *user, double F[]) {
01782
01783     int iatom, jatom, natoms, atomIndex, atomList[VATOMMAX], nAtomList;
01784     int gotAtom, numSring, isimp, iver, sid;
01785     double *position, charge, phi[VAPBS_NVS], phix[VAPBS_NVS][3], value;
01786     Vatom *atom;
01787     Vhal_PBEType pdetype;
01788     SS *sring[VRINGMAX];
01789     VV *vertex = (VV *)user;
01790
01791     pdetype = var.fetk->type;
01792
01793     F[0] = 0.0;
01794
01795     if ((pdetype == PBE_LPBE) || (pdetype == PBE_NPBE) || (pdetype ==
PBE_SMPBE) /* SMPBE Added */) {
01796         VASSERT( vertex != VNULL);
01797         numSring = 0;
01798         sring[numSring] = VV_firstSS(vertex);
01799         while (sring[numSring] != VNULL) {
01800             numSring++;
01801             sring[numSring] = SS_link(sring[numSring-1], vertex);
01802         }
01803         VASSERT( numSring > 0 );
01804         VASSERT( numSring <= VRINGMAX );
01805
01806         /* Move around the simplex ring and determine the charge locations */
01807         F[0] = 0.;
01808         charge = 0.;
01809         nAtomList = 0;
01810         for (isimp=0; isimp<numSring; isimp++) {
01811             sid = SS_id(sring[isimp]);
01812             natoms = Vcsm_getNumberAtoms(Vfetk_getVcsm(var.
fetk), sid);
01813             for (iatom=0; iatom<natoms; iatom++) {
01814                 /* Get the delta function information */
01815                 atomIndex = Vcsm_getAtomIndex(Vfetk_getVcsm(var.
fetk),
iatom, sid);
01816                 gotAtom = 0;
01817                 for (jatom=0; jatom<nAtomList; jatom++) {
01818                     if (atomList[jatom] == atomIndex) {
01819                         gotAtom = 1;
01820                         break;
01821                     }
01822                 }
01823                 if (!gotAtom) {
01824                     VASSERT(nAtomList < VATOMMAX);
01825                     atomList[nAtomList] = atomIndex;
01826                     nAtomList++;
01827                 }
01828                 atom = Vcsm_getAtom(Vfetk_getVcsm(var.
fetk), iatom, sid);
01829                 charge = Vatom_getCharge(atom);
01830                 position = Vatom_getPosition(atom);
01831
01832                 /* Get the test function value at the delta function I
* used to do a VASSERT to make sure the point was in the
* simplex (i.e., make sure round-off error isn't an
* issue), but round off errors became an issue */
01833                 if (!Gem_pointInSimplexVal(Vfetk_getGem(var.fetk),
sring[isimp], position, phi, phix)) {
01834                     if (!Gem_pointInSimplex(Vfetk_getGem(var.
fetk),
sring[isimp], position)) {
01840                         Vnm_print(2, "delta: Both Gem_pointInSimplexVal \
01841 and Gem_pointInSimplex detected misplaced point charge!\n");

```

```

01843             Vnm_print(2, "delta: I think you have problems: \
01844 phi = {");
01845             for (ivert=0; ivert<Gem_dimVV(Vfetk_getGem(var.
01846 fetk)); ivert++) Vnm_print(2, "%e ", phi[ivert]);
01847         Vnm_print(2, "}\n");
01848     }
01849     value = 0;
01850     for (ivert=0; ivert<Gem_dimVV(Vfetk_getGem(var.
01851 fetk)); ivert++) {
01852         if (VV_id(SS_vertex(sring[isimp], ivert)) == VV_id(vertex)) value += phi[ivert];
01853     }
01854     F[0] += (value * Vpbe_getZmagic(var.fetk->
01855 pbe) * charge);
01856     } /* if !gotAtom */
01857     } /* for iatom */
01858     } /* for isimp */
01859     } else if ((pdetype == PBE_LRPBE) || (pdetype == PBE_NRPBE)) {
01860         F[0] = 0.0;
01861     } else { VASSERT(0); }
01862     var.delta = F[0];
01863 }
01864
01865 }
01866
01867 VPUBLIC void Vfetk_PDE_u_D(PDE *thee, int type, int chart, double txq[],
01868 double F[]) {
01869     if ((var.fetk->type == PBE_LPBE) || (var.fetk->type ==
01870 PBE_NPBE) || (var.fetk->type == PBE_SMPBE) /* SMPBE Added */) {
01871         F[0] = debye_U(var.fetk->pbe, thee->dim, txq);
01872     } else if ((var.fetk->type == PBE_LRPBE) || (var.fetk->
01873 type == PBE_NRPBE)) {
01874         F[0] = debye_Udiff(var.fetk->pbe, thee->dim, txq);
01875     } else VASSERT(0);
01876     var.u_D = F[0];
01877 }
01878 }
01879
01880 VPUBLIC void Vfetk_PDE_u_T(PDE *thee, int type, int chart, double txq[],
01881 double F[]) {
01882     /*VPUBLIC void Vfetk_PDE_u_T(sPDE *thee,
01883 int type,
01884 int chart,
01885 double txq[],
01886 double F[],
01887 double dF[][3]
01888 ) { */
01889     F[0] = 0.0;
01890     var.u_T = F[0];
01891 }
01892 }
01893
01894 VPUBLIC void Vfetk_PDE_bisectEdge(int dim, int dimII, int edgeType,
01895 int chart[], double vx[][3]) {
01896     int i;
01897     for (i=0; i<dimII; i++) vx[2][i] = .5 * (vx[0][i] + vx[1][i]);
01898     chart[2] = chart[0];
01899 }
01900 }
01901
01902 VPUBLIC void Vfetk_PDE_mapBoundary(int dim, int dimII, int vertexType,
01903 int chart, double vx[3]) {
01904 }
01905 }
01906
01907 VPUBLIC int Vfetk_PDE_markSimplex(int dim, int dimII, int simplexType,
01908 int faceType[VAPBS_NVS], int vertexType[VAPBS_NVS], int chart[], double vx[][3],
01909 void *simplex) {
01910     double targetRes, edgeLength, srad, swin, myAcc, refAcc;
01911     int i, natoms;
01912     Vsurf_Meth srfm;

```

```

01924     Vhal_PBEType type;
01925     FEMparm *feparm = VNULL;
01926     PBEParm *pbeparm = VNULL;
01927     Vpbe *pbe = VNULL;
01928     Vacc *acc = VNULL;
01929     Vcsm *csm = VNULL;
01930     SS *simp = VNULL;
01931
01932     VASSERT(var.fetk->feparm != VNULL);
01933     feparm = var.fetk->feparm;
01934     VASSERT(var.fetk->pbeparm != VNULL);
01935     pbeparm = var.fetk->pbeparm;
01936     pbe = var.fetk->pbe;
01937     csm = Vfetk_getVcsm(var.fetk);
01938     acc = pbe->acc;
01939     targetRes = feparm->targetRes;
01940     srfrm = pbeparm->srfrm;
01941     srad = pbeparm->srad;
01942     swin = pbeparm->swin;
01943     simp = (SS *)simplex;
01944     type = var.fetk->type;
01945
01946     /* Check to see if this simplex is smaller than the target size */
01947     /* NAB WARNING: I am providing face=-1 here to conform to the new MC API; however, I'm not sure if
this is the correct behavior. */
01948     Gem_longestEdge(var.fetk->gm, simp, -1, &edgeLength);
01949     if (edgeLength < targetRes) return 0;
01950
01951     /* For non-regularized PBE, check charge-simplex map */
01952     if ((type == PBE_LPBE) || (type == PBE_NPBE) || (type ==
PBE_SMPBE) /* SMPBE Added */) {
01953         natoms = Vcsm_getNumberAtoms(csm, SS_id(simp));
01954         if (natoms > 0) {
01955             return 1;
01956         }
01957     }
01958
01959     /* We would like to resolve the mesh between the van der Waals surface the
* max distance from this surface where there could be coefficient
* changes */
01960     switch(srfrm) {
01961     case VSM_MOL:
01962         refAcc = Vacc_molAcc(acc, vx[0], srad);
01963         for (i=1; i<(dim+1); i++) {
01964             myAcc = Vacc_molAcc(acc, vx[i], srad);
01965             if (myAcc != refAcc) {
01966                 return 1;
01967             }
01968         }
01969         break;
01970     case VSM_MOLSMOOTH:
01971         refAcc = Vacc_molAcc(acc, vx[0], srad);
01972         for (i=1; i<(dim+1); i++) {
01973             myAcc = Vacc_molAcc(acc, vx[i], srad);
01974             if (myAcc != refAcc) {
01975                 return 1;
01976             }
01977         }
01978         break;
01979     case VSM_SPLINE:
01980         refAcc = Vacc_splineAcc(acc, vx[0], swin, 0.0);
01981         for (i=1; i<(dim+1); i++) {
01982             myAcc = Vacc_splineAcc(acc, vx[i], swin, 0.0);
01983             if (myAcc != refAcc) {
01984                 return 1;
01985             }
01986         }
01987         break;
01988     default:
01989         VASSERT(0);
01990         break;
01991     }
01992     return 0;
01993 }
01994
01995 VPUBLIC void Vfetk_PDE_oneChart(int dim, int dimII, int objType, int chart[],
double vx[][3], int dimV) {
02000
02001 }
02002

```



```

02003 VPUBLIC double Vfetk_PDE_Ju(PDE *three, int key) {
02004
02005     int i, ichop;
02006     double dielE, qmE, coef2, u2;
02007     double value = 0.;
02008     Vhal_PBEType type;
02009
02010     type = var.fetk->type;
02011
02012     /* interior form case */
02013     if (key == 0) {
02014         dielE = 0;
02015         for (i=0; i<3; i++) dielE += VSQR(var.dU[0][i]);
02016         dielE = dielE*var.diel;
02017
02018         switch (type) {
02019             case PBE_LPBE:
02020                 if ((var.ionacc > VSMALL) && (var.zkappa2 > VSMALL)) {
02021                     qmE = var.ionacc*var.zkappa2*VSQR(var.U[0]);
02022                 } else qmE = 0;
02023                 break;
02024             case PBE_NPBE:
02025                 if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
02026                     qmE = 0.;
02027                     for (i=0; i<var.nion; i++) {
02028                         coef2 = var.ionacc * var.zks2 * var.ionConc[i] * var.
ionQ[i];
02029                         u2 = -1.0 * (var.U[0]) * var.ionQ[i];
02030                         qmE += (coef2 * (Vcap_exp(u2, &ichop) - 1.0));
02031                     }
02032                 } else qmE = 0;
02033                 break;
02034             case PBE_LRPBE:
02035                 if ((var.ionacc > VSMALL) && (var.zkappa2 > VSMALL)) {
02036                     qmE = var.ionacc*var.zkappa2*VSQR((var.U[0] + var.W));
02037                 } else qmE = 0;
02038                 break;
02039             case PBE_NRPBE:
02040                 if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
02041                     qmE = 0.;
02042                     for (i=0; i<var.nion; i++) {
02043                         coef2 = var.ionacc * var.zks2 * var.ionConc[i] * var.
ionQ[i];
02044                         u2 = -1.0 * (var.U[0] + var.W) * var.ionQ[i];
02045                         qmE += (coef2 * (Vcap_exp(u2, &ichop) - 1.0));
02046                     }
02047                 } else qmE = 0;
02048                 break;
02049             case PBE_SMPBE: /* SMPBE Temp */
02050                 if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
02051                     qmE = 0.;
02052                     for (i=0; i<var.nion; i++) {
02053                         coef2 = var.ionacc * var.zks2 * var.ionConc[i] * var.
ionQ[i];
02054                         u2 = -1.0 * (var.U[0]) * var.ionQ[i];
02055                         qmE += (coef2 * (Vcap_exp(u2, &ichop) - 1.0));
02056                     }
02057                 } else qmE = 0;
02058                 break;
02059             default:
02060                 Vnm_print(2, "Vfetk_PDE_Ju: Invalid PBE type (%d)!\n", type);
02061                 VASSERT(0);
02062                 break;
02063         }
02064
02065         value = 0.5*(dielE + qmE)/Vpbe_getZmagic(var.fetk->pbe);
02066
02067     /* boundary form case */
02068     } else if (key == 1) {
02069         value = 0.0;
02070
02071     /* how did we get here? */
02072     } else VASSERT(0);
02073
02074     return value;
02075 }
02076
02077
02078 VPUBLIC void Vfetk_externalUpdateFunction(SS **simps, int num) {
02079
02080     Vcsm *csm = VNULL;

```

```

02081     int rc;
02082
02083     VASSERT(var.fetk != VNULL);
02084     csm = Vfetk_getVcsm(var.fetk);
02085     VASSERT(csm != VNULL);
02086
02087     rc = Vcsm_update(csm, simps, num);
02088
02089     if (!rc) {
02090         Vnm_print(2, "Error while updating charge-simplex map!\n");
02091         VASSERT(0);
02092     }
02093 }
02094
02095 VPRIVATE void polyEval(int numP, double p[], double c[][VMAXP], double xv[]) {
02096     int i;
02097     double x, y, z;
02098
02099     x = xv[0];
02100     y = xv[1];
02101     z = xv[2];
02102     for (i=0; i<numP; i++) {
02103         p[i] = c[i][0]
02104             + c[i][1] * x
02105             + c[i][2] * y
02106             + c[i][3] * z
02107             + c[i][4] * x*x
02108             + c[i][5] * y*y
02109             + c[i][6] * z*z
02110             + c[i][7] * x*y
02111             + c[i][8] * x*z
02112             + c[i][9] * y*z
02113             + c[i][10] * x*x*x
02114             + c[i][11] * y*y*y
02115             + c[i][12] * z*z*z
02116             + c[i][13] * x*x*y
02117             + c[i][14] * x*x*z
02118             + c[i][15] * x*y*y
02119             + c[i][16] * y*y*z
02120             + c[i][17] * x*z*z
02121             + c[i][18] * y*z*z;
02122     }
02123 }
02124
02125 VPRIVATE void setCoef(int numP, double c[][VMAXP], double cx[][VMAXP],
02126     double cy[][VMAXP], double cz[][VMAXP], int ic[][VMAXP], int icx[][VMAXP],
02127     int icy[][VMAXP], int icz[][VMAXP]) {
02128
02129     int i, j;
02130     for (i=0; i<numP; i++) {
02131         for (j=0; j<VMAXP; j++) {
02132             c[i][j] = 0.5 * (double)ic[i][j];
02133             cx[i][j] = 0.5 * (double)icx[i][j];
02134             cy[i][j] = 0.5 * (double)icy[i][j];
02135             cz[i][j] = 0.5 * (double)icz[i][j];
02136         }
02137     }
02138 }
02139
02140 VPUBLIC int Vfetk_PDE_simplexBasisInit(int key, int dim, int comp, int *ndof,
02141     int dof[]) {
02142
02143     int qorder, bump, dimIS[VAPBS_NVS];
02144
02145     /* necessary quadrature order to return at the end */
02146     qorder = P_DEG;
02147
02148     /* deal with bump function requests */
02149     if ((key == 0) || (key == 1)) {
02150         bump = 0;
02151     } else if ((key == 2) || (key == 3)) {
02152         bump = 1;
02153     } else { VASSERT(0); }
02154
02155     /* for now use same element for all components, both trial and test */
02156     if (dim==2) {
02157         /* 2D simplex dimensions */
02158         dimIS[0] = 3; /* number of vertices */
02159         dimIS[1] = 3; /* number of edges */
02160         dimIS[2] = 0; /* number of faces (3D only) */
02161         dimIS[3] = 1; /* number of simplices (always=1) */

```

```

02162         if (bump==0) {
02163             if (P_DEG==1) {
02164                 init_2DPl(dimIS, ndof, dof, c, cx, cy, cz);
02165             } else if (P_DEG==2) {
02166                 init_2DPl(dimIS, ndof, dof, c, cx, cy, cz);
02167             } else if (P_DEG==3) {
02168                 init_2DPl(dimIS, ndof, dof, c, cx, cy, cz);
02169             } else Vnm_print(2, "..bad order..");
02170         } else if (bump==1) {
02171             if (P_DEG==1) {
02172                 init_2DPl(dimIS, ndof, dof, c, cx, cy, cz);
02173             } else Vnm_print(2, "..bad order..");
02174         } else Vnm_print(2, "..bad bump..");
02175     } else if (dim==3) {
02176         /* 3D simplex dimensions */
02177         dimIS[0] = 4; /* number of vertices */
02178         dimIS[1] = 6; /* number of edges */
02179         dimIS[2] = 4; /* number of faces (3D only) */
02180         dimIS[3] = 1; /* number of simplices (always=1) */
02181         if (bump==0) {
02182             if (P_DEG==1) {
02183                 init_3DPl(dimIS, ndof, dof, c, cx, cy, cz);
02184             } else if (P_DEG==2) {
02185                 init_3DPl(dimIS, ndof, dof, c, cx, cy, cz);
02186             } else if (P_DEG==3) {
02187                 init_3DPl(dimIS, ndof, dof, c, cx, cy, cz);
02188             } else Vnm_print(2, "..bad order..");
02189         } else if (bump==1) {
02190             if (P_DEG==1) {
02191                 init_3DPl(dimIS, ndof, dof, c, cx, cy, cz);
02192             } else Vnm_print(2, "..bad order..");
02193         } else Vnm_print(2, "..bad bump..");
02194     } else Vnm_print(2, "..bad dimension..");
02195     /* save number of DF */
02196     numP = *ndof;
02197
02198     /* return the required quarature order */
02199     return qorder;
02200 }
02201
02202
02203 VPUBLIC void Vfetk_PDE_simplexBasisForm(int key, int dim, int comp, int pdkey,
02204     double xq[], double basis[]) {
02205
02206     if (pdkey == 0) {
02207         polyEval(numP, basis, c, xq);
02208     } else if (pdkey == 1) {
02209         polyEval(numP, basis, cx, xq);
02210     } else if (pdkey == 2) {
02211         polyEval(numP, basis, cy, xq);
02212     } else if (pdkey == 3) {
02213         polyEval(numP, basis, cz, xq);
02214     } else { VASSERT(0); }
02215 }
02216
02217 VPRIVATE void init_2DPl(int dimIS[], int *ndof, int dof[], double c[][VMAXP],
02218     double cx[][VMAXP], double cy[][VMAXP], double cz[][VMAXP]) {
02219
02220     int i;
02221
02222     /* dof number and locations */
02223     dof[0] = 1;
02224     dof[1] = 0;
02225     dof[2] = 0;
02226     dof[3] = 0;
02227     *ndof = 0;
02228     for (i=0; i<VAPBS_NV; i++) *ndof += dimIS[i] * dof[i];
02229     VASSERT( *ndof == dim_2DPl );
02230     VASSERT( *ndof <= VMAXP );
02231
02232     /* coefficients of the polynomials */
02233     setCoef( *ndof, c, cx, cy, cz, lgr_2DPl, lgr_2DPlx, lgr_2DPlz );
02234 }
02235
02236 VPRIVATE void init_3DPl(int dimIS[], int *ndof, int dof[], double c[][VMAXP],
02237     double cx[][VMAXP], double cy[][VMAXP], double cz[][VMAXP]) {
02238
02239     int i;
02240
02241     /* dof number and locations */
02242     dof[0] = 1;

```

```

02243     dof[1] = 0;
02244     dof[2] = 0;
02245     dof[3] = 0;
02246     *ndof = 0;
02247     for (i=0; i<VAPBS_NVNS; i++) *ndof += dimIS[i] * dof[i];
02248     VASSERT( *ndof == dim_3DP1 );
02249     VASSERT( *ndof <= VMAXP );
02250
02251     /* coefficients of the polynomials */
02252     setCoef( *ndof, c, cx, cy, cz, lgr_3DP1, lgr_3DP1x, lgr_3DP1y, lgr_3DP1z );
02253 }
02254
02255 VPUBLIC void Vfetc_dumpLocalVar() {
02256
02257     int i;
02258
02259     Vnm_print(1, "DEBUG: nvec = (%g, %g, %g)\n", var.nvec[0], var.nvec[1],
02260               var.nvec[2]);
02261     Vnm_print(1, "DEBUG: nverts = %d\n", var.nverts);
02262     for (i=0; i<var.nverts; i++) {
02263         Vnm_print(1, "DEBUG: verts[%d] ID = %d\n", i, VV_id(var.verts[i]));
02264         Vnm_print(1, "DEBUG: vx[%d] = (%g, %g, %g)\n", i, var.vx[i][0],
02265               var.vx[i][1], var.vx[i][2]);
02266     }
02267     Vnm_print(1, "DEBUG: simp ID = %d\n", SS_id(var.simp));
02268     Vnm_print(1, "DEBUG: sType = %d\n", var.sType);
02269     Vnm_print(1, "DEBUG: fType = %d\n", var.fType);
02270     Vnm_print(1, "DEBUG: xq = (%g, %g, %g)\n", var.xq[0], var.xq[1], var.xq[2]);
02271     Vnm_print(1, "DEBUG: U[0] = %g\n", var.U[0]);
02272     Vnm_print(1, "DEBUG: dU[0] = (%g, %g, %g)\n", var.dU[0][0], var.dU[0][1],
02273               var.dU[0][2]);
02274     Vnm_print(1, "DEBUG: W = %g\n", var.W);
02275     Vnm_print(1, "DEBUG: d2W = %g\n", var.d2W);
02276     Vnm_print(1, "DEBUG: dW = (%g, %g, %g)\n", var.dW[0], var.dW[1], var.dW[2]);
02277     Vnm_print(1, "DEBUG: diel = %g\n", var.diel);
02278     Vnm_print(1, "DEBUG: ionacc = %g\n", var.ionacc);
02279     Vnm_print(1, "DEBUG: A = %g\n", var.A);
02280     Vnm_print(1, "DEBUG: F = %g\n", var.F);
02281     Vnm_print(1, "DEBUG: B = %g\n", var.B);
02282     Vnm_print(1, "DEBUG: DB = %g\n", var.DB);
02283     Vnm_print(1, "DEBUG: nion = %d\n", var.nion);
02284     for (i=0; i<var.nion; i++) {
02285         Vnm_print(1, "DEBUG: ionConc[%d] = %g\n", i, var.ionConc[i]);
02286         Vnm_print(1, "DEBUG: ionQ[%d] = %g\n", i, var.ionQ[i]);
02287         Vnm_print(1, "DEBUG: ionRadii[%d] = %g\n", i, var.ionRadii[i]);
02288     }
02289     Vnm_print(1, "DEBUG: zkappa2 = %g\n", var.zkappa2);
02290     Vnm_print(1, "DEBUG: zks2 = %g\n", var.zks2);
02291     Vnm_print(1, "DEBUG: Fu_v = %g\n", var.Fu_v);
02292     Vnm_print(1, "DEBUG: DFu_wv = %g\n", var.DFu_wv);
02293     Vnm_print(1, "DEBUG: delta = %g\n", var.delta);
02294     Vnm_print(1, "DEBUG: u_D = %g\n", var.u_D);
02295     Vnm_print(1, "DEBUG: u_T = %g\n", var.u_T);
02296
02297 };
02298
02299 VPUBLIC int Vfetc_fillArray(Vfetc *thee, Bvec *vec,
02300                           Vdata_Type type) {
02301
02302     int i, j, ichop;
02303     double coord[3], chi, q, conc, val;
02304     VV *vert;
02305     Bvec *u, *u_d;
02306     AM *am;
02307     Gem *gm;
02308     PBEparm *pbeparm;
02309     Vacc *acc;
02310     Vpbe *pbe;
02311
02312     gm = thee->gm;
02313     am = thee->am;
02314     pbe = thee->pbe;
02315     pbeparm = thee->pbeparm;
02316     acc = pbe->acc;
02317
02318     /* Make sure vec has enough rows to accomodate the vertex data */
02319     if (Bvec_numRB(vec, 0) != Gem_numVV(gm)) {
02320         Vnm_print(2, "Vfetc_fillArray: insufficient space in Bvec!\n");
02321         Vnm_print(2, "Vfetc_fillArray: Have %d, need %d!\n", Bvec_numRB(vec, 0),
02322               Gem_numVV(gm));
02323         return 0;
02324     }

```

```

02323     }
02324
02325     switch (type) {
02326
02327         case VDT_CHARGE:
02328             Vnm_print(2, "Vfetk_fillArray: can't write out charge distribution!\n");
02329             return 0;
02330             break;
02331
02332         case VDT_POT:
02333             u = am->u;
02334             u_d = am->ud;
02335             /* Copy in solution */
02336             Bvec_copy(vec, u);
02337             /* Add dirichlet condition */
02338             Bvec_axpy(vec, u_d, 1.0);
02339             break;
02340
02341         case VDT_SMOL:
02342             for (i=0; i<Gem_numVV(gm); i++) {
02343                 vert = Gem_VV(gm, i);
02344                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02345                 chi = Vacc_molAcc(acc, coord, pbe->solventRadius);
02346                 Bvec_set(vec, i, chi);
02347             }
02348             break;
02349
02350         case VDT_SSPL:
02351             for (i=0; i<Gem_numVV(gm); i++) {
02352                 vert = Gem_VV(gm, i);
02353                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02354                 chi = Vacc_splineAcc(acc, coord, pbeparm->swin, 0.0);
02355                 Bvec_set(vec, i, chi);
02356             }
02357             break;
02358
02359         case VDT_VDW:
02360             for (i=0; i<Gem_numVV(gm); i++) {
02361                 vert = Gem_VV(gm, i);
02362                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02363                 chi = Vacc_vdwAcc(acc, coord);
02364                 Bvec_set(vec, i, chi);
02365             }
02366             break;
02367
02368         case VDT_IVDW:
02369             for (i=0; i<Gem_numVV(gm); i++) {
02370                 vert = Gem_VV(gm, i);
02371                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02372                 chi = Vacc_ivdwAcc(acc, coord, pbe->maxIonRadius);
02373                 Bvec_set(vec, i, chi);
02374             }
02375             break;
02376
02377         case VDT_LAP:
02378             Vnm_print(2, "Vfetk_fillArray: can't write out Laplacian!\n");
02379             return 0;
02380             break;
02381
02382         case VDT_EDENS:
02383             Vnm_print(2, "Vfetk_fillArray: can't write out energy density!\n");
02384             return 0;
02385             break;
02386
02387         case VDT_NDENS:
02388             u = am->u;
02389             u_d = am->ud;
02390             /* Copy in solution */
02391             Bvec_copy(vec, u);
02392             /* Add dirichlet condition */
02393             Bvec_axpy(vec, u_d, 1.0);
02394             /* Load up ions */
02395             ichop = 0;
02396             for (i=0; i<Gem_numVV(gm); i++) {
02397                 val = 0;
02398                 for (j=0; j<pbe->numIon; j++) {
02399                     q = pbe->ionQ[j];
02400                     conc = pbe->ionConc[j];
02401                     if (three->type == PBE_NPBE || three->type ==
PBE_SMPBE /* SMPBE Added */) {
02402                         val += (conc*Vcap_exp(-q*Bvec_val(vec, i), &ichop));

```

```

02403         } else if (thee->type == PBE_LPBE) {
02404             val += (conc * ( 1 - q*Bvec_val(vec, i)));
02405         }
02406     }
02407     Bvec_set(vec, i, val);
02408 }
02409 break;
02410
02411 case VDT_QDENS:
02412     u = am->u;
02413     u_d = am->ud;
02414     /* Copy in solution */
02415     Bvec_copy(vec, u);
02416     /* Add dirichlet condition */
02417     Bvec_axpy(vec, u_d, 1.0);
02418     /* Load up ions */
02419     ichop = 0;
02420     for (i=0; i<Gem_numVV(gm); i++) {
02421         val = 0;
02422         for (j=0; j<pbe->numIon; j++) {
02423             q = pbe->ionQ[j];
02424             conc = pbe->ionConc[j];
02425             if (thee->type == PBE_NPBE || thee->type ==
PBE_SMPBE /* SMPBE Added */) {
02426                 val += (q*conc*Vcap_exp(-q*Bvec_val(vec, i), &ichop));
02427             } else if (thee->type == PBE_LPBE) {
02428                 val += (q*conc*(1 - q*Bvec_val(vec, i)));
02429             }
02430         }
02431         Bvec_set(vec, i, val);
02432     }
02433     break;
02434
02435 case VDT_DIELX:
02436     Vnm_print(2, "Vfetc_fillArray: can't write out x-shifted diel!\n");
02437     return 0;
02438     break;
02439
02440 case VDT_DIELY:
02441     Vnm_print(2, "Vfetc_fillArray: can't write out y-shifted diel!\n");
02442     return 0;
02443     break;
02444
02445 case VDT_DIELZ:
02446     Vnm_print(2, "Vfetc_fillArray: can't write out z-shifted diel!\n");
02447     return 0;
02448     break;
02449
02450 case VDT_KAPPA:
02451     Vnm_print(2, "Vfetc_fillArray: can't write out kappa!\n");
02452     return 0;
02453     break;
02454
02455 default:
02456     Vnm_print(2, "Vfetc_fillArray: invalid data type (%d)!\n", type);
02457     return 0;
02458     break;
02459 }
02460
02461 return 1;
02462 }
02463
02464 VPUBLIC int Vfetc_write(Vfetc *thee, const char *iodev, const char *iofmt,
02465     const char *thost, const char *fname, Bvec *vec, Vdata_Format format) {
02466     int i, j, ichop;
02467     Aprx *aprx;
02468     Gem *gm;
02469     Vio *sock;
02470
02471     VASSERT(thee != VNULL);
02472     aprx = thee->aprx;
02473     gm = thee->gm;
02474
02475     sock = Vio_ctor(iodev, iofmt, thost, fname, "w");
02476     if (sock == VNULL) {
02477         Vnm_print(2, "Vfetc_write: Problem opening virtual socket %s\n",
02478             fname);
02479         return 0;
02480     }
02481     if (Vio_connect(sock, 0) < 0) {

```

```

02483     Vnm_print(2, "Vfetk_write: Problem connecting to virtual socket %s\n",
02484               fname);
02485     return 0;
02486 }
02487
02488 /* Make sure vec has enough rows to accomodate the vertex data */
02489 if (Bvec_numRB(vec, 0) != Gem_numVV(gm)) {
02490     Vnm_print(2, "Vfetk_fillArray: insufficient space in Bvec!\n");
02491     Vnm_print(2, "Vfetk_fillArray: Have %d, need %d!\n", Bvec_numRB(vec, 0),
02492             Gem_numVV(gm));
02493     return 0;
02494 }
02495
02496 switch (format) {
02497
02498     case VDF_DX:
02499         Aprx_writeSQL(aprx, sock, vec, "DX");
02500         break;
02501     case VDF_AVS:
02502         Aprx_writeSQL(aprx, sock, vec, "UCD");
02503         break;
02504     case VDF_UHBD:
02505         Vnm_print(2, "Vfetk_write: UHBD format not supported!\n");
02506         return 0;
02507     default:
02508         Vnm_print(2, "Vfetk_write: Invalid data format (%d)!\n", format);
02509         return 0;
02510 }
02511
02512 Vio_connectFree(sock);
02513 Vio_dtor(&sock);
02514
02515 return 1;
02516 }
02517
02518

```

10.9 src/fem/vfetk.h File Reference

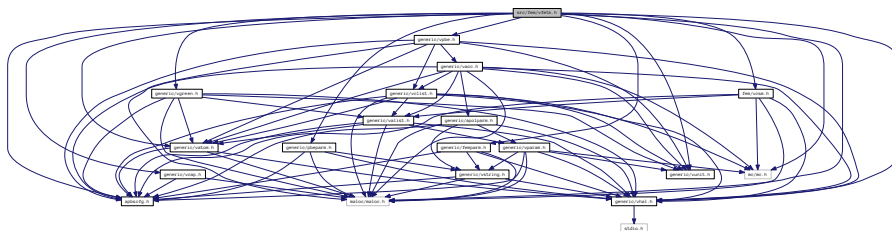
Contains declarations for class Vfetk.

```

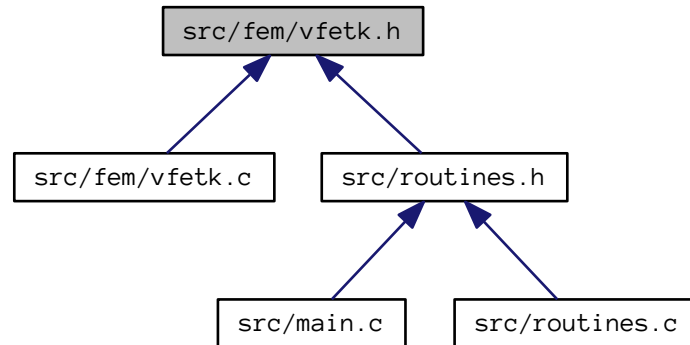
#include "apbscfg.h"
#include "malloc/malloc.h"
#include "mc/mc.h"
#include "generic/vhal.h"
#include "generic/vatom.h"
#include "generic/vpbe.h"
#include "generic/vunit.h"
#include "generic/vgreen.h"
#include "generic/vcap.h"
#include "generic/pbeparm.h"
#include "generic/femparm.h"
#include "fem/vcsm.h"

```

Include dependency graph for vfetk.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [sVfetc](#)
Contains public data members for Vfetc class/module.
- struct [sVfetc_LocalVar](#)
Vfetc LocalVar subclass.

Typedefs

- typedef enum [eVfetc_LsolvType](#) [Vfetc_LsolvType](#)
Declare FEMparm_LsolvType type.
- typedef enum [eVfetc_MeshLoad](#) [Vfetc_MeshLoad](#)
Declare FEMparm_GuessType type.
- typedef enum [eVfetc_NsolvType](#) [Vfetc_NsolvType](#)
Declare FEMparm_NsolvType type.
- typedef enum [eVfetc_GuessType](#) [Vfetc_GuessType](#)
Declare FEMparm_GuessType type.
- typedef enum [eVfetc_PrecType](#) [Vfetc_PrecType](#)
Declare FEMparm_GuessType type.
- typedef struct [sVfetc](#) [Vfetc](#)
Declaration of the Vfetc class as the Vfetc structure.
- typedef struct [sVfetc_LocalVar](#) [Vfetc_LocalVar](#)
Declaration of the Vfetc_LocalVar subclass as the Vfetc_LocalVar structure.

Enumerations

- enum `eVfetc_LsolvType` { `VLT_SLU` =0, `VLT_MG` =1, `VLT_CG` =2, `VLT_BCG` =3 }
Linear solver type.
- enum `eVfetc_MeshLoad` { `VML_DIRICUBE`, `VML_NEUMCUBE`, `VML_EXTERNAL` }
Mesh loading operation.
- enum `eVfetc_NsolvType` { `VNT_NEW` =0, `VNT_INC` =1, `VNT_ARC` =2 }
Non-linear solver type.
- enum `eVfetc_GuessType` { `VGT_ZERO` =0, `VGT_DIRI` =1, `VGT_PREV` =2 }
Initial guess type.
- enum `eVfetc_PrecType` { `VPT_IDEN` =0, `VPT_DIAG` =1, `VPT_MG` =2 }
Preconditioner type.

Functions

- VEXTERNC Gem * `Vfetc_getGem` (`Vfetc` *thee)
Get a pointer to the Gem (grid manager) object.
- VEXTERNC AM * `Vfetc_getAM` (`Vfetc` *thee)
Get a pointer to the AM (algebra manager) object.
- VEXTERNC Vpbe * `Vfetc_getVpbe` (`Vfetc` *thee)
Get a pointer to the Vpbe (PBE manager) object.
- VEXTERNC Vcsm * `Vfetc_getVcsm` (`Vfetc` *thee)
Get a pointer to the Vcsm (charge-simplex map) object.
- VEXTERNC int `Vfetc_getAtomColor` (`Vfetc` *thee, int iatom)
Get the partition information for a particular atom.
- VEXTERNC `Vfetc` * `Vfetc_ctor` (`Vpbe` *pbe, `Vhal_PBEType` type)
Constructor for Vfetc object.
- VEXTERNC int `Vfetc_ctor2` (`Vfetc` *thee, `Vpbe` *pbe, `Vhal_PBEType` type)
FORTTRAN stub constructor for Vfetc object.
- VEXTERNC void `Vfetc_dtor` (`Vfetc` **thee)
Object destructor.
- VEXTERNC void `Vfetc_dtor2` (`Vfetc` *thee)
FORTTRAN stub object destructor.
- VEXTERNC double * `Vfetc_getSolution` (`Vfetc` *thee, int *length)
Create an array containing the solution (electrostatic potential in units of $k_B T / e$) at the finest mesh level.
- VEXTERNC void `Vfetc_setParameters` (`Vfetc` *thee, `PBEparm` *pbeparm, `FEMparm` *feparm)
Set the parameter objects.
- VPUBLIC double `Vfetc_energy` (`Vfetc` *thee, int color, int nonlin)
Return the total electrostatic energy.
- VEXTERNC double `Vfetc_dqmEnergy` (`Vfetc` *thee, int color)
Get the "mobile charge" and "polarization" contributions to the electrostatic energy.
- VEXTERNC double `Vfetc_qfEnergy` (`Vfetc` *thee, int color)
Get the "fixed charge" contribution to the electrostatic energy.
- VEXTERNC unsigned long int `Vfetc_memChk` (`Vfetc` *thee)
Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC void `Vfetc_setAtomColors` (`Vfetc` *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)
FORTTRAN 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 txq[][VAPBS_DIM], void *data)
Do once-per-element initialization.
- VEXTERNC void [Vfetk_PDE_initFace](#) (PDE *thee, int faceType, int chart, double tvec[])
Do once-per-face initialization.
- VEXTERNC void [Vfetk_PDE_initPoint](#) (PDE *thee, int pointType, int chart, double txq[], double tU[], double td←U[][VAPBS_DIM])
Do once-per-point initialization.
- VEXTERNC void [Vfetk_PDE_Fu](#) (PDE *thee, int key, double F[])
Evaluate strong form of PBE. For interior points, this is:

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

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

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

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

- VEXTERNC double [Vfetk_PDE_Fu_v](#) (PDE *thee, int key, double V[], double dV[][VAPBS_DIM])
This is the weak form of the PBE; i.e. the strong form integrated with a test function to give:

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

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

- VEXTERNC double [Vfetk_PDE_DFu_wv](#) (PDE *thee, int key, double W[], double dW[][VAPBS_DIM], double V[], double dV[][VAPBS_DIM])
This is the linearization of the weak form of the PBE; e.g., for use in a Newton iteration. This is the functional linearization of the strong form integrated with a test function to give:

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

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

- VEXTERNC void [Vfetc_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 [Vfetc_PDE_u_D](#) (PDE *thee, int type, int chart, double txq[], double F[])
Evaluate the Dirichlet boundary condition at the given point.
- VEXTERNC void [Vfetc_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 [Vfetc_PDE_bisectEdge](#) (int dim, int dimII, int edgeType, int chart[], double vx[][VAPBS_DIM])
Define the way manifold edges are bisected.
- VEXTERNC void [Vfetc_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 [Vfetc_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 [Vfetc_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 [Vfetc_PDE_Ju](#) (PDE *thee, int key)
Energy functional. This returns the energy (less delta function terms) in the form:
$$c^{-1}/2 \int (\epsilon(\nabla u)^2 + \kappa^2(\cosh u - 1)) dx$$

for a 1:1 electrolyte where c is the output from Vpbe_getZmagic.
- VEXTERNC void [Vfetc_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 [Vfetc_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 [Vfetc_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 [Vfetc_readMesh](#) (Vfetc *thee, int skey, Vio *sock)
Read in mesh and initialize associated internal structures.
- VEXTERNC void [Vfetc_dumpLocalVar](#) ()
Debugging routine to print out local variables used by PDE object.
- VEXTERNC int [Vfetc_fillArray](#) (Vfetc *thee, Bvec *vec, Vdata_Type type)
Fill an array with the specified data.
- VEXTERNC int [Vfetc_write](#) (Vfetc *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, Bvec *vec, Vdata_Format format)
Write out data.
- VEXTERNC Vrc_Codes [Vfetc_loadGem](#) (Vfetc *thee, Gem *gm)
Load a Gem geometry manager object into Vfetc.

10.9.1 Detailed Description

Contains declarations for class `Vfetk`.

Version

`Id`

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [vfetk.h](#).

10.10 vfetk.h

```

00001
00062 #ifndef _VFETK_H_
00063 #define _VFETK_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "malloc/malloc.h"
00068 #include "mc/mc.h"
00069
00070 #include "generic/vhal.h"
00071 #include "generic/vatom.h"
00072 // #include "generic/valist.h"
00073 #include "generic/vpbe.h"
00074 #include "generic/vunit.h"
00075 #include "generic/vgreen.h"
00076 #include "generic/vcap.h"
00077 #include "generic/pbeparm.h"
00078 #include "generic/femparm.h"
00079 #include "fem/vcsm.h"
00080
00086 enum eVfetk_LsolvType {
00087     VLT_SLU=0,
00088     VLT_MG=1,
00089     VLT_CG=2,
00090     VLT_BCG=3
00091 };
00092
00097 typedef enum eVfetk_LsolvType Vfetk_LsolvType;
00098
00099
00104 enum eVfetk_MeshLoad {
00105     VML_DIRICUBE,
00106     VML_NEUMCUBE,
00107     VML_EXTERNAL
00108 };
00109
00114 typedef enum eVfetk_MeshLoad Vfetk_MeshLoad;
00115
00121 enum eVfetk_NsolvType {
00122     VNT_NEW=0,
00123     VNT_INC=1,
00124     VNT_ARC=2
00125 };
00126
00131 typedef enum eVfetk_NsolvType Vfetk_NsolvType;
00132
00138 enum eVfetk_GuessType {
00139     VGT_ZERO=0,
00140     VGT_DIRI=1,
00141     VGT_PREV=2
00142 };
00143
00148 typedef enum eVfetk_GuessType Vfetk_GuessType;
00149
00155 enum eVfetk_PrecType {
00156     VPT_IDEN=0,
00157     VPT_DIAG=1,
00158     VPT_MG=2
00159 };
00160
00165 typedef enum eVfetk_PrecType Vfetk_PrecType;
00166
00176 struct sVfetk {
00177
00178     Vmem *vmem;
00179     Gem *gm;
00182     AM *am;
00183     Aprx *aprx;
00184     PDE *pde;
00185     Vpbe *pbpe;
00186     Vcsm *csm;
00187     Vfetk_LsolvType lkey;
00188     int lmax;
00189     double ltol;
00190     Vfetk_NsolvType nkey;
00191     int nmax;
00192     double ntol;
00193     Vfetk_GuessType gues;

```

```

00194  Vfetc_PrecType lprec;
00195  int pjac;
00197  PBEparm *pbeparm;
00198  FEMparm *feparm;
00199  Vhal_PBEType type;
00200  int level;
00202 };
00203
00207 typedef struct sVfetc Vfetc;
00208
00215 struct sVfetc_LocalVar {
00216     double nvec[VAPBS_DIM];
00217     double vx[4][VAPBS_DIM];
00218     double xq[VAPBS_DIM];
00219     double U[MAXV];
00220     double dU[MAXV][VAPBS_DIM];
00221     double W;
00222     double dW[VAPBS_DIM];
00223     double d2W;
00224     int sType;
00225     int fType;
00226     double diel;
00227     double ionacc;
00228     double A;
00229     double F;
00230     double B;
00231     double DB;
00232     double jumpDiel;
00233     Vfetc *fetc;
00234     Vgreen *green;
00235     int initGreen;
00237     SS *simp;
00239     VV *verts[4];
00240     int nverts;
00241     double ionConc[MAXION];
00242     double ionQ[MAXION];
00243     double ionRadii[MAXION];
00244     double zkappa2;
00245     double zks2;
00246     double ionstr;
00247     int nion;
00248     double Fu_v;
00249     double DFu_wv;
00250     double delta;
00251     double u_D;
00252     double u_T;
00253 };
00254
00259 typedef struct sVfetc_LocalVar Vfetc_LocalVar;
00260
00261 #if !defined(VINLINE_VFETK)
00262
00268     VEXTERNC Gem* Vfetc_getGem(
00269         Vfetc *thee
00270     );
00271
00277     VEXTERNC AM* Vfetc_getAM(
00278         Vfetc *thee
00279     );
00280
00286     VEXTERNC Vpbe* Vfetc_getVpbe(
00287         Vfetc *thee
00288     );
00289
00295     VEXTERNC Vcsm* Vfetc_getVcsm(
00296         Vfetc *thee
00297     );
00298
00305     VEXTERNC int Vfetc_getAtomColor(
00306         Vfetc *thee,
00307         int iatom
00308     );
00309
00310 #else /* if defined(VINLINE_VFETK) */
00311 #   define Vfetc_getGem(thee) ((thee)->gm)
00312 #   define Vfetc_getAM(thee) ((thee)->am)
00313 #   define Vfetc_getVpbe(thee) ((thee)->pbe)
00314 #   define Vfetc_getVcsm(thee) ((thee)->csm)
00315 #   define Vfetc_getAtomColor(thee, iatom) (Vatom_getPartID(Valist_getAtom(Vpbe_getValist(thee->pbe),
00316                                     iatom)))
00316 #endif /* if !defined(VINLINE_VFETK) */

```

```

00317
00318 /* ////////////////////////////////////////
00319 // Class Vfetk: Non-Inlineable methods (vfetk.c)
00320
00321
00331 VEXTERNC Vfetk* Vfetk_ctor(
00332     Vpbe *pbe,
00333     Vhal_PBEType type
00334 );
00335
00345 VEXTERNC int Vfetk_ctor2(
00346     Vfetk *thee,
00347     Vpbe *pbe,
00348     Vhal_PBEType type
00349 );
00350
00356 VEXTERNC void Vfetk_dtor(
00357     Vfetk **thee
00358 );
00359
00365 VEXTERNC void Vfetk_dtor2(
00366     Vfetk *thee
00367 );
00368
00378 VEXTERNC double* Vfetk_getSolution(
00379     Vfetk *thee,
00380     int *length
00381 );
00382
00388 VEXTERNC void Vfetk_setParameters(
00389     Vfetk *thee,
00390     PBEparm *pbeparm,
00391     FEMparm *feparm
00392 );
00393
00412 VEXTERNC double Vfetk_energy(
00413     Vfetk *thee,
00414     int color,
00418     int nonlin
00420 );
00421
00451 VEXTERNC double Vfetk_dqmEnergy(
00452     Vfetk *thee,
00453     int color
00457 );
00458
00476 VEXTERNC double Vfetk_qfEnergy(
00477     Vfetk *thee,
00478     int color
00480 );
00481
00489 VEXTERNC unsigned long int Vfetk_memChk(
00490     Vfetk *thee
00491 );
00492
00508 VEXTERNC void Vfetk_setAtomColors(
00509     Vfetk *thee
00510 );
00511
00520 VEXTERNC void Bmat_printHB(
00521     Bmat *thee,
00522     char *fname
00523 );
00524
00530 VEXTERNC Vrc_Codes Vfetk_genCube(
00531     Vfetk *thee,
00532     double center[3],
00533     double length[3],
00534     Vfetk_MeshLoad meshType
00535 );
00536
00542 VEXTERNC Vrc_Codes Vfetk_loadMesh(
00543     Vfetk *thee,
00544     double center[3],
00545     double length[3],
00546     Vfetk_MeshLoad meshType,
00547     Vio *sock
00548 );
00549
00556 VEXTERNC PDE* Vfetk_PDE_ctor(
00557     Vfetk *fetk
00558 );

```

```
00559
00566 VEXTERNC int Vfetc_PDE_ctor2(
00567     PDE *thee,
00568     Vfetc *fetc
00569 );
00570
00577 VEXTERNC void Vfetc_PDE_dtor(
00578     PDE **thee
00579 );
00580
00587 VEXTERNC void Vfetc_PDE_dtor2(
00588     PDE *thee
00589 );
00590
00596 VEXTERNC void Vfetc_PDE_initAssemble(
00597     PDE *thee,
00598     int ip[],
00599     double rp[]
00600 );
00601
00608 VEXTERNC void Vfetc_PDE_initElement(
00609     PDE *thee,
00610     int elementType,
00611     int chart,
00614     double tvx[][VAPBS_DIM],
00615     void *data
00616 );
00617
00623 VEXTERNC void Vfetc_PDE_initFace(
00624     PDE *thee,
00625     int faceType,
00627     int chart,
00629     double tnvec[]
00630 );
00631
00639 VEXTERNC void Vfetc_PDE_initPoint(
00640     PDE *thee,
00641     int pointType,
00642     int chart,
00644     double txq[],
00645     double tU[],
00646     double tdU[][VAPBS_DIM]
00647 );
00648
00666 VEXTERNC void Vfetc_PDE_Fu(
00667     PDE *thee,
00668     int key,
00670     double F[]
00671 );
00672
00683 VEXTERNC double Vfetc_PDE_Fu_v(
00684     PDE *thee,
00685     int key,
00687     double V[],
00688     double dV[][VAPBS_DIM]
00689 );
00690
00702 VEXTERNC double Vfetc_PDE_DF_uwv(
00703     PDE *thee,
00704     int key,
00706     double W[],
00707     double dW[][VAPBS_DIM],
00708     double V[],
00709     double dV[][VAPBS_DIM]
00710 );
00711
00718 VEXTERNC void Vfetc_PDE_delta(
00719     PDE *thee,
00720     int type,
00721     int chart,
00722     double txq[],
00723     void *user,
00724     double F[]
00725 );
00726
00734 VEXTERNC void Vfetc_PDE_u_D(
00735     PDE *thee,
00736     int type,
00737     int chart,
00738     double txq[],
00739     double F[]
```



```
00740         );
00741
00749 VEXTERNC void Vfetk_PDE_u_T(
00750     PDE *thee,
00751     int type,
00752     int chart,
00753     double txq[],
00754     double F[]
00755 );
00756
00762 VEXTERNC void Vfetk_PDE_bisectEdge(
00763     int dim,
00764     int dimII,
00765     int edgeType,
00766     int chart[],
00768     double vx[][VAPBS_DIM]
00769 );
00770
00776 VEXTERNC void Vfetk_PDE_mapBoundary(
00777     int dim,
00778     int dimII,
00779     int vertexType,
00780     int chart,
00781     double vx[VAPBS_DIM]
00782 );
00783
00792 VEXTERNC int Vfetk_PDE_markSimplex(
00793     int dim,
00794     int dimII,
00795     int simplexType,
00796     int faceType[VAPBS_NVS],
00797     int vertexType[VAPBS_NVS],
00798     int chart[],
00799     double vx[][VAPBS_DIM],
00800     void *simplex
00801 );
00802
00808 VEXTERNC void Vfetk_PDE_oneChart(
00809     int dim,
00810     int dimII,
00811     int objType,
00812     int chart[],
00813     double vx[][VAPBS_DIM],
00814     int dimV
00815 );
00816
00826 VEXTERNC double Vfetk_PDE_Ju(
00827     PDE *thee,
00828     int key
00829 );
00830
00838 VEXTERNC void Vfetk_externalUpdateFunction(
00839     SS **simps,
00841     int num
00842 );
00843
00844
00907 VEXTERNC int Vfetk_PDE_simplexBasisInit(
00908     int key,
00910     int dim,
00911     int comp,
00913     int *ndof,
00914     int dof[]
00915 );
00916
00924 VEXTERNC void Vfetk_PDE_simplexBasisForm(
00925     int key,
00927     int dim,
00928     int comp ,
00929     int pdkey,
00938     double xq[],
00939     double basis[]
00941 );
00942
00948 VEXTERNC void Vfetk_readMesh(
00949     Vfetk *thee,
00950     int skey,
00951     Vio *sock
00952 );
00953
00959 VEXTERNC void Vfetk_dumpLocalVar();
```

```

00960
00968 VEXTERNC int Vfetc_fillArray(
00969     Vfetc *thee,
00970     Bvec *vec,
00971     Vdata_Type type
00972 );
00973
00988 VEXTERNC int Vfetc_write(
00989     Vfetc *thee,
00990     const char *iodev,
00992     const char *iofmt,
00994     const char *thost,
00995     const char *fname,
00996     Bvec *vec,
00997     Vdata_Format format
00998 );
00999
01005 VEXTERNC Vrc_Codes Vfetc_loadGem(
01006     Vfetc *thee,
01007     Gem *gm
01008 );
01009
01010
01011 #endif /* ifndef _VFETK_H_ */

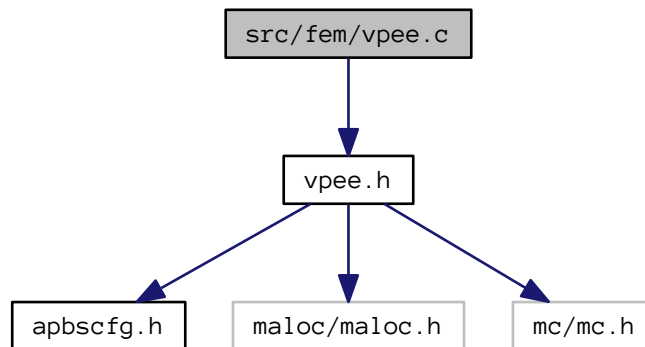
```

10.11 src/fem/vpee.c File Reference

Class Vpee methods.

```
#include "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.

10.11.1 Detailed Description

Class Vpee methods.

Author

Nathan Baker

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
```

```

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

```

Definition in file [vpee.c](#).

10.12 vpee.c

```

00001
00057 #include "vpee.h"
00058
00059 VPRIVATE int Vpee_userDefined(Vpee *thee,
00060                               SS *sm
00061                               );
00062 VPRIVATE int Vpee_ourSimp(Vpee *thee,
00063                           SS *sm,
00064                           int rcol
00065                           );
00066 VEXTERNC double Aprx_estNonlinResid(Aprx *thee,
00067                                     SS *sm,
00068                                     Bvec *u,
00069                                     Bvec *ud,
00070                                     Bvec *f
00071                                     );
00072 VEXTERNC double Aprx_estLocalProblem(Aprx *thee,
00073                                      SS *sm,
00074                                      Bvec *u,
00075                                      Bvec *ud,
00076                                      Bvec *f);
00077 VEXTERNC double Aprx_estDualProblem(Aprx *thee,
00078                                     SS *sm,
00079                                     Bvec *u,
00080                                     Bvec *ud,
00081                                     Bvec *f
00082                                     );
00083
00084 /* ////////////////////////////////////////////////////////////////////
00085 // Class Vpee: Non-inlineable methods
00086 ////////////////////////////////////////////////////////////////////
00087
00088 /* ////////////////////////////////////////////////////////////////////
00089 // Routine: Vpee_ctor
00090 //
00091 // Author:   Nathan Baker
00093 VPUBLIC Vpee* Vpee_ctor(Gem *gm,
00094                       int localPartID,
00095                       int killFlag,
00096                       double killParam
00097                       ) {
00098
00099     Vpee *thee = VNULL;
00100
00101     /* Set up the structure */
00102     thee = Vmem_malloc(VNULL, 1, sizeof(Vpee) );
00103     VASSERT( thee != VNULL);
00104     VASSERT( Vpee_ctor2(thee, gm, localPartID, killFlag, killParam));
00105
00106     return thee;
00107 }
00108
00109 /* ////////////////////////////////////////////////////////////////////
00110 // Routine: Vpee_ctor2
00111 //

```

```

00112 // Author:   Nathan Baker
00114 VPUBLIC int Vpee_ctor2(Vpee *thee,
00115                       Gem *gm,
00116                       int localPartID,
00117                       int killFlag,
00118                       double killParam
00119                       ) {
00120
00121     int ivert,
00122         nLocalVerts;
00123     SS *simp;
00124     VV *vert;
00125     double radius,
00126         dx,
00127         dy,
00128         dz;
00129
00130     VASSERT(thee != VNULL);
00131
00132     /* Sanity check on input values */
00133     if (killFlag == 0) {
00134         Vnm_print(0, "Vpee_ctor2: No error attenuation outside partition.\n");
00135     } else if (killFlag == 1) {
00136         Vnm_print(0, "Vpee_ctor2: Error outside local partition ignored.\n");
00137     } else if (killFlag == 2) {
00138         Vnm_print(0, "Vpee_ctor2: Error ignored outside sphere with radius %4.3f times the radius of the
circumscribing sphere\n", killParam);
00139         if (killParam < 1.0) {
00140             Vnm_print(2, "Vpee_ctor2: Warning! Parameter killParam = %4.3 < 1.0!\n",
killParam);
00141             Vnm_print(2, "Vpee_ctor2: This may result in non-optimal marking and refinement!\n");
00142         }
00143     } else if (killFlag == 3) {
00144         Vnm_print(0, "Vpee_ctor2: Error outside local partition and immediate neighbors ignored [NOT
IMPLEMENTED].\n");
00145     } else {
00146         Vnm_print(2, "Vpee_ctor2: UNRECOGNIZED killFlag PARAMETER! BAILING!\n");
00147         VASSERT(0);
00148     }
00149
00150     thee->gm = gm;
00151     thee->localPartID = localPartID;
00152     thee->killFlag = killFlag;
00153     thee->killParam = killParam;
00154     thee->mem = Vmem_ctor("APBS::VPEE");
00155
00156     /* Now, figure out the center of geometry for the local partition. The
00157     * general plan is to loop through the vertices, loop through the
00158     * vertices' simplex lists and find the vertices with simplices containing
00159     * chart values we're interested in. */
00160     thee->localPartCenter[0] = 0.0;
00161     thee->localPartCenter[1] = 0.0;
00162     thee->localPartCenter[2] = 0.0;
00163     nLocalVerts = 0;
00164     for (ivert=0; ivert<Gem_numVV(thee->gm); ivert++) {
00165         vert = Gem_VV(thee->gm, ivert);
00166         simp = VV_firstSS(vert);
00167         VASSERT(simp != VNULL);
00168         while (simp != VNULL) {
00169             if (SS_chart(simp) == thee->localPartID) {
00170                 thee->localPartCenter[0] += VV_coord(vert, 0);
00171                 thee->localPartCenter[1] += VV_coord(vert, 1);
00172                 thee->localPartCenter[2] += VV_coord(vert, 2);
00173                 nLocalVerts++;
00174                 break;
00175             }
00176             simp = SS_link(simp, vert);
00177         }
00178     }
00179     VASSERT(nLocalVerts > 0);
00180     thee->localPartCenter[0] =
thee->localPartCenter[0]/((double) (nLocalVerts));
00181     thee->localPartCenter[1] =
thee->localPartCenter[1]/((double) (nLocalVerts));
00182     thee->localPartCenter[2] =
thee->localPartCenter[2]/((double) (nLocalVerts));
00183     Vnm_print(0, "Vpee_ctor2: Part %d centered at (%4.3f, %4.3f, %4.3f)\n",
thee->localPartID, thee->localPartCenter[0], thee->localPartCenter[1],
thee->localPartCenter[2]);
00184
00185
00186
00187
00188
00189
00190
00191

```

```

00192     /* Now, figure out the radius of the sphere circumscribing the local
00193     * partition. We need to keep track of vertices so we don't double count
00194     * them. */
00195     thee->localPartRadius = 0.0;
00196     for (ivert=0; ivert<Gem_numVV(thee->gm); ivert++) {
00197         vert = Gem_VV(thee->gm, ivert);
00198         simp = VV_firstSS(vert);
00199         VASSERT(simp != VNULL);
00200         while (simp != VNULL) {
00201             if (SS_chart(simp) == thee->localPartID) {
00202                 dx = thee->localPartCenter[0] - VV_coord(vert, 0);
00203                 dy = thee->localPartCenter[1] - VV_coord(vert, 1);
00204                 dz = thee->localPartCenter[2] - VV_coord(vert, 2);
00205                 radius = dx*dx + dy*dy + dz*dz;
00206                 if (radius > thee->localPartRadius) thee->localPartRadius =
00207                     radius;
00208                 break;
00209             }
00210             simp = SS_link(simp, vert);
00211         }
00212     }
00213     thee->localPartRadius = VSQRT(thee->localPartRadius);
00214     Vnm_print(0, "Vpee_ctor2: Part %d has circumscribing sphere of radius %4.3f\n",
00215         thee->localPartID, thee->localPartRadius);
00216     return 1;
00217 }
00218
00219 /* =====
00220 // Routine: Vpee_dtor
00221 //
00222 // Author: Nathan Baker
00225 VPUBLIC void Vpee_dtor(Vpee **thee) {
00226     if ((*thee) != VNULL) {
00227         Vpee_dtor2(*thee);
00228         Vmem_free(VNULL, 1, sizeof(Vpee), (void **)thee);
00229         (*thee) = VNULL;
00230     }
00231 }
00232
00233 }
00234
00235 /* =====
00236 // Routine: Vpee_dtor2
00237 //
00238 // Author: Nathan Baker
00240 VPUBLIC void Vpee_dtor2(Vpee *thee) {
00241     Vmem_dtor(&(thee->mem));
00242 }
00243
00244 /* =====
00245 // Routine: Vpee_markRefine
00246 //
00247 // Author: Nathan Baker (and Michael Holst: the author of AM_markRefine, on
00248 // which this is based)
00250 VPUBLIC int Vpee_markRefine(Vpee *thee,
00251                             AM *am,
00252                             int level,
00253                             int akey,
00254                             int rcol,
00255                             double etol,
00256                             int bkey
00257                             ) {
00258
00259     Aprx *aprx;
00260     int marked = 0,
00261         markMe,
00262         i,
00263         smid,
00264         count,
00265         currentQ;
00266     double minError = 0.0,
00267         maxError = 0.0,
00268         errEst = 0.0,
00269         mlevel,
00270         barrier;
00271     SS *sm;
00272
00273     VASSERT(thee != VNULL);
00275

```

```

00276      /* Get the Aprx object from AM */
00277      aprx = am->aprx;
00278
00279      /* input check and some i/o */
00280      if ( ! ((-1 <= akey) && (akey <= 4)) ) {
00281          Vnm_print(0,"Vpee_markRefine: bad refine key; simplices marked = %d\n",
00282                  marked);
00283          return marked;
00284      }
00285
00286      /* For uniform markings, we have no effect */
00287      if ((-1 <= akey) && (akey <= 0)) {
00288          marked = Gem_markRefine(thee->gm, akey, rcol);
00289          return marked;
00290      }
00291
00292      /* Informative I/O */
00293      if (akey == 2) {
00294          Vnm_print(0,"Vpee_estRefine: using Aprx_estNonlinResid().\n");
00295      } else if (akey == 3) {
00296          Vnm_print(0,"Vpee_estRefine: using Aprx_estLocalProblem().\n");
00297      } else if (akey == 4) {
00298          Vnm_print(0,"Vpee_estRefine: using Aprx_estDualProblem().\n");
00299      } else {
00300          Vnm_print(0,"Vpee_estRefine: bad key given; simplices marked = %d\n",
00301                  marked);
00302          return marked;
00303      }
00304      if (thee->killFlag == 0) {
00305          Vnm_print(0, "Vpee_markRefine: No error attenuation -- simplices in all partitions will be marked.
00306          \n");
00307      } else if (thee->killFlag == 1) {
00308          Vnm_print(0, "Vpee_markRefine: Maximum error attenuation -- only simplices in local partition will
00309          be marked.\n");
00310      } else if (thee->killFlag == 2) {
00311          Vnm_print(0, "Vpee_markRefine: Spherical error attenuation -- simplices within a sphere of %4.3f
00312          times the size of the partition will be marked\n",
00313                  thee->killParam);
00314      } else if (thee->killFlag == 2) {
00315          Vnm_print(0, "Vpee_markRefine: Neighbor-based error attenuation -- simplices in the local and
00316          neighboring partitions will be marked [NOT IMPLEMENTED]!\n");
00317          VASSERT(0);
00318      } else {
00319          Vnm_print(2,"Vpee_markRefine: bogus killFlag given; simplices marked = %d\n",
00320                  marked);
00321          return marked;
00322      }
00323
00324      /* set the barrier type */
00325      mlevel = (etol*etol) / Gem_numSS(thee->gm);
00326      if (bkey == 0) {
00327          barrier = (etol*etol);
00328          Vnm_print(0,"Vpee_estRefine: forcing [err per S] < [TOL] = %g\n",
00329                  barrier);
00330      } else if (bkey == 1) {
00331          barrier = mlevel;
00332          Vnm_print(0,"Vpee_estRefine: forcing [err per S] < [(TOL^2/numS)^(1/2)] = %g\n",
00333                  VSQRT(barrier));
00334      } else {
00335          Vnm_print(0,"Vpee_estRefine: bad bkey given; simplices marked = %d\n",
00336                  marked);
00337          return marked;
00338      }
00339
00340      /* timer */
00341      Vnm_tstart(30, "error estimation");
00342
00343      /* count = num generations to produce from marked simplices (minimally) */
00344      count = 1; /* must be >= 1 */
00345
00346      /* check the refinement Q for emptiness */
00347      currentQ = 0;
00348      if (Gem_numSQ(thee->gm,currentQ) > 0) {
00349          Vnm_print(0,"Vpee_markRefine: non-empty refinement Q%d....clearing..",
00350                  currentQ);
00351          Gem_resetSQ(thee->gm,currentQ);
00352          Vnm_print(0,"..done.\n");
00353      }
00354      if (Gem_numSQ(thee->gm,!currentQ) > 0) {
00355          Vnm_print(0,"Vpee_markRefine: non-empty refinement Q%d....clearing..",
00356                  !currentQ);
00357      }

```

```

00353     Gem_resetSQ(thee->gm,!currentQ);
00354     Vnm_print(0,"..done.\n");
00355 }
00356 VASSERT( Gem_numSQ(thee->gm,currentQ) == 0 );
00357 VASSERT( Gem_numSQ(thee->gm,!currentQ) == 0 );
00358
00359 /* clear everyone's refinement flags */
00360 Vnm_print(0,"Vpee_markRefine: clearing all simplex refinement flags..");
00361 for (i=0; i<Gem_numSS(thee->gm); i++) {
00362     if ( (i>0) && (i % VPRTKEY) == 0 ) Vnm_print(0,"[MS:%d]",i);
00363     sm = Gem_SS(thee->gm,i);
00364     SS_setRefineKey(sm,currentQ,0);
00365     SS_setRefineKey(sm,!currentQ,0);
00366     SS_setRefinementCount(sm,0);
00367 }
00368 Vnm_print(0,"..done.\n");
00369
00370 /* NON-ERROR-BASED METHODS */
00371 /* Simplex flag clearing */
00372 if (akey == -1) return marked;
00373 /* Uniform & user-defined refinement*/
00374 if ((akey == 0) || (akey == 1)) {
00375     smid = 0;
00376     while ( smid < Gem_numSS(thee->gm)) {
00377         /* Get the simplex and find out if it's markable */
00378         sm = Gem_SS(thee->gm,smid);
00379         markMe = Vpee_ourSimp(thee, sm, rcol);
00380         if (markMe) {
00381             if (akey == 0) {
00382                 marked++;
00383                 Gem_appendSQ(thee->gm,currentQ, sm);
00384                 SS_setRefineKey(sm,currentQ,1);
00385                 SS_setRefinementCount(sm,count);
00386             } else if (Vpee_userDefined(thee, sm)) {
00387                 marked++;
00388                 Gem_appendSQ(thee->gm,currentQ, sm);
00389                 SS_setRefineKey(sm,currentQ,1);
00390                 SS_setRefinementCount(sm,count);
00391             }
00392         }
00393         smid++;
00394     }
00395 }
00396
00397 /* ERROR-BASED METHODS */
00398 /* gerror = global error accumulation */
00399 aprx->gerror = 0.;
00400
00401 /* traverse the simplices and process the error estimates */
00402 Vnm_print(0,"Vpee_markRefine: estimating error..");
00403 smid = 0;
00404 while ( smid < Gem_numSS(thee->gm)) {
00405
00406     /* Get the simplex and find out if it's markable */
00407     sm = Gem_SS(thee->gm,smid);
00408     markMe = Vpee_ourSimp(thee, sm, rcol);
00409
00410     if ( (smid>0) && (smid % VPRTKEY) == 0 ) Vnm_print(0,"[MS:%d]",smid);
00411
00412     /* Produce an error estimate for this element if it is in the set */
00413     if (markMe) {
00414         if (akey == 2) {
00415             errEst = Aprx_estNonlinResid(aprx, sm, am->u,am->ud,am->f);
00416         } else if (akey == 3) {
00417             errEst = Aprx_estLocalProblem(aprx, sm, am->u,am->ud,am->f);
00418         } else if (akey == 4) {
00419             errEst = Aprx_estDualProblem(aprx, sm, am->u,am->ud,am->f);
00420         }
00421         VASSERT( errEst >= 0. );
00422
00423         /* if error estimate above tol, mark element for refinement */
00424         if ( errEst > barrier ) {
00425             marked++;
00426             Gem_appendSQ(thee->gm,currentQ, sm); /*add to refinement Q*/
00427             SS_setRefineKey(sm,currentQ,1);      /* note now on refine Q */
00428             SS_setRefinementCount(sm,count);      /* refine X many times? */
00429         }
00430
00431         /* keep track of min/max errors over the mesh */
00432         minError = VMIN2( VSQRT(VABS(errEst)), minError );
00433         maxError = VMAX2( VSQRT(VABS(errEst)), maxError );

```



```

00434
00435     /* store the estimate */
00436     Bvec_set( aprx->wev, smid, errEst );
00437
00438     /* accumulate into global error (errEst is SQUARED already) */
00439     aprx->gerror += errEst;
00440
00441     /* otherwise store a zero for the estimate */
00442     } else {
00443         Bvec_set( aprx->wev, smid, 0. );
00444     }
00445
00446     smid++;
00447 }
00448
00449 /* do some i/o */
00450 Vnm_print(0, "..done. [marked=<%d/%d>]\n", marked, Gem_numSS(thee->gm));
00451 Vnm_print(0, "Vpee_estRefine: TOL=<%g> Global_Error=<%g>\n",
00452     etol, aprx->gerror);
00453 Vnm_print(0, "Vpee_estRefine: (TOL^2/numS)^(1/2)=<%g> Max_Ele_Error=<%g>\n",
00454     VSQRT(mlevel), maxError);
00455 Vnm_tstop(30, "error estimation");
00456
00457 /* check for making the error tolerance */
00458 if ((bkey == 1) && (aprx->gerror <= etol)) {
00459     Vnm_print(0,
00460         "Vpee_estRefine: *****\n");
00461     Vnm_print(0,
00462         "Vpee_estRefine: Global Error criterion met; setting marked=0.\n");
00463     Vnm_print(0,
00464         "Vpee_estRefine: *****\n");
00465     marked = 0;
00466 }
00467
00468 /* return */
00469 return marked;
00470
00471 }
00472
00473
00474 /* ////////////////////////////////////////
00475 // Routine: Vpee_numSS
00476 //
00477 // Author: Nathan Baker
00479 VPUBLIC int Vpee_numSS(Vpee *thee) {
00480     int num = 0;
00481     int isimp;
00482
00483     for (isimp=0; isimp<Gem_numSS(thee->gm); isimp++) {
00484         if (SS_chart(Gem_SS(thee->gm, isimp)) == thee->localPartID) num++;
00485     }
00486
00487     return num;
00488 }
00489
00490 /* ////////////////////////////////////////
00491 // Routine: Vpee_userDefined
00492 //
00493 // Purpose: Reduce code bloat by wrapping up the common steps for getting the
00494 //           user-defined error estimate
00495 //
00496 // Author: Nathan Baker
00498 VPRIVATE int Vpee_userDefined(Vpee *thee,
00499                             SS *sm
00500                             ) {
00501
00502     int ivert,
00503         icoord,
00504         chart[4],
00505         fType[4],
00506         vType[4];
00507     double vx[4][3];
00508
00509     for (ivert=0; ivert<Gem_dimVV(thee->gm); ivert++) {
00510         fType[ivert] = SS_faceType(sm, ivert);
00511         vType[ivert] = VV_type(SS_vertex(sm, ivert));
00512         chart[ivert] = VV_chart(SS_vertex(sm, ivert));
00513         for (icoord=0; icoord<Gem_dimII(thee->gm); icoord++) {
00514             vx[ivert][icoord] = VV_coord(SS_vertex(sm, ivert), icoord);
00515         }
00516     }

```

```

00517     return thee->gm->pde->markSimplex(Gem_dim(thee->gm), Gem_dimII(thee->gm),
00518         SS_type(sm), fType, vType, chart, vx, sm);
00519 }
00520
00521 /* ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00522 // Routine:  Vpee_ourSimp
00523 //
00524 // Purpose:  Reduce code bloat by wrapping up the common steps for determining
00525 //           whether the given simplex can be marked (i.e., belongs to our
00526 //           partition or overlap region)
00527 //
00528 // Returns:  1 if could be marked, 0 otherwise
00529 //
00530 // Author:   Nathan Baker
00531 VPRIVATE int Vpee_ourSimp(Vpee *thee,
00532     SS *sm,
00533     int rcol
00534 ) {
00535
00536     int ivert;
00537     double dist,
00538         dx,
00539         dy,
00540         dz;
00541
00542     if (thee->killFlag == 0) return 1;
00543     else if (thee->killFlag == 1) {
00544         if ((SS_chart(sm) == rcol) || (rcol < 0)) return 1;
00545     } else if (thee->killFlag == 2) {
00546         if (rcol < 0) return 1;
00547     } else {
00548         /* We can only do distance-based searches on the local partition */
00549         VASSERT(rcol == thee->localPartID);
00550         /* Find the closest distance between this simplex and the
00551          * center of the local partition and check it against
00552          * (thee->localPartRadius*thee->killParam) */
00553         dist = 0;
00554         for (ivert=0; ivert<SS_dimVV(sm); ivert++) {
00555             dx = VV_coord(SS_vertex(sm, ivert), 0) -
00556                 thee->localPartCenter[0];
00557             dy = VV_coord(SS_vertex(sm, ivert), 1) -
00558                 thee->localPartCenter[1];
00559             dz = VV_coord(SS_vertex(sm, ivert), 2) -
00560                 thee->localPartCenter[2];
00561             dist = VSQRT((dx*dx + dy*dy + dz*dz));
00562         }
00563         if (dist < thee->localPartRadius*thee->killParam) return 1;
00564     }
00565     } else if (thee->killFlag == 3) VASSERT(0);
00566     else VASSERT(0);
00567
00568     return 0;
00569 }
00570
00571 }

```

10.13 src/fem/vpee.h File Reference

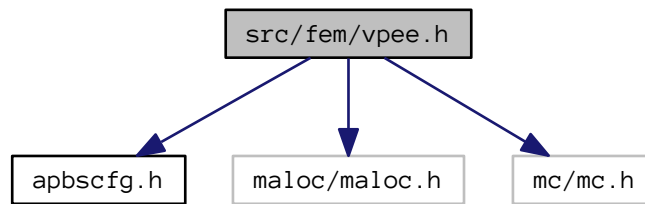
Contains declarations for class Vpee.

```

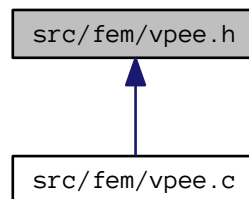
#include "apbscfg.h"
#include "malloc/malloc.h"
#include "mc/mc.h"

```

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)
FORTTRAN stub to construct the Vpee object.

- VEXTERNC void `Vpee_dtor` (`Vpee **thee`)
Object destructor.
- VEXTERNC void `Vpee_dtor2` (`Vpee *thee`)
FORTTRAN 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.

10.13.1 Detailed Description

Contains declarations for class `Vpee`.

Version

\$Id\$

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
```

```

* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vpee.h](#).

10.14 vpee.h

```

00001
00076 #ifndef _VPEE_H
00077 #define _VPEE_H
00078
00079 #include "apbscfg.h"
00080
00081 #include "malloc/malloc.h"
00082 #include "mc/mc.h"
00083
00089 struct sVpee {
00090
00091     Gem *gm;
00092     int localPartID;
00095     double localPartCenter[3];
00097     double localPartRadius;
00099     int killFlag;
00102     double killParam;
00104     Vmem *mem;
00106 };
00107
00112 typedef struct sVpee Vpee;
00113
00114 /* ////////////////////////////////////////
00115 // Class Vpee Inlineable methods
00117
00118 #if !defined(VINLINE_VPEE)
00119 #else /* if defined(VINLINE_VPEE) */
00120 #endif /* if !defined(VINLINE_VPEE) */
00121
00122 /* ////////////////////////////////////////
00123 // Class Vpee: Non-Inlineable methods (vpee.c)
00125
00132 VEXTERNC Vpee* Vpee_ctor(
00133     Gem *gm,
00134     int localPartID,
00135     int killFlag,
00146     double killParam
00147 );
00148
00155 VEXTERNC int Vpee_ctor2(
00156     Vpee *thee,
00157     Gem *gm,
00158     int localPartID,
00159     int killFlag,
00170     double killParam
00171 );
00172
00177 VEXTERNC void Vpee_dtor(
00178     Vpee **thee
00179 );
00180
00185 VEXTERNC void Vpee_dtor2(
00186     Vpee *thee
00187 );
00188
00204 VEXTERNC int Vpee_markRefine(
00205     Vpee *thee,
00206     AM *am,
00207     int level,
00208     int akey,

```

```

00216         int rcol,
00219         double etol,
00220         int bkey
00224     );
00225
00231 VEXTERNC int Vpee_numSS(
00232     Vpee *thee
00233 );
00234
00235 #endif      /* ifndef _VPEE_H_ */

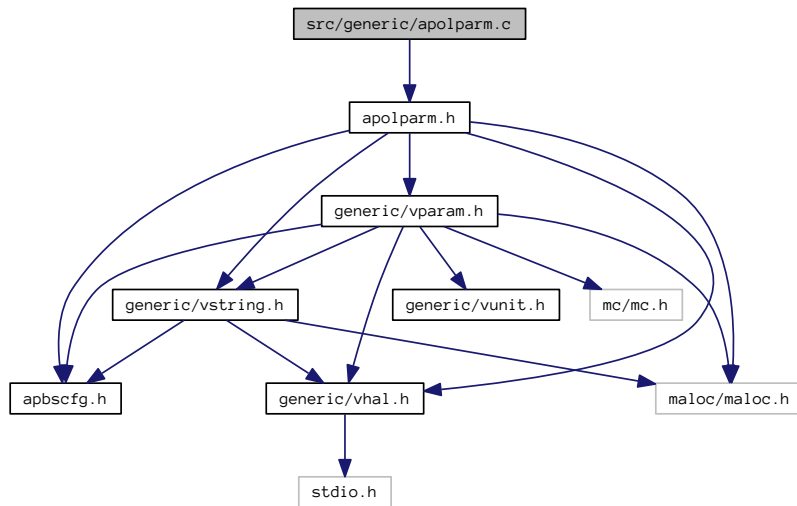
```

10.15 src/generic/apolparm.c File Reference

Class APOLparm methods.

```
#include "apolparm.h"
```

Include dependency graph for apolparm.c:



Functions

- VPUBLIC [APOLparm](#) * [APOLparm_ctor](#) ()
Construct APOLparm.
- VPUBLIC Vrc_Codes [APOLparm_ctor2](#) ([APOLparm](#) *thee)
FORTTRAN 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)
FORTTRAN 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.

10.15.1 Detailed Description

Class APOLparm methods.

Author

David Gohara

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
```

```

* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [apolparm.c](#).

10.16 apolparm.c

```

00001
00057 #include "apolparm.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_MGPARM)
00062
00063 #endif /* if !defined(VINLINE_MGPARM) */
00064
00065 VPUBLIC APOLparm* APOLparm_ctor() {
00066
00067     /* Set up the structure */
00068     APOLparm *thee = VNULL;
00069     thee = (APOLparm*)Vmem_malloc(VNULL, 1, sizeof(APOLparm));
00070     VASSERT( thee != VNULL);
00071     VASSERT( APOLparm_ctor2(thee) == VRC_SUCCESS );
00072
00073     return thee;
00074 }
00075
00076 VPUBLIC Vrc_Codes APOLparm_ctor2(APOLparm *thee) {
00077
00078     int i;
00079
00080     if (thee == VNULL) return VRC_FAILURE;
00081
00082     thee->parsed = 0;
00083
00084     thee->setgrid = 0;
00085     thee->setmolid = 0;
00086     thee->setbconc = 0;
00087     thee->setsdens = 0;
00088     thee->setdpos = 0;
00089     thee->setpress = 0;
00090     thee->setsrfm = 0;
00091     thee->setsrad = 0;
00092     thee->setswin = 0;
00093
00094     thee->settemp = 0;
00095     thee->setgamma = 0;
00096
00097     thee->setwat = 0;
00098
00099     thee->sav = 0.0;
00100     thee->sasa = 0.0;
00101     thee->wcaEnergy = 0.0;
00102

```



```

00103     for(i=0;i<3;i++) thee->totForce[i] = 0.0;
00104
00105     return VRC_SUCCESS;
00106 }
00107
00108 VPUBLIC void APOLparm_copy(
00109     APOLparm *thee,
00110     APOLparm *source
00111 ) {
00112
00113     int i;
00114
00115     thee->parsed = source->parsed;
00116
00117     for (i=0; i<3; i++) thee->grid[i] = source->grid[i];
00118     thee->setgrid = source->setgrid;
00119
00120     thee->molid = source->molid;
00121     thee->setmolid = source->setmolid;
00122
00123     thee->bconc = source->bconc ;
00124     thee->setbconc= source->setbconc ;
00125
00126     thee->sdens = source->sdens ;
00127     thee->setsdens= source->setsdens ;
00128
00129     thee->dpos = source->dpos ;
00130     thee->setdpos= source->setdpos ;
00131
00132     thee->press = source->press ;
00133     thee->setpress = source->setpress ;
00134
00135     thee->srfm = source->srfm ;
00136     thee->setsrfm = source->setsrfm ;
00137
00138     thee->srad = source->srad ;
00139     thee->setsrad = source->setsrad ;
00140
00141     thee->swin = source->swin ;
00142     thee->setswin = source->setswin ;
00143
00144     thee->temp = source->temp ;
00145     thee->settemp = source->settemp ;
00146
00147     thee->gamma = source->gamma ;
00148     thee->setgamma = source->setgamma ;
00149
00150     thee->calcenergy = source->calcenergy ;
00151     thee->setcalcenergy = source->setcalcenergy ;
00152
00153     thee->calcforce = source->calcforce ;
00154     thee->setcalcforce = source->setcalcforce ;
00155
00156     thee->setwat = source->setwat ;
00157
00158     thee->sav = source->sav;
00159     thee->sasa = source->sasa;
00160     thee->wcaEnergy = source->wcaEnergy;
00161
00162     for(i=0;i<3;i++) thee->totForce[i] = source->totForce[i];
00163
00164     return;
00165 }
00166
00167 VPUBLIC void APOLparm_dtor(APOLparm **thee) {
00168     if ((*thee) != VNULL) {
00169         APOLparm_dtor2(*thee);
00170         Vmem_free(VNULL, 1, sizeof(APOLparm), (void **)thee);
00171         (*thee) = VNULL;
00172     }
00173
00174     return;
00175 }
00176
00177 VPUBLIC void APOLparm_dtor2(APOLparm *thee) { ; }
00178
00179 VPUBLIC Vrc_Codes APOLparm_check(APOLparm *thee) {
00180
00181     Vrc_Codes rc;
00182     rc = VRC_SUCCESS;
00183

```

```

00184
00185     if (!thee->parsed) {
00186         Vnm_print(2, "APOLparm_check:  not filled!\n");
00187         return VRC_FAILURE;
00188     }
00189     if (!thee->setgrid) {
00190         Vnm_print(2, "APOLparm_check:  grid not set!\n");
00191         rc = VRC_FAILURE;
00192     }
00193     if (!thee->setmolid) {
00194         Vnm_print(2, "APOLparm_check:  molid not set!\n");
00195         rc = VRC_FAILURE;
00196     }
00197     if (!thee->setbconc) {
00198         Vnm_print(2, "APOLparm_check:  bconc not set!\n");
00199         rc = VRC_FAILURE;
00200     }
00201     if (!thee->setsdens) {
00202         Vnm_print(2, "APOLparm_check:  sdens not set!\n");
00203         rc = VRC_FAILURE;
00204     }
00205     if (!thee->setdpos) {
00206         Vnm_print(2, "APOLparm_check:  dpos not set!\n");
00207         rc = VRC_FAILURE;
00208     }
00209     if (!thee->setpress) {
00210         Vnm_print(2, "APOLparm_check:  press not set!\n");
00211         rc = VRC_FAILURE;
00212     }
00213     if (!thee->setsrfm) {
00214         Vnm_print(2, "APOLparm_check:  srfm not set!\n");
00215         rc = VRC_FAILURE;
00216     }
00217     if (!thee->setsrad) {
00218         Vnm_print(2, "APOLparm_check:  srad not set!\n");
00219         rc = VRC_FAILURE;
00220     }
00221     if (!thee->setswin) {
00222         Vnm_print(2, "APOLparm_check:  swin not set!\n");
00223         rc = VRC_FAILURE;
00224     }
00225     if (!thee->settemp) {
00226         Vnm_print(2, "APOLparm_check:  temp not set!\n");
00227         rc = VRC_FAILURE;
00228     }
00229     if (!thee->setgamma) {
00230         Vnm_print(2, "APOLparm_check:  gamma not set!\n");
00231         rc = VRC_FAILURE;
00232     }
00233     return rc;
00234 }
00235
00236
00237 VPRIVATE Vrc_Codes APOLparm_parseGRID(APOLparm *thee, Vio *sock) {
00238
00239     char tok[VMAX_BUFSIZE];
00240     double tf;
00241
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[0] = 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[1] = tf;
00254     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00255     if (sscanf(tok, "%lf", &tf) == 0) {
00256         Vnm_print(2, "Nosh:  Read non-float (%s) while parsing GRID \
00257 keyword!\n", tok);
00258         return VRC_WARNING;
00259     } else thee->grid[2] = tf;
00260     thee->setgrid = 1;
00261     return VRC_SUCCESS;
00262
00263 ERROR1:
00264     Vnm_print(2, "parseAPOL:  ran out of tokens!\n");

```

```

00265     return VRC_WARNING;
00266 }
00267
00268 VPRIVATE Vrc_Codes APOLparm_parseMOL(APOLparm *thee, Vio *sock) {
00269     int ti;
00270     char tok[VMAX_BUFSIZE];
00271
00272     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00273     if (sscanf(tok, "%d", &ti) == 0) {
00274         Vnm_print(2, "Nosh: Read non-int (%s) while parsing MOL \
00275 keyword!\n", tok);
00276         return VRC_WARNING;
00277     }
00278     thee->molid = ti;
00279     thee->setmolid = 1;
00280     return VRC_SUCCESS;
00281
00282 ERROR1:
00283     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00284     return VRC_WARNING;
00285 }
00286
00287 VPRIVATE Vrc_Codes APOLparm_parseSRFM(APOLparm *thee, Vio *sock) {
00288     char tok[VMAX_BUFSIZE];
00289
00290     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00291
00292     if (Vstring_strcasecmp(tok, "sacc") == 0) {
00293         thee->srfm = VSM_MOL;
00294         thee->setsrfm = 1;
00295         return VRC_SUCCESS;
00296     } else {
00297         Vnm_print(2, "parseAPOL: Unrecongized keyword (%s) when parsing srfm!\n", tok);
00298         Vnm_print(2, "parseAPOL: Accepted values for srfm = sacc\n");
00299         return VRC_WARNING;
00300     }
00301
00302     return VRC_FAILURE;
00303
00304 ERROR1:
00305     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00306     return VRC_WARNING;
00307 }
00308
00309 VPRIVATE Vrc_Codes APOLparm_parseSRAD(APOLparm *thee, Vio *sock) {
00310     char tok[VMAX_BUFSIZE];
00311     double tf;
00312
00313     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00314     if (sscanf(tok, "%lf", &tf) == 0) {
00315         Vnm_print(2, "Nosh: Read non-float (%s) while parsing SRAD \
00316 keyword!\n", tok);
00317         return VRC_WARNING;
00318     }
00319     thee->srad = tf;
00320     thee->setsrad = 1;
00321     return VRC_SUCCESS;
00322
00323 ERROR1:
00324     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00325     return VRC_WARNING;
00326 }
00327
00328 VPRIVATE Vrc_Codes APOLparm_parseSWIN(APOLparm *thee, Vio *sock) {
00329     char tok[VMAX_BUFSIZE];
00330     double tf;
00331
00332     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00333     if (sscanf(tok, "%lf", &tf) == 0) {
00334         Vnm_print(2, "Nosh: Read non-float (%s) while parsing SWIN \
00335 keyword!\n", tok);
00336         return VRC_WARNING;
00337     }
00338     thee->swin = tf;
00339     thee->setswin = 1;
00340     return VRC_SUCCESS;
00341
00342 ERROR1:
00343     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00344     return VRC_WARNING;
00345 }

```

```

00346
00347 VPRIVATE Vrc_Codes APOLparm_parseTEMP(APOLparm *thee, Vio *sock) {
00348     char tok[VMAX_BUFSIZE];
00349     double tf;
00350
00351     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00352     if (sscanf(tok, "%lf", &tf) == 0) {
00353         Vnm_print(2, "Nosh: Read non-float (%s) while parsing TEMP \
00354 keyword!\n", tok);
00355         return VRC_WARNING;
00356     }
00357     thee->temp = tf;
00358     thee->settemp = 1;
00359     return VRC_SUCCESS;
00360
00361 ERROR1:
00362     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00363     return VRC_WARNING;
00364 }
00365
00366 VPRIVATE Vrc_Codes APOLparm_parseGAMMA(APOLparm *thee, Vio *sock) {
00367     char tok[VMAX_BUFSIZE];
00368     double tf;
00369
00370     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00371     if (sscanf(tok, "%lf", &tf) == 0) {
00372         Vnm_print(2, "Nosh: Read non-float (%s) while parsing GAMMA \
00373 keyword!\n", tok);
00374         return VRC_WARNING;
00375     }
00376     thee->gamma = tf;
00377     thee->setgamma = 1;
00378     return VRC_SUCCESS;
00379
00380 ERROR1:
00381     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00382     return VRC_WARNING;
00383 }
00384
00385 VPRIVATE Vrc_Codes APOLparm_parseCALCENERGY(APOLparm *thee, Vio *sock) {
00386     char tok[VMAX_BUFSIZE];
00387     int ti;
00388
00389     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00390     /* Parse number */
00391     if (sscanf(tok, "%d", &ti) == 1) {
00392         thee->calcenergy = (APOLparm_calcEnergy)ti;
00393         thee->setcalcenergy = 1;
00394
00395         Vnm_print(2, "parseAPOL: Warning -- parsed deprecated \"calcenergy \
00396 %d\" statement.\n", ti);
00397         Vnm_print(2, "parseAPOL: Please use \"calcenergy \");
00398         switch (thee->calcenergy) {
00399             case ACE_NO:
00400                 Vnm_print(2, "no");
00401                 break;
00402             case ACE_TOTAL:
00403                 Vnm_print(2, "total");
00404                 break;
00405             case ACE_COMPS:
00406                 Vnm_print(2, "comps");
00407                 break;
00408             default:
00409                 Vnm_print(2, "UNKNOWN");
00410                 break;
00411         }
00412         Vnm_print(2, "\" instead.\n");
00413         return VRC_SUCCESS;
00414     } else if (Vstring_strcasecmp(tok, "no") == 0) {
00415         thee->calcenergy = ACE_NO;
00416         thee->setcalcenergy = 1;
00417         return VRC_SUCCESS;
00418     } else if (Vstring_strcasecmp(tok, "total") == 0) {
00419         thee->calcenergy = ACE_TOTAL;
00420         thee->setcalcenergy = 1;
00421         return VRC_SUCCESS;
00422     } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00423         thee->calcenergy = ACE_COMPS;
00424         thee->setcalcenergy = 1;
00425         return VRC_SUCCESS;
00426     } else {

```

```

00427         Vnm_print(2, "Nosh: Unrecognized parameter (%s) while parsing \
00428 calcenergy!\n", tok);
00429         return VRC_WARNING;
00430     }
00431     return VRC_FAILURE;
00432
00433 ERROR1:
00434     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00435     return VRC_WARNING;
00436 }
00437
00438 VPRIVATE Vrc_Codes APOLparm_parseCALCFORCE(APOLparm *thee, Vio *sock) {
00439     char tok[VMAX_BUFSIZE];
00440     int ti;
00441
00442     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00443     /* Parse number */
00444     if (sscanf(tok, "%d", &ti) == 1) {
00445         thee->calcforce = (APOLparm_calcForce)ti;
00446         thee->setcalcforce = 1;
00447
00448         Vnm_print(2, "parseAPOL: Warning -- parsed deprecated \"calcforce \
00449 %d\" statement.\n", ti);
00450         Vnm_print(2, "parseAPOL: Please use \"calcforce ");
00451         switch (thee->calcenergy) {
00452             case ACF_NO:
00453                 Vnm_print(2, "no");
00454                 break;
00455             case ACF_TOTAL:
00456                 Vnm_print(2, "total");
00457                 break;
00458             case ACF_COMPS:
00459                 Vnm_print(2, "comps");
00460                 break;
00461             default:
00462                 Vnm_print(2, "UNKNOWN");
00463                 break;
00464         }
00465         Vnm_print(2, "\" instead.\n");
00466         return VRC_SUCCESS;
00467     } else if (Vstring_strcasecmp(tok, "no") == 0) {
00468         thee->calcforce = ACF_NO;
00469         thee->setcalcforce = 1;
00470         return VRC_SUCCESS;
00471     } else if (Vstring_strcasecmp(tok, "total") == 0) {
00472         thee->calcforce = ACF_TOTAL;
00473         thee->setcalcforce = 1;
00474         return VRC_SUCCESS;
00475     } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00476         thee->calcforce = ACF_COMPS;
00477         thee->setcalcforce = 1;
00478         return VRC_SUCCESS;
00479     } else {
00480         Vnm_print(2, "Nosh: Unrecognized parameter (%s) while parsing \
00481 calcforce!\n", tok);
00482         return VRC_WARNING;
00483     }
00484     return VRC_FAILURE;
00485
00486 ERROR1:
00487     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00488     return VRC_WARNING;
00489 }
00490
00491 VPRIVATE Vrc_Codes APOLparm_parseBCONC(APOLparm *thee, Vio *sock) {
00492     char tok[VMAX_BUFSIZE];
00493     double tf;
00494
00495     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00496     if (sscanf(tok, "%lf", &tf) == 0) {
00497         Vnm_print(2, "Nosh: Read non-float (%s) while parsing BCONC \
00498 keyword!\n", tok);
00499         return VRC_WARNING;
00500     }
00501     thee->bconc = tf;
00502     thee->setbconc = 1;
00503     return VRC_SUCCESS;
00504
00505 ERROR1:
00506     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00507     return VRC_WARNING;

```

```

00508 }
00509
00510 VPRIVATE Vrc_Codes APOLparm_parseSDENS(APOLparm *thee, Vio *sock) {
00511     char tok[VMAX_BUFSIZE];
00512     double tf;
00513
00514     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00515     if (sscanf(tok, "%lf", &tf) == 0) {
00516         Vnm_print(2, "Nosh: Read non-float (%s) while parsing SDENS \
00517 keyword!\n", tok);
00518         return VRC_WARNING;
00519     }
00520     thee->sdens = tf;
00521     thee->setsdens = 1;
00522     return VRC_SUCCESS;
00523
00524 ERROR1:
00525     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00526     return VRC_WARNING;
00527 }
00528
00529 VPRIVATE Vrc_Codes APOLparm_parseDPOS(APOLparm *thee, Vio *sock) {
00530     char tok[VMAX_BUFSIZE];
00531     double tf;
00532
00533     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00534     if (sscanf(tok, "%lf", &tf) == 0) {
00535         Vnm_print(2, "Nosh: Read non-float (%s) while parsing SDENS \
00536 keyword!\n", tok);
00537         return VRC_WARNING;
00538     }
00539     thee->dpos = tf;
00540     thee->setdpos = 1;
00541
00542     if (thee->dpos < 0.001) {
00543         Vnm_print(1, "\nWARNING WARNING WARNING WARNING\n");
00544         Vnm_print(1, "Nosh: dpos is set to a very small value.\n");
00545         Vnm_print(1, "Nosh: If you are not using a PQR file, you can \
00546 safely ignore this message.\n");
00547         Vnm_print(1, "Nosh: Otherwise please choose a value greater than \
00548 or equal to 0.001.\n\n");
00549     }
00550
00551     return VRC_SUCCESS;
00552
00553 ERROR1:
00554     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00555     return VRC_WARNING;
00556 }
00557
00558 VPRIVATE Vrc_Codes APOLparm_parsePRESS(APOLparm *thee, Vio *sock) {
00559     char tok[VMAX_BUFSIZE];
00560     double tf;
00561
00562     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00563     if (sscanf(tok, "%lf", &tf) == 0) {
00564         Vnm_print(2, "Nosh: Read non-float (%s) while parsing PRESS \
00565 keyword!\n", tok);
00566         return VRC_WARNING;
00567     }
00568     thee->press = tf;
00569     thee->setpress = 1;
00570     return VRC_SUCCESS;
00571
00572 ERROR1:
00573     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00574     return VRC_WARNING;
00575 }
00576
00577 VPUBLIC Vrc_Codes APOLparm_parseToken(APOLparm *thee, char tok[VMAX_BUFSIZE],
00578     Vio *sock) {
00579
00580     if (thee == VNULL) {
00581         Vnm_print(2, "parseAPOL: got NULL thee!\n");
00582         return VRC_WARNING;
00583     }
00584
00585     if (sock == VNULL) {
00586         Vnm_print(2, "parseAPOL: got NULL socket!\n");
00587         return VRC_WARNING;
00588     }

```

```

00589
00590     if (Vstring_strcasecmp(tok, "mol") == 0) {
00591         return APOLparm_parseMOL(thee, sock);
00592     } else if (Vstring_strcasecmp(tok, "grid") == 0) {
00593         return APOLparm_parseGRID(thee, sock);
00594     } else if (Vstring_strcasecmp(tok, "dime") == 0) {
00595         Vnm_print(2, "APOLparm_parseToken: The DIME and GLEN keywords for APOLAR have been replaced with
GRID.\n");
00596         Vnm_print(2, "APOLparm_parseToken: Please see the APBS User Guide for more information.\n");
00597         return VRC_WARNING;
00598     } else if (Vstring_strcasecmp(tok, "glen") == 0) {
00599         Vnm_print(2, "APOLparm_parseToken: The DIME and GLEN keywords for APOLAR have been replaced with
GRID.\n");
00600         Vnm_print(2, "APOLparm_parseToken: Please see the APBS User Guide for more information.\n");
00601         return VRC_WARNING;
00602     } else if (Vstring_strcasecmp(tok, "bconc") == 0) {
00603         return APOLparm_parseBCONC(thee, sock);
00604     } else if (Vstring_strcasecmp(tok, "sdens") == 0) {
00605         return APOLparm_parseSDENS(thee, sock);
00606     } else if (Vstring_strcasecmp(tok, "dpos") == 0) {
00607         return APOLparm_parseDPOS(thee, sock);
00608     } else if (Vstring_strcasecmp(tok, "srfm") == 0) {
00609         return APOLparm_parseSRFM(thee, sock);
00610     } else if (Vstring_strcasecmp(tok, "srad") == 0) {
00611         return APOLparm_parseSRAD(thee, sock);
00612     } else if (Vstring_strcasecmp(tok, "swin") == 0) {
00613         return APOLparm_parseSWIN(thee, sock);
00614     } else if (Vstring_strcasecmp(tok, "temp") == 0) {
00615         return APOLparm_parseTEMP(thee, sock);
00616     } else if (Vstring_strcasecmp(tok, "gamma") == 0) {
00617         return APOLparm_parseGAMMA(thee, sock);
00618     } else if (Vstring_strcasecmp(tok, "press") == 0) {
00619         return APOLparm_parsePRESS(thee, sock);
00620     } else if (Vstring_strcasecmp(tok, "calcenergy") == 0) {
00621         return APOLparm_parseCALCENERGY(thee, sock);
00622     } else if (Vstring_strcasecmp(tok, "calcforce") == 0) {
00623         return APOLparm_parseCALCFORCE(thee, sock);
00624     } //else {
00625         Vnm_print(2, "parseAPOL: Unrecognized keyword (%s)!\n", tok);
00626         return VRC_WARNING;
00627     }
00628
00629
00630 //     return VRC_FAILURE;
00631
00632 }

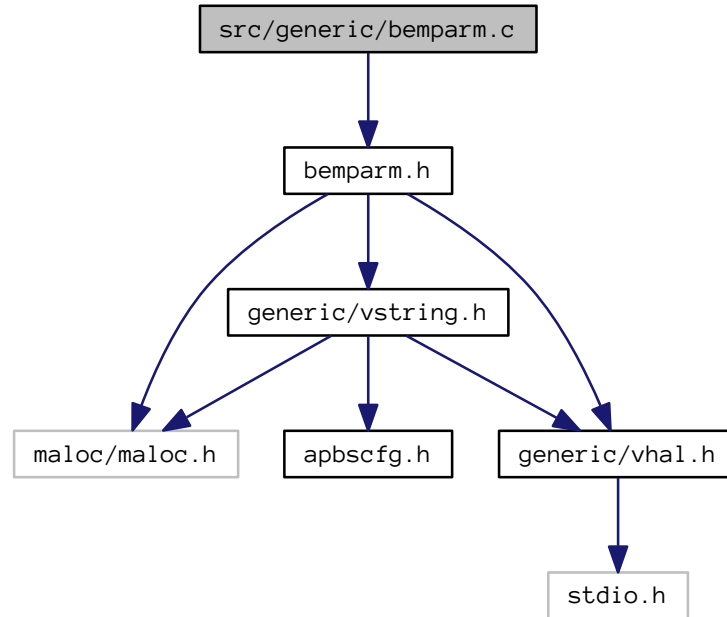
```

10.17 src/generic/bemparm.c File Reference

Class BEMparm methods.

```
#include "bemparm.h"
```

Include dependency graph for bemparm.c:



Functions

- `VPUBLIC BEMparm * BEMparm_ctor (BEMparm_CalcType type)`
Construct BEMparm object.
- `VPUBLIC Vrc_Codes BEMparm_ctor2 (BEMparm *thee, BEMparm_CalcType type)`
FORTTRAN stub to construct BEMparm object.
- `VPUBLIC void BEMparm_dtor (BEMparm **thee)`
Object destructor.
- `VPUBLIC void BEMparm_dtor2 (BEMparm *thee)`
FORTTRAN stub for object destructor.
- `VPUBLIC Vrc_Codes BEMparm_check (BEMparm *thee)`
Consistency check for parameter values stored in object.
- `VPUBLIC void BEMparm_copy (BEMparm *thee, BEMparm *parm)`
- `VPRIVATE Vrc_Codes BEMparm_parseTREE_ORDER (BEMparm *thee, Vio *sock)`
- `VPRIVATE Vrc_Codes BEMparm_parseTREE_N0 (BEMparm *thee, Vio *sock)`
- `VPRIVATE Vrc_Codes BEMparm_parseMAC (BEMparm *thee, Vio *sock)`
- `VPUBLIC Vrc_Codes BEMparm_parseToken (BEMparm *thee, char tok[VMAX_BUFSIZE], Vio *sock)`
Parse an MG keyword from an input file.

10.17.1 Detailed Description

Class BEMparm methods.

Author

Nathan A. Baker, Weihua Geng, and Andrew J. Stevens

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [bemparm.c](#).

10.18 bemparm.c

```

00001
00057 #include "bemparm.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_MGPARM)
00062
00063 #endif /* if !defined(VINLINE_MGPARM) */
00064
00065
00066 VPUBLIC BEMparm* BEMparm_ctor(BEMparm_CalcType type) {
00067
00068     /* Set up the structure */
00069     BEMparm *thee = VNULL;
00070     thee = (BEMparm*)Vmem_malloc(VNULL, 1, sizeof(BEMparm));
00071     VASSERT( thee != VNULL);
00072     VASSERT( BEMparm_ctor2(thee, type) == VRC_SUCCESS );
00073
00074     return thee;
00075 }
00076
00077 VPUBLIC Vrc_Codes BEMparm_ctor2(BEMparm *thee,
00078     BEMparm_CalcType type) {
00079
00080     int i;
00081
00082     if (thee == VNULL) return VRC_FAILURE;
00083
00084     thee->parsed = 0;
00085     thee->type = type;
00086
00087     /* *** GENERIC PARAMETERS *** */
00088
00089     /* *** TYPE 0 PARAMETERS *** */
00090     thee->tree_order = 1;
00091     thee->settree_order = 0;
00092     thee->tree_n0 = 500;
00093     thee->settree_n0 = 0;
00094     thee->mac = 0.8;
00095     thee->setmac = 0;
00096
00097     /* *** TYPE 1 & 2 PARAMETERS *** */
00098
00099     /* *** TYPE 2 PARAMETERS *** */
00100     thee->nonlintype = 0;
00101     thee->setnonlintype = 0;
00102
00103     /* *** Default parameters for TINKER *** */
00104     thee->chgs = VCM_CHARGE;
00105
00106     return VRC_SUCCESS;
00107 }
00108
00109 VPUBLIC void BEMparm_dtor(BEMparm **thee) {
00110     if ((*thee) != VNULL) {
00111         BEMparm_dtor2(*thee);
00112         Vmem_free(VNULL, 1, sizeof(BEMparm), (void **)thee);
00113         (*thee) = VNULL;
00114     }
00115 }
00116
00117 VPUBLIC void BEMparm_dtor2(BEMparm *thee) { ; }
00118
00119 VPUBLIC Vrc_Codes BEMparm_check(BEMparm *thee) {
00120
00121     Vrc_Codes rc;
00122     int i, tdime[3], ti, tnlev[3], nlev;
00123
00124     rc = VRC_SUCCESS;
00125
00126     Vnm_print(0, "BEMparm_check:  checking BEMparm object of type %d.\n",
00127         thee->type);
00128
00129     /* Check to see if we were even filled... */
00130     if (!thee->parsed) {
00131         Vnm_print(2, "BEMparm_check:  not filled!\n");
00132         return VRC_FAILURE;
00133     }

```

```

00133
00134
00135     /* Check type settings */
00136     if ((thee->type != BCT_MANUAL) && (thee->type != BCT_NONE)) {
00137         Vnm_print(2, "BEMparm_check: type not set");
00138         rc = VRC_FAILURE;
00139     }
00140
00141     /* Check treecode setting */
00142     if (thee->tree_order < 1) {
00143         Vnm_print(2, "BEMparm_check: treecode order is less than 1");
00144         rc = VRC_FAILURE;
00145     }
00146     if (thee->tree_n0 < 1) {
00147         Vnm_print(2, "BEMparm_check: treecode leaf size is less than 1");
00148         rc = VRC_FAILURE;
00149     }
00150     if (thee->mac > 1 || thee->mac <= 0) {
00151         Vnm_print(2, "BEMparm_check: MAC criterion fails");
00152         rc = VRC_FAILURE;
00153     }
00154     return rc;
00155 }
00156
00157 VPUBLIC void BEMparm_copy(BEMparm *thee, BEMparm *parm) {
00158     int i;
00159
00160     VASSERT(thee != VNULL);
00161     VASSERT(parm != VNULL);
00162
00163     thee->type = parm->type;
00164     thee->parsed = parm->parsed;
00165
00166     /* *** GENERIC PARAMETERS *** */
00167
00168     /* *** TYPE 0 PARMS *** */
00169
00170     thee->tree_order = parm->tree_order;
00171     thee->settree_order = parm->settree_order;
00172     thee->tree_n0 = parm->tree_n0;
00173     thee->settree_n0 = parm->settree_n0;
00174     thee->mac = parm->mac;
00175     thee->setmac = parm->setmac;
00176
00177     /* *** TYPE 1 & 2 PARMS *** */
00178
00179     /* *** TYPE 2 PARMS *** */
00180     thee->nonlotype = parm->nonlotype;
00181     thee->setnonlotype = parm->setnonlotype;
00182 }
00183
00184 VPRIVATE Vrc_Codes BEMparm_parseTREE_ORDER(BEMparm *thee, Vio *sock) {
00185     char tok[VMAX_BUFSIZE];
00186     int ti;
00187
00188     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00189     if (sscanf(tok, "%d", &ti) == 0) {
00190         Vnm_print(2, "Nosh: Read non-integer (%s) while parsing TREE_ORDER \
00191 keyword!\n", tok);
00192         return VRC_WARNING;
00193     } else if (ti <= 0) {
00194         Vnm_print(2, "parseBEM: tree_order must be greater than 0!\n");
00195         return VRC_WARNING;
00196     } else thee->tree_order = ti;
00197     thee->settree_order = 1;
00198     return VRC_SUCCESS;
00199 }
00200
00201 VERROR1:
00202     Vnm_print(2, "parseBEM: ran out of tokens!\n");
00203     return VRC_WARNING;
00204 }
00205
00206 VPRIVATE Vrc_Codes BEMparm_parseTREE_N0(BEMparm *thee, Vio *sock) {
00207     char tok[VMAX_BUFSIZE];

```

```

00214     int ti;
00215
00216     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00217     if (sscanf(tok, "%d", &ti) == 0) {
00218         Vnm_print(2, "Nosh: Read non-integer (%s) while parsing TREE_N0 \
00219 keyword!\n", tok);
00220         return VRC_WARNING;
00221     } else if (ti <= 0) {
00222         Vnm_print(2, "parseBEM: tree_n0 must be greater than 0!\n");
00223         return VRC_WARNING;
00224     } else thee->tree_n0 = ti;
00225     thee->settree_n0 = 1;
00226     return VRC_SUCCESS;
00227
00228     ERROR1:
00229     Vnm_print(2, "parseBEM: ran out of tokens!\n");
00230     return VRC_WARNING;
00231 }
00232
00233
00234 VPRIVATE Vrc_Codes BEMparm_parseMAC(BEMparm *thee, Vio *sock) {
00235     char tok[VMAX_BUFSIZE];
00236     double tf;
00237
00238     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00239     if (sscanf(tok, "%lf", &tf) == 0) {
00240         Vnm_print(2, "Nosh: Read non-float (%s) while parsing mac \
00241 keyword!\n", tok);
00242         return VRC_WARNING;
00243     } else if (tf <= 0.0 || tf > 1.0) {
00244         Vnm_print(2, "parseBEM: mac must be between 0 and 1!\n");
00245         return VRC_WARNING;
00246     } else thee->mac = tf;
00247     thee->setmac = 1;
00248     return VRC_SUCCESS;
00249
00250     ERROR1:
00251     Vnm_print(2, "parseBEM: ran out of tokens!\n");
00252     return VRC_WARNING;
00253 }
00254
00255
00256
00257 VPUBLIC Vrc_Codes BEMparm_parseToken(BEMparm *thee, char tok[VMAX_BUFSIZE],
00258     Vio *sock) {
00259     if (thee == VNULL) {
00260         Vnm_print(2, "parseBEM: got NULL thee!\n");
00261         return VRC_WARNING;
00262     }
00263     if (sock == VNULL) {
00264         Vnm_print(2, "parseBEM: got NULL socket!\n");
00265         return VRC_WARNING;
00266     }
00267
00268     Vnm_print(0, "BEMparm_parseToken: trying %s...\n", tok);
00269
00270
00271     if (Vstring_strcasecmp(tok, "tree_order") == 0) {
00272         return BEMparm_parseTREE_ORDER(thee, sock);
00273     } else if (Vstring_strcasecmp(tok, "tree_n0") == 0) {
00274         return BEMparm_parseTREE_N0(thee, sock);
00275     } else if (Vstring_strcasecmp(tok, "mac") == 0) {
00276         return BEMparm_parseMAC(thee, sock);
00277     } else {
00278         Vnm_print(2, "parseBEM: Unrecognized keyword (%s)!\n", tok);
00279         return VRC_WARNING;
00280     }
00281
00282     return VRC_FAILURE;
00283
00284 }
00285 }

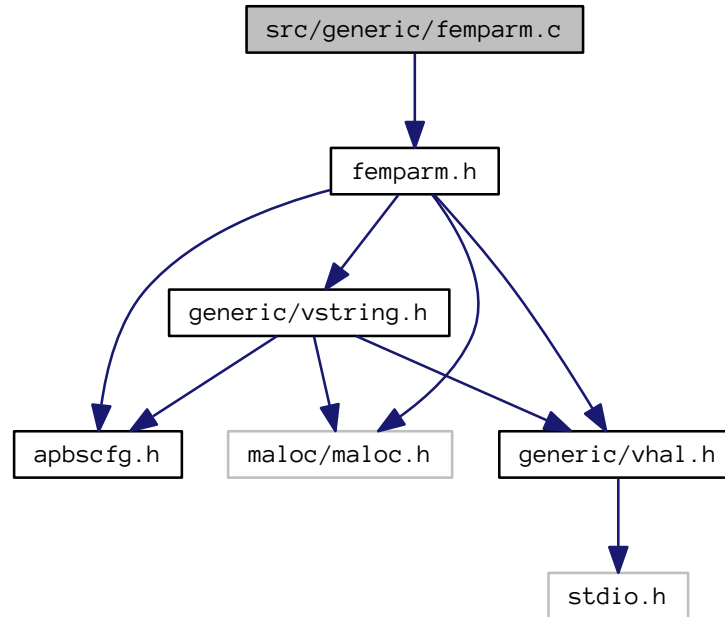
```

10.19 src/generic/femparm.c File Reference

Class FEMparm methods.

```
#include "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)
FORTTRAN 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)
FORTTRAN 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.

10.19.1 Detailed Description

Class FEMparm methods.

Author

Nathan Baker

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
```

```

* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [femparm.c](#).

10.20 femparm.c

```

00001
00057 #include "femparm.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_MGPARM)
00062
00063 #endif /* if !defined(VINLINE_MGPARM) */
00064
00065 VPUBLIC FEMparm* FEMparm_ctor(FEMparm_CalcType type) {
00066
00067     /* Set up the structure */
00068     FEMparm *thee = VNULL;
00069     thee = (FEMparm*)Vmem_malloc(VNULL, 1, sizeof(FEMparm));
00070     VASSERT( thee != VNULL);
00071     VASSERT( FEMparm_ctor2(thee, type) );
00072
00073     return thee;
00074 }
00075
00076 VPUBLIC int FEMparm_ctor2(FEMparm *thee,
00077                          FEMparm_CalcType type
00078                          ) {
00079
00080     if (thee == VNULL) return 0;
00081
00082     thee->parsed = 0;
00083     thee->type = type;
00084     thee->settype = 1;
00085
00086     thee->setglen = 0;
00087     thee->setetol = 0;
00088     thee->setekey = 0;
00089     thee->setakeyPRE = 0;
00090     thee->setakeySOLVE = 0;
00091     thee->settargetNum = 0;
00092     thee->settargetRes = 0;
00093     thee->setmaxsolve = 0;
00094     thee->setmaxvert = 0;
00095     thee->useMesh = 0;
00096
00097     return 1;
00098 }
00099
00100 VPUBLIC void FEMparm_copy(
00101                          FEMparm *thee,
00102                          FEMparm *source
00103                          ) {
00104
00105     int i;
00106
00107     thee->parsed = source->parsed;
00108     thee->type = source->type;
00109     thee->settype = source->settype;
00110     for (i=0; i<3; i++) thee->glen[i] = source->glen[i];
00111     thee->setglen = source->setglen;
00112     thee->etol = source->etol;
00113     thee->setetol = source->setetol;
00114     thee->ekey = source->ekey;
00115     thee->setekey = source->setekey;
00116     thee->akeyPRE = source->akeyPRE;
00117     thee->setakeyPRE = source->setakeyPRE;
00118     thee->akeySOLVE = source->akeySOLVE;
00119     thee->setakeySOLVE = source->setakeySOLVE;

```

```

00120     thee->targetNum = source->targetNum;
00121     thee->settargetNum = source->settargetNum;
00122     thee->targetRes = source->targetRes;
00123     thee->settargetRes = source->settargetRes;
00124     thee->maxsolve = source->maxsolve;
00125     thee->setmaxsolve = source->setmaxsolve;
00126     thee->maxvert = source->maxvert;
00127     thee->setmaxvert = source->setmaxvert;
00128     thee->pkey = source->pkey;
00129     thee->useMesh = source->useMesh;
00130     thee->meshID = source->meshID;
00131 }
00132
00133 VPUBLIC void FEMparm_dtor(FEMparm **thee) {
00134     if ((*thee) != VNULL) {
00135         FEMparm_dtor2(*thee);
00136         Vmem_free(VNULL, 1, sizeof(FEMparm), (void **)thee);
00137         (*thee) = VNULL;
00138     }
00139 }
00140
00141 VPUBLIC void FEMparm_dtor2(FEMparm *thee) { ; }
00142
00143 VPUBLIC int FEMparm_check(FEMparm *thee) {
00144     int rc;
00145     rc = 1;
00146
00147     if (!thee->parsed) {
00148         Vnm_print(2, "FEMparm_check: not filled!\n");
00149         return 0;
00150     }
00151     if (!thee->settype) {
00152         Vnm_print(2, "FEMparm_check: type not set!\n");
00153         rc = 0;
00154     }
00155     if (!thee->setglen) {
00156         Vnm_print(2, "FEMparm_check: glen not set!\n");
00157         rc = 0;
00158     }
00159     if (!thee->setetol) {
00160         Vnm_print(2, "FEMparm_check: etol not set!\n");
00161         rc = 0;
00162     }
00163     if (!thee->setekey) {
00164         Vnm_print(2, "FEMparm_check: ekey not set!\n");
00165         rc = 0;
00166     }
00167     if (!thee->setakeyPRE) {
00168         Vnm_print(2, "FEMparm_check: akeyPRE not set!\n");
00169         rc = 0;
00170     }
00171     if (!thee->setakeySOLVE) {
00172         Vnm_print(2, "FEMparm_check: akeySOLVE not set!\n");
00173         rc = 0;
00174     }
00175     if (!thee->settargetNum) {
00176         Vnm_print(2, "FEMparm_check: targetNum not set!\n");
00177         rc = 0;
00178     }
00179     if (!thee->settargetRes) {
00180         Vnm_print(2, "FEMparm_check: targetRes not set!\n");
00181         rc = 0;
00182     }
00183     if (!thee->setmaxsolve) {
00184         Vnm_print(2, "FEMparm_check: maxsolve not set!\n");
00185         rc = 0;
00186     }
00187     if (!thee->setmaxvert) {
00188         Vnm_print(2, "FEMparm_check: maxvert not set!\n");
00189         rc = 0;
00190     }
00191     return rc;
00192 }
00193
00194 }
00195
00196 VPRIVATE Vrc_Codes FEMparm_parseDOMAINLENGTH(FEMparm *thee,
00197                                             Vio *sock
00198                                             ) {
00199     int i;
00200

```



```

00201     double tf;
00202     char tok[VMAX_BUFSIZE];
00203
00204     for (i=0; i<3; i++) {
00205         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00206         if (sscanf(tok, "%lf", &tf) == 0) {
00207             Vnm_print(2, "parseFE: Read non-double (%s) while parsing \
00208 DOMAINLENGTH keyword!\n", tok);
00209             return VRC_FAILURE;
00210         }
00211         thee->glen[i] = tf;
00212     }
00213     thee->setglen = 1;
00214     return VRC_SUCCESS;
00215 VERROR1:
00216     Vnm_print(2, "parseFE: ran out of tokens!\n");
00217     return VRC_FAILURE;
00218
00219 }
00220
00221 VPRIVATE Vrc_Codes FEMparm_parseETOL(FEMparm *thee,
00222                                     Vio *sock
00223                                     ) {
00224
00225     double tf;
00226     char tok[VMAX_BUFSIZE];
00227
00228     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00229     if (sscanf(tok, "%lf", &tf) == 0) {
00230         Vnm_print(2, "parseFE: Read non-double (%s) while parsing \
00231 ETOL keyword!\n", tok);
00232         return VRC_FAILURE;
00233     }
00234     thee->etol = tf;
00235     thee->setetol = 1;
00236     return VRC_SUCCESS;
00237 VERROR1:
00238     Vnm_print(2, "parseFE: ran out of tokens!\n");
00239     return VRC_FAILURE;
00240
00241 }
00242
00243
00244 VPRIVATE Vrc_Codes FEMparm_parseEKEY(FEMparm *thee,
00245                                     Vio *sock
00246                                     ) {
00247
00248     char tok[VMAX_BUFSIZE];
00249     Vrc_Codes vrc = VRC_FAILURE;
00250
00251     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00252     if (Vstring_strcasecmp(tok, "simp") == 0) {
00253         thee->ekey = FET_SIMP;
00254         thee->setekey = 1;
00255         vrc = VRC_SUCCESS;
00256     } else if (Vstring_strcasecmp(tok, "glob") == 0) {
00257         thee->ekey = FET_GLOB;
00258         thee->setekey = 1;
00259         vrc = VRC_SUCCESS;
00260     } else if (Vstring_strcasecmp(tok, "frac") == 0) {
00261         thee->ekey = FET_FRAC;
00262         thee->setekey = 1;
00263         vrc = VRC_SUCCESS;
00264     } else {
00265         Vnm_print(2, "parseFE: undefined value (%s) for ekey!\n", tok);
00266         vrc = VRC_FAILURE;
00267     }
00268
00269     return vrc;
00270 VERROR1:
00271     Vnm_print(2, "parseFE: ran out of tokens!\n");
00272     return VRC_FAILURE;
00273
00274 }
00275
00276 VPRIVATE Vrc_Codes FEMparm_parseAKEYPRE(FEMparm *thee, Vio *sock) {
00277
00278     char tok[VMAX_BUFSIZE];
00279     Vrc_Codes vrc = VRC_FAILURE;
00280
00281     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);

```

```

00282     if (Vstring_strcasecmp(tok, "unif") == 0) {
00283         thee->akeyPRE = FRT_UNIF;
00284         thee->setakeyPRE = 1;
00285         vrc = VRC_SUCCESS;
00286     } else if (Vstring_strcasecmp(tok, "geom") == 0) {
00287         thee->akeyPRE = FRT_GEOM;
00288         thee->setakeyPRE = 1;
00289         vrc = VRC_SUCCESS;
00290     } else {
00291         Vnm_print(2, "parseFE: undefined value (%s) for akeyPRE!\n", tok);
00292         vrc = VRC_FAILURE;
00293     }
00294
00295     return vrc;
00296
00297 ERROR1:
00298     Vnm_print(2, "parseFE: ran out of tokens!\n");
00299     return VRC_FAILURE;
00300 }
00301
00302
00303 VPRIVATE Vrc_Codes FEMparm_parseAKEYSOLVE(FEMparm *thee, Vio *sock) {
00304
00305     char tok[VMAX_BUFSIZE];
00306     Vrc_Codes vrc = VRC_FAILURE;
00307
00308     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00309     if (Vstring_strcasecmp(tok, "resi") == 0) {
00310         thee->akeySOLVE = FRT_RESI;
00311         thee->setakeySOLVE = 1;
00312         vrc = VRC_SUCCESS;
00313     } else if (Vstring_strcasecmp(tok, "dual") == 0) {
00314         thee->akeySOLVE = FRT_DUAL;
00315         thee->setakeySOLVE = 1;
00316         vrc = VRC_SUCCESS;
00317     } else if (Vstring_strcasecmp(tok, "loca") == 0) {
00318         thee->akeySOLVE = FRT_LOCA;
00319         thee->setakeySOLVE = 1;
00320         vrc = VRC_SUCCESS;
00321     } else {
00322         Vnm_print(2, "parseFE: undefined value (%s) for akeyPRE!\n", tok);
00323         vrc = VRC_FAILURE;
00324     }
00325
00326     return vrc;
00327 ERROR1:
00328     Vnm_print(2, "parseFE: ran out of tokens!\n");
00329     return VRC_SUCCESS;
00330 }
00331
00332
00333 VPRIVATE Vrc_Codes FEMparm_parseTARGETNUM(FEMparm *thee, Vio *sock) {
00334
00335     char tok[VMAX_BUFSIZE];
00336     int ti;
00337
00338     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00339     if (sscanf(tok, "%d", &ti) == 0) {
00340         Vnm_print(2, "parseFE: read non-int (%s) for targetNum!\n", tok);
00341         return VRC_FAILURE;
00342     }
00343     thee->targetNum = ti;
00344     thee->settargetNum = 1;
00345     return VRC_SUCCESS;
00346 ERROR1:
00347     Vnm_print(2, "parseFE: ran out of tokens!\n");
00348     return VRC_FAILURE;
00349 }
00350
00351
00352 VPRIVATE Vrc_Codes FEMparm_parseTARGETRES(FEMparm *thee, Vio *sock) {
00353
00354     char tok[VMAX_BUFSIZE];
00355     double tf;
00356
00357     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00358     if (sscanf(tok, "%lf", &tf) == 0) {
00359         Vnm_print(2, "parseFE: read non-double (%s) for targetNum!\n",
00360             tok);
00361         return VRC_FAILURE;
00362     }

```

```

00363     thee->targetRes = tf;
00364     thee->settargetRes = 1;
00365     return VRC_SUCCESS;
00366 VERROR1:
00367     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00368     return VRC_FAILURE;
00369 }
00370 }
00371
00372 VPRIVATE Vrc_Codes FEMparm_parseMAXSOLVE(FEMparm *thee, Vio *sock) {
00373
00374     char tok[VMAX_BUFSIZE];
00375     int ti;
00376
00377     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00378     if (sscanf(tok, "%d", &ti) == 0) {
00379         Vnm_print(2, "parseFE:  read non-int (%s) for maxsolve!\n", tok);
00380         return VRC_FAILURE;
00381     }
00382     thee->maxsolve = ti;
00383     thee->setmaxsolve = 1;
00384     return VRC_SUCCESS;
00385 VERROR1:
00386     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00387     return VRC_FAILURE;
00388 }
00389 }
00390
00391 VPRIVATE Vrc_Codes FEMparm_parseMAXVERT(FEMparm *thee, Vio *sock) {
00392
00393     char tok[VMAX_BUFSIZE];
00394     int ti;
00395
00396     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00397     if (sscanf(tok, "%d", &ti) == 0) {
00398         Vnm_print(2, "parseFE:  read non-int (%s) for maxvert!\n", tok);
00399         return VRC_FAILURE;
00400     }
00401     thee->maxvert = ti;
00402     thee->setmaxvert = 1;
00403     return VRC_SUCCESS;
00404 }
00405 VERROR1:
00406     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00407     return VRC_FAILURE;
00408 }
00409 }
00410
00411 VPRIVATE Vrc_Codes FEMparm_parseUSEMESH(FEMparm *thee, Vio *sock) {
00412     char tok[VMAX_BUFSIZE];
00413     int ti;
00414
00415     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00416     if (sscanf(tok, "%d", &ti) == 0) {
00417         Vnm_print(2, "parseFE:  read non-int (%s) for usemesh!\n", tok);
00418         return VRC_FAILURE;
00419     }
00420     thee->useMesh = 1;
00421     thee->meshID = ti;
00422
00423     return VRC_SUCCESS;
00424 }
00425 VERROR1:
00426     Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00427     return VRC_FAILURE;
00428 }
00429 }
00430
00431 VPUBLIC Vrc_Codes FEMparm_parseToken(FEMparm *thee, char tok[VMAX_BUFSIZE],
00432     Vio *sock) {
00433
00434     //int i, ti; // gcc says unused
00435     //double tf; // gcc says unused
00436
00437     if (thee == VNULL) {
00438         Vnm_print(2, "parseFE:  got NULL thee!\n");
00439         return VRC_FAILURE;
00440     }
00441
00442     if (sock == VNULL) {
00443         Vnm_print(2, "parseFE:  got NULL socket!\n");

```

```

00444         return VRC_FAILURE;
00445     }
00446
00447     if (Vstring_strcasecmp(tok, "domainLength") == 0) {
00448         return FEMparm_parseDOMAINLENGTH(thee, sock);
00449     } else if (Vstring_strcasecmp(tok, "etol") == 0) {
00450         return FEMparm_parseETOL(thee, sock);
00451     } else if (Vstring_strcasecmp(tok, "ekey") == 0) {
00452         return FEMparm_parseEKEY(thee, sock);
00453     } else if (Vstring_strcasecmp(tok, "akeyPRE") == 0) {
00454         return FEMparm_parseAKEYPRE(thee, sock);
00455     } else if (Vstring_strcasecmp(tok, "akeySOLVE") == 0) {
00456         return FEMparm_parseAKEYSOLVE(thee, sock);
00457     } else if (Vstring_strcasecmp(tok, "targetNum") == 0) {
00458         return FEMparm_parseTARGETNUM(thee, sock);
00459     } else if (Vstring_strcasecmp(tok, "targetRes") == 0) {
00460         return FEMparm_parseTARGETRES(thee, sock);
00461     } else if (Vstring_strcasecmp(tok, "maxsolve") == 0) {
00462         return FEMparm_parseMAXSOLVE(thee, sock);
00463     } else if (Vstring_strcasecmp(tok, "maxvert") == 0) {
00464         return FEMparm_parseMAXVERT(thee, sock);
00465     } else if (Vstring_strcasecmp(tok, "usemesh") == 0) {
00466         return FEMparm_parseUSEMESH(thee, sock);
00467     }
00468
00469     return VRC_WARNING;
00470
00471 }

```

10.21 src/generic/femparm.h File Reference

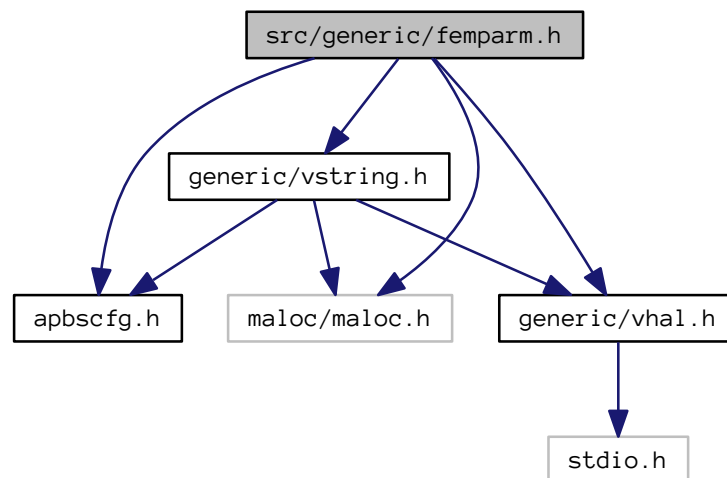
Contains declarations for class APOLparm.

```

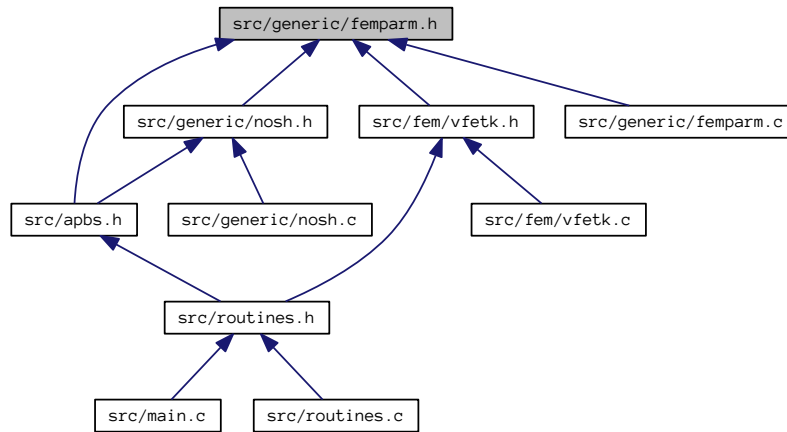
#include "apbscfg.h"
#include "malloc/malloc.h"
#include "generic/vhal.h"
#include "generic/vstring.h"

```

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 refinement error estimate tolerance key.
- enum [eFEMparm_EstType](#) { FRT_UNIF =0, FRT_GEOM =1, FRT_RESI =2, FRT_DUAL =3, FRT_LOCA =4 }
Adaptive refinement error estimator method.
- enum [eFEMparm_CalcType](#) { FCT_MANUAL, FCT_NONE }
Calculation type.

Functions

- VEXTERNC `FEMparm * FEMparm_ctor` (`FEMparm_CalcType` type)
Construct FEMparm.
- VEXTERNC `int FEMparm_ctor2` (`FEMparm *thee`, `FEMparm_CalcType` type)
FORTTRAN stub to construct FEMparm.
- VEXTERNC `void FEMparm_dtor` (`FEMparm **thee`)
Object destructor.
- VEXTERNC `void FEMparm_dtor2` (`FEMparm *thee`)
FORTTRAN 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\$

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
```

```

*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [femparm.h](#).

10.22 femparm.h

```

00001
00063 #ifndef _FEMPARM_H_
00064 #define _FEMPARM_H_
00065
00066 /* Generic header files */
00067 #include "apbscfg.h"
00068
00069 #include "malloc/malloc.h"
00070
00071 #include "generic/vhal.h"
00072 #include "generic/vstring.h"
00073
00079 enum eFEMparm_EtolType {
00080     FET_SIMP=0,
00081     FET_GLOB=1,
00082     FET_FRAC=2
00083 };
00084
00090 typedef enum eFEMparm_EtolType FEMparm_EtolType;
00091
00098 enum eFEMparm_EstType {
00099     FRT_UNIF=0,
00100     FRT_GEOM=1,
00101     FRT_RESI=2,
00102     FRT_DUAL=3,
00104     FRT_LOCA=4
00105 };
00106
00111 typedef enum eFEMparm_EstType FEMparm_EstType;
00112
00117 enum eFEMparm_CalcType {
00118     FCT_MANUAL,
00119     FCT_NONE
00120 };
00121
00126 typedef enum eFEMparm_CalcType FEMparm_CalcType;
00127
00133 struct sFEMparm {
00134
00135     int parsed;
00138     FEMparm_CalcType type;
00139     int settype;
00140     double glen[3];
00141     int setglen;
00142     double etol;

```

```

00144     int  setetol;
00145     FEMparm_EtolType ekey;
00147     int  setekey;
00148     FEMparm_EstType akeyPRE;
00151     int  setakeyPRE;
00152     FEMparm_EstType akeySOLVE;
00154     int  setakeySOLVE;
00155     int  targetNum;
00159     int  settargetNum;
00160     double targetRes;
00164     int  settargetRes;
00165     int  maxsolve;
00166     int  setmaxsolve;
00167     int  maxvert;
00169     int  setmaxvert;
00170     int  pkey;
00173     int  useMesh;
00174     int  meshID;
00176 };
00177
00182 typedef struct sFEMparm FEMparm;
00183
00184 /* ////////////////////////////////////////////////////////////////////
00185 // Class NOsh: Non-inlineable methods (nosh.c)
00187
00194 VEXTERNC FEMparm* FEMparm_ctor(FEMparm_CalcType type);
00195
00203 VEXTERNC int FEMparm_ctor2(FEMparm *thee, FEMparm_CalcType type);
00204
00210 VEXTERNC void FEMparm_dtor(FEMparm **thee);
00211
00217 VEXTERNC void FEMparm_dtor2(FEMparm *thee);
00218
00226 VEXTERNC int FEMparm_check(FEMparm *thee);
00227
00234 VEXTERNC void FEMparm_copy(FEMparm *thee, FEMparm *source);
00235
00246 VEXTERNC Vrc_Codes FEMparm_parseToken(FEMparm *thee, char tok[VMAX_BUFSIZE],
00247     Vio *sock);
00248
00249 #endif
00250

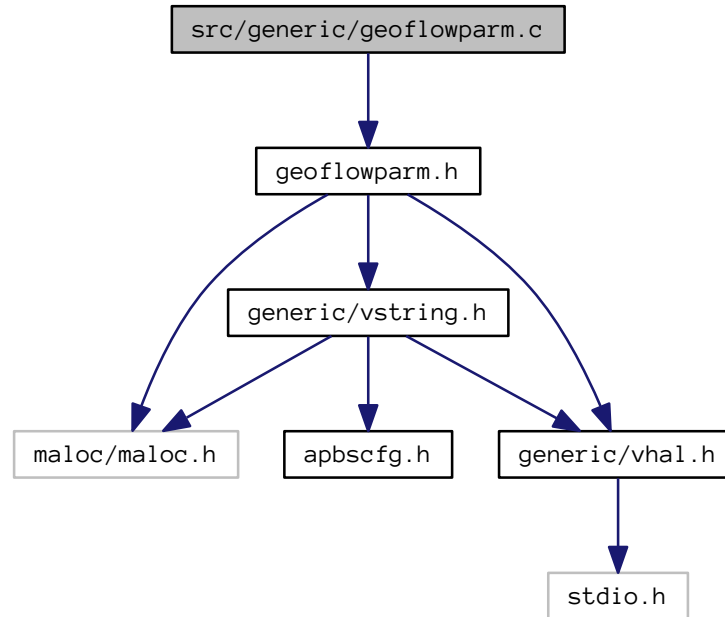
```

10.23 src/generic/geoflowparm.c File Reference

Class GEOFLOWparm methods.


```
#include "geoflowparm.h"
```

Include dependency graph for geoflowparm.c:



Functions

- VPUBLIC [GEOFLOWparm](#) * [GEOFLOWparm_ctor](#) ([GEOFLOWparm_CalcType](#) type)
Construct GEOFLOWparm object.
- VPUBLIC Vrc_Codes [GEOFLOWparm_ctor2](#) ([GEOFLOWparm](#) *thee, [GEOFLOWparm_CalcType](#) type)
FORTTRAN stub to construct GEOFLOWparm object ??????????!!!!!!!
- VPUBLIC void [GEOFLOWparm_dtor](#) ([GEOFLOWparm](#) **thee)
Object destructor.
- VPUBLIC void [GEOFLOWparm_dtor2](#) ([GEOFLOWparm](#) *thee)
FORTTRAN stub for object destructor ??????????!!!!!!!
- VPUBLIC Vrc_Codes [GEOFLOWparm_check](#) ([GEOFLOWparm](#) *thee)
Consistency check for parameter values stored in object.
- VPUBLIC void [GEOFLOWparm_copy](#) ([GEOFLOWparm](#) *thee, [GEOFLOWparm](#) *parm)
- Vrc_Codes **FUBAR** (const char *name)
- Vrc_Codes **parseNonNeg** (double *tf, double def, int *set, char *name, Vio *sock)
- VPRIVATE Vrc_Codes **GEOFLOWparm_parseVDW** ([GEOFLOWparm](#) *thee, Vio *sock)
- VPUBLIC Vrc_Codes [GEOFLOWparm_parseToken](#) ([GEOFLOWparm](#) *thee, char tok[VMAX_BUFSIZE], Vio *sock)
Parse an MG keyword from an input file.

10.23.1 Detailed Description

Class GEOFLOWparm methods.

Author

Andrew Stevens

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [geoflowparm.c](#).

10.24 geoflowparm.c

```

00001
00057 #include "geoflowparm.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_MGPARM)
00062
00063 #endif /* if !defined(VINLINE_MGPARM) */
00064
00065
00066 VPUBLIC GEOFLOWparm* GEOFLOWparm_ctor(
    GEOFLOWparm_CalcType type) {
00067
00068     /* Set up the structure */
00069     GEOFLOWparm *thee = VNULL;
00070     thee = (GEOFLOWparm*)Vmem_malloc(VNULL, 1, sizeof(GEOFLOWparm));
00071     VASSERT( thee != VNULL);
00072     VASSERT( GEOFLOWparm_ctor2(thee, type) == VRC_SUCCESS );
00073
00074     return thee;
00075 }
00076
00077 VPUBLIC Vrc_Codes GEOFLOWparm_ctor2(GEOFLOWparm *thee,
    GEOFLOWparm_CalcType type) {
00078
00079     int i;
00080
00081     if (thee == VNULL) return VRC_FAILURE;
00082
00083     thee->parsed = 0;
00084     thee->type = type;
00085     thee->vdw = 0;
00086     // thee->dcel = 0.25;
00087     // thee->pres = 0.008;
00088     // thee->gama = 0.0001;
00089
00090     return VRC_SUCCESS;
00091 }
00092
00093 VPUBLIC void GEOFLOWparm_dtor(GEOFLOWparm **thee) {
00094     if ((*thee) != VNULL) {
00095         GEOFLOWparm_dtor2(*thee);
00096         Vmem_free(VNULL, 1, sizeof(GEOFLOWparm), (void **)thee);
00097         (*thee) = VNULL;
00098     }
00099 }
00100
00101 VPUBLIC void GEOFLOWparm_dtor2(GEOFLOWparm *thee) { ; }
00102
00103 VPUBLIC Vrc_Codes GEOFLOWparm_check(GEOFLOWparm *thee) {
00104
00105     Vrc_Codes rc;
00106
00107     rc = VRC_SUCCESS;
00108
00109     Vnm_print(0, "GEOFLOWparm_check: checking GEOFLOWparm object of type %d.\n",
        thee->type);
00110
00111
00112     /* Check to see if we were even filled... */
00113     if (!thee->parsed) {
00114         Vnm_print(2, "GEOFLOWparm_check: not filled!\n");
00115         return VRC_FAILURE;
00116     }
00117
00118
00119     /* Check type settings */
00120     if ((thee->type != GFCT_MANUAL)&& (thee->type !=
        GFCT_AUTO)&& (thee->type != GFCT_NONE)) {
00121         Vnm_print(2, "GEOFLOWparm_check: type not set");
00122         rc = VRC_FAILURE;
00123     }
00124
00125     return rc;
00126 }
00127
00128 VPUBLIC void GEOFLOWparm_copy(GEOFLOWparm *thee, GEOFLOWparm *parm) {
00129     VASSERT(thee != VNULL);
00130     VASSERT(parm != VNULL);

```

```

00131
00132     thee->type = parm->type;
00133     thee->parsed = parm->parsed;
00134
00135     thee->vdw = parm->vdw;
00136     //     thee->dcel = parm->dcel;
00137     //     thee->pres = parm->pres;
00138     //     thee->gama = parm->gama;
00139 }
00140
00141 Vrc_Codes FUBAR(const char* name){
00142     Vnm_print(2, "parseGEOFLOW: ran out of tokens on %s!\n", name);
00143     return VRC_WARNING;
00144 }
00145
00146 Vrc_Codes parseNonNeg(double* tf, double def, int* set, char* name, Vio* sock){
00147     char tok[VMAX_BUFSIZE];
00148     if(Vio_scanf(sock, "%s", tok) == 0) {
00149         *tf = def;
00150         return FUBAR(name);
00151     }
00152
00153     if (sscanf(tok, "%lf", tf) == 0){
00154         Vnm_print(2, "NOsh: Read non-float (%s) while parsing %s keyword!\n", tok, name);
00155         *tf = def;
00156         return VRC_WARNING;
00157     }else if(*tf < 0.0){
00158         Vnm_print(2, "parseGEOFLOW: %s must be greater than 0!\n", name);
00159         *tf = def;
00160         return VRC_WARNING;
00161     }
00162
00163     *set = 1;
00164     return VRC_SUCCESS;
00165 }
00166
00167 VPRIVATE Vrc_Codes GEOFLOWparm_parseVDW(GEOFLOWparm *thee, Vio *sock){
00168     const char* name = "vdw";
00169     char tok[VMAX_BUFSIZE];
00170     int tf;
00171     if(Vio_scanf(sock, "%s", tok) == 0) {
00172         return FUBAR(name);
00173     }
00174
00175
00176     if (sscanf(tok, "%u", &tf) == 0){
00177         Vnm_print(2, "NOsh: Read non-unsigned int (%s) while parsing %s keyword!\n", tok, name);
00178         return VRC_WARNING;
00179     }else if(tf != 0 && tf != 1){
00180         Vnm_print(2, "parseGEOFLOW: %s must be 0 or 1!\n", name);
00181         return VRC_WARNING;
00182     }else{
00183         thee->vdw = tf;
00184     }
00185     thee->setvdw = 1;
00186     return VRC_SUCCESS;
00187 }
00188
00189 VPUBLIC Vrc_Codes GEOFLOWparm_parseToken(GEOFLOWparm *thee, char tok[
VMAX_BUFSIZE],
Vio *sock) {
00191
00192     if (thee == VNULL) {
00193         Vnm_print(2, "parseGEOFLOW: got NULL thee!\n");
00194         return VRC_WARNING;
00195     }
00196     if (sock == VNULL) {
00197         Vnm_print(2, "parseGEOFLOW: got NULL socket!\n");
00198         return VRC_WARNING;
00199     }
00200
00201     Vnm_print(0, "GEOFLOWparm_parseToken: trying %s...\n", tok);
00202
00203
00204     //     if (Vstring_strcasecmp(tok, "press") == 0) {
00205     //         return parseNonNeg(&(thee->pres), 0.008, &(thee->setpres), "pres", sock);
00206     //     } else if (Vstring_strcasecmp(tok, "gamma") == 0) {
00207     //         return parseNonNeg(&(thee->gama), 0.0001, &(thee->setgama), "gama", sock);
00208     //     } else if (Vstring_strcasecmp(tok, "grid") == 0) {
00209     //         return parseNonNeg(&(thee->dcel), 0.25, &(thee->setdcel), "dcel", sock);
00210     //     } else

```

```

00211     if (Vstring_strcasecmp(tok, "vdwdisp") == 0) {
00212         return GEOFLOWparm_parseVDW(thee, sock);
00213     } else {
00214         Vnm_print(2, "parseGEOFLOW: Unrecognized keyword (%s)!\n", tok);
00215         return VRC_WARNING;
00216     }
00217
00218     return VRC_FAILURE;
00219 }

```

10.25 src/generic/geoflowparm.h File Reference

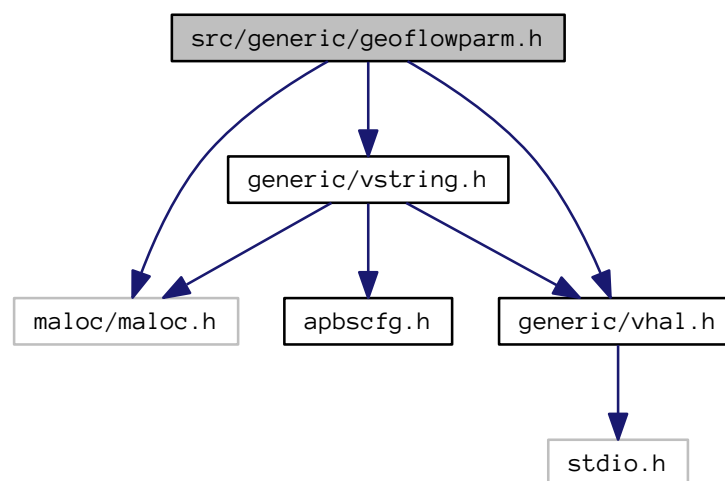
Contains declarations for class GEOFLOWparm.

```

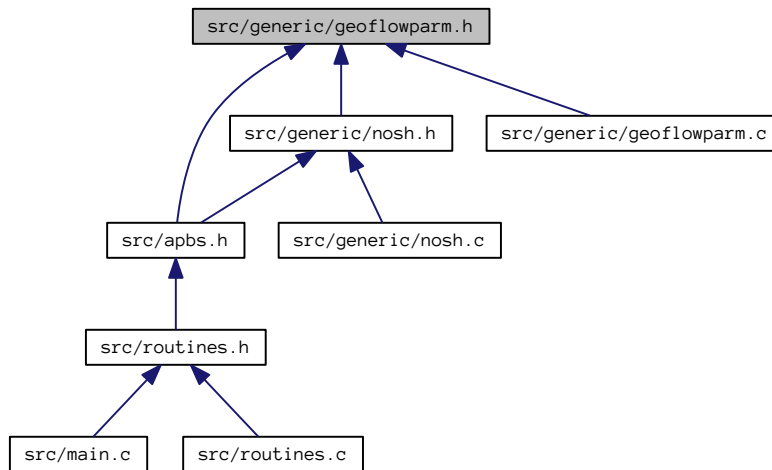
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/vstring.h"

```

Include dependency graph for geoflowparm.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [sGEOFLOWparm](#)
Parameter structure for GEOFLOW-specific variables from input files.

Typedefs

- typedef enum [eGEOFLOWparm_CalcType](#) [GEOFLOWparm_CalcType](#)
Declare GEOFLOWparm_CalcType type.
- typedef struct [sGEOFLOWparm](#) [GEOFLOWparm](#)
Parameter structure for GEOFLOW-specific variables from input files.

Enumerations

- enum [eGEOFLOWparm_CalcType](#) { [GFCT_MANUAL](#) =0, [GFCT_AUTO](#) =1, [GFCT_NONE](#) =2 }
Calculation type.

Functions

- VEXTERNC [GEOFLOWparm](#) * [GEOFLOWparm_ctor](#) ([GEOFLOWparm_CalcType](#) type)
Construct GEOFLOWparm object.
- VEXTERNC Vrc_Codes [GEOFLOWparm_ctor2](#) ([GEOFLOWparm](#) *thee, [GEOFLOWparm_CalcType](#) type)
FORTTRAN stub to construct GEOFLOWparm object ??????????!!!!!!
- VEXTERNC void [GEOFLOWparm_dtor](#) ([GEOFLOWparm](#) **thee)
Object destructor.

- VEXTERNC void [GEOFLOWparm_dtor2](#) ([GEOFLOWparm](#) *thee)
FORTTRAN stub for object destructor ?????????!!!!!!!
- VEXTERNC Vrc_Codes [GEOFLOWparm_check](#) ([GEOFLOWparm](#) *thee)
Consistency check for parameter values stored in object.
- VEXTERNC Vrc_Codes [GEOFLOWparm_parseToken](#) ([GEOFLOWparm](#) *thee, char tok[VMAX_BUFSIZE], Vio *sock)
Parse an MG keyword from an input file.
- VPRIVATE Vrc_Codes [GEOFLOWparm_parseVDW](#) ([GEOFLOWparm](#) *thee, Vio *sock)

10.25.1 Detailed Description

Contains declarations for class [GEOFLOWparm](#).

Version

\$Id\$

Author

Andrew Stevens

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
```

```

* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [geoflowparm.h](#).

10.26 geoflowparm.h

```

00001
00064 #ifndef _GEOFLOWPARM_H_
00065 #define _GEOFLOWPARM_H_
00066
00067 /* Generic header files */
00068 #include "maloc/maloc.h"
00069
00070 #include "generic/vhal.h"
00071 #include "generic/vstring.h"
00072
00077 enum eGEOFLOWparm_CalcType {
00078     GFCT_MANUAL=0,
00079     GFCT_AUTO=1,
00080     GFCT_NONE=2
00081 };
00082
00087 typedef enum eGEOFLOWparm_CalcType GEOFLOWparm_CalcType;
00088
00097 typedef struct sGEOFLOWparm {
00098
00099     GEOFLOWparm_CalcType type;
00100     int parsed;
00102     /* *** GENERIC PARAMETERS *** */
00103     // double dcel;
00104     // double pres;
00105     // double gama;
00106     int vdw;
00107
00108     // int setdcel;
00109     // int setpres;
00110     // int setgama;
00111     int setvdw;
00112
00113 } GEOFLOWparm;
00114
00121 VEXTERNC GEOFLOWparm* GEOFLOWparm_ctor(
    GEOFLOWparm_CalcType type);
00122
00130 VEXTERNC Vrc_Codes GEOFLOWparm_ctor2(GEOFLOWparm *thee,
    GEOFLOWparm_CalcType type);
00131
00137 VEXTERNC void GEOFLOWparm_dtor(GEOFLOWparm **thee);
00138
00144 VEXTERNC void GEOFLOWparm_dtor2(GEOFLOWparm *thee);
00145
00152 VEXTERNC Vrc_Codes GEOFLOWparm_check(GEOFLOWparm *thee);
00153
00163 VEXTERNC Vrc_Codes GEOFLOWparm_parseToken(GEOFLOWparm *thee, char tok
    [VMAX_BUFSIZE],
00164     Vio *sock);
00165
00166 VPRIVATE Vrc_Codes GEOFLOWparm_parseVDW(GEOFLOWparm *thee, Vio *sock);
00167
00168 #endif
00169

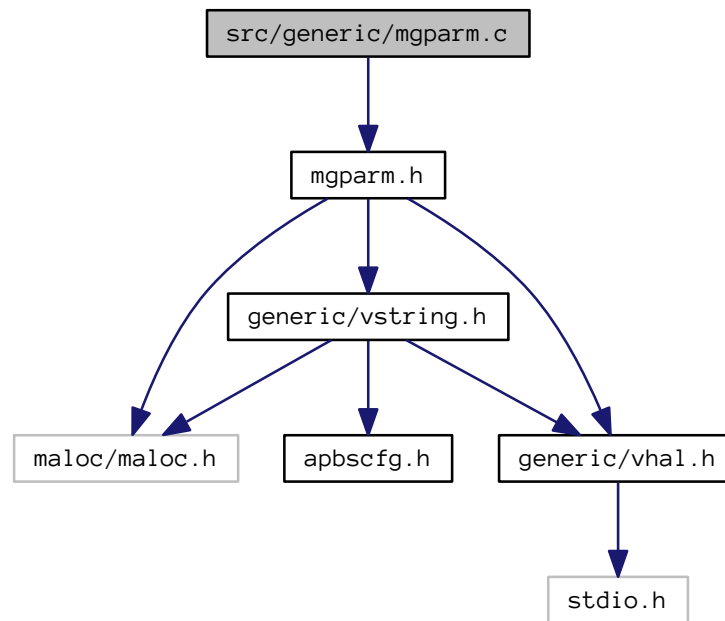
```


10.27 src/generic/mgparm.c File Reference

Class MGparm methods.

```
#include "mgparm.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)
- FORTTRAN stub to construct MGparm object.*
- VPUBLIC void [MGparm_dtor](#) ([MGparm](#) **thee)
- Object destructor.*
- VPUBLIC void [MGparm_dtor2](#) ([MGparm](#) *thee)
- FORTTRAN 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.*

10.27.1 Detailed Description

Class MGparm methods.

Author

Nathan Baker

Version

\$Id\$

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [mgparm.c](#).

10.28 mgparm.c

```

00001
00057 #include "mgparm.h"
00058
00059 #ifdef VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_MGPARM)
00062
00063 #endif /* if !defined(VINLINE_MGPARM) */

```

```

00064
00065 VPUBLIC void MGparm_setCenterX(MGparm *thee, double x) {
00066     VASSERT(thee != VNULL);
00067     thee->center[0] = x;
00068 }
00069 VPUBLIC void MGparm_setCenterY(MGparm *thee, double y) {
00070     VASSERT(thee != VNULL);
00071     thee->center[1] = y;
00072 }
00073 VPUBLIC void MGparm_setCenterZ(MGparm *thee, double z) {
00074     VASSERT(thee != VNULL);
00075     thee->center[2] = z;
00076 }
00077 VPUBLIC double MGparm_getCenterX(MGparm *thee) {
00078     VASSERT(thee != VNULL);
00079     return thee->center[0];
00080 }
00081 VPUBLIC double MGparm_getCenterY(MGparm *thee) {
00082     VASSERT(thee != VNULL);
00083     return thee->center[1];
00084 }
00085 VPUBLIC double MGparm_getCenterZ(MGparm *thee) {
00086     VASSERT(thee != VNULL);
00087     return thee->center[2];
00088 }
00089 VPUBLIC int MGparm_getNx(MGparm *thee) {
00090     VASSERT(thee != VNULL);
00091     return thee->dime[0];
00092 }
00093 VPUBLIC int MGparm_getNy(MGparm *thee) {
00094     VASSERT(thee != VNULL);
00095     return thee->dime[1];
00096 }
00097 VPUBLIC int MGparm_getNz(MGparm *thee) {
00098     VASSERT(thee != VNULL);
00099     return thee->dime[2];
00100 }
00101 VPUBLIC double MGparm_getHx(MGparm *thee) {
00102     VASSERT(thee != VNULL);
00103     return thee->grid[0];
00104 }
00105 VPUBLIC double MGparm_getHy(MGparm *thee) {
00106     VASSERT(thee != VNULL);
00107     return thee->grid[1];
00108 }
00109 VPUBLIC double MGparm_getHz(MGparm *thee) {
00110     VASSERT(thee != VNULL);
00111     return thee->grid[2];
00112 }
00113
00114 VPUBLIC MGparm* MGparm_ctor(MGparm_CalcType type) {
00115
00116     /* Set up the structure */
00117     MGparm *thee = VNULL;
00118     thee = (MGparm*)Vmem_malloc(VNULL, 1, sizeof(MGparm));
00119     VASSERT( thee != VNULL);
00120     VASSERT( MGparm_ctor2(thee, type) == VRC_SUCCESS );
00121
00122     return thee;
00123 }
00124
00125 VPUBLIC Vrc_Codes MGparm_ctor2(MGparm *thee, MGparm_CalcType type) {
00126
00127     int i;
00128
00129     if (thee == VNULL) return VRC_FAILURE;
00130
00131     for (i=0; i<3; i++) {
00132         thee->dime[i] = -1;
00133         thee->pdime[i] = 1;
00134     }
00135
00136     thee->parsed = 0;
00137     thee->type = type;
00138
00139     /* *** GENERIC PARAMETERS *** */
00140     thee->setdime = 0;
00141     thee->setchgmm = 0;
00142
00143     /* *** TYPE 0 PARAMETERS *** */
00144     thee->nlev = VMGNLEV;

```

```

00145     thee->setnlev = 1;
00146     thee->etol = 1.0e-6;
00147     thee->setetol = 0;
00148     thee->setgrid = 0;
00149     thee->setglen = 0;
00150     thee->setgcent = 0;
00151
00152     /* *** TYPE 1 & 2 PARAMETERS *** */
00153     thee->setcglen = 0;
00154     thee->setfglen = 0;
00155     thee->setcgcent = 0;
00156     thee->setfgcent = 0;
00157
00158     /* *** TYPE 2 PARAMETERS *** */
00159     thee->setpdime = 0;
00160     thee->setrank = 0;
00161     thee->setsize = 0;
00162     thee->setofrac = 0;
00163     for (i=0; i<6; i++) thee->partDisjOwnSide[i] = 0;
00164     thee->setasync = 0;
00165
00166     /* *** Default parameters for TINKER *** */
00167     thee->chgs = VCM_CHARGE;
00168
00169     thee->useAqua = 0;
00170     thee->setUseAqua = 0;
00171
00172     return VRC_SUCCESS;
00173 }
00174
00175 VPUBLIC void MGparm_dtor(MGparm **thee) {
00176     if ((*thee) != VNULL) {
00177         MGparm_dtor2(*thee);
00178         Vmem_free(VNULL, 1, sizeof(MGparm), (void **)thee);
00179         (*thee) = VNULL;
00180     }
00181 }
00182
00183 VPUBLIC void MGparm_dtor2(MGparm *thee) { ; }
00184
00185 VPUBLIC Vrc_Codes MGparm_check(MGparm *thee) {
00186     Vrc_Codes rc;
00187     int i, tdime[3], ti, tnlev[3], nlev;
00188
00189     rc = VRC_SUCCESS;
00190
00191     Vnm_print(0, "MGparm_check:  checking MGparm object of type %d.\n",
00192             thee->type);
00193
00194     /* Check to see if we were even filled... */
00195     if (!thee->parsed) {
00196         Vnm_print(2, "MGparm_check:  not filled!\n");
00197         return VRC_FAILURE;
00198     }
00199
00200     /* Check generic settings */
00201     if (!thee->setdime) {
00202         Vnm_print(2, "MGparm_check:  DIME not set!\n");
00203         rc = VRC_FAILURE;
00204     }
00205
00206     if (!thee->setchgm) {
00207         Vnm_print(2, "MGparm_check:  CHGM not set!\n");
00208         return VRC_FAILURE;
00209     }
00210
00211     /* Check sequential manual & dummy settings */
00212     if ((thee->type == MCT_MANUAL) || (thee->type == MCT_DUMMY)) {
00213         if ((!thee->setgrid) && (!thee->setglen)) {
00214             Vnm_print(2, "MGparm_check:  Neither GRID nor GLEN set!\n");
00215             rc = VRC_FAILURE;
00216         }
00217         if ((thee->setgrid) && (thee->setglen)) {
00218             Vnm_print(2, "MGparm_check:  Both GRID and GLEN set!\n");
00219             rc = VRC_FAILURE;
00220         }
00221         if (!thee->setgcent) {
00222             Vnm_print(2, "MGparm_check:  GCENT not set!\n");
00223             rc = VRC_FAILURE;
00224         }
00225     }

```

```

00226     }
00227
00228     /* Check sequential and parallel automatic focusing settings */
00229     if ((thee->type == MCT_AUTO) || (thee->type == MCT_PARALLEL)) {
00230         if (!thee->setcglen) {
00231             Vnm_print(2, "MGparm_check:  CGLEN not set!\n");
00232             rc = VRC_FAILURE;
00233         }
00234         if (!thee->setfglen) {
00235             Vnm_print(2, "MGparm_check:  FGLEN not set!\n");
00236             rc = VRC_FAILURE;
00237         }
00238         if (!thee->setcgcent) {
00239             Vnm_print(2, "MGparm_check:  CGCENT not set!\n");
00240             rc = VRC_FAILURE;
00241         }
00242         if (!thee->setfgcent) {
00243             Vnm_print(2, "MGparm_check:  FGCENT not set!\n");
00244             rc = VRC_FAILURE;
00245         }
00246     }
00247
00248     /* Check parallel automatic focusing settings */
00249     if (thee->type == MCT_PARALLEL) {
00250         if (!thee->setpdime) {
00251             Vnm_print(2, "MGparm_check:  PDIME not set!\n");
00252             rc = VRC_FAILURE;
00253         }
00254         if (!thee->setrank) {
00255             Vnm_print(2, "MGparm_check:  PROC_RANK not set!\n");
00256             rc = VRC_FAILURE;
00257         }
00258         if (!thee->setsize) {
00259             Vnm_print(2, "MGparm_check:  PROC_SIZE not set!\n");
00260             rc = VRC_FAILURE;
00261         }
00262         if (!thee->setofrac) {
00263             Vnm_print(2, "MGparm_check:  OFRAC not set!\n");
00264             rc = VRC_FAILURE;
00265         }
00266     }
00267
00268     /* Perform a sanity check on nlev and dime, resetting values as necessary */
00269     if (rc == 1) {
00270         /* Calculate the actual number of grid points and nlev to satisfy the
00271          * formula:  $n = c * 2^{(l+1)} + 1$ , where  $n$  is the number of grid points,
00272          *  $c$  is an integer, and  $l$  is the number of levels */
00273         if (thee->type != MCT_DUMMY) {
00274             for (i=0; i<3; i++) {
00275                 /* See if the user picked a reasonable value, if not then fix it */
00276                 ti = thee->dime[i] - 1;
00277                 if (ti == VPOW(2, (thee->nlev+1))) {
00278                     tnlev[i] = thee->nlev;
00279                     tdime[i] = thee->dime[i];
00280                 } else {
00281                     tdime[i] = thee->dime[i];
00282                     ti = tdime[i] - 1;
00283                     tnlev[i] = 0;
00284                     /* Find the maximum number of times this dimension can be
00285                      * divided by two */
00286                     while (VEVEN(ti)) {
00287                         (tnlev[i])++;
00288                         ti = (int)ceil(0.5*ti);
00289                     }
00290                     (tnlev[i])--;
00291                     /* We'd like to have at least VMGNLEV levels in the multigrid
00292                      * hierarchy. This means that the dimension needs to be
00293                      *  $c * 2^{VMGNLEV} + 1$ , where  $c$  is an integer. */
00294                     if ((tdime[i] > 65) && (tnlev[i] < VMGNLEV)) {
00295                         Vnm_print(2, "Nosh: Bad dime[%d] = %d (%d nlev)!\n",
00296                             i, tdime[i], tnlev[i]);
00297                         ti = (int)(tdime[i]/VPOW(2., (VMGNLEV+1)));
00298                         if (ti < 1) ti = 1;
00299                         tdime[i] = ti*(int)(VPOW(2., (VMGNLEV+1))) + 1;
00300                         tnlev[i] = 4;
00301                         Vnm_print(2, "Nosh: Reset dime[%d] to %d and (nlev = %d).\n", i, tdime[i],
VMGNLEV);
00302                     }
00303                 }
00304             }
00305         } else { /* We are a dummy calculation, but we still need positive numbers of points */

```

```

00306         for (i=0; i<3; i++) {
00307             tnlev[i] = thee->nlev;
00308             tdime[i] = thee->dime[i];
00309             if (thee->dime[i] <= 0) {
00310                 Vnm_print(2, "Nosh: Resetting dime[%d] from %d to 3.\n", i, thee->
dime[i]);
00311                 thee->dime[i] = 3;
00312             }
00313         }
00314     }
00315
00316     /* The actual number of levels we'll be using is the smallest number of
00317        * possible levels in any dimensions */
00318     nlev = VMIN2(tnlev[0], tnlev[1]);
00319     nlev = VMIN2(nlev, tnlev[2]);
00320     /* Set the number of levels and dimensions */
00321     Vnm_print(0, "Nosh: nlev = %d, dime = (%d, %d, %d)\n", nlev, tdime[0],
00322             tdime[1], tdime[2]);
00323     thee->nlev = nlev;
00324     if (thee->nlev <= 0) {
00325         Vnm_print(2, "MGparm_check: illegal nlev (%d); check your grid dimensions!\n", thee->
nlev);
00326         rc = VRC_FAILURE;
00327     }
00328     if (thee->nlev < 2) {
00329         Vnm_print(2, "MGparm_check: you're using a very small nlev (%d) and therefore\n", thee->
nlev);
00330         Vnm_print(2, "MGparm_check: will not get the optimal performance of the multigrid\n");
00331         Vnm_print(2, "MGparm_check: algorithm. Please check your grid dimensions.\n");
00332     }
00333     for (i=0; i<3; i++) thee->dime[i] = tdime[i];
00334 }
00335
00336 if (!thee->setUseAqua) thee->useAqua = 0;
00337
00338 return rc;
00339 }
00340
00341 VPUBLIC void MGparm_copy(MGparm *thee, MGparm *parm) {
00342
00343     int i;
00344
00345     VASSERT(thee != VNULL);
00346     VASSERT(parm != VNULL);
00347
00348     thee->type = parm->type;
00349     thee->parsed = parm->parsed;
00350
00351     /* *** GENERIC PARAMETERS *** */
00352     for (i=0; i<3; i++) thee->dime[i] = parm->dime[i];
00353     thee->setdime = parm->setdime;
00354     thee->chgm = parm->chgm;
00355     thee->setchgm = parm->setchgm;
00356     thee->chgs = parm->chgs;
00357
00358     /* *** TYPE 0 PARMS *** */
00359     thee->nlev = parm->nlev;
00360     thee->setnlev = parm->setnlev;
00361     thee->etol = parm->etol;
00362     thee->setetol = parm->setetol;
00363     for (i=0; i<3; i++) thee->grid[i] = parm->grid[i];
00364     thee->setgrid = parm->setgrid;
00365     for (i=0; i<3; i++) thee->glen[i] = parm->glen[i];
00366     thee->setglen = parm->setglen;
00367     thee->cmeth = parm->cmeth;
00368     for (i=0; i<3; i++) thee->center[i] = parm->center[i];
00369     thee->setgcent = parm->setgcent;
00370     thee->centmol = parm->centmol;
00371
00372     /* *** TYPE 1 & 2 PARMS *** */
00373     for (i=0; i<3; i++) thee->cglen[i] = parm->cglen[i];
00374     thee->setcglen = parm->setcglen;
00375     for (i=0; i<3; i++) thee->fglen[i] = parm->fglen[i];
00376     thee->setfglen = parm->setfglen;
00377     thee->ccmeth = parm->ccmeth;
00378     for (i=0; i<3; i++) thee->ccenter[i] = parm->ccenter[i];
00379     thee->setcgcent = parm->setcgcent;
00380     thee->ccentmol = parm->ccentmol;
00381     thee->fcmeth = parm->fcmeth;
00382     for (i=0; i<3; i++) thee->fcenter[i] = parm->fcenter[i];
00383

```

```

00384     thee->setfgcent = parm->setfgcent;
00385     thee->fcentmol = parm->fcentmol;
00386
00387     /* *** TYPE 2 PARMS *** */
00388     for (i=0; i<3; i++)
00389         thee->partDisjCenter[i] = parm->partDisjCenter[i];
00390     for (i=0; i<3; i++)
00391         thee->partDisjLength[i] = parm->partDisjLength[i];
00392     for (i=0; i<6; i++)
00393         thee->partDisjOwnSide[i] = parm->partDisjOwnSide[i];
00394     for (i=0; i<3; i++) thee->pdime[i] = parm->pdime[i];
00395     thee->setpdime = parm->setpdime;
00396     thee->proc_rank = parm->proc_rank;
00397     thee->setrank = parm->setrank;
00398     thee->proc_size = parm->proc_size;
00399     thee->setsize = parm->setsize;
00400     thee->ofrac = parm->ofrac;
00401     thee->setofrac = parm->setofrac;
00402     thee->setasync = parm->setasync;
00403     thee->async = parm->async;
00404
00405     thee->nonlintype = parm->nonlintype;
00406     thee->setnonlintype = parm->setnonlintype;
00407
00408     thee->method = parm->method;
00409     thee->method = parm->method;
00410
00411     thee->useAqua = parm->useAqua;
00412     thee->setUseAqua = parm->setUseAqua;
00413 }
00414
00415 VPRIVATE Vrc_Codes MGparm_parseDIME(MGparm *thee, Vio *sock) {
00416
00417     char tok[VMAX_BUFSIZE];
00418     int ti;
00419
00420     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00421     if (sscanf(tok, "%d", &ti) == 0){
00422         Vnm_print(2, "parseMG: Read non-integer (%s) while parsing DIME \
00423 keyword!\n", tok);
00424         return VRC_WARNING;
00425     } else thee->dime[0] = ti;
00426     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00427     if (sscanf(tok, "%d", &ti) == 0) {
00428         Vnm_print(2, "Nosh: Read non-integer (%s) while parsing DIME \
00429 keyword!\n", tok);
00430         return VRC_WARNING;
00431     } else thee->dime[1] = ti;
00432     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00433     if (sscanf(tok, "%d", &ti) == 0) {
00434         Vnm_print(2, "Nosh: Read non-integer (%s) while parsing DIME \
00435 keyword!\n", tok);
00436         return VRC_WARNING;
00437     } else thee->dime[2] = ti;
00438     thee->setdime = 1;
00439     return VRC_SUCCESS;
00440
00441     ERROR1:
00442         Vnm_print(2, "parseMG: ran out of tokens!\n");
00443         return VRC_WARNING;
00444 }
00445
00446 VPRIVATE Vrc_Codes MGparm_parseCHGM(MGparm *thee, Vio *sock) {
00447
00448     char tok[VMAX_BUFSIZE];
00449     Vchrg_Meth ti;
00450
00451     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00452     if (sscanf(tok, "%d", (int*)&ti) == 1) {
00453         thee->chgm = ti;
00454         thee->setchgm = 1;
00455         Vnm_print(2, "Nosh: Warning -- parsed deprecated statment \"chgm %d\".\n", ti);
00456         Vnm_print(2, "Nosh: Please use \"chgm \"");
00457         switch (thee->chgm) {
00458             case VCM_TRIL:
00459                 Vnm_print(2, "spl0");
00460                 break;
00461             case VCM_BSPL2:
00462                 Vnm_print(2, "spl2");
00463                 break;
00464             case VCM_BSPL4:

```



```

00465         Vnm_print(2, "spl4");
00466         break;
00467     default:
00468         Vnm_print(2, "UNKNOWN");
00469         break;
00470     }
00471     Vnm_print(2, "\" instead!\n");
00472     return VRC_SUCCESS;
00473 } else if (Vstring_strcasecmp(tok, "spl0") == 0) {
00474     thee->chgm = VCM_TRIL;
00475     thee->setchgm = 1;
00476     return VRC_SUCCESS;
00477 } else if (Vstring_strcasecmp(tok, "spl2") == 0) {
00478     thee->chgm = VCM_BSPL2;
00479     thee->setchgm = 1;
00480     return VRC_SUCCESS;
00481 } else if (Vstring_strcasecmp(tok, "spl4") == 0) {
00482     thee->chgm = VCM_BSPL4;
00483     thee->setchgm = 1;
00484     return VRC_SUCCESS;
00485 } else {
00486     Vnm_print(2, "Nosh: Unrecognized parameter (%s) when parsing \
00487 chgm!\n", tok);
00488     return VRC_WARNING;
00489 }
00490 return VRC_WARNING;
00491
00492 ERROR1:
00493     Vnm_print(2, "parseMG: ran out of tokens!\n");
00494     return VRC_WARNING;
00495 }
00496
00497 VPRIVATE Vrc_Codes MGparm_parseNLEV(MGparm *thee, Vio *sock) {
00498     char tok[VMAX_BUFSIZE];
00499     int ti;
00500
00501     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00502     if (sscanf(tok, "%d", &ti) == 0) {
00503         Vnm_print(2, "Nosh: Read non-integer (%s) while parsing NLEV \
00504 keyword!\n", tok);
00505         return VRC_WARNING;
00506     } else thee->nlev = ti;
00507     thee->setnlev = 1;
00508     return VRC_SUCCESS;
00509
00510     ERROR1:
00511         Vnm_print(2, "parseMG: ran out of tokens!\n");
00512         return VRC_WARNING;
00513 }
00514
00515 VPRIVATE Vrc_Codes MGparm_parseETOL(MGparm *thee, Vio *sock) {
00516     char tok[VMAX_BUFSIZE];
00517     double tf;
00518
00519     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00520     if (sscanf(tok, "%lf", &tf) == 0) {
00521         Vnm_print(2, "Nosh: Read non-float (%s) while parsing etol \
00522 keyword!\n", tok);
00523         return VRC_WARNING;
00524     } else if (tf <= 0.0) {
00525         Vnm_print(2, "parseMG: etol must be greater than 0!\n");
00526         return VRC_WARNING;
00527     } else thee->etol = tf;
00528     thee->setetol = 1;
00529     return VRC_SUCCESS;
00530
00531     ERROR1:
00532         Vnm_print(2, "parseMG: ran out of tokens!\n");
00533         return VRC_WARNING;
00534 }
00535
00536 VPRIVATE Vrc_Codes MGparm_parseGRID(MGparm *thee, Vio *sock) {
00537     char tok[VMAX_BUFSIZE];
00538     double tf;
00539
00540     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00541     if (sscanf(tok, "%lf", &tf) == 0) {

```

```

00546     Vnm_print(2, "Nosh: Read non-float (%s) while parsing GRID \
00547 keyword!\n", tok);
00548     return VRC_WARNING;
00549 } else thee->grid[0] = tf;
00550 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00551 if (sscanf(tok, "%lf", &tf) == 0) {
00552     Vnm_print(2, "Nosh: Read non-float (%s) while parsing GRID \
00553 keyword!\n", tok);
00554     return VRC_WARNING;
00555 } else thee->grid[1] = tf;
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 GRID \
00559 keyword!\n", tok);
00560     return VRC_WARNING;
00561 } else thee->grid[2] = tf;
00562 thee->setgrid = 1;
00563 return VRC_SUCCESS;
00564
00565 ERROR1:
00566     Vnm_print(2, "parseMG: ran out of tokens!\n");
00567     return VRC_WARNING;
00568 }
00569
00570 VPRIVATE Vrc_Codes MGparm_parseGLEN(MGparm *thee, Vio *sock) {
00571
00572     char tok[VMAX_BUFSIZE];
00573     double tf;
00574
00575     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00576     if (sscanf(tok, "%lf", &tf) == 0) {
00577         Vnm_print(2, "Nosh: Read non-float (%s) while parsing GLEN \
00578 keyword!\n", tok);
00579         return VRC_WARNING;
00580     } else thee->glen[0] = tf;
00581     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00582     if (sscanf(tok, "%lf", &tf) == 0) {
00583         Vnm_print(2, "Nosh: Read non-float (%s) while parsing GLEN \
00584 keyword!\n", tok);
00585         return VRC_WARNING;
00586     } else thee->glen[1] = tf;
00587     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00588     if (sscanf(tok, "%lf", &tf) == 0) {
00589         Vnm_print(2, "Nosh: Read non-float (%s) while parsing GLEN \
00590 keyword!\n", tok);
00591         return VRC_WARNING;
00592     } else thee->glen[2] = tf;
00593     thee->setglen = 1;
00594     return VRC_SUCCESS;
00595
00596     ERROR1:
00597         Vnm_print(2, "parseMG: ran out of tokens!\n");
00598         return VRC_WARNING;
00599 }
00600
00601 VPRIVATE Vrc_Codes MGparm_parseGAMMA(MGparm *thee, Vio *sock) {
00602
00603     char tok[VMAX_BUFSIZE];
00604
00605     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00606     Vnm_print(2, "parseMG: GAMMA keyword deprecated!\n");
00607     Vnm_print(2, "parseMG: If you are using PyMOL or VMD and still seeing this message,\n");
00608     Vnm_print(2, "parseMG: please contact the developers of those programs regarding this message.\n");
00609     return VRC_SUCCESS;
00610
00611     ERROR1:
00612         Vnm_print(2, "parseMG: ran out of tokens!\n");
00613         return VRC_WARNING;
00614 }
00615
00616 VPRIVATE Vrc_Codes MGparm_parseGCENT(MGparm *thee, Vio *sock) {
00617
00618     char tok[VMAX_BUFSIZE];
00619     double tf;
00620     int ti;
00621
00622     /* If the next token isn't a float, it probably means we want to
00623      * center on a molecule */
00624     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00625     if (sscanf(tok, "%lf", &tf) == 0) {
00626         if (Vstring_strcasecmp(tok, "mol") == 0) {

```

```

00627         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00628         if (sscanf(tok, "%d", &ti) == 0) {
00629             Vnm_print(2, "Nosh: Read non-int (%s) while parsing \
00630 GCENT MOL keyword!\n", tok);
00631             return VRC_WARNING;
00632         } else {
00633             thee->cmeth = MCM_MOLECULE;
00634             /* Subtract 1 here to convert user numbering (1, 2, 3, ...) into
00635             array index */
00636             thee->centmol = ti - 1;
00637         }
00638     } else {
00639         Vnm_print(2, "Nosh: Unexpected keyword (%s) while parsing \
00640 GCENT!\n", tok);
00641         return VRC_WARNING;
00642     }
00643 } else {
00644     thee->center[0] = tf;
00645     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00646     if (sscanf(tok, "%lf", &tf) == 0) {
00647         Vnm_print(2, "Nosh: Read non-float (%s) while parsing \
00648 GCENT keyword!\n", tok);
00649         return VRC_WARNING;
00650     }
00651     thee->center[1] = tf;
00652     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00653     if (sscanf(tok, "%lf", &tf) == 0) {
00654         Vnm_print(2, "Nosh: Read non-float (%s) while parsing \
00655 GCENT keyword!\n", tok);
00656         return VRC_WARNING;
00657     }
00658     thee->center[2] = tf;
00659 }
00660 thee->setgcent = 1;
00661 return VRC_SUCCESS;
00662
00663 ERROR1:
00664     Vnm_print(2, "parseMG: ran out of tokens!\n");
00665     return VRC_WARNING;
00666 }
00667
00668 VPRIVATE Vrc_Codes MGparm_parseCGLEN(MGparm *thee, Vio *sock) {
00669
00670     char tok[VMAX_BUFSIZE];
00671     double tf;
00672
00673     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00674     if (sscanf(tok, "%lf", &tf) == 0) {
00675         Vnm_print(2, "Nosh: Read non-float (%s) while parsing CGLEN \
00676 keyword!\n", tok);
00677         return VRC_WARNING;
00678     } else thee->cglen[0] = tf;
00679     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00680     if (sscanf(tok, "%lf", &tf) == 0) {
00681         Vnm_print(2, "Nosh: Read non-float (%s) while parsing CGLEN \
00682 keyword!\n", tok);
00683         return VRC_WARNING;
00684     } else thee->cglen[1] = tf;
00685     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00686     if (sscanf(tok, "%lf", &tf) == 0) {
00687         Vnm_print(2, "Nosh: Read non-float (%s) while parsing CGLEN \
00688 keyword!\n", tok);
00689         return VRC_WARNING;
00690     } else thee->cglen[2] = tf;
00691     thee->setcglen = 1;
00692     return VRC_SUCCESS;
00693
00694     ERROR1:
00695         Vnm_print(2, "parseMG: ran out of tokens!\n");
00696         return VRC_WARNING;
00697 }
00698
00699 VPRIVATE Vrc_Codes MGparm_parseFGLEN(MGparm *thee, Vio *sock) {
00700
00701     char tok[VMAX_BUFSIZE];
00702     double tf;
00703
00704     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00705     if (sscanf(tok, "%lf", &tf) == 0) {
00706         Vnm_print(2, "Nosh: Read non-float (%s) while parsing FGLEN \
00707 keyword!\n", tok);

```

```

00708         return VRC_WARNING;
00709     } else thee->fglen[0] = tf;
00710     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00711     if (sscanf(tok, "%lf", &tf) == 0) {
00712         Vnm_print(2, "Nosh: Read non-float (%s) while parsing FGLEN \
00713 keyword!\n", tok);
00714         return VRC_WARNING;
00715     } else thee->fglen[1] = tf;
00716     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00717     if (sscanf(tok, "%lf", &tf) == 0) {
00718         Vnm_print(2, "Nosh: Read non-float (%s) while parsing FGLEN \
00719 keyword!\n", tok);
00720         return VRC_WARNING;
00721     } else thee->fglen[2] = tf;
00722     thee->setfglen = 1;
00723     return VRC_SUCCESS;
00724
00725     ERROR1:
00726     Vnm_print(2, "parseMG: ran out of tokens!\n");
00727     return VRC_WARNING;
00728 }
00729
00730 VPRIVATE Vrc_Codes MGparm_parseCGCENT(MGparm *thee, Vio *sock) {
00731
00732     char tok[VMAX_BUFSIZE];
00733     double tf;
00734     int ti;
00735
00736     /* If the next token isn't a float, it probably means we want to
00737      * center on a molecule */
00738     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00739     if (sscanf(tok, "%lf", &tf) == 0) {
00740         if (Vstring_strcasecmp(tok, "mol") == 0) {
00741             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00742             if (sscanf(tok, "%d", &ti) == 0) {
00743                 Vnm_print(2, "Nosh: Read non-int (%s) while parsing \
00744 CGCENT MOL keyword!\n", tok);
00745                 return VRC_WARNING;
00746             } else {
00747                 thee->ccmeth = MCM_MOLECULE;
00748                 /* Subtract 1 here to convert user numbering (1, 2, 3, ...) into
00749                  array index */
00750                 thee->ccentmol = ti - 1;
00751             }
00752         } else {
00753             Vnm_print(2, "Nosh: Unexpected keyword (%s) while parsing \
00754 CGCENT!\n", tok);
00755             return VRC_WARNING;
00756         }
00757     } else {
00758         thee->ccenter[0] = tf;
00759         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00760         if (sscanf(tok, "%lf", &tf) == 0) {
00761             Vnm_print(2, "Nosh: Read non-float (%s) while parsing \
00762 CGCENT keyword!\n", tok);
00763             return VRC_WARNING;
00764         }
00765         thee->ccenter[1] = tf;
00766         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00767         if (sscanf(tok, "%lf", &tf) == 0) {
00768             Vnm_print(2, "Nosh: Read non-float (%s) while parsing \
00769 CGCENT keyword!\n", tok);
00770             return VRC_WARNING;
00771         }
00772         thee->ccenter[2] = tf;
00773     }
00774     thee->setcgcent = 1;
00775     return VRC_SUCCESS;
00776
00777     ERROR1:
00778     Vnm_print(2, "parseMG: ran out of tokens!\n");
00779     return VRC_WARNING;
00780 }
00781
00782 VPRIVATE Vrc_Codes MGparm_parseFGCENT(MGparm *thee, Vio *sock) {
00783
00784     char tok[VMAX_BUFSIZE];
00785     double tf;
00786     int ti;
00787
00788     /* If the next token isn't a float, it probably means we want to

```

```

00789      * center on a molecule */
00790      VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00791      if (sscanf(tok, "%lf", &tf) == 0) {
00792          if (Vstring_strcasecmp(tok, "mol") == 0) {
00793              VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00794              if (sscanf(tok, "%d", &ti) == 0) {
00795                  Vnm_print(2, "Nosh: Read non-int (%s) while parsing \
00796 FGCENT MOL keyword!\n", tok);
00797                  return VRC_WARNING;
00798              } else {
00799                  thee->fcmeth = MCM_MOLECULE;
00800                  /* Subtract 1 here to convert user numbering (1, 2, 3, ...) into
00801 array index */
00802                  thee->fcentmol = ti - 1;
00803              }
00804          } else {
00805              Vnm_print(2, "Nosh: Unexpected keyword (%s) while parsing \
00806 FGCENT!\n", tok);
00807              return VRC_WARNING;
00808          }
00809      } else {
00810          thee->fcenter[0] = tf;
00811          VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00812          if (sscanf(tok, "%lf", &tf) == 0) {
00813              Vnm_print(2, "Nosh: Read non-float (%s) while parsing \
00814 FGCENT keyword!\n", tok);
00815              return VRC_WARNING;
00816          }
00817          thee->fcenter[1] = tf;
00818          VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00819          if (sscanf(tok, "%lf", &tf) == 0) {
00820              Vnm_print(2, "Nosh: Read non-float (%s) while parsing \
00821 FGCENT keyword!\n", tok);
00822              return VRC_WARNING;
00823          }
00824          thee->fcenter[2] = tf;
00825      }
00826      thee->setfgcent = 1;
00827      return VRC_SUCCESS;
00828
00829  ERROR1:
00830      Vnm_print(2, "parseMG: ran out of tokens!\n");
00831      return VRC_WARNING;
00832 }
00833
00834 VPRIVATE Vrc_Codes MGparm_parsePDIME(MGparm *thee, Vio *sock) {
00835
00836     char tok[VMAX_BUFSIZE];
00837     int ti;
00838
00839     /* Read the number of grid points */
00840     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00841     if (sscanf(tok, "%d", &ti) == 0) {
00842         Vnm_print(2, "Nosh: Read non-integer (%s) while parsing PDIME \
00843 keyword!\n", tok);
00844         return VRC_WARNING;
00845     } else {
00846         thee->pdime[0] = ti;
00847     }
00848     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00849     if (sscanf(tok, "%d", &ti) == 0) {
00850         Vnm_print(2, "Nosh: Read non-integer (%s) while parsing PDIME \
00851 keyword!\n", tok);
00852         return VRC_WARNING;
00853     } else {
00854         thee->pdime[1] = ti;
00855     }
00856     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00857     if (sscanf(tok, "%d", &ti) == 0) {
00858         Vnm_print(2, "Nosh: Read non-integer (%s) while parsing PDIME \
00859 keyword!\n", tok);
00860         return VRC_WARNING;
00861     } else {
00862         thee->pdime[2] = ti;
00863     }
00864     thee->setpdime = 1;
00865     return VRC_SUCCESS;
00866
00867  ERROR1:
00868     Vnm_print(2, "parseMG: ran out of tokens!\n");
00869     return VRC_WARNING;

```

```

00870 }
00871
00872 VPRIVATE Vrc_Codes MGparm_parseOFRAC(MGparm *thee, Vio *sock) {
00873     char tok[VMAX_BUFSIZE];
00874     double tf;
00875
00876     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00877     if (sscanf(tok, "%lf", &tf) == 0) {
00878         Vnm_print(2, "Nosh: Read non-int (%s) while parsing OFRAC \
00879 keyword!\n", tok);
00880         return VRC_WARNING;
00881     }
00882     thee->ofrac = tf;
00883     thee->setofrac = 1;
00884     return VRC_SUCCESS;
00885
00886     VERROR1:
00887     Vnm_print(2, "parseMG: ran out of tokens!\n");
00888     return VRC_WARNING;
00889 }
00890
00891 VPRIVATE Vrc_Codes MGparm_parseASYNCR(MGparm *thee, Vio *sock) {
00892     char tok[VMAX_BUFSIZE];
00893     int ti;
00894
00895     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00896     if (sscanf(tok, "%i", &ti) == 0) {
00897         Vnm_print(2, "Nosh: Read non-integer (%s) while parsing ASYNCR \
00898 keyword!\n", tok);
00899         return VRC_WARNING;
00900     }
00901     thee->async = ti;
00902     thee->setasync = 1;
00903     return VRC_SUCCESS;
00904
00905     VERROR1:
00906     Vnm_print(2, "parseMG: ran out of tokens!\n");
00907     return VRC_WARNING;
00908 }
00909
00910 VPRIVATE Vrc_Codes MGparm_parseUSEAQUA(MGparm *thee, Vio *sock) {
00911     Vnm_print(0, "Nosh: parsed useaqua\n");
00912     thee->useAqua = 1;
00913     thee->setUseAqua = 1;
00914     return VRC_SUCCESS;
00915 }
00916
00917 VPUBLIC Vrc_Codes MGparm_parseToken(MGparm *thee, char tok[VMAX_BUFSIZE],
00918 Vio *sock) {
00919     if (thee == VNULL) {
00920         Vnm_print(2, "parseMG: got NULL thee!\n");
00921         return VRC_WARNING;
00922     }
00923     if (sock == VNULL) {
00924         Vnm_print(2, "parseMG: got NULL socket!\n");
00925         return VRC_WARNING;
00926     }
00927
00928     Vnm_print(0, "MGparm_parseToken: trying %s...\n", tok);
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     }

```

```

00951     } else if (Vstring_strcasecmp(tok, "fglen") == 0) {
00952         return MGparm_parseFGLEN(thee, sock);
00953     } else if (Vstring_strcasecmp(tok, "cgcent") == 0) {
00954         return MGparm_parseCGCENT(thee, sock);
00955     } else if (Vstring_strcasecmp(tok, "fgcent") == 0) {
00956         return MGparm_parseFGCENT(thee, sock);
00957     } else if (Vstring_strcasecmp(tok, "pdime") == 0) {
00958         return MGparm_parsePDIME(thee, sock);
00959     } else if (Vstring_strcasecmp(tok, "ofrac") == 0) {
00960         return MGparm_parseOFRAC(thee, sock);
00961     } else if (Vstring_strcasecmp(tok, "async") == 0) {
00962         return MGparm_parseASYNC(thee, sock);
00963     } else if (Vstring_strcasecmp(tok, "gamma") == 0) {
00964         return MGparm_parseGAMMA(thee, sock);
00965     } else if (Vstring_strcasecmp(tok, "useaqua") == 0) {
00966         return MGparm_parseUSEAQUA(thee, sock);
00967     } else {
00968         Vnm_print(2, "parseMG: Unrecognized keyword (%s)!\n", tok);
00969         return VRC_WARNING;
00970     }
00971
00972     return VRC_FAILURE;
00973 }
00974 }

```

10.29 src/generic/mgparm.h File Reference

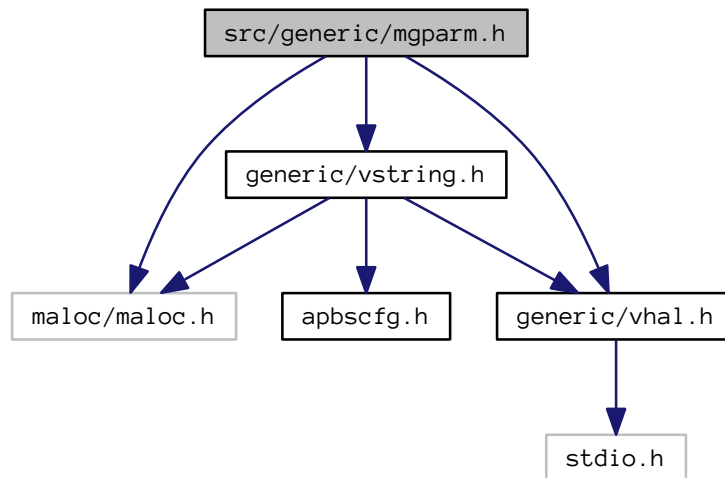
Contains declarations for class MGparm.

```

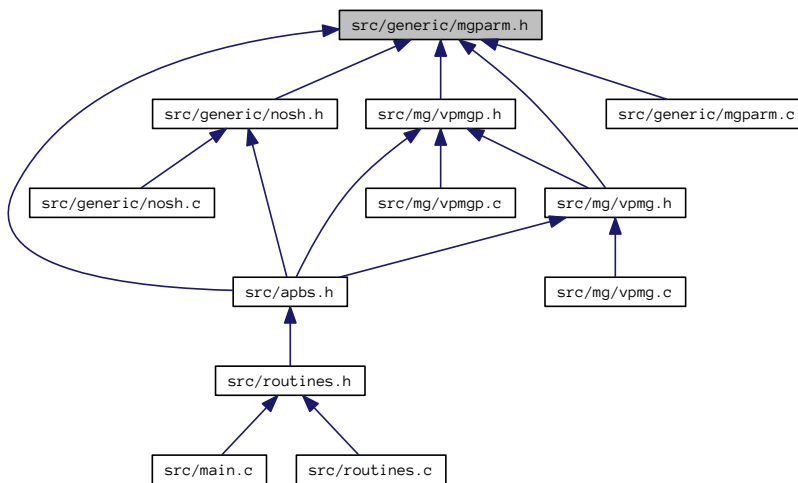
#include "malloc/malloc.h"
#include "generic/vhal.h"
#include "generic/vstring.h"

```

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)
- FORTTRAN stub to construct MGparm object.*
- VEXTERNC void [MGparm_dtor](#) ([MGparm](#) **thee)
- Object destructor.*
- VEXTERNC void [MGparm_dtor2](#) ([MGparm](#) *thee)
- FORTTRAN 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.*

10.29.1 Detailed Description

Contains declarations for class MGparm.

Version

\$Id\$

Author

Nathan A. Baker

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [mgparm.h](#).

10.30 mgparm.h

```

00001
00064 #ifndef _MGPARAM_H_
00065 #define _MGPARAM_H_
00066
00067 /* Generic header files */
00068 #include "malloc/malloc.h"
00069
00070 #include "generic/vhal.h"

```

```

00071 #include "generic/vstring.h"
00072
00077 enum eMGparm_CalcType {
00078     MCT_MANUAL=0,
00079     MCT_AUTO=1,
00080     MCT_PARALLEL=2,
00081     MCT_DUMMY=3,
00082     MCT_NONE=4
00083 };
00084
00089 typedef enum eMGparm_CalcType MGparm_CalcType;
00090
00095 enum eMGparm_CentMeth {
00096     MCM_POINT=0,
00097     MCM_MOLECULE=1,
00098     MCM_FOCUS=2
00099 };
00100
00105 typedef enum eMGparm_CentMeth MGparm_CentMeth;
00114 struct sMGparm {
00115     MGparm_CalcType type;
00116     int parsed;
00119     /* *** GENERIC PARAMETERS *** */
00120     int dime[3];
00121     int setdime;
00122     Vchrg_Meth chgm;
00123     int setchgm;
00124     Vchrg_Src chgs;
00127     /* *** TYPE 0 PARAMETERS (SEQUENTIAL MANUAL) *** */
00128     int nlev;
00130     int setnlev;
00131     double etol;
00132     int setetol;
00133     double grid[3];
00134     int setgrid;
00135     double glen[3];
00136     int setglen;
00137     MGparm_CentMeth cmeth;
00138     double center[3];
00146     int centmol;
00149     int setgcent;
00151     /* ***** TYPE 1 & 2 PARAMETERS (SEQUENTIAL & PARALLEL AUTO-FOCUS) *** */
00152     double cglen[3];
00153     int setcglen;
00154     double fglen[3];
00155     int setfglen;
00156     MGparm_CentMeth ccmeth;
00157     double ccenter[3];
00158     int ccentmol;
00161     int setcgcent;
00162     MGparm_CentMeth fcmeth;
00163     double fcenter[3];
00164     int fcentmol;
00167     int setfgcent;
00170     /* ***** TYPE 2 PARAMETERS (PARALLEL AUTO-FOCUS) ***** */
00171     double partDisjCenter[3];
00173     double partDisjLength[3];
00175     int partDisjOwnSide[6];
00178     int pdime[3];
00179     int setpdime;
00180     int proc_rank;
00181     int setrank;
00182     int proc_size;
00183     int setsize;
00184     double ofrac;
00185     int setofrac;
00186     int async;
00187     int setasync;
00189     int nonlintype;
00190     int setnonlintype;
00192     int method;
00193     int setmethod;
00195     int useAqua;
00196     int setUseAqua;
00197 };
00198
00203 typedef struct sMGparm MGparm;
00204
00211 VEXTERNC int MGparm_getNx(MGparm *thee);
00212

```

```

00219 VEXTERNC int MGparm_getNy(MGparm *thee);
00220
00227 VEXTERNC int MGparm_getNz(MGparm *thee);
00228
00235 VEXTERNC double MGparm_getHx(MGparm *thee);
00236
00243 VEXTERNC double MGparm_getHy(MGparm *thee);
00244
00251 VEXTERNC double MGparm_getHz(MGparm *thee);
00252
00259 VEXTERNC void MGparm_setCenterX(MGparm *thee, double x);
00260
00267 VEXTERNC void MGparm_setCenterY(MGparm *thee, double y);
00268
00275 VEXTERNC void MGparm_setCenterZ(MGparm *thee, double z);
00276
00283 VEXTERNC double MGparm_getCenterX(MGparm *thee);
00284
00291 VEXTERNC double MGparm_getCenterY(MGparm *thee);
00292
00299 VEXTERNC double MGparm_getCenterZ(MGparm *thee);
00300
00307 VEXTERNC MGparm* MGparm_ctor(MGparm_CalcType
type);
00308
00316 VEXTERNC Vrc_Codes MGparm_ctor2(MGparm *thee,
MGparm_CalcType type);
00317
00323 VEXTERNC void MGparm_dtor(MGparm **thee);
00324
00330 VEXTERNC void MGparm_dtor2(MGparm *thee);
00331
00338 VEXTERNC Vrc_Codes MGparm_check(MGparm *thee);
00339
00346 VEXTERNC void MGparm_copy(MGparm *thee, MGparm *parm);
00347
00357 VEXTERNC Vrc_Codes MGparm_parseToken(MGparm *thee, char tok[VMAX_BUFSIZE],
00358 Vio *sock);
00359
00360 #endif
00361

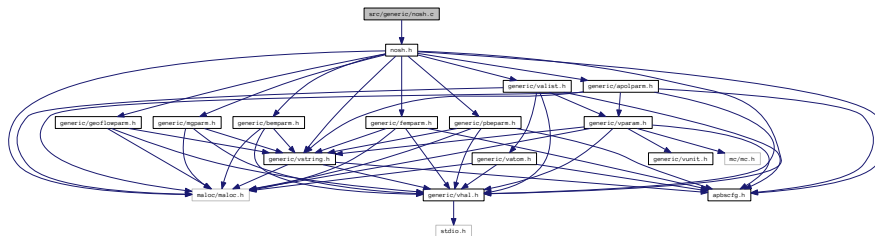
```

10.31 src/generic/nosh.c File Reference

Class NOsh methods.

```
#include "nosh.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_parseBEM** (**NOsh** *thee, Vio *sock, **NOsh_calc** *elec)
- VEXTERNC int **NOsh_parseGEOFLOW** (**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_setupCalcBEM** (**NOsh** *thee, **NOsh_calc** *elec)
- VPRIVATE int **NOsh_setupCalcGEOFLOW** (**NOsh** *thee, **NOsh_calc** *elec)
- VPRIVATE int **NOsh_setupCalcBEMMANUAL** (**NOsh** *thee, **NOsh_calc** *elec)
- VPRIVATE int **NOsh_setupCalcGEOFLOWMANUAL** (**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)
- FORTTRAN stub to construct NOsh.*
- VPUBLIC void [NOsh_dtor](#) ([NOsh](#) **thee)
- Object destructor.*
- VPUBLIC void [NOsh_dtor2](#) ([NOsh](#) *thee)
- FORTTRAN 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.*

10.31.1 Detailed Description

Class NOsh methods.

Author

Nathan Baker

Version

\$Id\$

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [nosh.c](#).

10.32 nosh.c

```

00001
00057 #include "nosh.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061
00062 VPRIVATE int NOsh_parseREAD(
00063     NOsh *thee,
00064     Vio *sock);
00065
00066 VPRIVATE int NOsh_parsePRINT(
00067     NOsh *thee,
00068     Vio *sock);
00069

```

```
00070 VPRIVATE int NOsh_parseELEC(  
00071     NOsh *thee,  
00072     Vio *sock  
00073 );  
00074  
00075 VPRIVATE int NOsh_parseAPOLAR(  
00076     NOsh *thee,  
00077     Vio *sock  
00078 );  
00079  
00080 VEXTERNC int NOsh_parseFEM(  
00081     NOsh *thee,  
00082     Vio *sock,  
00083     NOsh_calc *elec  
00084 );  
00085  
00086 VEXTERNC int NOsh_parseMG(  
00087     NOsh *thee,  
00088     Vio *sock,  
00089     NOsh_calc *elec  
00090 );  
00091  
00092 VEXTERNC int NOsh_parseBEM(  
00093     NOsh *thee,  
00094     Vio *sock,  
00095     NOsh_calc *elec  
00096 );  
00097  
00098 VEXTERNC int NOsh_parseGEOFLOW(  
00099     NOsh *thee,  
00100     Vio *sock,  
00101     NOsh_calc *elec  
00102 );  
00103  
00104 VEXTERNC int NOsh_parseAPOL(  
00105     NOsh *thee,  
00106     Vio *sock,  
00107     NOsh_calc *elec  
00108 );  
00109  
00110 VPRIVATE int NOsh_setupCalcMG(  
00111     NOsh *thee,  
00112     NOsh_calc *elec  
00113 );  
00114  
00115  
00116 VPRIVATE int NOsh_setupCalcMGAUTO(  
00117     NOsh *thee,  
00118     NOsh_calc *elec  
00119 );  
00120  
00121 VPRIVATE int NOsh_setupCalcMGMANUAL(  
00122     NOsh *thee,  
00123     NOsh_calc *elec  
00124 );  
00125  
00126 VPRIVATE int NOsh_setupCalcMGPARA(  
00127     NOsh *thee,  
00128     NOsh_calc *elec  
00129 );  
00130  
00131 VPRIVATE int NOsh_setupCalcFEM(  
00132     NOsh *thee,  
00133     NOsh_calc *elec  
00134 );  
00135  
00136 VPRIVATE int NOsh_setupCalcFEMANUAL(  
00137     NOsh *thee,  
00138     NOsh_calc *elec  
00139 );  
00140  
00141 VPRIVATE int NOsh_setupCalcBEM(  
00142     NOsh *thee,  
00143     NOsh_calc *elec  
00144 );  
00145  
00146 VPRIVATE int NOsh_setupCalcGEOFLOW(  
00147     NOsh *thee,  
00148     NOsh_calc *elec  
00149 );  
00150
```



```

00151 VPRIVATE int Nosh_setupCalcBEMMANUAL(
00152     Nosh *thee,
00153     Nosh_calc *elec
00154 );
00155
00156 VPRIVATE int Nosh_setupCalcGEOFLOWMANUAL(
00157     Nosh *thee,
00158     Nosh_calc *elec
00159 );
00160
00161 VPRIVATE int Nosh_setupCalcAPOL(
00162     Nosh *thee,
00163     Nosh_calc *elec
00164 );
00165
00166 #if !defined(VINLINE_NOSH)
00167
00168 VPUBLIC char* Nosh_getMolpath(Nosh *thee, int imol) {
00169     VASSERT(thee != VNULL);
00170     VASSERT(imol < thee->nmol);
00171     return thee->molpath[imol];
00172 }
00173 VPUBLIC char* Nosh_getDielXpath(Nosh *thee, int imol) {
00174     VASSERT(thee != VNULL);
00175     VASSERT(imol < thee->nmol);
00176     return thee->dielXpath[imol];
00177 }
00178 VPUBLIC char* Nosh_getDielYpath(Nosh *thee, int imol) {
00179     VASSERT(thee != VNULL);
00180     VASSERT(imol < thee->nmol);
00181     return thee->dielYpath[imol];
00182 }
00183 VPUBLIC char* Nosh_getDielZpath(Nosh *thee, int imol) {
00184     VASSERT(thee != VNULL);
00185     VASSERT(imol < thee->nmol);
00186     return thee->dielZpath[imol];
00187 }
00188 VPUBLIC char* Nosh_getKappapath(Nosh *thee, int imol) {
00189     VASSERT(thee != VNULL);
00190     VASSERT(imol < thee->nmol);
00191     return thee->kappapath[imol];
00192 }
00193 VPUBLIC char* Nosh_getPotpath(Nosh *thee, int imol) {
00194     VASSERT(thee != VNULL);
00195     VASSERT(imol < thee->nmol);
00196     return thee->potpath[imol];
00197 }
00198 VPUBLIC char* Nosh_getChargepath(Nosh *thee, int imol) {
00199     VASSERT(thee != VNULL);
00200     VASSERT(imol < thee->nmol);
00201     return thee->chargepath[imol];
00202 }
00203 VPUBLIC Nosh_calc* Nosh_getCalc(Nosh *thee, int icalc) {
00204     VASSERT(thee != VNULL);
00205     VASSERT(icalc < thee->ncalc);
00206     return thee->calc[icalc];
00207 }
00208 VPUBLIC int Nosh_getDielfmt(Nosh *thee, int i) {
00209     VASSERT(thee != VNULL);
00210     VASSERT(i < thee->ndiel);
00211     return (thee->dielfmt[i]);
00212 }
00213 VPUBLIC int Nosh_getKappafmt(Nosh *thee, int i) {
00214     VASSERT(thee != VNULL);
00215     VASSERT(i < thee->nkappa);
00216     return (thee->kappafmt[i]);
00217 }
00218 VPUBLIC int Nosh_getPotfmt(Nosh *thee, int i) {
00219     VASSERT(thee != VNULL);
00220     VASSERT(i < thee->npot);
00221     return (thee->potfmt[i]);
00222 }
00223 VPUBLIC int Nosh_getChargefmt(Nosh *thee, int i) {
00224     VASSERT(thee != VNULL);
00225     VASSERT(i < thee->ncharge);
00226     return (thee->chargefmt[i]);
00227 }
00228
00229
00230 #endif /* if !defined(VINLINE_NOSH) */
00231

```

```

00232 VPUBLIC NOsh_PrintType NOsh_printWhat(NOsh *thee, int iprint) {
00233     VASSERT(thee != VNULL);
00234     VASSERT(iprint < thee->nprint);
00235     return thee->printwhat[iprint];
00236 }
00237
00238 VPUBLIC int NOsh_printNarg(NOsh *thee, int iprint) {
00239     VASSERT(thee != VNULL);
00240     VASSERT(iprint < thee->nprint);
00241     return thee->printnarg[iprint];
00242 }
00243
00244 VPUBLIC int NOsh_elec2calc(NOsh *thee, int icalc) {
00245     VASSERT(thee != VNULL);
00246     VASSERT(icalc < thee->ncalc);
00247     return thee->elec2calc[icalc];
00248 }
00249
00250 VPUBLIC int NOsh_apol2calc(NOsh *thee, int icalc) {
00251     VASSERT(thee != VNULL);
00252     VASSERT(icalc < thee->ncalc);
00253     return thee->apol2calc[icalc];
00254 }
00255
00256 VPUBLIC char* NOsh_elecname(NOsh *thee, int ielec) {
00257     VASSERT(thee != VNULL);
00258     VASSERT(ielec < thee->nelec + 1);
00259     return thee->elecname[ielec];
00260 }
00261
00262 VPUBLIC int NOsh_printOp(NOsh *thee, int iprint, int iarg) {
00263     VASSERT(thee != VNULL);
00264     VASSERT(iprint < thee->nprint);
00265     VASSERT(iarg < thee->printnarg[iprint]);
00266     return thee->printop[iprint][iarg];
00267 }
00268
00269 VPUBLIC int NOsh_printCalc(NOsh *thee, int iprint, int iarg) {
00270     VASSERT(thee != VNULL);
00271     VASSERT(iprint < thee->nprint);
00272     VASSERT(iarg < thee->printnarg[iprint]);
00273     return thee->printcalc[iprint][iarg];
00274 }
00275
00276 VPUBLIC NOsh* NOsh_ctor(int rank, int size) {
00277     /* Set up the structure */
00278     NOsh *thee = VNULL;
00279     thee = (NOsh*)Vmem_malloc(VNULL, 1, sizeof(NOsh) );
00280     VASSERT( thee != VNULL);
00281     VASSERT( NOsh_ctor2(thee, rank, size) );
00282     return thee;
00283 }
00284
00285
00286
00287 VPUBLIC int NOsh_ctor2(NOsh *thee, int rank, int size) {
00288     int i;
00289
00290     if (thee == VNULL) return 0;
00291
00292     thee->proc_rank = rank;
00293     thee->proc_size = size;
00294
00295     thee->ispara = 0;
00296     thee->parsed = 0;
00297
00298     thee->nmol = 0;
00299     thee->gotparm = 0;
00300     thee->ncharge = 0;
00301     thee->ndiel = 0;
00302     thee->nkappa = 0;
00303     thee->npot = 0;
00304     thee->nprint = 0;
00305
00306     for (i=0; i<NOSH_MAXCALC; i++) {
00307         thee->calc[i] = VNULL;
00308         thee->elec[i] = VNULL;
00309         thee->apol[i] = VNULL;
00310     }
00311     for (i=0; i<NOSH_MAXMOL; i++) {

```

```

00313     thee->alist[i] = VNULL;
00314 }
00315 thee->ncalc = 0;
00316 thee->nelec = 0;
00317 thee->napol = 0;
00318
00319 return 1;
00320 }
00321
00322 VPUBLIC void NOsh_dtor(NOsh **thee) {
00323     if ((*thee) != VNULL) {
00324         NOsh_dtor2(*thee);
00325         Vmem_free(VNULL, 1, sizeof(NOsh), (void **)thee);
00326         (*thee) = VNULL;
00327     }
00328 }
00329
00330 VPUBLIC void NOsh_dtor2(NOsh *thee) {
00331     int i;
00332
00333     if (thee != VNULL) {
00334         for (i=0; i<(thee->ncalc); i++) NOsh_calc_dtor(&(thee->
00335 calc[i]));
00336         for (i=0; i<(thee->nelec); i++) NOsh_calc_dtor(&(thee->
00337 elec[i]));
00337         for (i=0; i<(thee->napol); i++) NOsh_calc_dtor(&(thee->
00338 apol[i]));
00339     }
00340 }
00341
00342 VPUBLIC NOsh_calc* NOsh_calc_ctor(
00343     NOsh_CalcType calctype
00344 ) {
00345     NOsh_calc *thee;
00346     thee = (NOsh_calc *)Vmem_malloc(VNULL, 1, sizeof(NOsh_calc));
00347     thee->calctype = calctype;
00348
00349     thee->mgparm = VNULL;
00350     thee->femparm = VNULL;
00351     thee->apolparm = VNULL;
00352     thee->bemparm = VNULL;
00353     thee->geoflowparm = VNULL;
00354
00355     switch (calctype) {
00356     case NCT_MG:
00357         thee->mgparm = MGparm_ctor(MCT_NONE);
00358         break;
00359     case NCT_FEM:
00360         thee->femparm = FEMparm_ctor(FCT_NONE);
00361         break;
00362     case NCT_APOL:
00363         thee->apolparm = APOLparm_ctor();
00364         break;
00365     case NCT_BEM:
00366         thee->bemparm = BEMparm_ctor(BCT_MANUAL);
00367         break;
00368     case NCT_GEOFLOW:
00369         thee->geoflowparm = GEOFLOWparm_ctor(GFCT_NONE);
00370         thee->apolparm = APOLparm_ctor();
00371         break;
00372     default:
00373         Vnm_print(2, "NOsh_calc_ctor: unknown calculation type (%d)!\n",
00374 calctype);
00375         VASSERT(0);
00376     }
00377     thee->pbeparm = PBEParm_ctor();
00378
00379     return thee;
00380 }
00381
00382 VPUBLIC void NOsh_calc_dtor(
00383     NOsh_calc **thee
00384 ) {
00385
00386     NOsh_calc *calc = VNULL;
00387     calc = *thee;
00388     if (calc == VNULL) return;
00389
00390     switch (calc->calctype) {

```

```

00391         case NCT_MG:
00392             MGparm_dtor(&(calc->mgparm));
00393             break;
00394         case NCT_FEM:
00395             FEMparm_dtor(&(calc->femparm));
00396             break;
00397         case NCT_APOL:
00398             APOLparm_dtor(&(calc->apolparm));
00399             break;
00400         case NCT_BEM:
00401             BEMparm_dtor(&(calc->bemparm));
00402             break;
00403         case NCT_GEOFLOW:
00404             GEOFLOWparm_dtor(&(calc->geoflowparm));
00405             APOLparm_dtor(&(calc->apolparm));
00406             break;
00407         default:
00408             Vnm_print(2, "Nosh_calc_ctor: unknown calculation type (%d)!\n",
00409                     calc->calctype);
00410             VASSERT(0);
00411     }
00412     PBEparm_dtor(&(calc->pbeparm));
00413
00414     Vmem_free(VNULL, 1, sizeof(NOsh_calc), (void **)thee);
00415     calc = VNULL;
00416 }
00417 }
00418
00419 VPUBLIC int NOsh_calc_copy(
00420     NOsh_calc *thee,
00421     NOsh_calc *source
00422 ) {
00423
00424     VASSERT(thee != VNULL);
00425     VASSERT(source != VNULL);
00426     VASSERT(thee->calctype == source->calctype);
00427     if (source->mgparm != VNULL)
00428         MGparm_copy(thee->mgparm, source->mgparm);
00429     if (source->femparm != VNULL)
00430         FEMparm_copy(thee->femparm, source->femparm);
00431     if (source->bemparm != VNULL)
00432         BEMparm_copy(thee->bemparm, source->bemparm);
00433     if (source->pbeparm != VNULL)
00434         PBEparm_copy(thee->pbeparm, source->pbeparm);
00435     if (source->apolparm != VNULL)
00436         APOLparm_copy(thee->apolparm, source->apolparm);
00437
00438     return 1;
00439 }
00440 }
00441
00442 VPUBLIC int NOsh_parseInputFile(
00443     NOsh *thee,
00444     char *filename
00445 ) {
00446
00447     Vio *sock;
00448     int rc;
00449
00450     sock = Vio_ctor("FILE", "ASC", VNULL, filename, "r");
00451     rc = NOsh_parseInput(thee, sock);
00452     Vio_dtor(&sock);
00453
00454     return rc;
00455 }
00456
00457 VPUBLIC int NOsh_parseInput(
00458     NOsh *thee,
00459     Vio *sock
00460 ) {
00461
00462     char *MCwhiteChars = " =,;\t\r\n";
00463     char *MCcommChars = "#%";
00464     char tok[VMAX_BUFSIZE];
00465
00466     if (thee == VNULL) {
00467         Vnm_print(2, "NOsh_parseInput: Got NULL thee!\n");
00468         return 0;
00469     }
00470
00471     if (sock == VNULL) {

```

```

00472     Vnm_print(2, "Nosh_parseInput:  Got pointer to NULL socket!\n");
00473     Vnm_print(2, "Nosh_parseInput:  The specified input file was not found!\n");
00474     return 0;
00475 }
00476
00477 if (thee->parsed) {
00478     Vnm_print(2, "Nosh_parseInput:  Already parsed an input file!\n");
00479     return 0;
00480 }
00481
00482 if (Vio_accept(sock, 0) < 0) {
00483     Vnm_print(2, "Nosh_parseInput:  Problem reading from socket!\n");
00484     return 0;
00485 }
00486
00487 /* Set up the whitespace and comment character definitions */
00488 Vio_setWhiteChars(sock, MCwhiteChars);
00489 Vio_setCommChars(sock, MCcommChars);
00490
00491 /* We parse the file until we run out of tokens */
00492 Vnm_print(0, "Nosh_parseInput:  Starting file parsing...\n");
00493 while (Vio_scanf(sock, "%s", tok) == 1) {
00494     /* At the highest level, we look for keywords that indicate functions like:
00495
00496     read => Read in a molecule file
00497     elec => Do an electrostatics calculation
00498     print => Print some results
00499     apolar => do a non-polar calculation
00500     quit => Quit
00501
00502     These cause the code to go to a lower-level parser routine which
00503     handles keywords specific to the particular function.  Each
00504     lower-level parser routine then returns when it hits the "end"
00505     keyword.  Due to this simple layout, no nesting of these "function"
00506     sections is allowed.
00507     */
00508     if (Vstring_strcasecmp(tok, "read") == 0) {
00509         Vnm_print(0, "Nosh: Parsing READ section\n");
00510         if (!Nosh_parseREAD(thee, sock)) return 0;
00511         Vnm_print(0, "Nosh: Done parsing READ section \
00512 (nmol=%d, ndiel=%d, nkappa=%d, ncharge=%d, npot=%d)\n", thee->nmol, thee->
ndiel,
00513                 thee->nkappa, thee->ncharge, thee->npot);
00514     } else if (Vstring_strcasecmp(tok, "print") == 0) {
00515         Vnm_print(0, "Nosh: Parsing PRINT section\n");
00516         if (!Nosh_parsePRINT(thee, sock)) return 0;
00517         Vnm_print(0, "Nosh: Done parsing PRINT section\n");
00518     } else if (Vstring_strcasecmp(tok, "elec") == 0) {
00519         Vnm_print(0, "Nosh: Parsing ELEC section\n");
00520         if (!Nosh_parseELEC(thee, sock)) return 0;
00521         Vnm_print(0, "Nosh: Done parsing ELEC section (nelec = %d)\n",
thee->nelec);
00522     } else if (Vstring_strcasecmp(tok, "apolar") == 0) {
00523         Vnm_print(0, "Nosh: Parsing APOLAR section\n");
00524         if (!Nosh_parseAPOLAR(thee, sock)) return 0;
00525         Vnm_print(0, "Nosh: Done parsing APOLAR section (nelec = %d)\n",
thee->nelec);
00526     } else if (Vstring_strcasecmp(tok, "quit") == 0) {
00527         Vnm_print(0, "Nosh: Done parsing file (got QUIT)\n");
00528         break;
00529     } else {
00530         Vnm_print(2, "Nosh_parseInput: Ignoring undefined keyword %s!\n", tok);
00531     }
00532 }
00533
00534 }
00535
00536 thee->parsed = 1;
00537 return 1;
00538 }
00539 }
00540
00541 VPRIVATE int Nosh_parseREAD_MOL(Nosh *thee, Vio *sock) {
00542
00543     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00544     Nosh_MolFormat molfmt;
00545
00546     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00547     if (Vstring_strcasecmp(tok, "pqr") == 0) {
00548         molfmt = NMF_PQR;
00549         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00550         if (tok[0]=='\"') {
00551             strcpy(strnew, "");

```

```

00552         while (tok[strlen(tok)-1] != '\0') {
00553             strcat(str, tok);
00554             strcat(str, " ");
00555             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00556         }
00557         strcat(str, tok);
00558         strncpy(strnew, str+1, strlen(str)-2);
00559         strcpy(tok, strnew);
00560     }
00561     Vnm_print(0, "Nosh: Storing molecule %d path %s\n",
00562             thee->nmol, tok);
00563     thee->molfmt[thee->nmol] = molfmt;
00564     strncpy(thee->molpath[thee->nmol], tok, VMAX_ARGLEN);
00565     (thee->nmol)++;
00566 } else if (Vstring_strcasecmp(tok, "pdb") == 0) {
00567     molfmt = NMF_PDB;
00568     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00569     if (tok[0]!='\0') {
00570         strcpy(strnew, "");
00571         while (tok[strlen(tok)-1] != '\0') {
00572             strcat(str, tok);
00573             strcat(str, " ");
00574             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00575         }
00576         strcat(str, tok);
00577         strncpy(strnew, str+1, strlen(str)-2);
00578         strcpy(tok, strnew);
00579     }
00580     Vnm_print(0, "Nosh: Storing molecule %d path %s\n",
00581             thee->nmol, tok);
00582     thee->molfmt[thee->nmol] = molfmt;
00583     strncpy(thee->molpath[thee->nmol], tok, VMAX_ARGLEN);
00584     (thee->nmol)++;
00585 } else if (Vstring_strcasecmp(tok, "xml") == 0) {
00586     molfmt = NMF_XML;
00587     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00588     if (tok[0]!='\0') {
00589         strcpy(strnew, "");
00590         while (tok[strlen(tok)-1] != '\0') {
00591             strcat(str, tok);
00592             strcat(str, " ");
00593             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00594         }
00595         strcat(str, tok);
00596         strncpy(strnew, str+1, strlen(str)-2);
00597         strcpy(tok, strnew);
00598     }
00599     Vnm_print(0, "Nosh: Storing molecule %d path %s\n",
00600             thee->nmol, tok);
00601     thee->molfmt[thee->nmol] = molfmt;
00602     strncpy(thee->molpath[thee->nmol], tok, VMAX_ARGLEN);
00603     (thee->nmol)++;
00604 } else {
00605     Vnm_print(2, "Nosh_parseREAD: Ignoring undefined mol format \
00606 %s!\n", tok);
00607 }
00608
00609 return 1;
00610
00611
00612 VERROR1:
00613     Vnm_print(2, "Nosh_parseREAD_MOL: Ran out of tokens while parsing READ section!\n");
00614     return 0;
00615 }
00616
00617
00618 VPRIVATE int Nosh_parseREAD_PARM(NOsh *thee, Vio *sock) {
00619     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00620     NOsh_ParmFormat parmfmt;
00621
00622     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00623     if (Vstring_strcasecmp(tok, "flat") == 0) {
00624         parmfmt = NPF_FLAT;
00625         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00626         if (tok[0]!='\0') {
00627             strcpy(strnew, "");
00628             while (tok[strlen(tok)-1] != '\0') {
00629                 strcat(str, tok);
00630                 strcat(str, " ");
00631                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00632             }

```

```

00633         }
00634         strcat(str, tok);
00635         strncpy(strnew, str+1, strlen(str)-2);
00636         strcpy(tok, strnew);
00637     }
00638     if (thee->gotparm) {
00639         Vnm_print(2, "Nosh: Hey! You already specified a parameterfile (%s)!\n", thee->
parmpath);
00640         Vnm_print(2, "Nosh: I'm going to ignore this one (%s)!\n", tok);
00641     } else {
00642         thee->parmfmt = parmfmt;
00643         thee->gotparm = 1;
00644         strncpy(thee->parmpath, tok, VMAX_ARGLEN);
00645     }
00646 } else if(Vstring_strcasecmp(tok, "xml") == 0) {
00647     parmfmt = NPF_XML;
00648     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00649     if (tok[0]=='') {
00650         strcpy(strnew, "");
00651         while (tok[strlen(tok)-1] != '') {
00652             strcat(str, tok);
00653             strcat(str, " ");
00654             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00655         }
00656         strcat(str, tok);
00657         strncpy(strnew, str+1, strlen(str)-2);
00658         strcpy(tok, strnew);
00659     }
00660     if (thee->gotparm) {
00661         Vnm_print(2, "Nosh: Hey! You already specified a parameterfile (%s)!\n", thee->
parmpath);
00662         Vnm_print(2, "Nosh: I'm going to ignore this one (%s)!\n", tok);
00663     } else {
00664         thee->parmfmt = parmfmt;
00665         thee->gotparm = 1;
00666         strncpy(thee->parmpath, tok, VMAX_ARGLEN);
00667     }
00668 } else {
00669     Vnm_print(2, "Nosh_parseREAD: Ignoring undefined parm format \
00671 %s!\n", tok);
00672 }
00673 return 1;
00674
00675
00676 ERROR1:
00677     Vnm_print(2, "Nosh_parseREAD_PARM: Ran out of tokens while parsing READ section!\n");
00678     return 0;
00679 }
00680
00681 VPRIVATE int Nosh_parseREAD_DIEL(NOsh *thee, Vio *sock) {
00682     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00683     Vdata_Format dielfmt;
00684     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00685     if (Vstring_strcasecmp(tok, "dx") == 0) {
00686         dielfmt = VDF_DX;
00687     } else if (Vstring_strcasecmp(tok, "gz") == 0) {
00688         dielfmt = VDF_GZ;
00689     } else {
00690         Vnm_print(2, "Nosh_parseREAD: Ignoring undefined format \
00694 %s!\n", tok);
00695         return VRC_FAILURE;
00696     }
00697
00698     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00699     if (tok[0]=='') {
00700         strcpy(strnew, "");
00701         while (tok[strlen(tok)-1] != '') {
00702             strcat(str, tok);
00703             strcat(str, " ");
00704             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00705         }
00706         strcat(str, tok);
00707         strncpy(strnew, str+1, strlen(str)-2);
00708         strcpy(tok, strnew);
00709     }
00710     Vnm_print(0, "Nosh: Storing x-shifted dielectric map %d path \
00711 %s\n", thee->ndiel, tok);

```

```

00712     strncpy(thee->dielXpath[thee->ndiel], tok, VMAX_ARGLEN);
00713     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00714     Vnm_print(0, "Nosh: Storing y-shifted dielectric map %d path \
00715               %s\n", thee->ndiel, tok);
00716     strncpy(thee->dielYpath[thee->ndiel], tok, VMAX_ARGLEN);
00717     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00718     Vnm_print(0, "Nosh: Storing z-shifted dielectric map %d path \
00719               %s\n", thee->ndiel, tok);
00720     strncpy(thee->dielZpath[thee->ndiel], tok, VMAX_ARGLEN);
00721     thee->dielfmt[thee->ndiel] = dielfmt;
00722     (thee->ndiel)++;
00723
00724     return 1;
00725
00726 VERROR1:
00727     Vnm_print(2, "Nosh_parseREAD_DIEL: Ran out of tokens while parsing READ \
00728 section!\n");
00729     return 0;
00730
00731 }
00732
00733 VPRIVATE int Nosh_parseREAD_KAPPA(Nosh *thee, Vio *sock) {
00734
00735     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00736     Vdata_Format kappafmt;
00737
00738     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00739     if (Vstring_strcasecmp(tok, "dx") == 0) {
00740         kappafmt = VDF_DX;
00741     } else if (Vstring_strcasecmp(tok, "gz") == 0) {
00742         kappafmt = VDF_GZ;
00743     } else {
00744         Vnm_print(2, "Nosh_parseREAD: Ignoring undefined format \
00745               %s!\n", tok);
00746         return VRC_FAILURE;
00747     }
00748
00749     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00750     if (tok[0]=='"') {
00751         strcpy(strnew, "");
00752         while (tok[strlen(tok)-1] != '"') {
00753             strcat(str, tok);
00754             strcat(str, " ");
00755             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00756         }
00757         strcat(str, tok);
00758         strncpy(strnew, str+1, strlen(str)-2);
00759         strcpy(tok, strnew);
00760     }
00761     Vnm_print(0, "Nosh: Storing kappa map %d path %s\n",
00762               thee->nkappa, tok);
00763     thee->kappafmt[thee->nkappa] = kappafmt;
00764     strncpy(thee->kappapath[thee->nkappa], tok, VMAX_ARGLEN);
00765     (thee->nkappa)++;
00766
00767     return 1;
00768
00769 VERROR1:
00770     Vnm_print(2, "Nosh_parseREAD: Ran out of tokens while parsing READ \
00771 section!\n");
00772     return 0;
00773
00774 }
00775
00776 VPRIVATE int Nosh_parseREAD_POTENTIAL(Nosh *thee, Vio *sock) {
00777
00778     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00779     Vdata_Format potfmt;
00780
00781     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00782     if (Vstring_strcasecmp(tok, "dx") == 0) {
00783         potfmt = VDF_DX;
00784     } else if (Vstring_strcasecmp(tok, "gz") == 0) {
00785         potfmt = VDF_GZ;
00786     } else {
00787         Vnm_print(2, "Nosh_parseREAD: Ignoring undefined format \
00788               %s!\n", tok);
00789         return VRC_FAILURE;
00790     }
00791
00792     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);

```



```

00793     if (tok[0]==' ') {
00794         strcpy(strnew, "");
00795         while (tok[strlen(tok)-1] != ' ') {
00796             strcat(str, tok);
00797             strcat(str, " ");
00798             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00799         }
00800         strcat(str, tok);
00801         strncpy(strnew, str+1, strlen(str)-2);
00802         strcpy(tok, strnew);
00803     }
00804     Vnm_print(0, "Nosh: Storing potential map %d path %s\n",
00805             thee->npot, tok);
00806     thee->potfmt[thee->npot] = potfmt;
00807     strncpy(thee->potpath[thee->npot], tok, VMAX_ARGLEN);
00808     (thee->npot)++;
00809
00810     return 1;
00811
00812 ERROR1:
00813     Vnm_print(2, "Nosh_parseREAD: Ran out of tokens while parsing READ \
00814             section!\n");
00815     return 0;
00816 }
00817
00818 VPRIVATE int Nosh_parseREAD_CHARGE(Nosh *thee, Vio *sock) {
00819
00820     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00821     Vdata_Format chargefmt;
00822
00823     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00824     if (Vstring_strcasecmp(tok, "dx") == 0) {
00825         chargefmt = VDF_DX;
00826     } else if (Vstring_strcasecmp(tok, "gz") == 0) {
00827         chargefmt = VDF_GZ;
00828     } else {
00829         Vnm_print(2, "Nosh_parseREAD: Ignoring undefined format \
00830             %s!\n", tok);
00831         return VRC_FAILURE;
00832     }
00833
00834     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00835     if (tok[0]==' ') {
00836         strcpy(strnew, "");
00837         while (tok[strlen(tok)-1] != ' ') {
00838             strcat(str, tok);
00839             strcat(str, " ");
00840             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00841         }
00842         strcat(str, tok);
00843         strncpy(strnew, str+1, strlen(str)-2);
00844         strcpy(tok, strnew);
00845     }
00846     Vnm_print(0, "Nosh: Storing charge map %d path %s\n",
00847             thee->ncharge, tok);
00848     thee->chargefmt[thee->ncharge] = chargefmt;
00849     strncpy(thee->chargepath[thee->ncharge], tok, VMAX_ARGLEN);
00850     (thee->ncharge)++;
00851
00852     return 1;
00853
00854 ERROR1:
00855     Vnm_print(2, "Nosh_parseREAD: Ran out of tokens while parsing READ \
00856             section!\n");
00857     return 0;
00858 }
00859
00860
00861 VPRIVATE int Nosh_parseREAD_MESH(Nosh *thee, Vio *sock) {
00862
00863     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00864     Vdata_Format meshfmt;
00865
00866     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00867     if (Vstring_strcasecmp(tok, "mcsf") == 0) {
00868         meshfmt = VDF_MCSF;
00869         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00870         if (tok[0]==' ') {
00871             strcpy(strnew, "");
00872             while (tok[strlen(tok)-1] != ' ') {

```

```

00874         strcat(str, tok);
00875         strcat(str, " ");
00876         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00877     }
00878     strcat(str, tok);
00879     strncpy(strnew, str+1, strlen(str)-2);
00880     strcpy(tok, strnew);
00881 }
00882 Vnm_print(0, "Nosh: Storing mesh %d path %s\n",
00883     thee->nmesh, tok);
00884 thee->meshfmt[thee->nmesh] = meshfmt;
00885 strncpy(thee->meshpath[thee->nmesh], tok, VMAX_ARGLEN);
00886 (thee->nmesh)++;
00887 } else {
00888     Vnm_print(2, "Nosh_parseREAD: Ignoring undefined mesh format \
00889         %s!\n", tok);
00890 }
00891
00892 return 1;
00893
00894 VERROR1:
00895 Vnm_print(2, "Nosh_parseREAD: Ran out of tokens while parsing READ \
00896     section!\n");
00897 return 0;
00898 }
00899
00900
00901
00902 VPRIVATE int NOsh_parseREAD(NOsh *thee, Vio *sock) {
00903
00904     char tok[VMAX_BUFSIZE];
00905
00906     if (thee == VNULL) {
00907         Vnm_print(2, "Nosh_parseREAD: Got NULL thee!\n");
00908         return 0;
00909     }
00910
00911     if (sock == VNULL) {
00912         Vnm_print(2, "Nosh_parseREAD: Got pointer to NULL socket!\n");
00913         return 0;
00914     }
00915
00916     if (thee->parsed) {
00917         Vnm_print(2, "Nosh_parseREAD: Already parsed an input file!\n");
00918         return 0;
00919     }
00920
00921     /* Read until we run out of tokens (bad) or hit the "END" keyword (good) */
00922     while (Vio_scanf(sock, "%s", tok) == 1) {
00923         if (Vstring_strcasecmp(tok, "end") == 0) {
00924             Vnm_print(0, "Nosh: Done parsing READ section\n");
00925             return 1;
00926         } else if (Vstring_strcasecmp(tok, "mol") == 0) {
00927             NOsh_parseREAD_MOL(thee, sock);
00928         } else if (Vstring_strcasecmp(tok, "parm") == 0) {
00929             NOsh_parseREAD_PARM(thee, sock);
00930         } else if (Vstring_strcasecmp(tok, "diel") == 0) {
00931             NOsh_parseREAD_DIEL(thee, sock);
00932         } else if (Vstring_strcasecmp(tok, "kappa") == 0) {
00933             NOsh_parseREAD_KAPPA(thee, sock);
00934         } else if (Vstring_strcasecmp(tok, "pot") == 0) {
00935             NOsh_parseREAD_POTENTIAL(thee, sock);
00936         } else if (Vstring_strcasecmp(tok, "charge") == 0) {
00937             NOsh_parseREAD_CHARGE(thee, sock);
00938         } else if (Vstring_strcasecmp(tok, "mesh") == 0) {
00939             NOsh_parseREAD_MESH(thee, sock);
00940         } else {
00941             Vnm_print(2, "Nosh_parseREAD: Ignoring undefined keyword %s!\n",
00942                 tok);
00943         }
00944     }
00945
00946     /* We ran out of tokens! */
00947     Vnm_print(2, "Nosh_parseREAD: Ran out of tokens while parsing READ \
00948         section!\n");
00949     return 0;
00950 }
00951
00952
00953 VPRIVATE int NOsh_parsePRINT(NOsh *thee, Vio *sock) {
00954

```

```

00955     char tok[VMAX_BUFSIZE];
00956     char name[VMAX_BUFSIZE];
00957     int ti, idx, expect, ielec, iapol;
00958
00959     if (thee == VNULL) {
00960         Vnm_print(2, "Nosh_parsePRINT: Got NULL thee!\n");
00961         return 0;
00962     }
00963
00964     if (sock == VNULL) {
00965         Vnm_print(2, "Nosh_parsePRINT: Got pointer to NULL socket!\n");
00966         return 0;
00967     }
00968
00969     if (thee->parsed) {
00970         Vnm_print(2, "Nosh_parsePRINT: Already parsed an input file!\n");
00971         return 0;
00972     }
00973
00974     idx = thee->nprint;
00975     if (thee->nprint >= NOSH_MAXPRINT) {
00976         Vnm_print(2, "Nosh_parsePRINT: Exceeded max number (%d) of PRINT \
00977 sections\n",
00978                 NOSH_MAXPRINT);
00979         return 0;
00980     }
00981
00982
00983     /* The first thing we read is the thing we want to print */
00984     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00985     if (Vstring_strcasecmp(tok, "energy") == 0) {
00986         thee->printwhat[idx] = NPT_ENERGY;
00987         thee->printnarg[idx] = 0;
00988     } else if (Vstring_strcasecmp(tok, "force") == 0) {
00989         thee->printwhat[idx] = NPT_FORCE;
00990         thee->printnarg[idx] = 0;
00991     } else if (Vstring_strcasecmp(tok, "elecEnergy") == 0) {
00992         thee->printwhat[idx] = NPT_ELECENERGY;
00993         thee->printnarg[idx] = 0;
00994     } else if (Vstring_strcasecmp(tok, "elecForce") == 0) {
00995         thee->printwhat[idx] = NPT_ELECFORCE;
00996         thee->printnarg[idx] = 0;
00997     } else if (Vstring_strcasecmp(tok, "apolEnergy") == 0) {
00998         thee->printwhat[idx] = NPT_APOLENERGY;
00999         thee->printnarg[idx] = 0;
01000     } else if (Vstring_strcasecmp(tok, "apolForce") == 0) {
01001         thee->printwhat[idx] = NPT_APOLFORCE;
01002         thee->printnarg[idx] = 0;
01003     } else {
01004         Vnm_print(2, "Nosh_parsePRINT: Undefined keyword %s while parsing \
01005 PRINT section!\n", tok);
01006         return 0;
01007     }
01008
01009     expect = 0; /* We first expect a calculation ID (0) then an op (1) */
01010
01011     /* Read until we run out of tokens (bad) or hit the "END" keyword (good) */
01012     while (Vio_scanf(sock, "%s", tok) == 1) {
01013
01014         /* The next thing we read is either END or an ARG OP ARG statement */
01015         if (Vstring_strcasecmp(tok, "end") == 0) {
01016             if (expect != 0) {
01017                 (thee->nprint)++;
01018                 (thee->printnarg[idx])++;
01019                 Vnm_print(0, "Nosh: Done parsing PRINT section\n");
01020                 return 1;
01021             } else {
01022                 Vnm_print(2, "Nosh_parsePRINT: Got premature END to PRINT!\n");
01023                 return 0;
01024             }
01025         } else {
01026
01027             /* Grab a calculation ID */
01028             if ((sscanf(tok, "%d", &ti) == 1) &&
01029                 (Vstring_isdigit(tok) == 1)) {
01030                 if (expect == 0) {
01031                     thee->printcalc[idx][thee->printnarg[idx]] = ti-1;
01032                     expect = 1;
01033                 } else {
01034                     Vnm_print(2, "Nosh_parsePRINT: Syntax error in PRINT \
01035 section while reading %s!\n", tok);

```

```

01036         return 0;
01037     }
01038     /* Grab addition operation */
01039     } else if (Vstring_strcasecmp(tok, "+") == 0) {
01040         if (expect == 1) {
01041             thee->printop[idx][thee->printnarg[idx]] = 0;
01042             (thee->printnarg[idx])++;
01043             expect = 0;
01044             if (thee->printnarg[idx] >= NOSH_MAXPOP) {
01045                 Vnm_print(2, "Nosh_parsePRINT: Exceeded max number \
01046 (%d) of arguments for PRINT section!\n",
01047                         NOSH_MAXPOP);
01048                 return 0;
01049             }
01050         } else {
01051             Vnm_print(2, "Nosh_parsePRINT: Syntax error in PRINT \
01052 section while reading %s!\n", tok);
01053             return 0;
01054         }
01055         /* Grab subtraction operation */
01056     } else if (Vstring_strcasecmp(tok, "-") == 0) {
01057         if (expect == 1) {
01058             thee->printop[idx][thee->printnarg[idx]] = 1;
01059             (thee->printnarg[idx])++;
01060             expect = 0;
01061             if (thee->printnarg[idx] >= NOSH_MAXPOP) {
01062                 Vnm_print(2, "Nosh_parseREAD: Exceeded max number \
01063 (%d) of arguments for PRINT section!\n",
01064                         NOSH_MAXPOP);
01065                 return 0;
01066             }
01067         } else {
01068             Vnm_print(2, "Nosh_parsePRINT: Syntax error in PRINT \
01069 section while reading %s!\n", tok);
01070             return 0;
01071         }
01072         /* Grab a calculation name from elec ID */
01073     } else if (sscanf(tok, "%s", name) == 1) {
01074         if (expect == 0) {
01075             for (ielec=0; ielec<thee->nelec; ielec++) {
01076                 if (Vstring_strcasecmp(thee->elecname[ielec], name) == 0)
01077                 {
01078                     thee->printcalc[idx][thee->printnarg[idx]] = ielec;
01079                     expect = 1;
01080                     break;
01081                 }
01082                 for (iapol=0; iapol<thee->napol; iapol++) {
01083                     if (Vstring_strcasecmp(thee->apolname[iapol], name) == 0)
01084                     {
01085                         thee->printcalc[idx][thee->printnarg[idx]] = iapol;
01086                         expect = 1;
01087                         break;
01088                     }
01089                     if (expect == 0) {
01090                         Vnm_print(2, "No ELEC or APOL statement has been named %s!\n",
01091                                 name);
01092                         return 0;
01093                     }
01094                 } else {
01095                     Vnm_print(2, "Nosh_parsePRINT: Syntax error in PRINT \
01096 section while reading %s!\n", tok);
01097                     return 0;
01098                 }
01099                 /* Got bad operation */
01100             } else {
01101                 Vnm_print(2, "Nosh_parsePRINT: Undefined keyword %s while \
01102 parsing PRINT section!\n", tok);
01103                 return 0;
01104             }
01105         } /* end parse token */
01106     } /* end while */
01107     } /* end while */
01108     VJMPERR1(0);
01109     /* We ran out of tokens! */
01110     VERROR1:
01111     Vnm_print(2, "Nosh_parsePRINT: Ran out of tokens while parsing PRINT \
01112 section!\n");

```

```

01115         return 0;
01116
01117     }
01118
01119     VPRIVATE int Nosh_parseELEC(NOsh *thee, Vio *sock) {
01120
01121         NOsh_calc *calc = VNULL;
01122
01123         char tok[VMAX_BUFSIZE];
01124
01125         if (thee == VNULL) {
01126             Vnm_print(2, "Nosh_parseELEC: Got NULL thee!\n");
01127             return 0;
01128         }
01129
01130         if (sock == VNULL) {
01131             Vnm_print(2, "Nosh_parseELEC: Got pointer to NULL socket!\n");
01132             return 0;
01133         }
01134
01135         if (thee->parsed) {
01136             Vnm_print(2, "Nosh_parseELEC: Already parsed an input file!\n");
01137             return 0;
01138         }
01139
01140         /* Get a pointer to the latest ELEC calc object and update the ELEC
01141            statement number */
01142         if (thee->nelec >= NOSH_MAXCALC) {
01143             Vnm_print(2, "Nosh: Too many electrostatics calculations in this \
01144 run!\n");
01145             Vnm_print(2, "Nosh: Current max is %d; ignoring this calculation\n",
01146                     NOSH_MAXCALC);
01147             return 1;
01148         }
01149
01150         /* The next token HAS to be the method OR "name" */
01151         if (Vio_scanf(sock, "%s", tok) == 1) {
01152             if (Vstring_strcasecmp(tok, "name") == 0) {
01153                 Vio_scanf(sock, "%s", tok);
01154                 strncpy(thee->elecname[thee->nelec], tok, VMAX_ARGLEN);
01155                 if (Vio_scanf(sock, "%s", tok) != 1) {
01156                     Vnm_print(2, "Nosh_parseELEC: Ran out of tokens while reading \
01157 ELEC section!\n");
01158                     return 0;
01159                 }
01160             }
01161             if (Vstring_strcasecmp(tok, "mg-manual") == 0) {
01162                 thee->elec[thee->nelec] = Nosh_calc_ctor(
01163                     NCT_MG);
01164                 calc = thee->elec[thee->nelec];
01165                 (thee->nelec)++;
01166                 calc->mgparm->type = MCT_MANUAL;
01167                 return Nosh_parseMG(thee, sock, calc);
01168             } else if (Vstring_strcasecmp(tok, "mg-auto") == 0) {
01169                 thee->elec[thee->nelec] = Nosh_calc_ctor(
01170                     NCT_MG);
01171                 calc = thee->elec[thee->nelec];
01172                 (thee->nelec)++;
01173                 calc->mgparm->type = MCT_AUTO;
01174                 return Nosh_parseMG(thee, sock, calc);
01175             } else if (Vstring_strcasecmp(tok, "mg-para") == 0) {
01176                 thee->elec[thee->nelec] = Nosh_calc_ctor(
01177                     NCT_MG);
01178                 calc = thee->elec[thee->nelec];
01179                 (thee->nelec)++;
01180                 calc->mgparm->type = MCT_PARALLEL;
01181                 return Nosh_parseMG(thee, sock, calc);
01182             } else if (Vstring_strcasecmp(tok, "mg-dummy") == 0) {
01183                 thee->elec[thee->nelec] = Nosh_calc_ctor(
01184                     NCT_MG);
01185                 calc = thee->elec[thee->nelec];
01186                 (thee->nelec)++;
01187                 calc->mgparm->type = MCT_DUMMY;
01188                 return Nosh_parseMG(thee, sock, calc);
01189             } else if (Vstring_strcasecmp(tok, "fe-manual") == 0) {
01190                 thee->elec[thee->nelec] = Nosh_calc_ctor(
01191                     NCT_FEM);
01192                 calc = thee->elec[thee->nelec];
01193                 (thee->nelec)++;
01194                 calc->femparm->type = FCT_MANUAL;
01195                 return Nosh_parseFEM(thee, sock, calc);
01196             }
01197         }
01198     }
01199 }

```

```

01191         } else if (Vstring_strcasecmp(tok, "bem-manual") == 0) {
01192             thee->elec[thee->nelec] = NOsh_calc_ctor(
NCT_BEM);
01193             calc = thee->elec[thee->nelec];
01194             (thee->nelec)++;
01195             calc->bemparm->type = BCT_MANUAL;
01196             return NOsh_parseBEM(thee, sock, calc);
01197         } else if (Vstring_strcasecmp(tok, "geoflow-manual") == 0) {
01198             thee->elec[thee->nelec] = NOsh_calc_ctor(
NCT_GEOFLOW);
01199             calc = thee->elec[thee->nelec];
01200             (thee->nelec)++;
01201             calc->geoflowparm->type = GFCT_MANUAL;
01202             return NOsh_parseGEOFLOW(thee, sock, calc);
01203         } else if (Vstring_strcasecmp(tok, "geoflow-auto") == 0) {
01204             thee->elec[thee->nelec] = NOsh_calc_ctor(
NCT_GEOFLOW);
01205             calc = thee->elec[thee->nelec];
01206             (thee->nelec)++;
01207             calc->geoflowparm->type = GFCT_AUTO;
01208             return NOsh_parseGEOFLOW(thee, sock, calc);
01209         } else {
01210             Vnm_print(2, "NOsh_parseELEC: The method (\\"mg\\",\\"fem\\", \\"bem\\", \\"geoflow\\") or \
01211 \\"name\\" must be the first keyword in the ELEC section\\n");
01212             return 0;
01213         }
01214     }
01215     Vnm_print(2, "NOsh_parseELEC: Ran out of tokens while reading ELEC section!\\n");
01216     return 0;
01217 }
01218 }
01219 }
01220
01221 VPRIVATE int NOsh_parseAPOLAR(NOsh *thee, Vio *sock) {
01222     NOsh_calc *calc = VNULL;
01223     char tok[VMAX_BUFSIZE];
01224     if (thee == VNULL) {
01225         Vnm_print(2, "NOsh_parseAPOLAR: Got NULL thee!\\n");
01226         return 0;
01227     }
01228     if (sock == VNULL) {
01229         Vnm_print(2, "NOsh_parseAPOLAR: Got pointer to NULL socket!\\n");
01230         return 0;
01231     }
01232     if (thee->parsed) {
01233         Vnm_print(2, "NOsh_parseAPOLAR: Already parsed an input file!\\n");
01234         return 0;
01235     }
01236     /* Get a pointer to the latest ELEC calc object and update the ELEC
01237     statement number */
01238     if (thee->napol >= NOSH_MAXCALC) {
01239         Vnm_print(2, "NOsh: Too many non-polar calculations in this \
01240 run!\\n");
01241         Vnm_print(2, "NOsh: Current max is %d; ignoring this calculation\\n",
01242             NOSH_MAXCALC);
01243         return 1;
01244     }
01245     /* The next token HAS to be the method OR "name" */
01246     if (Vio_scanf(sock, "%s", tok) == 1) {
01247         if (Vstring_strcasecmp(tok, "name") == 0) {
01248             Vio_scanf(sock, "%s", tok);
01249             strncpy(thee->apolname[thee->napol], tok, VMAX_ARGLEN);
01250             /* Parse the non-polar parameters */
01251             thee->apol[thee->napol] = NOsh_calc_ctor(
NCT_APOL);
01252             calc = thee->apol[thee->napol];
01253             (thee->napol)++;
01254             return NOsh_parseAPOL(thee, sock, calc);
01255         } else if (Vstring_strcasecmp(tok, "geoflow-manual") == 0) {
01256             thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_GEOFLOW);
01257             calc = thee->elec[thee->nelec];
01258             (thee->nelec)++;
01259             calc->geoflowparm->type = GFCT_MANUAL;

```

```

01268 //         return Nosh_parseGEOFLOW(thee, sock, calc);
01269 //     } else if (Vstring_strcasecmp(tok, "geoflow-auto") == 0) {
01270 //         thee->elec[thee->nelec] = Nosh_calc_ctor(NCT_GEOFLOW);
01271 //         calc = thee->elec[thee->nelec];
01272 //         (thee->nelec)++;
01273 //         calc->geoflowparm->type = GFCT_AUTO;
01274 //         return Nosh_parseGEOFLOW(thee, sock, calc);
01275     }
01276 }
01277
01278 return 1;
01279
01280 }
01281
01282 VPUBLIC int Nosh_setupElecCalc(
01283     Nosh *thee,
01284     Valist *alist[NOSH_MAXMOL]
01285 ) {
01286     int ielec, imol, i;
01287     Nosh_calc *elec = VNULL;
01288     MGparm *mgparm = VNULL;
01289     Valist *mymol = VNULL;
01290
01291     VASSERT(thee != VNULL);
01292     for (imol=0; imol<thee->nmol; imol++) {
01293         thee->alist[imol] = alist[imol];
01294     }
01295
01296     for (ielec=0; ielec<(thee->nelec); ielec++) {
01297         /* Unload the calculation object containing the ELEC information */
01298         elec = thee->elec[ielec];
01299
01300         if (((thee->ndiel != 0) || (thee->nkappa != 0) ||
01301             (thee->ncharge != 0) || (thee->npot != 0)) &&
01302             (elec->pbeparm->calcforce != PCF_NO)) {
01303             Vnm_print(2, "Nosh_setupElecCalc: Calculation of forces disabled because surface \
01304 map is used!\n");
01305             elec->pbeparm->calcforce = PCF_NO;
01306         }
01307     }
01308
01309     /* Setup the calculation */
01310     switch (elec->calctype) {
01311     case NCT_MG:
01312         /* Center on the molecules, if requested */
01313         mgparm = elec->mgparm;
01314         VASSERT(mgparm != VNULL);
01315         if (elec->mgparm->cmeth == MCM_MOLECULE) {
01316             VASSERT(mgparm->centmol >= 0);
01317             VASSERT(mgparm->centmol < thee->nmol);
01318             mymol = thee->alist[mgparm->centmol];
01319             VASSERT(mymol != VNULL);
01320             for (i=0; i<3; i++) {
01321                 mgparm->center[i] = mymol->center[i];
01322             }
01323         }
01324         if (elec->mgparm->fcmeth == MCM_MOLECULE) {
01325             VASSERT(mgparm->fcentmol >= 0);
01326             VASSERT(mgparm->fcentmol < thee->nmol);
01327             mymol = thee->alist[mgparm->fcentmol];
01328             VASSERT(mymol != VNULL);
01329             for (i=0; i<3; i++) {
01330                 mgparm->fcenter[i] = mymol->center[i];
01331             }
01332         }
01333         if (elec->mgparm->ccmeth == MCM_MOLECULE) {
01334             VASSERT(mgparm->ccentmol >= 0);
01335             VASSERT(mgparm->ccentmol < thee->nmol);
01336             mymol = thee->alist[mgparm->ccentmol];
01337             VASSERT(mymol != VNULL);
01338             for (i=0; i<3; i++) {
01339                 mgparm->ccenter[i] = mymol->center[i];
01340             }
01341         }
01342         Nosh_setupCalcMG(thee, elec);
01343         break;
01344     case NCT_FEM:
01345         Nosh_setupCalcFEM(thee, elec);
01346         break;
01347     case NCT_BEM:
01348         Nosh_setupCalcBEM(thee, elec);

```

```

01349         break;
01350     case NCT_GEOFLOW:
01351         Nosh_setupCalcGEOFLOW(thee, elec);
01352         break;
01353     default:
01354         Vnm_print(2, "Nosh_setupCalc: Invalid calculation type (%d)!\n",
01355             elec->calctype);
01356         return 0;
01357 }
01358
01359 /* At this point, the most recently-created Nosh_calc object should be the
01360    one we use for results for this ELEC statement. Assign it. */
01361 /* Associate ELEC statement with the calculation */
01362 thee->elec2calc[ielec] = thee->ncalc-1;
01363 Vnm_print(0, "Nosh_setupCalc: Mapping ELEC statement %d (%d) to \
01364 calculation %d (%d)\n", ielec, ielec+1, thee->elec2calc[ielec],
01365     thee->elec2calc[ielec]+1);
01366 }
01367 return 1;
01368 }
01369 }
01370
01371 VPUBLIC int Nosh_setupApolCalc(
01372     Nosh *thee,
01373     Valist *alist[NOSH_MAXMOL]
01374 ) {
01375     int iapol, imol;
01376     int doCalc = ACD_NO;
01377     Nosh_calc *calc = VNULL;
01378
01379     VASSERT(thee != VNULL);
01380     for (imol=0; imol<thee->nmol; imol++) {
01381         thee->alist[imol] = alist[imol];
01382     }
01383
01384     for (iapol=0; iapol<(thee->napol); iapol++) {
01385         /* Unload the calculation object containing the APOL information */
01386         calc = thee->apol[iapol];
01387
01388         /* Setup the calculation */
01389         switch (calc->calctype) {
01390             case NCT_APOL:
01391                 Nosh_setupCalcAPOL(thee, calc);
01392                 doCalc = ACD_YES;
01393                 break;
01394             default:
01395                 Vnm_print(2, "Nosh_setupCalc: Invalid calculation type (%d)!\n", calc->
01396                     calctype);
01397                 return ACD_ERROR;
01398         }
01399         /* At this point, the most recently-created Nosh_calc object should be the
01400            one we use for results for this APOL statement. Assign it. */
01401         /* Associate APOL statement with the calculation */
01402         thee->apol2calc[iapol] = thee->ncalc-1;
01403         Vnm_print(0, "Nosh_setupCalc: Mapping APOL statement %d (%d) to calculation %d (%d)\n", iapol,
01404             iapol+1, thee->apol2calc[iapol], thee->apol2calc[iapol]+1);
01405     }
01406
01407     if (doCalc == ACD_YES) {
01408         return ACD_YES;
01409     } else {
01410         return ACD_NO;
01411     }
01412 }
01413
01414 VPUBLIC int Nosh_parseMG(
01415     Nosh *thee,
01416     Vio *sock,
01417     Nosh_calc *elec
01418 ) {
01419     char tok[VMAX_BUFSIZE];
01420     MGparm *mgparm = VNULL;
01421     PBeparm *pbeparm = VNULL;
01422     int rc;
01423
01424     /* Check the arguments */
01425     if (thee == VNULL) {
01426         Vnm_print(2, "Nosh: Got NULL thee!\n");
01427         return 0;
01428     }

```



```

01428     if (sock == VNULL) {
01429         Vnm_print(2, "Nosh: Got pointer to NULL socket!\n");
01430         return 0;
01431     }
01432     if (elec == VNULL) {
01433         Vnm_print(2, "Nosh: Got pointer to NULL elec object!\n");
01434         return 0;
01435     }
01436     mgparm = elec->mgparm;
01437     if (mgparm == VNULL) {
01438         Vnm_print(2, "Nosh: Got pointer to NULL mgparm object!\n");
01439         return 0;
01440     }
01441     pbeparm = elec->pbeparm;
01442     if (pbeparm == VNULL) {
01443         Vnm_print(2, "Nosh: Got pointer to NULL pbeparm object!\n");
01444         return 0;
01445     }
01446
01447     Vnm_print(0, "Nosh_parseMG: Parsing parameters for MG calculation\n");
01448
01449     /* Parallel stuff */
01450     if (mgparm->type == MCT_PARALLEL) {
01451         mgparm->proc_rank = thee->proc_rank;
01452         mgparm->proc_size = thee->proc_size;
01453         mgparm->setrank = 1;
01454         mgparm->setsize = 1;
01455     }
01456
01457     /* Start snarfing tokens from the input stream */
01458     rc = 1;
01459     while (Vio_scanf(sock, "%s", tok) == 1) {
01460
01461         Vnm_print(0, "Nosh_parseMG: Parsing %s...\n", tok);
01462
01463         /* See if it's an END token */
01464         if (Vstring_strcasecmp(tok, "end") == 0) {
01465             mgparm->parsed = 1;
01466             pbeparm->parsed = 1;
01467             rc = 1;
01468             break;
01469         }
01470     }
01471
01472     /* Pass the token through a series of parsers */
01473     rc = PBEParm_parseToken(pbeparm, tok, sock);
01474     if (rc == -1) {
01475         Vnm_print(0, "Nosh_parseMG: parsePBE error!\n");
01476         break;
01477     } else if (rc == 0) {
01478         /* Pass the token to the generic MG parser */
01479         rc = MGparm_parseToken(mgparm, tok, sock);
01480         if (rc == -1) {
01481             Vnm_print(0, "Nosh_parseMG: parseMG error!\n");
01482             break;
01483         } else if (rc == 0) {
01484             /* We ran out of parsers! */
01485             Vnm_print(2, "Nosh: Unrecognized keyword: %s\n", tok);
01486             break;
01487         }
01488     }
01489 }
01490
01491 /* Handle various errors arising in the token-snarfing loop -- these all
01492 just result in simple returns right now */
01493 if (rc == -1) return 0;
01494 if (rc == 0) return 0;
01495
01496 /* Check the status of the parameter objects */
01497 if ((MGparm_check(mgparm) == VRC_FAILURE) || (!
PBEParm_check(pbeparm))) {
01498     Vnm_print(2, "Nosh: MG parameters not set correctly!\n");
01499     return 0;
01500 }
01501
01502 return 1;
01503 }
01504
01505 VPRIVATE int Nosh_setupCalcMG(
01506     Nosh *thee,
01507     Nosh_calc *calc

```

```

01508         ) {
01509
01510         MGparm *mgparm = VNULL;
01511
01512         VASSERT(thee != VNULL);
01513         VASSERT(calc != VNULL);
01514         mgparm = calc->mgparm;
01515         VASSERT(mgparm != VNULL);
01516
01517
01518         /* Now we're ready to whatever sorts of post-processing operations that are
01519            necessary for the various types of calculations */
01520         switch (mgparm->type) {
01521             case MCT_MANUAL:
01522                 return NOsh_setupCalcMGMANUAL(thee, calc);
01523             case MCT_DUMMY:
01524                 return NOsh_setupCalcMGMANUAL(thee, calc);
01525             case MCT_AUTO:
01526                 return NOsh_setupCalcMGAUTO(thee, calc);
01527             case MCT_PARALLEL:
01528                 return NOsh_setupCalcMGPARA(thee, calc);
01529             default:
01530                 Vnm_print(2, "NOsh_setupCalcMG: undefined MG calculation type (%d)!\n",
01531                     mgparm->type);
01532                 return 0;
01533         }
01534
01535         /* Shouldn't get here */
01536         return 0;
01537     }
01538
01539
01540
01541     VPRIVATE int NOsh_setupCalcBEM(
01542         NOsh *thee,
01543         NOsh_calc *calc
01544     ) {
01545
01546         BEMparm *bemparm = VNULL;
01547
01548         VASSERT(thee != VNULL);
01549         VASSERT(calc != VNULL);
01550         bemparm = calc->bemparm;
01551         VASSERT(bemparm != VNULL);
01552
01553
01554         /* Now we're ready to whatever sorts of post-processing operations that are
01555            necessary for the various types of calculations */
01556         switch (bemparm->type) {
01557             case BCT_MANUAL:
01558                 return NOsh_setupCalcBEMMANUAL(thee, calc);
01559             default:
01560                 Vnm_print(2, "NOsh_setupCalcBEM: undefined BEM calculation type (%d)!\n",
01561                     bemparm->type);
01562                 return 0;
01563         }
01564
01565         /* Shouldn't get here */
01566         return 0;
01567     }
01568
01569     VPRIVATE int NOsh_setupCalcGEOFLOW(NOsh *thee, NOsh_calc *calc) {
01570
01571         GEOFLOWparm *parm = VNULL;
01572
01573         VASSERT(thee != VNULL);
01574         VASSERT(calc != VNULL);
01575         parm = calc->geoflowparm;
01576         VASSERT(parm != VNULL);
01577
01578
01579         /* Now we're ready to whatever sorts of post-processing operations that are
01580            necessary for the various types of calculations */
01581         if (parm->type == GFCT_MANUAL || parm->type == GFCT_AUTO) {
01582             return NOsh_setupCalcGEOFLOWMANUAL(thee, calc);
01583         } else {
01584             Vnm_print(2, "NOsh_setupCalcGEOFLOW: undefined GEOFLOW calculation type (%d)!\n", parm->
01585                 type);
01586             return 0;
01587         }
01588     }

```

```

01588
01589
01590 VPRIVATE int NOsh_setupCalcFEM(
01591     Nosh *thee,
01592     Nosh_calc *calc
01593 ) {
01594
01595     VASSERT(thee != VNULL);
01596     VASSERT(calc != VNULL);
01597     VASSERT(calc->femparm != VNULL);
01598
01599     /* Now we're ready to whatever sorts of post-processing operations that are
01600      * necessary for the various types of calculations */
01601     switch (calc->femparm->type) {
01602     case FCT_MANUAL:
01603         return NOsh_setupCalcFEMANUAL(thee, calc);
01604     default:
01605         Vnm_print(2, "NOsh_parseFEM: unknown calculation type (%d)!\n",
01606             calc->femparm->type);
01607         return 0;
01608     }
01609
01610     /* Shouldn't get here */
01611     return 0;
01612 }
01613
01614
01615 VPRIVATE int NOsh_setupCalcMGMANUAL(
01616     Nosh *thee,
01617     Nosh_calc *elec
01618 ) {
01619
01620     MGparm *mgparm = VNULL;
01621     PBEParm *pbeparm = VNULL;
01622     Nosh_calc *calc = VNULL;
01623
01624     if (thee == VNULL) {
01625         Vnm_print(2, "NOsh_setupCalcMGMANUAL: Got NULL thee!\n");
01626         return 0;
01627     }
01628     if (elec == VNULL) {
01629         Vnm_print(2, "NOsh_setupCalcMGMANUAL: Got NULL calc!\n");
01630         return 0;
01631     }
01632     mgparm = elec->mgparm;
01633     if (mgparm == VNULL) {
01634         Vnm_print(2, "NOsh_setupCalcMGMANUAL: Got NULL mgparm -- was this calculation \
01635 set up?\n");
01636         return 0;
01637     }
01638     pbeparm = elec->pbeparm;
01639     if (pbeparm == VNULL) {
01640         Vnm_print(2, "NOsh_setupCalcMGMANUAL: Got NULL pbeparm -- was this calculation \
01641 set up?\n");
01642         return 0;
01643     }
01644
01645     /* Set up missing MG parameters */
01646     if (mgparm->setgrid == 0) {
01647         VASSERT(mgparm->setglen);
01648         mgparm->grid[0] = mgparm->glen[0]/((double) (mgparm->dime[0]-1));
01649         mgparm->grid[1] = mgparm->glen[1]/((double) (mgparm->dime[1]-1));
01650         mgparm->grid[2] = mgparm->glen[2]/((double) (mgparm->dime[2]-1));
01651     }
01652     if (mgparm->setglen == 0) {
01653         VASSERT(mgparm->setgrid);
01654         mgparm->glen[0] = mgparm->grid[0]*((double) (mgparm->dime[0]-1));
01655         mgparm->glen[1] = mgparm->grid[1]*((double) (mgparm->dime[1]-1));
01656         mgparm->glen[2] = mgparm->grid[2]*((double) (mgparm->dime[2]-1));
01657     }
01658
01659     /* Check to see if he have any room left for this type of calculation, if
01660      so: set the calculation type, update the number of calculations of this type,
01661      and parse the rest of the section */
01662     if (thee->ncalc >= NOSH_MAXCALC) {
01663         Vnm_print(2, "NOsh: Too many calculations in this run!\n");
01664         Vnm_print(2, "NOsh: Current max is %d; ignoring this calculation\n",
01665             NOSH_MAXCALC);
01666         return 0;
01667     }
01668

```

```

01669      /* Get the next calculation object and increment the number of calculations */
01670      thee->calc[thee->ncalc] = NOsh_calc_ctor(NCT_MG);
01671      calc = thee->calc[thee->ncalc];
01672      (thee->ncalc)++;
01673
01674
01675
01676      /* Copy over contents of ELEC */
01677      NOsh_calc_copy(calc, elec);
01678
01679
01680      return 1;
01681 }
01682
01683 VPUBLIC int NOsh_setupCalcMGAUTO(
01684     NOsh *thee,
01685     NOsh_calc *elec
01686 ) {
01687
01688     NOsh_calc *calcf = VNULL;
01689     NOsh_calc *calcc = VNULL;
01690     double fgrid[3], cgrid[3];
01691     double d[3], minf[3], maxf[3], minc[3], maxc[3];
01692     double redfrac, redrat[3], td;
01693     int ifocus, nfocus, tnfocus[3];
01694     int j;
01695     int icalc;
01696     int dofix;
01697
01698     /* A comment about the coding style in this function. I use lots and lots
01699        and lots of pointer deferencing. I could (and probably should) save
01700        these in temporary variables. However, since there are so many MGparm,
01701        etc. and NOsh_calc, etc. objects running around in this function, the
01702        current scheme is easiest to debug. */
01703
01704
01705     if (thee == VNULL) {
01706         Vnm_print(2, "NOsh_setupCalcMGAUTO: Got NULL thee!\n");
01707         return 0;
01708     }
01709     if (elec == VNULL) {
01710         Vnm_print(2, "NOsh_setupCalcMGAUTO: Got NULL elec!\n");
01711         return 0;
01712     }
01713     if (elec->mgparm == VNULL) {
01714         Vnm_print(2, "NOsh_setupCalcMGAUTO: Got NULL mgparm!\n");
01715         return 0;
01716     }
01717     if (elec->pbeparm == VNULL) {
01718         Vnm_print(2, "NOsh_setupCalcMGAUTO: Got NULL pbeparm!\n");
01719         return 0;
01720     }
01721
01722     Vnm_print(0, "NOsh_setupCalcMGAUTO(%s, %d): coarse grid center = %g %g %g\n",
01723         __FILE__, __LINE__,
01724         elec->mgparm->ccenter[0],
01725         elec->mgparm->ccenter[1],
01726         elec->mgparm->ccenter[2]);
01727     Vnm_print(0, "NOsh_setupCalcMGAUTO(%s, %d): fine grid center = %g %g %g\n",
01728         __FILE__, __LINE__,
01729         elec->mgparm->fcenter[0],
01730         elec->mgparm->fcenter[1],
01731         elec->mgparm->fcenter[2]);
01732
01733     /* Calculate the grid spacing on the coarse and fine levels */
01734     for (j=0; j<3; j++) {
01735         cgrid[j] = (elec->mgparm->cglen[j])/((double)(elec->mgparm->
01736             dime[j]-1));
01737         fgrid[j] = (elec->mgparm->fglen[j])/((double)(elec->mgparm->
01738             dime[j]-1));
01739         d[j] = elec->mgparm->fcenter[j] - elec->mgparm->
01740             ccenter[j];
01741     }
01742     Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): Coarse grid spacing = %g, %g, %g\n",
01743         __FILE__, __LINE__, cgrid[0], cgrid[1], cgrid[2]);
01744     Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): Fine grid spacing = %g, %g, %g\n",
01745         __FILE__, __LINE__, fgrid[0], fgrid[1], fgrid[2]);
01746     Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): Displacement between fine \
01747 coarse grids = %g, %g, %g\n", __FILE__, __LINE__, d[0], d[1], d[2]);
01748
01749     /* Now calculate the number of focusing levels, never reducing the grid

```

```

01747     spacing by more than redfrac at each level */
01748     for (j=0; j<3; j++) {
01749         if (fgrid[j]/cgrid[j] < VREDFRAC) {
01750             redfrac = fgrid[j]/cgrid[j];
01751             td = log(redfrac)/log(VREDFRAC);
01752             tnfocus[j] = (int)ceil(td) + 1;
01753         } else tnfocus[j] = 2;
01754     }
01755     nfocus = VMAX2(VMAX2(tnfocus[0], tnfocus[1]), tnfocus[2]);
01756
01757     /* Now set redrat to the actual value by which the grid spacing is reduced
01758        at each level of focusing */
01759     for (j=0; j<3; j++) {
01760         redrat[j] = VPOW((fgrid[j]/cgrid[j]), 1.0/((double)nfocus-1.0));
01761     }
01762     Vnm_print(0, "Nosh:  %d levels of focusing with %g, %g, %g reductions\n",
01763         nfocus, redrat[0], redrat[1], redrat[2]);
01764
01765     /* Now that we know how many focusing levels to use, we're ready to set up
01766        the parameter objects */
01767     if (nfocus > (NOSH_MAXCALC-(thee->ncalc))) {
01768         Vnm_print(2, "Nosh:  Require more calculations than max (%d)!\n",
01769             NOSH_MAXCALC);
01770         return 0;
01771     }
01772
01773     for (ifocus=0; ifocus<nfocus; ifocus++) {
01774
01775         /* Generate the new calc object */
01776         icalc = thee->ncalc;
01777         thee->calc[icalc] = Nosh_calc_ctor(NCT_MG);
01778         (thee->ncalc)++;
01779
01780         /* This is the _current_ Nosh_calc object */
01781         calcf = thee->calc[icalc];
01782         /* This is the _previous_ Nosh_calc object */
01783         if (ifocus != 0) {
01784             calcc = thee->calc[icalc-1];
01785         } else {
01786             calcc = VNULL;
01787         }
01788
01789         /* Copy over most of the parameters from the ELEC object */
01790         Nosh_calc_copy(calcf, elec);
01791
01792         /* Set up the grid lengths and spacings */
01793         if (ifocus == 0) {
01794             for (j=0; j<3; j++) {
01795                 calcf->mgparm->grid[j] = cgrid[j];
01796                 calcf->mgparm->glen[j] = elec->mgparm->cglen[j];
01797             }
01798         } else {
01799             for (j=0; j<3; j++) {
01800                 calcf->mgparm->grid[j] = redrat[j]*(calcc->mgparm->
01801 grid[j]);
01802                 calcf->mgparm->glen[j] = redrat[j]*(calcc->mgparm->
01803 glen[j]);
01804             }
01805             calcf->mgparm->setgrid = 1;
01806             calcf->mgparm->setglen = 1;
01807
01808             /* Get centers and centering method from coarse and fine meshes */
01809             if (ifocus == 0) {
01810                 calcf->mgparm->cmeth = elec->mgparm->ccmeth;
01811                 calcf->mgparm->centmol = elec->mgparm->ccentmol;
01812                 for (j=0; j<3; j++) {
01813                     calcf->mgparm->center[j] = elec->mgparm->
01814 ccenter[j];
01815                 }
01816             } else if (ifocus == (nfocus-1)) {
01817                 calcf->mgparm->cmeth = elec->mgparm->fcmeth;
01818                 calcf->mgparm->centmol = elec->mgparm->fcentmol;
01819                 for (j=0; j<3; j++) {
01820                     calcf->mgparm->center[j] = elec->mgparm->
01821 fcenter[j];
01822                 }
01823             } else {
01824                 calcf->mgparm->cmeth = MCM_FOCUS;
01825                 /* TEMPORARILY move the current grid center
01826                    to the fine grid center.  In general, this will move portions of

```

```

01824         the current mesh off the immediately-coarser mesh. We'll fix that
01825         in the next step. */
01826         for (j=0; j<3; j++) {
01827             calcf->mgparm->center[j] = elec->mgparm->
fcenter[j];
01828         }
01829     }
01830
01831
01832     /* As mentioned above, it is highly likely that the previous "jump"
01833        to the fine grid center put portions of the current mesh off the
01834        previous (coarser) mesh. Fix this by displacing the current mesh
01835        back onto the previous coarser mesh. */
01836     if (ifocus != 0) {
01837         Vnm_print(0, "Nosh_setupCalcMGAUTO (%s, %d): starting mesh \
01838 repositioning.\n", __FILE__, __LINE__);
01839         Vnm_print(0, "Nosh_setupCalcMGAUTO (%s, %d): coarse mesh center = \
01840 %g %g %g\n", __FILE__, __LINE__,
01841             calcc->mgparm->center[0],
01842             calcc->mgparm->center[1],
01843             calcc->mgparm->center[2]);
01844         Vnm_print(0, "Nosh_setupCalcMGAUTO (%s, %d): coarse mesh upper corner = \
01845 %g %g %g\n", __FILE__, __LINE__,
01846             calcc->mgparm->center[0]+0.5*(calcc->mgparm->
glen[0]),
01847             calcc->mgparm->center[1]+0.5*(calcc->mgparm->
glen[1]),
01848             calcc->mgparm->center[2]+0.5*(calcc->mgparm->
glen[2]));
01849         Vnm_print(0, "Nosh_setupCalcMGAUTO (%s, %d): coarse mesh lower corner = \
01850 %g %g %g\n", __FILE__, __LINE__,
01851             calcc->mgparm->center[0]-0.5*(calcc->mgparm->
glen[0]),
01852             calcc->mgparm->center[1]-0.5*(calcc->mgparm->
glen[1]),
01853             calcc->mgparm->center[2]-0.5*(calcc->mgparm->
glen[2]));
01854         Vnm_print(0, "Nosh_setupCalcMGAUTO (%s, %d): initial fine mesh upper corner = \
01855 %g %g %g\n", __FILE__, __LINE__,
01856             calcf->mgparm->center[0]+0.5*(calcf->mgparm->
glen[0]),
01857             calcf->mgparm->center[1]+0.5*(calcf->mgparm->
glen[1]),
01858             calcf->mgparm->center[2]+0.5*(calcf->mgparm->
glen[2]));
01859         Vnm_print(0, "Nosh_setupCalcMGAUTO (%s, %d): initial fine mesh lower corner = \
01860 %g %g %g\n", __FILE__, __LINE__,
01861             calcf->mgparm->center[0]-0.5*(calcf->mgparm->
glen[0]),
01862             calcf->mgparm->center[1]-0.5*(calcf->mgparm->
glen[1]),
01863             calcf->mgparm->center[2]-0.5*(calcf->mgparm->
glen[2]));
01864         for (j=0; j<3; j++) {
01865             /* Check if we've fallen off of the lower end of the mesh */
01866             dofix = 0;
01867             minf[j] = calcf->mgparm->center[j]
- 0.5*(calcf->mgparm->glen[j]);
01868             minc[j] = calcc->mgparm->center[j]
- 0.5*(calcc->mgparm->glen[j]);
01869             d[j] = minc[j] - minf[j];
01870             if (d[j] >= VSMALL) {
01871                 if (ifocus == (nfocus-1)) {
01872                     Vnm_print(2, "Nosh_setupCalcMGAUTO: Error! Finest \
01873 mesh has fallen off the coarser meshes!\n");
01874                     Vnm_print(2, "Nosh_setupCalcMGAUTO: difference in min %d-\
01875 direction = %g\n", j, d[j]);
01876                     Vnm_print(2, "Nosh_setupCalcMGAUTO: min fine = %g %g %g\n",
minf[0], minf[1], minf[2]);
01877                     Vnm_print(2, "Nosh_setupCalcMGAUTO: min coarse = %g %g %g\n",
minc[0], minc[1], minc[2]);
01878                     VASSERT(0);
01879                 } else {
01880                     Vnm_print(0, "Nosh_setupCalcMGAUTO (%s, %d): ifocus = %d, \
01881 fixing mesh min violation (%g in %d-direction).\n", __FILE__, __LINE__, ifocus,
d[j], j);
01882                     calcf->mgparm->center[j] += d[j];
01883                     dofix = 1;
01884                 }
01885             }
01886         }
01887     }
01888     /* Check if we've fallen off of the upper end of the mesh */

```

```

01892         maxf[j] = calcf->mgparm->center[j] \
01893             + 0.5*(calcf->mgparm->glen[j]);
01894         maxc[j] = calcc->mgparm->center[j] \
01895             + 0.5*(calcc->mgparm->glen[j]);
01896         d[j] = maxf[j] - maxc[j];
01897         if (d[j] >= VSMALL) {
01898             if (ifocus == (nfocus-1)) {
01899                 Vnm_print(2, "Nosh_setupCalcMGAUTO: Error! Finest \
01900 mesh has fallen off the coarser meshes!\n");
01901                 Vnm_print(2, "Nosh_setupCalcMGAUTO: difference in %d-\
01902 direction = %g\n", j, d[j]);
01903                 VASSERT(0);
01904             } else {
01905                 /* If we already fixed the lower boundary and we now need
01906                  to fix the upper boundary, we have a serious problem. */
01907                 if (dofix) {
01908                     Vnm_print(2, "Nosh_setupCalcMGAUTO: Error! Both \
01909 ends of the finer mesh do not fit in the bigger mesh!\n");
01910                     VASSERT(0);
01911                 }
01912                 Vnm_print(0, "Nosh_setupCalcMGAUTO(%s, %d): ifocus = %d, \
01913 fixing mesh max violation (%g in %d-direction).\n", __FILE__, __LINE__, ifocus,
01914                     d[j], j);
01915                 calcf->mgparm->center[j] -= d[j];
01916                 dofix = 1;
01917             }
01918         }
01919     }
01920     Vnm_print(0, "Nosh_setupCalcMGAUTO (%s, %d): final fine mesh upper corner = \
01921 %g %g %g\n", __FILE__, __LINE__,
01922         calcf->mgparm->center[0]+0.5*(calcf->mgparm->
01923             glen[0]),
01924         calcf->mgparm->center[1]+0.5*(calcf->mgparm->
01925             glen[1]),
01926         calcf->mgparm->center[2]+0.5*(calcf->mgparm->
01927             glen[2]));
01928     Vnm_print(0, "Nosh_setupCalcMGAUTO (%s, %d): final fine mesh lower corner = \
01929 %g %g %g\n", __FILE__, __LINE__,
01930         calcf->mgparm->center[0]-0.5*(calcf->mgparm->
01931             glen[0]),
01932         calcf->mgparm->center[1]-0.5*(calcf->mgparm->
01933             glen[1]),
01934         calcf->mgparm->center[2]-0.5*(calcf->mgparm->
01935             glen[2]));
01936     }
01937     /* finer levels have focusing boundary conditions */
01938     if (ifocus != 0) calcf->pbeparm->bcfl = BCFL_FOCUS;
01939     /* Only the finest level handles I/O and needs to worry about disjoint
01940     partitioning */
01941     if (ifocus != (nfocus-1)) calcf->pbeparm->numwrite = 0;
01942     /* Reset boundary flags for everything except parallel focusing */
01943     if (calcf->mgparm->type != MCT_PARALLEL) {
01944         Vnm_print(0, "Nosh_setupMGAUTO: Resetting boundary flags\n");
01945         for (j=0; j<6; j++) calcf->mgparm->partDisjOwnSide[j] = 0;
01946         for (j=0; j<3; j++) {
01947             calcf->mgparm->partDisjCenter[j] = 0;
01948             calcf->mgparm->partDisjLength[j] = calcf->
01949                 mgparm->glen[j];
01950         }
01951     }
01952     calcf->mgparm->parsed = 1;
01953     }
01954     return 1;
01955 }
01956
01957 /* Author: Nathan Baker and Todd Dolinsky */
01958 VPUBLIC int Nosh_setupCalcMGAPARA(
01959     NOsh *thee,
01960     NOsh_calc *elec
01961 ) {
01962     /* NEW (25-Jul-2006): This code should produce modify the ELEC statement
01963     and pass it on to MGAUTO for further processing. */
01964 }
01965

```

```

01966     MGparm *mgparm = VNULL;
01967     double ofrac;
01968     double hx, hy, hzed;
01969     double xofrac, yofrac, zofrac;
01970     int rank, size, npx, npy, npz, nproc, ip, jp, kp;
01971     int xeffGlob, yeffGlob, zeffGlob, xDisj, yDisj, zDisj;
01972     int xigminDisj, xigmaxDisj, yigminDisj, yigmaxDisj, zigminDisj, zigmaxDisj;
01973     int xigminOlap, xigmaxOlap, yigminOlap, yigmaxOlap, zigminOlap, zigmaxOlap;
01974     int xOlapReg, yOlapReg, zOlapReg;
01975     double xlenDisj, ylenDisj, zlenDisj;
01976     double xcentDisj, ycentDisj, zcentDisj;
01977     double xcentOlap, ycentOlap, zcentOlap;
01978     double xlenOlap, ylenOlap, zlenOlap;
01979     double xminOlap, xmaxOlap, yminOlap, ymaxOlap, zminOlap, zmaxOlap;
01980     double xminDisj, xmaxDisj, yminDisj, ymaxDisj, zminDisj, zmaxDisj;
01981     double xcent, ycent, zcent;
01982
01983     /* Grab some useful variables */
01984     VASSERT(thee != VNULL);
01985     VASSERT(elec != VNULL);
01986     mgparm = elec->mgparm;
01987     VASSERT(mgparm != VNULL);
01988
01989     /* Grab some useful variables */
01990     ofrac = mgparm->ofrac;
01991     npx = mgparm->pdime[0];
01992     npy = mgparm->pdime[1];
01993     npz = mgparm->pdime[2];
01994     nproc = npx*npy*npz;
01995
01996     /* If this is not an asynchronous calculation, then we need to make sure we
01997        have all the necessary MPI information */
01998     if (mgparm->setasync == 0) {
01999
02000 #ifndef HAVE_MPI_H
02001
02002         Vnm_tprint(2, "NOsh_setupCalcMGPARA:  Oops!  You're trying to perform \
02003 an 'mg-para' (parallel) calculation\n");
02004         Vnm_tprint(2, "NOsh_setupCalcMGPARA:  with a version of APBS that wasn't \
02005 compiled with MPI!\n");
02006         Vnm_tprint(2, "NOsh_setupCalcMGPARA:  Perhaps you meant to use the \
02007 'async' flag?\n");
02008         Vnm_tprint(2, "NOsh_setupCalcMGPARA:  Bailing out!\n");
02009
02010         return 0;
02011
02012 #endif
02013
02014         rank = thee->proc_rank;
02015         size = thee->proc_size;
02016         Vnm_print(0, "NOsh_setupCalcMGPARA:  Hello from processor %d of %d\n", rank,
02017                 size);
02018
02019         /* Check to see if we have too many processors.  If so, then simply set
02020            this processor to duplicating the work of processor 0. */
02021         if (rank > (nproc-1)) {
02022             Vnm_print(2, "NOsh_setupMGPARA:  There are more processors available than\
02023 the %d you requested.\n", nproc);
02024             Vnm_print(2, "NOsh_setupMGPARA:  Eliminating processor %d\n", rank);
02025             thee->bogus = 1;
02026             rank = 0;
02027         }
02028
02029         /* Check to see if we have too few processors.  If so, this is a fatal
02030            error. */
02031         if (size < nproc) {
02032             Vnm_print(2, "NOsh_setupMGPARA:  There are too few processors (%d) to \
02033 satisfy requirements (%d)\n", size, nproc);
02034             return 0;
02035         }
02036
02037         Vnm_print(0, "NOsh_setupMGPARA:  Hello (again) from processor %d of %d\n",
02038                 rank, size);
02039
02040     } else { /* Setting up for an asynchronous calculation. */
02041
02042         rank = mgparm->async;
02043
02044         thee->ispara = 1;
02045         thee->proc_rank = rank;
02046

```



```

02047      /* Check to see if the async id is greater than the number of
02048      * processors. If so, this is a fatal error. */
02049      if (rank > (nproc-1)) {
02050          Vnm_print(2, "Nosh_setupMGPARA: The processor id you requested (%d) \
02051 is not within the range of processors available (0-%d)\n", rank, (nproc-1));
02052          return 0;
02053      }
02054  }
02055
02056  /* Calculate the processor's coordinates in the processor grid */
02057  kp = (int)floor(rank/(npx*ncpy));
02058  jp = (int)floor((rank-kp*npx*ncpy)/npx);
02059  ip = rank - kp*npx*ncpy - jp*npx;
02060  Vnm_print(0, "Nosh_setupMGPARA: Hello world from PE (%d, %d, %d)\n",
02061            ip, jp, kp);
02062
02063  /* Calculate effective overlap fractions for uneven processor distributions */
02064  if (npx == 1) xofrac = 0.0;
02065  else xofrac = ofrac;
02066  if (ncpy == 1) yofrac = 0.0;
02067  else yofrac = ofrac;
02068  if (npz == 1) zofrac = 0.0;
02069  else zofrac = ofrac;
02070
02071  /* Calculate the global grid size and spacing */
02072  xDisj = (int)VFFLOOR(mgparm->dime[0]/(1 + 2*xofrac) + 0.5);
02073  xeffGlob = npx*xDisj;
02074  hx = mgparm->fglen[0]/(double)(xeffGlob-1);
02075  yDisj = (int)VFFLOOR(mgparm->dime[1]/(1 + 2*yofrac) + 0.5);
02076  yeffGlob = npx*yDisj;
02077  hy = mgparm->fglen[1]/(double)(yeffGlob-1);
02078  zDisj = (int)VFFLOOR(mgparm->dime[2]/(1 + 2*zofrac) + 0.5);
02079  zeffGlob = npz*zDisj;
02080  hzed = mgparm->fglen[2]/(double)(zeffGlob-1);
02081  Vnm_print(0, "Nosh_setupMGPARA: Global Grid size = (%d, %d, %d)\n",
02082            xeffGlob, yeffGlob, zeffGlob);
02083  Vnm_print(0, "Nosh_setupMGPARA: Global Grid Spacing = (%.3f, %.3f, %.3f)\n",
02084            hx, hy, hzed);
02085  Vnm_print(0, "Nosh_setupMGPARA: Processor Grid Size = (%d, %d, %d)\n",
02086            xDisj, yDisj, zDisj);
02087
02088  /* Calculate the maximum and minimum processor grid points */
02089  xigminDisj = ip*xDisj;
02090  xigmaxDisj = xigminDisj + xDisj - 1;
02091  yigminDisj = jp*yDisj;
02092  yigmaxDisj = yigminDisj + yDisj - 1;
02093  zigminDisj = kp*zDisj;
02094  zigmaxDisj = zigminDisj + zDisj - 1;
02095  Vnm_print(0, "Nosh_setupMGPARA: Min Grid Points for this proc. (%d, %d, %d)\n",
02096            xigminDisj, yigminDisj, zigminDisj);
02097  Vnm_print(0, "Nosh_setupMGPARA: Max Grid Points for this proc. (%d, %d, %d)\n",
02098            xigmaxDisj, yigmaxDisj, zigmaxDisj);
02099
02100
02101  /* Calculate the disjoint partition length and center displacement */
02102  xminDisj = VMAX2(hx*(xigminDisj-0.5), 0.0);
02103  xmaxDisj = VMIN2(hx*(xigmaxDisj+0.5), mgparm->fglen[0]);
02104  xlenDisj = xmaxDisj - xminDisj;
02105  yminDisj = VMAX2(hy*(yigminDisj-0.5), 0.0);
02106  ymaxDisj = VMIN2(hy*(yigmaxDisj+0.5), mgparm->fglen[1]);
02107  ylenDisj = ymaxDisj - yminDisj;
02108  zminDisj = VMAX2(hzed*(zigminDisj-0.5), 0.0);
02109  zmaxDisj = VMIN2(hzed*(zigmaxDisj+0.5), mgparm->fglen[2]);
02110  zlenDisj = zmaxDisj - zminDisj;
02111
02112  xcen = 0.5*mgparm->fglen[0];
02113  ycen = 0.5*mgparm->fglen[1];
02114  zcen = 0.5*mgparm->fglen[2];
02115
02116  xcenDisj = xminDisj + 0.5*xlenDisj - xcen;
02117  ycenDisj = yminDisj + 0.5*ylenDisj - ycen;
02118  zcenDisj = zminDisj + 0.5*zlenDisj - zcen;
02119  if (VABS(xcenDisj) < VSMALL) xcenDisj = 0.0;
02120  if (VABS(ycenDisj) < VSMALL) ycenDisj = 0.0;
02121  if (VABS(zcenDisj) < VSMALL) zcenDisj = 0.0;
02122
02123  Vnm_print(0, "Nosh_setupMGPARA: Disj part length = (%g, %g, %g)\n",
02124            xlenDisj, ylenDisj, zlenDisj);
02125  Vnm_print(0, "Nosh_setupMGPARA: Disj part center displacement = (%g, %g, %g)\n",
02126            xcenDisj, ycenDisj, zcenDisj);
02127

```

```

02128      /* Calculate the overlapping partition length and center displacement */
02129      xOlapReg = 0;
02130      yOlapReg = 0;
02131      zOlapReg = 0;
02132      if (npx != 1) xOlapReg = (int)VFFLOOR(xofrac*mgparm->fglen[0]/npx/hx + 0.5) + 1;
02133      if (npy != 1) yOlapReg = (int)VFFLOOR(yofrac*mgparm->fglen[1]/npy/hy + 0.5) + 1;
02134      if (npz != 1) zOlapReg = (int)VFFLOOR(zofrac*mgparm->fglen[2]/npz/hzed + 0.5) + 1;
02135
02136      Vnm_print(0, "Nosh_setupMGPARA: No. of Grid Points in Overlap (%d, %d, %d)\n",
02137                xOlapReg, yOlapReg, zOlapReg);
02138
02139      if (ip == 0) xigminOlap = 0;
02140      else if (ip == (npx - 1)) xigminOlap = xeffGlob - mgparm->dime[0];
02141      else xigminOlap = xigminDisj - xOlapReg;
02142      xigmaxOlap = xigminOlap + mgparm->dime[0] - 1;
02143
02144      if (jp == 0) yigminOlap = 0;
02145      else if (jp == (npy - 1)) yigminOlap = yeffGlob - mgparm->dime[1];
02146      else yigminOlap = yigminDisj - yOlapReg;
02147      yigmaxOlap = yigminOlap + mgparm->dime[1] - 1;
02148
02149      if (kp == 0) zigminOlap = 0;
02150      else if (kp == (npz - 1)) zigminOlap = zeffGlob - mgparm->dime[2];
02151      else zigminOlap = zigminDisj - zOlapReg;
02152      zigmaxOlap = zigminOlap + mgparm->dime[2] - 1;
02153
02154      Vnm_print(0, "Nosh_setupMGPARA: Min Grid Points with Overlap (%d, %d, %d)\n",
02155                xigminOlap, yigminOlap, zigminOlap);
02156      Vnm_print(0, "Nosh_setupMGPARA: Max Grid Points with Overlap (%d, %d, %d)\n",
02157                xigmaxOlap, yigmaxOlap, zigmaxOlap);
02158
02159      xminOlap = hx * xigminOlap;
02160      xmaxOlap = hx * xigmaxOlap;
02161      yminOlap = hy * yigminOlap;
02162      ymaxOlap = hy * yigmaxOlap;
02163      zminOlap = hzed * zigminOlap;
02164      zmaxOlap = hzed * zigmaxOlap;
02165
02166      xlenOlap = xmaxOlap - xminOlap;
02167      ylenOlap = ymaxOlap - yminOlap;
02168      zlenOlap = zmaxOlap - zminOlap;
02169
02170      xcentOlap = (xminOlap + 0.5*xlenOlap) - xcent;
02171      ycentOlap = (yminOlap + 0.5*ylenOlap) - ycent;
02172      zcentOlap = (zminOlap + 0.5*zlenOlap) - zcent;
02173      if (VABS(xcentOlap) < VSMALL) xcentOlap = 0.0;
02174      if (VABS(ycentOlap) < VSMALL) ycentOlap = 0.0;
02175      if (VABS(zcentOlap) < VSMALL) zcentOlap = 0.0;
02176
02177      Vnm_print(0, "Nosh_setupMGPARA: Olap part length = (%g, %g, %g)\n",
02178                xlenOlap, ylenOlap, zlenOlap);
02179      Vnm_print(0, "Nosh_setupMGPARA: Olap part center displacement = (%g, %g, %g)\n",
02180                xcentOlap, ycentOlap, zcentOlap);
02181
02182
02183      /* Calculate the boundary flags:
02184         Flags are set to 1 when another processor is present along the boundary
02185         Flags are otherwise set to 0. */
02186
02187      if (ip == 0) mgparm->partDisjOwnSide[VAPBS_LEFT] = 0;
02188      else mgparm->partDisjOwnSide[VAPBS_LEFT] = 1;
02189      if (ip == (npx-1)) mgparm->partDisjOwnSide[VAPBS_RIGHT] = 0;
02190      else mgparm->partDisjOwnSide[VAPBS_RIGHT] = 1;
02191      if (jp == 0) mgparm->partDisjOwnSide[VAPBS_BACK] = 0;
02192      else mgparm->partDisjOwnSide[VAPBS_BACK] = 1;
02193      if (jp == (npy-1)) mgparm->partDisjOwnSide[VAPBS_FRONT] = 0;
02194      else mgparm->partDisjOwnSide[VAPBS_FRONT] = 1;
02195      if (kp == 0) mgparm->partDisjOwnSide[VAPBS_DOWN] = 0;
02196      else mgparm->partDisjOwnSide[VAPBS_DOWN] = 1;
02197      if (kp == (npz-1)) mgparm->partDisjOwnSide[VAPBS_UP] = 0;
02198      else mgparm->partDisjOwnSide[VAPBS_UP] = 1;
02199
02200      Vnm_print(0, "Nosh_setupMGPARA: partDisjOwnSide[LEFT] = %d\n",
02201                mgparm->partDisjOwnSide[VAPBS_LEFT]);
02202      Vnm_print(0, "Nosh_setupMGPARA: partDisjOwnSide[RIGHT] = %d\n",
02203                mgparm->partDisjOwnSide[VAPBS_RIGHT]);
02204      Vnm_print(0, "Nosh_setupMGPARA: partDisjOwnSide[FRONT] = %d\n",
02205                mgparm->partDisjOwnSide[VAPBS_FRONT]);
02206      Vnm_print(0, "Nosh_setupMGPARA: partDisjOwnSide[BACK] = %d\n",
02207                mgparm->partDisjOwnSide[VAPBS_BACK]);
02208      Vnm_print(0, "Nosh_setupMGPARA: partDisjOwnSide[UP] = %d\n",

```

```

02209         mgparm->partDisjOwnSide[VAPBS_UP]);
02210     Vnm_print(0, "Nosh_setupMGPARA: partDisjOwnSide[DOWN] = %d\n",
02211         mgparm->partDisjOwnSide[VAPBS_DOWN]);
02212
02213     /* Set the mesh parameters */
02214     mgparm->fglen[0] = xlenOlap;
02215     mgparm->fglen[1] = ylenOlap;
02216     mgparm->fglen[2] = zlenOlap;
02217     mgparm->partDisjLength[0] = xlenDisj;
02218     mgparm->partDisjLength[1] = ylenDisj;
02219     mgparm->partDisjLength[2] = zlenDisj;
02220     mgparm->partDisjCenter[0] = mgparm->fcenter[0] + xcentDisj;
02221     mgparm->partDisjCenter[1] = mgparm->fcenter[1] + ycentDisj;
02222     mgparm->partDisjCenter[2] = mgparm->fcenter[2] + zcentDisj;
02223     mgparm->fcenter[0] += xcentOlap;
02224     mgparm->fcenter[1] += ycentOlap;
02225     mgparm->fcenter[2] += zcentOlap;
02226
02227     Vnm_print(0, "Nosh_setupCalcMGPARA (%s, %d): Set up *relative* partition \
02228 centers...\n", __FILE__, __LINE__);
02229     Vnm_print(0, "Nosh_setupCalcMGPARA (%s, %d): Absolute centers will be set \
02230 in Nosh_setupMGAUTO\n", __FILE__, __LINE__);
02231     Vnm_print(0, "Nosh_setupCalcMGPARA (%s, %d): partDisjCenter = %g %g %g\n",
02232         __FILE__, __LINE__,
02233         mgparm->partDisjCenter[0],
02234         mgparm->partDisjCenter[1],
02235         mgparm->partDisjCenter[2]);
02236     Vnm_print(0, "Nosh_setupCalcMGPARA (%s, %d): ccenter = %g %g %g\n",
02237         __FILE__, __LINE__,
02238         mgparm->ccenter[0],
02239         mgparm->ccenter[1],
02240         mgparm->ccenter[2]);
02241     Vnm_print(0, "Nosh_setupCalcMGPARA (%s, %d): fcenter = %g %g %g\n",
02242         __FILE__, __LINE__,
02243         mgparm->fcenter[0],
02244         mgparm->fcenter[1],
02245         mgparm->fcenter[2]);
02246
02247
02248     /* Setup the automatic focusing calculations associated with this processor */
02249     return Nosh_setupCalcMGAUTO(thee, elec);
02250
02251 }
02252
02253 VPUBLIC int Nosh_parseFEM(
02254     NOSH *thee,
02255     Vio *sock,
02256     NOSH_calc *elec
02257 ) {
02258
02259     char tok[VMAX_BUFSIZE];
02260     FEMparm *feparm = VNULL;
02261     PBEParm *pbeparm = VNULL;
02262     int rc;
02263     Vrc_Codes vrc;
02264
02265     /* Check the arguments */
02266     if (thee == VNULL) {
02267         Vnm_print(2, "Nosh_parseFEM: Got NULL thee!\n");
02268         return 0;
02269     }
02270     if (sock == VNULL) {
02271         Vnm_print(2, "Nosh_parseFEM: Got pointer to NULL socket!\n");
02272         return 0;
02273     }
02274     if (elec == VNULL) {
02275         Vnm_print(2, "Nosh_parseFEM: Got pointer to NULL elec object!\n");
02276         return 0;
02277     }
02278     feparm = elec->feparm;
02279     if (feparm == VNULL) {
02280         Vnm_print(2, "Nosh_parseFEM: Got pointer to NULL feparm object!\n");
02281         return 0;
02282     }
02283     pbeparm = elec->pbeparm;
02284     if (pbeparm == VNULL) {
02285         Vnm_print(2, "Nosh_parseFEM: Got pointer to NULL pbeparm object!\n");
02286         return 0;
02287     }
02288
02289     Vnm_print(0, "Nosh_parseFEM: Parsing parameters for FEM calculation\n");

```

```

02290
02291 /* Start snarfing tokens from the input stream */
02292 rc = 1;
02293 while (Vio_scanf(sock, "%s", tok) == 1) {
02294
02295     Vnm_print(0, "Nosh_parseFEM: Parsing %s...\n", tok);
02296
02297     /* See if it's an END token */
02298     if (Vstring_strcasecmp(tok, "end") == 0) {
02299         feparm->parsed = 1;
02300         pbeparm->parsed = 1;
02301         rc = 1;
02302         break;
02303     }
02304
02305     /* Pass the token through a series of parsers */
02306     rc = PBEparam_parseToken(pbeparm, tok, sock);
02307     if (rc == -1) {
02308         Vnm_print(0, "Nosh_parseFEM: parsePBE error!\n");
02309         break;
02310     } else if (rc == 0) {
02311         /* Pass the token to the generic MG parser */
02312         vrc = FEMparam_parseToken(feparm, tok, sock);
02313         if (vrc == VRC_FAILURE) {
02314             Vnm_print(0, "Nosh_parseFEM: parseMG error!\n");
02315             break;
02316         } else if (vrc == VRC_WARNING) {
02317             /* We ran out of parsers! */
02318             Vnm_print(2, "Nosh: Unrecognized keyword: %s\n", tok);
02319             break;
02320         }
02321     }
02322 }
02323
02324 /* Handle various errors arising in the token-snarfing loop -- these all
02325  * just result in simple returns right now */
02326 if (rc == -1) return 0;
02327 if (rc == 0) return 0;
02328
02329 /* Check the status of the parameter objects */
02330 if ((!FEMparam_check(feparm)) || (!PBEparam_check(pbeparm))) {
02331     Vnm_print(2, "Nosh: FEM parameters not set correctly!\n");
02332     return 0;
02333 }
02334
02335 return 1;
02336 }
02337 }
02338
02339 VPRIVATE int Nosh_setupCalcFEMANUAL(
02340     Nosh *thee,
02341     Nosh_calc *elec
02342 ) {
02343
02344     FEMparam *feparm = VNULL;
02345     PBEparam *pbeparm = VNULL;
02346     Nosh_calc *calc = VNULL;
02347
02348     VASSERT(thee != VNULL);
02349     VASSERT(elec != VNULL);
02350     feparm = elec->feparm;
02351     VASSERT(feparm != VNULL);
02352     pbeparm = elec->pbeparm;
02353     VASSERT(pbeparm);
02354
02355     /* Check to see if he have any room left for this type of
02356      * calculation, if so: set the calculation type, update the number
02357      * of calculations of this type, and parse the rest of the section
02358      */
02359     if (thee->ncalc >= NOSH_MAXCALC) {
02360         Vnm_print(2, "Nosh: Too many calculations in this run!\n");
02361         Vnm_print(2, "Nosh: Current max is %d; ignoring this calculation\n",
02362             NOSH_MAXCALC);
02363         return 0;
02364     }
02365     thee->calc[thee->ncalc] = Nosh_calc_ctor(NCT_FEM);
02366     calc = thee->calc[thee->ncalc];
02367     (thee->ncalc)++;
02368
02369     /* Copy over contents of ELEC */
02370     Nosh_calc_copy(calc, elec);

```

```

02371
02372
02373     return 1;
02374 }
02375
02376 VPUBLIC int NOsh_parseAPOL(
02377     NOsh *thee,
02378     Vio *sock,
02379     NOsh_calc *elec
02380 ) {
02381
02382     char tok[VMAX_BUFSIZE];
02383     APOLparm *apolparm = VNULL;
02384     int rc;
02385
02386     /* Check the arguments */
02387     if (thee == VNULL) {
02388         Vnm_print(2, "NOsh_parseAPOL: Got NULL thee!\n");
02389         return 0;
02390     }
02391     if (sock == VNULL) {
02392         Vnm_print(2, "NOsh_parseAPOL: Got pointer to NULL socket!\n");
02393         return 0;
02394     }
02395     if (elec == VNULL) {
02396         Vnm_print(2, "NOsh_parseAPOL: Got pointer to NULL elec object!\n");
02397         return 0;
02398     }
02399     apolparm = elec->apolparm;
02400     if (apolparm == VNULL) {
02401         Vnm_print(2, "NOsh_parseAPOL: Got pointer to NULL apolparm object!\n");
02402         return 0;
02403     }
02404
02405     Vnm_print(0, "NOsh_parseAPOL: Parsing parameters for APOL calculation\n");
02406
02407     /* Start snarfing tokens from the input stream */
02408     rc = 1;
02409     while (Vio_scanf(sock, "%s", tok) == 1) {
02410
02411         Vnm_print(0, "NOsh_parseAPOL: Parsing %s...\n", tok);
02412         /* See if it's an END token */
02413         if (Vstring_strcasecmp(tok, "end") == 0) {
02414             apolparm->parsed = 1;
02415             rc = 1;
02416             break;
02417         }
02418
02419         /* Pass the token through a series of parsers */
02420         /* Pass the token to the generic non-polar parser */
02421         rc = APOLparm_parseToken(apolparm, tok, sock);
02422         if (rc == -1) {
02423             Vnm_print(0, "NOsh_parseFEM: parseMG error!\n");
02424             break;
02425         } else if (rc == 0) {
02426             /* We ran out of parsers! */
02427             Vnm_print(2, "NOsh: Unrecognized keyword: %s\n", tok);
02428             break;
02429         }
02430     }
02431 }
02432
02433 /* Handle various errors arising in the token-snarfing loop -- these all
02434    * just result in simple returns right now */
02435 if (rc == -1) return 0;
02436 if (rc == 0) return 0;
02437
02438 /* Check the status of the parameter objects */
02439 if (!APOLparm_check(apolparm)) {
02440     Vnm_print(2, "NOsh: APOL parameters not set correctly!\n");
02441     return 0;
02442 }
02443
02444 return 1;
02445 }
02446 }
02447
02448
02449 VPRIVATE int NOsh_setupCalcAPOL(
02450     NOsh *thee,
02451     NOsh_calc *apol

```

```

02452         ) {
02453
02454     NOsh_calc *calc = VNULL;
02455
02456     VASSERT(thee != VNULL);
02457     VASSERT(apol != VNULL);
02458
02459     /* Check to see if he have any room left for this type of
02460      * calculation, if so: set the calculation type, update the number
02461      * of calculations of this type, and parse the rest of the section
02462      */
02463     if (thee->ncalc >= NOSH_MAXCALC) {
02464         Vnm_print(2, "Nosh: Too many calculations in this run!\n");
02465         Vnm_print(2, "Nosh: Current max is %d; ignoring this calculation\n",
02466                 NOSH_MAXCALC);
02467         return 0;
02468     }
02469     thee->calc[thee->ncalc] = NOsh_calc_ctor(NCT_APOL);
02470     calc = thee->calc[thee->ncalc];
02471     (thee->ncalc)++;
02472
02473     /* Copy over contents of APOL */
02474     NOsh_calc_copy(calc, apol);
02475
02476     return 1;
02477 }
02478
02479 VPRIVATE int NOsh_setupCalcBEMMANUAL(
02480     NOsh *thee,
02481     NOsh_calc *elec
02482 ) {
02483
02484     BEMparm *bemparm = VNULL;
02485     PBEParm *pbeparm = VNULL;
02486     NOsh_calc *calc = VNULL;
02487
02488     if (thee == VNULL) {
02489         Vnm_print(2, "Nosh_setupCalcBEMMANUAL: Got NULL thee!\n");
02490         return 0;
02491     }
02492     if (elec == VNULL) {
02493         Vnm_print(2, "Nosh_setupCalcBEMMANUAL: Got NULL calc!\n");
02494         return 0;
02495     }
02496     bemparm = elec->bemparm;
02497     if (bemparm == VNULL) {
02498         Vnm_print(2, "Nosh_setupCalcBEMMANUAL: Got NULL mgparm -- was this calculation \
02499 set up?\n");
02500         return 0;
02501     }
02502     pbeparm = elec->pbeparm;
02503     if (pbeparm == VNULL) {
02504         Vnm_print(2, "Nosh_setupCalcBEMMANUAL: Got NULL pbeparm -- was this calculation \
02505 set up?\n");
02506         return 0;
02507     }
02508
02509     /* Set up missing BEM parameters */
02510     if (bemparm->settree_order == 0) {
02511         bemparm->tree_order=1;
02512     }
02513     if (bemparm->settree_n0 == 0) {
02514         bemparm->tree_n0=500;
02515     }
02516     if (bemparm->setmac == 0) {
02517         bemparm->mac=0.8;
02518     }
02519
02520     /* Check to see if he have any room left for this type of calculation, if
02521      * so: set the calculation type, update the number of calculations of this type,
02522      * and parse the rest of the section */
02523     if (thee->ncalc >= NOSH_MAXCALC) {
02524         Vnm_print(2, "Nosh: Too many calculations in this run!\n");
02525         Vnm_print(2, "Nosh: Current max is %d; ignoring this calculation\n",
02526                 NOSH_MAXCALC);
02527         return 0;
02528     }
02529
02530     }
02531
02532

```

```

02533      /* Get the next calculation object and increment the number of calculations */
02534      thee->calc[thee->ncalc] = Nosh_calc_ctor(NCT_BEM);
02535      calc = thee->calc[thee->ncalc];
02536      (thee->ncalc)++;
02537
02538      /* Copy over contents of ELEC */
02539      Nosh_calc_copy(calc, elec);
02540
02541
02542      return 1;
02543 }
02544
02545 VPRIVATE int Nosh_setupCalcGEOFLOWMANUAL(
02546     Nosh *thee,
02547     Nosh_calc *elec
02548 ) {
02549
02550     GEOFLOWparm *parm = VNULL;
02551     APOLparm *apolparm = VNULL;
02552     PBEParm *pbeparm = VNULL;
02553     Nosh_calc *calc = VNULL;
02554
02555     if (thee == VNULL) {
02556         Vnm_print(2, "Nosh_setupCalcGEOFLOWMANUAL: Got NULL thee!\n");
02557         return 0;
02558     }
02559     if (elec == VNULL) {
02560         Vnm_print(2, "Nosh_setupCalcGEOFLOWMANUAL: Got NULL calc!\n");
02561         return 0;
02562     }
02563     parm = elec->geoflowparm;
02564     if (parm == VNULL) {
02565         Vnm_print(2, "Nosh_setupCalcGEOFLOWMANUAL: Got NULL geoflowparm -- was this calculation \
02566 set up?\n");
02567         return 0;
02568     }
02569     apolparm = elec->apolparm;
02570     if (parm == VNULL) {
02571         Vnm_print(2, "Nosh_setupCalcGEOFLOWMANUAL: Got NULL apolparm -- was this calculation \
02572 set up?\n");
02573         return 0;
02574     }
02575     pbeparm = elec->pbeparm;
02576     if (pbeparm == VNULL) {
02577         Vnm_print(2, "Nosh_setupCalcGEOFLOWMANUAL: Got NULL pbeparm -- was this calculation \
02578 set up?\n");
02579         return 0;
02580     }
02581
02582     /* Check to see if he have any room left for this type of calculation, if
02583        so: set the calculation type, update the number of calculations of this type,
02584        and parse the rest of the section */
02585     if (thee->ncalc >= NOSH_MAXCALC) {
02586         Vnm_print(2, "Nosh: Too many calculations in this run!\n");
02587         Vnm_print(2, "Nosh: Current max is %d; ignoring this calculation\n",
02588             NOSH_MAXCALC);
02589         return 0;
02590     }
02591
02592     /* Get the next calculation object and increment the number of calculations */
02593     thee->calc[thee->ncalc] = Nosh_calc_ctor(NCT_GEOFLOW);
02594     calc = thee->calc[thee->ncalc];
02595     (thee->ncalc)++;
02596
02597     /* Copy over contents of ELEC */
02598     Nosh_calc_copy(calc, elec);
02599
02600     return 1;
02601 }
02602
02603
02604 VPUBLIC int Nosh_parseBEM(
02605     Nosh *thee,
02606     Vio *sock,
02607     Nosh_calc *elec
02608 ) {
02609
02610     char tok[VMAX_BUFSIZE];
02611     BEMparm *bemparm = VNULL;
02612     PBEParm *pbeparm = VNULL;
02613     int rc;

```

```

02614
02615     /* Check the arguments */
02616     if (thee == VNULL) {
02617         Vnm_print(2, "Nosh: Got NULL thee!\n");
02618         return 0;
02619     }
02620     if (sock == VNULL) {
02621         Vnm_print(2, "Nosh: Got pointer to NULL socket!\n");
02622         return 0;
02623     }
02624     if (elec == VNULL) {
02625         Vnm_print(2, "Nosh: Got pointer to NULL elec object!\n");
02626         return 0;
02627     }
02628     bemparm = elec->bemparm;
02629     if (bemparm == VNULL) {
02630         Vnm_print(2, "Nosh: Got pointer to NULL bemparm object!\n");
02631         return 0;
02632     }
02633     pbeparm = elec->pbeparm;
02634     if (pbeparm == VNULL) {
02635         Vnm_print(2, "Nosh: Got pointer to NULL pbeparm object!\n");
02636         return 0;
02637     }
02638
02639     Vnm_print(0, "Nosh_parseBEM: Parsing parameters for BEM calculation\n");
02640
02641
02642     /* Start snarfing tokens from the input stream */
02643     rc = 1;
02644     while (Vio_scanf(sock, "%s", tok) == 1) {
02645
02646         Vnm_print(0, "Nosh_parseBEM: Parsing %s...\n", tok);
02647
02648         /* See if it's an END token */
02649         if (Vstring_strcasecmp(tok, "end") == 0) {
02650             bemparm->parsed = 1;
02651             pbeparm->parsed = 1;
02652             rc = 1;
02653             break;
02654         }
02655
02656         /* Pass the token through a series of parsers */
02657         rc = PBeparm_parseToken(pbeparm, tok, sock);
02658         if (rc == -1) {
02659             Vnm_print(0, "Nosh_parseBEM: parsePBE error!\n");
02660             break;
02661         } else if (rc == 0) {
02662             /* Pass the token to the generic BEM parser */
02663             rc = BEMparm_parseToken(bemparm, tok, sock);
02664             if (rc == -1) {
02665                 Vnm_print(0, "Nosh_parseBEM: parseBEM error!\n");
02666                 break;
02667             } else if (rc == 0) {
02668                 /* We ran out of parsers! */
02669                 Vnm_print(2, "Nosh: Unrecognized keyword: %s\n", tok);
02670                 break;
02671             }
02672         }
02673     }
02674
02675     /* Handle various errors arising in the token-snarfing loop -- these all
02676        just result in simple returns right now */
02677     if (rc == -1) return 0;
02678     if (rc == 0) return 0;
02679
02680     /* Check the status of the parameter objects */
02681     if ((BEMparm_check(bemparm) == VRC_FAILURE) || (!
PBeparm_check(pbeparm))) {
02682         Vnm_print(2, "Nosh: BEM parameters not set correctly!\n");
02683         return 0;
02684     }
02685
02686     return 1;
02687 }
02688
02689 VPUBLIC int Nosh_parseGEOFLOW(
02690     Nosh *thee,
02691     Vio *sock,
02692     Nosh_calc *elec
02693 ) {

```



```

02694
02695     char tok[VMAX_BUFSIZE];
02696     GEOFLOWparm *parm = VNULL;
02697     APOLparm *apolparm = VNULL;
02698     PBEparm *pbeparm = VNULL;
02699     int rc;
02700
02701     /* Check the arguments */
02702     if (thee == VNULL) {
02703         Vnm_print(2, "NOsh: Got NULL thee!\n");
02704         return 0;
02705     }
02706     if (sock == VNULL) {
02707         Vnm_print(2, "NOsh: Got pointer to NULL socket!\n");
02708         return 0;
02709     }
02710     if (elec == VNULL) {
02711         Vnm_print(2, "NOsh: Got pointer to NULL elec object!\n");
02712         return 0;
02713     }
02714     parm = elec->geoflowparm;
02715     if (parm == VNULL) {
02716         Vnm_print(2, "NOsh: Got pointer to NULL geoflowparm object!\n");
02717         return 0;
02718     }
02719     apolparm = elec->apolparm;
02720     if (apolparm == VNULL) {
02721         Vnm_print(2, "NOsh: Got pointer to NULL apolparm object!\n");
02722         return 0;
02723     }
02724     pbeparm = elec->pbeparm;
02725     if (pbeparm == VNULL) {
02726         Vnm_print(2, "NOsh: Got pointer to NULL pbeparm object!\n");
02727         return 0;
02728     }
02729
02730     Vnm_print(0, "NOsh_parseGEOFLOW: Parsing parameters for GEOFLOW calculation\n");
02731
02732
02733     /* Start snarfing tokens from the input stream */
02734     rc = 1;
02735     while (Vio_scanf(sock, "%s", tok) == 1) {
02736
02737         Vnm_print(0, "NOsh_parseGEOFLOW: Parsing %s...\n", tok);
02738
02739         /* See if it's an END token */
02740         if (Vstring_strcasecmp(tok, "end") == 0) {
02741             parm->parsed = 1;
02742             pbeparm->parsed = 1;
02743             apolparm->parsed = 1;
02744             rc = 1;
02745             break;
02746         }
02747
02748         if (Vstring_strcasecmp(tok, "ion") == 0) {
02749             Vnm_print(2, "parseGEOFLOW: WARNING! ion not implemented for geometric flow!\n");
02750         }
02751
02752         /* Pass the token through a series of parsers */
02753         rc = PBEparm_parseToken(pbeparm, tok, sock);
02754         if (rc == -1) {
02755             Vnm_print(0, "NOsh_parseGEOFLOW: parsePBE error!\n");
02756             break;
02757         } else if (rc == 0) {
02758             /* Pass the token to the generic GEOFLOW parser */
02759             rc = APOLparm_parseToken(apolparm, tok, sock);
02760             if (rc == -1) {
02761                 Vnm_print(0, "NOsh_parseAPOL: parseAPOL error!\n");
02762                 break;
02763             } else if (rc == 0) {
02764                 rc = GEOFLOWparm_parseToken(parm, tok, sock);
02765                 if (rc == -1) {
02766                     Vnm_print(0, "NOsh_parseGEOFLOW: parseGEOFLOW error!\n");
02767                     break;
02768                 } else if (rc == 0) {
02769                     /* We ran out of parsers! */
02770                     Vnm_print(2, "NOsh: Unrecognized keyword: %s\n", tok);
02771                     break;
02772                 }
02773             }
02774         }
02775     }

```

```

02775     }
02776
02777     pbeparm->setsrfm=1;
02778     pbeparm->srاد=0.0;
02779     pbeparm->setsrad=1;
02780     pbeparm->settemp=1;
02781
02782     /* Handle various errors arising in the token-snarfing loop -- these all
02783        just result in simple returns right now */
02784     if (rc == -1) return 0;
02785     if (rc == 0) return 0;
02786
02787     /* Check the status of the parameter objects */
02788     if ((GEOFLOWparm_check(parm) == VRC_FAILURE) || (!
PBEparm_check(pbeparm))) {
02789         Vnm_print(2, "NOsh: GEOFLOW parameters not set correctly!\n");
02790         return 0;
02791     }
02792
02793     return 1;
02794 }
02795

```

10.33 src/generic/nosh.h File Reference

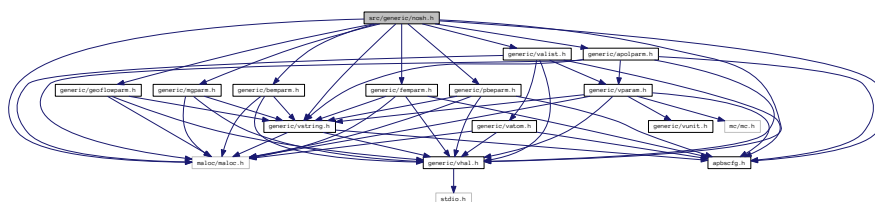
Contains declarations for class NOsh.

```

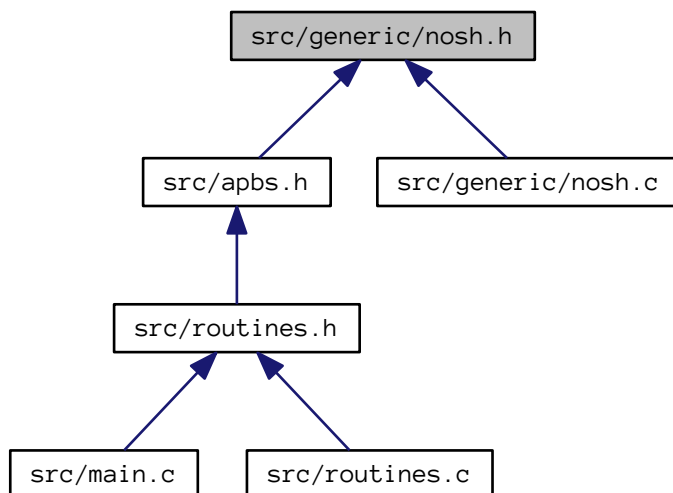
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/vstring.h"
#include "generic/pbeparm.h"
#include "generic/mgparm.h"
#include "generic/apolparm.h"
#include "generic/femparm.h"
#include "generic/valist.h"
#include "generic/bemparm.h"
#include "generic/geoflowparm.h"

```

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.

Macros

- #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, [NCT_BEM](#) =3, [NCT_GEOFLOW](#) =4 }
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)
FORTTRAN stub to construct NOsh.
- VEXTERNC void [NOsh_dtor](#) ([NOsh](#) **thee)
Object destructor.
- VEXTERNC void [NOsh_dtor2](#) ([NOsh](#) *thee)
FORTTRAN 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.

10.33.1 Detailed Description

Contains declarations for class NOsh.

Version

\$Id\$

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [nosh.h](#).

10.34 nosh.h

```
00001
00062 #ifndef _NOSH_H_
00063 #define _NOSH_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "malloc/malloc.h"
00068
00069 #include "generic/vhal.h"
00070 #include "generic/vstring.h"
00071 #include "generic/pbeparm.h"
00072 #include "generic/mgparm.h"
00073 #include "generic/apolparm.h"
00074 #include "generic/femparm.h"
00075 #include "generic/valist.h"
00076 #include "generic/bemparm.h"
00077 #include "generic/geoflowparm.h"
00078
00081 #define NOSH_MAXMOL 20
00082
00085 #define NOSH_MAXCALC 20
00086
00089 #define NOSH_MAXPRINT 20
00090
00093 #define NOSH_MAXPOP 20
00094
00099 enum eNosh_MolFormat {
00100     NMF_PQR=0,
00101     NMF_PDB=1,
00102     NMF_XML=2
00103 };
00104
00109 typedef enum eNosh_MolFormat Nosh_MolFormat;
00110
00115 enum eNosh_CalcType {
00116     NCT_MG=0,
00117     NCT_FEM=1,
00118     NCT_APOL=2,
00119     NCT_BEM=3,
00120     NCT_GEOFLOW=4
00121 };
00122
00127 typedef enum eNosh_CalcType Nosh_CalcType;
00128
00133 enum eNosh_ParmFormat {
00134     NPF_FLAT=0,
00135     NPF_XML=1
00136 };
00137
00142 typedef enum eNosh_ParmFormat Nosh_ParmFormat;
00143
00148 enum eNosh_PrintType {
00149     NPT_ENERGY=0,
00150     NPT_FORCE=1,
00151     NPT_ELECENERGY,
00152     NPT_ELECFORCE,
00153     NPT_APOLENERGY,
00154     NPT_APOLFORCE
00155 };
00156
00161 typedef enum eNosh_PrintType Nosh_PrintType;
00162
00168 struct sNosh_calc {
00169     MGparm *mgparm;
00170     FEMparm *femparm;
00171     BEMparm *bemparm;
00172     GEOFLOWparm *geoflowparm;
00173     PBEparm *pbeparm;
00174     APOLparm *apolparm;
00175     Nosh_CalcType calctype;
00176 };
00177
00182 typedef struct sNosh_calc Nosh_calc;
00183
00189 struct sNosh {
00190
00191     Nosh_calc *calc[NOSH_MAXCALC];
00194     int ncalc;
```

```

00196     Nosh_calc *elec[NOSH_MAXCALC];
00199     int nelec;
00202     Nosh_calc *apol[NOSH_MAXCALC];
00205     int napol;
00208     int ispara;
00209     int proc_rank;
00210     int proc_size;
00211     int bogus;
00215     int elec2calc[NOSH_MAXCALC];
00223     int apol2calc[NOSH_MAXCALC];
00225     int nmol;
00226     char molpath[NOSH_MAXMOL][VMAX_ARGLEN];
00227     Nosh_MolFormat molfmt[NOSH_MAXMOL];
00228     Valist *alist[NOSH_MAXMOL];
00230     int gotparm;
00231     char parmpath[VMAX_ARGLEN];
00232     Nosh_ParmFormat parmfmt;
00233     int ndiel;
00234     char dielXpath[NOSH_MAXMOL][VMAX_ARGLEN];
00236     char dielYpath[NOSH_MAXMOL][VMAX_ARGLEN];
00238     char dielZpath[NOSH_MAXMOL][VMAX_ARGLEN];
00240     Vdata_Format dielfmt[NOSH_MAXMOL];
00241     int nkappa;
00242     char kappapath[NOSH_MAXMOL][VMAX_ARGLEN];
00243     Vdata_Format kappafmt[NOSH_MAXMOL];
00244     int npot;
00245     char potpath[NOSH_MAXMOL][VMAX_ARGLEN];
00246     Vdata_Format potfmt[NOSH_MAXMOL];
00247     int ncharge;
00248     char chargepath[NOSH_MAXMOL][VMAX_ARGLEN];
00249     Vdata_Format chargefmt[NOSH_MAXMOL];
00250     int nmesh;
00251     char meshpath[NOSH_MAXMOL][VMAX_ARGLEN];
00252     Vdata_Format meshfmt[NOSH_MAXMOL];
00253     int nprint;
00254     Nosh_PrintType printwhat[NOSH_MAXPRINT];
00256     int printnarg[NOSH_MAXPRINT];
00257     int printcalc[NOSH_MAXPRINT][NOSH_MAXPOP];
00258     int printop[NOSH_MAXPRINT][NOSH_MAXPOP];
00260     int parsed;
00261     char elecname[NOSH_MAXCALC][VMAX_ARGLEN];
00263     char apolname[NOSH_MAXCALC][VMAX_ARGLEN];
00265 };
00266
00271 typedef struct sNosh Nosh;
00272
00273 /* ////////////////////////////////////////////////////
00274 // Class Nosh: Inlineable methods (mcsh.c)
00275 ////////////////////////////////////////////////////
00276 #if !defined(VINLINE_NOSH)
00284 VEXTERNC char* Nosh_getMolpath(Nosh *thee, int imol);
00285
00293 VEXTERNC char* Nosh_getDielXpath(Nosh *thee, int imap);
00294
00302 VEXTERNC char* Nosh_getDielYpath(Nosh *thee, int imap);
00303
00311 VEXTERNC char* Nosh_getDielZpath(Nosh *thee, int imap);
00312
00320 VEXTERNC char* Nosh_getKappapath(Nosh *thee, int imap);
00321
00329 VEXTERNC char* Nosh_getPotpath(Nosh *thee, int imap);
00330
00338 VEXTERNC char* Nosh_getChargepath(Nosh *thee, int imap);
00339
00347 VEXTERNC Nosh_calc* Nosh_getCalc(Nosh *thee, int icalc);
00348
00356 VEXTERNC int Nosh_getDielfmt(Nosh *thee, int imap);
00357
00365 VEXTERNC int Nosh_getKappafmt(Nosh *thee, int imap);
00366
00374 VEXTERNC int Nosh_getPotfmt(Nosh *thee, int imap);
00375
00383 VEXTERNC int Nosh_getChargefmt(Nosh *thee, int imap);
00384
00385 #else
00386
00387 # define Nosh_getMolpath(thee, imol) ((thee)->molpath[(imol)])
00388 # define Nosh_getDielXpath(thee, imol) ((thee)->dielXpath[(imol)])
00389 # define Nosh_getDielYpath(thee, imol) ((thee)->dielYpath[(imol)])
00390 # define Nosh_getDielZpath(thee, imol) ((thee)->dielZpath[(imol)])
00391 # define Nosh_getKappapath(thee, imol) ((thee)->kappapath[(imol)])
00392 # define Nosh_getPotpath(thee, imol) ((thee)->potpath[(imol)])

```



```

00393 #   define NOsh_getChargepath(thee, imol) ((thee)->chargepath[(imol)])
00394 #   define NOsh_getCalc(thee, icalc) ((thee)->calc[(icalc)])
00395 #   define NOsh_getDielfmt(thee, imap) ((thee)->dielfmt[(imap)])
00396 #   define NOsh_getKappafmt(thee, imap) ((thee)->kappafmt[(imap)])
00397 #   define NOsh_getPotfmt(thee, imap) ((thee)->potfmt[(imap)])
00398 #   define NOsh_getChargefmt(thee, imap) ((thee)->chargefmt[(imap)])
00399
00400 #endif
00401
00402
00403 /* ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00404 // Class NOsh: Non-inlineable methods (mcsh.c)
00405
00406
00414 VEXTERNC NOsh_PrintType NOsh_printWhat(NOsh *thee, int iprint);
00415
00425 VEXTERNC char* NOsh_elecname(NOsh *thee, int ielec);
00426
00434 VEXTERNC int NOsh_elec2calc(NOsh *thee, int icalc);
00435
00443 VEXTERNC int NOsh_apol2calc(NOsh *thee, int icalc);
00444
00452 VEXTERNC int NOsh_printNarg(NOsh *thee, int iprint);
00453
00462 VEXTERNC int NOsh_printOp(NOsh *thee, int iprint, int iarg);
00463
00474 VEXTERNC int NOsh_printCalc(NOsh *thee, int iprint, int iarg);
00475
00485 VEXTERNC NOsh* NOsh_ctor(int rank, int size);
00486
00493 VEXTERNC NOsh_calc* NOsh_calc_ctor(
00494                                     NOsh_CalcType calcType
00495                                     );
00496
00503 VEXTERNC int NOsh_calc_copy(
00504                                     NOsh_calc *thee,
00505                                     NOsh_calc *source
00506                                     );
00507
00513 VEXTERNC void NOsh_calc_dtor(NOsh_calc **thee);
00514
00525 VEXTERNC int NOsh_ctor2(NOsh *thee, int rank, int size);
00526
00532 VEXTERNC void NOsh_dtor(NOsh **thee);
00533
00539 VEXTERNC void NOsh_dtor2(NOsh *thee);
00540
00549 VEXTERNC int NOsh_parseInput(NOsh *thee, Vio *sock);
00550
00560 VEXTERNC int NOsh_parseInputFile(NOsh *thee, char *filename);
00561
00571 VEXTERNC int NOsh_setupElecCalc(
00572                                     NOsh *thee,
00573                                     Valist *alist[NOSH_MAXMOL]
00574                                     );
00575
00585 VEXTERNC int NOsh_setupApolCalc(
00586                                     NOsh *thee,
00587                                     Valist *alist[NOSH_MAXMOL]
00588                                     );
00589
00590 #endif
00591

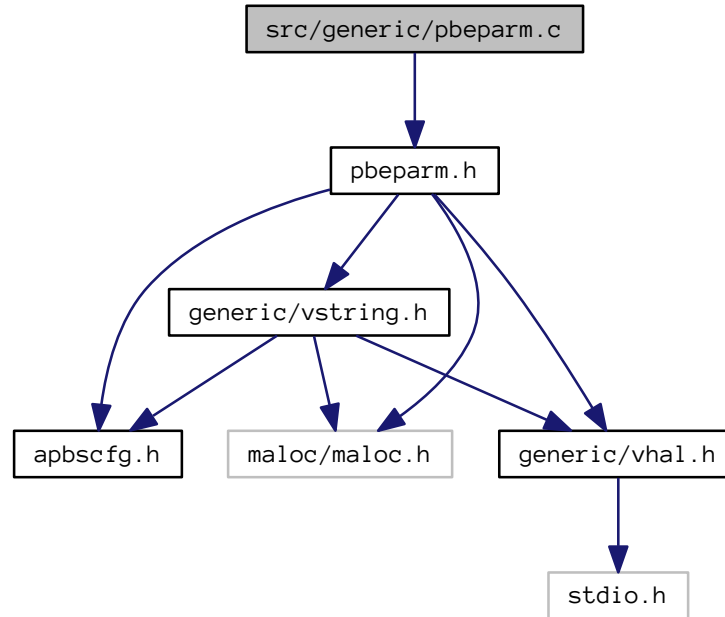
```

10.35 src/generic/pbeparm.c File Reference

Class PBEparm methods.

```
#include "pbeparm.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)
FORTTRAN stub to construct PBEparm object.
- VPUBLIC void [PBEparm_dtor](#) (PBEparm **thee)
Object destructor.
- VPUBLIC void [PBEparm_dtor2](#) (PBEparm *thee)
FORTTRAN 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.

10.35.1 Detailed Description

Class PBEparm methods.

Author

Nathan Baker

Version

\$Id\$

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [pbeparm.c](#).

10.36 pbeparm.c

```

00001
00057 #include "pbeparm.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_MGPARM)
00062
00063 #endif /* if !defined(VINLINE_MGPARM) */
00064
00065 VPUBLIC double PBEparm_getIonCharge(PBEparm *thee, int i) {
00066     VASSERT(thee != VNULL);
00067     VASSERT(i < thee->nion);
00068     return thee->ionq[i];
00069 }

```

```

00070
00071 VPUBLIC double PBEparm_getIonConc(PBEparm *thee, int i) {
00072     VASSERT(thee != VNULL);
00073     VASSERT(i < thee->nion);
00074     return thee->ionc[i];
00075 }
00076
00077 VPUBLIC double PBEparm_getIonRadius(PBEparm *thee, int i) {
00078     VASSERT(thee != VNULL);
00079     VASSERT(i < thee->nion);
00080     return thee->ionr[i];
00081 }
00082
00083 /*-----*/
00084 /* Added by Michael Grabe */
00085 /*-----*/
00086
00087 VPUBLIC double PBEparm_getzmem(PBEparm *thee) {
00088     VASSERT(thee != VNULL);
00089     return thee->zmem;
00090 }
00091 VPUBLIC double PBEparm_getLmem(PBEparm *thee) {
00092     VASSERT(thee != VNULL);
00093     return thee->Lmem;
00094 }
00095 VPUBLIC double PBEparm_getmembraneDiel(PBEparm *thee) {
00096     VASSERT(thee != VNULL);
00097     return thee->mdie;
00098 }
00099 VPUBLIC double PBEparm_getmemv(PBEparm *thee) {
00100     VASSERT(thee != VNULL);
00101     return thee->memv;
00102 }
00103
00104 VPUBLIC PBEparm* PBEparm_ctor() {
00105
00106     /* Set up the structure */
00107     PBEparm *thee = VNULL;
00108     thee = (PBEparm*)Vmem_malloc(VNULL, 1, sizeof(PBEparm));
00109     VASSERT( thee != VNULL);
00110     VASSERT( PBEparm_ctor2(thee) );
00111
00112     return thee;
00113 }
00114
00115 VPUBLIC int PBEparm_ctor2(PBEparm *thee) {
00116
00117     int i;
00118
00119     if (thee == VNULL) return 0;
00120
00121     thee->parsed = 0;
00122
00123     thee->setmolid = 0;
00124     thee->setpbetype = 0;
00125     thee->setbcfl = 0;
00126     thee->setnion = 0;
00127     for (i=0; i<MAXION; i++){
00128         thee->setion[i] = 0;
00129         thee->ionq[i] = 0.0;
00130         thee->ionc[i] = 0.0;
00131         thee->ionr[i] = 0.0;
00132     }
00133     thee->setpdie = 0;
00134     thee->setsdie = 0;
00135     thee->setsrfm = 0;
00136     thee->setsrad = 0;
00137     thee->setswin = 0;
00138     thee->settemp = 0;
00139     thee->setcalcenergy = 0;
00140     thee->setcalcforce = 0;
00141     thee->setsdens = 0;
00142     thee->numwrite = 0;
00143     thee->setwritemat = 0;
00144     thee->nion = 0;
00145     thee->sdens = 0;
00146     thee->swin = 0;
00147     thee->srad = 1.4;
00148     thee->useDielMap = 0;
00149     thee->useKappaMap = 0;
00150     thee->usePotMap = 0;

```

```

00151     thee->useChargeMap = 0;
00152
00153     /*-----*/
00154     /* Added by Michael Grabe */
00155     /*-----*/
00156
00157     thee->setzmem = 0;
00158     thee->setLmem = 0;
00159     thee->setmdie = 0;
00160     thee->setmemv = 0;
00161
00162     /*-----*/
00163
00164     thee->smsize = 0.0;
00165     thee->smvolume = 0.0;
00166
00167     thee->setsmsize = 0;
00168     thee->setsmvolume = 0;
00169
00170     return 1;
00171 }
00172
00173 VPUBLIC void PBEparm_dtor(PBEparm **thee) {
00174     if ((*thee) != VNULL) {
00175         PBEparm_dtor2(*thee);
00176         Vmem_free(VNULL, 1, sizeof(PBEparm), (void **)thee);
00177         (*thee) = VNULL;
00178     }
00179 }
00180
00181 VPUBLIC void PBEparm_dtor2(PBEparm *thee) { ; }
00182
00183 VPUBLIC int PBEparm_check(PBEparm *thee) {
00184     int i;
00185
00186     /* Check to see if we were even filled... */
00187     if (!thee->parsed) {
00188         Vnm_print(2, "PBEparm_check: not filled!\n");
00189         return 0;
00190     }
00191
00192     if (!thee->setmolid) {
00193         Vnm_print(2, "PBEparm_check: MOL not set!\n");
00194         return 0;
00195     }
00196
00197     if (!thee->setpbetype) {
00198         Vnm_print(2, "PBEparm_check: LPBE/NPBE/LRPBE/NRPBE/SMPBE not set!\n");
00199         return 0;
00200     }
00201     if (!thee->setbcfl) {
00202         Vnm_print(2, "PBEparm_check: BCFL not set!\n");
00203         return 0;
00204     }
00205     if (!thee->setnion) {
00206         thee->setnion = 1;
00207         thee->nion = 0;
00208     }
00209     for (i=0; i<thee->nion; i++) {
00210         if (!thee->setion[i]) {
00211             Vnm_print(2, "PBEparm_check: ION #%d not set!\n",i);
00212             return 0;
00213         }
00214     }
00215     if (!thee->setpdie) {
00216         Vnm_print(2, "PBEparm_check: PDIE not set!\n");
00217         return 0;
00218     }
00219     if (((thee->srfm==VSM_MOL) || (thee->srfm==VSM_MOLSMOOTH)) \
00220         && (!thee->setsdens) && (thee->srad > VSMALL)) {
00221         Vnm_print(2, "PBEparm_check: SDENS not set!\n");
00222         return 0;
00223     }
00224     if (!thee->setsdie) {
00225         Vnm_print(2, "PBEparm_check: SDIE not set!\n");
00226         return 0;
00227     }
00228     if (!thee->setsrfm) {
00229         Vnm_print(2, "PBEparm_check: SRFM not set!\n");
00230         return 0;
00231     }

```

```

00232     if ((thee->srfm==VSM_MOL) || (thee->srfm==VSM_MOLSMOOTH)) \
00233         && (!thee->setsrad)) {
00234         Vnm_print(2, "PBeparm_check: SRAD not set!\n");
00235         return 0;
00236     }
00237     if ((thee->srfm==VSM_SPLINE) && (!thee->setswin)) {
00238         Vnm_print(2, "PBeparm_check: SWIN not set!\n");
00239         return 0;
00240     }
00241     if ((thee->srfm==VSM_SPLINE3) && (!thee->setswin)) {
00242         Vnm_print(2, "PBeparm_check: SWIN not set!\n");
00243         return 0;
00244     }
00245     if ((thee->srfm==VSM_SPLINE4) && (!thee->setswin)) {
00246         Vnm_print(2, "PBeparm_check: SWIN not set!\n");
00247         return 0;
00248     }
00249     if (!thee->settemp) {
00250         Vnm_print(2, "PBeparm_check: TEMP not set!\n");
00251         return 0;
00252     }
00253     if (!thee->setcalcenergy) thee->calcenergy = PCE_NO;
00254     if (!thee->setcalcforce) thee->calcforce = PCF_NO;
00255     if (!thee->setwritemat) thee->writemat = 0;
00256
00257     /*-----*/
00258     /* Added by Michael Grabe */
00259     /*-----*/
00260
00261     if (!thee->setzmem) && (thee->bcfl == 3){
00262         Vnm_print(2, "PBeparm_check: ZMEM not set!\n");
00263         return 0;
00264     }
00265     if (!thee->setlmem) && (thee->bcfl == 3){
00266         Vnm_print(2, "PBeparm_check: LMEM not set!\n");
00267         return 0;
00268     }
00269     if (!thee->setmdie) && (thee->bcfl == 3){
00270         Vnm_print(2, "PBeparm_check: MDIE not set!\n");
00271         return 0;
00272     }
00273     if (!thee->setmemv) && (thee->bcfl == 3){
00274         Vnm_print(2, "PBeparm_check: MEMV not set!\n");
00275         return 0;
00276     }
00277
00278     /*-----*/
00279
00280     return 1;
00281 }
00282
00283 VPUBLIC void PBeparm_copy(PBeparm *thee, PBeparm *parm) {
00284
00285     int i, j;
00286
00287     VASSERT(thee != VNULL);
00288     VASSERT(parm != VNULL);
00289
00290     thee->molid = parm->molid;
00291     thee->setmolid = parm->setmolid;
00292     thee->useDielMap = parm->useDielMap;
00293     thee->dielMapID = parm->dielMapID;
00294     thee->useKappaMap = parm->useKappaMap;
00295     thee->kappaMapID = parm->kappaMapID;
00296     thee->usePotMap = parm->usePotMap;
00297     thee->potMapID = parm->potMapID;
00298     thee->useChargeMap = parm->useChargeMap;
00299     thee->chargeMapID = parm->chargeMapID;
00300     thee->pbetype = parm->pbetype;
00301     thee->setpbetype = parm->setpbetype;
00302     thee->bcfl = parm->bcfl;
00303     thee->setbcfl = parm->setbcfl;
00304     thee->nion = parm->nion;
00305     thee->setnion = parm->setnion;
00306     for (i=0; i<MAXION; i++) {
00307         thee->ionq[i] = parm->ionq[i];
00308         thee->ionc[i] = parm->ionc[i];
00309         thee->ionr[i] = parm->ionr[i];
00310         thee->setion[i] = parm->setion[i];
00311     };
00312     thee->pdie = parm->pdie;

```

```

00313     thee->setpdie = parm->setpdie;
00314     thee->sdens = parm->sdens;
00315     thee->setsdens = parm->setsdens;
00316     thee->sdie = parm->sdie;
00317     thee->setsdie = parm->setsdie;
00318     thee->srfm = parm->srfm;
00319     thee->setsrfm = parm->setsrfm;
00320     thee->srad = parm->srad;
00321     thee->setsrad = parm->setsrad;
00322     thee->swin = parm->swin;
00323     thee->setswin = parm->setswin;
00324     thee->temp = parm->temp;
00325     thee->settemp = parm->settemp;
00326     thee->calcenergy = parm->calcenergy;
00327     thee->setcalcenergy = parm->setcalcenergy;
00328     thee->calcforce = parm->calcforce;
00329     thee->setcalcforce = parm->setcalcforce;
00330
00331     /*-----*/
00332     /* Added by Michael Grabe */
00333     /*-----*/
00334
00335     thee->zmem = parm->zmem;
00336     thee->setzmem = parm->setzmem;
00337     thee->Lmem = parm->Lmem;
00338     thee->setLmem = parm->setLmem;
00339     thee->mdie = parm->mdie;
00340     thee->setmdie = parm->setmdie;
00341     thee->memv = parm->memv;
00342     thee->setmemv = parm->setmemv;
00343
00344     /*-----*/
00345
00346     thee->numwrite = parm->numwrite;
00347     for (i=0; i<PBEPARM_MAXWRITE; i++) {
00348         thee->writetype[i] = parm->writetype[i];
00349         thee->writefmt[i] = parm->writefmt[i];
00350         for (j=0; j<VMAX_ARGLEN; j++)
00351             thee->writestem[i][j] = parm->writestem[i][j];
00352     }
00353     thee->writemat = parm->writemat;
00354     thee->setwritemat = parm->setwritemat;
00355     for (i=0; i<VMAX_ARGLEN; i++) thee->writematstem[i] = parm->
writematstem[i];
00356     thee->writematflag = parm->writematflag;
00357
00358     thee->smsize = parm->smsize;
00359     thee->smvolume = parm->smvolume;
00360
00361     thee->setsmsize = parm->setsmsize;
00362     thee->setsmvolume = parm->setsmvolume;
00363
00364     thee->parsed = parm->parsed;
00365
00366 }
00367
00368 VPRIVATE int PBEparm_parseLPBE(PBEparm *thee, Vio *sock) {
00369     Vnm_print(0, "Nosh: parsed lpbe\n");
00370     thee->pbetype = PBE_LPBE;
00371     thee->setpbetype = 1;
00372     return 1;
00373 }
00374
00375 VPRIVATE int PBEparm_parseNPBE(PBEparm *thee, Vio *sock) {
00376     Vnm_print(0, "Nosh: parsed npbe\n");
00377     thee->pbetype = PBE_NPBE;
00378     thee->setpbetype = 1;
00379     return 1;
00380 }
00381
00382 VPRIVATE int PBEparm_parseMOL(PBEparm *thee, Vio *sock) {
00383     int ti;
00384     char tok[VMAX_BUFSIZE];
00385
00386     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00387     if (sscanf(tok, "%d", &ti) == 0) {
00388         Vnm_print(2, "Nosh: Read non-int (%s) while parsing MOL \
00389 keyword!\n", tok);
00390         return -1;
00391     }
00392     thee->molid = ti;

```



```

00393     thee->setmolid = 1;
00394     return 1;
00395
00396     ERROR1:
00397         Vnm_print(2, "parsePBE: ran out of tokens!\n");
00398         return -1;
00399 }
00400
00401 VPRIVATE int PBEparm_parseLRPBE(PBEparm *thee, Vio *sock) {
00402     Vnm_print(0, "Nosh: parsed lrpbe\n");
00403     thee->pbetype = PBE_LRPBE;
00404     thee->setpbetype = 1;
00405     return 1;
00406 }
00407
00408 VPRIVATE int PBEparm_parseNRPBE(PBEparm *thee, Vio *sock) {
00409     Vnm_print(0, "Nosh: parsed nrpbe\n");
00410     thee->pbetype = PBE_NRPBE;
00411     thee->setpbetype = 1;
00412     return 1;
00413 }
00414
00415 VPRIVATE int PBEparm_parseSMPBE(PBEparm *thee, Vio *sock) {
00416
00417     int i;
00418
00419     char type[VMAX_BUFSIZE]; /* vol or size (keywords) */
00420     char value[VMAX_BUFSIZE]; /* floating point value */
00421
00422     char setVol = 1;
00423     char setSize = 1;
00424     char keyValuePairs = 2;
00425
00426     double size, volume;
00427
00428     for(i=0;i<keyValuePairs;i++){
00429
00430         /* The line two tokens at a time */
00431         VJMPERR1(Vio_scanf(sock, "%s", type) == 1);
00432         VJMPERR1(Vio_scanf(sock, "%s", value) == 1);
00433
00434         if(!strcmp(type,"vol")){
00435             if ((setVol = sscanf(value, "%lf", &volume)) == 0){
00436                 Vnm_print(2,"Nosh: Read non-float (%s) while parsing smpbe keyword!\n", value);
00437                 return VRC_FAILURE;
00438             }
00439         }else if(!strcmp(type,"size")){
00440             if ((setSize = sscanf(value, "%lf", &size)) == 0){
00441                 Vnm_print(2,"Nosh: Read non-float (%s) while parsing smpbe keyword!\n", value);
00442                 return VRC_FAILURE;
00443             }
00444         }else{
00445             Vnm_print(2,"Nosh: Read non-float (%s) while parsing smpbe keyword!\n", value);
00446             return VRC_FAILURE;
00447         }
00448     }
00449
00450     /* If either the volume or size isn't set, throw an error */
00451     if((setVol == 0) || (setSize == 0)){
00452         Vnm_print(2,"Nosh: Error while parsing smpbe keywords! Only size or vol was specified.\n");
00453         return VRC_FAILURE;
00454     }
00455
00456     Vnm_print(0, "Nosh: parsed smpbe\n");
00457     thee->pbetype = PBE_SMPBE;
00458     thee->setpbetype = 1;
00459
00460     thee->smsize = size;
00461     thee->setsmsize = 1;
00462
00463     thee->smvolume = volume;
00464     thee->setsmvolume = 1;
00465
00466     return VRC_SUCCESS;
00467
00468     ERROR1:
00469     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00470     return VRC_FAILURE;
00471 }
00472 }
00473

```

```

00474 VPRIVATE int PBEparm_parseBCFL(PBEparm *thee, Vio *sock) {
00475     char tok[VMAX_BUFSIZE];
00476     int ti;
00477
00478     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00479
00480     /* We can either parse int flag... */
00481     if (sscanf(tok, "%d", &ti) == 1) {
00482
00483         thee->bcfl = (Vbcfl)ti;
00484         thee->setbcfl = 1;
00485         /* Warn that this usage is deprecated */
00486         Vnm_print(2, "parsePBE: Warning -- parsed deprecated \"bcfl %d\" \"
00487 statement\n", ti);
00488         Vnm_print(2, "parsePBE: Please use \"bcfl ");
00489         switch (thee->bcfl) {
00490             case BCFL_ZERO:
00491                 Vnm_print(2, "zero");
00492                 break;
00493             case BCFL_SDH:
00494                 Vnm_print(2, "sdh");
00495                 break;
00496             case BCFL_MDH:
00497                 Vnm_print(2, "mdh");
00498                 break;
00499             case BCFL_FOCUS:
00500                 Vnm_print(2, "focus");
00501                 break;
00502             case BCFL_MEM:
00503                 Vnm_print(2, "mem");
00504                 break;
00505             case BCFL_MAP:
00506                 Vnm_print(2, "map");
00507                 break;
00508             default:
00509                 Vnm_print(2, "UNKNOWN");
00510                 break;
00511         }
00512         Vnm_print(2, "\" instead.\n");
00513         return 1;
00514
00515         /* ...or the word */
00516     } else {
00517
00518         if (Vstring_strcasecmp(tok, "zero") == 0) {
00519             thee->bcfl = BCFL_ZERO;
00520             thee->setbcfl = 1;
00521             return 1;
00522         } else if (Vstring_strcasecmp(tok, "sdh") == 0) {
00523             thee->bcfl = BCFL_SDH;
00524             thee->setbcfl = 1;
00525             return 1;
00526         } else if (Vstring_strcasecmp(tok, "mdh") == 0) {
00527             thee->bcfl = BCFL_MDH;
00528             thee->setbcfl = 1;
00529             return 1;
00530         } else if (Vstring_strcasecmp(tok, "focus") == 0) {
00531             thee->bcfl = BCFL_FOCUS;
00532             thee->setbcfl = 1;
00533             return 1;
00534         } else if (Vstring_strcasecmp(tok, "mem") == 0) {
00535             thee->bcfl = BCFL_MEM;
00536             thee->setbcfl = 1;
00537             return 1;
00538         } else if (Vstring_strcasecmp(tok, "map") == 0) {
00539             thee->bcfl = BCFL_MAP;
00540             thee->setbcfl = 1;
00541             return 1;
00542         } else {
00543             Vnm_print(2, "Nosh: parsed unknown BCFL parameter (%s)!\n",
00544 tok);
00545             return -1;
00546         }
00547     }
00548     return 0;
00549
00550     VERR01:
00551     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00552     return -1;
00553 }
00554

```

```

00555 VPRIVATE int PBEParm_parseION(PBEParm *thee, Vio *sock) {
00556
00557     int i;
00558     int meth = 0;
00559
00560     char tok[VMAX_BUFSIZE];
00561     char value[VMAX_BUFSIZE];
00562
00563     double tf;
00564     double charge, conc, radius;
00565
00566     int setCharge = 0;
00567     int setConc = 0;
00568     int setRadius = 0;
00569     int keyValuePairs = 3;
00570
00571     /* Get the initial token for the ION statement */
00572     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00573
00574     /* Scan the token once to determine the type (old style or new key-value pair) */
00575     meth = sscanf(tok, "%lf", &tf);
00576     /* If tok is a non-zero float value, we are using the old method */
00577     if(meth != 0){
00578
00579         Vnm_print(2, "NOsh: Deprecated use of ION keyword! Use key-value pairs\n", tok);
00580
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->ionq[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->ionc[thee->nion] = tf;
00592         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00593         if (sscanf(tok, "%lf", &tf) == 0) {
00594             Vnm_print(2, "NOsh: Read non-float (%s) while parsing ION keyword!\n", tok);
00595             return VRC_FAILURE;
00596         }
00597         thee->ionr[thee->nion] = tf;
00598     }else{
00599
00600         /* Three key-value pairs (charge, radius and conc) */
00601         for(i=0;i<keyValuePairs;i++){
00602
00603             /* Now scan for the value (float) to be used with the key token parsed
00604              * above the if-else statement */
00605             VJMPERR1(Vio_scanf(sock, "%s", value) == 1);
00606             if(!strcmp(tok,"charge")){
00607                 setCharge = sscanf(value, "%lf", &charge);
00608                 if (setCharge == 0){
00609                     Vnm_print(2,"NOsh: Read non-float (%s) while parsing ION %s keyword!\n", value, tok);
00610                     return VRC_FAILURE;
00611                 }
00612                 thee->ionq[thee->nion] = charge;
00613             }else if(!strcmp(tok,"radius")){
00614                 setRadius = sscanf(value, "%lf", &radius);
00615                 if (setRadius == 0){
00616                     Vnm_print(2,"NOsh: Read non-float (%s) while parsing ION %s keyword!\n", value, tok);
00617                     return VRC_FAILURE;
00618                 }
00619                 thee->ionr[thee->nion] = radius;
00620             }else if(!strcmp(tok,"conc")){
00621                 setConc = sscanf(value, "%lf", &conc);
00622                 if (setConc == 0){
00623                     Vnm_print(2,"NOsh: Read non-float (%s) while parsing ION %s keyword!\n", value, tok);
00624                     return VRC_FAILURE;
00625                 }
00626                 thee->ionc[thee->nion] = conc;
00627             }else{
00628                 Vnm_print(2,"NOsh: Illegal or missing key-value pair for ION keyword!\n");
00629                 return VRC_FAILURE;
00630             }
00631         }
00632
00633         /* If all three values haven't be set (setValue = 0) then read the next token */
00634         if((setCharge != 1) || (setConc != 1) || (setRadius != 1)){
00635             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);

```

```

00636         }
00637
00638     } /* end for */
00639 } /* end if */
00640
00641 /* Finally set the setion, nion and setnion flags and return success */
00642 thee->setion[thee->nion] = 1;
00643 (thee->nion)++;
00644 thee->setnion = 1;
00645 return VRC_SUCCESS;
00646
00647 ERROR1:
00648     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00649     return VRC_FAILURE;
00650 }
00651
00652 VPRIVATE int PBEparm_parsePDIE(PBEparm *thee, Vio *sock) {
00653     char tok[VMAX_BUFSIZE];
00654     double tf;
00655
00656     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00657     if (sscanf(tok, "%lf", &tf) == 0) {
00658         Vnm_print(2, "Nosh: Read non-float (%s) while parsing PDIE \
00659 keyword!\n", tok);
00660         return -1;
00661     }
00662     thee->pdie = tf;
00663     thee->setpdie = 1;
00664     return 1;
00665
00666     ERROR1:
00667         Vnm_print(2, "parsePBE: ran out of tokens!\n");
00668         return -1;
00669 }
00670
00671 VPRIVATE int PBEparm_parseSDENS(PBEparm *thee, Vio *sock) {
00672     char tok[VMAX_BUFSIZE];
00673     double tf;
00674
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 SDENS \
00678 keyword!\n", tok);
00679         return -1;
00680     }
00681     thee->sdens = tf;
00682     thee->setsdens = 1;
00683     return 1;
00684
00685     ERROR1:
00686         Vnm_print(2, "parsePBE: ran out of tokens!\n");
00687         return -1;
00688 }
00689
00690 VPRIVATE int PBEparm_parseSDIE(PBEparm *thee, Vio *sock) {
00691     char tok[VMAX_BUFSIZE];
00692     double tf;
00693
00694     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00695     if (sscanf(tok, "%lf", &tf) == 0) {
00696         Vnm_print(2, "Nosh: Read non-float (%s) while parsing SDIE \
00697 keyword!\n", tok);
00698         return -1;
00699     }
00700     thee->sdie = tf;
00701     thee->setsdie = 1;
00702     return 1;
00703
00704     ERROR1:
00705         Vnm_print(2, "parsePBE: ran out of tokens!\n");
00706         return -1;
00707 }
00708
00709 VPRIVATE int PBEparm_parseSRFM(PBEparm *thee, Vio *sock) {
00710     char tok[VMAX_BUFSIZE];
00711     int ti;
00712
00713     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00714
00715     /* Parse old-style int arg */
00716     if (sscanf(tok, "%d", &ti) == 1) {

```

```

00717     thee->srfm = (Vsurf_Meth)ti;
00718     thee->setsrfm = 1;
00719
00720     Vnm_print(2, "parsePBE: Warning -- parsed deprecated \"srfm %d\" \"\
00721 statement.\n", ti);
00722     Vnm_print(2, "parsePBE: Please use \"srfm \");
00723     switch (thee->srfm) {
00724         case VSM_MOL:
00725             Vnm_print(2, "mol");
00726             break;
00727         case VSM_MOLSMOOTH:
00728             Vnm_print(2, "smol");
00729             break;
00730         case VSM_SPLINE:
00731             Vnm_print(2, "spl2");
00732             break;
00733         case VSM_SPLINE3:
00734             Vnm_print(2, "spl3");
00735             break;
00736         case VSM_SPLINE4:
00737             Vnm_print(2, "spl4");
00738             break;
00739         default:
00740             Vnm_print(2, "UNKNOWN");
00741             break;
00742     }
00743     Vnm_print(2, "\" instead.\n");
00744     return 1;
00745
00746     /* Parse newer text-based args */
00747     } else if (Vstring_strcasecmp(tok, "mol") == 0) {
00748         thee->srfm = VSM_MOL;
00749         thee->setsrfm = 1;
00750         return 1;
00751     } else if (Vstring_strcasecmp(tok, "smol") == 0) {
00752         thee->srfm = VSM_MOLSMOOTH;
00753         thee->setsrfm = 1;
00754         return 1;
00755     } else if (Vstring_strcasecmp(tok, "spl2") == 0) {
00756         thee->srfm = VSM_SPLINE;
00757         thee->setsrfm = 1;
00758         return 1;
00759     } else if (Vstring_strcasecmp(tok, "spl3") == 0) {
00760         thee->srfm = VSM_SPLINE3;
00761         thee->setsrfm = 1;
00762         return 1;
00763     } else if (Vstring_strcasecmp(tok, "spl4") == 0) {
00764         thee->srfm = VSM_SPLINE4;
00765         thee->setsrfm = 1;
00766         return 1;
00767     } else {
00768         Vnm_print(2, "Nosh: Unrecognized keyword (%s) when parsing \"
00769 srfm!\n", tok);
00770         return -1;
00771     }
00772
00773     return 0;
00774
00775     ERROR1:
00776     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00777     return -1;
00778 }
00779
00780 VPRIVATE int PBEparm_parseSRAD(PBEparm *thee, Vio *sock) {
00781     char tok[VMAX_BUFSIZE];
00782     double tf;
00783
00784     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00785     if (sscanf(tok, "%lf", &tf) == 0) {
00786         Vnm_print(2, "Nosh: Read non-float (%s) while parsing SRAD \"
00787 keyword!\n", tok);
00788         return -1;
00789     }
00790     thee->srاد = tf;
00791     thee->setsrad = 1;
00792     return 1;
00793
00794     ERROR1:
00795     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00796     return -1;
00797 }

```

```

00798
00799 VPRIVATE int PBEparm_parseSWIN(PBEparm *thee, Vio *sock) {
00800     char tok[VMAX_BUFSIZE];
00801     double tf;
00802
00803     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00804     if (sscanf(tok, "%lf", &tf) == 0) {
00805         Vnm_print(2, "Nosh: Read non-float (%s) while parsing SWIN \
00806 keyword!\n", tok);
00807         return -1;
00808     }
00809     thee->swin = tf;
00810     thee->setswin = 1;
00811     return 1;
00812
00813     ERROR1:
00814     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00815     return -1;
00816 }
00817
00818 VPRIVATE int PBEparm_parseTEMP(PBEparm *thee, Vio *sock) {
00819     char tok[VMAX_BUFSIZE];
00820     double tf;
00821
00822     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00823     if (sscanf(tok, "%lf", &tf) == 0) {
00824         Vnm_print(2, "Nosh: Read non-float (%s) while parsing TEMP \
00825 keyword!\n", tok);
00826         return -1;
00827     }
00828     thee->temp = tf;
00829     thee->settemp = 1;
00830     return 1;
00831
00832     ERROR1:
00833     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00834     return -1;
00835 }
00836
00837 VPRIVATE int PBEparm_parseUSEMAP(PBEparm *thee, Vio *sock) {
00838     char tok[VMAX_BUFSIZE];
00839     int ti;
00840
00841     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00842     Vnm_print(0, "PBEparm_parseToken: Read %s...\n", tok);
00843     if (Vstring_strcasecmp(tok, "diel") == 0) {
00844         thee->useDielMap = 1;
00845         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00846         if (sscanf(tok, "%d", &ti) == 0) {
00847             Vnm_print(2, "Nosh: Read non-int (%s) while parsing \
00848 USEMAP DIELE keyword!\n", tok);
00849             return -1;
00850         }
00851         thee->dielMapID = ti;
00852         return 1;
00853     } else if (Vstring_strcasecmp(tok, "kappa") == 0) {
00854         thee->useKappaMap = 1;
00855         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00856         if (sscanf(tok, "%d", &ti) == 0) {
00857             Vnm_print(2, "Nosh: Read non-int (%s) while parsing \
00858 USEMAP KAPPA keyword!\n", tok);
00859             return -1;
00860         }
00861         thee->kappaMapID = ti;
00862         return 1;
00863     } else if (Vstring_strcasecmp(tok, "pot") == 0) {
00864         thee->usePotMap = 1;
00865         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00866         if (sscanf(tok, "%d", &ti) == 0) {
00867             Vnm_print(2, "Nosh: Read non-int (%s) while parsing \
00868 USEMAP POT keyword!\n", tok);
00869             return -1;
00870         }
00871         thee->potMapID = ti;
00872         return 1;
00873     } else if (Vstring_strcasecmp(tok, "charge") == 0) {
00874         thee->useChargeMap = 1;
00875         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00876         if (sscanf(tok, "%d", &ti) == 0) {
00877             Vnm_print(2, "Nosh: Read non-int (%s) while parsing \
00878 USEMAP CHARGE keyword!\n", tok);

```

```

00879         return -1;
00880     }
00881     thee->chargeMapID = ti;
00882     return 1;
00883 } else {
00884     Vnm_print(2, "Nosh: Read undefined keyword (%s) while parsing \
00885 USEMAP statement!\n", tok);
00886     return -1;
00887 }
00888 return 0;
00889
00890 ERROR1:
00891     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00892     return -1;
00893 }
00894
00895 VPRIVATE int PBEparm_parseCALCENERGY(PBEparm *thee, Vio *sock) {
00896     char tok[VMAX_BUFSIZE];
00897     int ti;
00898
00899     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00900     /* Parse number */
00901     if (sscanf(tok, "%d", &ti) == 1) {
00902         thee->calcenergy = (PBEparm_calcEnergy)ti;
00903         thee->setcalcenergy = 1;
00904
00905         Vnm_print(2, "parsePBE: Warning -- parsed deprecated \"calcenergy \
00906 %d\" statement.\n", ti);
00907         Vnm_print(2, "parsePBE: Please use \"calcenergy ");
00908         switch (thee->calcenergy) {
00909             case PCE_NO:
00910                 Vnm_print(2, "no");
00911                 break;
00912             case PCE_TOTAL:
00913                 Vnm_print(2, "total");
00914                 break;
00915             case PCE_COMPS:
00916                 Vnm_print(2, "comps");
00917                 break;
00918             default:
00919                 Vnm_print(2, "UNKNOWN");
00920                 break;
00921         }
00922         Vnm_print(2, "\" instead.\n");
00923         return 1;
00924     } else if (Vstring_strcasecmp(tok, "no") == 0) {
00925         thee->calcenergy = PCE_NO;
00926         thee->setcalcenergy = 1;
00927         return 1;
00928     } else if (Vstring_strcasecmp(tok, "total") == 0) {
00929         thee->calcenergy = PCE_TOTAL;
00930         thee->setcalcenergy = 1;
00931         return 1;
00932     } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00933         thee->calcenergy = PCE_COMPS;
00934         thee->setcalcenergy = 1;
00935         return 1;
00936     } else {
00937         Vnm_print(2, "Nosh: Unrecognized parameter (%s) while parsing \
00938 calcenergy!\n", tok);
00939         return -1;
00940     }
00941     return 0;
00942
00943 ERROR1:
00944     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00945     return -1;
00946 }
00947
00948 VPRIVATE int PBEparm_parseCALCFORCE(PBEparm *thee, Vio *sock) {
00949     char tok[VMAX_BUFSIZE];
00950     int ti;
00951
00952     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00953     /* Parse number */
00954     if (sscanf(tok, "%d", &ti) == 1) {
00955         thee->calcforce = (PBEparm_calcForce)ti;
00956         thee->setcalcforce = 1;
00957
00958         Vnm_print(2, "parsePBE: Warning -- parsed deprecated \"calcforce \
00959 %d\" statement.\n", ti);

```

```

00960     Vnm_print(2, "parsePBE: Please use \"calcforce ");
00961     switch (thee->calcenergy) {
00962         case PCF_NO:
00963             Vnm_print(2, "no");
00964             break;
00965         case PCF_TOTAL:
00966             Vnm_print(2, "total");
00967             break;
00968         case PCF_COMPS:
00969             Vnm_print(2, "comps");
00970             break;
00971         default:
00972             Vnm_print(2, "UNKNOWN");
00973             break;
00974     }
00975     Vnm_print(2, "\" instead.\n");
00976     return 1;
00977 } else if (Vstring_strcasecmp(tok, "no") == 0) {
00978     thee->calcforce = PCF_NO;
00979     thee->setcalcforce = 1;
00980     return 1;
00981 } else if (Vstring_strcasecmp(tok, "total") == 0) {
00982     thee->calcforce = PCF_TOTAL;
00983     thee->setcalcforce = 1;
00984     return 1;
00985 } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00986     thee->calcforce = PCF_COMPS;
00987     thee->setcalcforce = 1;
00988     return 1;
00989 } else {
00990     Vnm_print(2, "Nosh: Unrecognized parameter (%s) while parsing \"
00991 calcforce!\n", tok);
00992     return -1;
00993 }
00994 return 0;
00995
00996 ERROR1:
00997     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00998     return -1;
00999 }
01000
01001 /*-----*/
01002 /* Added by Michael Grabe */
01003 /*-----*/
01004
01005 VPRIVATE int PBEparm_parseZMEM(PBEparm *thee, Vio *sock) {
01006     char tok[VMAX_BUFSIZE];
01007     double tf;
01008
01009     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01010     if (sscanf(tok, "%lf", &tf) == 0) {
01011         Vnm_print(2, "Nosh: Read non-float (%s) while parsing ZMEM \"
01012             keyword!\n", tok);
01013         return -1;
01014     }
01015     thee->zmem = tf;
01016     thee->setzmem = 1;
01017     return 1;
01018
01019 ERROR1:
01020     Vnm_print(2, "parsePBE: ran out of tokens!\n");
01021     return -1;
01022 }
01023
01024
01025 VPRIVATE int PBEparm_parseLMEM(PBEparm *thee, Vio *sock) {
01026     char tok[VMAX_BUFSIZE];
01027     double tf;
01028
01029     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01030     if (sscanf(tok, "%lf", &tf) == 0) {
01031         Vnm_print(2, "Nosh: Read non-float (%s) while parsing LMEM \"
01032             keyword!\n", tok);
01033         return -1;
01034     }
01035     thee->Lmem = tf;
01036     thee->setLmem = 1;
01037     return 1;
01038
01039 ERROR1:
01040     Vnm_print(2, "parsePBE: ran out of tokens!\n");

```



```

01041     return -1;
01042 }
01043
01044 VPRIVATE int PBeparm_parseMDIE(PBeparm *thee, Vio *sock) {
01045     char tok[VMAX_BUFSIZE];
01046     double tf;
01047
01048     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01049     if (sscanf(tok, "%lf", &tf) == 0) {
01050         Vnm_print(2, "Nosh: Read non-float (%s) while parsing MDIE \
01051             keyword!\n", tok);
01052         return -1;
01053     }
01054     thee->mdie = tf;
01055     thee->setmdie = 1;
01056     return 1;
01057
01058 ERROR1:
01059     Vnm_print(2, "parsePBE: ran out of tokens!\n");
01060     return -1;
01061 }
01062
01063 VPRIVATE int PBeparm_parseMEMV(PBeparm *thee, Vio *sock) {
01064     char tok[VMAX_BUFSIZE];
01065     double tf;
01066
01067     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01068     if (sscanf(tok, "%lf", &tf) == 0) {
01069         Vnm_print(2, "Nosh: Read non-float (%s) while parsing MEMV \
01070             keyword!\n", tok);
01071         return -1;
01072     }
01073     thee->memv = tf;
01074     thee->setmemv = 1;
01075     return 1;
01076
01077 ERROR1:
01078     Vnm_print(2, "parsePBE: ran out of tokens!\n");
01079     return -1;
01080 }
01081
01082 /*-----*/
01083
01084 VPRIVATE int PBeparm_parseWRITE(PBeparm *thee, Vio *sock) {
01085     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
01086     Vdata_Type writetype;
01087     Vdata_Format writefmt;
01088
01089     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01090     if (Vstring_strcasecmp(tok, "pot") == 0) {
01091         writetype = VDT_POT;
01092     } else if (Vstring_strcasecmp(tok, "atompot") == 0) {
01093         writetype = VDT_ATOMPOT;
01094     } else if (Vstring_strcasecmp(tok, "charge") == 0) {
01095         writetype = VDT_CHARGE;
01096     } else if (Vstring_strcasecmp(tok, "smol") == 0) {
01097         writetype = VDT_SMOL;
01098     } else if (Vstring_strcasecmp(tok, "dielx") == 0) {
01099         writetype = VDT_DIELX;
01100     } else if (Vstring_strcasecmp(tok, "diely") == 0) {
01101         writetype = VDT_DIELY;
01102     } else if (Vstring_strcasecmp(tok, "dielz") == 0) {
01103         writetype = VDT_DIELZ;
01104     } else if (Vstring_strcasecmp(tok, "kappa") == 0) {
01105         writetype = VDT_KAPPA;
01106     } else if (Vstring_strcasecmp(tok, "sspl") == 0) {
01107         writetype = VDT_SSPL;
01108     } else if (Vstring_strcasecmp(tok, "vdw") == 0) {
01109         writetype = VDT_VDW;
01110     } else if (Vstring_strcasecmp(tok, "ivdw") == 0) {
01111         writetype = VDT_IVDW;
01112     } else if (Vstring_strcasecmp(tok, "lap") == 0) {
01113         writetype = VDT_LAP;
01114     } else if (Vstring_strcasecmp(tok, "edens") == 0) {
01115         writetype = VDT_EDENS;
01116     } else if (Vstring_strcasecmp(tok, "ndens") == 0) {
01117         writetype = VDT_NDENS;
01118     } else if (Vstring_strcasecmp(tok, "qdens") == 0) {
01119         writetype = VDT_QDENS;
01120     } else {
01121         Vnm_print(2, "PBeparm_parse: Invalid data type (%s) to write!\n",

```

```

01122         tok);
01123         return -1;
01124     }
01125     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01126     if (Vstring_strcasecmp(tok, "dx") == 0) {
01127         writefmt = VDF_DX;
01128     } else if (Vstring_strcasecmp(tok, "uhbd") == 0) {
01129         writefmt = VDF_UHBD;
01130     } else if (Vstring_strcasecmp(tok, "avs") == 0) {
01131         writefmt = VDF_AVS;
01132     } else if (Vstring_strcasecmp(tok, "gz") == 0) {
01133         writefmt = VDF_GZ;
01134     } else if (Vstring_strcasecmp(tok, "flat") == 0) {
01135         writefmt = VDF_FLAT;
01136     } else {
01137         Vnm_print(2, "PBEParm_parse: Invalid data format (%s) to write!\n",
01138             tok);
01139         return -1;
01140     }
01141     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01142     if (tok[0]=='"') {
01143         strcpy(strnew, "");
01144         while (tok[strlen(tok)-1] != '"') {
01145             strcat(str, tok);
01146             strcat(str, " ");
01147             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01148         }
01149         strcat(str, tok);
01150         strncpy(strnew, str+1, strlen(str)-2);
01151         strcpy(tok, strnew);
01152     }
01153     if (thee->numwrite < (PBEPARM_MAXWRITE-1)) {
01154         strncpy(thee->writestem[thee->numwrite], tok, VMAX_ARGLEN);
01155         thee->writetype[thee->numwrite] = writetype;
01156         thee->writefmt[thee->numwrite] = writefmt;
01157         (thee->numwrite)++;
01158     } else {
01159         Vnm_print(2, "PBEParm_parse: You have exceeded the maximum number of write statements!\n");
01160         Vnm_print(2, "PBEParm_parse: Ignoring additional write statements!\n");
01161     }
01162     return 1;
01163 }
01164
01165 ERROR1:
01166     Vnm_print(2, "parsePBE: ran out of tokens!\n");
01167     return -1;
01168 }
01169
01170 VPRIVATE int PBEParm_parseWRITEMAT(PBEParm *thee, Vio *sock) {
01171     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
01172     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01173     if (Vstring_strcasecmp(tok, "poisson") == 0) {
01174         thee->writematflag = 0;
01175     } else if (Vstring_strcasecmp(tok, "full") == 0) {
01176         thee->writematflag = 1;
01177     } else {
01178         Vnm_print(2, "Nosh: Invalid format (%s) while parsing \
01179 WRITEMAT keyword!\n", tok);
01180         return -1;
01181     }
01182
01183     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01184     if (tok[0]=='"') {
01185         strcpy(strnew, "");
01186         while (tok[strlen(tok)-1] != '"') {
01187             strcat(str, tok);
01188             strcat(str, " ");
01189             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01190         }
01191         strcat(str, tok);
01192         strncpy(strnew, str+1, strlen(str)-2);
01193         strcpy(tok, strnew);
01194     }
01195     strncpy(thee->writematstem, tok, VMAX_ARGLEN);
01196     thee->setwritemat = 1;
01197     thee->writemat = 1;
01198     return 1;
01199 }
01200
01201 ERROR1:
01202     Vnm_print(2, "parsePBE: ran out of tokens!\n");
01203     return -1;

```

```

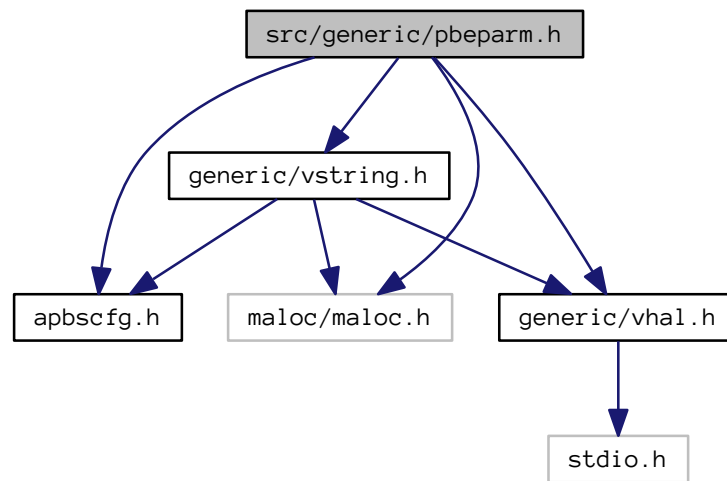
01203
01204 }
01205
01206 VPUBLIC int PBeparm_parseToken(PBeparm *thee, char tok[VMAX_BUFSIZE],
01207     Vio *sock) {
01208
01209     if (thee == VNULL) {
01210         Vnm_print(2, "parsePBE: got NULL thee!\n");
01211         return -1;
01212     }
01213     if (sock == VNULL) {
01214         Vnm_print(2, "parsePBE: got NULL socket!\n");
01215         return -1;
01216     }
01217
01218     Vnm_print(0, "PBeparm_parseToken: trying %s...\n", tok);
01219
01220     if (Vstring_strcasecmp(tok, "mol") == 0) {
01221         return PBeparm_parseMOL(thee, sock);
01222     } else if (Vstring_strcasecmp(tok, "lpbe") == 0) {
01223         return PBeparm_parseLPBE(thee, sock);
01224     } else if (Vstring_strcasecmp(tok, "npbe") == 0) {
01225         return PBeparm_parseNPBE(thee, sock);
01226     } else if (Vstring_strcasecmp(tok, "lrpbe") == 0) {
01227         return PBeparm_parseLRPBE(thee, sock);
01228     } else if (Vstring_strcasecmp(tok, "nrpbe") == 0) {
01229         return PBeparm_parseNRPBE(thee, sock);
01230     } else if (Vstring_strcasecmp(tok, "smpbe") == 0) {
01231         return PBeparm_parseSMPBE(thee, sock);
01232     } else if (Vstring_strcasecmp(tok, "bcfl") == 0) {
01233         return PBeparm_parseBCFL(thee, sock);
01234     } else if (Vstring_strcasecmp(tok, "ion") == 0) {
01235         return PBeparm_parseION(thee, sock);
01236     } else if (Vstring_strcasecmp(tok, "pdie") == 0) {
01237         return PBeparm_parsePDIE(thee, sock);
01238     } else if (Vstring_strcasecmp(tok, "sdens") == 0) {
01239         return PBeparm_parseSDENS(thee, sock);
01240     } else if (Vstring_strcasecmp(tok, "sdie") == 0) {
01241         return PBeparm_parseSDIE(thee, sock);
01242     } else if (Vstring_strcasecmp(tok, "srfm") == 0) {
01243         return PBeparm_parseSRFM(thee, sock);
01244     } else if (Vstring_strcasecmp(tok, "srad") == 0) {
01245         return PBeparm_parseSRAD(thee, sock);
01246     } else if (Vstring_strcasecmp(tok, "swin") == 0) {
01247         return PBeparm_parseSWIN(thee, sock);
01248     } else if (Vstring_strcasecmp(tok, "temp") == 0) {
01249         return PBeparm_parseTEMP(thee, sock);
01250     } else if (Vstring_strcasecmp(tok, "usemap") == 0) {
01251         return PBeparm_parseUSEMAP(thee, sock);
01252     } else if (Vstring_strcasecmp(tok, "calcenergy") == 0) {
01253         return PBeparm_parseCALCENERGY(thee, sock);
01254     } else if (Vstring_strcasecmp(tok, "calcforce") == 0) {
01255         return PBeparm_parseCALCFORCE(thee, sock);
01256     } else if (Vstring_strcasecmp(tok, "write") == 0) {
01257         return PBeparm_parseWRITE(thee, sock);
01258     } else if (Vstring_strcasecmp(tok, "writemat") == 0) {
01259         return PBeparm_parseWRITEMAT(thee, sock);
01260
01261         /*-----*/
01262         /* Added by Michael Grabe */
01263         /*-----*/
01264
01265     } else if (Vstring_strcasecmp(tok, "zmem") == 0) {
01266         return PBeparm_parseZMEM(thee, sock);
01267     } else if (Vstring_strcasecmp(tok, "lmem") == 0) {
01268         return PBeparm_parseLMEM(thee, sock);
01269     } else if (Vstring_strcasecmp(tok, "mdie") == 0) {
01270         return PBeparm_parseMDIE(thee, sock);
01271     } else if (Vstring_strcasecmp(tok, "memv") == 0) {
01272         return PBeparm_parseMEMV(thee, sock);
01273     }
01274
01275     /*-----*/
01276
01277     return 0;
01278
01279 }

```

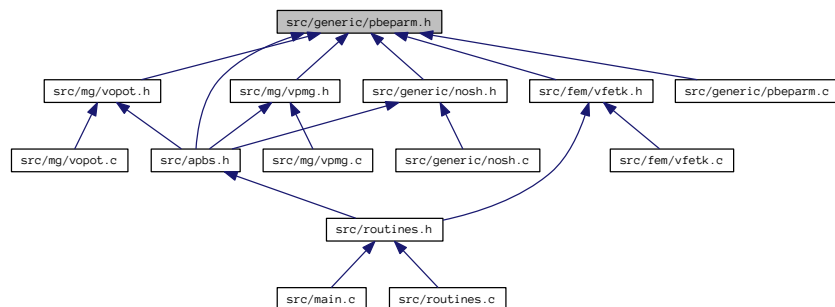
10.37 src/generic/pbeparm.h File Reference

Contains declarations for class PBEparm.

```
#include "apbscfg.h"
#include "malloc/malloc.h"
#include "generic/vhal.h"
#include "generic/vstring.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.

Macros

- #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 (Å) of specified ion species.
- VEXTERNC [PBEParm](#) * [PBEParm_ctor](#) ()
Construct PBEParm object.
- VEXTERNC int [PBEParm_ctor2](#) ([PBEParm](#) *thee)
FORTTRAN stub to construct PBEParm object.
- VEXTERNC void [PBEParm_dtor](#) ([PBEParm](#) **thee)
Object destructor.
- VEXTERNC void [PBEParm_dtor2](#) ([PBEParm](#) *thee)
FORTTRAN 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.

10.37.1 Detailed Description

Contains declarations for class PBEparm.

Version

\$Id\$

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*   Nathan A. Baker (nathan.baker@pnnl.gov)
*   Pacific Northwest National Laboratory
*
*   Additional contributing authors listed in the code documentation.
*
*   Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
*   Pacific Northwest National Laboratory, operated by Battelle Memorial
*   Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
*   Portions Copyright (c) 2002-2010, Washington University in St. Louis.
*   Portions Copyright (c) 2002-2010, Nathan A. Baker.
*   Portions Copyright (c) 1999-2002, The Regents of the University of
*   California.
*   Portions Copyright (c) 1995, Michael Holst.
*   All rights reserved.
*
*   Redistribution and use in source and binary forms, with or without
*   modification, are permitted provided that the following conditions are met:
*
*   Redistributions of source code must retain the above copyright notice, this
*   list of conditions and the following disclaimer.
*
*   Redistributions in binary form must reproduce the above copyright notice,
*   this list of conditions and the following disclaimer in the documentation
*   and/or other materials provided with the distribution.
*
*   Neither the name of the developer nor the names of its contributors may be
*   used to endorse or promote products derived from this software without
*   specific prior written permission.
*
*   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
*   AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
*   IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
*   ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
*   LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
*   CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
*   SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
*   INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
*   CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
*   ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
*   THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [pbeparm.h](#).

10.38 pbeparm.h

```

00001
00062 #ifndef _PBEPARM_H_
00063 #define _PBEPARM_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "malloc/malloc.h"
00068
00069 #include "generic/vhal.h"
00070 #include "generic/vstring.h"
00071
00075 #define PBEPARM_MAXWRITE 20
00076
00081 enum ePBEParm_calcEnergy {
00082     PCE_NO=0,
00083     PCE_TOTAL=1,
00084     PCE_COMPS=2
00085 };
00086
00091 typedef enum ePBEParm_calcEnergy PBEParm_calcEnergy;
00092
00097 enum ePBEParm_calcForce {
00098     PCF_NO=0,
00099     PCF_TOTAL=1,
00100     PCF_COMPS=2
00101 };
00102
00107 typedef enum ePBEParm_calcForce PBEParm_calcForce;
00108
00117 struct sPBEParm {
00118
00119     int molid;
00120     int setmolid;
00121     int useDielMap;
00122     int dielMapID;
00123     int useKappaMap;
00124     int kappaMapID;
00125     int usePotMap;
00126     int potMapID;
00127     int useChargeMap;
00128     int chargeMapID;
00129     Vhal_PBEType pbetype;
00130     int setpbetype;
00131     Vbcfl bcfl;
00132     int setbcfl;
00133     int nion;
00134     int setnion;
00135     double ionq[MAXION];
00136     double ionc[MAXION];
00137     double ionr[MAXION];
00138     int setion[MAXION];
00139     double pdie;
00140     int setpdie;
00141     double sdens;
00142     int setsdens;
00143     double sdie;
00144     int setsdie;
00145     Vsurf_Meth srfm;
00146     int setsrfm;
00147     double srad;
00148     int setsrad;
00149     double swin;
00150     int setswin;
00151     double temp;
00152     int settemp;
00153     double smsize;
00154     int setsmsize;
00155     double smvolume;
00156     int setsmvolume;
00157     PBEParm_calcEnergy calcenergy;
00158     int setcalcenergy;
00159     PBEParm_calcForce calcforce;
00160     int setcalcforce;
00161     /*-----*/
00162     /* Added by Michael Grabe */
00163     /*-----*/
00164
00174     double zmem;

```

```

00175     int  setzmem;
00176     double lmem;
00177     int  setLmem;
00178     double mdie;
00179     int  setmdie;
00180     double memv;
00181     int  setmemv;
00183     /*-----*/
00184
00185     int  numwrite;
00186     char writestem[PBE Parm_MAXWRITE][VMAX_ARGLEN];
00188     Vdata_Type writetype[PBE Parm_MAXWRITE];
00189     Vdata_Format writefmt[PBE Parm_MAXWRITE];
00191     int  writemat;
00194     int  setwritemat;
00195     char writematstem[VMAX_ARGLEN];
00196     int  writematflag;
00201     int  parsed;
00203 };
00204
00209 typedef struct sPBE Parm PBE Parm;
00210
00211 /* ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00212 // Class NOsh: Non-inlineable methods (mcsh.c)
00214
00220 VEXTERNC double PBE Parm_getIonCharge(
00221     PBE Parm *thee,
00222     int  iion
00223 );
00224
00230 VEXTERNC double PBE Parm_getIonConc(
00231     PBE Parm *thee,
00232     int  iion
00233 );
00234
00240 VEXTERNC double PBE Parm_getIonRadius(
00241     PBE Parm *thee,
00242     int  iion
00243 );
00244
00251 VEXTERNC PBE Parm* PBE Parm_ctor();
00252
00258 VEXTERNC int PBE Parm_ctor2(
00259     PBE Parm *thee
00260 );
00261
00266 VEXTERNC void PBE Parm_dtor(
00267     PBE Parm **thee
00268 );
00269
00274 VEXTERNC void PBE Parm_dtor2(
00275     PBE Parm *thee
00276 );
00277
00283 VEXTERNC int PBE Parm_check(
00284     PBE Parm *thee
00285 );
00286
00291 VEXTERNC void PBE Parm_copy(
00292     PBE Parm *thee,
00293     PBE Parm *parm
00294 );
00295
00302 VEXTERNC int PBE Parm_parseToken(
00303     PBE Parm *thee,
00304     char tok[VMAX_BUFSIZE],
00305     Vio *sock
00306 );
00307
00308
00309 #endif
00310

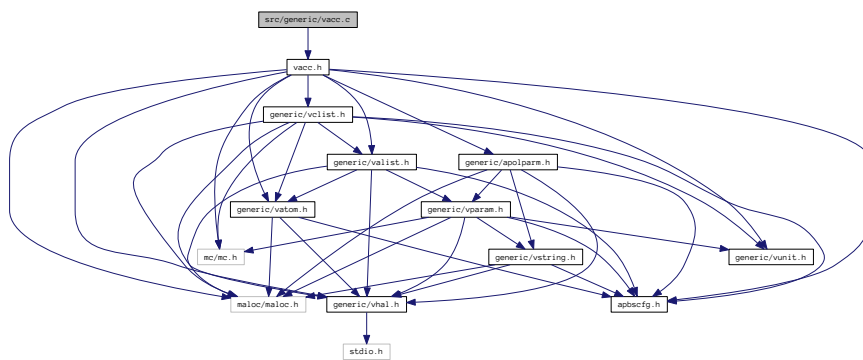
```


10.39 src/generic/vacc.c File Reference

Class Vacc methods.

```
#include "vacc.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)
FORTTRAN stub to construct the accessibility object.
- VPUBLIC void **Vacc_dtor** (**Vacc** **thee)
Destroy object.
- VPUBLIC void **Vacc_dtor2** (**Vacc** *thee)
FORTTRAN 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 [Vpmsg_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 [Vpmsg_splineAccAtom](#))
- VPUBLIC void [Vacc_atomdSAV](#) ([Vacc](#) *thee, double srاد, [Vatom](#) *atom, double *dSA)
Get the derivate 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 srاد, [Vatom](#) *atom, double *dSA)
Get the derivate of solvent accessible area.
- VPUBLIC void [Vacc_totalAtomdSASA](#) ([Vacc](#) *thee, double dpos, double srاد, [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.

10.39.1 Detailed Description

Class Vacc methods.

Author

Nathan Baker

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
```

```

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

Definition in file [vacc.c](#).

10.39.2 Function Documentation

10.39.2.1 VPRIVATE int ivdwAccExclus (Vacc * *thee*, double *center*[3], double *radius*, int *atomID*)

Determines if a point is within the union of the spheres centered at the atomic centers with radii equal to the sum of their van der Waals radii and the probe radius. Does not include contributions from the specified atom.

Returns

1 if accessible (outside the inflated van der Waals radius), 0 otherwise

Author

Nathan Baker

Parameters

<i>center</i>	Accessibility object
<i>radius</i>	Position to test
<i>atomID</i>	Radius of probe ID of atom to ignore

Definition at line 80 of file [vacc.c](#).

10.39.2.2 VPRIVATE double splineAcc (Vacc * *thee*, double *center*[VAPBS_DIM], double *win*, double *infrad*, VclistCell * *cell*)

Fast spline-based surface computation subroutine.

Returns

Spline value

Author

Todd Dolinsky and Nathan Baker

Parameters

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

Definition at line 493 of file [vacc.c](#).

10.39.2.3 VPRIVATE int Vacc_allocate (Vacc * *thee*)

Allocate (and clear) space for storage

Definition at line 193 of file [vacc.c](#).

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

Check and store parameters passed to constructor

Definition at line 148 of file [vacc.c](#).

10.40 vacc.c

```

00001
00057 #include "vacc.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_VACC)
00062
00063 VPUBLIC unsigned long int Vacc_memChk(Vacc *thee) {
00064     if (thee == VNULL)
00065         return 0;
00066     return Vmem_bytes(thee->mem);
00067 }
00068
00069 #endif /* if !defined(VINLINE_VACC) */
00070
00080 VPRIVATE int ivdwAccExclus(
00081     Vacc *thee,
00082     double center[3],
00083     double radius,
00084     int atomID
00085 ) {
00086
00087     int iatom;
00088     double dist2,
00089         *apos;
00090     Vatom *atom;
00091     VclistCell *cell;
00092
00093     VASSERT(thee != VNULL);
00094
00095     /* We can only test probes with radii less than the max specified */
00096     if (radius > Vclist_maxRadius(thee->clist)) {
00097         Vnm_print(2,
00098             "Vacc_ivdwAcc: got radius (%g) bigger than max radius (%g)\n",
00099             radius, Vclist_maxRadius(thee->clist));
00100         VASSERT(0);
00101     }
00102
00103     /* Get the relevant cell from the cell list */
00104     cell = Vclist_getCell(thee->clist, center);
00105
00106     /* If we have no cell, then no atoms are nearby and we're definitely
00107      * accessible */
00108     if (cell == VNULL) {

```

```

00109         return 1;
00110     }
00111
00112     /* Otherwise, check for overlap with the atoms in the cell */
00113     for (iatom=0; iatom<cell->natoms; iatom++) {
00114         atom = cell->atoms[iatom];
00115
00116         // We don't actually need to test this if the atom IDs do match; don't compute this if we're
00117         // comparing atom against itself.
00118         if (atom->id == atomID) continue;
00119
00120         apos = atom->position;
00121         dist2 = VSQR(center[0]-apos[0]) + VSQR(center[1]-apos[1])
00122             + VSQR(center[2]-apos[2]);
00123         if (dist2 < VSQR(atom->radius+radius)){
00124             return 0;
00125         }
00126     }
00127     /* If we're still here, then the point is accessible */
00128     return 1;
00129 }
00130
00131
00132 VPUBLIC Vacc* Vacc_ctor(Valist *alist,
00133                        Vclist *clist,
00134                        double surf_density /* Surface density */
00135                        ) {
00136
00137     Vacc *thee = VNULL;
00138
00139     /* Set up the structure */
00140     thee = (Vacc*)Vmem_malloc(VNULL, 1, sizeof(Vacc) );
00141     VASSERT( thee != VNULL);
00142     VASSERT( Vacc_ctor2(thee, alist, clist, surf_density));
00143     return thee;
00144 }
00145
00146
00147 VPRIVATE int Vacc_storeParms(Vacc *thee,
00148                             Valist *alist,
00149                             Vclist *clist,
00150                             double surf_density /* Surface density */
00151                             ) {
00152
00153     int nsphere,
00154         iatom;
00155     double maxrad = 0.0,
00156         maxarea,
00157         rad;
00158     Vatom *atom;
00159
00160     if (alist == VNULL) {
00161         Vnm_print(2, "Vacc_storeParms: Got NULL Valist!\n");
00162         return 0;
00163     } else thee->alist = alist;
00164     if (clist == VNULL) {
00165         Vnm_print(2, "Vacc_storeParms: Got NULL Vclist!\n");
00166         return 0;
00167     } else thee->clist = clist;
00168     thee->surf_density = surf_density;
00169
00170     /* Loop through the atoms to determine the maximum radius */
00171     maxrad = 0.0;
00172     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
00173         atom = Valist_getAtom(alist, iatom);
00174         rad = Vatom_getRadius(atom);
00175         if (rad > maxrad) maxrad = rad;
00176     }
00177     maxrad = maxrad + Vclist_maxRadius(thee->clist);
00178
00179     maxarea = 4.0*VPI*maxrad*maxrad;
00180     nsphere = (int)ceil(maxarea*surf_density);
00181
00182     Vnm_print(0, "Vacc_storeParms: Surf. density = %g\n", surf_density);
00183     Vnm_print(0, "Vacc_storeParms: Max area = %g\n", maxarea);
00184     thee->refSphere = VaccSurf_refSphere(thee->mem, nsphere);
00185     Vnm_print(0, "Vacc_storeParms: Using %d-point reference sphere\n",
00186             thee->refSphere->npts);
00187
00188     return 1;
00189 }

```

```

00190 }
00191
00193 VPRIVATE int Vacc_allocate(Vacc *thee) {
00194     int i,
00195         natoms;
00196
00197     natoms = Valist_getNumberAtoms(thee->alist);
00198
00199     thee->atomFlags = (int*)Vmem_malloc(thee->mem, natoms, sizeof(int));
00200     if (thee->atomFlags == VNULL) {
00201         Vnm_print(2,
00202             "Vacc_allocate: Failed to allocate %d (int)s for atomFlags!\n",
00203             natoms);
00204         return 0;
00205     }
00206     for (i=0; i<natoms; i++) (thee->atomFlags)[i] = 0;
00207
00208     return 1;
00209 }
00210 }
00211
00212
00213 VPUBLIC int Vacc_ctor2(Vacc *thee,
00214                       Valist *alist,
00215                       Vclist *clist,
00216                       double surf_density
00217                       ) {
00218
00219     /* Check and store parameters */
00220     if (!Vacc_storeParms(thee, alist, clist, surf_density)) {
00221         Vnm_print(2, "Vacc_ctor2: parameter check failed!\n");
00222         return 0;
00223     }
00224
00225     /* Set up memory management object */
00226     thee->mem = Vmem_ctor("APBS::VACC");
00227     if (thee->mem == VNULL) {
00228         Vnm_print(2, "Vacc_ctor2: memory object setup failed!\n");
00229         return 0;
00230     }
00231
00232     /* Setup and check probe */
00233     thee->surf = VNULL;
00234
00235     /* Allocate space */
00236     if (!Vacc_allocate(thee)) {
00237         Vnm_print(2, "Vacc_ctor2: memory allocation failed!\n");
00238         return 0;
00239     }
00240
00241     return 1;
00242 }
00243
00244
00245 VPUBLIC void Vacc_dtor(Vacc **thee) {
00246
00247     if ((*thee) != VNULL) {
00248         Vacc_dtor2(*thee);
00249         Vmem_free(VNULL, 1, sizeof(Vacc), (void **)thee);
00250         (*thee) = VNULL;
00251     }
00252 }
00253 }
00254
00255 VPUBLIC void Vacc_dtor2(Vacc *thee) {
00256
00257     int i,
00258         natoms;
00259
00260     natoms = Valist_getNumberAtoms(thee->alist);
00261     Vmem_free(thee->mem, natoms, sizeof(int), (void *)&(thee->atomFlags));
00262
00263     if (thee->refSphere != VNULL) {
00264         VaccSurf_dtor(&(thee->refSphere));
00265         thee->refSphere = VNULL;
00266     }
00267     if (thee->surf != VNULL) {
00268         for (i=0; i<natoms; i++) VaccSurf_dtor(&(thee->surf[i]));
00269         Vmem_free(thee->mem, natoms, sizeof(VaccSurf *),
00270             (void *)&(thee->surf));
00271         thee->surf = VNULL;

```

```

00272     }
00273
00274     Vmem_dtor(&(thee->mem));
00275 }
00276
00277 VPUBLIC double Vacc_vdwAcc(Vacc *thee,
00278                          double center[3]
00279                          ) {
00280
00281     VclistCell *cell;
00282     Vatom *atom;
00283     int iatom;
00284     double *apos,
00285            dist2;
00286
00287     /* Get the relevant cell from the cell list */
00288     cell = Vclist_getCell(thee->clist, center);
00289
00290     /* If we have no cell, then no atoms are nearby and we're definitely
00291      * accessible */
00292     if (cell == VNULL) return 1.0;
00293
00294     /* Otherwise, check for overlap with the atoms in the cell */
00295     for (iatom=0; iatom<cell->natoms; iatom++) {
00296         atom = cell->atoms[iatom];
00297         apos = Vatom_getPosition(atom);
00298         dist2 = VSQR(center[0]-apos[0]) + VSQR(center[1]-apos[1])
00299              + VSQR(center[2]-apos[2]);
00300         if (dist2 < VSQR(Vatom_getRadius(atom))) return 0.0;
00301     }
00302
00303     /* If we're still here, then the point is accessible */
00304     return 1.0;
00305 }
00306
00307 VPUBLIC double Vacc_ivdwAcc(Vacc *thee,
00308                          double center[3],
00309                          double radius
00310                          ) {
00311
00312     return (double)ivdwAccExclus(thee, center, radius, -1);
00313 }
00314 }
00315
00316 VPUBLIC void Vacc_splineAccGradAtomNorm(Vacc *thee,
00317                                       double center[VAPBS_DIM],
00318                                       double win,
00319                                       double infrad,
00320                                       Vatom *atom,
00321                                       double *grad
00322                                       ) {
00323
00324     int i;
00325     double dist,
00326            *apos,
00327            arad,
00328            sm,
00329            sm2,
00330            w2i, /* inverse of win squared */
00331            w3i, /* inverse of win cubed */
00332            mygrad,
00333            mychi = 1.0; /* Char. func. value for given atom */
00334
00335     VASSERT(thee != NULL);
00336
00337     /* Inverse squared window parameter */
00338     w2i = 1.0/(win*win);
00339     w3i = 1.0/(win*win*win);
00340
00341     /* The grad is zero by default */
00342     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
00343
00344     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
00345      * *** MAGNITUDE OF THE FORCE *** */
00346     apos = Vatom_getPosition(atom);
00347     /* Zero-radius atoms don't contribute */
00348     if (Vatom_getRadius(atom) > 0.0) {
00349         arad = Vatom_getRadius(atom) + infrad;
00350         dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
00351                   + VSQR(apos[2]-center[2]));
00352         /* If we're inside an atom, the entire characteristic function

```



```

00353     * will be zero and the grad will be zero, so we can stop */
00354     if (dist < (arad - win)) return;
00355     /* Likewise, if we're outside the smoothing window, the characteristic
00356     * function is unity and the grad will be zero, so we can stop */
00357     else if (dist > (arad + win)) return;
00358     /* Account for floating point error at the border
00359     * NAB: COULDN'T THESE TESTS BE COMBINED AS BELOW
00360     * (Vacc_splineAccAtom)? */
00361     else if ((VABS(dist - (arad - win)) < VSMALL) ||
00362              (VABS(dist - (arad + win)) < VSMALL)) return;
00363     /* If we're inside the smoothing window */
00364     else {
00365         sm = dist - arad + win;
00366         sm2 = VSQR(sm);
00367         mychi = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
00368         mygrad = 1.5*sm*w2i - 0.75*sm2*w3i;
00369     }
00370     /* Now assemble the grad vector */
00371     VASSERT(mychi > 0.0);
00372     for (i=0; i<VAPBS_DIM; i++)
00373         grad[i] = -(mygrad/mychi)*((center[i] - apos[i])/dist);
00374 }
00375 }
00376
00377 VPUBLIC void Vacc_splineAccGradAtomUnnorm(Vacc *thee,
00378                                           double center[VAPBS_DIM],
00379                                           double win,
00380                                           double infrad,
00381                                           Vatom *atom,
00382                                           double *grad
00383                                           ) {
00384
00385     int i;
00386     double dist,
00387            *apos,
00388            arad,
00389            sm,
00390            sm2,
00391            w2i, /* Inverse of win squared */
00392            w3i, /* Inverse of win cubed */
00393            mygrad,
00394            mychi = 1.0; /* Char. func. value for given atom */
00395
00396     VASSERT(thee != NULL);
00397
00398     /* Inverse squared window parameter */
00399     w2i = 1.0/(win*win);
00400     w3i = 1.0/(win*win*win);
00401
00402     /* The grad is zero by default */
00403     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
00404
00405     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
00406     * *** MAGNITUDE OF THE FORCE *** */
00407     apos = Vatom_getPosition(atom);
00408     /* Zero-radius atoms don't contribute */
00409     if (Vatom_getRadius(atom) > 0.0) {
00410         arad = Vatom_getRadius(atom) + infrad;
00411         dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
00412                     + VSQR(apos[2]-center[2]));
00413         /* If we're inside an atom, the entire characteristic function
00414         * will be zero and the grad will be zero, so we can stop */
00415         if (dist < (arad - win)) return;
00416         /* Likewise, if we're outside the smoothing window, the characteristic
00417         * function is unity and the grad will be zero, so we can stop */
00418         else if (dist > (arad + win)) return;
00419         /* Account for floating point error at the border
00420         * NAB: COULDN'T THESE TESTS BE COMBINED AS BELOW
00421         * (Vacc_splineAccAtom)? */
00422         else if ((VABS(dist - (arad - win)) < VSMALL) ||
00423                  (VABS(dist - (arad + win)) < VSMALL)) return;
00424         /* If we're inside the smoothing window */
00425         else {
00426             sm = dist - arad + win;
00427             sm2 = VSQR(sm);
00428             mychi = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
00429             mygrad = 1.5*sm*w2i - 0.75*sm2*w3i;
00430         }
00431         /* Now assemble the grad vector */
00432         VASSERT(mychi > 0.0);
00433         for (i=0; i<VAPBS_DIM; i++)

```

```

00434         grad[i] = -(mygrad)*((center[i] - apos[i])/dist);
00435     }
00436 }
00437
00438 VPUBLIC double Vacc_splineAccAtom(Vacc *thee,
00439                                   double center[VAPBS_DIM],
00440                                   double win,
00441                                   double infrad,
00442                                   Vatom *atom
00443                                   ) {
00444
00445     double dist,
00446           *apos,
00447           arad,
00448           sm,
00449           sm2,
00450           w2i, /* Inverse of win squared */
00451           w3i, /* Inverse of win cubed */
00452           value,
00453           stot,
00454           sctot;
00455
00456     VASSERT(thee != NULL);
00457
00458     /* Inverse squared window parameter */
00459     w2i = 1.0/(win*win);
00460     w3i = 1.0/(win*win*win);
00461
00462     apos = Vatom_getPosition(atom);
00463     /* Zero-radius atoms don't contribute */
00464     if (Vatom_getRadius(atom) > 0.0) {
00465         arad = Vatom_getRadius(atom) + infrad;
00466         stot = arad + win;
00467         sctot = VMAX2(0, (arad - win));
00468         dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
00469                     + VSQR(apos[2]-center[2]));
00470         /* If we're inside an atom, the entire characteristic function
00471          * will be zero */
00472         if ((dist < sctot) || (VABS(dist - sctot) < VSMALL)){
00473             value = 0.0;
00474         /* We're outside the smoothing window */
00475         } else if ((dist > stot) || (VABS(dist - stot) < VSMALL)) {
00476             value = 1.0;
00477         /* We're inside the smoothing window */
00478         } else {
00479             sm = dist - arad + win;
00480             sm2 = VSQR(sm);
00481             value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
00482         }
00483     } else value = 1.0;
00484
00485     return value;
00486 }
00487
00493 VPRIVATE double splineAcc(
00494     Vacc *thee,
00495     double center[VAPBS_DIM],
00496     double win,
00497     double infrad,
00498     VclistCell *cell
00499 ) {
00500
00501     int atomID, iatom;
00502     Vatom *atom;
00503     double value = 1.0;
00504
00505     VASSERT(thee != NULL);
00506
00507     /* Now loop through the atoms assembling the characteristic function */
00508     for (iatom=0; iatom<cell->natoms; iatom++) {
00509
00510         atom = cell->atoms[iatom];
00511         atomID = atom->id;
00512
00513         /* Check to see if we've counted this atom already */
00514         if ( !(thee->atomFlags[atomID]) ) {
00515
00516             thee->atomFlags[atomID] = 1;
00517             value *= Vacc_splineAccAtom(thee, center, win, infrad, atom);
00518
00519             if (value < VSMALL) return value;
00520

```

```

00521     }
00522 }
00523
00524     return value;
00525 }
00526
00527
00528 VPUBLIC double Vacc_splineAcc(Vacc *thee, double center[
    VAPBS_DIM], double win,
00529     double infrad) {
00530
00531     VclistCell *cell;
00532     Vatom *atom;
00533     int iatom, atomID;
00534
00535
00536     VASSERT(thee != NULL);
00537
00538     if (Vclist_maxRadius(thee->clist) < (win + infrad)) {
00539         Vnm_print(2, "Vacc_splineAcc: Vclist has max_radius=%g;\n",
00540             Vclist_maxRadius(thee->clist));
00541         Vnm_print(2, "Vacc_splineAcc: Insufficient for win=%g, infrad=%g\n",
00542             win, infrad);
00543         VASSERT(0);
00544     }
00545
00546     /* Get a cell or VNULL; in the latter case return 1.0 */
00547     cell = Vclist_getCell(thee->clist, center);
00548     if (cell == VNULL) return 1.0;
00549
00550     /* First, reset the list of atom flags
00551      * NAB: THIS SEEMS VERY INEFFICIENT */
00552     for (iatom=0; iatom<cell->natoms; iatom++) {
00553         atom = cell->atoms[iatom];
00554         atomID = atom->id;
00555         thee->atomFlags[atomID] = 0;
00556     }
00557
00558     return splineAcc(thee, center, win, infrad, cell);
00559 }
00560
00561 VPUBLIC void Vacc_splineAccGrad(Vacc *thee, double center[
    VAPBS_DIM],
00562     double win, double infrad, double *grad) {
00563
00564     int iatom, i, atomID;
00565     double acc = 1.0;
00566     double tgrad[VAPBS_DIM];
00567     VclistCell *cell;
00568     Vatom *atom = VNULL;
00569
00570     VASSERT(thee != NULL);
00571
00572     if (Vclist_maxRadius(thee->clist) < (win + infrad)) {
00573         Vnm_print(2, "Vacc_splineAccGrad: Vclist max_radius=%g;\n",
00574             Vclist_maxRadius(thee->clist));
00575         Vnm_print(2, "Vacc_splineAccGrad: Insufficient for win=%g, infrad=%g\n",
00576             win, infrad);
00577         VASSERT(0);
00578     }
00579
00580     /* Reset the gradient */
00581     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
00582
00583     /* Get the cell; check for nullity */
00584     cell = Vclist_getCell(thee->clist, center);
00585     if (cell == VNULL) return;
00586
00587     /* Reset the list of atom flags */
00588     for (iatom=0; iatom<cell->natoms; iatom++) {
00589         atom = cell->atoms[iatom];
00590         atomID = atom->id;
00591         thee->atomFlags[atomID] = 0;
00592     }
00593
00594     /* Get the local accessibility */
00595     acc = splineAcc(thee, center, win, infrad, cell);
00596
00597     /* Accumulate the gradient of all local atoms */
00598     if (acc > VSMALL) {
00599         for (iatom=0; iatom<cell->natoms; iatom++) {

```

```

00600         atom = cell->atoms[iatom];
00601         Vacc_splineAccGradAtomNorm(thee, center, win, infrad, atom, tgrad);
00602     }
00603     for (i=0; i<VAPBS_DIM; i++) grad[i] += tgrad[i];
00604 }
00605 for (i=0; i<VAPBS_DIM; i++) grad[i] *= -acc;
00606 }
00607
00608 VPUBLIC double Vacc_molAcc(Vacc *thee, double center[VAPBS_DIM],
00609     double radius) {
00610     double rc;
00611
00612     /* ***** CHECK IF OUTSIDE ATOM+PROBE RADIUS SURFACE ***** */
00613     if (Vacc_ivdwAcc(thee, center, radius) == 1.0) {
00614         /* Vnm_print(2, "DEBUG: ivdwAcc = 1.0\n"); */
00615         rc = 1.0;
00616
00617         /* ***** CHECK IF INSIDE ATOM RADIUS SURFACE ***** */
00618     } else if (Vacc_vdwAcc(thee, center) == 0.0) {
00619         /* Vnm_print(2, "DEBUG: vdwAcc = 0.0\n"); */
00620         rc = 0.0;
00621
00622         /* ***** CHECK IF OUTSIDE MOLECULAR SURFACE ***** */
00623     } else {
00624         /* Vnm_print(2, "DEBUG: calling fastMolAcc...\n"); */
00625         rc = Vacc_fastMolAcc(thee, center, radius);
00626     }
00627
00628     return rc;
00629 }
00630
00631 VPUBLIC double Vacc_fastMolAcc(Vacc *thee, double center[
00632     VAPBS_DIM],
00633     double radius) {
00634     Vatom *atom;
00635     VaccSurf *surf;
00636     VclistCell *cell;
00637     int ipt, iatom, atomID;
00638     double dist2, rad2;
00639
00640     rad2 = radius*radius;
00641
00642     /* Check to see if the SAS has been defined */
00643     if (thee->surf == VNULL) Vacc_SASA(thee, radius);
00644
00645     /* Get the cell associated with this point */
00646     cell = Vclist_getCell(thee->clist, center);
00647     if (cell == VNULL) {
00648         Vnm_print(2, "Vacc_fastMolAcc: unexpected VNULL VclistCell!\n");
00649         return 1.0;
00650     }
00651
00652     /* Loop through all the atoms in the cell */
00653     for (iatom=0; iatom<cell->natoms; iatom++) {
00654         atom = cell->atoms[iatom];
00655         atomID = Vatom_getAtomID(atom);
00656         surf = thee->surf[atomID];
00657         /* Loop through all SAS points associated with this atom */
00658         for (ipt=0; ipt<surf->npts; ipt++) {
00659             /* See if we're within a probe radius of the point */
00660             dist2 = VSQR(center[0]-(surf->xpts[ipt]))
00661                 + VSQR(center[1]-(surf->ypts[ipt]))
00662                 + VSQR(center[2]-(surf->zpts[ipt]));
00663             if (dist2 < rad2) return 1.0;
00664         }
00665     }
00666
00667     /* If all else failed, we are not inside the molecular surface */
00668     return 0.0;
00669 }
00670
00671 #if defined(HAVE_MC_H)
00672 VPUBLIC void Vacc_writeGMV(Vacc *thee, double radius, int meth, Gem *gm,

```

```

00680 char *iodev, char *iofmt, char *iohost, char *iofile) {
00681
00682     double *accVals[MAXV], coord[3];
00683     Vio *sock;
00684     int ivert, icoord;
00685
00686     for (ivert=0; ivert<MAXV; ivert++) accVals[ivert] = VNULL;
00687     accVals[0] = (void *)Vmem_malloc(thee->mem, Gem_numVV(gm), sizeof(double));
00688     accVals[1] = (void *)Vmem_malloc(thee->mem, Gem_numVV(gm), sizeof(double));
00689     for (ivert=0; ivert<Gem_numVV(gm); ivert++) {
00690         for (icoord=0; icoord<3; icoord++)
00691             coord[icoord] = VV_coord(Gem_VV(gm, ivert), icoord);
00692         if (meth == 0) {
00693             accVals[0][ivert] = Vacc_molAcc(thee, coord, radius);
00694             accVals[1][ivert] = Vacc_molAcc(thee, coord, radius);
00695         } else if (meth == 1) {
00696             accVals[0][ivert] = Vacc_ivdwAcc(thee, coord, radius);
00697             accVals[1][ivert] = Vacc_ivdwAcc(thee, coord, radius);
00698         } else if (meth == 2) {
00699             accVals[0][ivert] = Vacc_vdwAcc(thee, coord);
00700             accVals[1][ivert] = Vacc_vdwAcc(thee, coord);
00701         } else VASSERT(0);
00702     }
00703     sock = Vio_ctor(iodev, iofmt, iohost, iofile, "w");
00704     Gem_writeGMV(gm, sock, 1, accVals);
00705     Vio_dtor(&sock);
00706     Vmem_free(thee->mem, Gem_numVV(gm), sizeof(double),
00707             (void **)&(accVals[0]));
00708     Vmem_free(thee->mem, Gem_numVV(gm), sizeof(double),
00709             (void **)&(accVals[1]));
00710 }
00711 #endif /* defined(HAVE_MC_H) */
00712
00713 VPUBLIC double Vacc_SASA(Vacc *thee,
00714                         double radius
00715                         ) {
00716
00717     int i,
00718         natom;
00719     double area;
00720     /*apos; // gcc says unused
00721     Vatom *atom;
00722     VaccSurf *asurf;
00723
00724     time_t ts; // PCE: temp
00725     ts = clock();
00726
00727     //unsigned long long mbeg; // gcc says unused
00728
00729     natom = Valist_getNumberAtoms(thee->alist);
00730
00731     /* Check to see if we need to build the surface */
00732     if (thee->surf == VNULL) {
00733         thee->surf = Vmem_malloc(thee->mem, natom, sizeof(VaccSurf *));
00734
00735     #if defined(DEBUG_MAC_OSX_OCL) || defined(DEBUG_MAC_OSX_STANDARD)
00736     #include "mach_chud.h"
00737         machm_(&mbeg);
00738     #pragma omp parallel for private(i,atom)
00739     #endif
00740         for (i=0; i<natom; i++) {
00741             atom = Valist_getAtom(thee->alist, i);
00742             /* NOTE: RIGHT NOW WE DO THIS FOR THE ENTIRE MOLECULE WHICH IS
00743              * INCREDIBLY INEFFICIENT, PARTICULARLY DURING FOCUSING!!! */
00744             thee->surf[i] = Vacc_atomSurf(thee, atom, thee->
refSphere,
                                radius);
00745         }
00746     }
00747
00748     /* Calculate the area */
00749     area = 0.0;
00750     for (i=0; i<natom; i++) {
00751         atom = Valist_getAtom(thee->alist, i);
00752         asurf = thee->surf[i];
00753         /* See if this surface needs to be rebuilt */
00754         if (asurf->probe_radius != radius) {
00755             Vnm_print(2, "Vacc_SASA: Warning -- probe radius changed from %g to %g!\n",
00756                     asurf->probe_radius, radius);
00757             VaccSurf_dtor2(asurf);
00758             thee->surf[i] = Vacc_atomSurf(thee, atom, thee->

```

```

        refSphere, radius);
00760         asurf = thee->surf[i];
00761     }
00762     area += (asurf->area);
00763 }
00764
00765 #if defined(DEBUG_MAC_OSX_OCL) || defined(DEBUG_MAC_OSX_STANDARD)
00766 mets_(&mbeg, "Vacc_SASA - Parallel");
00767 #endif
00768
00769 Vnm_print(0, "Vacc_SASA: Time elapsed: %f\n", ((double)clock() - ts) / CLOCKS_PER_SEC);
00770 return area;
00771 }
00772 }
00773
00774 VPUBLIC double Vacc_totalSASA(Vacc *thee, double radius) {
00775     return Vacc_SASA(thee, radius);
00776 }
00777 }
00778
00779 VPUBLIC double Vacc_atomSASA(Vacc *thee, double radius, Vatom *atom) {
00780     VaccSurf *asurf;
00781     int id;
00782
00783     if (thee->surf == VNULL) Vacc_SASA(thee, radius);
00784
00785     id = Vatom_getAtomID(atom);
00786     asurf = thee->surf[id];
00787
00788     /* See if this surface needs to be rebuilt */
00789     if (asurf->probe_radius != radius) {
00790         Vnm_print(2, "Vacc_SASA: Warning -- probe radius changed from %g to %g!\n",
00791             asurf->probe_radius, radius);
00792         VaccSurf_dtor2(asurf);
00793         thee->surf[id] = Vacc_atomSurf(thee, atom, thee->
00794             refSphere, radius);
00795         asurf = thee->surf[id];
00796     }
00797
00798     return asurf->area;
00799 }
00800 }
00801 }
00802
00803 VPUBLIC VaccSurf* VaccSurf_ctor(Vmem *mem, double probe_radius, int nsphere) {
00804     VaccSurf *thee;
00805
00806     //thee = Vmem_malloc(mem, 1, sizeof(Vacc) );
00807     if (nsphere >= MAX_SPHERE_PTS) {
00808         Vnm_print(2, "VaccSurf_ctor: Error! The requested number of grid points (%d) exceeds the maximum
00809             (%d)!\n", nsphere, MAX_SPHERE_PTS);
00810         Vnm_print(2, "VaccSurf_ctor: Please check the variable MAX_SPHERE_PTS to reset.\n");
00811         VASSERT(0);
00812     }
00813     thee = (VaccSurf*)calloc(1, sizeof(Vacc));
00814     VASSERT( VaccSurf_ctor2(thee, mem, probe_radius, nsphere) );
00815
00816     return thee;
00817 }
00818
00819 VPUBLIC int VaccSurf_ctor2(VaccSurf *thee, Vmem *mem, double probe_radius,
00820     int nsphere) {
00821     if (thee == VNULL)
00822         return 0;
00823
00824     thee->mem = mem;
00825     thee->npts = nsphere;
00826     thee->probe_radius = probe_radius;
00827     thee->area = 0.0;
00828
00829     if (thee->npts > 0) {
00830         thee->xpts = Vmem_malloc(thee->mem, thee->npts, sizeof(double));
00831         thee->ypts = Vmem_malloc(thee->mem, thee->npts, sizeof(double));
00832         thee->zpts = Vmem_malloc(thee->mem, thee->npts, sizeof(double));
00833         thee->bpts = Vmem_malloc(thee->mem, thee->npts, sizeof(char));
00834     } else {
00835         thee->xpts = VNULL;
00836         thee->ypts = VNULL;
00837         thee->zpts = VNULL;

```

```

00838     thee->bpts = VNULL;
00839 }
00840
00841     return 1;
00842 }
00843
00844 VPUBLIC void VaccSurf_dtor(VaccSurf **thee) {
00845     Vmem *mem;
00846
00847     if ((*thee) != VNULL) {
00848         mem = (*thee)->mem;
00849         VaccSurf_dtor2(*thee);
00850         //Vmem_free(mem, 1, sizeof(VaccSurf), (void **)thee);
00851         free(*thee);
00852         (*thee) = VNULL;
00853     }
00854 }
00855
00856 }
00857
00858 VPUBLIC void VaccSurf_dtor2(VaccSurf *thee) {
00859     if (thee->npts > 0) {
00860         Vmem_free(thee->mem, thee->npts, sizeof(double),
00861             (void *)&(thee->xpts));
00862         Vmem_free(thee->mem, thee->npts, sizeof(double),
00863             (void *)&(thee->ypts));
00864         Vmem_free(thee->mem, thee->npts, sizeof(double),
00865             (void *)&(thee->zpts));
00866         Vmem_free(thee->mem, thee->npts, sizeof(char),
00867             (void *)&(thee->bpts));
00868     }
00869 }
00870
00871 }
00872
00873 VPUBLIC VaccSurf* Vacc_atomSurf(Vacc *thee,
00874     Vatom *atom,
00875     VaccSurf *ref,
00876     double prad
00877 ) {
00878     VaccSurf *surf;
00879     int i,
00880         j,
00881         npts,
00882         atomID;
00883     double arad,
00884         rad,
00885         pos[3],
00886         *apos;
00887     char bpts[MAX_SPHERE_PTS];
00888
00889     /* Get atom information */
00890     arad = Vatom_getRadius(atom);
00891     apos = Vatom_getPosition(atom);
00892     atomID = Vatom_getAtomID(atom);
00893
00894     if (arad < VSMALL) {
00895         return VaccSurf_ctor(thee->mem, prad, 0);
00896     }
00897
00898     rad = arad + prad;
00899
00900     /* Determine which points will contribute */
00901     npts = 0;
00902     for (i=0; i<ref->npts; i++) {
00903         /* Reset point flag: zero-radius atoms do not contribute */
00904         pos[0] = rad*(ref->xpts[i]) + apos[0];
00905         pos[1] = rad*(ref->ypts[i]) + apos[1];
00906         pos[2] = rad*(ref->zpts[i]) + apos[2];
00907         if (ivdwAccExclus(thee, pos, prad, atomID)) {
00908             npts++;
00909             bpts[i] = 1;
00910         } else {
00911             bpts[i] = 0;
00912         }
00913     }
00914
00915     /* Allocate space for the points */
00916     surf = VaccSurf_ctor(thee->mem, prad, npts);
00917
00918

```

```

00919     /* Assign the points */
00920     j = 0;
00921     for (i=0; i<ref->npts; i++) {
00922         if (bpts[i]) {
00923             surf->bpts[j] = 1;
00924             surf->xpts[j] = rad*(ref->xpts[i]) + apos[0];
00925             surf->ypts[j] = rad*(ref->ypts[i]) + apos[1];
00926             surf->zpts[j] = rad*(ref->zpts[i]) + apos[2];
00927             j++;
00928         }
00929     }
00930
00931     /* Assign the area */
00932     surf->area = 4.0*VPI*rad*rad*((double)(surf->npts))/((double)(ref->
npts));
00933
00934     return surf;
00935
00936 }
00937
00938 VPUBLIC VaccSurf* VaccSurf_refSphere(Vmem *mem, int npts) {
00939     VaccSurf *surf;
00940     int nactual, i, itheta, ntheta, iphi, nphimax, nphi;
00941     double frac;
00942     double sintheta, costheta, theta, dtheta;
00943     double sinphi, cosphi, phi, dphi;
00944
00945     /* Setup "constants" */
00946     frac = ((double)(npts))/4.0;
00947     ntheta = VRINT(VSQRT(Vunit_pi*frac));
00948     dtheta = Vunit_pi/((double)(ntheta));
00949     nphimax = 2*ntheta;
00950
00951     /* Count the actual number of points to be used */
00952     nactual = 0;
00953     for (itheta=0; itheta<ntheta; itheta++) {
00954         theta = dtheta*((double)(itheta));
00955         sintheta = VSIN(theta);
00956         costheta = VCOS(theta);
00957         nphi = VRINT(sintheta*nphimax);
00958         nactual += nphi;
00959     }
00960
00961     /* Allocate space for the points */
00962     surf = VaccSurf_ctor(mem, 1.0, nactual);
00963
00964     /* Clear out the boolean array */
00965     for (i=0; i<nactual; i++) surf->bpts[i] = 1;
00966
00967     /* Assign the points */
00968     nactual = 0;
00969     for (itheta=0; itheta<ntheta; itheta++) {
00970         theta = dtheta*((double)(itheta));
00971         sintheta = VSIN(theta);
00972         costheta = VCOS(theta);
00973         nphi = VRINT(sintheta*nphimax);
00974         if (nphi != 0) {
00975             dphi = 2*Vunit_pi/((double)(nphi));
00976             for (iphi=0; iphi<nphi; iphi++) {
00977                 phi = dphi*((double)(iphi));
00978                 sinphi = VSIN(phi);
00979                 cosphi = VCOS(phi);
00980                 surf->xpts[nactual] = cosphi * sintheta;
00981                 surf->ypts[nactual] = sinphi * sintheta;
00982                 surf->zpts[nactual] = costheta;
00983                 nactual++;
00984             }
00985         }
00986     }
00987
00988     surf->npts = nactual;
00989
00990     return surf;
00991 }
00992
00993
00994 VPUBLIC VaccSurf* Vacc_atomSASPoints(Vacc *three, double radius,
00995     Vatom *atom) {
00996
00997     VaccSurf *asurf = VNULL;
00998     int id;

```



```

00999
01000     if (thee->surf == VNULL) Vacc_SASA(thee, radius);
01001     id = Vatom_getAtomID(atom);
01002
01003     asurf = thee->surf[id];
01004
01005     /* See if this surface needs to be rebuilt */
01006     if (asurf->probe_radius != radius) {
01007         Vnm_print(2, "Vacc_SASA: Warning -- probe radius changed from %g to %g!\n",
01008             asurf->probe_radius, radius);
01009         VaccSurf_dtor2(asurf);
01010         thee->surf[id] = Vacc_atomSurf(thee, atom, thee->
refSphere, radius);
01011         asurf = thee->surf[id];
01012     }
01013
01014     return asurf;
01015 }
01016
01017
01018 VPUBLIC void Vacc_splineAccGradAtomNorm4(Vacc *thee, double center[
VAPBS_DIM],
01019                                         double win, double infrad, Vatom *atom, double *grad) {
01020
01021     int i;
01022     double dist, *apos, arad, sm, sm2, sm3, sm4, sm5, sm6, sm7;
01023     double e, e2, e3, e4, e5, e6, e7;
01024     double b, b2, b3, b4, b5, b6, b7;
01025     double c0, c1, c2, c3, c4, c5, c6, c7;
01026     double denom, mygrad;
01027     double mychi = 1.0;          /* Char. func. value for given atom */
01028
01029     VASSERT(thee != NULL);
01030
01031     /* The grad is zero by default */
01032     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
01033
01034     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
01035        * *** MAGNITUDE OF THE FORCE *** */
01036     apos = Vatom_getPosition(atom);
01037     /* Zero-radius atoms don't contribute */
01038     if (Vatom_getRadius(atom) > 0.0) {
01039
01040         arad = Vatom_getRadius(atom);
01041         arad = arad + infrad;
01042         b = arad - win;
01043         e = arad + win;
01044
01045         e2 = e * e;
01046         e3 = e2 * e;
01047         e4 = e3 * e;
01048         e5 = e4 * e;
01049         e6 = e5 * e;
01050         e7 = e6 * e;
01051         b2 = b * b;
01052         b3 = b2 * b;
01053         b4 = b3 * b;
01054         b5 = b4 * b;
01055         b6 = b5 * b;
01056         b7 = b6 * b;
01057
01058         denom = e7 - 7.0*b*e6 + 21.0*b2*e5 - 35.0*e4*b3
01059             + 35.0*e3*b4 - 21.0*b5*e2 + 7.0*e*b6 - b7;
01060         c0 = b4*(35.0*e3 - 21.0*b*e2 + 7*e*b2 - b3)/denom;
01061         c1 = -140.0*b3*e3/denom;
01062         c2 = 210.0*e2*b2*(e + b)/denom;
01063         c3 = -140.0*e*b*(e2 + 3.0*b*e + b2)/denom;
01064         c4 = 35.0*(e3 + 9.0*b*e2 + 9.0*e*b2 + b3)/denom;
01065         c5 = -84.0*(e2 + 3.0*b*e + b2)/denom;
01066         c6 = 70.0*(e + b)/denom;
01067         c7 = -20.0/denom;
01068
01069         dist = VSQR(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
01070             + VSQR(apos[2]-center[2]));
01071
01072         /* If we're inside an atom, the entire characteristic function
01073            * will be zero and the grad will be zero, so we can stop */
01074         if (dist < (arad - win)) return;
01075         /* Likewise, if we're outside the smoothing window, the characteristic
01076            * function is unity and the grad will be zero, so we can stop */
01077         else if (dist > (arad + win)) return;

```

```

01078      /* Account for floating point error at the border
01079      * NAB: COULDN'T THESE TESTS BE COMBINED AS BELOW
01080      * (Vacc_splineAccAtom)? */
01081      else if ((VABS(dist - (arad - win)) < VSMALL) ||
01082              (VABS(dist - (arad + win)) < VSMALL)) return;
01083      /* If we're inside the smoothing window */
01084      else {
01085          sm = dist;
01086          sm2 = sm * sm;
01087          sm3 = sm2 * sm;
01088          sm4 = sm3 * sm;
01089          sm5 = sm4 * sm;
01090          sm6 = sm5 * sm;
01091          sm7 = sm6 * sm;
01092          mychi = c0 + c1*sm + c2*sm2 + c3*sm3
01093                + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
01094          mygrad = c1 + 2.0*c2*sm + 3.0*c3*sm2 + 4.0*c4*sm3
01095                + 5.0*c5*sm4 + 6.0*c6*sm5 + 7.0*c7*sm6;
01096          if (mychi <= 0.0) {
01097              /* Avoid numerical round off errors */
01098              return;
01099          } else if (mychi > 1.0) {
01100              /* Avoid numerical round off errors */
01101              mychi = 1.0;
01102          }
01103      }
01104      /* Now assemble the grad vector */
01105      VASSERT(mychi > 0.0);
01106      for (i=0; i<VAPBS_DIM; i++)
01107          grad[i] = -(mygrad/mychi)*((center[i] - apos[i])/dist);
01108      }
01109 }
01110
01111 VPUBLIC void Vacc_splineAccGradAtomNorm3(Vacc *thee, double center[
01112     VAPBS_DIM],
01113                                         double win, double infrad, Vatom *atom, double *grad) {
01114
01115     int i;
01116     double dist, *apos, arad, sm, sm2, sm3, sm4, sm5;
01117     double e, e2, e3, e4, e5;
01118     double b, b2, b3, b4, b5;
01119     double c0, c1, c2, c3, c4, c5;
01120     double denom, mygrad;
01121     double mychi = 1.0;          /* Char. func. value for given atom */
01122
01123     VASSERT(thee != NULL);
01124
01125     /* The grad is zero by default */
01126     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
01127
01128     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
01129      * *** MAGNITUDE OF THE FORCE *** */
01130     apos = Vatom_getPosition(atom);
01131     /* Zero-radius atoms don't contribute */
01132     if (Vatom_getRadius(atom) > 0.0) {
01133         arad = Vatom_getRadius(atom);
01134         arad = arad + infrad;
01135         b = arad - win;
01136         e = arad + win;
01137
01138         e2 = e * e;
01139         e3 = e2 * e;
01140         e4 = e3 * e;
01141         e5 = e4 * e;
01142         b2 = b * b;
01143         b3 = b2 * b;
01144         b4 = b3 * b;
01145         b5 = b4 * b;
01146
01147         denom = pow((e - b), 5.0);
01148         c0 = -10.0*e2*b3 + 5.0*e*b4 - b5;
01149         c1 = 30.0*e2*b2;
01150         c2 = -30.0*(e2*b + e*b2);
01151         c3 = 10.0*(e2 + 4.0*e*b + b2);
01152         c4 = -15.0*(e + b);
01153         c5 = 6;
01154         c0 = c0/denom;
01155         c1 = c1/denom;
01156         c2 = c2/denom;
01157         c3 = c3/denom;

```

```

01158         c4 = c4/denom;
01159         c5 = c5/denom;
01160
01161         dist = VSQR(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
01162                   + VSQR(apos[2]-center[2]));
01163
01164         /* If we're inside an atom, the entire characteristic function
01165          * will be zero and the grad will be zero, so we can stop */
01166         if (dist < (arad - win)) return;
01167         /* Likewise, if we're outside the smoothing window, the characteristic
01168          * function is unity and the grad will be zero, so we can stop */
01169         else if (dist > (arad + win)) return;
01170         /* Account for floating point error at the border
01171          * NAB: COULDN'T THESE TESTS BE COMBINED AS BELOW
01172          * (Vacc_splineAccAtom)? */
01173         else if ((VABS(dist - (arad - win)) < VSMALL) ||
01174                  (VABS(dist - (arad + win)) < VSMALL)) return;
01175         /* If we're inside the smoothing window */
01176         else {
01177             sm = dist;
01178             sm2 = sm * sm;
01179             sm3 = sm2 * sm;
01180             sm4 = sm3 * sm;
01181             sm5 = sm4 * sm;
01182             mychi = c0 + c1*sm + c2*sm2 + c3*sm3
01183                   + c4*sm4 + c5*sm5;
01184             mygrad = c1 + 2.0*c2*sm + 3.0*c3*sm2 + 4.0*c4*sm3
01185                   + 5.0*c5*sm4;
01186             if (mychi <= 0.0) {
01187                 /* Avoid numerical round off errors */
01188                 return;
01189             } else if (mychi > 1.0) {
01190                 /* Avoid numerical round off errors */
01191                 mychi = 1.0;
01192             }
01193         }
01194         /* Now assemble the grad vector */
01195         VASSERT(mychi > 0.0);
01196         for (i=0; i<VAPBS_DIM; i++)
01197             grad[i] = -(mygrad/mychi)*((center[i] - apos[i])/dist);
01198     }
01199 }
01200
01201 /* ////////////////////////////////////////////////////////////////////
01202 // Routine:  Vacc_atomdSAV
01203 //
01204 // Purpose:  Calculates the vector valued atomic derivative of volume
01205 //
01206 // Args:     radius  The radius of the solvent probe in Angstroms
01207 //           iatom    Index of the atom in thee->alist
01208 //
01209 // Author:   Jason Wagoner
01210 //           Nathan Baker (original FORTRAN routine from UHBD by Brock Luty)
01211 VPUBLIC void Vacc_atomdSAV(Vacc *thee,
01212                          double srad,
01213                          Vatom *atom,
01214                          double *dSA
01215                          ) {
01216
01217     int ipt, iatom;
01218
01219     double area;
01220     double *tPos, tRad, vec[3];
01221     double dx,dy,dz;
01222     VaccSurf *ref;
01223     dx = 0.0;
01224     dy = 0.0;
01225     dz = 0.0;
01226     /* Get the atom information */
01227     ref = thee->refSphere;
01228     iatom = Vatom_getAtomID(atom);
01229
01230     dSA[0] = 0.0;
01231     dSA[1] = 0.0;
01232     dSA[2] = 0.0;
01233
01234     tPos = Vatom_getPosition(atom);
01235     tRad = Vatom_getRadius(atom);
01236
01237     if (tRad == 0.0) return;
01238
01239

```

```

01240     area = 4.0*VPI*(tRad+srad)*(tRad+srad)/((double)(ref->npts));
01241     for (ipt=0; ipt<ref->npts; ipt++) {
01242         vec[0] = (tRad+srad)*ref->xpts[ipt] + tPos[0];
01243         vec[1] = (tRad+srad)*ref->ypts[ipt] + tPos[1];
01244         vec[2] = (tRad+srad)*ref->zpts[ipt] + tPos[2];
01245         if (ivdwAccExclus(thee, vec, srad, iatom)) {
01246             dx = dx+vec[0]-tPos[0];
01247             dy = dy+vec[1]-tPos[1];
01248             dz = dz+vec[2]-tPos[2];
01249         }
01250     }
01251
01252     if ((tRad+srad) != 0){
01253         dSA[0] = dx*area/(tRad+srad);
01254         dSA[1] = dy*area/(tRad+srad);
01255         dSA[2] = dz*area/(tRad+srad);
01256     }
01257
01258 }
01259
01260 /* Note: This is purely test code to make certain that the dSASA code is
01261     behaving properly. This function should NEVER be called by anyone
01262     other than an APBS developer at Wash U.
01263 */
01264 VPRIVATE double Vacc_SASAPos(Vacc *thee, double radius) {
01265
01266     int i, natom;
01267     double area;
01268     Vatom *atom;
01269     VaccSurf *asurf;
01270
01271     natom = Valist_getNumberAtoms(thee->alist);
01272
01273     /* Calculate the area */
01274     area = 0.0;
01275     for (i=0; i<natom; i++) {
01276         atom = Valist_getAtom(thee->alist, i);
01277         asurf = thee->surf[i];
01278
01279         VaccSurf_dtor2(asurf);
01280         thee->surf[i] = Vacc_atomSurf(thee, atom, thee->
refSphere, radius);
01281         asurf = thee->surf[i];
01282         area += (asurf->area);
01283     }
01284
01285     return area;
01286
01287 }
01288
01289 VPRIVATE double Vacc_atomSASAPos(Vacc *thee,
01290     double radius,
01291     Vatom *atom, /* The atom being manipulated */
01292     int mode
01293 ) {
01294
01295     VaccSurf *asurf;
01296     int id;
01297     static int warned = 0;
01298
01299     if ((thee->surf == VNULL) || (mode == 1)){
01300         if(!warned){
01301             Vnm_print(2, "WARNING: Recalculating entire surface!!!!\n");
01302             warned = 1;
01303         }
01304         Vacc_SASAPos(thee, radius); // reinitialize before we can do anything about doing a calculation on
a repositioned atom
01305     }
01306
01307     id = Vatom_getAtomID(atom);
01308     asurf = thee->surf[id];
01309
01310     VaccSurf_dtor(&asurf);
01311     thee->surf[id] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
01312     asurf = thee->surf[id];
01313
01314     //printf("%s: Time elapsed: %f\n", __func__, ((double)clock() - ts) / CLOCKS_PER_SEC);
01315
01316     return asurf->area;
01317 }
01318

```

```

01319 /* //////////////////////////////////////
01320 // Routine: Vacc_atomdSASA
01321 //
01322 // Purpose: Calculates the derivative of surface area with respect to atomic
01323 //           displacement using finite difference methods.
01324 //
01325 // Args:    radius The radius of the solvent probe in Angstroms
01326 //           iatom   Index of the atom in thee->alist
01327 //
01328 // Author:   Jason Wagoner
01329 //           David Gohara
01330 //           Nathan Baker (original FORTRAN routine from UHBD by Brock Luty)
01332 VPUBLIC void Vacc_atomdSASA(Vacc *thee,
01333                             double dpos,
01334                             double srad,
01335                             Vatom *atom,
01336                             double *dSA
01337                             ) {
01338
01339     double *temp_Pos,
01340            tPos[3],
01341            axbl,
01342            axtl,
01343            aybl,
01344            aytl,
01345            azbl,
01346            aztl;
01347     VaccSurf *ref;
01348
01349     //printf("%s: entering\n", __func__);
01350     time_t ts;
01351     ts = clock();
01352
01353     /* Get the atom information */
01354     ref = thee->refSphere;
01355     temp_Pos = Vatom_getPosition(atom); // Get a pointer to the position object. You
    actually manipulate the atom doing this...
01356
01357     tPos[0] = temp_Pos[0];
01358     tPos[1] = temp_Pos[1];
01359     tPos[2] = temp_Pos[2];
01360
01361     /* Shift by pos +/- on x */
01362     temp_Pos[0] -= dpos;
01363     axbl = Vacc_atomSASAPos(thee, srad, atom, 0);
01364     temp_Pos[0] = tPos[0];
01365
01366     temp_Pos[0] += dpos;
01367     axtl = Vacc_atomSASAPos(thee, srad, atom, 0);
01368     temp_Pos[0] = tPos[0];
01369
01370     /* Shift by pos +/- on y */
01371     temp_Pos[1] -= dpos;
01372     aybl = Vacc_atomSASAPos(thee, srad, atom, 0);
01373     temp_Pos[1] = tPos[1];
01374
01375     temp_Pos[1] += dpos;
01376     aytl = Vacc_atomSASAPos(thee, srad, atom, 0);
01377     temp_Pos[1] = tPos[1];
01378
01379     /* Shift by pos +/- on z */
01380     temp_Pos[2] -= dpos;
01381     azbl = Vacc_atomSASAPos(thee, srad, atom, 0);
01382     temp_Pos[2] = tPos[2];
01383
01384     temp_Pos[2] += dpos;
01385     aztl = Vacc_atomSASAPos(thee, srad, atom, 0);
01386     temp_Pos[2] = tPos[2];
01387
01388     /* Reset the atom SASA to zero displacement */
01389     Vacc_atomSASAPos(thee, srad, atom, 0);
01390
01391     /* Calculate the final value */
01392     dSA[0] = (axtl-axbl)/(2.0 * dpos);
01393     dSA[1] = (aytl-aybl)/(2.0 * dpos);
01394     dSA[2] = (aztl-azbl)/(2.0 * dpos);
01395 }
01396
01397 /* Note: This is purely test code to make certain that the dSASA code is
01398          behaving properly. This function should NEVER be called by anyone
01399          other than an APBS developer at Wash U.

```

```

01400 */
01401 VPUBLIC void Vacc_totalAtomdSASA(Vacc *thee, double dpos, double
    srad, Vatom *atom, double *dSA) {
01402
01403     int iatom;
01404     double *temp_Pos, tRad;
01405     double tPos[3];
01406     double axbl,axtl,aybl,aytl,azbl,aztl;
01407     VaccSurf *ref;
01408
01409     /* Get the atom information */
01410     ref = thee->refSphere;
01411     temp_Pos = Vatom_getPosition(atom);
01412     tRad = Vatom_getRadius(atom);
01413     iatom = Vatom_getAtomID(atom);
01414
01415     dSA[0] = 0.0;
01416     dSA[1] = 0.0;
01417     dSA[2] = 0.0;
01418
01419     tPos[0] = temp_Pos[0];
01420     tPos[1] = temp_Pos[1];
01421     tPos[2] = temp_Pos[2];
01422
01423     /* Shift by pos +/- on x */
01424     temp_Pos[0] -= dpos;
01425     axbl = Vacc_atomSASAPos(thee, srad, atom, 1);
01426     temp_Pos[0] = tPos[0];
01427
01428     temp_Pos[0] += dpos;
01429     axtl = Vacc_atomSASAPos(thee, srad, atom, 1);
01430     temp_Pos[0] = tPos[0];
01431
01432     /* Shift by pos +/- on y */
01433     temp_Pos[1] -= dpos;
01434     aybl = Vacc_atomSASAPos(thee, srad, atom, 1);
01435     temp_Pos[1] = tPos[1];
01436
01437     temp_Pos[1] += dpos;
01438     aytl = Vacc_atomSASAPos(thee, srad, atom, 1);
01439     temp_Pos[1] = tPos[1];
01440
01441     /* Shift by pos +/- on z */
01442     temp_Pos[2] -= dpos;
01443     azbl = Vacc_atomSASAPos(thee, srad, atom, 1);
01444     temp_Pos[2] = tPos[2];
01445
01446     temp_Pos[2] += dpos;
01447     aztl = Vacc_atomSASAPos(thee, srad, atom, 1);
01448     temp_Pos[2] = tPos[2];
01449
01450     /* Calculate the final value */
01451     dSA[0] = (axtl-axbl)/(2.0 * dpos);
01452     dSA[1] = (aytl-aybl)/(2.0 * dpos);
01453     dSA[2] = (aztl-azbl)/(2.0 * dpos);
01454 }
01455
01456 /* Note: This is purely test code to make certain that the dSASA code is
01457     behaving properly. This function should NEVER be called by anyone
01458     other than an APBS developer at Wash U.
01459 */
01460 VPUBLIC void Vacc_totalAtomdSAV(Vacc *thee, double dpos, double
    srad, Vatom *atom, double *dSA, Vclist *clist) {
01461
01462     int iatom;
01463     double *temp_Pos, tRad;
01464     double tPos[3];
01465     double axbl,axtl,aybl,aytl,azbl,aztl;
01466     VaccSurf *ref;
01467
01468     /* Get the atom information */
01469     ref = thee->refSphere;
01470     temp_Pos = Vatom_getPosition(atom);
01471     tRad = Vatom_getRadius(atom);
01472     iatom = Vatom_getAtomID(atom);
01473
01474     dSA[0] = 0.0;
01475     dSA[1] = 0.0;
01476     dSA[2] = 0.0;
01477
01478     tPos[0] = temp_Pos[0];

```

```

01479     tPos[1] = temp_Pos[1];
01480     tPos[2] = temp_Pos[2];
01481
01482     /* Shift by pos +/- on x */
01483     temp_Pos[0] -= dpos;
01484     axb1 = Vacc_totalSAV(thee,clist, VNULL, srاد);
01485     temp_Pos[0] = tPos[0];
01486
01487     temp_Pos[0] += dpos;
01488     axt1 = Vacc_totalSAV(thee,clist, VNULL, srاد);
01489     temp_Pos[0] = tPos[0];
01490
01491     /* Shift by pos +/- on y */
01492     temp_Pos[1] -= dpos;
01493     ayb1 = Vacc_totalSAV(thee,clist, VNULL, srاد);
01494     temp_Pos[1] = tPos[1];
01495
01496     temp_Pos[1] += dpos;
01497     ayt1 = Vacc_totalSAV(thee,clist, VNULL, srاد);
01498     temp_Pos[1] = tPos[1];
01499
01500     /* Shift by pos +/- on z */
01501     temp_Pos[2] -= dpos;
01502     azb1 = Vacc_totalSAV(thee,clist, VNULL, srاد);
01503     temp_Pos[2] = tPos[2];
01504
01505     temp_Pos[2] += dpos;
01506     azt1 = Vacc_totalSAV(thee,clist, VNULL, srاد);
01507     temp_Pos[2] = tPos[2];
01508
01509     /* Calculate the final value */
01510     dSA[0] = (axt1-axb1)/(2.0 * dpos);
01511     dSA[1] = (ayt1-ayb1)/(2.0 * dpos);
01512     dSA[2] = (azt1-azb1)/(2.0 * dpos);
01513 }
01514
01515 VPUBLIC double Vacc_totalSAV(Vacc *thee, Vclist *clist,
APOLparm *apolparm, double radius) {
01516
01517     int i;
01518     int npts[3];
01519
01520     double spacs[3], vec[3];
01521     double w, wx, wy, wz, len, fn, x, y, z, vol;
01522     double vol_density, sav;
01523     double *lower_corner, *upper_corner;
01524
01525     sav = 0.0;
01526     vol = 1.0;
01527     vol_density = 2.0;
01528
01529     lower_corner = clist->lower_corner;
01530     upper_corner = clist->upper_corner;
01531
01532     for (i=0; i<3; i++) {
01533         len = upper_corner[i] - lower_corner[i];
01534         vol *= len;
01535         fn = len*vol_density + 1;
01536         npts[i] = (int)ceil(fn);
01537         spacs[i] = len/((double)(npts[i])-1.0);
01538         if (apolparm != VNULL) {
01539             if (apolparm->setgrid) {
01540                 if (apolparm->grid[i] > spacs[i]) {
01541                     Vnm_print(2, "Vacc_totalSAV: Warning, your GRID value (%g) is larger than the
recommended value (%g)!\n",
01542                             apolparm->grid[i], spacs[i]);
01543                 }
01544                 spacs[i] = apolparm->grid[i];
01545             }
01546         }
01547     }
01548 }
01549
01550     for (x=lower_corner[0]; x<=upper_corner[0]; x=x+spacs[0]) {
01551         if ( VABS(x - lower_corner[0]) < VSMALL) {
01552             wx = 0.5;
01553         } else if ( VABS(x - upper_corner[0]) < VSMALL) {
01554             wx = 0.5;
01555         } else {
01556             wx = 1.0;
01557         }

```

```

01558     vec[0] = x;
01559     for (y=lower_corner[1]; y<=upper_corner[1]; y=y+spacs[1]) {
01560         if ( VABS(y - lower_corner[1]) < VSMALL) {
01561             wy = 0.5;
01562         } else if ( VABS(y - upper_corner[1]) < VSMALL) {
01563             wy = 0.5;
01564         } else {
01565             wy = 1.0;
01566         }
01567         vec[1] = y;
01568         for (z=lower_corner[2]; z<=upper_corner[2]; z=z+spacs[2]) {
01569             if ( VABS(z - lower_corner[2]) < VSMALL) {
01570                 wz = 0.5;
01571             } else if ( VABS(z - upper_corner[2]) < VSMALL) {
01572                 wz = 0.5;
01573             } else {
01574                 wz = 1.0;
01575             }
01576             vec[2] = z;
01577
01578             w = wx*wy*wz;
01579
01580             sav += (w*(1.0-Vacc_ivdwAcc(thee, vec, radius)));
01581
01582             } /* z loop */
01583         } /* y loop */
01584     } /* x loop */
01585
01586     w = spacs[0]*spacs[1]*spacs[2];
01587     sav *= w;
01588
01589     return sav;
01590 }
01591
01592 int Vacc_wcaEnergyAtom(Vacc *thee, APOLparm *apolparm,
01593     Valist *alist,
01594                                     Vclist *clist, int iatom, double *value) {
01595
01596     int i;
01597     int npts[3];
01598     int pad = 14;
01599
01600     int xmin, ymin, zmin;
01601     int xmax, ymax, zmax;
01602
01603     double sigma6, sigma12;
01604
01605     double spacs[3], vec[3];
01606     double w, wx, wy, wz, len, fn, x, y, z, vol;
01607     double x2,y2,z2,r;
01608     double vol_density, energy, rho, srad;
01609     double psig, epsilon, watepsilon, sigma, watsigma, eni, chi;
01610
01611     double *pos;
01612     double *lower_corner, *upper_corner;
01613
01614     Vatom *atom = VNULL;
01615     VASSERT(apolparm != VNULL);
01616
01617     energy = 0.0;
01618     vol = 1.0;
01619     vol_density = 2.0;
01620
01621     lower_corner = clist->lower_corner;
01622     upper_corner = clist->upper_corner;
01623
01624     atom = Valist_getAtom(alist, iatom);
01625     pos = Vatom_getPosition(atom);
01626
01627     /* Note: these are the original temporary water parameters... they have been
01628        replaced by entries in a parameter file:
01629     watsigma = 1.7683;
01630     watepsilon = 0.1521;
01631     watepsilon = watepsilon*4.184;
01632     */
01633
01634     srad = apolparm->srad;
01635     rho = apolparm->bconc;
01636     watsigma = apolparm->watsigma;
01637     watepsilon = apolparm->watepsilon;
01638     psig = atom->radius;

```



```

01638     epsilon = atom->epsilon;
01639     sigma = psig + watsigma;
01640     epsilon = VSQRT((epsilon * watepsilon));
01641
01642     /* parameters */
01643     sigma6 = VPOW(sigma,6);
01644     sigma12 = VPOW(sigma,12);
01645     /* OPLS-style radius: double sigmar = sigma*VPOW(2, (1.0/6.0)); */
01646
01647     xmin = pos[0] - pad;
01648     xmax = pos[0] + pad;
01649     ymin = pos[1] - pad;
01650     ymax = pos[1] + pad;
01651     zmin = pos[2] - pad;
01652     zmax = pos[2] + pad;
01653
01654     for (i=0; i<3; i++) {
01655         len = (upper_corner[i] + pad) - (lower_corner[i] - pad);
01656         vol *= len;
01657         fn = len*vol_density + 1;
01658         npts[i] = (int)ceil(fn);
01659         spacs[i] = 0.5;
01660         if (apolparm->setgrid) {
01661             if (apolparm->grid[i] > spacs[i]) {
01662                 Vnm_print(2, "Vacc_totalSAV: Warning, your GRID value (%g) is larger than the recommended
value (%g)!\n",
01663                     apolparm->grid[i], spacs[i]);
01664             }
01665             spacs[i] = apolparm->grid[i];
01666         }
01667     }
01668
01669     for (x=xmin; x<=xmax; x=x+spacs[0]) {
01670         if ( VABS(x - xmin) < VSMALL) {
01671             wx = 0.5;
01672         } else if ( VABS(x - xmax) < VSMALL) {
01673             wx = 0.5;
01674         } else {
01675             wx = 1.0;
01676         }
01677         vec[0] = x;
01678         for (y=ymin; y<=ymax; y=y+spacs[1]) {
01679             if ( VABS(y - ymin) < VSMALL) {
01680                 wy = 0.5;
01681             } else if ( VABS(y - ymax) < VSMALL) {
01682                 wy = 0.5;
01683             } else {
01684                 wy = 1.0;
01685             }
01686             vec[1] = y;
01687             for (z=zmin; z<=zmax; z=z+spacs[2]) {
01688                 if ( VABS(z - zmin) < VSMALL) {
01689                     wz = 0.5;
01690                 } else if ( VABS(z - zmax) < VSMALL) {
01691                     wz = 0.5;
01692                 } else {
01693                     wz = 1.0;
01694                 }
01695                 vec[2] = z;
01696
01697                 w = wx*wy*wz;
01698
01699                 chi = Vacc_ivdwAcc(thee, vec, srad);
01700
01701                 if (VABS(chi) > VSMALL) {
01702
01703                     x2 = VSQR(vec[0]-pos[0]);
01704                     y2 = VSQR(vec[1]-pos[1]);
01705                     z2 = VSQR(vec[2]-pos[2]);
01706                     r = VSQRT(x2+y2+z2);
01707
01708                     if (r <= 14 && r >= sigma) {
01709                         eni = chi*rho*epsilon*(-2.0*sigma6/VPOW(r,6)+sigma12/VPOW(r,12));
01710                     }else if (r <= 14){
01711                         eni = -1.0*epsilon*chi*rho;
01712                     }else{
01713                         eni = 0.0;
01714                     }
01715                 }else{
01716                     eni = 0.0;
01717                 }

```

```

01718
01719         energy += eni*w;
01720
01721         } /* z loop */
01722     } /* y loop */
01723 } /* x loop */
01724
01725 w = spacs[0]*spacs[1]*spacs[2];
01726 energy *= w;
01727
01728 *value = energy;
01729
01730 return VRC_SUCCESS;
01731 }
01732
01733 VPUBLIC int Vacc_wcaEnergy(Vacc *acc, APOLparm *apolparm,
01734     Valist *alist,
01735     Vclist *clist){
01736
01737     int iatom;
01738     int rc = 0;
01739
01740     double energy = 0.0;
01741     double tenergy = 0.0;
01742     double rho = apolparm->bconc;
01743
01744     /* Do a sanity check to make sure that watepsilon and watsigma are set
01745      * If not, return with an error. */
01746     if (apolparm->setwat == 0){
01747         Vnm_print(2,"Vacc_wcaEnergy: Error. No value was set for watsigma and watepsilon.\n");
01748         return VRC_FAILURE;
01749     }
01750
01751     if (VABS(rho) < VSMALL) {
01752         apolparm->wcaEnergy = tenergy;
01753         return 1;
01754     }
01755
01756     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++){
01757         rc = Vacc_wcaEnergyAtom(acc, apolparm, alist, clist, iatom, &energy);
01758         if(rc == 0) return 0;
01759
01760         tenergy += energy;
01761     }
01762
01763     apolparm->wcaEnergy = tenergy;
01764
01765     return VRC_SUCCESS;
01766 }
01767
01768 VPUBLIC int Vacc_wcaForceAtom(Vacc *thee,
01769     APOLparm *apolparm,
01770     Vclist *clist,
01771     Vatom *atom,
01772     double *force
01773 ){
01774     int i,
01775         si,
01776         npts[3],
01777         pad = 14,
01778         xmin,
01779         ymin,
01780         zmin,
01781         xmax,
01782         ymax,
01783         zmax;
01784
01785     double sigma6,
01786         sigma12,
01787         spacs[3],
01788         vec[3],
01789         fpt[3],
01790         w,
01791         wx,
01792         wy,
01793         wz,
01794         len,
01795         fn,
01796         x,
01797         y,

```

```

01798         z,
01799         vol,
01800         x2,
01801         y2,
01802         z2,
01803         r,
01804         vol_density,
01805         fo,
01806         rho,
01807         srad,
01808         psig,
01809         epsilon,
01810         watepsilon,
01811         sigma,
01812         watsigma,
01813         chi,
01814         *pos,
01815         *lower_corner,
01816         *upper_corner;
01817
01818     /* Allocate needed variables now that we've asserted required conditions. */
01819     time_t ts;
01820     ts = clock();
01821
01822     VASSERT(apolparm != VNULL);
01823
01824     /* Do a sanity check to make sure that watepsilon and watsigma are set
01825      * If not, return with an error. */
01826     if(apolparm->setwat == 0){
01827         Vnm_print(2, "Vacc_wcaEnergy: Error. No value was set for watsigma and watepsilon.\n");
01828         return VRC_FAILURE;
01829     }
01830
01831     vol = 1.0;
01832     vol_density = 2.0;
01833
01834     lower_corner = clist->lower_corner;
01835     upper_corner = clist->upper_corner;
01836
01837     pos = Vatom_getPosition(atom);
01838
01839     srad = apolparm->srad;
01840     rho = apolparm->bconc;
01841     watsigma = apolparm->watsigma;
01842     watepsilon = apolparm->watepsilon;
01843
01844     psig = atom->radius;
01845     epsilon = atom->epsilon;
01846     sigma = psig + watsigma;
01847     epsilon = VSQRT((epsilon * watepsilon));
01848
01849     /* parameters */
01850     sigma6 = VPOW(sigma, 6);
01851     sigma12 = VPOW(sigma, 12);
01852     /* OPLS-style radius: double sigmar = sigma*VPOW(2, (1.0/6.0)); */
01853
01854     for (i=0; i<3; i++) {
01855         len = (upper_corner[i] + pad) - (lower_corner[i] - pad);
01856         vol *= len;
01857         fn = len*vol_density + 1;
01858         npts[i] = (int)ceil(fn);
01859         spacs[i] = 0.5;
01860         force[i] = 0.0;
01861         if (apolparm->setgrid) {
01862             if (apolparm->grid[i] > spacs[i]) {
01863                 Vnm_print(2, "Vacc_totalSAV: Warning, your GRID value (%g) is larger than the recommended
value (%g)!\n",
01864                     apolparm->grid[i], spacs[i]);
01865             }
01866             spacs[i] = apolparm->grid[i];
01867         }
01868     }
01869
01870     xmin = pos[0] - pad;
01871     xmax = pos[0] + pad;
01872     ymin = pos[1] - pad;
01873     ymax = pos[1] + pad;
01874     zmin = pos[2] - pad;
01875     zmax = pos[2] + pad;
01876
01877     for (x=xmin; x<=xmax; x=x+spacs[0]) {

```

```

01878         if ( VABS(x - xmin) < VSMALL) {
01879             wx = 0.5;
01880         } else if ( VABS(x - xmax) < VSMALL) {
01881             wx = 0.5;
01882         } else {
01883             wx = 1.0;
01884         }
01885         vec[0] = x;
01886         for (y=ymin; y<=ymax; y=y+spacs[1]) {
01887             if ( VABS(y - ymin) < VSMALL) {
01888                 wy = 0.5;
01889             } else if ( VABS(y - ymax) < VSMALL) {
01890                 wy = 0.5;
01891             } else {
01892                 wy = 1.0;
01893             }
01894             vec[1] = y;
01895             for (z=zmin; z<=zmax; z=z+spacs[2]) {
01896                 if ( VABS(z - zmin) < VSMALL) {
01897                     wz = 0.5;
01898                 } else if ( VABS(z - zmax) < VSMALL) {
01899                     wz = 0.5;
01900                 } else {
01901                     wz = 1.0;
01902                 }
01903                 vec[2] = z;
01904
01905                 w = wx*wy*wz;
01906
01907                 chi = Vacc_ivdwAcc(thee, vec, srاد);
01908
01909                 if (chi != 0.0) {
01910
01911                     x2 = VSQR(vec[0]-pos[0]);
01912                     y2 = VSQR(vec[1]-pos[1]);
01913                     z2 = VSQR(vec[2]-pos[2]);
01914                     r = VSQRT(x2+y2+z2);
01915
01916                     if (r <= 14 && r >= sigma){
01917
01918                         fo = 12.0*chi*rho*epsilon*(sigma6/VPOW(r,7)-sigma12/VPOW(r,13));
01919
01920                         fpt[0] = -1.0*(pos[0]-vec[0])*fo/r;
01921                         fpt[1] = -1.0*(pos[1]-vec[1])*fo/r;
01922                         fpt[2] = -1.0*(pos[2]-vec[2])*fo/r;
01923
01924                     }else {
01925                         for (si=0; si < 3; si++) fpt[si] = 0.0;
01926                     }
01927                 }else {
01928                     for (si=0; si < 3; si++) fpt[si] = 0.0;
01929                 }
01930
01931                 for(i=0;i<3;i++){
01932                     force[i] += (w*fpt[i]);
01933                 }
01934
01935             } /* z loop */
01936         } /* y loop */
01937     } /* x loop */
01938
01939     w = spacs[0]*spacs[1]*spacs[2];
01940     for(i=0;i<3;i++) force[i] *= w;
01941
01942     return VRC_SUCCESS;
01943 }
01944

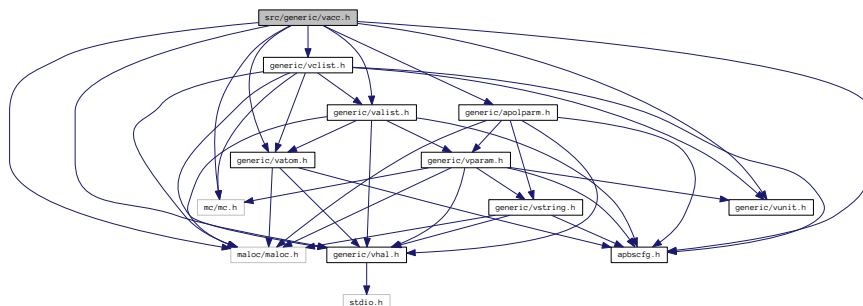
```

10.41 src/generic/vacc.h File Reference

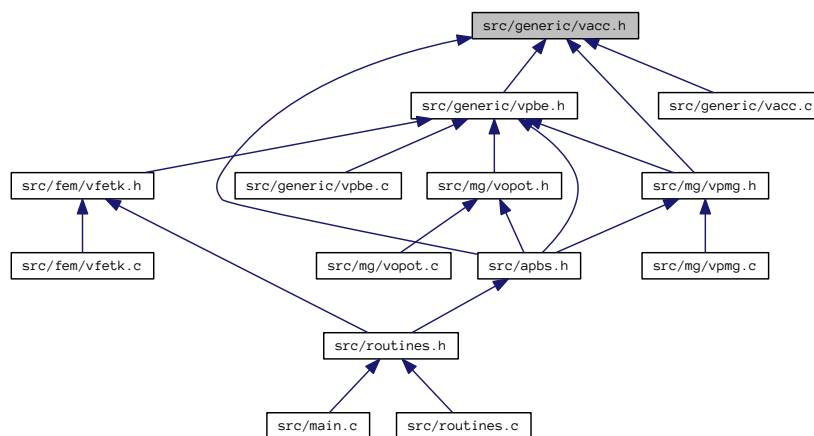
Contains declarations for class Vacc.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "mc/mc.h"
#include "generic/vhal.h"
#include "generic/valist.h"
#include "generic/vclist.h"
#include "generic/vatom.h"
#include "generic/vunit.h"
#include "generic/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)
FORTTRAN stub to construct the accessibility object.
- VEXTERNC void [Vacc_dtor](#) ([Vacc](#) **thee)
Destroy object.
- VEXTERNC void [Vacc_dtor2](#) ([Vacc](#) *thee)
FORTTRAN stub to destroy object.
- VEXTERNC double [Vacc_vdwAcc](#) ([Vacc](#) *thee, double center[VAPBS_DIM])
Report van der Waals accessibility.
- VEXTERNC double [Vacc_ivdwAcc](#) ([Vacc](#) *thee, double center[VAPBS_DIM], double radius)
Report inflated van der Waals accessibility.
- VEXTERNC double [Vacc_molAcc](#) ([Vacc](#) *thee, double center[VAPBS_DIM], double radius)
Report molecular accessibility.
- VEXTERNC double [Vacc_fastMolAcc](#) ([Vacc](#) *thee, double center[VAPBS_DIM], double radius)
Report molecular accessibility quickly.
- VEXTERNC double [Vacc_splineAcc](#) ([Vacc](#) *thee, double center[VAPBS_DIM], double win, double infrad)
Report spline-based accessibility.
- VEXTERNC void [Vacc_splineAccGrad](#) ([Vacc](#) *thee, double center[VAPBS_DIM], double win, double infrad, double *grad)
Report gradient of spline-based accessibility.
- VEXTERNC double [Vacc_splineAccAtom](#) ([Vacc](#) *thee, double center[VAPBS_DIM], double win, double infrad, [Vatom](#) *atom)
Report spline-based accessibility for a given atom.

- VEXTERNC void `Vacc_splineAccGradAtomUnnorm` (`Vacc *thee`, double center[`VAPBS_DIM`], double win, double infrad, `Vatom *atom`, double *force)
Report gradient of spline-based accessibility with respect to a particular atom (see `Vpmg_splineAccAtom`)
- VEXTERNC void `Vacc_splineAccGradAtomNorm` (`Vacc *thee`, double center[`VAPBS_DIM`], double win, double infrad, `Vatom *atom`, double *force)
Report gradient of spline-based accessibility with respect to a particular atom normalized by the accessibility value due to that atom at that point (see `Vpmg_splineAccAtom`)
- VEXTERNC void `Vacc_splineAccGradAtomNorm4` (`Vacc *thee`, double center[`VAPBS_DIM`], double win, double infrad, `Vatom *atom`, double *force)
Report gradient of spline-based accessibility with respect to a particular atom normalized by a 4th order accessibility value due to that atom at that point (see `Vpmg_splineAccAtom`)
- VEXTERNC void `Vacc_splineAccGradAtomNorm3` (`Vacc *thee`, double center[`VAPBS_DIM`], double win, double infrad, `Vatom *atom`, double *force)
Report gradient of spline-based accessibility with respect to a particular atom normalized by a 3rd order accessibility value due to that atom at that point (see `Vpmg_splineAccAtom`)
- VEXTERNC double `Vacc_SASA` (`Vacc *thee`, double radius)
Build the solvent accessible surface (SAS) and calculate the solvent accessible surface area.
- VEXTERNC double `Vacc_totalSASA` (`Vacc *thee`, double radius)
Return the total solvent accessible surface area (SASA)
- VEXTERNC double `Vacc_atomSASA` (`Vacc *thee`, double radius, `Vatom *atom`)
Return the atomic solvent accessible surface area (SASA)
- VEXTERNC `VaccSurf * Vacc_atomSASPoints` (`Vacc *thee`, double radius, `Vatom *atom`)
Get the set of points for this atom's solvent-accessible surface.
- VEXTERNC void `Vacc_atomdSAV` (`Vacc *thee`, double radius, `Vatom *atom`, double *dSA)
Get the derivative of solvent accessible volume.
- VEXTERNC void `Vacc_atomdSASA` (`Vacc *thee`, double dpos, double radius, `Vatom *atom`, double *dSA)
Get the derivative of solvent accessible area.
- VEXTERNC void `Vacc_totalAtomdSASA` (`Vacc *thee`, double dpos, double radius, `Vatom *atom`, double *dSA)
Testing purposes only.
- VEXTERNC void `Vacc_totalAtomdSAV` (`Vacc *thee`, double dpos, double radius, `Vatom *atom`, double *dSA, `Vclist *clist`)
Total solvent accessible volume.
- VEXTERNC double `Vacc_totalSAV` (`Vacc *thee`, `Vclist *clist`, `APOLparm *apolparm`, double radius)
Return the total solvent accessible volume (SAV)
- VEXTERNC int `Vacc_wcaEnergy` (`Vacc *thee`, `APOLparm *apolparm`, `Valist *alist`, `Vclist *clist`)
Return the WCA integral energy.
- VEXTERNC int `Vacc_wcaForceAtom` (`Vacc *thee`, `APOLparm *apolparm`, `Vclist *clist`, `Vatom *atom`, double *force)
Return the WCA integral force.
- VEXTERNC int `Vacc_wcaEnergyAtom` (`Vacc *thee`, `APOLparm *apolparm`, `Valist *alist`, `Vclist *clist`, int iatom, double *value)
Calculate the WCA energy for an atom.

10.41.1 Detailed Description

Contains declarations for class `Vacc`.

Version`Id`**Author**

Nathan A. Baker

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*   Nathan A. Baker (nathan.baker@pnnl.gov)
*   Pacific Northwest National Laboratory
*
*   Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vacc.h](#).**10.42 vacc.h**

```

00001
00062 #ifndef _VACC_H_

```



```

00063 #define _VACC_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "malloc/malloc.h"
00068 #if defined(HAVE_MC_H)
00069 #include "mc/mc.h"
00070 #endif
00071
00072 #include "generic/vhal.h"
00073 #include "generic/valist.h"
00074 #include "generic/vclist.h"
00075 #include "generic/vatom.h"
00076 #include "generic/vunit.h"
00077 #include "generic/apolparm.h"
00078
00084 struct sVaccSurf {
00085     Vmem *mem;
00086     double *xpts;
00087     double *ypts;
00088     double *zpts;
00089     char *bpts;
00091     double area;
00092     int npts;
00093     double probe_radius;
00095 };
00096
00101 typedef struct sVaccSurf VaccSurf;
00102
00108 struct sVacc {
00109
00110     Vmem *mem;
00111     Valist *alist;
00112     Vclist *clist;
00113     int *atomFlags;
00116     VaccSurf *refSphere;
00117     VaccSurf **surf;
00120     Vset acc;
00122     double surf_density;
00125 };
00126
00131 typedef struct sVacc Vacc;
00132
00133 #if !defined(VINLINE_VACC)
00134
00140     VEXTERNC unsigned long int Vacc_memChk(
00141         Vacc *thee
00142     );
00143
00144 #else /* if defined(VINLINE_VACC) */
00145
00146 #    define Vacc_memChk(thee) (Vmem_bytes((thee)->mem))
00147
00148 #endif /* if !defined(VINLINE_VACC) */
00149
00157 VEXTERNC VaccSurf* VaccSurf_ctor(
00158     Vmem *mem,
00159     double probe_radius,
00160     int nsphere
00161 );
00162
00170 VEXTERNC int VaccSurf_ctor2(
00171     VaccSurf *thee,
00172     Vmem *mem,
00173     double probe_radius,
00174     int nsphere
00175 );
00176
00182 VEXTERNC void VaccSurf_dtor(
00183     VaccSurf **thee
00184 );
00185
00191 VEXTERNC void VaccSurf_dtor2(
00192     VaccSurf *thee
00193 );
00194
00209 VEXTERNC VaccSurf* VaccSurf_refSphere(
00210     Vmem *mem,
00211     int npts
00212 );
00213

```

```
00221 VEXTERNC VaccSurf* Vacc_atomSurf(
00222     Vacc *thee,
00223     Vatom *atom,
00224     VaccSurf *ref,
00226     double probe_radius
00227 );
00228
00233 VEXTERNC Vacc* Vacc_ctor(
00234     Valist *alist,
00235     Vclist *clist,
00237     double surf_density
00239 );
00240
00245 VEXTERNC int Vacc_ctor2(
00246     Vacc *thee,
00247     Valist *alist,
00248     Vclist *clist,
00250     double surf_density
00252 );
00253
00258 VEXTERNC void Vacc_dtor(
00259     Vacc **thee
00260 );
00261
00266 VEXTERNC void Vacc_dtor2(
00267     Vacc *thee
00268 );
00269
00280 VEXTERNC double Vacc_vdwAcc(
00281     Vacc *thee,
00282     double center[VAPBS_DIM]
00283 );
00284
00296 VEXTERNC double Vacc_ivdwAcc(
00297     Vacc *thee,
00298     double center[VAPBS_DIM],
00299     double radius
00300 );
00301
00316 VEXTERNC double Vacc_molAcc(
00317     Vacc *thee,
00318     double center[VAPBS_DIM],
00319     double radius
00320 );
00321
00340 VEXTERNC double Vacc_fastMolAcc(
00341     Vacc *thee,
00342     double center[VAPBS_DIM],
00343     double radius
00344 );
00345
00357 VEXTERNC double Vacc_splineAcc(
00358     Vacc *thee,
00359     double center[VAPBS_DIM],
00360     double win,
00361     double infrad
00362 );
00363
00369 VEXTERNC void Vacc_splineAccGrad(
00370     Vacc *thee,
00371     double center[VAPBS_DIM],
00372     double win,
00373     double infrad,
00374     double *grad
00375 );
00376
00388 VEXTERNC double Vacc_splineAccAtom(
00389     Vacc *thee,
00390     double center[VAPBS_DIM],
00391     double win,
00392     double infrad,
00393     Vatom *atom
00394 );
00395
00406 VEXTERNC void Vacc_splineAccGradAtomUnnorm(
00407     Vacc *thee,
00408     double center[VAPBS_DIM],
00409     double win,
00410     double infrad,
00411     Vatom *atom,
00412     double *force
```

```
00413     );
00414
00426 VEXTERNC void Vacc_splineAccGradAtomNorm(
00427     Vacc *thee,
00428     double center[VAPBS_DIM],
00429     double win,
00430     double infrad,
00431     Vatom *atom,
00432     double *force
00433 );
00434
00442 VEXTERNC void Vacc_splineAccGradAtomNorm4(
00443     Vacc *thee,
00444     double center[VAPBS_DIM],
00445     double win,
00446     double infrad,
00447     Vatom *atom,
00448     double *force
00449 );
00450
00458 VEXTERNC void Vacc_splineAccGradAtomNorm3(
00459     Vacc *thee,
00460     double center[VAPBS_DIM],
00461     double win,
00462     double infrad,
00463     Vatom *atom,
00464     double *force
00465 );
00466
00467
00477 VEXTERNC double Vacc_SASA(
00478     Vacc *thee,
00479     double radius
00480 );
00481
00489 VEXTERNC double Vacc_totalSASA(
00490     Vacc *thee,
00491     double radius
00492 );
00493
00501 VEXTERNC double Vacc_atomSASA(
00502     Vacc *thee,
00503     double radius,
00504     Vatom *atom
00505 );
00506
00513 VEXTERNC VaccSurf* Vacc_atomSASPoints(
00514     Vacc *thee,
00515     double radius,
00516     Vatom *atom
00517 );
00518
00524 VEXTERNC void Vacc_atomdSAV(
00525     Vacc *thee,
00526     double radius,
00527     Vatom *atom,
00528     double *dSA
00529 );
00530
00536 VEXTERNC void Vacc_atomdSASA(
00537     Vacc *thee,
00538     double dpos,
00539     double radius,
00540     Vatom *atom,
00541     double *dSA
00542 );
00543
00549 VEXTERNC void Vacc_totalAtomdSASA(
00550     Vacc *thee,
00551     double dpos,
00552     double radius,
00553     Vatom *atom,
00554     double *dSA
00555 );
00556
00562 VEXTERNC void Vacc_totalAtomdSAV(
00563     Vacc *thee,
00564     double dpos,
00565     double radius,
00566     Vatom *atom,
00567     double *dSA,
```

```

00568             Vclist *clist
00569         );
00570
00578 VEXTERNC double Vacc_totalsAV(
00579     Vacc *thee,
00580     Vclist *clist,
00581     APOLparm *apolparm,
00582     double radius
00583 );
00584
00585
00592 VEXTERNC int Vacc_wcaEnergy(
00593     Vacc *thee,
00594     APOLparm *apolparm,
00595     Valist *alist,
00596     Vclist *clist
00597 );
00604 VEXTERNC int Vacc_wcaForceAtom(Vacc *thee,
00605     APOLparm *apolparm,
00606     Vclist *clist,
00607     Vatom *atom,
00608     double *force
00609 );
00610
00616 VEXTERNC int Vacc_wcaEnergyAtom(
00617     Vacc *thee,
00618     APOLparm *apolparm,
00619     Valist *alist,
00620     Vclist *clist,
00621     int iatom,
00622     double *value
00623 );
00624
00625 #endif /* ifndef _VACC_H_ */

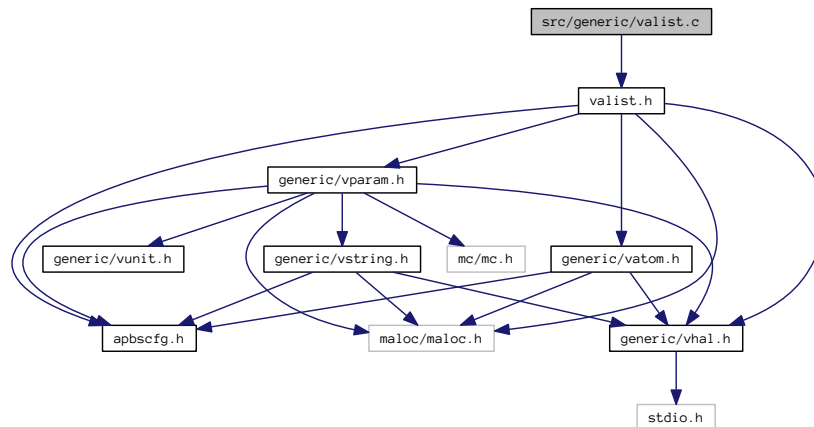
```

10.43 src/generic/valist.c File Reference

Class Valist methods.

```
#include "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)
- FORTTRAN stub to construct the atom list object.*
- VPUBLIC void [Valist_dtor](#) ([Valist](#) **thee)
- Destroys atom list object.*
- VPUBLIC void [Valist_dtor2](#) ([Valist](#) *thee)
- FORTTRAN 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<>"

10.43.1 Detailed Description

Class Valist methods.

Author

Nathan Baker

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [valist.c](#).

10.44 valist.c

```

00001
00056 #include "valist.h"
00057
00058 VEMBED(rcsid="$Id$")
00059
00060 VPRIVATE char *Valist_whiteChars = " \t\r\n";
00061 VPRIVATE char *Valist_commChars = "##";
00062 VPRIVATE char *Valist_xmlwhiteChars = " \t\r\n<>";
00063
00064 #if !defined(VINLINE_VATOM)
00065
00066 VPUBLIC double Valist_getCenterX(Valist *thee) {
00067
00068     if (thee == NULL) {
00069         Vnm_print(2, "Valist_getCenterX: Found null pointer when getting the center of X coordinate!\n");
00070         VASSERT(0);
00071     }
00072     return thee->center[0];
00073
00074 }
00075
00076 VPUBLIC double Valist_getCenterY(Valist *thee) {
00077
00078     if (thee == NULL) {
00079         Vnm_print(2, "Valist_getCenterY: Found null pointer when getting the center of Y coordinate!\n");
00080         VASSERT(0);
00081     }
00082     return thee->center[1];
00083
00084 }
00085 VPUBLIC double Valist_getCenterZ(Valist *thee) {
00086
00087     if (thee == NULL) {
00088         Vnm_print(2, "Valist_getCenterZ: Found null pointer when getting the center of Z coordinate!\n");
00089         VASSERT(0);
00090     }
00091     return thee->center[2];
00092
00093 }
00094
00095 VPUBLIC Vatom* Valist_getAtomList(Valist *thee) {
00096
00097     if (thee == NULL) {
00098         Vnm_print(2, "Valist_getAtomList: Found null pointer when getting the atom list!\n");
00099         VASSERT(0);
00100     }
00101     return thee->atoms;
00102
00103 }
00104
00105 VPUBLIC int Valist_getNumberAtoms(Valist *thee) {
00106
00107     if (thee == NULL) {
00108         Vnm_print(2, "Valist_getNumberAtoms: Found null pointer when getting the number of atoms!\n");
00109         VASSERT(0);
00110     }
00111     return thee->number;
00112
00113 }
00114
00115 VPUBLIC Vatom* Valist_getAtom(Valist *thee, int i) {
00116
00117     if (thee == NULL) {
00118         Vnm_print(2, "Valist_getAtom: Found null pointer when getting atoms!\n");
00119         VASSERT(0);
00120     }
00121     if (i >= thee->number) {
00122         Vnm_print(2, "Valist_getAtom: Requested atom number (%d) outside of atom list range (%d)!\n", i,
thee->number);
00123         VASSERT(0);
00124     }
00125     return &(thee->atoms[i]);
00126
00127 }
00128
00129 VPUBLIC unsigned long int Valist_memChk(Valist *thee) {
00130
00131     if (thee == NULL) return 0;

```

```

00132     return Vmem_bytes(thee->vmem);
00133
00134 }
00135
00136 #endif /* if !defined(VINLINE_VATOM) */
00137
00138 VPUBLIC Valist* Valist_ctor() {
00139
00140     /* Set up the structure */
00141     Valist *thee = VNULL;
00142     thee = (Valist*)Vmem_malloc(VNULL, 1, sizeof(Valist));
00143     if (thee == VNULL) {
00144         Vnm_print(2, "Valist_ctor: Got NULL pointer when constructing the atom list object!\n");
00145         VASSERT(0);
00146     }
00147     if (Valist_ctor2(thee) != VRC_SUCCESS) {
00148         Vnm_print(2, "Valist_ctor: Error in constructing the atom list object!\n");
00149         VASSERT(0);
00150     }
00151
00152     return thee;
00153 }
00154
00155 VPUBLIC Vrc_Codes Valist_ctor2(Valist *thee) {
00156     thee->atoms = VNULL;
00157     thee->number = 0;
00158
00159     /* Initialize the memory management object */
00160     thee->vmem = Vmem_ctor("APBS:VALIST");
00161
00162     return VRC_SUCCESS;
00163 }
00164
00165 }
00166
00167 VPUBLIC void Valist_dtor(Valist **thee)
00168 {
00169     if ((*thee) != VNULL) {
00170         Valist_dtor2(*thee);
00171         Vmem_free(VNULL, 1, sizeof(Valist), (void **)thee);
00172         (*thee) = VNULL;
00173     }
00174 }
00175
00176 VPUBLIC void Valist_dtor2(Valist *thee) {
00177
00178     Vmem_free(thee->vmem, thee->number, sizeof(Vatom), (void **)&(thee->
atoms));
00179     thee->atoms = VNULL;
00180     thee->number = 0;
00181
00182     Vmem_dtor(&(thee->vmem));
00183 }
00184
00185 /* Read serial number from PDB ATOM/HETATM field */
00186 VPRIVATE Vrc_Codes Valist_readPDBSerial(Valist *thee, Vio *sock, int *serial) {
00187
00188     char tok[VMAX_BUFSIZE];
00189     int ti = 0;
00190
00191     if (Vio_scanf(sock, "%s", tok) != 1) {
00192         Vnm_print(2, "Valist_readPDB: Ran out of tokens while parsing serial!\n");
00193         return VRC_FAILURE;
00194     }
00195     if (sscanf(tok, "%d", &ti) != 1) {
00196         Vnm_print(2, "Valist_readPDB: Unable to parse serial token (%s) as int!\n",
tok);
00197         return VRC_FAILURE;
00198     }
00199
00200     *serial = ti;
00201
00202     return VRC_SUCCESS;
00203 }
00204
00205 /* Read atom name from PDB ATOM/HETATM field */
00206 VPRIVATE Vrc_Codes Valist_readPDBAtomName(Valist *thee, Vio *sock,
char atomName[VMAX_ARGLEN]) {
00207
00208     char tok[VMAX_BUFSIZE];
00209
00210     if (Vio_scanf(sock, "%s", tok) != 1) {

```



```

00212         Vnm_print(2, "Valist_readPDB: Ran out of tokens while parsing atom name!\n");
00213         return VRC_FAILURE;
00214     }
00215     if (strlen(tok) < VMAX_ARGLEN) strcpy(atomName, tok);
00216     else {
00217         Vnm_print(2, "Valist_readPDB: Atom name (%s) too long!\n", tok);
00218         return VRC_FAILURE;
00219     }
00220     return VRC_SUCCESS;
00221 }
00222
00223 /* Read residue name from PDB ATOM/HETATM field */
00224 VPRIVATE Vrc_Codes Valist_readPDBResidueName(Valist *thee, Vio *sock,
00225     char resName[VMAX_ARGLEN]) {
00226
00227     char tok[VMAX_BUFSIZE];
00228
00229     if (Vio_scanf(sock, "%s", tok) != 1) {
00230         Vnm_print(2, "Valist_readPDB: Ran out of tokens while parsing residue name!\n");
00231         return VRC_FAILURE;
00232     }
00233     if (strlen(tok) < VMAX_ARGLEN) strcpy(resName, tok);
00234     else {
00235         Vnm_print(2, "Valist_readPDB: Residue name (%s) too long!\n", tok);
00236         return VRC_FAILURE;
00237     }
00238     return VRC_SUCCESS;
00239 }
00240
00241 /* Read residue number from PDB ATOM/HETATM field */
00242 VPRIVATE Vrc_Codes Valist_readPDBResidueNumber(
00243     Valist *thee, Vio *sock, int *resSeq) {
00244
00245     char tok[VMAX_BUFSIZE];
00246     char *resstring;
00247     int ti = 0;
00248
00249     if (Vio_scanf(sock, "%s", tok) != 1) {
00250         Vnm_print(2, "Valist_readPDB: Ran out of tokens while parsing resSeq!\n");
00251         return VRC_FAILURE;
00252     }
00253     if (sscanf(tok, "%d", &ti) != 1) {
00254
00255         /* One of three things can happen here:
00256            1) There is a chainID in the line:   THR A   1
00257            2) The chainID is merged with resSeq: THR A1001
00258            3) An actual error:                 THR foo
00259
00260         */
00261
00262         if (strlen(tok) == 1) {
00263             /* Case 1: Chain ID Present
00264                Read the next field and hope its a float */
00265
00266             if (Vio_scanf(sock, "%s", tok) != 1) {
00267                 Vnm_print(2, "Valist_readPDB: Ran out of tokens while parsing resSeq!\n");
00268                 return VRC_FAILURE;
00269             }
00270             if (sscanf(tok, "%d", &ti) != 1) {
00271                 Vnm_print(2, "Valist_readPDB: Unable to parse resSeq token (%s) as int!\n",
00272                     tok);
00273                 return VRC_FAILURE;
00274             }
00275
00276         } else {
00277             /* Case 2: Chain ID, merged string.
00278                Move pointer forward past the chainID and check
00279
00280             */
00281             //strcpy(resstring, tok);
00282             resstring = tok;
00283             resstring++;
00284
00285             if (sscanf(resstring, "%d", &ti) != 1) {
00286                 /* Case 3: More than one non-numeral char is present. Error.*/
00287                 Vnm_print(2, "Valist_readPDB: Unable to parse resSeq token (%s) as int!\n",
00288                     resstring);
00289                 return VRC_FAILURE;
00290             }
00291         }
00292     }
    *resSeq = ti;

```

```

00293
00294     return VRC_SUCCESS;
00295 }
00296
00297 /* Read atom coordinate from PDB ATOM/HETATM field */
00298 VPRIVATE Vrc_Codes Valist_readPDBAtomCoord(Valist *thee, Vio *sock, double *coord) {
00299     char tok[VMAX_BUFSIZE];
00300     double tf = 0;
00301
00302     if (Vio_scanf(sock, "%s", tok) != 1) {
00303         Vnm_print(2, "Valist_readPDB: Ran out of tokens while parsing atom coordinate!\n");
00304         return VRC_FAILURE;
00305     }
00306     if (sscanf(tok, "%lf", &tf) != 1) {
00307         return VRC_FAILURE;
00308     }
00309     *coord = tf;
00310
00311     return VRC_SUCCESS;
00312 }
00313
00314 /* Read charge and radius from PQR ATOM/HETATM field */
00315 VPRIVATE Vrc_Codes Valist_readPDBChargeRadius(Valist *thee, Vio *sock,
00316     double *charge, double *radius) {
00317     char tok[VMAX_BUFSIZE];
00318     double tf = 0;
00319
00320     if (Vio_scanf(sock, "%s", tok) != 1) {
00321         Vnm_print(2, "Valist_readPQR: Ran out of tokens while parsing charge!\n");
00322         return VRC_FAILURE;
00323     }
00324     if (sscanf(tok, "%lf", &tf) != 1) {
00325         return VRC_FAILURE;
00326     }
00327     *charge = tf;
00328
00329     if (Vio_scanf(sock, "%s", tok) != 1) {
00330         Vnm_print(2, "Valist_readPQR: Ran out of tokens while parsing radius!\n");
00331         return VRC_FAILURE;
00332     }
00333     if (sscanf(tok, "%lf", &tf) != 1) {
00334         return VRC_FAILURE;
00335     }
00336     *radius = tf;
00337
00338     return VRC_SUCCESS;
00339 }
00340
00341 /* Read ATOM/HETATM field of PDB through the X/Y/Z fields */
00342 VPRIVATE Vrc_Codes Valist_readPDB_throughXYZ(
00343     Valist *thee,
00344     Vio *sock, /* Socket ready for reading */
00345     int *serial, /* Set to atom number */
00346     char atomName[VMAX_ARGLEN], /* Set to atom name */
00347     char resName[VMAX_ARGLEN], /* Set to residue name */
00348     int *resSeq, /* Set to residue number */
00349     double *x, /* Set to x-coordinate */
00350     double *y, /* Set to y-coordinate */
00351     double *z /* Set to z-coordinate */
00352 ) {
00353     int i, njunk, gotit;
00354
00355     /* Grab serial */
00356     if (Valist_readPDBSerial(thee, sock, serial) == VRC_FAILURE) {
00357         Vnm_print(2, "Valist_readPDB: Error while parsing serial!\n");
00358     }
00359
00360     /* Grab atom name */
00361     if (Valist_readPDBAtomName(thee, sock, atomName) == VRC_FAILURE) {
00362         Vnm_print(2, "Valist_readPDB: Error while parsing atom name!\n");
00363         return VRC_FAILURE;
00364     }
00365
00366     /* Grab residue name */
00367     if (Valist_readPDBResidueName(thee, sock, resName) == VRC_FAILURE) {
00368         Vnm_print(2, "Valist_readPDB: Error while parsing residue name!\n");
00369         return VRC_FAILURE;
00370     }
00371
00372     return VRC_SUCCESS;
00373 }

```

```
00374     }
00375
00376
00377     /* Grab residue number */
00378     if (Valist_readPDBResidueNumber(thee, sock, resSeq) == VRC_FAILURE) {
00379         Vnm_print(2, "Valist_readPDB: Error while parsing residue number!\n");
00380         return VRC_FAILURE;
00381     }
00382
00383
00384     /* Read tokens until we find one that can be parsed as an atom
00385      * x-coordinate. We will allow njunk=1 intervening field that
00386      * cannot be parsed as a coordinate */
00387     njunk = 1;
00388     gotit = 0;
00389     for (i=0; i<(njunk+1); i++) {
00390         if (Valist_readPDBAtomCoord(thee, sock, x) == VRC_SUCCESS) {
00391             gotit = 1;
00392             break;
00393         }
00394     }
00395     if (!gotit) {
00396         Vnm_print(2, "Valist_readPDB: Can't find x!\n");
00397         return VRC_FAILURE;
00398     }
00399     /* Read y-coordinate */
00400     if (Valist_readPDBAtomCoord(thee, sock, y) == VRC_FAILURE) {
00401         Vnm_print(2, "Valist_readPDB: Can't find y!\n");
00402         return VRC_FAILURE;
00403     }
00404     /* Read z-coordinate */
00405     if (Valist_readPDBAtomCoord(thee, sock, z) == VRC_FAILURE) {
00406         Vnm_print(2, "Valist_readPDB: Can't find z!\n");
00407         return VRC_FAILURE;
00408     }
00409
00410     #if 0 /* Set to 1 if you want to debug */
00411         Vnm_print(1, "Valist_readPDB: serial = %d\n", *serial);
00412         Vnm_print(1, "Valist_readPDB: atomName = %s\n", atomName);
00413         Vnm_print(1, "Valist_readPDB: resName = %s\n", resName);
00414         Vnm_print(1, "Valist_readPDB: resSeq = %d\n", *resSeq);
00415         Vnm_print(1, "Valist_readPDB: pos = (%g, %g, %g)\n",
00416                 *x, *y, *z);
00417     #endif
00418
00419     return VRC_SUCCESS;
00420 }
00421
00422 /* Get a the next available atom storage location, increasing the storage
00423  * space if necessary. Return VNULL if something goes wrong. */
00424 VPRIVATE Vatom* Valist_getAtomStorage(
00425     Valist *thee,
00426     Vatom **plist, /* Pointer to existing list of atoms */
00427     int *pnlist, /* Size of existing list, may be changed */
00428     int *pnatoms /* Existing number of atoms in list; incremented
00429                  before exit */
00430 ) {
00431
00432     Vatom *oldList, *newList, *theList;
00433     Vatom *oldAtom, *newAtom;
00434     int iatom, inext, oldLength, newLength, natoms;
00435
00436     newList = VNULL;
00437
00438     /* See if we need more space */
00439     if (*pnatoms >= *pnlist) {
00440
00441         /* Double the storage space */
00442         oldLength = *pnlist;
00443         newLength = 2*oldLength;
00444         newList = (Vatom*)Vmem_malloc(thee->vmem, newLength, sizeof(
00445             Vatom));
00446         oldList = *plist;
00447
00448         /* Check the allocation */
00449         if (newList == VNULL) {
00450             Vnm_print(2, "Valist_readPDB: failed to allocate space for %d (Vatom)s!\n", newLength);
00451             return VNULL;
00452         }
00453
00454         /* Copy the atoms over */
00455     }
```

```

00454     natoms = *pnatoms;
00455     for (iatom=0; iatom<natoms; iatom++) {
00456         oldAtom = &(oldList[iatom]);
00457         newAtom = &(newList[iatom]);
00458         Vatom_copyTo(oldAtom, newAtom);
00459         Vatom_dtor2(oldAtom);
00460     }
00461
00462     /* Free the old list */
00463     Vmem_free(thee->vmem, oldLength, sizeof(Vatom), (void **)plist);
00464
00465     /* Copy new list to plist */
00466     *plist = newList;
00467     *pnlist = newLength;
00468 }
00469
00470 theList = *plist;
00471 inext = *pnatoms;
00472
00473 /* Get the next available spot and increment counters */
00474 newAtom = &(theList[inext]);
00475 *pnatoms = inext + 1;
00476
00477 return newAtom;
00478 }
00479
00480 VPRIVATE Vrc_Codes Valist_setAtomArray(Valist *thee,
00481     Vatom **plist, /* Pointer to list of atoms to store */
00482     int nlist, /* Length of list */
00483     int natoms /* Number of real atom entries in list */
00484 ) {
00485     Vatom *list, *newAtom, *oldAtom;
00486     int i;
00487
00488     list = *plist;
00489
00490     /* Allocate necessary space */
00491     thee->number = 0;
00492     thee->atoms = (Vatom*)Vmem_malloc(thee->vmem, natoms, sizeof(
00493     Vatom));
00494     if (thee->atoms == VNULL) {
00495         Vnm_print(2, "Valist_readPDB: Unable to allocate space for %d (Vatom)s!\n",
00496             natoms);
00497         return VRC_FAILURE;
00498     }
00499     thee->number = natoms;
00500
00501     /* Copy over data */
00502     for (i=0; i<thee->number; i++) {
00503         newAtom = &(thee->atoms[i]);
00504         oldAtom = &(list[i]);
00505         Vatom_copyTo(oldAtom, newAtom);
00506         Vatom_dtor2(oldAtom);
00507     }
00508
00509     /* Free old array */
00510     Vmem_free(thee->vmem, nlist, sizeof(Vatom), (void **)plist);
00511
00512     return VRC_SUCCESS;
00513 }
00514
00515 VPUBLIC Vrc_Codes Valist_readPDB(Valist *thee, Vparam *param, Vio *sock) {
00516
00517     /* WE DO NOT DIRECTLY CONFORM TO PDB STANDARDS -- TO ALLOW LARGER FILES, WE
00518      * REQUIRE ALL FIELDS TO BE WHITESPACE DELIMITED */
00519
00520     Vatom *atoms = VNULL;
00521     Vatom *nextAtom = VNULL;
00522     Vparam_AtomData *atomData = VNULL;
00523
00524     char tok[VMAX_BUFSIZE];
00525     char atomName[VMAX_ARGLEN], resName[VMAX_ARGLEN];
00526
00527     int nlist, natoms, serial, resSeq;
00528
00529     double x, y, z, charge, radius, epsilon;
00530     double pos[3];
00531
00532     if (thee == VNULL) {
00533         Vnm_print(2, "Valist_readPDB: Got NULL pointer when reading PDB file!\n");

```

```

00534     VASSERT(0);
00535 }
00536 thee->number = 0;
00537
00538 Vio_setWhiteChars(sock, Valist_whiteChars);
00539 Vio_setCommChars(sock, Valist_commChars);
00540
00541 /* Allocate some initial space for the atoms */
00542 nlist = 200;
00543 atoms = (Vatom*)Vmem_malloc(thee->vmem, nlist, sizeof(Vatom));
00544
00545 natoms = 0;
00546 /* Read until we run out of lines */
00547 while (Vio_scanf(sock, "%s", tok) == 1) {
00548
00549     /* Parse only ATOM/HETATOM fields */
00550     if ((Vstring_strcasecmp(tok, "ATOM") == 0) ||
00551         (Vstring_strcasecmp(tok, "HETATM") == 0)) {
00552
00553         /* Read ATOM/HETATM field of PDB through the X/Y/Z fields */
00554         if (Valist_readPDB_throughXYZ(thee, sock, &serial, atomName,
00555             resName, &resSeq, &x, &y, &z) == VRC_FAILURE) {
00556             Vnm_print(2, "Valist_readPDB: Error parsing atom %d!\n",
00557                 serial);
00558             return VRC_FAILURE;
00559         }
00560
00561         /* Try to find the parameters. */
00562         atomData = Vparam_getAtomData(param, resName, atomName);
00563         if (atomData == VNULL) {
00564             Vnm_print(2, "Valist_readPDB: Couldn't find parameters for \
00565 atom = %s, residue = %s\n", atomName, resName);
00566             return VRC_FAILURE;
00567         }
00568         charge = atomData->charge;
00569         radius = atomData->radius;
00570         epsilon = atomData->epsilon;
00571
00572         /* Get pointer to next available atom position */
00573         nextAtom = Valist_getAtomStorage(thee, &atoms, &nlist, &natoms);
00574         if (nextAtom == VNULL) {
00575             Vnm_print(2, "Valist_readPDB: Error in allocating spacing for atoms!\n");
00576             return VRC_FAILURE;
00577         }
00578
00579         /* Store the information */
00580         pos[0] = x; pos[1] = y; pos[2] = z;
00581         Vatom_setPosition(nextAtom, pos);
00582         Vatom_setCharge(nextAtom, charge);
00583         Vatom_setRadius(nextAtom, radius);
00584         Vatom_setEpsilon(nextAtom, epsilon);
00585         Vatom_setAtomID(nextAtom, natoms-1);
00586         Vatom_setResName(nextAtom, resName);
00587         Vatom_setAtomName(nextAtom, atomName);
00588
00589     } /* if ATOM or HETATM */
00590 } /* while we haven't run out of tokens */
00591
00592 Vnm_print(0, "Valist_readPDB: Counted %d atoms\n", natoms);
00593 fflush(stdout);
00594
00595 /* Store atoms internally */
00596 if (Valist_setAtomArray(thee, &atoms, nlist, natoms) == VRC_FAILURE) {
00597     Vnm_print(2, "Valist_readPDB: unable to store atoms!\n");
00598     return VRC_FAILURE;
00599 }
00600
00601 return Valist_getStatistics(thee);
00602
00603 }
00604 }
00605
00606 VPUBLIC Vrc_Codes Valist_readPQR(Valist *thee, Vparam *params, Vio *sock) {
00607
00608     /* WE DO NOT DIRECTLY CONFORM TO PDB STANDARDS -- TO ALLOW LARGER FILES, WE
00609      * REQUIRE ALL FIELDS TO BE WHITESPACE DELIMITED */
00610
00611
00612     Vatom *atoms = VNULL;
00613     Vatom *nextAtom = VNULL;
00614     Vparam_AtomData *atomData = VNULL;

```

```

00615
00616     char tok[VMAX_BUFSIZE];
00617     char atomName[VMAX_ARGLEN], resName[VMAX_ARGLEN];
00618     char chs[VMAX_BUFSIZE];
00619
00620     int use_params = 0;
00621     int nlist, natoms, serial, resSeq;
00622
00623     double x, y, z, charge, radius, epsilon;
00624     double pos[3];
00625
00626     epsilon = 0.0;
00627
00628     if (thee == VNULL) {
00629         Vnm_print(2, "Valist_readPQR: Got NULL pointer when reading PQR file!\n");
00630         VASSERT(0);
00631     }
00632     thee->number = 0;
00633
00634     Vio_setWhiteChars(sock, Valist_whiteChars);
00635     Vio_setCommChars(sock, Valist_commChars);
00636
00637     /* Allocate some initial space for the atoms */
00638     nlist = 200;
00639     atoms = (Vatom*)Vmem_malloc(thee->vmem, nlist, sizeof(Vatom));
00640
00641     /* Check if we are using a parameter file or not */
00642     if (params != VNULL) use_params = 1;
00643
00644     natoms = 0;
00645     /* Read until we run out of lines */
00646     while (Vio_scanf(sock, "%s", tok) == 1) {
00647
00648         /* Parse only ATOM/HETATOM fields */
00649         if ((Vstring_strcasecmp(tok, "ATOM") == 0) ||
00650             (Vstring_strcasecmp(tok, "HETATM") == 0)) {
00651
00652             /* Read ATOM/HETATM field of PDB through the X/Y/Z fields */
00653             if (Valist_readPDB_throughXYZ(thee, sock, &serial, atomName,
00654                 resName, &resSeq, &x, &y, &z) == VRC_FAILURE) {
00655                 Vnm_print(2, "Valist_readPQR: Error parsing atom %d!\n", serial);
00656                 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             /* Read Q/R fields */
00661             if (Valist_readPDBChargeRadius(thee, sock, &charge, &radius) ==
VRC_FAILURE) {
00662                 Vnm_print(2, "Valist_readPQR: Error parsing atom %d!\n",
serial);
00663                 Vnm_print(2, "Please double check this atom in the pqr file, e.g., make sure there are no
concatenated fields.\n");
00664                 return VRC_FAILURE;
00665             }
00666
00667             if (use_params) {
00668                 /* Try to find the parameters. */
00669                 atomData = Vparam_getAtomData(params, resName, atomName);
00670                 if (atomData == VNULL) {
00671                     Vnm_print(2, "Valist_readPDB: Couldn't find parameters for \
00672 atom = %s, residue = %s\n", atomName, resName);
00673                     return VRC_FAILURE;
00674                 }
00675                 charge = atomData->charge;
00676                 radius = atomData->radius;
00677                 epsilon = atomData->epsilon;
00678             }
00679
00680             /* Get pointer to next available atom position */
00681             nextAtom = Valist_getAtomStorage(thee, &atoms, &nlist, &natoms);
00682             if (nextAtom == VNULL) {
00683                 Vnm_print(2, "Valist_readPQR: Error in allocating spacing for atoms!\n");
00684                 return VRC_FAILURE;
00685             }
00686
00687             /* Store the information */
00688             pos[0] = x; pos[1] = y; pos[2] = z;
00689             Vatom_setPosition(nextAtom, pos);
00690             Vatom_setCharge(nextAtom, charge);
00691             Vatom_setRadius(nextAtom, radius);
00692

```

```

00693         Vatom_setEpsilon(nextAtom, epsilon);
00694         Vatom_setAtomID(nextAtom, natoms-1);
00695         Vatom_setResName(nextAtom, resName);
00696         Vatom_setAtomName(nextAtom, atomName);
00697
00698     } /* if ATOM or HETATM */
00699     else {
00700         /*
00701          * nop
00702          * Note that if we find a line that starts with something that's not
00703          * ATOM or HETATM we'll just keep parsing strings until we find one
00704          * of the acceptable keywords.
00705          * Extraordinary measures are not necessary, and only add to the
00706          * befuddlement.
00707          */
00708     }
00709 } /* while we haven't run out of tokens */
00710
00711 Vnm_print(0, "Valist_readPQR: Counted %d atoms\n", natoms);
00712 fflush(stdout);
00713
00714 /* Store atoms internally */
00715 if (Valist_setAtomArray(thee, &atoms, nlist, natoms) == VRC_FAILURE) {
00716     Vnm_print(2, "Valist_readPDB: unable to store atoms!\n");
00717     return VRC_FAILURE;
00718 }
00719
00720 return Valist_getStatistics(thee);
00721
00722 }
00723
00724
00725 VPUBLIC Vrc_Codes Valist_readXML(Valist *thee, Vparam *params, Vio *sock) {
00726     Vatom *atoms = VNULL;
00727     Vatom *nextAtom = VNULL;
00728
00729     char tok[VMAX_BUFSIZE];
00730     char endtag[VMAX_BUFSIZE];
00731
00732     int nlist, natoms;
00733     int xset, yset, zset, chgset, radset;
00734
00735     double x, y, z, charge, radius, dtmp;
00736     double pos[3];
00737
00738     if (thee == VNULL) {
00739         Vnm_print(2, "Valist_readXML: Got NULL pointer when reading XML file!\n");
00740         VASSERT(0);
00741     }
00742     thee->number = 0;
00743
00744     Vio_setWhiteChars(sock, Valist_xmlwhiteChars);
00745     Vio_setCommChars(sock, Valist_commChars);
00746
00747     /* Allocate some initial space for the atoms */
00748     nlist = 200;
00749     atoms = (Vatom*)Vmem_malloc(thee->vmem, nlist, sizeof(Vatom));
00750
00751     /* Initialize some variables */
00752     natoms = 0;
00753     xset = 0;
00754     yset = 0;
00755     zset = 0;
00756     chgset = 0;
00757     radset = 0;
00758     strcpy(endtag, "/");
00759
00760     if (params == VNULL) {
00761         Vnm_print(1, "\nValist_readXML: Warning Warning Warning Warning\n");
00762         Vnm_print(1, "Valist_readXML: The use of XML input files with parameter\n");
00763         Vnm_print(1, "Valist_readXML: files is currently not supported.\n");
00764         Vnm_print(1, "Valist_readXML: Warning Warning Warning Warning\n\n");
00765     }
00766
00767     /* Read until we run out of lines */
00768     while (Vio_scanf(sock, "%s", tok) == 1) {
00769         /* The first tag taken is the start tag - save it to detect end */
00770         if (Vstring_strcasecmp(endtag, "/") == 0) strcat(endtag, tok);
00771     }
00772 }
00773

```

```

00774         if (Vstring_strcasecmp(tok, "x") == 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 x!\n", tok);
00779                 return VRC_FAILURE;
00780             }
00781             x = dtmp;
00782             xset = 1;
00783         } else if (Vstring_strcasecmp(tok, "y") == 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 y!\n", tok);
00788                 return VRC_FAILURE;
00789             }
00790             y = dtmp;
00791             yset = 1;
00792         } else if (Vstring_strcasecmp(tok, "z") == 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 z!\n", tok);
00797                 return VRC_FAILURE;
00798             }
00799             z = dtmp;
00800             zset = 1;
00801         } else if (Vstring_strcasecmp(tok, "charge") == 0) {
00802             Vio_scanf(sock, "%s", tok);
00803             if (sscanf(tok, "%lf", &dtmp) != 1) {
00804                 Vnm_print(2, "Valist_readXML: Unexpected token (%s) while \
00805 reading charge!\n", tok);
00806                 return VRC_FAILURE;
00807             }
00808             charge = dtmp;
00809             chgset = 1;
00810         } else if (Vstring_strcasecmp(tok, "radius") == 0) {
00811             Vio_scanf(sock, "%s", tok);
00812             if (sscanf(tok, "%lf", &dtmp) != 1) {
00813                 Vnm_print(2, "Valist_readXML: Unexpected token (%s) while \
00814 reading radius!\n", tok);
00815                 return VRC_FAILURE;
00816             }
00817             radius = dtmp;
00818             radset = 1;
00819         } else if (Vstring_strcasecmp(tok, "/atom") == 0) {
00820
00821             /* Get pointer to next available atom position */
00822             nextAtom = Valist_getAtomStorage(thee, &atoms, &nlist, &natoms);
00823             if (nextAtom == VNULL) {
00824                 Vnm_print(2, "Valist_readXML: Error in allocating spacing for atoms!\n");
00825                 return VRC_FAILURE;
00826             }
00827
00828             if (xset && yset && zset && chgset && radset){
00829
00830                 /* Store the information */
00831                 pos[0] = x; pos[1] = y; pos[2] = z;
00832                 Vatom_setPosition(nextAtom, pos);
00833                 Vatom_setCharge(nextAtom, charge);
00834                 Vatom_setRadius(nextAtom, radius);
00835                 Vatom_setAtomID(nextAtom, natoms-1);
00836
00837                 /* Reset the necessary flags */
00838                 xset = 0;
00839                 yset = 0;
00840                 zset = 0;
00841                 chgset = 0;
00842                 radset = 0;
00843             } else {
00844                 Vnm_print(2, "Valist_readXML: Missing field(s) in atom tag:\n");
00845                 if (!xset) Vnm_print(2, "\tx value not set!\n");
00846                 if (!yset) Vnm_print(2, "\ty value not set!\n");
00847                 if (!zset) Vnm_print(2, "\tz value not set!\n");
00848                 if (!chgset) Vnm_print(2, "\tcharge value not set!\n");
00849                 if (!radset) Vnm_print(2, "\tradius value not set!\n");
00850                 return VRC_FAILURE;
00851             }
00852         } else if (Vstring_strcasecmp(tok, endtag) == 0) break;
00853     }
00854

```



```

00855     Vnm_print(0, "Valist_readXML: Counted %d atoms\n", natoms);
00856     fflush(stdout);
00857
00858     /* Store atoms internally */
00859     if (Valist_setAtomArray(thee, &atoms, nlist, natoms) == VRC_FAILURE) {
00860         Vnm_print(2, "Valist_readXML: unable to store atoms!\n");
00861         return VRC_FAILURE;
00862     }
00863
00864     return Valist_getStatistics(thee);
00865
00866 }
00867
00868 /* Load up Valist with various statistics */
00869 VPUBLIC Vrc_Codes Valist_getStatistics(Valist *thee) {
00870     Vatom *atom;
00871     int i, j;
00872
00873     if (thee == VNULL) {
00874         Vnm_print(2, "Valist_getStatistics: Got NULL pointer when loading up Valist with various
00875 statistics!\n");
00876         VASSERT(0);
00877     }
00878
00879     thee->center[0] = 0.;
00880     thee->center[1] = 0.;
00881     thee->center[2] = 0.;
00882     thee->maxrad = 0.;
00883     thee->charge = 0.;
00884
00885     if (thee->number == 0) return VRC_FAILURE;
00886
00887     /* Reset stat variables */
00888     atom = &(thee->atoms[0]);
00889     for (i=0; i<3; i++) {
00890         thee->maxcrd[i] = thee->mincrd[i] = atom->position[i];
00891     }
00892     thee->maxrad = atom->radius;
00893     thee->charge = 0.0;
00894
00895     for (i=0; i<thee->number; i++) {
00896
00897         atom = &(thee->atoms[i]);
00898         for (j=0; j<3; j++) {
00899             if (atom->position[j] < thee->mincrd[j])
00900                 thee->mincrd[j] = atom->position[j];
00901             if (atom->position[j] > thee->maxcrd[j])
00902                 thee->maxcrd[j] = atom->position[j];
00903         }
00904         if (atom->radius > thee->maxrad) thee->maxrad = atom->
radius;
00905         thee->charge = thee->charge + atom->charge;
00906     }
00907
00908     thee->center[0] = 0.5*(thee->maxcrd[0] + thee->mincrd[0]);
00909     thee->center[1] = 0.5*(thee->maxcrd[1] + thee->mincrd[1]);
00910     thee->center[2] = 0.5*(thee->maxcrd[2] + thee->mincrd[2]);
00911
00912     Vnm_print(0, "Valist_getStatistics: Max atom coordinate: (%g, %g, %g)\n",
00913               thee->maxcrd[0], thee->maxcrd[1], thee->maxcrd[2]);
00914     Vnm_print(0, "Valist_getStatistics: Min atom coordinate: (%g, %g, %g)\n",
00915               thee->mincrd[0], thee->mincrd[1], thee->mincrd[2]);
00916     Vnm_print(0, "Valist_getStatistics: Molecule center: (%g, %g, %g)\n",
00917               thee->center[0], thee->center[1], thee->center[2]);
00918
00919     return VRC_SUCCESS;
00920 }

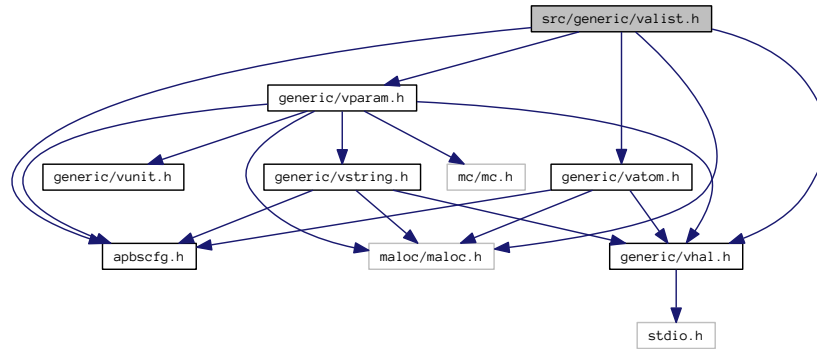
```

10.45 src/generic/valist.h File Reference

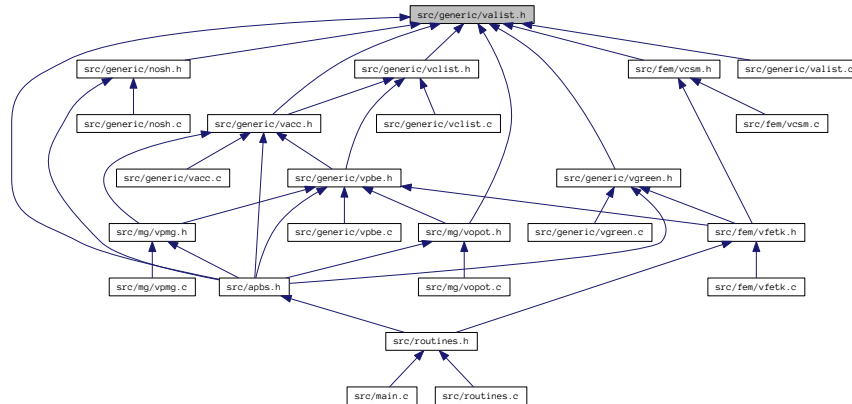
Contains declarations for class Valist.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/vatom.h"
#include "generic/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)`
FORTTRAN stub to construct the atom list object.
- VEXTERNC void `Valist_dtor (Valist **thee)`
Destroys atom list object.
- VEXTERNC void `Valist_dtor2 (Valist *thee)`
FORTTRAN 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.

10.45.1 Detailed Description

Contains declarations for class Valist.

Version

\$Id\$

Author

Nathan A. Baker

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [valist.h](#).

10.46 valist.h

```

00001
00062 #ifndef _VALIST_H_
00063 #define _VALIST_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "malloc/malloc.h"
00068
00069 #include "generic/vhal.h"
00070 #include "generic/vatom.h"
00071 #include "generic/vparam.h"
00072
00078 struct sValist {
00079

```

```

00080     int number;
00081     double center[3];
00082     double mincrd[3];
00083     double maxcrd[3];
00084     double maxrad;
00085     double charge;
00086     Vatom *atoms;
00087     Vmem *vmem;
00088 };
00089
00090
00095 typedef struct sValist Valist;
00096
00097 #if !defined(VINLINE_VATOM)
00098
00105 VEXTERNC Vatom* Valist_getAtomList(
00106     Valist *thee
00107 );
00108
00114 VEXTERNC double Valist_getCenterX(
00115     Valist *thee
00116 );
00117
00123 VEXTERNC double Valist_getCenterY(
00124     Valist *thee
00125 );
00126
00132 VEXTERNC double Valist_getCenterZ(
00133     Valist *thee
00134 );
00135
00141 VEXTERNC int Valist_getNumberAtoms(
00142     Valist *thee
00143 );
00144
00150 VEXTERNC Vatom* Valist_getAtom(
00151     Valist *thee,
00152     int i
00153 );
00154
00160 VEXTERNC unsigned long int Valist_memChk(
00161     Valist *thee
00162 );
00163
00164 #else /* if defined(VINLINE_VATOM) */
00165 #   define Valist_getAtomList(thee) ((thee)->atoms)
00166 #   define Valist_getNumberAtoms(thee) ((thee)->number)
00167 #   define Valist_getAtom(thee, i) (&((thee)->atoms[i]))
00168 #   define Valist_memChk(thee) (Vmem_bytes((thee)->vmem))
00169 #   define Valist_getCenterX(thee) ((thee)->center[0])
00170 #   define Valist_getCenterY(thee) ((thee)->center[1])
00171 #   define Valist_getCenterZ(thee) ((thee)->center[2])
00172 #endif /* if !defined(VINLINE_VATOM) */
00173
00179 VEXTERNC Valist* Valist_ctor();
00180
00186 VEXTERNC Vrc_Codes Valist_ctor2(
00187     Valist *thee
00188 );
00189
00194 VEXTERNC void Valist_dtor(
00195     Valist **thee
00196 );
00197
00202 VEXTERNC void Valist_dtor2(
00203     Valist *thee
00204 );
00205
00217 VEXTERNC Vrc_Codes Valist_readPQR(
00218     Valist *thee,
00219     Vparam *param,
00220     Vio *sock
00221 );
00222
00232 VEXTERNC Vrc_Codes Valist_readPDB(
00233     Valist *thee,
00234     Vparam *param,
00235     Vio *sock
00236 );
00237
00247 VEXTERNC Vrc_Codes Valist_readXML(
00248     Valist *thee,

```

```

00249         Vparam *param,
00250         Vio *sock
00251     );
00252
00259 VEXTERNC Vrc_Codes Valist_getStatistics(Valist *thee);
00260
00261
00262 #endif /* ifndef _VALIST_H_ */

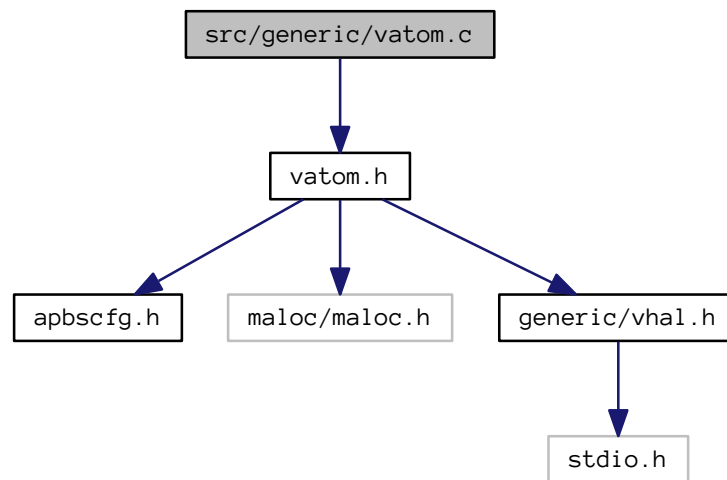
```

10.47 src/generic/vatom.c File Reference

Class Vatom methods.

```
#include "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 void [Vatom_setEpsilon](#) ([Vatom](#) *thee, double epsilon)
- Set atomic epsilon.*
- VPUBLIC double [Vatom_getEpsilon](#) ([Vatom](#) *thee)
- Get atomic epsilon.*
- 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)
- FORTTRAN stub constructor for the Vatom class.*
- VPUBLIC void [Vatom_dtor](#) ([Vatom](#) **thee)
- Object destructor.*
- VPUBLIC void [Vatom_dtor2](#) ([Vatom](#) *thee)
- FORTTRAN 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.*

10.47.1 Detailed Description

Class Vatom methods.

Author

Nathan Baker

Version

\$Id\$

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vatom.c](#).

10.48 vatom.c

```

00001
00057 #include "vatom.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_VATOM)
00062
00063 VPUBLIC double *Vatom_getPosition(Vatom *thee) {
00064
00065     VASSERT(thee != VNULL);
00066     return thee->position;
00067 }
00068
00069

```



```

00070 VPUBLIC double Vatom_getPartID(Vatom *thee) {
00071
00072     VASSERT(thee != VNULL);
00073     return thee->partID;
00074 }
00075 }
00076
00077 VPUBLIC void Vatom_setPartID(Vatom *thee, int partID) {
00078
00079     VASSERT(thee != VNULL);
00080     thee->partID = (double)partID;
00081 }
00082 }
00083
00084 VPUBLIC double Vatom_getAtomID(Vatom *thee) {
00085
00086     VASSERT(thee != VNULL);
00087     return thee->id;
00088 }
00089 }
00090
00091 VPUBLIC void Vatom_setAtomID(Vatom *thee, int atomID) {
00092
00093     VASSERT(thee != VNULL);
00094     thee->id = atomID;
00095 }
00096 }
00097
00098 VPUBLIC void Vatom_setRadius(Vatom *thee, double radius) {
00099
00100     VASSERT(thee != VNULL);
00101     thee->radius = radius;
00102 }
00103 }
00104
00105 VPUBLIC double Vatom_getRadius(Vatom *thee) {
00106
00107     VASSERT(thee != VNULL);
00108     return thee->radius;
00109 }
00110 }
00111
00112 VPUBLIC void Vatom_setCharge(Vatom *thee, double charge) {
00113
00114     VASSERT(thee != VNULL);
00115     thee->charge = charge;
00116 }
00117 }
00118
00119 VPUBLIC double Vatom_getCharge(Vatom *thee) {
00120
00121     VASSERT(thee != VNULL);
00122     return thee->charge;
00123 }
00124 }
00125
00126 VPUBLIC void Vatom_setEpsilon(Vatom *thee, double epsilon) {
00127
00128     VASSERT(thee != VNULL);
00129     thee->epsilon = epsilon;
00130 }
00131
00132 VPUBLIC double Vatom_getEpsilon(Vatom *thee) {
00133
00134     VASSERT(thee != VNULL);
00135     return thee->epsilon;
00136 }
00137
00138 VPUBLIC unsigned long int Vatom_memChk(Vatom *thee) { return sizeof(
    Vatom); }
00139
00140 #endif /* if !defined(VINLINE_VATOM) */
00141
00142 VPUBLIC Vatom* Vatom_ctor() {
00143
00144     /* Set up the structure */
00145     Vatom *thee = VNULL;
00146     thee = (Vatom *)Vmem_malloc( VNULL, 1, sizeof(Vatom) );
00147     VASSERT( thee != VNULL);
00148     VASSERT( Vatom_ctor2(thee));
00149 }

```

```

00150     return thee;
00151 }
00152
00153 VPUBLIC int Vatom_ctor2(Vatom *thee) {
00154     thee->partID = -1;
00155     return 1;
00156 }
00157
00158 VPUBLIC void Vatom_dtor(Vatom **thee) {
00159     if ((*thee) != VNULL) {
00160         Vatom_dtor2(*thee);
00161         Vmem_free(VNULL, 1, sizeof(Vatom), (void **)thee);
00162         (*thee) = VNULL;
00163     }
00164 }
00165
00166 VPUBLIC void Vatom_dtor2(Vatom *thee) { ; }
00167
00168 VPUBLIC void Vatom_setPosition(Vatom *thee, double position[3]) {
00169
00170     VASSERT(thee != VNULL);
00171     (thee->position)[0] = position[0];
00172     (thee->position)[1] = position[1];
00173     (thee->position)[2] = position[2];
00174 }
00175 }
00176
00177 VPUBLIC void Vatom_copyTo(Vatom *thee, Vatom *dest) {
00178
00179     VASSERT(thee != VNULL);
00180     VASSERT(dest != VNULL);
00181
00182     memcpy(dest, thee, sizeof(Vatom));
00183 }
00184 }
00185
00186 VPUBLIC void Vatom_copyFrom(Vatom *thee, Vatom *src) {
00187
00188     Vatom_copyTo(src, thee);
00189 }
00190 }
00191
00192 VPUBLIC void Vatom_setResName(Vatom *thee, char resName[
VMAX_RECLEN]) {
00193
00194     VASSERT(thee != VNULL);
00195     strcpy(thee->resName, resName);
00196 }
00197 }
00198
00199 VPUBLIC void Vatom_getResName(Vatom *thee, char resName[
VMAX_RECLEN]) {
00200
00201
00202     VASSERT(thee != VNULL);
00203     strcpy(resName,thee->resName);
00204 }
00205 }
00206
00207 VPUBLIC void Vatom_setAtomName(Vatom *thee, char atomName[
VMAX_RECLEN]) {
00208
00209     VASSERT(thee != VNULL);
00210     strcpy(thee->atomName, atomName);
00211 }
00212 }
00213
00214 VPUBLIC void Vatom_getAtomName(Vatom *thee, char atomName[
VMAX_RECLEN]) {
00215
00216     VASSERT(thee != VNULL);
00217     strcpy(atomName,thee->atomName);
00218 }
00219 }
00220
00221 #if defined(WITH_TINKER)
00222
00223 VPUBLIC void Vatom_setDipole(Vatom *thee, double dipole[3]) {
00224
00225     VASSERT(thee != VNULL);
00226     (thee->dipole)[0] = dipole[0];

```

```

00227     (thee->dipole)[1] = dipole[1];
00228     (thee->dipole)[2] = dipole[2];
00229
00230 }
00231
00232 VPUBLIC void Vatom_setQuadrupole(Vatom *thee, double quadrupole[9]) {
00233     int i;
00234     VASSERT(thee != VNULL);
00235     for (i=0; i<9; i++) (thee->quadrupole)[i] = quadrupole[i];
00236 }
00237
00238
00239 VPUBLIC void Vatom_setInducedDipole(Vatom *thee, double dipole[3]) {
00240     VASSERT(thee != VNULL);
00241     (thee->inducedDipole)[0] = dipole[0];
00242     (thee->inducedDipole)[1] = dipole[1];
00243     (thee->inducedDipole)[2] = dipole[2];
00244 }
00245
00246
00247 VPUBLIC void Vatom_setNLInducedDipole(Vatom *thee, double dipole[3]) {
00248     VASSERT(thee != VNULL);
00249     (thee->nlInducedDipole)[0] = dipole[0];
00250     (thee->nlInducedDipole)[1] = dipole[1];
00251     (thee->nlInducedDipole)[2] = dipole[2];
00252 }
00253
00254 }
00255
00256 VPUBLIC double *Vatom_getDipole(Vatom *thee) {
00257     VASSERT(thee != VNULL);
00258     return thee->dipole;
00259 }
00260
00261 }
00262
00263 VPUBLIC double *Vatom_getQuadrupole(Vatom *thee) {
00264     VASSERT(thee != VNULL);
00265     return thee->quadrupole;
00266 }
00267
00268 }
00269
00270 VPUBLIC double *Vatom_getInducedDipole(Vatom *thee) {
00271     VASSERT(thee != VNULL);
00272     return thee->inducedDipole;
00273 }
00274
00275 }
00276
00277 VPUBLIC double *Vatom_getNLInducedDipole(Vatom *thee) {
00278     VASSERT(thee != VNULL);
00279     return thee->nlInducedDipole;
00280 }
00281
00282 }
00283
00284 #endif /* if defined(WITH_TINKER) */

```

10.49 src/generic/vatom.h File Reference

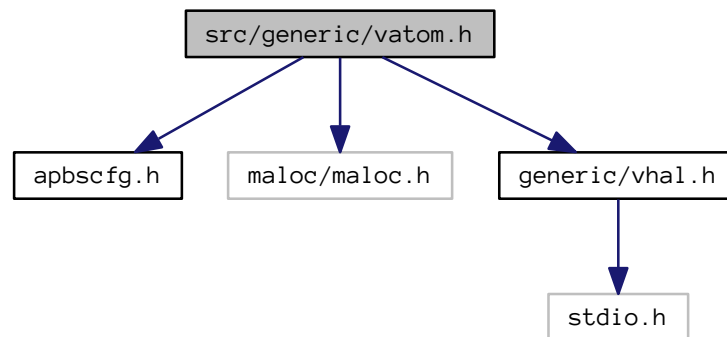
Contains declarations for class Vatom.

```

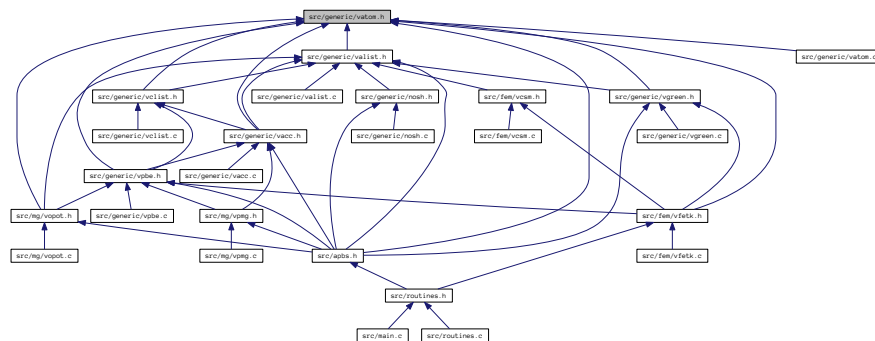
#include "apbscfg.h"
#include "malloc/malloc.h"
#include "generic/vhal.h"

```

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.

Macros

- `#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)
FORTTRAN stub constructor for the Vatom class.
- VEXTERNC void [Vatom_dtor](#) ([Vatom](#) **thee)
Object destructor.
- VEXTERNC void [Vatom_dtor2](#) ([Vatom](#) *thee)
FORTTRAN 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.

10.49.1 Detailed Description

Contains declarations for class Vatom.

Version

\$Id\$

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [vatom.h](#).

10.50 vatom.h

```

00001
00062 #ifndef _VATOM_H_
00063 #define _VATOM_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "malloc/malloc.h"
00068
00069 #include "generic/vhal.h"
00070
00077 #define VMAX_RECLEN      64
00078
00084 struct sVatom {
00085
00086     double position[3];
00087     double radius;
00088     double charge;
00089     double partID;
00091     double epsilon;
00093     int id;
00097     char resName[VMAX_RECLEN];
00098     char atomName[VMAX_RECLEN];
00100 #if defined(WITH_TINKER)
00101
00102     double dipole[3];
00103     double quadrupole[9];
00104     double inducedDipole[3];
00105     double nlInducedDipole[3];
00107 #endif /* if defined(WITH_TINKER) */
00108 };
00109
00114 typedef struct sVatom Vatom;
00115
00116 #if !defined(VINLINE_VATOM)
00117
00124     VEXTERNC double* Vatom_getPosition(Vatom *thee);
00125
00132     VEXTERNC void Vatom_setRadius(Vatom *thee, double
radius);
00133
00140     VEXTERNC double Vatom_getRadius(Vatom *thee);
00141
00149     VEXTERNC void Vatom_setPartID(Vatom *thee, int partID);
00150
00158     VEXTERNC double Vatom_getPartID(Vatom *thee);
00159
00166     VEXTERNC void Vatom_setAtomID(Vatom *thee, int id);
00167
00174     VEXTERNC double Vatom_getAtomID(Vatom *thee);
00175
00182     VEXTERNC void Vatom_setCharge(Vatom *thee, double
charge);
00183
00190     VEXTERNC double Vatom_getCharge(Vatom *thee);
00191
00198     VEXTERNC void Vatom_setEpsilon(Vatom *thee, double
epsilon);
00199
00206     VEXTERNC double Vatom_getEpsilon(Vatom *thee);
00207
00215     VEXTERNC unsigned long int Vatom_memChk(Vatom *thee);
00216
00217 #else /* if defined(VINLINE_VATOM) */
00218 # define Vatom_getPosition(thee) ((thee)->position)
00219 # define Vatom_setRadius(thee, tRadius) ((thee)->radius = (tRadius))
00220 # define Vatom_getRadius(thee) ((thee)->radius)
00221 # define Vatom_setPartID(thee, tpartID) ((thee)->partID = (double)(tpartID))
00222 # define Vatom_getPartID(thee) ((thee)->partID)
00223 # define Vatom_setAtomID(thee, tatomID) ((thee)->id = (tatomID))
00224 # define Vatom_getAtomID(thee) ((thee)->id)
00225 # define Vatom_setCharge(thee, tCharge) ((thee)->charge = (tCharge))
00226 # define Vatom_getCharge(thee) ((thee)->charge)
00227 # define Vatom_setEpsilon(thee, tEpsilon) ((thee)->epsilon = (tEpsilon))
00228 # define Vatom_getEpsilon(thee) ((thee)->epsilon)
00229 # define Vatom_memChk(thee) (sizeof(Vatom))
00230 #endif /* if !defined(VINLINE_VATOM) */
00231
00232 /* //////////////////////////////////////

```

```

00233 // Class Vatom: Non-Inlineable methods (vatom.c)
00235
00242 VEXTERNC void    Vatom_setResName(Vatom *thee, char resName[VMAX_RECLEN]);
00243
00248 VEXTERNC void    Vatom_setAtomName(
00249     Vatom *thee,
00250     char atomName[VMAX_RECLEN]
00251 );
00252
00259 VEXTERNC void    Vatom_getResName(Vatom *thee, char resName[VMAX_RECLEN]);
00260
00265 VEXTERNC void    Vatom_getAtomName(
00266     Vatom *thee,
00267     char atomName[VMAX_RECLEN]
00268 );
00269
00275 VEXTERNC Vatom*  Vatom_ctor();
00276
00283 VEXTERNC int     Vatom_ctor2(Vatom *thee);
00284
00290 VEXTERNC void    Vatom_dtor(Vatom **thee);
00291
00297 VEXTERNC void    Vatom_dtor2(Vatom *thee);
00298
00305 VEXTERNC void    Vatom_setPosition(Vatom *thee, double position[3]);
00306
00314 VEXTERNC void    Vatom_copyTo(Vatom *thee, Vatom *dest);
00315
00323 VEXTERNC void    Vatom_copyFrom(Vatom *thee, Vatom *src);
00324
00325 #if defined(WITH_TINKER)
00326
00333 VEXTERNC void    Vatom_setInducedDipole(Vatom *thee,
00334     double inducedDipole[3]);
00335
00342 VEXTERNC void    Vatom_setNLInducedDipole(Vatom *thee,
00343     double nlInducedDipole[3]);
00344
00351 VEXTERNC void    Vatom_setDipole(Vatom *thee, double dipole[3]);
00352
00359 VEXTERNC void    Vatom_setQuadrupole(Vatom *thee, double quadrupole[9]);
00360
00366 VEXTERNC double* Vatom_getDipole(Vatom *thee);
00367
00373 VEXTERNC double* Vatom_getQuadrupole(Vatom *thee);
00374
00380 VEXTERNC double* Vatom_getInducedDipole(Vatom *thee);
00381
00387 VEXTERNC double* Vatom_getNLInducedDipole(Vatom *thee);
00388 #endif /* if defined(WITH_TINKER) */
00389
00390 #endif /* ifndef _VATOM_H_ */

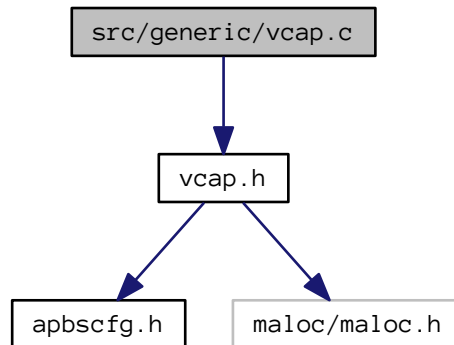
```

10.51 src/generic/vcap.c File Reference

Class Vcap methods.


```
#include "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.

10.51.1 Detailed Description

Class Vcap methods.

Author

Nathan Baker

Version

\$Id\$

Attention

```

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

```

```

* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vcap.c](#).

10.52 vcap.c

```

00001
00057 #include "vcap.h"
00058
00059 VPUBLIC double Vcap_exp(double x, int *ichop) {
00060
00061     /* The two chopped arguments */
00062     if (x > EXPMAX) {
00063         (*ichop) = 1;
00064         return VEXP(EXPMAX);
00065     } else if (x < EXPMIN) {
00066         (*ichop) = 1;
00067         return VEXP(EXPMIN);
00068     }
00069
00070     /* The normal EXP */
00071     (*ichop) = 0;
00072     return VEXP(x);
00073 }
00074
00075 VPUBLIC double Vcap_sinh(double x, int *ichop) {
00076
00077     /* The two chopped arguments */
00078     if (x > EXPMAX) {
00079         (*ichop) = 1;
00080         return VSINH(EXPMAX);
00081     } else if (x < EXPMIN) {

```

```
00082     (*ichop) = 1;
00083     return VSINH(EXPMIN);
00084 }
00085
00086 /* The normal SINH */
00087 (*ichop) = 0;
00088 return VSINH(x);
00089 }
00090
00091 VPUBLIC double Vcap_cosh(double x, int *ichop) {
00092
00093     /* The two chopped arguments */
00094     if (x > EXPMAX) {
00095         (*ichop) = 1;
00096         return VCOSH(EXPMAX);
00097     } else if (x < EXPMIN) {
00098         (*ichop) = 1;
00099         return VCOSH(EXPMIN);
00100     }
00101
00102     /* The normal COSH */
00103     (*ichop) = 0;
00104     return VCOSH(x);
00105 }
00106
```

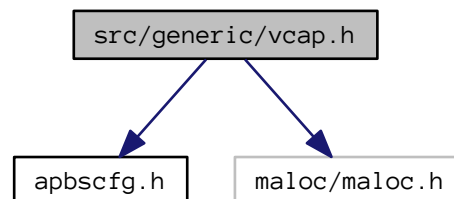
10.53 src/generic/vcap.h File Reference

Contains declarations for class Vcap.

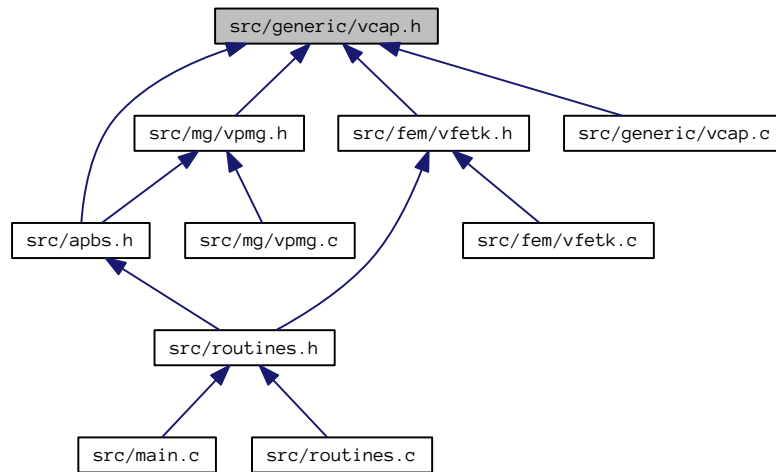
```
#include "apbscfg.h"
```

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

Include dependency graph for vcap.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#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.

10.53.1 Detailed Description

Contains declarations for class `Vcap`.

Version

\$Id\$

Author

Nathan A. Baker

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vcap.h](#).

10.54 vcap.h

```

00001
00064 #ifndef _VCAP_H_
00065 #define _VCAP_H_
00066
00067 #include "apbscfg.h"
00068
00072 #define EXPMAX 85.00
00073
00077 #define EXPMIN -85.00
00078
00079 #include "malloc/malloc.h"
00080
00099 VEXTERNC double Vcap_exp(
00100     double x,

```

```

00101         int *ichop
00102         );
00103
00104
00123 VEXTERNC double Vcap_sinh(
00124         double x,
00125         int *ichop
00126         );
00127
00146 VEXTERNC double Vcap_cosh(
00147         double x,
00148         int *ichop
00149         );
00150
00151 #endif      /* ifndef _VCAP_H_ */

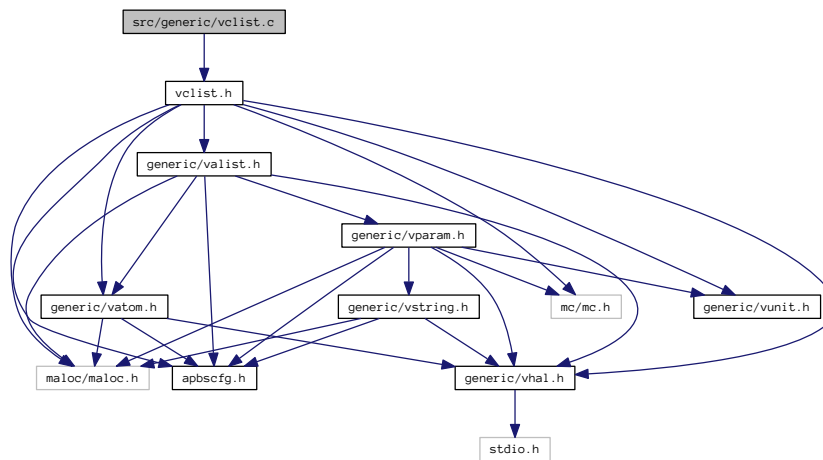
```

10.55 src/generic/vclist.c File Reference

Class Vclist methods.

```
#include "vclist.h"
```

Include dependency graph for vclist.c:



Macros

- #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 Å) 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**])
FORTTRAN stub to construct the cell list object.
- VPUBLIC void **Vclist_dtor** (**Vclist** **thee)
Destroy object.
- VPUBLIC void **Vclist_dtor2** (**Vclist** *thee)
FORTTRAN 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)
FORTTRAN stub to destroy object.

10.55.1 Detailed Description

Class Vclist methods.

Author

Nathan Baker

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
```

```

* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vclist.c](#).

10.56 vclist.c

```

00001
00057 #include "vclist.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_VCLIST)
00062
00063 VPUBLIC unsigned long int Vclist_memChk(Vclist *thee) {
00064     if (thee == VNULL) return 0;
00065     return Vmem_bytes(thee->vmem);
00066 }
00067
00068 VPUBLIC double Vclist_maxRadius(Vclist *thee) {
00069     VASSERT(thee != VNULL);
00070     return thee->max_radius;
00071 }
00072
00073 #endif /* if !defined(VINLINE_VCLIST) */
00074
00075 VPUBLIC Vclist* Vclist_ctor(Valist *alist, double max_radius,
00076     int npts[VAPBS_DIM], Vclist_DomainMode mode,
00077     double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM]) {
00078
00079     Vclist *thee = VNULL;
00080
00081     /* Set up the structure */
00082     thee = (Vclist*)Vmem_malloc(VNULL, 1, sizeof(Vclist) );
00083     VASSERT( thee != VNULL);
00084     VASSERT( Vclist_ctor2(thee, alist, max_radius, npts, mode, lower_corner,
00085         upper_corner) == VRC_SUCCESS );
00086     return thee;
00087 }

```



```

00088
00089 /* Get the dimensions of the molecule stored in thee->alist */
00090 VPRIVATE void Vclist_getMolDims(
00091     Vclist *thee,
00092     double lower_corner[VAPBS_DIM], /* Set to lower corner of molecule */
00093     double upper_corner[VAPBS_DIM], /* Set to lower corner of molecule */
00094     double *r_max /* Set to max atom radius */
00095 ) {
00096
00097     int i, j;
00098     double pos;
00099     Valist *alist;
00100     Vatom *atom;
00101
00102     alist = thee->alist;
00103
00104     /* Initialize */
00105     for (i=0; i<VAPBS_DIM; i++) {
00106         lower_corner[i] = VLARGE;
00107         upper_corner[i] = -VLARGE;
00108     }
00109     *r_max = -1.0;
00110
00111     /* Check each atom */
00112     for (i=0; i<Valist_getNumberAtoms(alist); i++) {
00113         atom = Valist_getAtom(alist, i);
00114         for (j=0; j<VAPBS_DIM; j++) {
00115             pos = (Vatom_getPosition(atom))[j];
00116             if ( pos < lower_corner[j] ) lower_corner[j] = pos;
00117             if ( pos > upper_corner[j] ) upper_corner[j] = pos;
00118         }
00119         if (Vatom_getRadius(atom) > *r_max) *r_max =
00120             Vatom_getRadius(atom);
00121     }
00122 }
00123
00124 /* Setup lookup grid */
00125 VPRIVATE Vrc_Codes Vclist_setupGrid(Vclist *thee) {
00126
00127     /* Inflation factor ~ sqrt(2)*/
00128     #define VCLIST_INFLATE 1.42
00129
00130     int i;
00131     double length[VAPBS_DIM], r_max;
00132
00133     /* Set up the grid corners */
00134     switch (thee->mode) {
00135         case CLIST_AUTO_DOMAIN:
00136             /* Get molecule dimensions */
00137             Vclist_getMolDims(thee, thee->lower_corner, thee->
00138                 upper_corner,
00139                 &r_max);
00140             /* Set up grid spacings */
00141             for (i=0; i<VAPBS_DIM; i++) {
00142                 thee->upper_corner[i] = thee->upper_corner[i]
00143                     + VCLIST_INFLATE*(r_max+thee->max_radius);
00144                 thee->lower_corner[i] = thee->lower_corner[i]
00145                     - VCLIST_INFLATE*(r_max+thee->max_radius);
00146             }
00147             break;
00148         case CLIST_MANUAL_DOMAIN:
00149             /* Grid corners established in constructor */
00150             break;
00151         default:
00152             Vnm_print(2, "Vclist_setupGrid: invalid setup mode (%d)!\n",
00153                 thee->mode);
00154             return VRC_FAILURE;
00155     }
00156
00157     /* Set up the grid lengths and spacings */
00158     for (i=0; i<VAPBS_DIM; i++) {
00159         length[i] = thee->upper_corner[i] - thee->lower_corner[i];
00160         thee->spacs[i] = length[i]/((double)(thee->npts[i] - 1));
00161     }
00162     Vnm_print(0, "Vclist_setupGrid: Grid lengths = (%g, %g, %g)\n",
00163         length[0], length[1], length[2]);
00164
00165     Vnm_print(0, "Vclist_setupGrid: Grid lower corner = (%g, %g, %g)\n",
00166         (thee->lower_corner)[0], (thee->lower_corner)[1],
00167         (thee->lower_corner)[2]);

```

```

00167
00168     return VRC_SUCCESS;
00169
00170     #undef VCLIST_INFLATE
00171 }
00172
00173 /* Check and store parameters passed to constructor */
00174 VPRIVATE Vrc_Codes Vclist_storeParms(Vclist *thee, Valist *alist,
00175     double max_radius, int npts[VAPBS_DIM], Vclist_DomainMode mode,
00176     double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM] ) {
00177
00178     int i = 0;
00179
00180     if (alist == VNULL) {
00181         Vnm_print(2, "Vclist_ctor2: Got NULL Valist!\n");
00182         return VRC_FAILURE;
00183     } else thee->alist = alist;
00184
00185     thee->n = 1;
00186     for (i=0; i<VAPBS_DIM; i++) {
00187         if (npts[i] < 3) {
00188             Vnm_print(2,
00189                 "Vclist_ctor2: n[%d] (%d) must be greater than 2!\n",
00190                 i, npts[i]);
00191             return VRC_FAILURE;
00192         }
00193         thee->npts[i] = npts[i];
00194         thee->n *= npts[i];
00195     }
00196     Vnm_print(0, "Vclist_ctor2: Using %d x %d x %d hash table\n",
00197         npts[0], npts[1], npts[2]);
00198
00199     thee->mode = mode;
00200     switch (thee->mode) {
00201         case CLIST_AUTO_DOMAIN:
00202             Vnm_print(0, "Vclist_ctor2: automatic domain setup.\n");
00203             break;
00204         case CLIST_MANUAL_DOMAIN:
00205             Vnm_print(0, "Vclist_ctor2: manual domain setup.\n");
00206             Vnm_print(0, "Vclist_ctor2: lower corner = [ \n");
00207             for (i=0; i<VAPBS_DIM; i++) {
00208                 thee->lower_corner[i] = lower_corner[i];
00209                 Vnm_print(0, "%g ", lower_corner[i]);
00210             }
00211             Vnm_print(0, "]\n");
00212             Vnm_print(0, "Vclist_ctor2: upper corner = [ \n");
00213             for (i=0; i<VAPBS_DIM; i++) {
00214                 thee->upper_corner[i] = upper_corner[i];
00215                 Vnm_print(0, "%g ", upper_corner[i]);
00216             }
00217             Vnm_print(0, "]\n");
00218             break;
00219         default:
00220             Vnm_print(2, "Vclist_ctor2: invalid setup mode (%d)!\n", mode);
00221             return VRC_FAILURE;
00222     }
00223
00224     thee->max_radius = max_radius;
00225     Vnm_print(0, "Vclist_ctor2: Using %g max radius\n", max_radius);
00226
00227     return VRC_SUCCESS;
00228 }
00229
00230 /* Calculate the gridpoints an atom spans */
00231 VPRIVATE void Vclist_gridSpan(Vclist *thee,
00232     Vatom *atom, /* Atom */
00233     int imin[VAPBS_DIM], /* Set to min grid indices */
00234     int imax[VAPBS_DIM] /* Set to max grid indices */
00235 ) {
00236
00237     int i;
00238     double *coord, dc, idc, rtot;
00239
00240     /* Get the position in the grid's frame of reference */
00241     coord = Vatom_getPosition(atom);
00242
00243     /* Get the range the atom radius + probe radius spans */
00244     rtot = Vatom_getRadius(atom) + thee->max_radius;
00245
00246     /* Calculate the range of grid points the inflated atom spans in the x
00247     * direction. */

```

```

00248     for (i=0; i<VAPBS_DIM; i++) {
00249         dc = coord[i] - (thee->lower_corner)[i];
00250         idc = (dc + rtot)/(thee->spacs[i]);
00251         imax[i] = (int)(ceil(idc));
00252         imax[i] = VMIN2(imax[i], thee->npts[i]-1);
00253         idc = (dc - rtot)/(thee->spacs[i]);
00254         imin[i] = (int)(floor(idc));
00255         imin[i] = VMAX2(imin[i], 0);
00256     }
00257 }
00258 }
00259
00260 /* Get the array index for a particular cell based on its i,j,k
00261  * coordinates */
00262 VPRIVATE int Vclist_arrayIndex(Vclist *thee, int i, int j, int k) {
00263     return (thee->npts[2])*(thee->npts[1])*i + (thee->npts[2])*j + k;
00264 }
00265
00266 }
00267
00268
00269 /* Assign atoms to cells */
00270 VPRIVATE Vrc_Codes Vclist_assignAtoms(Vclist *thee) {
00271     int iatom, i, j, k, ui, inext;
00272     int imax[VAPBS_DIM], imin[VAPBS_DIM];
00273     int totatoms;
00274     Vatom *atom;
00275     VclistCell *cell;
00276
00277     /* Find out how many atoms are associated with each grid point */
00278     totatoms = 0;
00279     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
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     }
00300 }
00301
00302 Vnm_print(0, "Vclist_assignAtoms: Have %d atom entries\n", totatoms);
00303
00304 /* Allocate the space to store the pointers to the atoms */
00305 for (ui=0; ui<thee->n; ui++) {
00306     cell = &(thee->cells[ui]);
00307     if ( VclistCell_ctor2(cell, cell->natoms) ==
VRC_FAILURE ) {
00308         Vnm_print(2, "Vclist_assignAtoms: cell error!\n");
00309         return VRC_FAILURE;
00310     }
00311     /* Clear the counter for later use */
00312     cell->natoms = 0;
00313 }
00314
00315 /* Assign the atoms to grid points */
00316 for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
00317     /* Get grid span for atom */
00318     atom = Valist_getAtom(thee->alist, iatom);
00319     Vclist_gridSpan(thee, atom, imin, imax);
00320
00321     /* Now find and assign the grid points */
00322     for (i = imin[0]; i <= imax[0]; i++) {
00323         for (j = imin[1]; j <= imax[1]; j++) {
00324             for (k = imin[2]; k <= imax[2]; k++) {
00325                 /* Get index to array */
00326                 ui = Vclist_arrayIndex(thee, i, j, k);

```

```

00328         cell = &(thee->cells[ui]);
00329         /* Index of next available array location */
00330         inext = cell->natoms;
00331         cell->atoms[inext] = atom;
00332         /* Increment number of atoms */
00333         (cell->natoms)++;
00334     }
00335 }
00336 }
00337 }
00338
00339     return VRC_SUCCESS;
00340 }
00341
00342 /* Main (FORTRAN stub) constructor */
00343 VPUBLIC Vrc_Codes Vclist_ctor2(Vclist *thee, Valist *alist, double max_radius,
00344     int npts[VAPBS_DIM], Vclist_DomainMode mode,
00345     double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM]) {
00346
00347     int i;
00348     VclistCell *cell;
00349
00350     /* Check and store parameters */
00351     if ( Vclist_storeParms(thee, alist, max_radius, npts, mode, lower_corner,
00352         upper_corner) == VRC_FAILURE ) {
00353         Vnm_print(2, "Vclist_ctor2:  parameter check failed!\n");
00354         return VRC_FAILURE;
00355     }
00356
00357     /* Set up memory */
00358     thee->vmem = Vmem_ctor("APBS::VCLIST");
00359     if (thee->vmem == VNULL) {
00360         Vnm_print(2, "Vclist_ctor2:  memory object setup failed!\n");
00361         return VRC_FAILURE;
00362     }
00363
00364     /* Set up cells */
00365     thee->cells = (VclistCell*)Vmem_malloc( thee->vmem, thee->
n, sizeof(VclistCell) );
00366     if (thee->cells == VNULL) {
00367         Vnm_print(2,
00368             "Vclist_ctor2:  Failed allocating %d VclistCell objects!\n",
00369             thee->n);
00370         return VRC_FAILURE;
00371     }
00372     for (i=0; i<thee->n; i++) {
00373         cell = &(thee->cells[i]);
00374         cell->natoms = 0;
00375     }
00376
00377     /* Set up the grid */
00378     if ( Vclist_setupGrid(thee) == VRC_FAILURE ) {
00379         Vnm_print(2, "Vclist_ctor2:  grid setup failed!\n");
00380         return VRC_FAILURE;
00381     }
00382
00383     /* Assign atoms to grid cells */
00384     if (Vclist_assignAtoms(thee) == VRC_FAILURE) {
00385         Vnm_print(2, "Vclist_ctor2:  atom assignment failed!\n");
00386         return VRC_FAILURE;
00387     }
00388
00389
00390
00391
00392
00393     return VRC_SUCCESS;
00394 }
00395
00396 /* Destructor */
00397 VPUBLIC void Vclist_dtor(Vclist **thee) {
00398
00399     if ((*thee) != VNULL) {
00400         Vclist_dtor2(*thee);
00401         Vmem_free(VNULL, 1, sizeof(Vclist), (void **)thee);
00402         (*thee) = VNULL;
00403     }
00404
00405 }
00406
00407 /* Main (stub) destructor */

```

```

00408 VPUBLIC void Vclist_dtor2(Vclist *thee) {
00409
00410     VclistCell *cell;
00411     int i;
00412
00413     for (i=0; i<thee->n; i++) {
00414         cell = &(thee->cells[i]);
00415         VclistCell_dtor2(cell);
00416     }
00417     Vmem_free(thee->vmem, thee->n, sizeof(VclistCell),
00418             (void **)&(thee->cells));
00419     Vmem_dtor(&(thee->vmem));
00420 }
00421 }
00422
00423 VPUBLIC VclistCell* Vclist_getCell(Vclist *thee,
00424                                   double pos[VAPBS_DIM]
00425                                   ) {
00426
00427     int i,
00428         ic[VAPBS_DIM],
00429         ui;
00430     double c[VAPBS_DIM];
00431
00432     /* Assert this before we do anything else, since its failure should fail the function */
00433     VASSERT(VAPBS_DIM == 3);
00434
00435     /* Convert to grid based coordinates */
00436     for (i=0; i<VAPBS_DIM; i++) {
00437         c[i] = pos[i] - (thee->lower_corner)[i];
00438         ic[i] = (int)(c[i]/thee->spacs[i]);
00439
00440         if (ic[i] < 0 || ic[i] >= thee->npts[i]) {
00441             return VNULL;
00442         }
00443     }
00444
00445     /* Get the array index */
00446     ui = Vclist_arrayIndex(thee, ic[0], ic[1], ic[2]);
00447
00448     return &(thee->cells[ui]);
00449 }
00450 }
00451
00452 VPUBLIC VclistCell* VclistCell_ctor(int natoms) {
00453
00454     VclistCell *thee = VNULL;
00455
00456     /* Set up the structure */
00457     thee = (VclistCell*)Vmem_malloc(VNULL, 1, sizeof(VclistCell));
00458     VASSERT( thee != VNULL);
00459     VASSERT( VclistCell_ctor2(thee, natoms) == VRC_SUCCESS );
00460
00461     return thee;
00462 }
00463
00464 VPUBLIC Vrc_Codes VclistCell_ctor2(VclistCell *thee, int natoms) {
00465
00466     if (thee == VNULL) {
00467         Vnm_print(2, "VclistCell_ctor2: NULL thee!\n");
00468         return VRC_FAILURE;
00469     }
00470
00471     thee->natoms = natoms;
00472     if (thee->natoms > 0) {
00473         thee->atoms = (Vatom**)Vmem_malloc(VNULL, natoms, sizeof(Vatom *));
00474         if (thee->atoms == VNULL) {
00475             Vnm_print(2,
00476                 "VclistCell_ctor2: unable to allocate space for %d atom pointers!\n",
00477                 natoms);
00478             return VRC_FAILURE;
00479         }
00480     }
00481
00482     return VRC_SUCCESS;
00483 }
00484 }
00485
00486 VPUBLIC void VclistCell_dtor(VclistCell **thee) {
00487
00488     if ((*thee) != VNULL) {

```

```

00489     VclistCell_dtor2(*thee);
00490     Vmem_free(VNULL, 1, sizeof(VclistCell), (void **)thee);
00491     (*thee) = VNULL;
00492 }
00493
00494 }
00495
00496 /* Main (stub) destructor */
00497 VPUBLIC void VclistCell_dtor2(VclistCell *thee) {
00498
00499     if (thee->natoms > 0) {
00500         Vmem_free(VNULL, thee->natoms, sizeof(Vatom *),
00501             (void **)&(thee->atoms));
00502     }
00503 }
00504 }
00505

```

10.57 src/generic/vclist.h File Reference

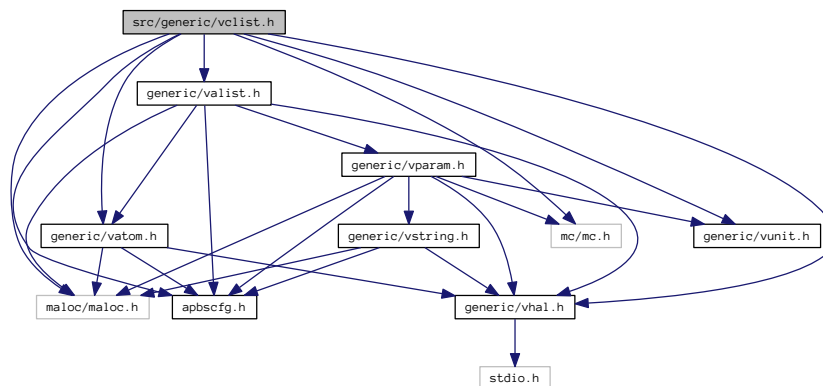
Contains declarations for class Vclist.

```

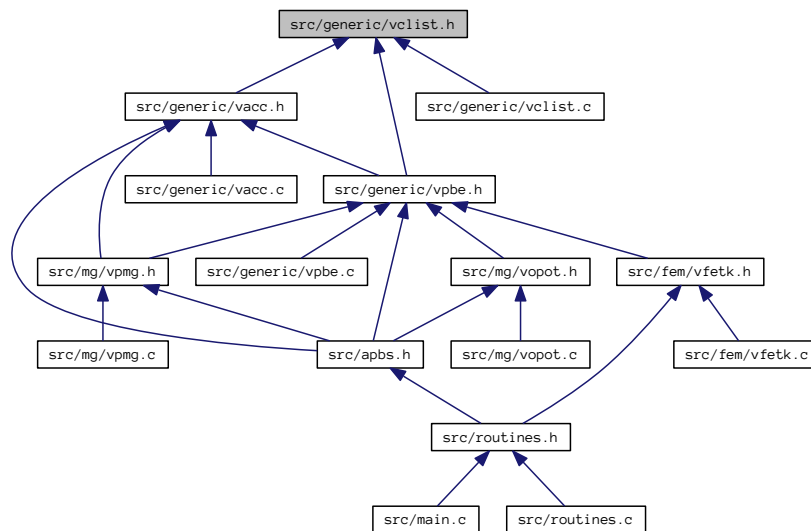
#include "apbscfg.h"
#include "malloc/malloc.h"
#include "mc/mc.h"
#include "generic/vhal.h"
#include "generic/valist.h"
#include "generic/vatom.h"
#include "generic/vunit.h"

```

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 Å) 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]`)
FORTTRAN stub to construct the cell list object.
- VEXTERNC void `Vclist_dtor` (`Vclist **thee`)
Destroy object.
- VEXTERNC void `Vclist_dtor2` (`Vclist *thee`)
FORTTRAN 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`)
FORTTRAN stub to destroy object.

10.57.1 Detailed Description

Contains declarations for class `Vclist`.

Version

\$Id\$

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
```



```

* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vclist.h](#).

10.58 vclist.h

```

00001
00062 #ifndef _VCLIST_H_
00063 #define _VCLIST_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "malloc/malloc.h"
00068 #if defined(HAVE_MC_H)
00069 #include "mc/mc.h"
00070 #endif
00071
00072 #include "generic/vhal.h"
00073 #include "generic/valist.h"
00074 #include "generic/vatom.h"
00075 #include "generic/vunit.h"
00076
00082 enum eVclist_DomainMode {
00083     CLIST_AUTO_DOMAIN,
00085     CLIST_MANUAL_DOMAIN
00087 };
00088
00094 typedef enum eVclist_DomainMode Vclist_DomainMode;
00095
00101 struct sVclistCell {
00102     Vatom **atoms;
00103     int natoms;
00104 };
00105
00110 typedef struct sVclistCell VclistCell;
00111
00117 struct sVclist {
00118
00119     Vmem *vmem;
00120     Valist *alist;
00121     Vclist_DomainMode mode;
00122     int npts[VAPBS_DIM];

```

```

00123     int n;
00124     double max_radius;
00125     VclistCell *cells;
00126     double lower_corner[VAPBS_DIM];
00127     double upper_corner[VAPBS_DIM];
00128     double spacs[VAPBS_DIM];
00130 };
00131
00136 typedef struct sVclist Vclist;
00137
00138 #if !defined(VINLINE_VCLIST)
00139
00145     VEXTERNC unsigned long int Vclist_memChk(
00146         Vclist *thee
00147     );
00148
00156     VEXTERNC double Vclist_maxRadius(
00157         Vclist *thee
00158     );
00159
00160 #else /* if defined(VINLINE_VCLIST) */
00161
00162 #   define Vclist_memChk(thee) (Vmem_bytes((thee)->vmem))
00163 #   define Vclist_maxRadius(thee) ((thee)->max_radius)
00164
00165 #endif /* if !defined(VINLINE_VCLIST) */
00166
00167 /* ////////////////////////////////////////
00168 // Class Vclist: Non-Inlineable methods (vclist.c)
00169 ////////////////////////////////////////
00170
00175 VEXTERNC Vclist* Vclist_ctor(
00176     Valist *alist,
00177     double max_radius,
00178     int npts[VAPBS_DIM],
00179     Vclist_DomainMode mode,
00180     double lower_corner[VAPBS_DIM],
00181     double upper_corner[VAPBS_DIM]
00182 );
00183
00193 VEXTERNC Vrc_Codes Vclist_ctor2(
00194     Vclist *thee,
00195     Valist *alist,
00196     double max_radius,
00197     int npts[VAPBS_DIM],
00198     Vclist_DomainMode mode,
00199     double lower_corner[VAPBS_DIM],
00200     double upper_corner[VAPBS_DIM]
00201 );
00202
00212 VEXTERNC void Vclist_dtor(
00213     Vclist **thee
00214 );
00215
00220 VEXTERNC void Vclist_dtor2(
00221     Vclist *thee
00222 );
00223
00231 VEXTERNC VclistCell* Vclist_getCell(
00232     Vclist *thee,
00233     double position[VAPBS_DIM]
00234 );
00235
00242 VEXTERNC VclistCell* VclistCell_ctor(
00243     int natoms
00244 );
00245
00252 VEXTERNC Vrc_Codes VclistCell_ctor2(
00253     VclistCell *thee,
00254     int natoms
00255 );
00256
00261 VEXTERNC void VclistCell_dtor(
00262     VclistCell **thee
00263 );
00264
00269 VEXTERNC void VclistCell_dtor2(
00270     VclistCell *thee
00271 );
00272
00273 #endif /* ifndef _VCLIST_H_ */

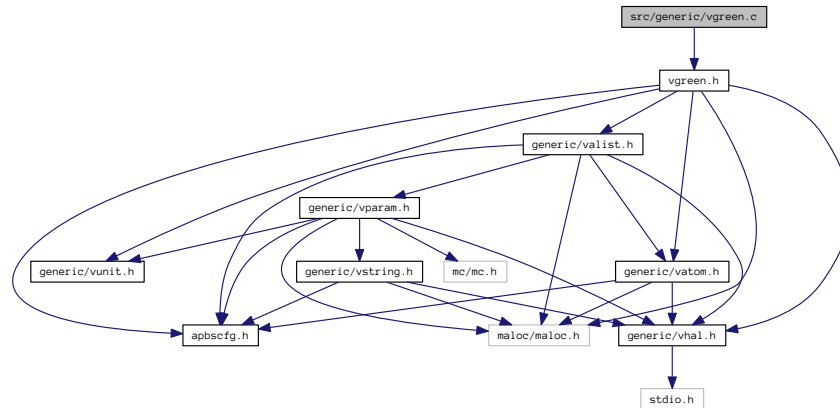
```

10.59 src/generic/vgreen.c File Reference

Class Vgreen methods.

```
#include "vgreen.h"
```

Include dependency graph for vgreen.c:



Functions

- VPRIVATE int **tree~~se~~setup** (Vgreen *thee)
- VPRIVATE int **tree~~se~~cleanup** (Vgreen *thee)
- VPRIVATE int **tree~~se~~calc** (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)

FORTTRAN 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)

FORTTRAN 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)

10.59.1 Detailed Description

Class Vgreen methods.

Author

Nathan Baker

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
```

```

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

```

Definition in file [vgreen.c](#).

10.60 vgreen.c

```

00001
00057 #include "vgreen.h"
00058
00059 /* Define wrappers for F77 treecode routines */
00060 #ifndef HAVE_TREE
00061 # define F77TREEPEFORCE VF77_MANGLE(treepeforce, TREEPEFORCE)
00062 # define F77DIRECT_ENG_FORCE VF77_MANGLE(direct_eng_force, DIRECT_ENG_FORCE)
00063 # define F77CLEANUP VF77_MANGLE(mycleanup, MYCLEANUP)
00064 # define F77TREE_COMPP VF77_MANGLE(mytree_compp, MYTREE_COMP)
00065 # define F77TREE_COMPPF VF77_MANGLE(mytree_comppf, MYTREE_COMPPF)
00066 # define F77CREATE_TREE VF77_MANGLE(mycreate_tree, MYCREATE_TREE)
00067 # define F77INITLEVELS VF77_MANGLE(myinitlevels, MYINITLEVELS)
00068 # define F77SETUP VF77_MANGLE(mysetup, MYSETUP)
00069 #endif /* ifndef HAVE_TREE */
00070
00071 /* Some constants associated with the tree code */
00072 #ifndef HAVE_TREE
00073
00076 # define FMM_DIST_TOL VSMALL
00077
00082 # define FMM_IFLAG 2
00083
00086 # define FMM_ORDER 4
00087
00090 # define FMM_THETA 0.5
00091
00094 # define FMM_MAXPARNODE 150
00095
00098 # define FMM_SHRINK 1
00099
00102 # define FMM_MINLEVEL 50000
00103
00106 # define FMM_MAXLEVEL 0
00107 #endif /* ifndef HAVE_TREE */
00108
00109
00110 /*
00111  * @brief Setup treecode internal structures
00112  * @ingroup Vgreen
00113  * @author Nathan Baker
00114  * @param thee Vgreen object
00115  * @return 1 if successful, 0 otherwise
00116  */
00117 VPRIVATE int treesetup(Vgreen *thee);
00118
00119 /*
00120  * @brief Clean up treecode internal structures
00121  * @ingroup Vgreen
00122  * @author Nathan Baker
00123  * @param thee Vgreen object
00124  * @return 1 if successful, 0 otherwise
00125  */
00126 VPRIVATE int treecleanup(Vgreen *thee);

```

```

00127
00128 /*
00129  * @brief Calculate forces or potential
00130  * @ingroup Vgreen
00131  * @author Nathan Baker
00132  * @param thee Vgreen object
00133  * @return 1 if successful, 0 otherwise
00134  */
00135 VPRIVATE int treecalcul(Vgreen *thee, double *xtar, double *ytar, double *zstar,
00136     double *qtar, int numtars, double *tpengtar, double *x, double *y,
00137     double *z, double *q, int numpars, double *fx, double *fy, double *fz,
00138     int iflag, int farrdim, int arrdim);
00139
00140 #if !defined(VINLINE_VGREEN)
00141
00142 VPUBLIC Valist* Vgreen_getValist(Vgreen *thee) {
00143
00144     VASSERT(thee != VNULL);
00145     return thee->alist;
00146 }
00147
00148
00149 VPUBLIC unsigned long int Vgreen_memChk(Vgreen *thee) {
00150     if (thee == VNULL) return 0;
00151     return Vmem_bytes(thee->vmem);
00152 }
00153
00154 #endif /* if !defined(VINLINE_VGREEN) */
00155
00156 VPUBLIC Vgreen* Vgreen_ctor(Valist *alist) {
00157
00158     /* Set up the structure */
00159     Vgreen *thee = VNULL;
00160     thee = (Vgreen *)Vmem_malloc(VNULL, 1, sizeof(Vgreen));
00161     VASSERT(thee != VNULL);
00162     VASSERT(Vgreen_ctor2(thee, alist));
00163
00164     return thee;
00165 }
00166
00167 VPUBLIC int Vgreen_ctor2(Vgreen *thee, Valist *alist) {
00168
00169     VASSERT(thee != VNULL);
00170
00171     /* Memory management object */
00172     thee->vmem = Vmem_ctor("APBS:VGREEN");
00173
00174     /* Set up the atom list and grid manager */
00175     if (alist == VNULL) {
00176         Vnm_print(2, "Vgreen_ctor2: got null pointer to Valist object!\n");
00177     }
00178
00179     thee->alist = alist;
00180
00181     /* Setup FMM tree (if applicable) */
00182     #ifdef HAVE_TREE
00183     if (!treesetup(thee)) {
00184         Vnm_print(2, "Vgreen_ctor2: Error setting up FMM tree!\n");
00185         return 0;
00186     }
00187     #endif /* if !defined(HAVE_TREE) */
00188
00189     return 1;
00190 }
00191
00192 VPUBLIC void Vgreen_dtor(Vgreen **thee) {
00193     if ((*thee) != VNULL) {
00194         Vgreen_dtor2(*thee);
00195         Vmem_free(VNULL, 1, sizeof(Vgreen), (void **)thee);
00196         (*thee) = VNULL;
00197     }
00198 }
00199
00200 VPUBLIC void Vgreen_dtor2(Vgreen *thee) {
00201
00202     #ifdef HAVE_TREE
00203     treecleanup(thee);
00204     #endif
00205     Vmem_dtor(&(thee->vmem));
00206 }
00207

```

```

00208
00209 VPUBLIC int Vgreen_helmholtz(Vgreen *thee, int npos, double *x, double *y,
00210     double *z, double *val, double kappa) {
00211
00212     Vnm_print(2, "Error -- Vgreen_helmholtz not implemented yet!\n");
00213     return 0;
00214 }
00215
00216 VPUBLIC int Vgreen_helmholtzD(Vgreen *thee, int npos, double *x, double *y,
00217     double *z, double *gradx, double *grady, double *gradz, double kappa) {
00218
00219     Vnm_print(2, "Error -- Vgreen_helmholtzD not implemented yet!\n");
00220     return 0;
00221 }
00222
00223 VPUBLIC int Vgreen_coulomb_direct(Vgreen *thee, int npos, double *x,
00224     double *y, double *z, double *val) {
00225
00226     Vatom *atom;
00227     double *apos, charge, dist, dx, dy, dz, scale;
00228     double *q, qtemp, fx, fy, fz;
00229     int iatom, ipos;
00230
00231     if (thee == VNULL) {
00232         Vnm_print(2, "Vgreen_coulomb: Got NULL thee!\n");
00233         return 0;
00234     }
00235
00236     for (ipos=0; ipos<npos; ipos++) val[ipos] = 0.0;
00237
00238     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
00239         atom = Valist_getAtom(thee->alist, iatom);
00240         apos = Vatom_getPosition(atom);
00241         charge = Vatom_getCharge(atom);
00242         for (ipos=0; ipos<npos; ipos++) {
00243             dx = apos[0] - x[ipos];
00244             dy = apos[1] - y[ipos];
00245             dz = apos[2] - z[ipos];
00246             dist = VSQRT(VSQR(dx) + VSQR(dy) + VSQR(dz));
00247             if (dist > VSMALL) val[ipos] += (charge/dist);
00248         }
00249     }
00250
00251     scale = Vunit_ec/(4*Vunit_pi*Vunit_eps0*1.0e-10);
00252     for (ipos=0; ipos<npos; ipos++) val[ipos] = val[ipos]*scale;
00253
00254     return 1;
00255 }
00256
00257 VPUBLIC int Vgreen_coulomb(Vgreen *thee, int npos, double *x, double *y,
00258     double *z, double *val) {
00259
00260     Vatom *atom;
00261     double *apos, charge, dist, dx, dy, dz, scale;
00262     double *q, qtemp, fx, fy, fz;
00263     int iatom, ipos;
00264
00265     if (thee == VNULL) {
00266         Vnm_print(2, "Vgreen_coulomb: Got NULL thee!\n");
00267         return 0;
00268     }
00269
00270     for (ipos=0; ipos<npos; ipos++) val[ipos] = 0.0;
00271
00272     #ifdef HAVE_TREE
00273     /* Allocate charge array (if necessary) */
00274     if (Valist_getNumberAtoms(thee->alist) > 1) {
00275         if (npos > 1) {
00276             q = VNULL;
00277             q = Vmem_malloc(thee->vmem, npos, sizeof(double));
00278             if (q == VNULL) {
00279                 Vnm_print(2, "Vgreen_coulomb: Error allocating charge array!\n");
00280                 return 0;
00281             }
00282         } else {
00283             q = &(qtemp);
00284         }
00285         for (ipos=0; ipos<npos; ipos++) q[ipos] = 1.0;
00286     }
00287
00288

```

```

00289      /* Calculate */
00290      treecalc(thee, x, y, z, q, npos, val, thee->xp, thee->yp, thee->zp,
00291      thee->qp, thee->np, &fx, &fy, &fz, 1, 1, thee->np);
00292  } else return Vgreen_coulomb_direct(thee, npos, x, y, z, val);
00293
00294  /* De-allocate charge array (if necessary) */
00295  if (npos > 1) Vmem_free(thee->vmem, npos, sizeof(double), (void **)&q);
00296
00297  scale = Vunit_ec/(4*Vunit_pi*Vunit_eps0*1.0e-10);
00298  for (ipos=0; ipos<npos; ipos++) val[ipos] = val[ipos]*scale;
00299
00300  return 1;
00301
00302 #else /* ifdef HAVE_TREE */
00303
00304  return Vgreen_coulomb_direct(thee, npos, x, y, z, val);
00305
00306 #endif
00307 }
00308 }
00309
00310 VPUBLIC int Vgreen_coulombD_direct(Vgreen *thee, int npos,
00311 double *x, double *y, double *z, double *pot, double *gradx,
00312 double *grady, double *gradz) {
00313
00314  Vatom *atom;
00315  double *apos, charge, dist, dist2, idist3, dy, dz, dx, scale;
00316  double *q, qtemp;
00317  int iatom, ipos;
00318
00319  if (thee == VNULL) {
00320      Vnm_print(2, "Vgreen_coulombD: Got VNULL thee!\n");
00321      return 0;
00322  }
00323
00324  for (ipos=0; ipos<npos; ipos++) {
00325      pot[ipos] = 0.0;
00326      gradx[ipos] = 0.0;
00327      grady[ipos] = 0.0;
00328      gradz[ipos] = 0.0;
00329  }
00330
00331  for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
00332      atom = Valist_getAtom(thee->alist, iatom);
00333      apos = Vatom_getPosition(atom);
00334      charge = Vatom_getCharge(atom);
00335      for (ipos=0; ipos<npos; ipos++) {
00336          dx = apos[0] - x[ipos];
00337          dy = apos[1] - y[ipos];
00338          dz = apos[2] - z[ipos];
00339          dist2 = VSQR(dx) + VSQR(dy) + VSQR(dz);
00340          dist = VSQRT(dist2);
00341          if (dist > VSMALL) {
00342              idist3 = 1.0/(dist*dist2);
00343              gradx[ipos] -= (charge*dx*idist3);
00344              grady[ipos] -= (charge*dy*idist3);
00345              gradz[ipos] -= (charge*dz*idist3);
00346              pot[ipos] += (charge/dist);
00347          }
00348      }
00349  }
00350
00351  scale = Vunit_ec/(4*VPI*Vunit_eps0*(1.0e-10));
00352  for (ipos=0; ipos<npos; ipos++) {
00353      gradx[ipos] = gradx[ipos]*scale;
00354      grady[ipos] = grady[ipos]*scale;
00355      gradz[ipos] = gradz[ipos]*scale;
00356      pot[ipos] = pot[ipos]*scale;
00357  }
00358
00359  return 1;
00360 }
00361
00362 VPUBLIC int Vgreen_coulombD(Vgreen *thee, int npos, double *x, double *y,
00363 double *z, double *pot, double *gradx, double *grady, double *gradz) {
00364
00365  Vatom *atom;
00366  double *apos, charge, dist, dist2, idist3, dy, dz, dx, scale;
00367  double *q, qtemp;
00368  int iatom, ipos;
00369

```



```

00370     if (thee == VNULL) {
00371         Vnm_print(2, "Vgreen_coulombD: Got VNULL thee!\n");
00372         return 0;
00373     }
00374
00375     for (ipos=0; ipos<npos; ipos++) {
00376         pot[ipos] = 0.0;
00377         gradx[ipos] = 0.0;
00378         grady[ipos] = 0.0;
00379         gradz[ipos] = 0.0;
00380     }
00381
00382 #ifdef HAVE_TREE
00383
00384     if (Valist_getNumberAtoms(thee->alist) > 1) {
00385         if (npos > 1) {
00386             q = VNULL;
00387             q = Vmem_malloc(thee->vmem, npos, sizeof(double));
00388             if (q == VNULL) {
00389                 Vnm_print(2, "Vgreen_coulomb: Error allocating charge array!\n");
00390                 return 0;
00391             }
00392         } else {
00393             q = &(qtemp);
00394         }
00395         for (ipos=0; ipos<npos; ipos++) q[ipos] = 1.0;
00396
00397         /* Calculate */
00398         treecalc(thee, x, y, z, q, npos, pot, thee->xp, thee->yp, thee->zp,
00399                 thee->qp, thee->np, gradx, grady, gradz, 2, npos, thee->np);
00400
00401         /* De-allocate charge array (if necessary) */
00402         if (npos > 1) Vmem_free(thee->vmem, npos, sizeof(double), (void **)&q);
00403     } else return Vgreen_coulombD_direct(thee, npos, x, y, z, pot,
00404                                         gradx, grady, gradz);
00405
00406     scale = Vunit_ec/(4*VPI*Vunit_eps0*(1.0e-10));
00407     for (ipos=0; ipos<npos; ipos++) {
00408         gradx[ipos] = gradx[ipos]*scale;
00409         grady[ipos] = grady[ipos]*scale;
00410         gradz[ipos] = gradz[ipos]*scale;
00411         pot[ipos] = pot[ipos]*scale;
00412     }
00413
00414     return 1;
00415
00416 #else /* ifdef HAVE_TREE */
00417
00418     return Vgreen_coulombD_direct(thee, npos, x, y, z, pot,
00419                                   gradx, grady, gradz);
00420
00421 #endif
00422 }
00423
00424
00425 VPRIVATE int treesetup(Vgreen *thee) {
00426
00427 #ifdef HAVE_TREE
00428
00429     double dist_tol = FMM_DIST_TOL;
00430     int iflag = FMM_IFLAG;
00431     double order = FMM_ORDER;
00432     int theta = FMM_THETA;
00433     int shrink = FMM_SHRINK;
00434     int maxparnode = FMM_MAXPARNODE;
00435     int minlevel = FMM_MINLEVEL;
00436     int maxlevel = FMM_MAXLEVEL;
00437     int level = 0;
00438     int one = 1;
00439     Vatom *atom;
00440     double xyzminmax[6], *pos;
00441     int i;
00442
00443     /* Set up particle arrays with atomic coordinates and charges */
00444     Vnm_print(0, "treesetup: Initializing FMM particle arrays...\n");
00445     thee->np = Valist_getNumberAtoms(thee->alist);
00446     thee->xp = VNULL;
00447     thee->xp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(double));
00448     if (thee->xp == VNULL) {
00449         Vnm_print(2, "Vgreen_ctor2: Failed to allocate %d*sizeof(double)!\n",
00450                 thee->np);

```

```

00451         return 0;
00452     }
00453     thee->yp = VNULL;
00454     thee->yp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(double));
00455     if (thee->yp == VNULL) {
00456         Vnm_print(2, "Vgreen_ctor2: Failed to allocate %d*sizeof(double)!\n",
00457             thee->np);
00458         return 0;
00459     }
00460     thee->zp = VNULL;
00461     thee->zp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(double));
00462     if (thee->zp == VNULL) {
00463         Vnm_print(2, "Vgreen_ctor2: Failed to allocate %d*sizeof(double)!\n",
00464             thee->np);
00465         return 0;
00466     }
00467     thee->qp = VNULL;
00468     thee->qp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(double));
00469     if (thee->qp == VNULL) {
00470         Vnm_print(2, "Vgreen_ctor2: Failed to allocate %d*sizeof(double)!\n",
00471             thee->np);
00472         return 0;
00473     }
00474     for (i=0; i<thee->np; i++) {
00475         atom = Valist_getAtom(thee->alist, i);
00476         pos = Vatom_getPosition(atom);
00477         thee->xp[i] = pos[0];
00478         thee->yp[i] = pos[1];
00479         thee->zp[i] = pos[2];
00480         thee->qp[i] = Vatom_getCharge(atom);
00481     }
00482
00483     Vnm_print(0, "treesetup: Setting things up...\n");
00484     F77SETUP(thee->xp, thee->yp, thee->zp, &(thee->np), &order, &theta, &iflag,
00485         &dist_tol, xyzminmax, &(thee->np));
00486
00487     Vnm_print(0, "treesetup: Initializing levels...\n");
00488     F77INITLEVELS(&minlevel, &maxlevel);
00489
00490     Vnm_print(0, "treesetup: Creating tree...\n");
00491     F77CREATE_TREE(&one, &(thee->np), thee->xp, thee->yp, thee->zp, thee->
00492     qp,
00493         &shrink, &maxparnode, xyzminmax, &level, &(thee->np));
00494
00495     return 1;
00496
00497 #else /* ifdef HAVE_TREE */
00498
00499     Vnm_print(2, "treesetup: Error! APBS not linked with treecode!\n");
00500     return 0;
00501
00502 #endif /* ifdef HAVE_TREE */
00503 }
00504
00505 VPRIVATE int treecleanup(Vgreen *thee) {
00506
00507 #ifdef HAVE_TREE
00508
00509     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->xp));
00510     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->yp));
00511     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->zp));
00512     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->qp));
00513     F77CLEANUP();
00514
00515     return 1;
00516
00517 #else /* ifdef HAVE_TREE */
00518
00519     Vnm_print(2, "treecleanup: Error! APBS not linked with treecode!\n");
00520     return 0;
00521
00522 #endif /* ifdef HAVE_TREE */
00523 }
00524
00525 VPRIVATE int treecalc(Vgreen *thee, double *xtar, double *ytar, double *ztar,
00526     double *qtar, int numtars, double *tpengtars, double *x, double *y,
00527     double *z, double *q, int numpars, double *fx, double *fy, double *fz,
00528     int iflag, int farrdim, int arrdim) {
00529
00530 #ifdef HAVE_TREE

```

```

00531     int i, level, err, maxlevel, minlevel, one;
00532     double xyzminmax[6];
00533
00534
00535     if (iflag != 1) {
00536         F77TREE_COMPP(xtar, ytar, ztar, qtar, &numtars, tpengtar, x, y, z, q,
00537             fx, fy, fz, &numpars, &farrrdim, &arrdim);
00538     } else {
00539         F77TREE_COMPP(xtar, ytar, ztar, qtar, &numtars, tpengtar, &farrrdim, x,
00540             y, z, q, &numpars, &arrdim);
00541     }
00542
00543
00544     return 1;
00545
00546 #else /* ifdef HAVE_TREE */
00547
00548     Vnm_print(2, "treecalc: Error! APBS not linked with treecode!\n");
00549     return 0;
00550
00551 #endif /* ifdef HAVE_TREE */
00552 }
00553

```

10.61 src/generic/vgreen.h File Reference

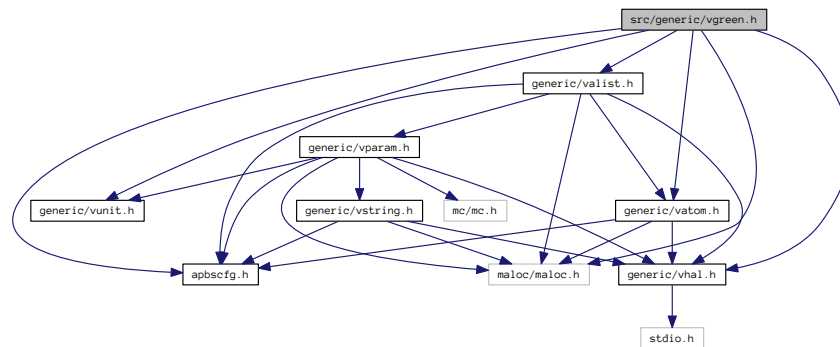
Contains declarations for class Vgreen.

```

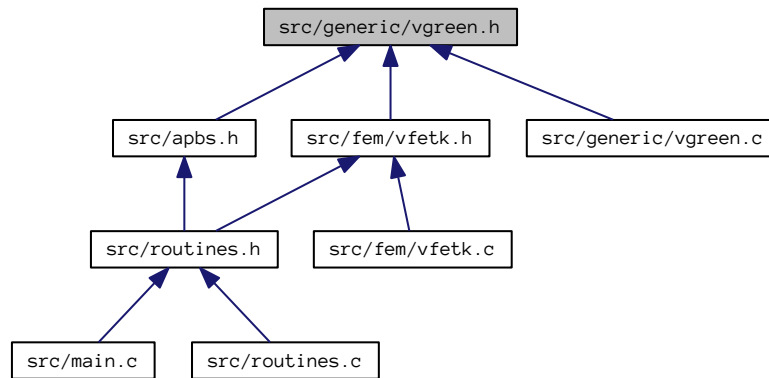
#include "apbscfg.h"
#include "malloc/malloc.h"
#include "generic/vhal.h"
#include "generic/vunit.h"
#include "generic/vatom.h"
#include "generic/valist.h"

```

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)
FORTTRAN 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)
FORTTRAN 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)

10.61.1 Detailed Description

Contains declarations for class [Vgreen](#).

Version

\$Id\$

Author

Nathan A. Baker

Definition in file [vgreen.h](#).

10.62 vgreen.h

```

00001
00065 #ifndef _VGREEN_H_
00066 #define _VGREEN_H_
00067
00068 #include "apbscfg.h"
00069
00070 #include "malloc/malloc.h"
00071
00072 #include "generic/vhal.h"
00073 #include "generic/vunit.h"
00074 #include "generic/vatom.h"
00075 #include "generic/valist.h"
00076
00082 struct sVgreen {
00083
00084     Valist *alist;
00085     Vmem *vmem;
00086     double *xp;
00088     double *yp;
00090     double *zp;
00092     double *qp;
00094     int np;
00095 };
00096
00101 typedef struct sVgreen Vgreen;

```

```

00102
00103 /* ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////// */
00104 // Class Vgreen: Inlineable methods (vgreen.c)
00106
00107 #if !defined(VINLINE_VGREEN)
00108
00116     VEXTERNC Valist* Vgreen_getValist(Vgreen *thee);
00117
00125     VEXTERNC unsigned long int Vgreen_memChk(Vgreen *thee);
00126
00127 #else /* if defined(VINLINE_VGREEN) */
00128 #   define Vgreen_getValist(thee) ((thee)->alist)
00129 #   define Vgreen_memChk(thee) (Vmem_bytes((thee)->vmem))
00130 #endif /* if !defined(VINLINE_VGREEN) */
00131
00132 /* ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////// */
00133 // Class Vgreen: Non-Inlineable methods (vgreen.c)
00135
00142 VEXTERNC Vgreen* Vgreen_ctor(Valist *alist);
00143
00151 VEXTERNC int Vgreen_ctor2(Vgreen *thee, Valist *alist);
00152
00158 VEXTERNC void Vgreen_dtor(Vgreen **thee);
00159
00165 VEXTERNC void Vgreen_dtor2(Vgreen *thee);
00166
00191 VEXTERNC int Vgreen_helmholtz(Vgreen *thee, int npos, double *x, double *y,
00192     double *z, double *val, double kappa);
00193
00221 VEXTERNC int Vgreen_helmholtzD(Vgreen *thee, int npos, double *x, double *y,
00222     double *z, double *gradx, double *grady, double *gradz, double kappa);
00223
00244 VEXTERNC int Vgreen_coulomb_direct(Vgreen *thee, int npos, double *x,
00245     double *y, double *z, double *val);
00246
00267 VEXTERNC int Vgreen_coulomb(Vgreen *thee, int npos, double *x, double *y,
00268     double *z, double *val);
00269
00293 VEXTERNC int Vgreen_coulombD_direct(Vgreen *thee, int npos, double *x,
00294     double *y, double *z, double *pot, double *gradx, double *grady, double
00295     *gradz);
00296
00321 VEXTERNC int Vgreen_coulombD(Vgreen *thee, int npos, double *x, double *y,
00322     double *z, double *pot, double *gradx, double *grady, double *gradz);
00323
00324 #endif /* ifndef _VGREEN_H_ */

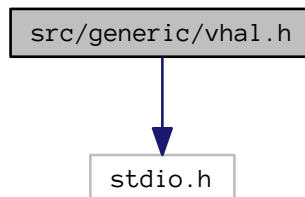
```

10.63 src/generic/vhal.h File Reference

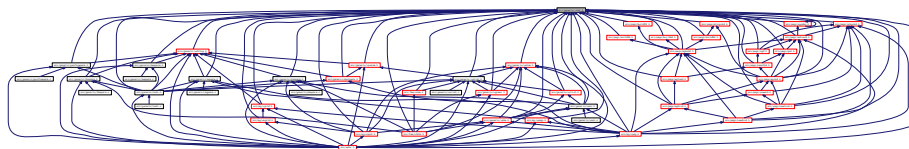
Contains generic macro definitions for APBS.

#include "stdio.h"

Include dependency graph for vhal.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#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-zero (etc.)*
#define **SINH_MIN** -85.0
- *Used to set the min values acceptable for sinh chopping.*
#define **SINH_MAX** 85.0
- *Used to set the max values acceptable for sinh chopping.*
#define **MAX_HASH_DIM** 75
- #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.
- #define **PRINT_FUNC** __PRETTY_FUNCTION__
- #define **OS_SEP_STR** "/"
- #define **OS_SEP_CHAR** '/'
- #define **ANNOUNCE_FUNCTION**
- #define **WARN_UNTESTED**
- #define **WARN_PARTTESTED**
- #define **VCHANNELEDMESSAGE0**(channel, msg)
- #define **VCHANNELEDMESSAGE1**(channel, msg, arg0)
- #define **VCHANNELEDMESSAGE2**(channel, msg, arg0, arg1)
- #define **VCHANNELEDMESSAGE3**(channel, msg, arg0, arg1, arg2)
- #define **VMESSAGE0**(msg) VCHANNELEDMESSAGE0(0, msg)
- #define **VMESSAGE1**(msg, arg0) VCHANNELEDMESSAGE1(0, msg, arg0)
- #define **VMESSAGE2**(msg, arg0, arg1) VCHANNELEDMESSAGE2(0, msg, arg0, arg1)
- #define **VMESSAGE3**(msg, arg0, arg1, arg2) VCHANNELEDMESSAGE3(0, msg, arg0, arg1, arg2)
- #define **VERRMSG0**(msg) VCHANNELEDMESSAGE0(2, msg)
- #define **VERRMSG1**(msg, arg0) VCHANNELEDMESSAGE1(2, msg, arg0)
- #define **VERRMSG2**(msg, arg0, arg1) VCHANNELEDMESSAGE2(2, msg, arg0, arg1)
- #define **VERRMSG3**(msg, arg0, arg1, arg2) VCHANNELEDMESSAGE3(2, msg, arg0, arg1, arg2)
- #define **VASSERT_MSG0**(cnd, msg)
- #define **VASSERT_MSG1**(cnd, msg, arg)
- #define **VASSERT_MSG2**(cnd, msg, arg0, arg1)
- #define **VWARN_MSG0**(cnd, msg)
- #define **VWARN_MSG1**(cnd, msg, arg0)
- #define **VWARN_MSG2**(cnd, msg, arg0, arg1)
- #define **VABORT_MSG0**(msg)
- #define **VABORT_MSG1**(msg, arg)
- #define **VABORT_MSG2**(msg, arg0, arg1)
- #define **PRINT_INT**(expr)
- #define **PRINT_DBL**(expr)
- #define **VMALLOC**(vmem, n, type) ((type*)Vmem_malloc(vmem, n, sizeof(type)))
- #define **VFREE**(vmem, n, type, ptr) (Vmem_free(vmem, n, sizeof(type), (void **)&(ptr)))
- #define **VFILL**(vec, n, val)
- #define **VCOPY**(srcvec, dstvec, i, n)
- #define **VAT**(array, i) ((array)[i - 1])
- #define **RAT**(array, i) ((array) + i - 1)

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.

Functions

- `char * wrap_text` (`char *str`, `int right_margin`, `int left_padding`)

Variables

- `FILE * data`

10.63.1 Detailed Description

Contains generic macro definitions for APBS.

Version

\$Id\$

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) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* 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](#).

10.63.2 Macro Definition Documentation

10.63.2.1 #define PRINT_FUNC __PRETTY_FUNCTION__

OS specific flags and etcetera

Definition at line 578 of file [vhal.h](#).

10.63.2.2 #define VABORT_MSG0(msg)

Value:

```

do {
    Vnm_print(2, "[%s()]: ABORTING:\n" \
              "    %s\n", \
              __FUNCTION__, msg); \
    abort();
} while (0)

```

Definition at line 906 of file [vhal.h](#).

10.63.2.3 #define VABORT_MSG1(msg, arg)

Value:

```

do {
    char buff[1000];

```

```

    snprintf( buff, 1000, msg, arg ); \
    Vnm_print(2, "[%s()]: ABORTING:\n" \
               "    %s\n\n", \
               __FUNCTION__, buff); \
    abort(); \
} while(0)

```

Definition at line 914 of file [vhal.h](#).

10.63.2.4 #define VABORT_MSG2(msg, arg0, arg1)

Value:

```

do {
    char buff[1000]; \
    snprintf( buff, 1000, msg, arg0, arg1); \
    Vnm_print(2, "[%s()]: ABORTING:\n" \
               "    %s\n\n", \
               __FUNCTION__, buff); \
    abort(); \
} while(0)

```

Definition at line 924 of file [vhal.h](#).

10.63.2.5 #define VASSERT_MSG0(cnd, msg)

Value:

```

do {
    if( (cnd) == 0 ) { \
        Vnm_print(2, "[%s()]: ERROR:\n" \
                   "    Assertion Failed (%s): %s\n\n", \
                   __FUNCTION__, #cnd, msg); \
        abort(); \
    } \
} while(0)

```

Definition at line 738 of file [vhal.h](#).

10.63.2.6 #define VASSERT_MSG1(cnd, msg, arg)

Value:

```

do {
    if( (cnd) == 0 ) { \
        char buff[1000]; \
        snprintf( buff, 1000, msg, arg ); \
        Vnm_print(2, "[%s()]: ERROR:\n" \
                   "    Assertion Failed (%s): %s\n\n", \
                   __FUNCTION__, #cnd, buff); \
        abort(); \
    } \
} while(0)

```

Definition at line 748 of file [vhal.h](#).

10.63.2.7 #define VASSERT_MSG2(cnd, msg, arg0, arg1)

Value:

```

do {
    if( (cnd) == 0 ) {
        char buff[1000];
        snprintf( buff, 1000, msg, arg0, arg1 );
        Vnm_print(2, "[%s()]: ERROR:\n"
                  "    Assertion Failed (%s): %s\n\n",
                  __FUNCTION__, #cnd, buff);
        abort();
    }
} while(0)

```

Definition at line 760 of file [vhal.h](#).

10.63.2.8 #define VCHANNELEDMESSAGE0(channel, msg)

Value:

```

do {
    Vnm_print(channel, "%s: %s\n", __FUNCTION__, msg); \
} while(0)

```

Definition at line 660 of file [vhal.h](#).

10.63.2.9 #define VCHANNELEDMESSAGE1(channel, msg, arg0)

Value:

```

do {
    char buff[1000];
    snprintf( buff, 1000, msg, arg0 );
    Vnm_print(channel, "%s: %s\n", __FUNCTION__, buff); \
} while(0)

```

Definition at line 665 of file [vhal.h](#).

10.63.2.10 #define VCHANNELEDMESSAGE2(channel, msg, arg0, arg1)

Value:

```

do {
    char buff[1000];
    snprintf( buff, 1000, msg, arg0, arg1 );
    Vnm_print(channel, "%s: %s\n", __FUNCTION__, buff); \
} while(0)

```

Definition at line 672 of file [vhal.h](#).

10.63.2.11 #define VCHANNELEDMESSAGE3(channel, msg, arg0, arg1, arg2)

Value:

```

do {
    char buff[1000];
    snprintf(buff, 1000, msg, arg0, arg1, arg2);
    Vnm_print(channel, "%s: %s\n", __FUNCTION__, buff); \
} while(0)

```

Definition at line 679 of file [vhal.h](#).

10.63.2.12 #define VCOPY(*srcvec*, *dstvec*, *i*, *n*)**Value:**

```
do {
    for (i = 0; i < n; i++) \
        dstvec[i] = srcvec[i]; \
} while(0)
```

Definition at line 966 of file [vhal.h](#).

10.63.2.13 #define VFILL(*vec*, *n*, *val*)**Value:**

```
do {
    int fill_idx;
    for (fill_idx = 0; fill_idx < n; fill_idx++) \
        vec[fill_idx] = val; \
} while(0)
```

Definition at line 959 of file [vhal.h](#).

10.63.2.14 #define VWARN_MSG0(*cnd*, *msg*)**Value:**

```
do {
    if( (cnd) == 0 ) {
        Vnm_print(
            2,
            "[%s()]: WARNING:\n"
            "    %s\n\n",
            __FUNCTION__,
            msg
        );
    }
} while(0)
```

Definition at line 832 of file [vhal.h](#).

10.63.2.15 #define VWARN_MSG1(*cnd*, *msg*, *arg0*)**Value:**

```
do {
    if( (cnd) == 0 ) {
        char buff[1000];
        snprintf(buff, 1000, msg, arg0);
        Vnm_print(
            2,
            "[%s()]: WARNING:\n"
            "    %s\n\n",
            __FUNCTION__,
            buff
        );
    }
} while(0)
```

Definition at line 845 of file [vhal.h](#).

10.63.2.16 `#define VWARN_MSG2(cnd, msg, arg0, arg1)`

Value:

```
do {
    if( (cnd) == 0 ) {
        char buff[1000];
        snprintf(buff, 1000, msg, arg0, arg1);
        Vnm_print(
            2,
            "[%s()]: WARNING:\n"
            "    %s\n\n",
            __FUNCTION__,
            buff
        );
    }
} while (0)
```

Definition at line 860 of file `vhal.h`.

10.64 vhal.h

```
00001
00055 #ifndef _VAPBSHAL_H_
00056 #define _VAPBSHAL_H_
00057
00058 #include "stdio.h"
00059
00060 FILE *data;
00061
00068 enum eVrc_Codes {
00069     VRC_WARNING=-1,
00071     VRC_FAILURE=0,
00072     VRC_SUCCESS=1
00074 };
00075 typedef enum eVrc_Codes Vrc_Codes;
00076
00083 enum eVsol_Meth {
00084     VSOL_CGMG, /* 0: conjugate gradient multigrid */
00086     VSOL_Newton, /* 1: newton */
00087     VSOL_MG, /* 2: multigrid */
00088     VSOL_CG, /* 3: conjugate gradient */
00089     VSOL_SOR, /* 4: successive overrelaxation */
00090     VSOL_RBGS, /* 5: red-black gauss-seidel */
00091     VSOL_WJ, /* 6: weighted jacobi */
00092     VSOL_Richardson, /* 7: richardson */
00093     VSOL_CGMGAqua, /* 8: conjugate gradient multigrid aqua */
00094     VSOL_NewtonAqua /* 9: newton aqua */
00095 };
00096 };
00097 typedef enum eVsol_Meth Vsol_Meth;
00098
00104 enum eVsurf_Meth {
00105     VSM_MOL=0,
00109     VSM_MOLSMOOTH=1,
00111     VSM_SPLINE=2,
00121     VSM_SPLINE3=3,
00125     VSM_SPLINE4=4
00129 };
00130
00135 typedef enum eVsurf_Meth Vsurf_Meth;
00136
00141 enum eVhal_PBEType {
00142     PBE_LPBE,
00143     PBE_NPBE,
00144     PBE_LRPBE,
00145     PBE_NRPBE,
00146     PBE_SMPBE
00147 };
00148
00153 typedef enum eVhal_PBEType Vhal_PBEType;
00154
```

```
00159 enum eVhal_IPKEYType {
00160     IPKEY_SMPBE = -2,
00161     IPKEY_LPBE,
00162     IPKEY_NPBE
00163 };
00164
00169 typedef enum eVhal_IPKEYType Vhal_IPKEYType;
00170
00175 enum eVhal_NONLINType {
00176     NONLIN_LPBE = 0,
00177     NONLIN_NPBE,
00178     NONLIN_SMPBE,
00179     NONLIN_LPBEAQUA,
00180     NONLIN_NPBEAQUA
00181 };
00182
00187 typedef enum eVhal_NONLINType Vhal_NONLINType;
00188
00193 enum eVoutput_Format {
00194     OUTPUT_NULL,
00195     OUTPUT_FLAT,
00196 };
00197
00202 typedef enum eVoutput_Format Voutput_Format;
00203
00209 enum eVbcfl {
00210     BCFL_ZERO=0,
00211     BCFL_SDH=1,
00213     BCFL_MDH=2,
00215     BCFL_UNUSED=3,
00216     BCFL_FOCUS=4,
00217     BCFL_MEM=5,
00218     BCFL_MAP=6
00219 };
00220
00225 typedef enum eVbcfl Vbcfl;
00226
00232 enum eVchrg_Meth {
00233     VCM_TRIL=0,
00236     VCM_BSPL2=1,
00239     VCM_BSPL4=2
00240 };
00241
00246 typedef enum eVchrg_Meth Vchrg_Meth;
00247
00253 enum eVchrg_Src {
00254     VCM_CHARGE=0,
00255     VCM_PERMANENT=1,
00256     VCM_INDUCED=2,
00257     VCM_NLINDUCED=3
00258 };
00259
00264 typedef enum eVchrg_Src Vchrg_Src;
00265
00271 enum eVdata_Type {
00272     VDT_CHARGE,
00273     VDT_POT,
00274     VDT_ATOMPOT,
00275     VDT_SMOL,
00277     VDT_SSPL,
00279     VDT_VDW,
00281     VDT_IVDW,
00283     VDT_LAP,
00284     VDT_EDENS,
00286     VDT_NDENS,
00288     VDT_QDENS,
00290     VDT_DIELX,
00292     VDT_DIELY,
00294     VDT_DIELZ,
00296     VDT_KAPPA
00298 };
00299
00304 typedef enum eVdata_Type Vdata_Type;
00305
00311 enum eVdata_Format {
00312     VDF_DX=0,
00313     VDF_UHBD=1,
00314     VDF_AVS=2,
00315     VDF_MCSF=3,
00316     VDF_GZ=4,
00317     VDF_FLAT=5
```



```
00318 };
00319
00324 typedef enum eVdata_Format Vdata_Format;
00325
00330 #define APBS_TIMER_WALL_CLOCK 26
00331
00336 #define APBS_TIMER_SETUP 27
00337
00342 #define APBS_TIMER_SOLVER 28
00343
00348 #define APBS_TIMER_ENERGY 29
00349
00354 #define APBS_TIMER_FORCE 30
00355
00360 #define APBS_TIMER_TEMP1 31
00361
00366 #define APBS_TIMER_TEMP2 32
00367
00372 #define MAXMOL 5
00373
00378 #define MAXION 10
00379
00383 #define MAXFOCUS 5
00384
00388 #define VMGNLEV 4
00389
00393 #define VREDFRAC 0.25
00394
00398 #define VAPBS_NVS 4
00399
00403 #define VAPBS_DIM 3
00404
00409 #define VAPBS_RIGHT 0
00410
00415 #define MAX_SPHERE_PTS 50000
00416
00421 #define VAPBS_FRONT 1
00422
00427 #define VAPBS_UP 2
00428
00433 #define VAPBS_LEFT 3
00434
00439 #define VAPBS_BACK 4
00440
00445 #define VAPBS_DOWN 5
00446
00451 #define VPMGSMALL 1e-12
00452
00457 #define SINH_MIN -85.0
00458
00463 #define SINH_MAX 85.0
00464
00465 #define MAX_HASH_DIM 75
00466
00467 #if defined(VDEBUG)
00468 #   if !defined(APBS_NOINLINE)
00469 #       define APBS_NOINLINE 1
00470 #   endif
00471 #endif
00472
00473 #if !defined(APBS_NOINLINE)
00474 #   define VINLINE_VACC
00475 #   define VINLINE_VATOM
00476 #   define VINLINE_VCSM
00477 #   define VINLINE_VPBE
00478 #   define VINLINE_VPEE
00479 #   define VINLINE_VGREEN
00480 #   define VINLINE_VFETK
00481 #   define VINLINE_VPMG
00482 #endif
00483
00488 #   define VINLINE_VACC
00489 #   define VINLINE_VATOM
00490 #   define VINLINE_VCSM
00491 #   define VINLINE_VPBE
00492 #   define VINLINE_VPEE
00493 #   define VINLINE_VGREEN
00494 #   define VINLINE_VFETK
00495 #   define VINLINE_VPMG
00496 #endif
00497
00503 #   define VINLINE_VACC
00504 #   define VINLINE_VATOM
00505 #   define VINLINE_VCSM
00506 #   define VINLINE_VPBE
00507 #   define VINLINE_VPEE
00508 #   define VINLINE_VGREEN
00509 #   define VINLINE_VFETK
00510 #   define VINLINE_VPMG
00511 #endif
00512
00513 #   define VINLINE_VACC
00514 #   define VINLINE_VATOM
00515 #   define VINLINE_VCSM
00516 #   define VINLINE_VPBE
00517 #   define VINLINE_VPEE
00518 #   define VINLINE_VGREEN
00519 #   define VINLINE_VFETK
00520 #   define VINLINE_VPMG
00521 #endif
00522
00521 /* Fortran name mangling */
```

```

00522 #if defined(VF77_UPPERCASE)
00523 #   if defined(VF77_NOUNDERSCORE)
00524 #       define VF77_MANGLE(name,NAME) NAME
00525 #   elif defined(VF77_ONEUNDERSCORE)
00526 #       define VF77_MANGLE(name,NAME) NAME ## _
00527 #   else
00528 #       define VF77_MANGLE(name,NAME) name
00529 #   endif
00530 #else
00531 #   if defined(VF77_NOUNDERSCORE)
00532 #       define VF77_MANGLE(name,NAME) name
00533 #   elif defined(VF77_ONEUNDERSCORE)
00534 #       define VF77_MANGLE(name,NAME) name ## _
00535 #   else
00536 #       define VF77_MANGLE(name,NAME) name
00539 #   endif
00540 #endif
00541 #endif
00542
00543 /* Floating Point Error */
00544 #if defined(FLOAT_EPSILON)
00545 #   define VFLOOR(value) \
00546 #       ((floor(value) != floor(value + FLOAT_EPSILON)) ? \
00547 #       floor(value + FLOAT_EPSILON) : floor(value))
00548 #else
00549 #   define VFLOOR(value) floor(value)
00554 #endif
00555 #endif
00556
00557 /* String embedding for ident */
00558 #if defined(HAVE_EMBED)
00559 #   define VEMBED(rctag) \
00564 #       VPRIVATE const char* rctag; \
00565 #       static void* use_rcsid=(0 ? &use_rcsid : (void*)&rcsid);
00566 #else
00567 #   define VEMBED(rctag)
00571 #endif /* if defined(HAVE_EMBED) */
00572
00573
00574
00575
00577 #if !defined(WIN32) || defined(__MINGW32__)
00578 #define PRINT_FUNC __PRETTY_FUNCTION__
00579 #define OS_SEP_STR "/"
00580 #define OS_SEP_CHAR '/'
00581 #else
00582 #define OS_SEP_STR "\\"
00583 #define OS_SEP_CHAR '\\'
00584 #define PRINT_FUNC __FUNCSIG__
00585 #define snprintf sprintf_s
00586 #endif
00587
00588 #ifdef VERBOSE_DEBUG
00589 #define ANNOUNCE_FUNCTION
00590 #define do {
00591 #define Vnm_print(2, "%s() [%s:%d]\n",
00592 #define PRINT_FUNC, __FILE__, __LINE__ ); \
00593 #define } while(0)
00594
00595 #define WARN_UNTESTED
00596 #define do {
00597 #define Vnm_print(2, "%s() [%s:%d]: Untested Translation!\n",
00598 #define __FUNCTION__, __FILE__, __LINE__);
00599 #define } while(0)
00600
00601 #define WARN_PARTTESTED
00602 #define do{
00603 #define Vnm_print(2, "%s() [%s:%d]: Partially Tested Translation.\n",
00604 #define __FUNCTION__, __FILE__, __LINE__);
00605 #define } while(0)
00606 #else
00607 #define ANNOUNCE_FUNCTION
00608 #define WARN_UNTESTED
00609 #define WARN_PARTTESTED
00610 #endif
00611
00612
00613
00614 /* Utility messages. Print out messages with location information */
00615 #ifdef DEBUG

```

```

00616 #define VCHANNELEDMESSAGE0(channel, msg)          \
00617     do {                                             \
00618         Vnm_print(channel, "%s:%d [%s()]: MESSAGE:\n" \
00619             "    %s\n\n",                             \
00620             __FILE__, __LINE__, __FUNCTION__, msg); \
00621     } while(0)
00622
00623 #define VCHANNELEDMESSAGE1(channel, msg, arg)        \
00624     do {                                             \
00625         char buff[1000];                             \
00626         snprintf( buff, 1000, msg, arg );             \
00627         Vnm_print(channel, "%s:%d [%s()]: MESSAGE:\n" \
00628             "    %s\n\n",                             \
00629             __FILE__, __LINE__, __FUNCTION__, buff); \
00630     } while(0)
00631
00632 #define VCHANNELEDMESSAGE2(channel, msg, arg0, arg1) \
00633     do {                                             \
00634         char buff[1000];                             \
00635         snprintf( buff, 1000, msg, arg0, arg1 );     \
00636         Vnm_print(channel, "%s:%d [%s()]: MESSAGE:\n" \
00637             "    %s\n\n",                             \
00638             __FILE__, __LINE__, __FUNCTION__, buff); \
00639     } while(0)
00640
00641 #define VCHANNELEDMESSAGE3(channel, msg, arg0, arg1, arg2) \
00642     do {                                             \
00643         char buff[1000];                             \
00644         snprintf(buff, 1000, msg, arg0, arg1, arg2); \
00645         Vnm_print(channel, "%s:%d [%s()]: MESSAGE:\n" \
00646             "    %s\n\n",                             \
00647             __FILE__, __LINE__, __FUNCTION__, buff); \
00648     } while(0)
00649
00650 #define VMESAGE0(msg) VCHANNELEDMESSAGE0(2, msg)
00651 #define VMESAGE1(msg, arg0) VCHANNELEDMESSAGE1(2, msg, arg0)
00652 #define VMESAGE2(msg, arg0, arg1) VCHANNELEDMESSAGE2(2, msg, arg0, arg1)
00653 #define VMESAGE3(msg, arg0, arg1, arg2) VCHANNELEDMESSAGE3(2, msg, arg0, arg1, arg2)
00654
00655 #define VERRMSG0(msg) VMESAGE0(msg)
00656 #define VERRMSG1(msg, arg0) VMESAGE1(msg, arg0)
00657 #define VERRMSG2(msg, arg0, arg1) VMESAGE2(msg, arg0, arg1)
00658 #define VERRMSG3(msg, arg0, arg1, arg2) VMESAGE3(msg, arg0, arg1, arg2)
00659 #else
00660 #define VCHANNELEDMESSAGE0(channel, msg)          \
00661     do {                                             \
00662         Vnm_print(channel, "%s: %s\n", __FUNCTION__, msg); \
00663     } while(0)
00664
00665 #define VCHANNELEDMESSAGE1(channel, msg, arg0)        \
00666     do {                                             \
00667         char buff[1000];                             \
00668         snprintf( buff, 1000, msg, arg0 );             \
00669         Vnm_print(channel, "%s: %s\n", __FUNCTION__, buff); \
00670     } while(0)
00671
00672 #define VCHANNELEDMESSAGE2(channel, msg, arg0, arg1) \
00673     do {                                             \
00674         char buff[1000];                             \
00675         snprintf( buff, 1000, msg, arg0, arg1 );     \
00676         Vnm_print(channel, "%s: %s\n", __FUNCTION__, buff); \
00677     } while(0)
00678
00679 #define VCHANNELEDMESSAGE3(channel, msg, arg0, arg1, arg2) \
00680     do {                                             \
00681         char buff[1000];                             \
00682         snprintf(buff, 1000, msg, arg0, arg1, arg2); \
00683         Vnm_print(channel, "%s: %s\n", __FUNCTION__, buff); \
00684     } while(0)
00685
00686 #define VMESAGE0(msg) VCHANNELEDMESSAGE0(0, msg)
00687 #define VMESAGE1(msg, arg0) VCHANNELEDMESSAGE1(0, msg, arg0)
00688 #define VMESAGE2(msg, arg0, arg1) VCHANNELEDMESSAGE2(0, msg, arg0, arg1)
00689 #define VMESAGE3(msg, arg0, arg1, arg2) VCHANNELEDMESSAGE3(0, msg, arg0, arg1, arg2)
00690
00691 #define VERRMSG0(msg) VCHANNELEDMESSAGE0(2, msg)
00692 #define VERRMSG1(msg, arg0) VCHANNELEDMESSAGE1(2, msg, arg0)
00693 #define VERRMSG2(msg, arg0, arg1) VCHANNELEDMESSAGE2(2, msg, arg0, arg1)
00694 #define VERRMSG3(msg, arg0, arg1, arg2) VCHANNELEDMESSAGE3(2, msg, arg0, arg1, arg2)
00695 #endif
00696

```

```

00697
00698
00699 /* Utility assertions. If they fail, they print out messages with possible
00700 * arguments and then abort
00701 * The do{...} while(0) simply enforces that a semicolon appears at the end
00702 */
00703 #ifdef DEBUG
00704 #define VASSERT_MSG0(cnd, msg)
00705     do {
00706         if( (cnd) == 0 ) {
00707             Vnm_print(2, "%s:%d [%s()]: ERROR:\n"
00708                 "      Assertion Failed (%s): %s\n\n",
00709                 __FILE__, __LINE__, __FUNCTION__, #cnd, msg);
00710             abort();
00711         }
00712     } while(0)
00713
00714 #define VASSERT_MSG1(cnd, msg, arg)
00715     do {
00716         if( (cnd) == 0 ) {
00717             char buff[1000];
00718             snprintf( buff, 1000, msg, arg );
00719             Vnm_print(2, "%s:%d [%s()]: ERROR:\n"
00720                 "      Assertion Failed (%s): %s\n\n",
00721                 __FILE__, __LINE__, __FUNCTION__, #cnd, buff);
00722             abort();
00723         }
00724     } while(0)
00725
00726 #define VASSERT_MSG2(cnd, msg, arg0, arg1)
00727     do {
00728         if( (cnd) == 0 ) {
00729             char buff[1000];
00730             snprintf( buff, 1000, msg, arg0, arg1 );
00731             Vnm_print(2, "%s:%d [%s()]: ERROR:\n"
00732                 "      Assertion Failed (%s): %s\n\n",
00733                 __FILE__, __LINE__, __FUNCTION__, #cnd, buff);
00734             abort();
00735         }
00736     } while(0)
00737 #else
00738 #define VASSERT_MSG0(cnd, msg)
00739     do {
00740         if( (cnd) == 0 ) {
00741             Vnm_print(2, "%[%s()]: ERROR:\n"
00742                 "      Assertion Failed (%s): %s\n\n",
00743                 __FUNCTION__, #cnd, msg);
00744             abort();
00745         }
00746     } while(0)
00747
00748 #define VASSERT_MSG1(cnd, msg, arg)
00749     do {
00750         if( (cnd) == 0 ) {
00751             char buff[1000];
00752             snprintf( buff, 1000, msg, arg );
00753             Vnm_print(2, "[%s()]: ERROR:\n"
00754                 "      Assertion Failed (%s): %s\n\n",
00755                 __FUNCTION__, #cnd, buff);
00756             abort();
00757         }
00758     } while(0)
00759
00760 #define VASSERT_MSG2(cnd, msg, arg0, arg1)
00761     do {
00762         if( (cnd) == 0 ) {
00763             char buff[1000];
00764             snprintf( buff, 1000, msg, arg0, arg1 );
00765             Vnm_print(2, "[%s()]: ERROR:\n"
00766                 "      Assertion Failed (%s): %s\n\n",
00767                 __FUNCTION__, #cnd, buff);
00768             abort();
00769         }
00770     } while(0)
00771 #endif
00772
00773
00774
00775 /* Utility warning. Tests a condition and if it fails prints out a message
00776 * with optional arguments
00777 * The do{...} while(0) simply enforces that a semicolon at the end

```

```

00778  */
00779 #ifdef DEBUG
00780 #define VWARN_MSG0(cnd, msg)
00781     do {
00782         if( (cnd) == 0 ) {
00783             Vnm_print(
00784                 2,
00785                 "%s:%d [%s()]: WARNING:\n"
00786                 "    Condition Failed (%s):\n    %s\n\n",
00787                 __FILE__,
00788                 __LINE__,
00789                 __FUNCTION__,
00790                 #cnd,
00791                 msg
00792             );
00793         }
00794     } while(0)
00795
00796 #define VWARN_MSG1(cnd, msg, arg0)
00797     do {
00798         if( (cnd) == 0 ) {
00799             char buff[1000];
00800             snprintf(buff, 1000, msg, arg0);
00801             Vnm_print(
00802                 2,
00803                 "%s:%d [%s()]: WARNING:\n"
00804                 "    Condition Failed (%s):\n    %s\n\n",
00805                 __FILE__,
00806                 __LINE__,
00807                 __FUNCTION__,
00808                 #cnd,
00809                 buff
00810             );
00811         }
00812     } while(0)
00813
00814 #define VWARN_MSG2(cnd, msg, arg0, arg1)
00815     do {
00816         if( (cnd) == 0 ) {
00817             char buff[1000];
00818             snprintf(buff, 1000, msg, arg0, arg1);
00819             Vnm_print(
00820                 2,
00821                 "%s:%d [%s()]: WARNING:\n"
00822                 "    Condition Failed (%s):\n    %s\n\n",
00823                 __FILE__,
00824                 __LINE__,
00825                 __FUNCTION__,
00826                 #cnd,
00827                 buff
00828             );
00829         }
00830     } while(0)
00831 #else
00832 #define VWARN_MSG0(cnd, msg)
00833     do {
00834         if( (cnd) == 0 ) {
00835             Vnm_print(
00836                 2,
00837                 "[%s()]: WARNING:\n"
00838                 "    %s\n\n",
00839                 __FUNCTION__,
00840                 msg
00841             );
00842         }
00843     } while(0)
00844
00845 #define VWARN_MSG1(cnd, msg, arg0)
00846     do {
00847         if( (cnd) == 0 ) {
00848             char buff[1000];
00849             snprintf(buff, 1000, msg, arg0);
00850             Vnm_print(
00851                 2,
00852                 "[%s()]: WARNING:\n"
00853                 "    %s\n\n",
00854                 __FUNCTION__,
00855                 buff
00856             );
00857         }
00858     } while(0)

```

```

00859
00860 #define VWARN_MSG2(cnd, msg, arg0, arg1)
00861 do {
00862     if( (cnd) == 0 ) {
00863         char buff[1000];
00864         snprintf(buff, 1000, msg, arg0, arg1);
00865         Vnm_print(
00866             2,
00867             "[%s()]: WARNING:\n"
00868             "    %s\n\n",
00869             __FUNCTION__,
00870             buff
00871         );
00872     }
00873 } while(0)
00874 #endif
00875
00876 /* Utility Abort. Prints a message with optional arugments and aborts */
00877 #ifdef DEBUG
00878 #define VABORT_MSG0(msg)
00879 do {
00880     Vnm_print(2, "%s:%d [%s()]: ABORTING:\n"
00881         "    %s\n\n",
00882         __FILE__, __LINE__, __FUNCTION__, msg);
00883     abort();
00884 } while(0)
00885
00886 #define VABORT_MSG1(msg, arg)
00887 do {
00888     char buff[1000];
00889     snprintf( buff, 1000, msg, arg );
00890     Vnm_print(2, "%s:%d [%s()]: ABORTING:\n"
00891         "    %s\n\n",
00892         __FILE__, __LINE__, __FUNCTION__, buff);
00893     abort();
00894 } while(0)
00895
00896 #define VABORT_MSG2(msg, arg0, arg1)
00897 do {
00898     char buff[1000];
00899     snprintf( buff, 1000, msg, arg0, arg1);
00900     Vnm_print(2, "%s:%d [%s()]: ABORTING:\n"
00901         "    %s\n\n",
00902         __FILE__, __LINE__, __FUNCTION__, buff);
00903     abort();
00904 } while(0)
00905 #else
00906 #define VABORT_MSG0(msg)
00907 do {
00908     Vnm_print(2, "[%s()]: ABORTING:\n"
00909         "    %s\n\n",
00910         __FUNCTION__, msg);
00911     abort();
00912 } while(0)
00913
00914 #define VABORT_MSG1(msg, arg)
00915 do {
00916     char buff[1000];
00917     snprintf( buff, 1000, msg, arg );
00918     Vnm_print(2, "[%s()]: ABORTING:\n"
00919         "    %s\n\n",
00920         __FUNCTION__, buff);
00921     abort();
00922 } while(0)
00923
00924 #define VABORT_MSG2(msg, arg0, arg1)
00925 do {
00926     char buff[1000];
00927     snprintf( buff, 1000, msg, arg0, arg1);
00928     Vnm_print(2, "[%s()]: ABORTING:\n"
00929         "    %s\n\n",
00930         __FUNCTION__, buff);
00931     abort();
00932 } while(0)
00933 #endif
00934
00935
00936 /* Utility expression printers. Print the expression and its value */
00937 #ifdef DEBUG
00938 #define PRINT_INT(expr)

```

```

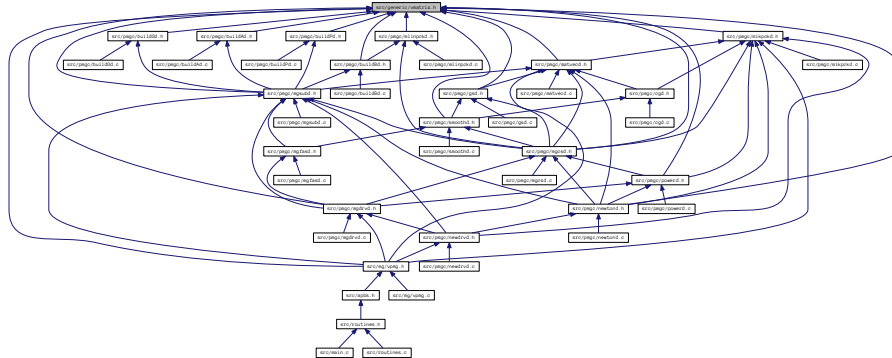
00940     do {
00941         Vnm_print(2, "%s:%d [%s()]: %s == %d\n",
00942                 __FILE__, __LINE__, __FUNCTION__, #expr, expr);
00943     } while(0)
00944
00945 #define PRINT_DBL(expr)
00946     do {
00947         Vnm_print(2, "%s:%d [%s()]: %s == %f\n\n",
00948                 __FILE__, __LINE__, __FUNCTION__, #expr, expr);
00949     } while(0)
00950 #else
00951 #define PRINT_INT(expr)
00952 #define PRINT_DBL(expr)
00953 #endif
00954
00955 #define VMALLOC(vmem, n, type) ((type*)Vmem_malloc(vmem, n, sizeof(type)))
00956
00957 #define VFREE(vmem, n, type, ptr) (Vmem_free(vmem, n, sizeof(type), (void **)&(ptr)))
00958
00959 #define VFILL(vec, n, val)
00960     do {
00961         int fill_idx;
00962         for (fill_idx = 0; fill_idx < n; fill_idx++)
00963             vec[fill_idx] = val;
00964     } while(0)
00965
00966 #define VCOPY(srcvec, dstvec, i, n) \
00967     do {
00968         for (i = 0; i < n; i++)
00969             dstvec[i] = srcvec[i];
00970     } while(0)
00971
00972
00973 char* wrap_text( char* str, int right_margin, int left_padding );
00974
00975 #define VAT(array, i) ((array)[(i) - 1])
00976 #define RAT(array, i) ((array) + i - 1)
00977
00978 #endif /* #ifndef _VAPBSHAL_H_ */

```

10.65 src/generic/vmatrix.h File Reference

Contains inclusions for matrix data wrappers.

This graph shows which files directly or indirectly include this file:



Macros

- #define **MAT2**(mat, dx, dy)
- #define **RAT2**(mat, x, y) &VAT2(mat, x, y)

- `#define VAT2(mat, x, y) mat[(y - 1) * dx_##mat + (x - 1)]`
- `#define MAT3(mat, dx, dy, dz)`
- `#define RAT3(mat, x, y, z) &VAT3(mat, x, y, z)`
- `#define VAT3(mat, x, y, z)`

10.65.1 Detailed Description

Contains inclusions for matrix data wrappers.

Version

Author

Tucker A. Beck

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) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory,
* operated by Battelle Memorial Institute,
* Pacific Northwest Division for the U.S. Department Energy.
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of California
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of 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 [vmatrix.h](#).

10.65.2 Macro Definition Documentation

10.65.2.1 #define MAT2(mat, dx, dy)

Value:

```
int dx_##mat = dx;    \
int dy_##mat = dy
```

Definition at line 64 of file [vmatrix.h](#).

10.65.2.2 #define MAT3(mat, dx, dy, dz)

Value:

```
int dx_##mat = dx;    \
int dy_##mat = dy;    \
int dz_##mat = dz
```

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

10.65.2.3 #define VAT3(mat, x, y, z)

Value:

```
mat[(z - 1) * dy_##mat * dx_##mat + \
(y - 1) * dx_##mat + \
(x - 1)]
```

Definition at line 84 of file [vmatrix.h](#).

10.66 vmatrix.h

```
00001
00061 #ifndef _VMATRIX_H_
00062 #define _VMATRIX_H_
00063
00064 #define MAT2(mat, dx, dy) \
00065     int dx_##mat = dx;    \
00066     int dy_##mat = dy
00067
00068 #define RAT2(mat, x, y) \
00069     &VAT2(mat, x, y)
00070
00071 #define VAT2(mat, x, y) \
00072     mat[(y - 1) * dx_##mat + (x - 1)]
00073
00074
00075
00076 #define MAT3(mat, dx, dy, dz) \
00077     int dx_##mat = dx;    \
00078     int dy_##mat = dy;    \
00079     int dz_##mat = dz
00080
00081 #define RAT3(mat, x, y, z) \
00082     &VAT3(mat, x, y, z)
00083
```

```

00084 #define VAT3(mat, x, y, z) \
00085     mat[(z - 1) * dy_##mat * dx_##mat + \
00086         (y - 1) * dx_##mat + \
00087         (x - 1)]
00088
00089 #endif /* _VMATRIX_H_ */

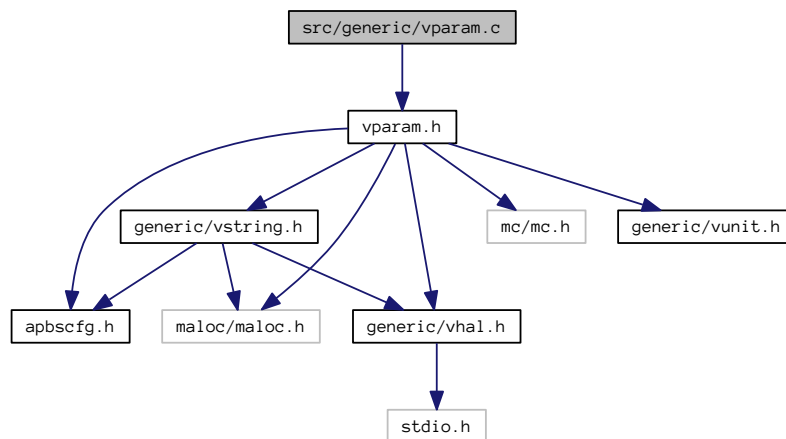
```

10.67 src/generic/vparam.c File Reference

Class [Vparam](#) methods.

```
#include "vparam.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)
FORTTRAN stub to construct the object.
- VPUBLIC void [Vparam_AtomData_dtor](#) ([Vparam_AtomData](#) **thee)
Destroy object.
- VPUBLIC void [Vparam_AtomData_dtor2](#) ([Vparam_AtomData](#) *thee)
FORTTRAN 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)

FORTTRAN stub to construct the object.

- VPUBLIC void `Vparam_ResData_dtor` (`Vparam_ResData` **thee)

Destroy object.

- VPUBLIC void `Vparam_ResData_dtor2` (`Vparam_ResData` *thee)

FORTTRAN stub to destroy object.

- VPUBLIC `Vparam` * `Vparam_ctor` ()

Construct the object.

- VPUBLIC int `Vparam_ctor2` (`Vparam` *thee)

FORTTRAN stub to construct the object.

- VPUBLIC void `Vparam_dtor` (`Vparam` **thee)

Destroy object.

- VPUBLIC void `Vparam_dtor2` (`Vparam` *thee)

FORTTRAN 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.

10.67.1 Detailed Description

Class [Vparam](#) methods.

Author

Nathan Baker

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [vparam.c](#).

10.68 vparam.c

```

00001
00057 #include "vparam.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061
00065 VPRIVATE char *MCwhiteChars = " =,;\t\n\r";
00066
00071 VPRIVATE char *MCcommChars = "##";
00072
00077 VPRIVATE char *MCxmlwhiteChars = " =,;\t\n\r<>";
00078
00087 VPRIVATE int readFlatFileLine(Vio *sock, Vparam_AtomData *atom);
00088
00097 VPRIVATE int readXMLFileAtom(Vio *sock, Vparam_AtomData *atom);
00098
00099
00100 #if !defined(VINLINE_VPARAM)
00101
00102 VPUBLIC unsigned long int Vparam_memChk(Vparam *thee) {
00103     if (thee == VNULL) return 0;
00104     return Vmem_bytes(thee->vmem);
00105 }
00106
00107 #endif /* if !defined(VINLINE_VPARAM) */
00108
00109 VPUBLIC Vparam_AtomData* Vparam_AtomData_ctor() {
00110
00111     Vparam_AtomData *thee = VNULL;
00112
00113     /* Set up the structure */
00114     thee = (Vparam_AtomData*)Vmem_malloc(VNULL, 1, sizeof(
Vparam_AtomData ));
00115     VASSERT(thee != VNULL);
00116     VASSERT(Vparam_AtomData_ctor2(thee));
00117
00118     return thee;
00119 }
00120
00121 VPUBLIC int Vparam_AtomData_ctor2(Vparam_AtomData *thee) { return 1; }
00122
00123 VPUBLIC void Vparam_AtomData_dtor(Vparam_AtomData **thee) {
00124
00125     if ((*thee) != VNULL) {
00126         Vparam_AtomData_dtor2(*thee);
00127         Vmem_free(VNULL, 1, sizeof(Vparam_AtomData), (void **)thee);
00128         (*thee) = VNULL;
00129     }
00130
00131 }
00132
00133 VPUBLIC void Vparam_AtomData_dtor2(Vparam_AtomData *thee) { ; }
00134
00135 VPUBLIC Vparam_ResData* Vparam_ResData_ctor(Vmem *mem) {
00136
00137     Vparam_ResData *thee = VNULL;
00138
00139     /* Set up the structure */
00140     thee = (Vparam_ResData*)Vmem_malloc(mem, 1, sizeof(
Vparam_ResData ));
00141     VASSERT(thee != VNULL);
00142     VASSERT(Vparam_ResData_ctor2(thee, mem));
00143
00144     return thee;
00145 }
00146
00147 VPUBLIC int Vparam_ResData_ctor2(Vparam_ResData *thee, Vmem *mem) {
00148
00149     if (thee == VNULL) {
00150         Vnm_print(2, "Vparam_ResData_ctor2: Got VNULL thee!\n");
00151         return 0;
00152     }
00153     thee->vmem = mem;
00154     thee->nAtomData = 0;
00155     thee->atomData = VNULL;
00156
00157     return 1;
00158 }

```

```

00159
00160 VPUBLIC void Vparam_ResData_dtor(Vparam_ResData **thee) {
00161
00162     if ((*thee) != VNULL) {
00163         Vparam_ResData_dtor2(*thee);
00164         Vmem_free((*thee)->vmem, 1, sizeof(Vparam_ResData), (void **)thee);
00165         (*thee) = VNULL;
00166     }
00167
00168 }
00169
00170 VPUBLIC void Vparam_ResData_dtor2(Vparam_ResData *thee) {
00171
00172     if (thee == VNULL) return;
00173     if (thee->nAtomData > 0) {
00174         Vmem_free(thee->vmem, thee->nAtomData, sizeof(
00175             Vparam_AtomData),
00176             (void **)&(thee->atomData));
00177     }
00178     thee->nAtomData = 0;
00179     thee->atomData = VNULL;
00180 }
00181 VPUBLIC Vparam* Vparam_ctor() {
00182
00183     Vparam *thee = VNULL;
00184
00185     /* Set up the structure */
00186     thee = (Vparam*)Vmem_malloc(VNULL, 1, sizeof(Vparam) );
00187     VASSERT(thee != VNULL);
00188     VASSERT(Vparam_ctor2(thee));
00189
00190     return thee;
00191 }
00192
00193 VPUBLIC int Vparam_ctor2(Vparam *thee) {
00194
00195     if (thee == VNULL) {
00196         Vnm_print(2, "Vparam_ctor2: got VNULL thee!\n");
00197         return 0;
00198     }
00199
00200     thee->vmem = VNULL;
00201     thee->vmem = Vmem_ctor("APBS:VPARAM");
00202     if (thee->vmem == VNULL) {
00203         Vnm_print(2, "Vparam_ctor2: failed to init Vmem!\n");
00204         return 0;
00205     }
00206
00207     thee->nResData = 0;
00208     thee->resData = VNULL;
00209
00210     return 1;
00211 }
00212
00213 VPUBLIC void Vparam_dtor(Vparam **thee) {
00214
00215     if ((*thee) != VNULL) {
00216         Vparam_dtor2(*thee);
00217         Vmem_free(VNULL, 1, sizeof(Vparam), (void **)thee);
00218         (*thee) = VNULL;
00219     }
00220
00221 }
00222
00223 VPUBLIC void Vparam_dtor2(Vparam *thee) {
00224
00225     int i;
00226
00227     if (thee == VNULL) return;
00228
00229     /* Destroy the residue data */
00230     for (i=0; i<thee->nResData; i++) Vparam_ResData_dtor2(&(thee->
00231         resData[i]));
00232     if (thee->nResData > 0) Vmem_free(thee->vmem, thee->nResData,
00233         sizeof(Vparam_ResData), (void **)&(thee->resData));
00234     thee->nResData = 0;
00235     thee->resData = VNULL;
00236
00237     if (thee->vmem != VNULL) Vmem_dtor(&(thee->vmem));
00238     thee->vmem = VNULL;

```

```

00238
00239 }
00240
00241 VPUBLIC Vparam_ResData* Vparam_getResData(Vparam *thee,
00242     char resName[VMAX_ARGLEN]) {
00243
00244     int i;
00245     Vparam_ResData *res = VNULL;
00246
00247     VASSERT(thee != VNULL);
00248
00249     if ((thee->nResData == 0) || (thee->resData == VNULL)) {
00250         res = VNULL;
00251         return res;
00252     }
00253
00254     /* Look for the matching residue */
00255     for (i=0; i<thee->nResData; i++) {
00256         res = &(thee->resData[i]);
00257         if (Vstring_strcasecmp(resName, res->name) == 0) return res;
00258     }
00259
00260     /* Didn't find a matching residue */
00261     res = VNULL;
00262     Vnm_print(2, "Vparam_getResData: unable to find res=%s\n", resName);
00263     return res;
00264 }
00265
00266
00267 VPUBLIC Vparam_AtomData* Vparam_getAtomData(
00268     Vparam *thee,
00269     char resName[VMAX_ARGLEN], char atomName[VMAX_ARGLEN]) {
00270
00271     int i;
00272     Vparam_ResData *res = VNULL;
00273     Vparam_AtomData *atom = VNULL;
00274
00275     VASSERT(thee != VNULL);
00276
00277     if ((thee->nResData == 0) || (thee->resData == VNULL)) {
00278         atom = VNULL;
00279         return atom;
00280     }
00281
00282     /* Look for the matching residue */
00283     res = Vparam_getResData(thee, resName);
00284     if (res == VNULL) {
00285         atom = VNULL;
00286         Vnm_print(2, "Vparam_getAtomData: Unable to find residue %s!\n", resName);
00287         return atom;
00288     }
00289     for (i=0; i<res->nAtomData; i++) {
00290         atom = &(res->atomData[i]);
00291         if (atom == VNULL) {
00292             Vnm_print(2, "Vparam_getAtomData: got NULL atom!\n");
00293             return VNULL;
00294         }
00295         if (Vstring_strcasecmp(atomName, atom->atomName) == 0) {
00296             return atom;
00297         }
00298     }
00299
00300     /* Didn't find a matching atom/residue */
00301     atom = VNULL;
00302     Vnm_print(2, "Vparam_getAtomData: unable to find atom '%s', res '%s'\n",
00303         atomName, resName);
00304     return atom;
00305 }
00306
00307 VPUBLIC int Vparam_readXMLFile(Vparam *thee, const char *iodev,
00308     const char *iofmt, const char *thost, const char *fname) {
00309
00310     int i, ires, natoms, nalloc, ralloc;
00311     Vparam_AtomData *atoms = VNULL;
00312     Vparam_AtomData *tatoms = VNULL;
00313     Vparam_AtomData *atom = VNULL;
00314     Vparam_ResData *res = VNULL;
00315     Vparam_ResData *residues = VNULL;
00316     Vparam_ResData *tresidues = VNULL;
00317     Vio *sock = VNULL;
00318     char currResName[VMAX_ARGLEN];

```

```

00318     char tok[VMAX_ARGLEN];
00319     char endtag[VMAX_ARGLEN];
00320
00321     VASSERT(thee != VNULL);
00322
00323     /* Setup communication */
00324     sock = Vio_ctor(iodev, iofmt, thost, fname, "r");
00325     if (sock == VNULL) {
00326         Vnm_print(2, "Vparam_readXMLFile: Problem opening virtual socket %s\n",
00327             fname);
00328         return 0;
00329     }
00330     if (Vio_accept(sock, 0) < 0) {
00331         Vnm_print(2, "Vparam_readXMLFile: Problem accepting virtual socket %s\n",
00332             fname);
00333         return 0;
00334     }
00335     Vio_setWhiteChars(sock, MCxmlwhiteChars);
00336     Vio_setCommChars(sock, MCcommChars);
00337
00338     /* Clear existing parameters */
00339     if (thee->nResData > 0) {
00340         Vnm_print(2, "WARNING -- CLEARING PARAMETER DATABASE!\n");
00341         for (i=0; i<thee->nResData; i++) {
00342             Vparam_ResData_dtor2(&(thee->resData[i]));
00343         }
00344         Vmem_free(thee->vmem, thee->nResData,
00345             sizeof(Vparam_ResData), (void *)&(thee->resData));
00346     }
00347
00348     strcpy(endtag, "/");
00349
00350     /* Set up temporary residue list */
00351     ralloc = 50;
00352     residues = (Vparam_ResData*)Vmem_malloc(thee->vmem, ralloc, sizeof(
00353 Vparam_ResData));
00354
00355     /* Read until we run out of entries, allocating space as needed */
00356     while (1) {
00357         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00358
00359         /* The first token should be the start tag */
00360
00361         if (Vstring_strcasecmp(endtag, "/") == 0) strcat(endtag, tok);
00362
00363         if (Vstring_strcasecmp(tok, "residue") == 0) {
00364             if (thee->nResData >= ralloc) {
00365                 tresidues = (Vparam_ResData*)Vmem_malloc(thee->
00366 vmem, 2*ralloc, sizeof(Vparam_ResData));
00367                 VASSERT(tresidues != VNULL);
00368                 for (i=0; i<thee->nResData; i++) {
00369                     Vparam_ResData_copyTo(&(residues[i]), &(tresidues[i]));
00370                 }
00371                 Vmem_free(thee->vmem, ralloc, sizeof(Vparam_ResData),
00372                     (void *)&(residues));
00373                 residues = tresidues;
00374                 tresidues = VNULL;
00375                 ralloc = 2*ralloc;
00376             }
00377
00378             /* Initial space for this residue's atoms */
00379             nalloc = 20;
00380             natoms = 0;
00381             atoms = (Vparam_AtomData*)Vmem_malloc(thee->vmem, nalloc, sizeof(
00382 Vparam_AtomData));
00383             } else if (Vstring_strcasecmp(tok, "name") == 0) {
00384                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1); /* value */
00385                 strcpy(currResName, tok);
00386                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1); /* </name> */
00387             } else if (Vstring_strcasecmp(tok, "atom") == 0) {
00388                 if (natoms >= nalloc) {
00389                     tatoms = (Vparam_AtomData*)Vmem_malloc(thee->vmem, 2*nalloc, sizeof(
00390 Vparam_AtomData));
00391                     VASSERT(tatoms != VNULL);
00392                     for (i=0; i<natoms; i++) {
00393                         Vparam_AtomData_copyTo(&(atoms[i]), &(tatoms[i]));
00394                     }
00395                     Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData),

```



```

00395             (void **)&(atoms));
00396             atoms = tatoms;
00397             tatoms = VNULL;
00398             nalloc = 2*nalloc;
00399         }
00400         atom = &(atoms[natoms]);
00401         if (!readXMLFileAtom(sock, atom)) break;
00402         natoms++;
00403     } else if (Vstring_strcasecmp(tok, "/residue") == 0) {
00404     }
00405     res = &(residues[thee->nResData]);
00406     Vparam_ResData_ctor2(res, thee->vmem);
00407     res->atomData = (Vparam_AtomData*)Vmem_malloc(thee->
00408 vmem, natoms,
00409                                     sizeof(Vparam_AtomData));
00410     res->nAtomData = natoms;
00411     strcpy(res->name, currResName);
00412     for (i=0; i<natoms; i++) {
00413         strcpy(atoms[i].resName, currResName);
00414         Vparam_AtomData_copyTo(&(atoms[i]), &(res->
00415 atomData[i]));
00416     }
00417     Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData), (void **)&(atoms));
00418     (thee->nResData)++;
00419 } else if (Vstring_strcasecmp(tok, endtag) == 0) break;
00420 }
00421 /* Initialize and copy the residues into the Vparam object */
00422 thee->resData = (Vparam_ResData*)Vmem_malloc(thee->vmem, thee->
00423 nResData,
00424                                     sizeof(Vparam_ResData));
00425 for (ires=0; ires<thee->nResData; ires++) {
00426     Vparam_ResData_copyTo(&(residues[ires]), &(thee->
00427 resData[ires]));
00428 }
00429 /* Destroy temporary atom space */
00430 Vmem_free(thee->vmem, nalloc, sizeof(Vparam_ResData), (void **)&(residues));
00431 /* Shut down communication */
00432 Vio_acceptFree(sock);
00433 Vio_dtor(&sock);
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 VPUBLIC int Vparam_readFlatFile(Vparam *thee, const char *iodev,
00442 const char *iofmt, const char *thost, const char *fname) {
00443     int i, iatom, jatom, ires, natoms, nalloc;
00444     Vparam_AtomData *atoms = VNULL;
00445     Vparam_AtomData *tatoms = VNULL;
00446     Vparam_AtomData *atom = VNULL;
00447     Vparam_ResData *res = VNULL;
00448     Vio *sock = VNULL;
00449     char currResName[VMAX_ARGLEN];
00450     VASSERT(thee != VNULL);
00451     /* Setup communication */
00452     sock = Vio_ctor(iodev, iofmt, thost, fname, "r");
00453     if (sock == VNULL) {
00454         Vnm_print(2, "Vparam_readFlatFile: Problem opening virtual socket %s\n",
00455 fname);
00456         return 0;
00457     }
00458     if (Vio_accept(sock, 0) < 0) {
00459         Vnm_print(2, "Vparam_readFlatFile: Problem accepting virtual socket %s\n",
00460 fname);
00461         return 0;
00462     }
00463     Vio_setWhiteChars(sock, MCwhiteChars);
00464     Vio_setCommChars(sock, MCcommChars);

```

```

00472
00473     /* Clear existing parameters */
00474     if (thee->nResData > 0) {
00475         Vnm_print(2, "WARNING -- CLEARING PARAMETER DATABASE!\n");
00476         for (i=0; i<thee->nResData; i++) {
00477             Vparam_ResData_dtor2(&(thee->resData[i]));
00478         }
00479         Vmem_free(thee->vmem, thee->nResData,
00480             sizeof(Vparam_ResData), (void **)&(thee->resData));
00481     }
00482
00483     /* Initial space for atoms */
00484     nalloc = 200;
00485     natoms = 0;
00486     atoms = (Vparam_AtomData*)Vmem_malloc(thee->vmem, nalloc, sizeof(
Vparam_AtomData));
00487
00488     /* Read until we run out of entries, allocating space as needed */
00489     while (1) {
00490         if (natoms >= nalloc) {
00491             tatoms = (Vparam_AtomData*)Vmem_malloc(thee->vmem, 2*nalloc, sizeof(
Vparam_AtomData));
00492             VASSERT(tatoms != VNULL);
00493             for (i=0; i<natoms; i++) {
00494                 Vparam_AtomData_copyTo(&(atoms[i]), &(tatoms[i]));
00495             }
00496             Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData),
00497                 (void **)&(atoms));
00498             atoms = tatoms;
00499             tatoms = VNULL;
00500             nalloc = 2*nalloc;
00501         }
00502         atom = &(atoms[natoms]);
00503         if (!readFlatFileLine(sock, atom)) break;
00504         natoms++;
00505     }
00506     if (natoms == 0) return 0;
00507
00508     /* Count the number of residues */
00509     thee->nResData = 1;
00510     strcpy(currResName, atoms[0].resName);
00511     for (i=1; i<natoms; i++) {
00512         if (Vstring_strcasecmp(atoms[i].resName, currResName) != 0) {
00513             strcpy(currResName, atoms[i].resName);
00514             (thee->nResData)++;
00515         }
00516     }
00517
00518     /* Create the residues */
00519     thee->resData = (Vparam_ResData*)Vmem_malloc(thee->vmem, thee->
nResData,
00520         sizeof(Vparam_ResData));
00521     VASSERT(thee->resData != VNULL);
00522     for (i=0; i<(thee->nResData); i++) {
00523         res = &(thee->resData[i]);
00524         Vparam_ResData_ctor2(res, thee->vmem);
00525     }
00526
00527     /* Count the number of atoms per residue */
00528     ires = 0;
00529     res = &(thee->resData[ires]);
00530     res->nAtomData = 1;
00531     strcpy(res->name, atoms[0].resName);
00532     for (i=1; i<natoms; i++) {
00533         if (Vstring_strcasecmp(atoms[i].resName, res->name) != 0) {
00534             (ires)++;
00535             res = &(thee->resData[ires]);
00536             res->nAtomData = 1;
00537             strcpy(res->name, atoms[i].resName);
00538         } else (res->nAtomData)++;
00539     }
00540
00541     /* Allocate per-residue space for atoms */
00542     for (ires=0; ires<thee->nResData; ires++) {
00543         res = &(thee->resData[ires]);
00544         res->atomData = (Vparam_AtomData*)Vmem_malloc(thee->
vmem, res->nAtomData,
00545             sizeof(Vparam_AtomData));
00546     }
00547
00548     /* Copy atoms into residues */

```

```

00549     iatom = 0;
00550     Vparam_AtomData_copyTo(&(atoms[0]), &(res->atomData[iatom]));
00551     for (ires=0; ires<thee->nResData; ires++) {
00552         res = &(thee->resData[ires]);
00553         for (jatom=0; jatom<res->nAtomData; jatom++) {
00554             Vparam_AtomData_copyTo(&(atoms[iatom]), &(res->
atomData[jatom]));
00555             iatom++;
00556         }
00557     }
00558
00559     /* Shut down communication */
00560     Vio_acceptFree(sock);
00561     Vio_dtor(&sock);
00562
00563     /* Destroy temporary atom space */
00564     Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData), (void **)&(atoms));
00565
00566     return 1;
00567 }
00568
00569 }
00570
00571 VEXTERNC void Vparam_AtomData_copyTo(Vparam_AtomData *thee,
00572 Vparam_AtomData *dest) {
00573
00574     VASSERT(thee != VNULL);
00575     VASSERT(dest != VNULL);
00576
00577     strcpy(dest->atomName, thee->atomName);
00578     strcpy(dest->resName, thee->resName);
00579     dest->charge = thee->charge;
00580     dest->radius = thee->radius;
00581     dest->epsilon = thee->epsilon;
00582 }
00583
00584
00585 VEXTERNC void Vparam_ResData_copyTo(Vparam_ResData *thee,
00586 Vparam_ResData *dest) {
00587
00588     int i;
00589
00590     VASSERT(thee != VNULL);
00591     VASSERT(dest != VNULL);
00592
00593     strcpy(dest->name, thee->name);
00594     dest->vmem = thee->vmem;
00595     dest->nAtomData = thee->nAtomData;
00596
00597     dest->atomData = (Vparam_AtomData*)Vmem_malloc(thee->
vmem, dest->nAtomData,
00598                                     sizeof(Vparam_AtomData));
00599
00600     for (i=0; i<dest->nAtomData; i++) {
00601         Vparam_AtomData_copyTo(&(thee->atomData[i]), &(dest->
atomData[i]));
00602     }
00603     Vmem_free(thee->vmem, thee->nAtomData, sizeof(Vparam_AtomData),
00604             (void **)&(thee->atomData));
00605 }
00606
00607 VEXTERNC void Vparam_AtomData_copyFrom(Vparam_AtomData *thee,
00608 Vparam_AtomData *src) { Vparam_AtomData_copyTo(src, thee); }
00609
00610 VPRIVATE int readXMLFileAtom(Vio *sock, Vparam_AtomData *atom) {
00611
00612     double dtmp;
00613     char tok[VMAX_BUFSIZE];
00614     int chgflag, radflag, nameflag;
00615
00616     VASSERT(atom != VNULL);
00617
00618     if (Vio_scanf(sock, "%s", tok) != 1) return 0;
00619
00620     chgflag = 0;
00621     radflag = 0;
00622     nameflag = 0;
00623
00624     while (1)
00625     {
00626         if (Vstring_strcasecmp(tok, "name") == 0) {

```

```

00627         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00628         if (strlen(tok) > VMAX_ARGLEN) {
00629             Vnm_print(2, "Vparam_readXMLFileAtom: string (%s) too long \
00630 (%d)!\n", tok, strlen(tok));
00631             return 0;
00632         }
00633         nameflag = 1;
00634         strcpy(atom->atomName, tok);
00635     } else if (Vstring_strcasecmp(tok, "charge") == 0) {
00636         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00637         if (sscanf(tok, "%lf", &dtmp) != 1) {
00638             Vnm_print(2, "Vparam_readXMLFileAtom: Unexpected token (%s) while \
00639 parsing charge!\n", tok);
00640             return 0;
00641         }
00642         chgflag = 1;
00643         atom->charge = dtmp;
00644     } else if (Vstring_strcasecmp(tok, "radius") == 0) {
00645         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00646         if (sscanf(tok, "%lf", &dtmp) != 1) {
00647             Vnm_print(2, "Vparam_readXMLFileAtom: Unexpected token (%s) while \
00648 parsing radius!\n", tok);
00649             return 0;
00650         }
00651         radflag = 1;
00652         atom->radius = dtmp;
00653     } else if (Vstring_strcasecmp(tok, "epsilon") == 0) {
00654         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00655         if (sscanf(tok, "%lf", &dtmp) != 1) {
00656             Vnm_print(2, "Vparam_readXMLFileAtom: Unexpected token (%s) while \
00657 parsing epsilon!\n", tok);
00658             return 0;
00659         }
00660         atom->epsilon = dtmp;
00661     } else if ((Vstring_strcasecmp(tok, "/atom") == 0) ||
00662                (Vstring_strcasecmp(tok, "atom") == 0)){
00663         if (chgflag && radflag && nameflag) return 1;
00664         else if (!chgflag) {
00665             Vnm_print(2, "Vparam_readXMLFileAtom: Reached end of atom without \
00666 setting the charge!\n");
00667             return 0;
00668         } else if (!radflag) {
00669             Vnm_print(2, "Vparam_readXMLFileAtom: Reached end of atom without \
00670 setting the radius!\n");
00671             return 0;
00672         } else if (!nameflag) {
00673             Vnm_print(2, "Vparam_readXMLFileAtom: Reached end of atom without \
00674 setting the name!\n");
00675             return 0;
00676         }
00677     }
00678     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00679 }
00680
00681 /* If we get here something wrong has happened */
00682
00683 VJMPERR1(1);
00684
00685 ERROR1:
00686     Vnm_print(2, "Vparam_readXMLFileAtom: Got unexpected EOF reading parameter file!\n");
00687     return 0;
00688 }
00689 }
00690
00691 VPRIVATE int readFlatFileLine(Vio *sock, Vparam_AtomData *atom) {
00692
00693     double dtmp;
00694     char tok[VMAX_BUFSIZE];
00695
00696     VASSERT(atom != VNULL);
00697
00698     if (Vio_scanf(sock, "%s", tok) != 1) return 0;
00699     if (strlen(tok) > VMAX_ARGLEN) {
00700         Vnm_print(2, "Vparam_readFlatFile: string (%s) too long (%d)!\n",
00701             tok, strlen(tok));
00702         return 0;
00703     }
00704     strcpy(atom->resName, tok);
00705     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00706     if (strlen(tok) > VMAX_ARGLEN) {
00707         Vnm_print(2, "Vparam_readFlatFile: string (%s) too long (%d)!\n",

```

```

00708         tok, strlen(tok));
00709     return 0;
00710 }
00711 strcpy(atom->atomName, tok);
00712 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00713 if (sscanf(tok, "%lf", &dtmp) != 1) {
00714     Vnm_print(2, "Vparam_readFlatFile: Unexpected token (%s) while \
00715 parsing charge!\n", tok);
00716     return 0;
00717 }
00718 atom->charge = dtmp;
00719 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00720 if (sscanf(tok, "%lf", &dtmp) != 1) {
00721     Vnm_print(2, "Vparam_readFlatFile: Unexpected token (%s) while \
00722 parsing radius!\n", tok);
00723     return 0;
00724 }
00725 atom->radius = dtmp;
00726 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00727 if (sscanf(tok, "%lf", &dtmp) != 1) {
00728     Vnm_print(2, "Vparam_readFlatFile: Unexpected token (%s) while \
00729 parsing radius!\n", tok);
00730     return 0;
00731 }
00732 atom->epsilon = dtmp;
00733 return 1;
00734 }
00735
00736 ERROR1:
00737 Vnm_print(2, "Vparam_readFlatFile: Got unexpected EOF reading parameter file!\n");
00738 return 0;
00739 }

```

10.69 src/generic/vparam.h File Reference

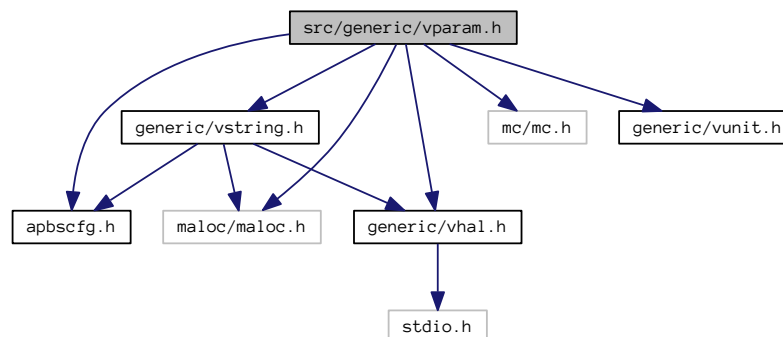
Contains declarations for class [Vparam](#).

```

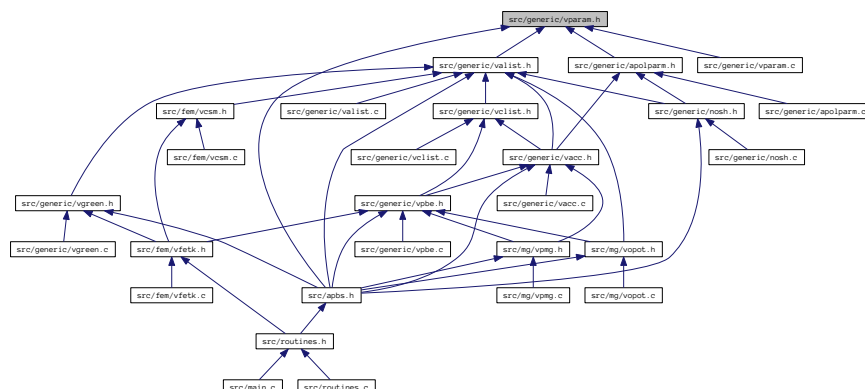
#include "apbscfg.h"
#include "malloc/malloc.h"
#include "mc/mc.h"
#include "generic/vhal.h"
#include "generic/vunit.h"
#include "generic/vstring.h"

```

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)
FORTTRAN stub to construct the object.
- VEXTERNC void [Vparam_AtomData_dtor](#) ([Vparam_AtomData](#) **thee)
Destroy object.
- VEXTERNC void [Vparam_AtomData_dtor2](#) ([Vparam_AtomData](#) *thee)
FORTTRAN 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)
FORTTRAN stub to construct the object.
- VEXTERNC void [Vparam_ResData_dtor](#) ([Vparam_ResData](#) **thee)
Destroy object.
- VEXTERNC void [Vparam_ResData_dtor2](#) ([Vparam_ResData](#) *thee)
FORTTRAN stub to destroy object.
- VEXTERNC [Vparam](#) * [Vparam_ctor](#) ()
Construct the object.
- VEXTERNC int [Vparam_ctor2](#) ([Vparam](#) *thee)
FORTTRAN stub to construct the object.
- VEXTERNC void [Vparam_dtor](#) ([Vparam](#) **thee)
Destroy object.
- VEXTERNC void [Vparam_dtor2](#) ([Vparam](#) *thee)
FORTTRAN 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.

10.69.1 Detailed Description

Contains declarations for class [Vparam](#).

Version

\$Id\$

Author

Nathan A. Baker

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vparam.h](#).

10.70 vparam.h

```

00001
00062 #ifndef _VPARAM_H_
00063 #define _VPARAM_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "maloc/maloc.h"
00068 #if defined(HAVE_MC_H)
00069 #include "mc/mc.h"
00070 #endif
00071
00072 #include "generic/vhal.h"
00073 #include "generic/vunit.h"
00074 #include "generic/vstring.h"

```



```

00075
00092 struct sVparam_AtomData {
00093     char atomName[VMAX_ARGLEN];
00094     char resName[VMAX_ARGLEN];
00095     double charge;
00096     double radius;
00097     double epsilon;
00099 };
00100
00106 typedef struct sVparam_AtomData Vparam_AtomData;
00107
00114 struct Vparam_ResData {
00115     Vmem *vmem;
00116     char name[VMAX_ARGLEN];
00117     int nAtomData;
00119     Vparam_AtomData *atomData;
00120 };
00121
00127 typedef struct Vparam_ResData Vparam_ResData;
00128
00135 struct Vparam {
00136     Vmem *vmem;
00137     int nResData;
00138     Vparam_ResData *resData;
00141 };
00142
00147 typedef struct Vparam Vparam;
00148
00149 /* ////////////////////////////////////////
00150 // Class Vparam: Inlineable methods (vparam.c)
00152
00153 #if !defined(VINLINE_VPARAM)
00154
00161     VEXTERNC unsigned long int Vparam_memChk(Vparam *thee);
00162
00163 #else /* if defined(VINLINE_VPARAM) */
00164 #   define Vparam_memChk(thee) (Vmem_bytes((thee)->vmem))
00166 #endif /* if !defined(VINLINE_VPARAM) */
00168
00169 /* ////////////////////////////////////////
00170 // Class Vparam: Non-Inlineable methods (vparam.c)
00172
00177 VEXTERNC Vparam_AtomData* Vparam_AtomData_ctor();
00178
00184 VEXTERNC int Vparam_AtomData_ctor2(Vparam_AtomData *thee);
00185
00190 VEXTERNC void Vparam_AtomData_dtor(Vparam_AtomData **thee);
00191
00196 VEXTERNC void Vparam_AtomData_dtor2(Vparam_AtomData *thee);
00197
00205 VEXTERNC void Vparam_AtomData_copyTo(Vparam_AtomData *thee,
00206     Vparam_AtomData *dest);
00207
00215 VEXTERNC void Vparam_ResData_copyTo(Vparam_ResData *thee,
00216     Vparam_ResData *dest);
00217
00225 VEXTERNC void Vparam_AtomData_copyFrom(Vparam_AtomData *thee,
00226     Vparam_AtomData *src);
00227
00233 VEXTERNC Vparam_ResData* Vparam_ResData_ctor(Vmem *mem);
00234
00241 VEXTERNC int Vparam_ResData_ctor2(Vparam_ResData *thee, Vmem *mem);
00242
00247 VEXTERNC void Vparam_ResData_dtor(Vparam_ResData **thee);
00248
00253 VEXTERNC void Vparam_ResData_dtor2(Vparam_ResData *thee);
00254
00259 VEXTERNC Vparam* Vparam_ctor();
00260
00266 VEXTERNC int Vparam_ctor2(Vparam *thee);
00267
00272 VEXTERNC void Vparam_dtor(Vparam **thee);
00273
00278 VEXTERNC void Vparam_dtor2(Vparam *thee);
00279
00290 VEXTERNC Vparam_ResData* Vparam_getResData(Vparam *thee,
00291     char resName[VMAX_ARGLEN]);
00292

```

```

00304 VEXTERNC Vparam_AtomData* Vparam_getAtomData(Vparam *thee,
00305     char resName[VMAX_ARGLEN], char atomName[VMAX_ARGLEN]);
00306
00335 VEXTERNC int Vparam_readFlatFile(Vparam *thee, const char *iodev,
00336     const char *iofmt, const char *thost, const char *fname);
00337
00348 VEXTERNC int Vparam_readXMLFile(Vparam *thee, const char *iodev,
00349     const char *iofmt, const char *thost, const char *fname);
00350
00351 #endif /* ifndef _VPARAM_H_ */

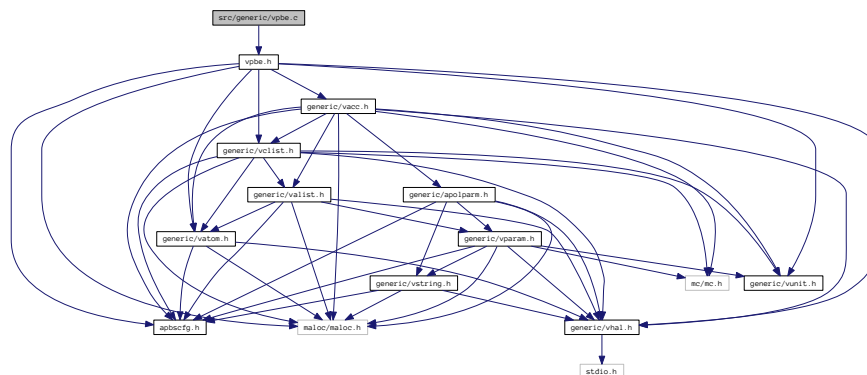
```

10.71 src/generic/vpbe.c File Reference

Class Vpbe methods.

```
#include "vpbe.h"
```

Include dependency graph for vpbe.c:



Macros

- #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)*
aaauthor 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)
- FORTTRAN stub to construct Vpbe objct.*
- VPUBLIC void [Vpbe_dtor](#) ([Vpbe](#) **thee)
- Object destructor.*
- VPUBLIC void [Vpbe_dtor2](#) ([Vpbe](#) *thee)
- FORTTRAN 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_getlons](#) ([Vpbe](#) *thee, int *nion, double ionConc[[MAXION](#)], double ionRadii[[MAXION](#)], double ionQ[[MAXION](#)])

Get information about the counterion species present.

10.71.1 Detailed Description

Class Vpbe methods.

Author

Nathan Baker

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [vpbe.c](#).

10.72 vpbe.c

```
00001
00057 #include "vpbe.h"
00058
00059 /* ////////////////////////////////////////
00060 // Class Vpbe: Private method declaration
00062 #define MAX_SPLINE_WINDOW 0.5
00063
00064 /* ////////////////////////////////////////
00065 // Class Vpbe: Inlineable methods
00067 #if !defined(VINLINE_VPBE)
00068
00069 VPUBLIC Valist* Vpbe_getValist(Vpbe *thee) {
00070
00071     VASSERT(thee != VNULL);
00072     return thee->alist;
00073 }
00074
00075
00076 VPUBLIC Vacc* Vpbe_getVacc(Vpbe *thee) {
00077
00078     VASSERT(thee != VNULL);
00079     VASSERT(thee->paramFlag);
00080     return thee->acc;
00081 }
00082
00083
00084 VPUBLIC double Vpbe_getBulkIonicStrength(Vpbe *thee) {
00085
00086     VASSERT(thee != VNULL);
00087     VASSERT(thee->paramFlag);
00088     return thee->bulkIonicStrength;
00089 }
00090
00091 VPUBLIC double Vpbe_getTemperature(Vpbe *thee) {
00092
00093     VASSERT(thee != VNULL);
00094     VASSERT(thee->paramFlag);
00095     return thee->T;
00096 }
00097
00098
00099 VPUBLIC double Vpbe_getSoluteDiel(Vpbe *thee) {
00100
00101     VASSERT(thee != VNULL);
00102     VASSERT(thee->paramFlag);
00103     return thee->soluteDiel;
00104 }
00105
00106
00107 VPUBLIC double* Vpbe_getSoluteCenter(Vpbe *thee) {
00108
00109     VASSERT(thee != VNULL);
00110     return thee->soluteCenter;
00111 }
00112
00113 VPUBLIC double Vpbe_getSolventDiel(Vpbe *thee) {
00114
00115     VASSERT(thee != VNULL);
00116     VASSERT(thee->paramFlag);
00117     return thee->solventDiel;
00118 }
00119
00120 VPUBLIC double Vpbe_getSolventRadius(Vpbe *thee) {
00121
00122     VASSERT(thee != VNULL);
00123     VASSERT(thee->paramFlag);
00124     return thee->solventRadius;
00125 }
00126
00127 VPUBLIC double Vpbe_getMaxIonRadius(Vpbe *thee) {
00128
00129     VASSERT(thee != VNULL);
00130     VASSERT(thee->paramFlag);
```

```

00131     return thee->maxIonRadius;
00132 }
00133
00134 VPUBLIC double Vpbe_getXkappa(Vpbe *thee) {
00135
00136     VASSERT(thee != VNULL);
00137     VASSERT(thee->paramFlag);
00138     return thee->xkappa;
00139 }
00140
00141 VPUBLIC double Vpbe_getDeblen(Vpbe *thee) {
00142
00143     VASSERT(thee != VNULL);
00144     VASSERT(thee->paramFlag);
00145     return thee->deblen;
00146 }
00147
00148 VPUBLIC double Vpbe_getZkappa2(Vpbe *thee) {
00149
00150     VASSERT(thee != VNULL);
00151     VASSERT(thee->paramFlag);
00152     return thee->zkappa2;
00153 }
00154
00155 VPUBLIC double Vpbe_getZmagic(Vpbe *thee) {
00156
00157     VASSERT(thee != VNULL);
00158     VASSERT(thee->paramFlag);
00159     return thee->zmagic;
00160 }
00161
00162 VPUBLIC double Vpbe_getSoluteRadius(Vpbe *thee) {
00163
00164     VASSERT(thee != VNULL);
00165     return thee->soluteRadius;
00166 }
00167
00168 VPUBLIC double Vpbe_getSoluteXlen(Vpbe *thee) {
00169
00170     VASSERT(thee != VNULL);
00171     return thee->soluteXlen;
00172 }
00173
00174 VPUBLIC double Vpbe_getSoluteYlen(Vpbe *thee) {
00175
00176     VASSERT(thee != VNULL);
00177     return thee->soluteYlen;
00178 }
00179
00180 VPUBLIC double Vpbe_getSoluteZlen(Vpbe *thee) {
00181
00182     VASSERT(thee != VNULL);
00183     return thee->soluteZlen;
00184 }
00185
00186 VPUBLIC double Vpbe_getSoluteCharge(Vpbe *thee) {
00187
00188     VASSERT(thee != VNULL);
00189     return thee->soluteCharge;
00190 }
00191
00192 /* ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00193 // Routine: Vpbe_getzmem
00194 // Purpose: This routine returns values stored in the structure thee.
00195 // Author: Michael Grabe
00197 VPUBLIC double Vpbe_getzmem(Vpbe *thee) {
00198
00199     VASSERT(thee != VNULL);
00200     VASSERT(thee->param2Flag);
00201     return thee->z_mem;
00202 }
00203
00204 /* ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00205 // Routine: Vpbe_getLmem
00206 // Purpose: This routine returns values stored in the structure thee.
00207 // Author: Michael Grabe
00209 VPUBLIC double Vpbe_getLmem(Vpbe *thee) {
00210
00211     VASSERT(thee != VNULL);
00212     VASSERT(thee->param2Flag);
00213     return thee->L;

```

```

00214 }
00215
00216 /* ////////////////////////////////////// */
00217 // Routine: Vpbe_getmembraneDiel
00218 // Purpose: This routine returns values stored in the structure thee.
00219 // Author: Michael Grabe
00221 VPUBLIC double Vpbe_getmembraneDiel(Vpbe *thee) {
00222
00223     VASSERT(thee != VNULL);
00224     VASSERT(thee->param2Flag);
00225     return thee->membraneDiel;
00226 }
00227
00228 /* ////////////////////////////////////// */
00229 // Routine: Vpbe_getmemv
00230 // Purpose: This routine returns values stored in the structure thee.
00231 // Author: Michael Grabe
00233 VPUBLIC double Vpbe_getmemv(Vpbe *thee) {
00234
00235     VASSERT(thee != VNULL);
00236     VASSERT(thee->param2Flag);
00237     return thee->V;
00238 }
00239
00240 #endif /* if !defined(VINLINE_VPBE) */
00241
00242 /* ////////////////////////////////////// */
00243 // Class Vpbe: Non-inlineable methods
00244
00246 VPUBLIC Vpbe* Vpbe_ctor(Valist *alist, int ionNum, double *ionConc,
00247                        double *ionRadii, double *ionQ, double T,
00248                        double soluteDiel, double solventDiel,
00249                        double solventRadius, int focusFlag, double sdens,
00250                        double z_mem, double L, double membraneDiel, double V ) {
00251
00252     /* Set up the structure */
00253     Vpbe *thee = VNULL;
00254     thee = (Vpbe*)Vmem_malloc(VNULL, 1, sizeof(Vpbe) );
00255     VASSERT( thee != VNULL);
00256     VASSERT( Vpbe_ctor2(thee, alist, ionNum, ionConc, ionRadii, ionQ,
00257                        T, soluteDiel, solventDiel, solventRadius, focusFlag, sdens,
00258                        z_mem, L, membraneDiel, V) );
00259
00260     return thee;
00261 }
00262
00263
00264 VPUBLIC int Vpbe_ctor2(Vpbe *thee, Valist *alist, int ionNum,
00265                      double *ionConc, double *ionRadii,
00266                      double *ionQ, double T, double soluteDiel,
00267                      double solventDiel, double solventRadius, int focusFlag,
00268                      double sdens, double z_mem, double L, double membraneDiel,
00269                      double V) {
00270
00271     int i, iatom, inhash[3];
00272     double atomRadius;
00273     Vatom *atom;
00274     double center[3] = {0.0, 0.0, 0.0};
00275     double lower_corner[3] = {0.0, 0.0, 0.0};
00276     double upper_corner[3] = {0.0, 0.0, 0.0};
00277     double disp[3], dist, radius, charge, xmin, xmax, ymin, ymax, zmin, zmax;
00278     double x, y, z, netCharge;
00279     double nhash[3];
00280     const double N_A = 6.022045000e+23;
00281     const double e_c = 4.803242384e-10;
00282     const double k_B = 1.380662000e-16;
00283     const double pi = 4. * VATAN(1.);
00284
00285     /* Set up memory management object */
00286     thee->vmem = Vmem_ctor("APBS::VPBE");
00287
00288     VASSERT(thee != VNULL);
00289     if (alist == VNULL) {
00290         Vnm_print(2, "Vpbe_ctor2: Got null pointer to Valist object!\n");
00291         return 0;
00292     }
00293
00294     /* **** STUFF THAT GETS DONE FOR EVERYONE **** */
00295     /* Set pointers */
00296     thee->alist = alist;
00297     thee->paramFlag = 0;

```

```

00298
00299     /* Determine solute center */
00300     center[0] = thee->alist->center[0];
00301     center[1] = thee->alist->center[1];
00302     center[2] = thee->alist->center[2];
00303     thee->soluteCenter[0] = center[0];
00304     thee->soluteCenter[1] = center[1];
00305     thee->soluteCenter[2] = center[2];
00306
00307     /* Determine solute length and charge*/
00308     radius = 0;
00309     atom = Valist_getAtom(thee->alist, 0);
00310     xmin = Vatom_getPosition(atom)[0];
00311     xmax = Vatom_getPosition(atom)[0];
00312     ymin = Vatom_getPosition(atom)[1];
00313     ymax = Vatom_getPosition(atom)[1];
00314     zmin = Vatom_getPosition(atom)[2];
00315     zmax = Vatom_getPosition(atom)[2];
00316     charge = 0;
00317     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
00318         atom = Valist_getAtom(thee->alist, iatom);
00319         atomRadius = Vatom_getRadius(atom);
00320         x = Vatom_getPosition(atom)[0];
00321         y = Vatom_getPosition(atom)[1];
00322         z = Vatom_getPosition(atom)[2];
00323         if ((x+atomRadius) > xmax) xmax = x + atomRadius;
00324         if ((x-atomRadius) < xmin) xmin = x - atomRadius;
00325         if ((y+atomRadius) > ymax) ymax = y + atomRadius;
00326         if ((y-atomRadius) < ymin) ymin = y - atomRadius;
00327         if ((z+atomRadius) > zmax) zmax = z + atomRadius;
00328         if ((z-atomRadius) < zmin) zmin = z - atomRadius;
00329         disp[0] = (x - center[0]);
00330         disp[1] = (y - center[1]);
00331         disp[2] = (z - center[2]);
00332         dist = (disp[0]*disp[0] + (disp[1]*disp[1]) + (disp[2]*disp[2]));
00333         dist = VSQRT(dist) + atomRadius;
00334         if (dist > radius) radius = dist;
00335         charge += Vatom_getCharge(Valist_getAtom(thee->
alist, iatom));
00336     }
00337     thee->soluteRadius = radius;
00338     Vnm_print(0, "Vpbe_ctor2: solute radius = %g\n", radius);
00339     thee->soluteXlen = xmax - xmin;
00340     thee->soluteYlen = ymax - ymin;
00341     thee->soluteZlen = zmax - zmin;
00342     Vnm_print(0, "Vpbe_ctor2: solute dimensions = %g x %g x %g\n",
thee->soluteXlen, thee->soluteYlen, thee->
soluteZlen);
00343
00344     thee->soluteCharge = charge;
00345     Vnm_print(0, "Vpbe_ctor2: solute charge = %g\n", charge);
00346
00347     /* Set parameters */
00348     thee->numIon = ionNum;
00349     if (thee->numIon >= MAXION) {
00350         Vnm_print(2, "Vpbe_ctor2: Too many ion species (MAX = %d)!\n",
MAXION);
00351         return 0;
00352     }
00353
00354     thee->bulkIonicStrength = 0.0;
00355     thee->maxIonRadius = 0.0;
00356     netCharge = 0.0;
00357     for (i=0; i<thee->numIon; i++) {
00358         thee->ionConc[i] = ionConc[i];
00359         thee->ionRadii[i] = ionRadii[i];
00360         if (ionRadii[i] > thee->maxIonRadius) thee->maxIonRadius = ionRadii[i];
00361         thee->ionQ[i] = ionQ[i];
00362         thee->bulkIonicStrength += (0.5*ionConc[i]*VSQR(ionQ[i]));
00363         netCharge += (ionConc[i]*ionQ[i]);
00364     }
00365 #ifndef VAPBSQUIET
00366     Vnm_print(1, " Vpbe_ctor: Using max ion radius (%g A) for exclusion \
function\n", thee->maxIonRadius);
00367 #endif
00368 #endif
00369     if (VABS(netCharge) > VSMALL) {
00370         Vnm_print(2, "Vpbe_ctor2: You have a counterion charge imbalance!\n");
00371         Vnm_print(2, "Vpbe_ctor2: Net charge conc. = %g M\n", netCharge);
00372         return 0;
00373     }
00374     thee->T = T;
00375     thee->soluteDiel = soluteDiel;
00376     thee->solventDiel = solventDiel;

```



```

00377     thee->solventRadius = solventRadius;
00378
00379     /* Compute parameters:
00380     *
00381     *  $\kappa^2 = (8 \pi N_A e_c^2) I_s / (1000 \epsilon_w k_B T)$ 
00382     *  $\kappa = 0.325567 * I_s^{1/2}$  angstroms-1
00383     *  $\text{deblen} = 1 / \kappa$ 
00384     *  $= 3.071564378 * I_s^{1/2}$  angstroms
00385     *  $\bar{\kappa}^2 = \epsilon_w * \kappa^2$ 
00386     *  $\text{zmagic} = (4 * \pi * e_c^2) / (k_B T)$  (we scale the diagonal later)
00387     *  $= 7046.528838$ 
00388     */
00389     if (thee->T == 0.0) {
00390         Vnm_print(2, "Vpbe_ctor2: You set the temperature to 0 K.\n");
00391         Vnm_print(2, "Vpbe_ctor2: That violates the 3rd Law of Thermo!");
00392         return 0;
00393     }
00394     if (thee->bulkIonicStrength == 0.) {
00395         thee->xkappa = 0.;
00396         thee->deblen = 0.;
00397         thee->zkappa2 = 0.;
00398     } else {
00399         thee->xkappa = VSQRT( thee->bulkIonicStrength * 1.0e-16 *
00400             ((8.0 * pi * N_A * e_c * e_c) /
00401             (1000.0 * thee->solventDiel * k_B * T))
00402         );
00403         thee->deblen = 1. / thee->xkappa;
00404         thee->zkappa2 = thee->solventDiel * VSQR(thee->xkappa);
00405     }
00406     Vnm_print(0, "Vpbe_ctor2: bulk ionic strength = %g\n",
00407         thee->bulkIonicStrength);
00408     Vnm_print(0, "Vpbe_ctor2: xkappa = %g\n", thee->xkappa);
00409     Vnm_print(0, "Vpbe_ctor2: Debye length = %g\n", thee->deblen);
00410     Vnm_print(0, "Vpbe_ctor2: zkappa2 = %g\n", thee->zkappa2);
00411     thee->zmagic = ((4.0 * pi * e_c * e_c) / (k_B * thee->T)) * 1.0e+8;
00412     Vnm_print(0, "Vpbe_ctor2: zmagic = %g\n", thee->zmagic);
00413
00414     /* Compute accessibility objects:
00415     * - Allow for extra room in the case of spline windowing
00416     * - Place some limits on the size of the hash table in the case of very
00417     *   large molecules
00418     */
00419     if (thee->maxIonRadius > thee->solventRadius)
00420         radius = thee->maxIonRadius + MAX_SPLINE_WINDOW;
00421     else radius = thee->solventRadius + MAX_SPLINE_WINDOW;
00422
00423     nhash[0] = (thee->soluteXlen)/0.5;
00424     nhash[1] = (thee->soluteYlen)/0.5;
00425     nhash[2] = (thee->soluteZlen)/0.5;
00426     for (i=0; i<3; i++) inhash[i] = (int) (nhash[i]);
00427
00428     for (i=0; i<3; i++){
00429         if (inhash[i] < 3) inhash[i] = 3;
00430         if (inhash[i] > MAX_HASH_DIM) inhash[i] = MAX_HASH_DIM;
00431     }
00432     Vnm_print(0, "Vpbe_ctor2: Constructing Vclist with %d x %d x %d table\n",
00433         inhash[0], inhash[1], inhash[2]);
00434
00435     thee->clist = Vclist_ctor(thee->alist, radius, inhash,
00436         CLIST_AUTO_DOMAIN, lower_corner, upper_corner);
00437
00438     VASSERT(thee->clist != VNULL);
00439     thee->acc = Vacc_ctor(thee->alist, thee->clist, sdens);
00440
00441     VASSERT(thee->acc != VNULL);
00442
00443     /* SMPBE Added */
00444     thee->smsize = 0.0;
00445     thee->smvolume = 0.0;
00446     thee->ipkey = 0;
00447
00448     thee->paramFlag = 1;
00449
00450     /*-----*/
00451     /* added by Michael Grabe */
00452     /*-----*/
00453
00454     thee->z_mem = z_mem;
00455     thee->L = L;
00456     thee->membraneDiel = membraneDiel;
00457     thee->V = V;

```

```

00458
00459     if (V != 0.0) thee->param2Flag = 1;
00460     else thee->param2Flag = 0;
00461
00462     /*-----*/
00463
00464     return 1;
00465 }
00466
00467 VPUBLIC void Vpbe_dtor(Vpbe **thee) {
00468     if ((*thee) != VNULL) {
00469         Vpbe_dtor2(*thee);
00470         Vmem_free(VNULL, 1, sizeof(Vpbe), (void **)thee);
00471         (*thee) = VNULL;
00472     }
00473 }
00474
00475 VPUBLIC void Vpbe_dtor2(Vpbe *thee) {
00476     Vclist_dtor(&(thee->clist));
00477     Vacc_dtor(&(thee->acc));
00478     Vmem_dtor(&(thee->vmem));
00479 }
00480
00481 VPUBLIC double Vpbe_getCoulombEnergy1(Vpbe *thee) {
00482     int i, j, k, natoms;
00483
00484     double dist, *ipos, *jpos, icharge, jcharge;
00485     double energy = 0.0;
00486     double eps, T;
00487     Vatom *iatom, *jatom;
00488     Valist *alist;
00489
00490     VASSERT(thee != VNULL);
00491     alist = Vpbe_getValist(thee);
00492     VASSERT(alist != VNULL);
00493     natoms = Valist_getNumberAtoms(alist);
00494
00495     /* Do the sum */
00496     for (i=0; i<natoms; i++) {
00497         iatom = Valist_getAtom(alist,i);
00498         icharge = Vatom_getCharge(iatom);
00499         ipos = Vatom_getPosition(iatom);
00500         for (j=i+1; j<natoms; j++) {
00501             jatom = Valist_getAtom(alist,j);
00502             jcharge = Vatom_getCharge(jatom);
00503             jpos = Vatom_getPosition(jatom);
00504             dist = 0;
00505             for (k=0; k<3; k++) dist += ((ipos[k]-jpos[k])*(ipos[k]-jpos[k]));
00506             dist = VSQRT(dist);
00507             energy = energy + icharge*jcharge/dist;
00508         }
00509     }
00510 }
00511
00512     /* Convert the result to J */
00513     T = Vpbe_getTemperature(thee);
00514     eps = Vpbe_getSoluteDiel(thee);
00515     energy = energy*Vunit_ec*Vunit_ec/(4*Vunit_pi*
Vunit_eps0*eps*(1.0e-10));
00516
00517     /* Scale by Boltzmann energy */
00518     energy = energy/(Vunit_kb*T);
00519
00520     return energy;
00521 }
00522
00523 VPUBLIC unsigned long int Vpbe_memChk(Vpbe *thee) {
00524     unsigned long int memUse = 0;
00525
00526     if (thee == VNULL) return 0;
00527
00528     memUse = memUse + sizeof(Vpbe);
00529     memUse = memUse + (unsigned long int)Vacc_memChk(thee->acc);
00530
00531     return memUse;
00532 }
00533
00534
00535 VPUBLIC int Vpbe_getIons(Vpbe *thee, int *nion, double ionConc[
MAXION],
00536     double ionRadii[MAXION], double ionQ[MAXION]) {

```

```

00537
00538     int i;
00539
00540     VASSERT(thee != VNULL);
00541
00542     *nion = thee->numIon;
00543     for (i=0; i<(*nion); i++) {
00544         ionConc[i] = thee->ionConc[i];
00545         ionRadii[i] = thee->ionRadii[i];
00546         ionQ[i] = thee->ionQ[i];
00547     }
00548
00549     return *nion;
00550 }

```

10.73 src/generic/vpbe.h File Reference

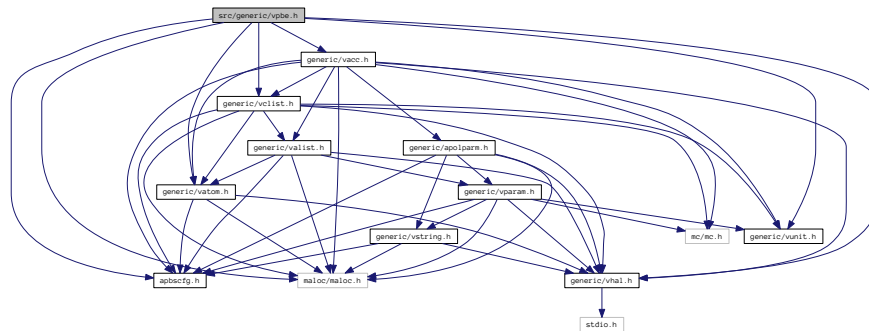
Contains declarations for class Vpbe.

```

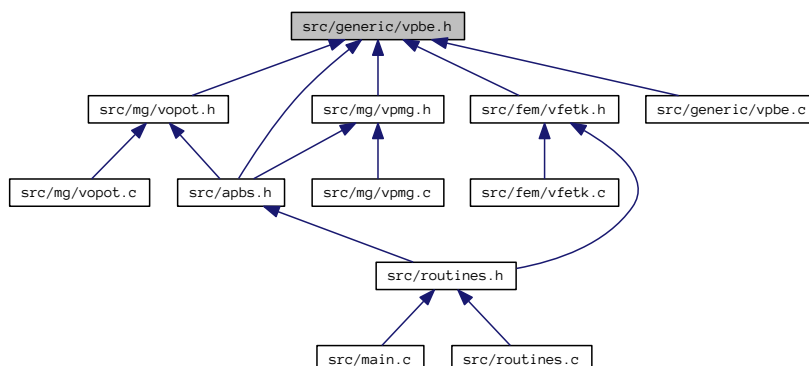
#include "apbscfg.h"
#include "malloc/malloc.h"
#include "generic/vhal.h"
#include "generic/vunit.h"
#include "generic/vatom.h"
#include "generic/vacc.h"
#include "generic/vclist.h"

```

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)*
aaauthor 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)
- FORTTRAN 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)
- FORTTRAN 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.*

10.73.1 Detailed Description

Contains declarations for class `Vpbe`.

Version

`Id`

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [vpbe.h](#).

10.74 vpbe.h

```

00001
00066 #ifndef _VPBE_H_
00067 #define _VPBE_H_
00068
00069 #include "apbscfg.h"
00070
00071 #include "malloc/malloc.h"
00072
00073 #include "generic/vhal.h"
00074 #include "generic/vunit.h"
00075 #include "generic/vatom.h"
00076 #include "generic/vacc.h"
00077 #include "generic/vclist.h"
00078
00084 struct sVpbe {
00085
00086     Vmem *vmem;
00088     Valist *alist;
00089     Vclist *clist;
00090     Vacc *acc;
00092     double T;
00093     double soluteDiel;
00094     double solventDiel;
00095     double solventRadius;
00099     double bulkIonicStrength;
00100     double maxIonRadius;
00103     int numIon;
00104     double ionConc[MAXION];
00105     double ionRadii[MAXION];
00106     double ionQ[MAXION];
00108     double xkappa;
00109     double deblen;
00110     double zkappa2;
00111     double zmagic;
00113     double soluteCenter[3];
00114     double soluteRadius;
00115     double soluteXlen;
00116     double soluteYlen;
00117     double soluteZlen;
00118     double soluteCharge;
00120     double smvolume;
00121     double smsize;
00122     int ipkey;
00125     int paramFlag;
00127     /*-----*/
00128     /* Added by Michael Grabe */
00129     /*-----*/
00130
00131     double z_mem;
00132     double L;
00133     double membraneDiel;
00134     double V;
00135     int param2Flag;
00136     /*-----*/
00137
00138 };
00139
00144 typedef struct sVpbe Vpbe;
00145
00146 /* ////////////////////////////////////// */
00147 // Class Vpbe: Inlineable methods (vpbe.c)
00148
00150 #if !defined(VINLINE_VPBE)
00151
00158 VEXTERNC Valist* Vpbe_getValist(Vpbe *thee);
00159
00166 VEXTERNC Vacc* Vpbe_getVacc(Vpbe *thee);
00167
00174 VEXTERNC double Vpbe_getBulkIonicStrength(Vpbe *thee);
00175
00182 VEXTERNC double Vpbe_getMaxIonRadius(Vpbe *thee);
00183
00190 VEXTERNC double Vpbe_getTemperature(Vpbe *thee);
00191
00198 VEXTERNC double Vpbe_getSoluteDiel(Vpbe *thee);
00199
00206 VEXTERNC double Vpbe_getGamma(Vpbe *thee);
00207

```

```

00214 VEXTERNC double  Vpbe_getSoluteRadius(Vpbe *thee);
00215
00222 VEXTERNC double  Vpbe_getSoluteXlen(Vpbe *thee);
00223
00230 VEXTERNC double  Vpbe_getSoluteYlen(Vpbe *thee);
00231
00238 VEXTERNC double  Vpbe_getSoluteZlen(Vpbe *thee);
00239
00246 VEXTERNC double* Vpbe_getSoluteCenter(Vpbe *thee);
00247
00254 VEXTERNC double  Vpbe_getSoluteCharge(Vpbe *thee);
00255
00262 VEXTERNC double  Vpbe_getSolventDiel(Vpbe *thee);
00263
00270 VEXTERNC double  Vpbe_getSolventRadius(Vpbe *thee);
00271
00278 VEXTERNC double  Vpbe_getXkappa(Vpbe *thee);
00279
00286 VEXTERNC double  Vpbe_getDeblen(Vpbe *thee);
00287
00294 VEXTERNC double  Vpbe_getZkappa2(Vpbe *thee);
00295
00302 VEXTERNC double  Vpbe_getZmagic(Vpbe *thee);
00303
00304 /*-----*/
00305 /* Added by Michael Grabe */
00306 /*-----*/
00307
00314 VEXTERNC double  Vpbe_getzmem(Vpbe *thee);
00315
00322 VEXTERNC double  Vpbe_getLmem(Vpbe *thee);
00323
00330 VEXTERNC double  Vpbe_getmembraneDiel(Vpbe *thee);
00331
00337 VEXTERNC double  Vpbe_getmemv(Vpbe *thee);
00338
00339 /*-----*/
00340
00341 #else /* if defined(VINLINE_VPBE) */
00342 #   define Vpbe_getValist(thee) ((thee)->alist)
00343 #   define Vpbe_getVacc(thee) ((thee)->acc)
00344 #   define Vpbe_getBulkIonicStrength(thee) ((thee)->bulkIonicStrength)
00345 #   define Vpbe_getTemperature(thee) ((thee)->T)
00346 #   define Vpbe_getSoluteDiel(thee) ((thee)->soluteDiel)
00347 #   define Vpbe_getSoluteCenter(thee) ((thee)->soluteCenter)
00348 #   define Vpbe_getSoluteRadius(thee) ((thee)->soluteRadius)
00349 #   define Vpbe_getSoluteXlen(thee) ((thee)->soluteXlen)
00350 #   define Vpbe_getSoluteYlen(thee) ((thee)->soluteYlen)
00351 #   define Vpbe_getSoluteZlen(thee) ((thee)->soluteZlen)
00352 #   define Vpbe_getSoluteCharge(thee) ((thee)->soluteCharge)
00353 #   define Vpbe_getSolventDiel(thee) ((thee)->solventDiel)
00354 #   define Vpbe_getSolventRadius(thee) ((thee)->solventRadius)
00355 #   define Vpbe_getMaxIonRadius(thee) ((thee)->maxIonRadius)
00356 #   define Vpbe_getXkappa(thee) ((thee)->xkappa)
00357 #   define Vpbe_getDeblen(thee) ((thee)->deblen)
00358 #   define Vpbe_getZkappa2(thee) ((thee)->zkappa2)
00359 #   define Vpbe_getZmagic(thee) ((thee)->zmagic)
00360
00361 /*-----*/
00362 /* Added by Michael Grabe */
00363 /*-----*/
00364
00365 #   define Vpbe_getzmem(thee) ((thee)->z_mem)
00366 #   define Vpbe_getLmem(thee) ((thee)->L)
00367 #   define Vpbe_getmembraneDiel(thee) ((thee)->membraneDiel)
00368 #   define Vpbe_getmemv(thee) ((thee)->V)
00369
00370 /*-----*/
00371
00372
00373 #endif /* if !defined(VINLINE_VPBE) */
00374
00375 /* ////////////////////////////////////// */
00376 // Class Vpbe: Non-Inlineable methods (vpbe.c)
00377
00399 VEXTERNC Vpbe*  Vpbe_ctor(
00400     Valist *alist,
00401     int ionNum,
00402     double *ionConc,
00403     double *ionRadii,
00404     double *ionQ,

```



```

00405             double T,
00406             double soluteDiel,
00407             double solventDiel,
00408             double solventRadius,
00409             int focusFlag,
00410             double sdens,
00411             double z_mem,
00412             double L,
00413             double membraneDiel,
00414             double V
00415         );
00416
00437 VEXTERNC int    Vpbe_ctor2(
00438                 Vpbe *thee,
00439                 Valist *alist,
00440                 int ionNum,
00441                 double *ionConc,
00442                 double *ionRadii,
00443                 double *ionQ,
00444                 double T,
00445                 double soluteDiel,
00446                 double solventDiel,
00447                 double solventRadius,
00448                 int focusFlag,
00449                 double sdens,
00450                 double z_mem,
00451                 double L,
00452                 double membraneDiel,
00453                 double V
00454             );
00455
00466 VEXTERNC int    Vpbe_getIons(Vpbe *thee, int *nion, double ionConc[MAXION],
00467                             double ionRadii[MAXION], double ionQ[MAXION]);
00468
00474 VEXTERNC void   Vpbe_dtor(Vpbe **thee);
00475
00481 VEXTERNC void   Vpbe_dtor2(Vpbe *thee);
00482
00497 VEXTERNC double Vpbe_getCoulombEnergy1(Vpbe *thee);
00498
00506 VEXTERNC unsigned long int Vpbe_memChk(Vpbe *thee);
00507
00508 #endif /* ifndef _VPBE_H_ */

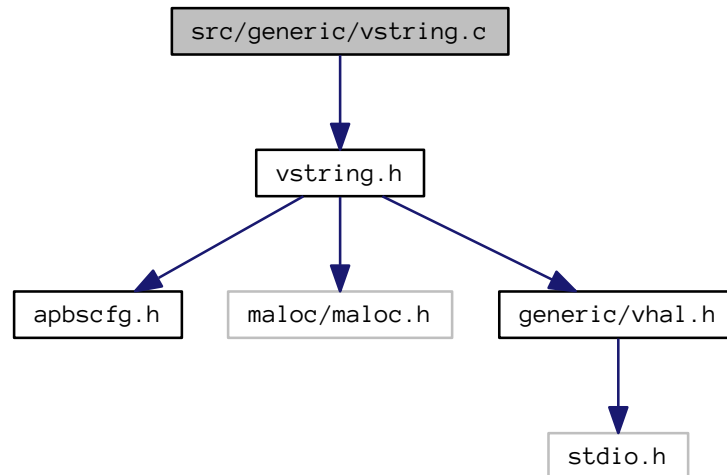
```

10.75 src/generic/vstring.c File Reference

Class Vstring methods.

```
#include "vstring.h"
```

Include dependency graph for vstring.c:



Functions

- `VPUBLIC int Vstring_strcasecmp (const char *s1, const char *s2)`
Case-insensitive string comparison (BSD standard)
- `VPUBLIC int Vstring_isdigit (const char *tok)`
A modified sscanf that examines the complete string.
- `char * Vstring_wrappedtext (const char *str, int right_margin, int left_padding)`

10.75.1 Detailed Description

Class Vstring methods.

Author

Nathan Baker

Version

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vstring.c](#).

10.76 vstring.c

```

00001
00057 #include "vstring.h"
00058
00059 /* ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////// */
00060 // Routine: Vstring_strcasecmp
00061 //
00062 // Copyright (c) 1988-1993 The Regents of the University of
00063 // California.
00064 // Copyright (c) 1995-1996 Sun Microsystems, Inc.
00066 VPUBLIC int Vstring_strcasecmp(const char *s1, const char *s2) {
00067
00068 #if !defined(HAVE_STRCASECMP)
00069     unsigned char charmap[] = {
00070         0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,

```

```

00071     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
00072     0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
00073     0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f,
00074     0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
00075     0x28, 0x29, 0x2a, 0x2b, 0x2c, 0x2d, 0x2e, 0x2f,
00076     0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
00077     0x38, 0x39, 0x3a, 0x3b, 0x3c, 0x3d, 0x3e, 0x3f,
00078     0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47,
00079     0x48, 0x49, 0x4a, 0x4b, 0x4c, 0x4d, 0x4e, 0x4f,
00080     0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57,
00081     0x58, 0x59, 0x5a, 0x5b, 0x5c, 0x5d, 0x5e, 0x5f,
00082     0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67,
00083     0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f,
00084     0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77,
00085     0x78, 0x79, 0x7a, 0x7b, 0x7c, 0x7d, 0x7e, 0x7f,
00086     0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
00087     0x88, 0x89, 0x8a, 0x8b, 0x8c, 0x8d, 0x8e, 0x8f,
00088     0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97,
00089     0x98, 0x99, 0x9a, 0x9b, 0x9c, 0x9d, 0x9e, 0x9f,
00090     0xa0, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7,
00091     0xa8, 0xa9, 0xaa, 0xab, 0xac, 0xad, 0xae, 0xaf,
00092     0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7,
00093     0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0xbd, 0xbe, 0xbf,
00094     0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7,
00095     0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf,
00096     0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7,
00097     0xd8, 0xd9, 0xda, 0xdb, 0xdc, 0xdd, 0xde, 0xdf,
00098     0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7,
00099     0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef,
00100     0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
00101     0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff,
00102 };
00103
00104 unsigned char u1, u2;
00105
00106 for ( ; ; s1++, s2++) {
00107     u1 = (unsigned char) *s1;
00108     u2 = (unsigned char) *s2;
00109     if ((u1 == '\0') || (charmap[u1] != charmap[u2])) {
00110         break;
00111     }
00112 }
00113 return charmap[u1] - charmap[u2];
00114
00115 #else
00116     return strcasecmp(s1, s2);
00117
00118 #endif
00119 }
00120
00121 }
00122
00123 /* ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00124 // Routine:  Vstring_isdigit
00125 //
00126 //           Improves upon sscanf to see if a token is an int or not
00127 //
00128 //           Returns isdigit: 1 if a digit, 0 otherwise
00129 //
00130 // PUBLIC int Vstring_isdigit(const char *tok) {
00131 //     int i, isdigit, ti;
00132 //     char checkchar[1];
00133 //     char name[VMAX_BUF_SIZE];
00134 //     strcpy(name, tok);
00135 //     isdigit = 1;
00136 //     for(i=0; ; i++){
00137 //         checkchar[0] = name[i];
00138 //         if (name[i] == '\0'){
00139 //             break;
00140 //         }
00141 //         if (sscanf(checkchar, "%d", &ti) != 1){
00142 //             isdigit = 0;
00143 //             break;
00144 //         }
00145 //     }
00146 //     return isdigit;
00147 // }
00148
00149
00150 char* Vstring_wrappedtext(const char* str, int right_margin, int left_padding)
00151 {
00152     int span = right_margin - left_padding;

```

```

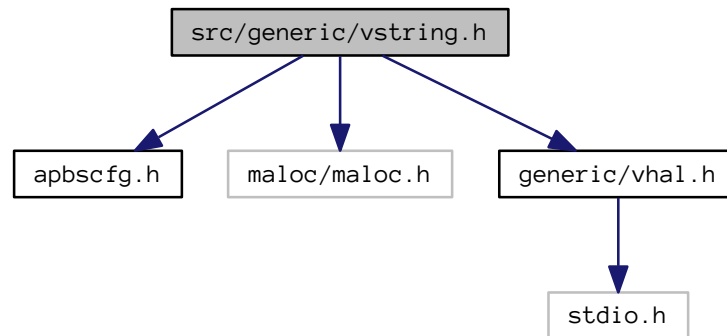
00158     int i = 0;
00159     int k = 0;
00160     int j = 0;
00161     int line_len = 0;
00162     int hyphenate = 0;
00163     char* wrap_str;
00164     int wrap_len;
00165     int len = strlen( str );
00166
00167     if( len == 0 )
00168         return VNULL;
00169
00170     wrap_str = (char*)malloc( len * sizeof(char) );
00171     wrap_len = len;
00172
00173     do
00174     {
00175         if( str[i] == ' ' )
00176         {
00177             i++;
00178         }
00179         else
00180         {
00181             if( k + right_margin + 2 > wrap_len )
00182             {
00183                 wrap_len += right_margin + 2;
00184                 wrap_str = (char*)realloc( wrap_str, wrap_len * sizeof( char ) );
00185             }
00186
00187             if( i + span >= len )
00188             {
00189                 hyphenate = 0;
00190                 line_len = len - i;
00191             }
00192             else
00193             {
00194                 j = span;
00195                 do
00196                 {
00197                     if( str[ i + j ] == ' ' )
00198                     {
00199                         hyphenate = 0;
00200                         line_len = j;
00201                         break;
00202                     }
00203                     else if( j == 0 )
00204                     {
00205                         hyphenate = 1;
00206                         line_len = span - 1;
00207                         break;
00208                     }
00209                     else
00210                     {
00211                         j--;
00212                     }
00213                 } while( 1 );
00214
00215                 memset( wrap_str + k, ' ', left_padding * sizeof( char ) );
00216                 k += left_padding;
00217
00218                 memcpy( wrap_str + k, str + i, line_len * sizeof( char ) );
00219                 k += line_len;
00220                 i += line_len;
00221
00222                 if( hyphenate )
00223                     wrap_str[k++] = '-';
00224
00225                 wrap_str[k++] = '\n';
00226
00227                 wrap_str[k] = '\0';
00228             }
00229         } while( i < len );
00230
00231     } while( i < len );
00232
00233     return wrap_str;
00234 }
00235
00236
00237
00238

```

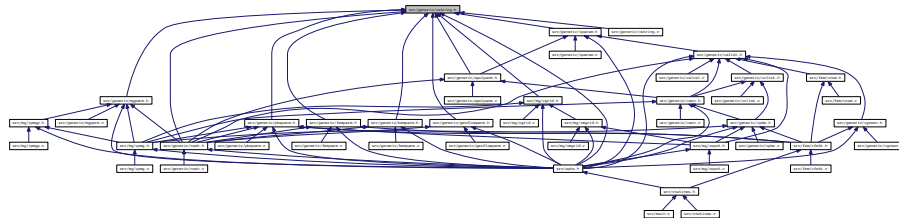
10.77 src/generic/vstring.h File Reference

Contains declarations for class Vstring.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/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.
- char * [Vstring_wrappedtext](#) (const char *str, int right_margin, int left_padding)

10.77.1 Detailed Description

Contains declarations for class Vstring.

Version`Id`**Author**

Nathan A. Baker

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*   Nathan A. Baker (nathan.baker@pnnl.gov)
*   Pacific Northwest National Laboratory
*
*   Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* - Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of 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.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS

```


- #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

10.79.1 Detailed Description

Contains a collection of useful constants and conversion factors.

Author

Nathan Baker
Nathan A. Baker

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
```

```

* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vunit.h](#).

10.80 vunit.h

```

00001
00063 #ifndef _VUNIT_H_
00064 #define _VUNIT_H_
00065
00068 #define Vunit_J_to_cal 4.1840000e+00
00069
00072 #define Vunit_cal_to_J 2.3900574e-01
00073
00076 #define Vunit_amu_to_kg 1.6605402e-27
00077
00080 #define Vunit_kg_to_amu 6.0221367e+26
00081
00084 #define Vunit_ec_to_C 1.6021773e-19
00085
00088 #define Vunit_C_to_ec 6.2415065e+18
00089
00092 #define Vunit_ec 1.6021773e-19
00093
00096 #define Vunit_kb 1.3806581e-23
00097
00100 #define Vunit_Na 6.0221367e+23
00101
00104 #define Vunit_pi VPI
00105
00108 #define Vunit_eps0 8.8541878e-12
00109
00112 #define Vunit_esu_ec2A 3.3206364e+02
00113
00116 #define Vunit_esu_kb 1.9871913e-03
00117
00118 #endif /* ifndef _VUNIT_H_ */

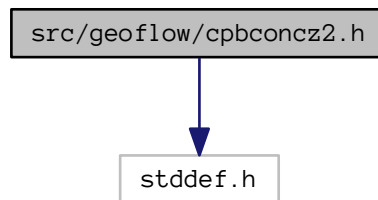
```

10.81 src/geoflow/cpbconcz2.h File Reference

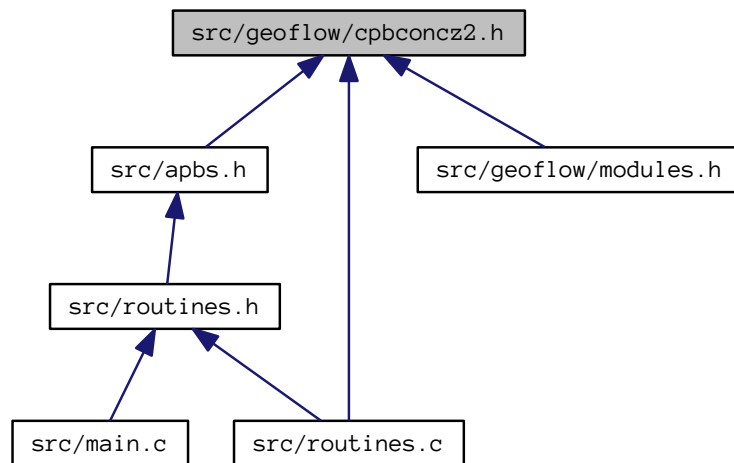
some definitons for APBS

```
#include <stddef.h>
```

Include dependency graph for cpbconcz2.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- [struct _GeoflowOutput](#)
- [struct _GeoflowInput](#)

Macros

- `#define MAXATOMS 15000`
- `#define XYZRWIDTH 4`

Typedefs

- `typedef struct _GeoflowOutput GeoflowOutput`
- `typedef struct _GeoflowInput GeoflowInput`

Functions

- `GeoflowOutput geoflowSolvation` (double xyzr[MAXATOMS][XYZRWIDTH], size_t natm, `GeoflowInput` gfin)

10.81.1 Detailed Description

some definitons for APBS

Author

Andrew Stevens,Kyle Monson

Version

\$Id\$

Attention

```

///
/// APBS -- Adaptive Poisson-Boltzmann Solver
///
/// Nathan A. Baker (nathan.baker@pnnl.gov)
/// Pacific Northwest National Laboratory
///
/// Additional contributing authors listed in the code documentation.
///
/// Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
/// Pacific Northwest National Laboratory, operated by Battelle Memorial
/// Institute, Pacific Northwest Division for the U.S. Department of Energy.
///
/// Portions Copyright (c) 2002-2010, Washington University in St. Louis.
/// Portions Copyright (c) 2002-2010, Nathan A. Baker.
/// Portions Copyright (c) 1999-2002, The Regents of the University of
/// California.
/// Portions Copyright (c) 1995, Michael Holst.
/// All rights reserved.
///
/// Redistribution and use in source and binary forms, with or without
/// modification, are permitted provided that the following conditions are met:
///
/// Redistributions of source code must retain the above copyright notice, this
/// list of conditions and the following disclaimer.
///
/// Redistributions in binary form must reproduce the above copyright notice,
/// this list of conditions and the following disclaimer in the documentation
/// and/or other materials provided with the distribution.
///
/// Neither the name of the developer nor the names of its contributors may be

```

```

/// used to endorse or promote products derived from this software without
/// specific prior written permission.
///
/// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
/// AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
/// IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
/// ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
/// LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
/// CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
/// SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
/// INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
/// CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
/// ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
/// THE POSSIBILITY OF SUCH DAMAGE.
///
///

```

Definition in file [cpbconcz2.h](#).

10.82 cpbconcz2.h

```

00001
00055 #pragma once
00056
00057 // From the original f90 code; need to keep this
00058 // constant since the f90 routines that we call from here
00059 // depend on it.
00060 #define MAXATOMS 15000
00061
00062 // Four because: 0-2 => pos, 3 => radius
00063 #define XYZRWIDTH 4
00064
00065 typedef struct _GeoflowOutput
00066 {
00067     double area,
00068     volume,
00069     attint,
00070     sumpot,
00071     totalSolvation,
00072     nonpolarSolvation,
00073     elecSolvation;
00074 } GeoflowOutput;
00075
00076 typedef struct _GeoflowInput
00077 {
00078     double* dcel;
00079     int ffmodel;
00080     double extvalue;
00081     double* pqr;
00082     int maxstep;
00083     double crevalue;
00084     int iadi;
00085     double tottf;
00086     double* ljepsilon;
00087     double alpha;
00088     int igfin;
00089     double epsilons;
00090     double epsilonp;
00091     int idacsl;
00092     double tol;
00093     int iterf;
00094     double tpb;
00095     int itert;
00096     double pres;
00097     double gama;
00098     double tauval;
00099     double prob;
00100     int vdwdispersion;
00101     double sigmas;
00102     double density;
00103     double epsilonw;
00104 } GeoflowInput;
00105

```

10.83 src/geoflow/Mat.h File Reference

```
#include <cmath>
#include <vector>
#include <Eigen/Core>
Include dependency graph for Mat.h:
```



- class `Mat< T >`
- class `Mat< T >`
- struct `Stencil< T >`

[illegible]

Functions

- double **dot** (double x, double y, double z)

10.83.1 Detailed Description

Wrapper class for 2d and 3d arrays.

Author

Andrew Stevens

Version

\$Id\$

Attention

```

///
/// APBS -- Adaptive Poisson-Boltzmann Solver
///
/// Nathan A. Baker (nathan.baker@pnnl.gov)
/// Pacific Northwest National Laboratory
///
/// Additional contributing authors listed in the code documentation.
///
/// Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
/// Pacific Northwest National Laboratory, operated by Battelle Memorial
/// Institute, Pacific Northwest Division for the U.S. Department of Energy.
///
/// Portions Copyright (c) 2002-2010, Washington University in St. Louis.
/// Portions Copyright (c) 2002-2010, Nathan A. Baker.
/// Portions Copyright (c) 1999-2002, The Regents of the University of
/// California.
/// Portions Copyright (c) 1995, Michael Holst.
/// All rights reserved.
///
/// Redistribution and use in source and binary forms, with or without
/// modification, are permitted provided that the following conditions are met:
///
/// Redistributions of source code must retain the above copyright notice, this
/// list of conditions and the following disclaimer.
///
/// Redistributions in binary form must reproduce the above copyright notice,
/// this list of conditions and the following disclaimer in the documentation
/// and/or other materials provided with the distribution.
///
/// Neither the name of the developer nor the names of its contributors may be
/// used to endorse or promote products derived from this software without
/// specific prior written permission.
///
/// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
/// AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
/// IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
/// ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE

```

```

    /// LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
    /// CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
    /// SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
    /// INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
    /// CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
    /// ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
    /// THE POSSIBILITY OF SUCH DAMAGE.
    ///
    ///

```

Definition in file [Mat.h](#).

10.84 Mat.h

```

00001
00055 #pragma once
00056
00057 #include <cmath>
00058 #include <vector>
00059
00060 #include <Eigen/Core>
00061
00062 // The following is from modules.h, which includes this file, Mat.h. We need
00063 // the declaration here because we use it below in the deriv method. If we
00064 // don't declare it, clang and gcc > 4.4.7-4 will complain about not being able
00065 // to resolve the function.
00066 double dot(double x, double y, double z);
00067
00069 // Nota Bene! Caveat Emptor! Ad Nauseam!
00070 // The geometric flow code was originally done in Fortran. These templates are
00071 // part of a port to C++ that was done quite literally. The net result is that
00072 // in _these_ templates, counting starts at '1' NOT at '0', as the creator,
00073 // EWD intended (https://www.cs.utexas.edu/users/EWD/ewd08xx/EWD831.PDF).
00075
00076 template <typename T> struct Stencil;
00077
00078 template <typename T = double> class Mat;
00079
00080 template <typename T>
00081 class Mat {
00082 private:
00083     friend struct Stencil<T>;
00084
00085 private:
00086     size_t _nx, _ny, _nz;
00087 public:
00088     size_t nx() const { return _nx; }
00089     size_t ny() const { return _ny; }
00090     size_t nz() const { return _nz; }
00091
00092 private:
00093     double _hx, _hy, _hz;
00094
00095 public:
00096     double hx() const { return _hx; }
00097     double hy() const { return _hy; }
00098     double hz() const { return _hz; }
00099
00100 private:
00101     Eigen::Matrix<T, Eigen::Dynamic, 1> vec;
00102
00103     Mat();
00104
00105 public:
00106     // hx, hy, and hz are the grid spacing, in Angstroms (presumably)
00107     Mat(size_t nx, size_t ny, size_t nz=1, T a=0):
00108         _nx(nx), _ny(ny), _nz(nz), _hx(1), _hy(1), _hz(1), vec(nx*ny*nz)
00109     { vec.fill(a); }
00110     Mat(size_t nx, size_t ny, size_t nz, double hx, double hy, double hz, T a=0):
00111         _nx(nx), _ny(ny), _nz(nz), _hx(hx), _hy(hy), _hz(hz), vec(nx*ny*nz)
00112     { vec.fill(a); }
00113
00114     //Mat(const Mat&) default, vec is copyable
00115

```



```

00116 ~Mat() {};
00117
00118 friend void swap(Mat<T> &a, Mat<T> &b) throw() {
00119     using std::swap;
00120     a.vec.swap(b.vec);
00121     swap(a._nx, b._nx);
00122     swap(a._ny, b._ny);
00123     swap(a._nz, b._nz);
00124     swap(a._hx, b._hx);
00125     swap(a._hy, b._hy);
00126     swap(a._hz, b._hz);
00127 }
00128
00129 Eigen::Matrix<T,Eigen::Dynamic,1>& baseInterface()
00130 {
00131     return vec;
00132 }
00133
00134 const Eigen::Matrix<T,Eigen::Dynamic,1>& baseInterface() const
00135 {
00136     return vec;
00137 }
00138
00139 bool equalSize(const Mat<T>& rhs) const
00140 {
00141     return (this->_nx == rhs._nx) && (this->_ny == rhs._ny) &&
00142            (this->_nz == rhs._nz);
00143 }
00144
00145 bool equalSpacing(const Mat<T>& rhs) const
00146 {
00147     return (this->_hx == rhs._hx) && (this->_hy == rhs._hy) &&
00148            (this->_hz == rhs._hz);
00149 }
00150
00151 bool operator==(const Mat<T>& rhs) const
00152 {
00153     return (data() == rhs.data()) && equalSize(rhs) && equalSpacing(rhs);
00154 }
00155
00156 T& operator()(size_t x, size_t y, size_t z=1)
00157 {
00158     return vec[index(x,y,z)];
00159 }
00160
00161 T operator()(size_t x, size_t y, size_t z=1) const
00162 {
00163     return vec[index(x,y,z)];
00164 }
00165
00166 T& operator[](size_t i)
00167 {
00168     return vec(i);
00169 }
00170
00171 T operator[](size_t i) const
00172 {
00173     return vec(i);
00174 }
00175
00176 size_t index(size_t x, size_t y, size_t z) const
00177 {
00178     return x-1 + _nx*((y-1) + _ny*(z-1));
00179 }
00180
00181 T* data()
00182 {
00183     return vec.data();
00184 }
00185
00186 const T* data() const
00187 {
00188     return vec.data();
00189 }
00190
00191 size_t size()
00192 {
00193     return vec.size();
00194 }
00195
00196 //should be iterator...

```

```

00197 T* end()
00198 {
00199     return vec.data() + size();
00200 }
00201
00202 Mat<T>& operator=(T a)
00203 {
00204     vec.fill(a); return *this;
00205 }
00206
00207 Mat<T>& operator=(Mat a)
00208 {
00209     swap(*this, a);
00210     return *this;
00211 }
00212
00213 Stencil<T> stencilBegin()
00214 {
00215     return Stencil<T>(*this, 2,2,2);
00216 }
00217
00218 Stencil<T> stencilEnd()
00219 {
00220     return Stencil<T>(*this, _nx-1,_ny-1,_nz-1);
00221 }
00222 };
00223
00224 template<typename T>
00225 struct Stencil : public std::iterator<std::forward_iterator_tag, T>
00226 {
00227     //this should be a const iterator :)
00228     Mat<T>& _mat;
00229     const T halfhx,h2x,qrth2x;
00230     const T halfhy,h2y,qrth2y;
00231     const T halfhz,h2z,qrth2z;
00232     size_t i;
00233     const size_t yStep, zStep;
00234
00235     T *c;
00236
00237     Stencil(Mat<T>& mat, size_t x, size_t y, size_t z) : _mat(mat),
00238         halfhx(0.5/mat.hx()), // W1(1)
00239         h2x(1.0/(mat.hx()*mat.hx())), // W2(1)
00240         qrth2x(0.25/(mat.hx()*mat.hx())), // WXY(1,1)
00241         halfhy(0.5/mat.hy()), // ...
00242         h2y(1.0/(mat.hy()*mat.hy())), // ...
00243         qrth2y(0.25/(mat.hy()*mat.hy())), // ...
00244         halfhz(0.5/mat.hz()), // ...
00245         h2z(1.0/(mat.hz()*mat.hz())), // ...
00246         qrth2z(0.25/(mat.hz()*mat.hz())), // ...
00247         i(mat.index(x,y,z)), // Current index
00248         yStep(mat.nx()), // Offset for each y increment
00249         zStep(mat.nx()*mat.ny()), // Offset for each z increment
00250         c(mat.data() + i) // Current value
00251     {}
00252
00253     bool operator==(const Stencil<T>& rhs) const
00254     {
00255         return (this->_mat == rhs._mat) && (this->c == rhs.c);
00256     }
00257
00258     bool operator!=(const Stencil<T>& rhs) const
00259     {
00260         return !(*this == rhs);
00261     }
00262
00263     T& operator*()
00264     {
00265         return *c;
00266     }
00267
00268     const T* operator->() const //???
00269     {
00270         return c;
00271     }
00272
00273     Stencil& operator++()
00274     {
00275         next();
00276         return *this;
00277     }

```

```

00278
00279 Stencil operator++ ( int )
00280 {
00281     Stencil<T> clone( *this );
00282     operator++;
00283     return clone;
00284 }
00285
00286 void next()
00287 {
00288     ptrStep(1);
00289     if ((i+1) % yStep == 0) {
00290         if ((i+1+yStep) % zStep == 0) {
00291             // Skip the last row in this z, and the first in the next z.
00292             // Plus we need to skip the last and first columns, as below.
00293             ptrStep(2*yStep + 2);
00294         } else {
00295             // Skip the last column in the current row, and the first
00296             // column in the next row.
00297             ptrStep(2);
00298         }
00299     }
00300 }
00301
00302 void ptrStep(size_t numSteps)
00303 {
00304     i += numSteps;
00305     c += numSteps;
00306 }
00307
00308 const T* xm() const { return c - 1; } // PHI (IX-1,IY,IZ)
00309 const T* xp() const { return c + 1; } // PHI (IX+1,IY,IZ)
00310 const T* ym() const { return c - yStep; } // PHI (IX,IY-1,IZ)
00311 const T* yp() const { return c + yStep; } // PHI (IX,IY+1,IZ)
00312 const T* zm() const { return c - zStep; } // PHI (IX,IY,IZ-1)
00313 const T* zp() const { return c + zStep; } // PHI (IX,IY,IZ+1)
00314
00315 const T* xym() const { return xm() - yStep; } // PHI (IX-1,IY-1,IZ)
00316 const T* xyp() const { return xp() + yStep; } // PHI (IX+1,IY+1,IZ)
00317 const T* xzm() const { return xm() - zStep; } // PHI (IX-1,IY,IZ-1)
00318 const T* xzp() const { return xp() + zStep; } // PHI (IX+1,IY,IZ+1)
00319 const T* yzm() const { return zm() - yStep; } // PHI (IX,IY-1,IZ-1)
00320 const T* yzp() const { return zp() + yStep; } // PHI (IX,IY+1,IZ+1)
00321
00322 const T* xm_yp() const { return xm() + yStep; } // PHI (IX-1,IY+1,IZ)
00323 const T* xp_ym() const { return xp() - yStep; } // PHI (IX+1,IY-1,IZ)
00324 const T* xm_zp() const { return xm() + zStep; } // PHI (IX-1,IY,IZ+1)
00325 const T* xp_zm() const { return xp() - zStep; } // PHI (IX+1,IY,IZ-1)
00326 const T* ym_zp() const { return ym() + zStep; } // PHI (IX,IY-1,IZ+1)
00327 const T* yp_zm() const { return yp() - zStep; } // PHI (IX,IY+1,IZ-1)
00328
00329 T dx() const { return halfhx * (*xp() - *xm()); } // PHIX
00330 T dy() const { return halfhy * (*yp() - *ym()); } // PHIY
00331 T dz() const { return halfhz * (*zp() - *zm()); } // PHIZ
00332
00333 T dxx() const { return h2x * (*xp() - 2.0*(*c) + *xm()); } // PHIXX
00334 T dyy() const { return h2y * (*yp() - 2.0*(*c) + *ym()); } // PHIYY
00335 T dzz() const { return h2z * (*zp() - 2.0*(*c) + *zm()); } // PHIZZ
00336
00337 T dxy() const { return qrth2x * (*xyp() + *xym() - *xm_yp() - *xp_ym()); } // PHIXY
00338 T dxz() const { return qrth2y * (*xzp() + *xzm() - *xm_zp() - *xp_zm()); } // PHIXZ
00339 T dyz() const { return qrth2z * (*yzp() + *yzm() - *ym_zp() - *yp_zm()); } // PHIYZ
00340
00341 T deriv(T tx) const {
00342     // From the following Fortran...
00343     // DPHI=(1.0D0+PHIX**2+PHIY**2)*PHIZZ+(1.0D0+PHIX**2+PHIZ**2)*PHIYY+ &
00344     // (1.0D0+PHIY**2+PHIZ**2)*PHIXX
00345     // ...
00346     // DPHI=DPHI-2.D0*(PHIXY*PHIX*PHIY+PHIXZ*PHIX*PHIZ+PHIYZ*PHIY*PHIZ)
00347     double dphi =
00348         ( 1.0 + dx()*dx() + dy()*dy() ) * dzz()
00349         + ( 1.0 + dx()*dx() + dz()*dz() ) * dyy()
00350         + ( 1.0 + dy()*dy() + dz()*dz() ) * dxx();
00351
00352     dphi -= 2.0 * ( dx()*dy()*dxy() + dx()*dz()*dxz() + dy()*dz()*dyz() );
00353
00354     // GRAM=(1.D0+PHIX**2+PHIY**2+PHIZ**2)
00355     double gram = 1.0 + dot(dx(),dy(),dz());
00356
00357     // DPHI=DPHI/GRAM+SQRT(GRAM)*PHITOTX(IX,IY,IZ)
00358     double d = dphi/gram + sqrt(gram)*tx;

```

```

00359     return d;
00360 }
00361
00362 };

```

10.85 src/geoflow/modules.h File Reference

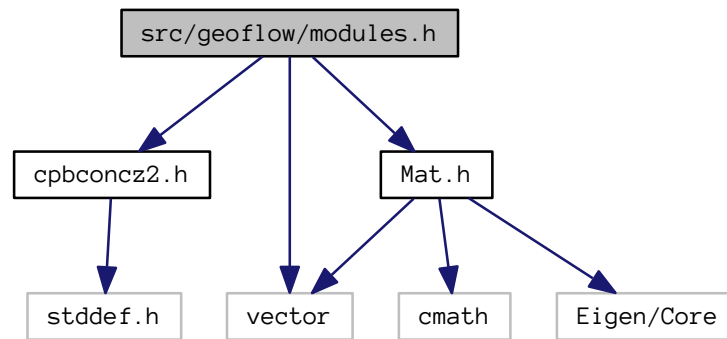
FORTTRAN globals and function headers.

```

#include <vector>
#include "cpbconcz2.h"
#include "Mat.h"

```

Include dependency graph for modules.h:



Data Structures

- struct [Comdata](#)
- struct [LJ](#)

Functions

- void **domainini** (double xyzr[MAXATOMS][XYZRWIDTH], const size_t natm, const double extvalue)
- void **chargedist** (double xyzr[MAXATOMS][XYZRWIDTH], double *chratm, [Mat](#)<> &charget, [Mat](#)<> &corlocqt, [Mat](#)< size_t > &locqt, size_t iatm)
- void **yhsurface** (double xyzr[MAXATOMS][XYZRWIDTH], double *ljepsilon, size_t natm, double tott, double deltatt, [Mat](#)<> &phix, [Mat](#)<> &surfu, int i, double &area, double &vol, double &attint, double alpha, int iadi, int igfin)
- void **seteqb** ([Mat](#)<> &bg, double xyzr[MAXATOMS][XYZRWIDTH], double *pqr, [Mat](#)<> &charget, [Mat](#)<> &corlocqt, double epsilonp)
- void **pbsolver** ([Mat](#)<> &eps, [Mat](#)<> &phi, [Mat](#)<> &bg, double tol, int iter)
- double **xvalue** (size_t i)
- double **yvalue** (size_t i)
- double **zvalue** (size_t i)

- `size_t inverx` (double x)
- `size_t invery` (double y)
- `size_t inverz` (double z)

Variables

- `Comdata comdata`
- `LJ lj`

10.85.1 Detailed Description

FORTTRAN globals and function headers.

Author

Andrew Stevens, Kyle Monson

Version

\$Id\$

Attention

```

///
/// APBS -- Adaptive Poisson-Boltzmann Solver
///
/// Nathan A. Baker (nathan.baker@pnnl.gov)
/// Pacific Northwest National Laboratory
///
/// Additional contributing authors listed in the code documentation.
///
/// Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
/// Pacific Northwest National Laboratory, operated by Battelle Memorial
/// Institute, Pacific Northwest Division for the U.S. Department of Energy.
///
/// Portions Copyright (c) 2002-2010, Washington University in St. Louis.
/// Portions Copyright (c) 2002-2010, Nathan A. Baker.
/// Portions Copyright (c) 1999-2002, The Regents of the University of
/// California.
/// Portions Copyright (c) 1995, Michael Holst.
/// All rights reserved.
///
/// Redistribution and use in source and binary forms, with or without
/// modification, are permitted provided that the following conditions are met:
///
/// Redistributions of source code must retain the above copyright notice, this
/// list of conditions and the following disclaimer.
///
/// Redistributions in binary form must reproduce the above copyright notice,
/// this list of conditions and the following disclaimer in the documentation
/// and/or other materials provided with the distribution.
///
/// Neither the name of the developer nor the names of its contributors may be
/// used to endorse or promote products derived from this software without
/// specific prior written permission.
///
/// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
/// AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
/// IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
/// ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE

```

```

    /// LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
    /// CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
    /// SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
    /// INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
    /// CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
    /// ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
    /// THE POSSIBILITY OF SUCH DAMAGE.
    ///
    ///

```

Definition in file [modules.h](#).

10.86 modules.h

```

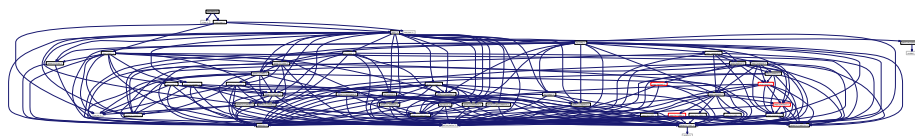
00001
00055 # pragma once
00056
00057 #include <vector>
00058
00059 #include "cpbconcz2.h"
00060 #include "Mat.h"
00061
00062 struct Comdata
00063 {
00064     char fname[100];
00065     size_t nx, ny, nz;
00066     double xleft, xright,
00067         yleft, yright,
00068         zleft, zright,
00069         deltax, deltax, deltaz,
00070         dcel, pi;
00071     std::vector<double> xc, yc, zc;
00072 };
00073 extern Comdata comdata;
00074
00075 struct LJ
00076 {
00077     double tauval, prob, vdwdispersion, sigmas, roro, conms, density, epsilonw;
00078     int ffmmodel;
00079     static const int ioasetar = 1, ioasetaa = 1, iwca = 1;
00080 };
00081 extern LJ lj;
00082
00083 void domainini(double xyzr[MAXATOMS][XYZRWIDTH], const size_t natm,
00084     const double extvalue);
00085
00086 void chargedist(double xyzr[MAXATOMS][XYZRWIDTH], double* chrattm,
00087     Mat<>& charget, Mat<>& corlocqt, Mat<size_t>& locqt, size_t iatm);
00088
00089 void yhsurface(double xyzr[MAXATOMS][XYZRWIDTH], double* ljepsilon, size_t natm,
00090     double tott, double deltat, Mat<>& phix, Mat<>& surfu, int i,
00091     double& area, double& vol, double& attint, double alpha, int iadi,
00092     int igfin);
00093
00094 void seteqb(Mat<>& bg, double xyzr[MAXATOMS][XYZRWIDTH], double* pqr,
00095     Mat<>& charget, Mat<>& corlocqt, double epsilonps);
00096
00097 void pbsolver(Mat<>& eps, Mat<>& phi, Mat<>& bg, double tol, int iter);
00098
00099 double xvalue(size_t i);
00100 double yvalue(size_t i);
00101 double zvalue(size_t i);
00102
00103 size_t inverx(double x);
00104 size_t invery(double y);
00105 size_t inverz(double z);

```

10.87 src/main.c File Reference

APBS "front end" program using formatted input files.

```
#include <time.h>
#include "routines.h"
Include dependency graph for main.c:
```



Functions

- int [main](#) (int argc, char **argv)
The main APBS function.

10.87.1 Detailed Description

APBS "front end" program using formatted input files.

Author

Nathan Baker This driver program represents a mish-mash of instructions for calculating electrostatic potentials, as well as free energies of binding and solvation. It is invoked as:

```
apbs apbs.in
```

where apbs.in is a formatted input file (see documentation and examples).

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
```

```

*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [main.c](#).

10.88 main.c

```

00001
00068 #include <time.h>
00069
00070 #include "routines.h"
00071
00072 VEMBED(rcsid="$Id$")
00073
00074
00080 int main(
00081     int argc,
00082     char **argv
00083 )
00084 {
00085     // PCE: Adding below variables temporarily
00086     clock_t ts, te;
00087     // End PCE
00088
00089     Nosh *nosh = VNULL;
00090
00091     MGparm *mgparm = VNULL;
00092     FEMparm *feparm = VNULL;
00093 #ifdef ENABLE_BEM
00094     BEMparm *bemparm = VNULL;
00095 #endif
00096     GEOFLOWparm *geoflowparm = VNULL;
00097     PBEparm *pbeparm = VNULL;
00098     APOLparm *apolparm = VNULL;
00099     Vparam *param = VNULL;
00100
00101     Vmem *mem = VNULL;
00102     Vcom *com = VNULL;
00103     Vio *sock = VNULL;
00104 #ifdef HAVE_MC_H
00105     Vfetc *fetc[NOSH_MAXCALC];
00106     Gem *gm[NOSH_MAXMOL];
00107     int isolate;
00108 #else
00109     void *fetc[NOSH_MAXCALC];
00110     void *gm[NOSH_MAXMOL];
00111 #endif
00112     Vpmg *pmg[NOSH_MAXCALC];
00113     Vpmgp *pmgp[NOSH_MAXCALC];
00114     Vpbe *pbe[NOSH_MAXCALC];
00115     Valist *alist[NOSH_MAXMOL];
00116     Vgrid *die1XMap[NOSH_MAXMOL],

```



```

00117         *dielYMap[NOSH_MAXMOL],
00118         *dielZMap[NOSH_MAXMOL],
00119         *kappaMap[NOSH_MAXMOL],
00120         *potMap[NOSH_MAXMOL],
00121         *chargeMap[NOSH_MAXMOL];
00122     char *input_path = VNULL,
00123         *output_path = VNULL;
00124     int i,
00125         rank,
00126         size,
00127         k;
00128     size_t bytesTotal,
00129         highWater;
00130     Voutput_Format outputformat;
00131
00132     int rc = 0;
00133
00134     /* The energy double arrays below store energies from various calculations. */
00135     double qfEnergy[NOSH_MAXCALC],
00136         qmEnergy[NOSH_MAXCALC];
00137     double dielEnergy[NOSH_MAXCALC],
00138         totEnergy[NOSH_MAXCALC];
00139     double *atomEnergy[NOSH_MAXCALC];
00140     AtomForce *atomForce[NOSH_MAXCALC]; /* Stores forces from various calculations. */
00141     int nenergy[NOSH_MAXCALC], /* Stores either a flag (0,1) displaying whether
00142                                * energies were calculated, or, if PCE_COMPS
00143                                * was used, the number of atom energies stored
00144                                * for the given calculation. */
00145         nforce[NOSH_MAXCALC]; /* Stores an integer which either says no
00146                                * calculation was performed (0) or gives the
00147                                * number of entries in the force array for each
00148                                * calculation. */
00149
00150     /* The real partition centers */
00151     double realCenter[3];
00152
00153     /* Instructions: */
00154     char header[] = {"\n\n\
00155 -----\n\
00156 APBS -- Adaptive Poisson-Boltzmann Solver\n\
00157 Version " PACKAGE_STRING "\n\
00158 \n\
00159 Nathan A. Baker (nathan.baker@pnnl.gov)\n\
00160 Pacific Northwest National Laboratory\n\
00161 \n\
00162 Additional contributing authors listed in the code documentation.\n\
00163 \n\
00164 Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the Pacific\n\
00165 Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific\n\
00166 Northwest Division for the U.S. Department of Energy.\n\
00167 \n\
00168 Portions Copyright (c) 2002-2010, Washington University in St. Louis.\n\
00169 Portions Copyright (c) 2002-2010, Nathan A. Baker.\n\
00170 Portions Copyright (c) 1999-2002, The Regents of the University of California.\n\
00171 Portions Copyright (c) 1995, Michael Holst.\n\
00172 All rights reserved.\n\
00173 \n\
00174 Redistribution and use in source and binary forms, with or without\n\
00175 modification, are permitted provided that the following conditions are met:\n\
00176 \n\
00177 * Redistributions of source code must retain the above copyright notice, this\n\
00178 list of conditions and the following disclaimer.\n\
00179 \n\
00180 * Redistributions in binary form must reproduce the above copyright notice,\n\
00181 this list of conditions and the following disclaimer in the documentation\n\
00182 and/or other materials provided with the distribution.\n\
00183 \n\
00184 * Neither the name of the developer nor the names of its contributors may be\n\
00185 used to endorse or promote products derived from this software without\n\
00186 specific prior written permission.\n\
00187 \n\
00188 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND\n\
00189 ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED\n\
00190 WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE\n\
00191 DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR\n\
00192 ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES\n\
00193 (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;\n\
00194 LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND\n\
00195 ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT\n\
00196 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS\n\
00197 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.\n\

```

```

00198 -----\n\
00199 APBS uses FETK (the Finite Element ToolKit) to solve the\n\
00200 Poisson-Boltzmann equation numerically. FETK is a portable collection\n\
00201 of finite element modeling class libraries developed by the Michael Holst\n\
00202 research group and written in an object-oriented form of C. FETk is\n\
00203 designed to solve general coupled systems of nonlinear partial differential\n\
00204 equations using adaptive finite element methods, inexact Newton methods,\n\
00205 and algebraic multilevel methods. More information about FETk may be found\n\
00206 at <http://www.FETk.ORG>.\n\
00207 -----\n\
00208 APBS also uses Aqua to solve the Poisson-Boltzmann equation numerically. \n\
00209 Aqua is a modified form of the Holst group PMG library <http://www.FETk.ORG>\n\
00210 which has been modified by Patrice Koehl\n\
00211 <http://koehllab.genomecenter.ucdavis.edu/> for improved efficiency and\n\
00212 memory usage when solving the Poisson-Boltzmann equation.\n\
00213 -----\n\
00214 Please cite your use of APBS as:\n\
00215 Baker NA, Sept D, Joseph S, Holst MJ, McCammon JA. Electrostatics of\n\
00216 nanosystems: application to microtubules and the ribosome. Proc.\n\
00217 Natl. Acad. Sci. USA 98, 10037-10041 2001.\n\
00218 \n\n";
00219 char *usage =
00220 {"\n\n\
00221 -----\n\
00222 This driver program calculates electrostatic potentials, energies,\n\
00223 and forces using both multigrid and finite element methods.\n\
00224 It is invoked as:\n\
00225 apbs [options] apbs.in\n\
00226 where apbs.in is a formatted input file and [options] are:\n\
00227 --output-file=<name> Enables output logging to the path\n\
00228 listed in <name>. Uses flat-file\n\
00229 format is --output-format is not used.\n\
00230 --output-format=<type> Specifies format for logging. Options\n\
00231 for type are either \"xml\" or \"flat\".\n\
00232 --help Display this help information.\n\
00233 --version Display the current APBS version.\n\
00234 -----\n\n\"};
00235
00236 /* ***** CHECK PARALLEL STATUS ***** */
00237 VASSERT(Vcom_init(&argc, &argv));
00238 com = Vcom_ctor(1);
00239 rank = Vcom_rank(com);
00240 size = Vcom_size(com);
00241 startVio();
00242 Vnm_setIoTag(rank, size);
00243 Vnm_tprint( 0, "Hello world from PE %d\n", rank);
00244
00245 /* A bit of array/pointer initialization */
00246 mem = Vmem_ctor("MAIN");
00247 for (i=0; i<NOSH_MAXCALC; i++) {
00248     pmg[i] = VNULL;
00249     pmgp[i] = VNULL;
00250     fetk[i] = VNULL;
00251     pbe[i] = VNULL;
00252     qfEnergy[i] = 0;
00253     qmEnergy[i] = 0;
00254     dielEnergy[i] = 0;
00255     totEnergy[i] = 0;
00256     atomForce[i] = VNULL;
00257     nenergy[i] = 0;
00258     nforce[i] = 0;
00259 }
00260 for (i=0; i<NOSH_MAXMOL; i++) {
00261     alist[i] = VNULL;
00262     dielXMap[i] = VNULL;
00263     dielYMap[i] = VNULL;
00264     dielZMap[i] = VNULL;
00265     kappaMap[i] = VNULL;
00266     potMap[i] = VNULL;
00267     chargeMap[i] = VNULL;
00268 }
00269
00270 /* ***** CHECK INVOCATION AND OPTIONS ***** */
00271 Vnm_tstart(APBS_TIMER_WALL_CLOCK, "APBS WALL CLOCK");
00272 Vnm_tprint( 1, "%s", header);
00273
00274 #ifdef APBS_FAST
00275 Vnm_tprint(, 2"WARNING: APBS was compiled with the --enable-fast option.\n"
00276 "WARNING: This mode is experimental and subject to change in future releases.\n"
00277 "WARNING: The fast mode enables: Gauss-Seidel Smoothing and \n"
00278 "WARNING: Conjugate Gradient Multigrid methods.\n\n");

```

```

00279 #endif
00280
00281     Vnm_tprint( 1, "This executable compiled on %s at %s\n\n", __DATE__, __TIME__);
00282
00283 #if defined(WITH_TINKER)
00284     Vnm_tprint( 2, "This executable was compiled with TINKER support and is not intended for stand-alone
execution.\n");
00285     Vnm_tprint( 2, "Please compile another version without TINKER support.\n");
00286     exit(2);
00287 #endif
00288
00289     /* Process program arguments */
00290     i=0;
00291     outputformat = OUTPUT_NULL;
00292     while (i<argc){
00293         if (strncmp(argv[i], "--", 2) == 0) {
00294
00295             /* Long Options */
00296             if (Vstring_strcasecmp("--version", argv[i]) == 0){
00297                 Vnm_tprint(2, "%s\n", PACKAGE_STRING);
00298                 VJMPERR1(0);
00299             } else if (Vstring_strcasecmp("--help", argv[i]) == 0){
00300                 Vnm_tprint(2, "%s\n", usage);
00301                 VJMPERR1(0);
00302             } else if (strncmp(argv[i], "--output-format", 15) == 0) {
00303                 if (strstr(argv[i], "xml") != NULL) {
00304                     Vnm_tprint(2, "XML output format is now deprecated, please use --output-format=flat
instead!\n\n");
00305                     VJMPERR1(0);
00306                 }
00307                 else if (strstr(argv[i], "flat") != NULL) {
00308                     outputformat = OUTPUT_FLAT;
00309                 } else {
00310                     Vnm_tprint(2, "Invalid output-format type!\n");
00311                     VJMPERR1(0);
00312                 }
00313             } else if (strncmp(argv[i], "--output-file=", 14) == 0){
00314                 output_path = strstr(argv[i], "=");
00315                 ++output_path;
00316                 if (outputformat == OUTPUT_NULL) outputformat =
OUTPUT_FLAT;
00317             } else {
00318                 Vnm_tprint(2, "UNRECOGNIZED COMMAND LINE OPTION %s!\n", argv[i]);
00319                 Vnm_tprint(2, "%s\n", usage);
00320                 VJMPERR1(0);
00321             }
00322         } else {
00323
00324             /* Set the path to the input file */
00325             if ((input_path == VNULL) && (i != 0))
00326                 input_path = argv[i];
00327             else if (i != 0) {
00328                 Vnm_tprint(2, "ERROR -- CALLED WITH TOO MANY ARGUMENTS!\n", \
00329                     argc);
00330                 Vnm_tprint(2, "%s\n", usage);
00331                 VJMPERR1(0);
00332             }
00333         }
00334         i++;
00335     }
00336
00337     /* If we set an output format but no path, error. */
00338     if ((outputformat != 0) && (output_path == NULL)) {
00339         Vnm_tprint(2, "The --output-path variable must be set when using --output-format!\n");
00340         VJMPERR1(0);
00341     }
00342
00343     /* If we failed to specify an input file, error. */
00344     if (input_path == NULL) {
00345         Vnm_tprint(2, "ERROR -- APBS input file not specified!\n", argc);
00346         Vnm_tprint(2, "%s\n", usage);
00347         VJMPERR1(0);
00348     }
00349
00350     /* Append rank info if a parallel run */
00351     if ((size > 1) && (output_path != NULL))
00352         printf(output_path, "%s%d", output_path, rank);
00353
00354     /* ***** PARSE INPUT FILE ***** */
00355     nosh = Nosh_ctor(rank, size);
00356     Vnm_tprint( 1, "Parsing input file %s...\n", input_path);

```

```

00357     sock = Vio_ctor("FILE", "ASC", VNULL, input_path, "r");
00358     if (sock == VNULL) {
00359         Vnm_tprint(2, "Error while opening input file %s!\n", input_path);
00360         VJMPERR1(0);
00361     }
00362     if (!Nosh_parseInput(nosh, sock)) {
00363         Vnm_tprint( 2, "Error while parsing input file.\n");
00364         VJMPERR1(0);
00365     }
00366     else
00367         Vnm_tprint( 1, "Parsed input file.\n");
00368     Vio_dtor(&sock);
00369
00370     /* ***** LOAD PARAMETERS AND MOLECULES ***** */
00371     param = loadParameter(nosh);
00372     if (loadMolecules(nosh, param, alist) != 1) {
00373         Vnm_tprint(2, "Error reading molecules!\n");
00374         VJMPERR1(0);
00375     }
00376
00377     /* ***** SETUP CALCULATIONS ***** */
00378     if (Nosh_setupElecCalc(nosh, alist) != 1) {
00379         Vnm_tprint(2, "Error setting up ELEC calculations\n");
00380         VJMPERR1(0);
00381     }
00382
00383     if ((rc = Nosh_setupApolCalc(nosh, alist)) == ACD_ERROR) {
00384         Vnm_tprint(2, "Error setting up APOL calculations\n");
00385         VJMPERR1(0);
00386     }
00387
00388     /* ***** CHECK APOL***** */
00389     /* if((nosh->gotparm == 0) && (rc == ACD_YES)){
00390         Vnm_print(1, "\nError you must provide a parameter file if you\n" \
00391             "          are performing an APOLAR calculation\n");
00392         VJMPERR1(0);
00393     } */
00394
00395     #if defined(DEBUG_MAC_OSX_OCL)
00396     #include "mach_chud.h"
00397     #include <stdint.h>
00398     uint64_t mbeg;
00399     machm_(&mbeg);
00400
00401     if (clFinish != NULL)
00402     {
00403         int ret = initOpenCL();
00404         printf("OpenCL runtime present - initialized = %i\n", ret);
00405     }
00406     else
00407     {
00408         setkOpenCLAvailable_(0);
00409         printf("OpenCL is not present!\n");
00410     }
00411     #endif
00412
00413     #if defined(DEBUG_MAC_OSX_STANDARD)
00414     #include "mach_chud.h"
00415     #include <stdint.h>
00416     uint64_t mbeg;
00417     machm_(&mbeg);
00418     #endif
00419
00420     /* ***** LOAD MAPS ***** */
00421     if (loadDielMaps(nosh, dielXMap, dielYMap, dielZMap) != 1) {
00422         Vnm_tprint(2, "Error reading dielectric maps!\n");
00423         VJMPERR1(0);
00424     }
00425     if (loadKappaMaps(nosh, kappaMap) != 1) {
00426         Vnm_tprint(2, "Error reading kappa maps!\n");
00427         VJMPERR1(0);
00428     }
00429     if (loadPotMaps(nosh, potMap) != 1) {
00430         Vnm_tprint(2, "Error reading potential maps!\n");
00431         VJMPERR1(0);
00432     }
00433     if (loadChargeMaps(nosh, chargeMap) != 1) {
00434         Vnm_tprint(2, "Error reading charge maps!\n");
00435         VJMPERR1(0);
00436     }
00437

```

```

00438  /* ***** DO THE CALCULATIONS ***** */
00439  Vnm_tprint( 1, "Preparing to run %d PBE calculations.\n",
00440             nosh->ncalc);
00441  for (i=0; i<nosh->ncalc; i++) {
00442      Vnm_tprint( 1, "-----\n");
00443
00444      switch (nosh->calc[i]->calctype) {
00445          /* Multigrid */
00446          case NCT_MG:
00447              /* What is this? This seems like a very awkward way to find
00448               the right ELEC statement... */
00449              for (k=0; k<nosh->nelec; k++) {
00450                  if (nosh->elec2calc[k] >= i) {
00451                      break;
00452                  }
00453              }
00454              if (Vstring_strcasecmp(nosh->elecname[k], "") == 0) {
00455                  Vnm_tprint( 1, "CALCULATION #d: MULTIGRID\n", i+1);
00456              } else {
00457                  Vnm_tprint( 1, "CALCULATION #d (%s): MULTIGRID\n",
00458                             i+1, nosh->elecname[k]);
00459              }
00460              /* Useful local variables */
00461              mgparm = nosh->calc[i]->mgparm;
00462              pbeparm = nosh->calc[i]->pbeparm;
00463
00464              /* Set up problem */
00465              Vnm_tprint( 1, "  Setting up problem...\n");
00466
00467              if (!initMG(i, nosh, mgparm, pbeparm, realCenter, pbe,
00468                         alist, dielXMap, dielYMap, dielZMap, kappaMap,
00469                         chargeMap, pmgp, pmg, potMap)) {
00470                  Vnm_tprint( 2, "Error setting up MG calculation!\n");
00471                  VJMPERR1(0);
00472              }
00473
00474              /* Print problem parameters */
00475              printMGPARAM(mgparm, realCenter);
00476              printPBEPARM(pbeparm);
00477
00478              /* Solve PDE */
00479              if (solveMG(nosh, pmg[i], mgparm->type) != 1) {
00480                  Vnm_tprint(2, "Error solving PDE!\n");
00481                  VJMPERR1(0);
00482              }
00483
00484              /* Set partition information for observables and I/O */
00485              if (setPartMG(nosh, mgparm, pmg[i]) != 1) {
00486                  Vnm_tprint(2, "Error setting partition info!\n");
00487                  VJMPERR1(0);
00488              }
00489
00490              /* Write out energies */
00491              energyMG(nosh, i, pmg[i],
00492                     &(nenergy[i]), &(totEnergy[i]), &(qfEnergy[i]),
00493                     &(qmEnergy[i]), &(dielEnergy[i]));
00494
00495              /* Write out forces */
00496              forceMG(mem, nosh, pbeparm, mgparm, pmg[i], &(nforce[i]),
00497                     &(atomForce[i]), alist);
00498
00499              /* Write out data folks might want */
00500              writedataMG(rank, nosh, pbeparm, pmg[i]);
00501
00502              /* Write matrix */
00503              writematMG(rank, nosh, pbeparm, pmg[i]);
00504
00505              /* If needed, cache atom energies */
00506              nenergy[i] = 0;
00507              if ((pbeparm->calcenergy == PCE_COMPS) && (outputformat !=
00508                  OUTPUT_NULL)){
00509                  storeAtomEnergy(pmg[i], i, &(atomEnergy[i]), &(nenergy[i]));
00510              }
00511              fflush(stdout);
00512              fflush(stderr);
00513
00514              break;
00515
00516              /* ***** Do FEM calculation ***** */
00517          case NCT_FEM:

```

```

00518 #ifdef HAVE_MC_H
00519     for (k=0; k<nosh->nelec; k++) {
00520         if (nosh->elec2calc[k] >= i) break;
00521     }
00522     if (Vstring_strcasecmp(nosh->elecname[i+1], "") == 0) {
00523         Vnm_tprint( 1, "CALCULATION #d: FINITE ELEMENT\n", i+1);
00524     } else {
00525         Vnm_tprint( 1, "CALCULATION #d (%s): FINITE ELEMENT\n", i+1, nosh->
elecname[k+1]);
00526     }
00527
00528     /* Useful local variables */
00529     feparm = nosh->calc[i]->feparm;
00530     pbeparm = nosh->calc[i]->pbeparm;
00531
00532     /* Warn the user about some things */
00533     Vnm_tprint(2, "##### WARNING #####\n");
00534     Vnm_tprint(2, "## FE support is currently very experimental! ##\n");
00535     Vnm_tprint(2, "##### WARNING #####\n");
00536
00537     /* Set up problem */
00538     Vnm_tprint( 1, " Setting up problem...\n");
00539     /* Attempt to initialize and do an initial refinement of the mesh data. The mesh data
00540      * will be stored in the Vfetk object fetk, which contains the appropriate geometry
00541      * manager (Gem) object and Vcsm object describing the mesh structure. The mesh will
00542      * either be loaded from an external source or generated from scratch. */
00543     if (initFE(i, nosh, feparm, pbeparm, pbe, alist, fetk) !=
VRC_SUCCESS) {
00544         Vnm_tprint( 2, "Error setting up FE calculation!\n");
00545         VJMPErr(0);
00546     }
00547
00548     /* Print problem parameters */
00549     printFEPARM(i, nosh, feparm, fetk);
00550     printPBEPARM(pbeparm);
00551
00552     /* Refine mesh - this continues to run the AM_markRefine procedure already run
00553      * in initFE() to arrive at some initial refinement, but does checks of the
00554      * simplices so that it refines until the error or size tolerances are reached.
00555      * Once this is done, we have a mesh that has been refined to the point where
00556      * we can attempt to solve - further refinement may be needed in the loop
00557      * below. */
00558     if (!preRefineFE(i, feparm, fetk)) {
00559         Vnm_tprint( 2, "Error pre-refining mesh!\n");
00560         VJMPErr(0);
00561     }
00562
00563     /* Solve-estimate-refine */
00564     Vnm_tprint(2, "\n\nWARNING! DO NOT EXPECT PERFORMANCE OUT OF THE APBS/FETk\n");
00565     Vnm_tprint(2, "INTERFACE AT THIS TIME. THE FINITE ELEMENT SOLVER IS\n");
00566     Vnm_tprint(2, "CURRENTLY NOT OPTIMIZED FOR THE PB EQUATION. IF YOU WANT\n");
00567     Vnm_tprint(2, "PERFORMANCE, PLEASE USE THE MULTIGRID-BASED METHODS, E.G.\n");
00568     Vnm_tprint(2, "MG-AUTO, MG-PARA, and MG-MANUAL (SEE DOCS.)\n\n");
00569     Vnm_tprint(1, " Beginning solve-estimate-refine cycle:\n");
00570
00571     for (isolve=0; isolve<feparm->maxsolve; isolve++) {
00572         Vnm_tprint(1, " Solve #d...\n", isolve);
00573
00574         /* Attempt to solve the mesh by using one of MC's solver types. */
00575         if (!solveFE(i, pbeparm, feparm, fetk)) {
00576             Vnm_tprint(2, "ERROR SOLVING EQUATION!\n");
00577             VJMPErr(0);
00578         }
00579
00580         /* Calculate the total electrostatic energy. */
00581         if (!energyFE(nosh, i, fetk, &(nenergy[i]),
00582             &(totEnergy[i]), &(qfEnergy[i]),
00583             &(qmEnergy[i]), &(dielEnergy[i]))) {
00584             Vnm_tprint(2, "ERROR SOLVING EQUATION!\n");
00585             VJMPErr(0);
00586         }
00587
00588         /* We're not going to refine if we've hit the max number
00589          * of solves */
00590         if (isolve < (feparm->maxsolve)-1) {
00591             /* Do a final error estimation and mesh refinement. */
00592             if (!postRefineFE(i, feparm, fetk)) {
00593                 break;
00594             }
00595         }
00596         bytesTotal = Vmem_bytesTotal();

```

```

00597             highWater = Vmem_highWaterTotal();
00598             Vnm_tprint(1, "          Current memory use:    %g MB\n",
00599                         ((double)bytesTotal/(1024.)/(1024.)));
00600             Vnm_tprint(1, "          High-water memory use:  %g MB\n",
00601                         ((double)highWater/(1024.)/(1024.)));
00602         }
00603
00604             Vnm_tprint(1, "   Writing FEM data to files.\n");
00605
00606             /* Save data. */
00607             if (!writedataFE(rank, nosh, pbeparm, fetk[i])) {
00608                 Vnm_tprint(2, "   Error while writing FEM data!\n");
00609             }
00610 #else /* ifdef HAVE_MC_H */
00611             Vnm_print(2, "Error!  APBS not compiled with FEtk!\n");
00612             exit(2);
00613 #endif /* ifdef HAVE_MC_H */
00614             break;
00615
00616             /* Do an apolar calculation */
00617             case NCT_APOL:
00618                 /* Copied from NCT_MG. See the note above (top of loop) for
00619                    information about this loop.
00620                 */
00621                 for (k=0; k<nosh->napol; k++) {
00622                     if (nosh->apol2calc[k] >= i) {
00623                         break;
00624                     }
00625                 }
00626
00627                 if (Vstring_strcasecmp(nosh->apolname[k], "") == 0) {
00628                     Vnm_tprint( 1, "CALCULATION #d: APOLAR\n", i+1);
00629                 } else {
00630                     Vnm_tprint( 1, "CALCULATION #d (%s): APOLAR\n",
00631                               i+1, nosh->apolname[k]);
00632                 }
00633
00634                 apolparm = nosh->calc[i]->apolparm;
00635                 // poor man's execution timer.
00636                 ts = clock();
00637                 rc = initAPOL(nosh, mem, param, apolparm, &(nforce[i]), &(atomForce[i]),
00638                             alist[(apolparm->molid)-1]);
00639                 Vnm_print(0, "initAPOL: Time elapsed: %f\n", ((double)clock() - ts) / CLOCKS_PER_SEC);
00640                 if (rc == 0) {
00641                     Vnm_tprint(2, "Error calculating apolar solvation quantities!\n");
00642                     VJMPERR1(0);
00643                 }
00644                 break;
00645
00646             /* Boundary Element (tabi) */
00647             case NCT_BEM:
00648 #ifdef ENABLE_BEM
00649                 /* What is this? This seems like a very awkward way to find
00650                    the right ELEC statement... */
00651                 for (k=0; k<nosh->nelec; k++) {
00652                     if (nosh->elec2calc[k] >= i) {
00653                         break;
00654                     }
00655                 }
00656                 if (Vstring_strcasecmp(nosh->elecname[k], "") == 0) {
00657                     Vnm_tprint( 1, "CALCULATION #d: BOUNDARY ELEMENT\n", i+1);
00658                 } else {
00659                     Vnm_tprint( 1, "CALCULATION #d (%s): BOUNDARY ELEMENT\n",
00660                               i+1, nosh->elecname[k]);
00661                 }
00662                 /* Useful local variables */
00663                 bemparm = nosh->calc[i]->bemparm;
00664                 pbeparm = nosh->calc[i]->pbeparm;
00665
00666                 /* Set up problem */
00667                 Vnm_tprint( 1, "   Setting up problem...\n");
00668
00669                 if (!initBEM(i,nosh, bemparm, pbeparm, pbe)) {
00670                     Vnm_tprint( 2, "Error setting up BEM calculation!\n");
00671                     VJMPERR1(0);
00672                 }
00673
00674                 /* Print problem parameters */
00675                 printBEMPARM(bemparm);
00676                 printPBEPARM(pbeparm);
00677

```

```

00678         /* Solve PDE */
00679         if (solveBEM(nosh, pbeparm, bemparm, bemparm->type) != 1) {
00680             Vnm_tprint(2, "Error solving PDE!\n");
00681             VJMPERR1(0);
00682         }
00683
00684         /* Write out energies */
00685         energyBEM(nosh, i,
00686                 &(nenergy[i]), &(totEnergy[i]), &(qfEnergy[i]),
00687                 &(qmEnergy[i]), &(dielEnergy[i]));
00688
00689         /* Write out forces */
00690         forceBEM(nosh, pbeparm, bemparm, &(nforce[i]),
00691                 &(atomForce[i]), alist);
00692
00693         /* Write out data folks might want */
00694         writedataBEM(rank, nosh, pbeparm);
00695
00696         /* Write matrix */
00697         writematBEM(rank, nosh, pbeparm);
00698
00699         /* If needed, cache atom energies */
00700         nenergy[i] = 0;
00701         if ((pbeparm->calcenergy == PCE_COMPS) && (outputformat !=
OUTPUT_NULL)){
00702             storeAtomEnergy(pmg[i], i, &(atomEnergy[i]), &(nenergy[i]));
00703         }
00704
00705         fflush(stdout);
00706         fflush(stderr);
00707     #else /* ifdef ENABLE_BEM */
00708         Vnm_tprint(2, "Error! APBS not compiled with BEM!\n");
00709         exit(2);
00710     #endif
00711         break;
00712
00713     /* geometric flow */
00714     case NCT_GEOFLOW:
00715         /* What is this? This seems like a very awkward way to find
00716         the right ELEC statement... */
00717         for (k=0; k<nosh->nelec; k++) {
00718             if (nosh->elec2calc[k] >= i) {
00719                 break;
00720             }
00721         }
00722         if (Vstring_strcasecmp(nosh->elecname[k], "") == 0) {
00723             Vnm_tprint( 1, "CALCULATION #d: GEOMETRIC FLOW\n", i+1);
00724         } else {
00725             Vnm_tprint( 1, "CALCULATION #d (%s): GEOMETRIC FLOW\n",
i+1, nosh->elecname[k]);
00726         }
00727
00728         /* Useful local variables */
00729         geoflowparm = nosh->calc[i]->geoflowparm;
00730         apolparm = nosh->calc[i]->apolparm;
00731         pbeparm = nosh->calc[i]->pbeparm;
00732
00733         /* Set up problem */
00734         Vnm_tprint( 1, " Setting up problem...\n");
00735
00736         if (!initGEOFLOW(i,nosh, geoflowparm, pbeparm, pbe)) {
00737             Vnm_tprint( 2, "Error setting up GEOFLOW calculation!\n");
00738             VJMPERR1(0);
00739         }
00740
00741         /* Print problem parameters */
00742         printGEOFLOWPARAM(geoflowparm);
00743         printPBEPARM(pbeparm);
00744
00745         /* Solve PDE */
00746         if (solveGEOFLOW(alist, nosh, pbeparm, apolparm, geoflowparm, geoflowparm->
type) != 1) {
00747             Vnm_tprint(2, "Error solving PDE!\n");
00748             VJMPERR1(0);
00749         }
00750
00751         /* Write out energies */
00752         energyGEOFLOW(nosh, i,
00753                 &(nenergy[i]), &(totEnergy[i]), &(qfEnergy[i]),
00754                 &(qmEnergy[i]), &(dielEnergy[i]));
00755
00756         /* Write out forces */

```



```

00757         forceGEOFLOW(nosh, pbeparm, geoflowparm, &(nforce[i]),
00758                     &(atomForce[i]), alist);
00759
00760         /* Write out data folks might want */
00761         writedataGEOFLOW(rank, nosh, pbeparm);
00762
00763         /* Write matrix */
00764         writematGEOFLOW(rank, nosh, pbeparm);
00765
00766         /* If needed, cache atom energies */
00767         nenergy[i] = 0;
00768         if ((pbeparm->calcenergy == PCE_COMPS) && (outputformat !=
OUTPUT_NULL)){
00769             storeAtomEnergy(pmg[i], i, &(atomEnergy[i]), &(nenergy[i]));
00770         }
00771
00772         fflush(stdout);
00773         fflush(stderr);
00774         break;
00775
00776         default:
00777             Vnm_tprint(2, " Unknown calculation type (%d)!\n", nosh->calc[i]->
calctype);
00778             exit(2);
00779             break;
00780     }
00781 }
00782
00783 //Clear out the parameter file memory
00784 if(param != VNULL) Vparam_dtor(&param);
00785
00786 /* ***** HANDLE PRINT STATEMENTS ***** */
00787 if (nosh->nprint > 0) {
00788     Vnm_tprint( 1, "-----\n");
00789     Vnm_tprint( 1, "PRINT STATEMENTS\n");
00790 }
00791 for (i=0; i<nosh->nprint; i++) {
00792     /* Print energy */
00793     if (nosh->printwhat[i] == NPT_ENERGY) {
00794         printEnergy(com, nosh, totEnergy, i);
00795         /* Print force */
00796     } else if (nosh->printwhat[i] == NPT_FORCE) {
00797         printForce(com, nosh, nforce, atomForce, i);
00798     } else if (nosh->printwhat[i] == NPT_ELECENERGY) {
00799         printElecEnergy(com, nosh, totEnergy, i);
00800     } else if (nosh->printwhat[i] == NPT_ELECFORCE) {
00801         printElecForce(com, nosh, nforce, atomForce, i);
00802     } else if (nosh->printwhat[i] == NPT_APOLENERGY) {
00803         printApolEnergy(nosh, i);
00804     } else if (nosh->printwhat[i] == NPT_APOLFORCE) {
00805         printApolForce(com, nosh, nforce, atomForce, i);
00806     } else {
00807         Vnm_tprint( 2, "Undefined PRINT keyword!\n");
00808         break;
00809     }
00810 }
00811 Vnm_tprint( 1, "-----\n");
00812
00813 /* ***** HANDLE LOGGING ***** */
00814 if (outputformat == OUTPUT_FLAT) {
00815     Vnm_tprint(2, " Writing data to flat file %s...\n", output_path);
00816     writedataFlat(nosh, com, output_path, totEnergy, qfEnergy, qmEnergy,
00817                 dielEnergy, nenergy, atomEnergy, nforce, atomForce);
00818 }
00819
00820 /* Destroy energy arrays if they still exist */
00821
00822 for (i=0; i<nosh->ncalc; i++) {
00823     if (nenergy[i] > 0) Vmem_free(mem, nenergy[i], sizeof(double),
00824                               (void **)&(atomEnergy[i]));
00825 }
00826
00827 /* ***** GARBAGE COLLECTION ***** */
00828
00829 Vnm_tprint( 1, "CLEANING UP AND SHUTTING DOWN...\n");
00830 /* Clean up APBS structures */
00831 killForce(mem, nosh, nforce, atomForce);
00832 killEnergy();
00833 killMG(nosh, pbe, pmgp, pmg);

```

```

00836 #ifdef HAVE_MC_H
00837     killFE(nosh, pbe, fetk, gm);
00838 #endif
00839     killChargeMaps(nosh, chargeMap);
00840     killKappaMaps(nosh, kappaMap);
00841     killDielMaps(nosh, dielXMap, dielYMap, dielZMap);
00842     killMolecules(nosh, alist);
00843     NOsh_dtor(&nosh);
00844
00845     /* Memory statistics */
00846     bytesTotal = Vmem_bytesTotal();
00847     highWater = Vmem_highWaterTotal();
00848     Vnm_tprint( 1, "Final memory usage:  %4.3f MB total, %4.3f MB high water\n",
00849                (double) (bytesTotal) / (1024.*1024.),
00850                (double) (highWater) / (1024.*1024.));
00851
00852     /* Clean up MALOC structures */
00853     Vcom_dtor(&com);
00854     Vmem_dtor(&mem);
00855
00856     /* And now it's time to so "so long"... */
00857     Vnm_tprint(1, "\n\n");
00858     Vnm_tprint( 1, "Thanks for using APBS!\n\n");
00859
00860 #if defined(DEBUG_MAC_OSX_OCL)
00861     mets_(&mbeg, "Main Program CL");
00862 #endif
00863 #if defined(DEBUG_MAC_OSX_STANDARD)
00864     mets_(&mbeg, "Main Program Standard");
00865 #endif
00866
00867     /* This should be last */
00868     Vnm_tstop(APBS_TIMER_WALL_CLOCK, "APBS WALL CLOCK");
00869     Vnm_flush(1);
00870     Vnm_flush(2);
00871     Vcom_finalize();
00872
00873     fflush(NULL);
00874
00875     return 0;
00876
00877     ERROR1:
00878     Vcom_finalize();
00879     Vcom_dtor(&com);
00880     Vmem_dtor(&mem);
00881     return APBSRC;
00882 }

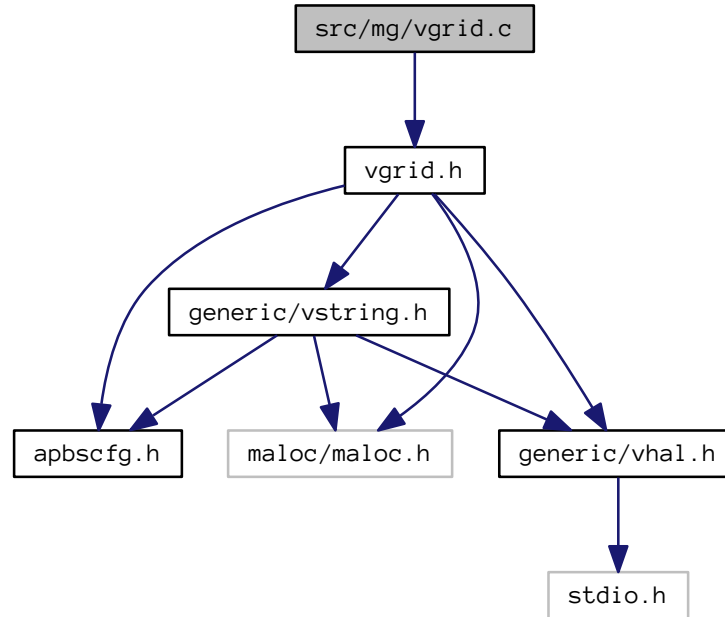
```

10.89 src/mg/vgrid.c File Reference

Class Vgrid methods.

```
#include "vgrid.h"
```

Include dependency graph for vgrid.c:



Macros

- `#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]

10.89.1 Detailed Description

Class Vgrid methods.

Author

Nathan Baker

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
```

```

* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vgrid.c](#).

10.89.2 Function Documentation

10.89.2.1 `VPUBLIC void Vgrid_writeGZ (Vgrid * thee, const char * iodev, const char * iofmt, const char * thost, const char * fname, char * title, double * pvec)`

Write out OpenDX data in GZIP format.

Author

Dave Gohara

Parameters

<i>thee</i>	Object to hold new grid data
<i>iodev</i>	I/O device
<i>iofmt</i>	I/O format
<i>thost</i>	Remote host name
<i>fname</i>	File name
<i>title</i>	Data title
<i>pvec</i>	Masking vector (0 = not written)

Definition at line [812](#) of file [vgrid.c](#).

10.90 vgrid.c

```

00001
00057 #include "vgrid.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_VGRID)
00062     VPUBLIC unsigned long int Vgrid_memChk(Vgrid *thee) {
00063         return Vmem_bytes(thee->mem);
00064     }
00065 #endif
00066 #define IJK(i,j,k)  (((k)*(nx)*(ny))+((j)*(nx))+(i))
00067
00068 #ifdef _WIN32
00069 #include <float.h>
00070 int isnan(double d)
00071 {
00072     return _isnan(d);
00073 }
00074 #endif
00075
00076 VPRIVATE char *MCwhiteChars = " =,;\t\n";
00077 VPRIVATE char *MCcommChars  = "#%";
00078 VPRIVATE double Vcompare;
00079 VPRIVATE char Vprecision[26];
00080
00081 /* ////////////////////////////////////////
00082 // Routine:  Vgrid_ctor
00083 // Author:   Nathan Baker
00085 VPUBLIC Vgrid* Vgrid_ctor(int nx,

```

```

00086             int ny,
00087             int nz,
00088             double hx,
00089             double hy,
00090             double hzed,
00091             double xmin,
00092             double ymin,
00093             double zmin,
00094             double *data
00095         ) {
00096
00097     Vgrid *thee = VNULL;
00098
00099     thee = (Vgrid*)Vmem_malloc(VNULL, 1, sizeof(Vgrid));
00100     VASSERT(thee != VNULL);
00101     VASSERT(Vgrid_ctor2(thee, nx, ny, nz, hx, hy, hzed,
00102         xmin, ymin, zmin, data));
00103
00104     return thee;
00105 }
00106
00107 /* ////////////////////////////////////////////////////////////////////
00108 // Routine:  Vgrid_ctor2
00109 // Author:   Nathan Baker
00111 VPUBLIC int Vgrid_ctor2(Vgrid *thee, int nx, int ny, int nz,
00112     double hx, double hy, double hzed,
00113     double xmin, double ymin, double zmin,
00114     double *data) {
00115
00116     if (thee == VNULL) return 0;
00117     thee->nx = nx;
00118     thee->ny = ny;
00119     thee->nz = nz;
00120     thee->hx = hx;
00121     thee->hy = hy;
00122     thee->hzed = hzed;
00123     thee->xmin = xmin;
00124     thee->xmax = xmin + (nx-1)*hx;
00125     thee->ymin = ymin;
00126     thee->ymax = ymin + (ny-1)*hy;
00127     thee->zmin = zmin;
00128     thee->zmax = zmin + (nz-1)*hzed;
00129     if (data == VNULL) {
00130         thee->ctordata = 0;
00131         thee->readdata = 0;
00132     } else {
00133         thee->ctordata = 1;
00134         thee->readdata = 0;
00135         thee->data = data;
00136     }
00137
00138     thee->mem = Vmem_ctor("APBS:VGRID");
00139
00140     Vcompare = pow(10,-1*(VGRID_DIGITS - 2));
00141     sprintf(Vprecision,"%12.2de %12.2de %12.2de", VGRID_DIGITS,
00142         VGRID_DIGITS, VGRID_DIGITS);
00143
00144     return 1;
00145 }
00146
00147 /* ////////////////////////////////////////////////////////////////////
00148 // Routine:  Vgrid_dtor
00149 // Author:   Nathan Baker
00151 VPUBLIC void Vgrid_dtor(Vgrid **thee) {
00152
00153     if ((*thee) != VNULL) {
00154         Vgrid_dtor2(*thee);
00155         Vmem_free(VNULL, 1, sizeof(Vgrid), (void **)thee);
00156         (*thee) = VNULL;
00157     }
00158 }
00159
00160 /* ////////////////////////////////////////////////////////////////////
00161 // Routine:  Vgrid_dtor2
00162 // Author:   Nathan Baker
00164 VPUBLIC void Vgrid_dtor2(Vgrid *thee) {
00165
00166     if (thee->readdata) {
00167         Vmem_free(thee->mem, (thee->nx*thee->ny*thee->nz), sizeof(double),
00168             (void **)&(thee->data));
00169     }

```

```

00170     Vmem_dtor(&(thee->mem));
00171
00172 }
00173
00174 /* ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00175 // Routine:  Vgrid_value
00176 // Author:   Nathan Baker
00177 VPUBLIC int Vgrid_value(Vgrid *thee, double pt[3], double *value) {
00178
00179     int nx, ny, nz;
00180     size_t ihi, jhi, khi, ilo, jlo, klo;
00181     double hx, hy, hzed, xmin, ymin, zmin, ifloat, jfloat, kfloat;
00182     double xmax, ymax, zmax;
00183     double u, dx, dy, dz;
00184
00185     if (thee == VNULL) {
00186         Vnm_print(2, "Vgrid_value:  Error -- got VNULL thee!\n");
00187         VASSERT(0);
00188     }
00189     if (!(thee->ctordata || thee->readdata)) {
00190         Vnm_print(2, "Vgrid_value:  Error -- no data available!\n");
00191         VASSERT(0);
00192     }
00193
00194     nx = thee->nx;
00195     ny = thee->ny;
00196     nz = thee->nz;
00197     hx = thee->hx;
00198     hy = thee->hy;
00199     hzed = thee->hzed;
00200     xmin = thee->xmin;
00201     ymin = thee->ymin;
00202     zmin = thee->zmin;
00203     xmax = thee->xmax;
00204     ymax = thee->ymax;
00205     zmax = thee->zmax;
00206
00207     u = 0;
00208
00209     ifloat = (pt[0] - xmin)/hx;
00210     jfloat = (pt[1] - ymin)/hy;
00211     kfloat = (pt[2] - zmin)/hzed;
00212
00213     ihi = (int)ceil(ifloat);
00214     jhi = (int)ceil(jfloat);
00215     khi = (int)ceil(kfloat);
00216     ilo = (int)floor(ifloat);
00217     jlo = (int)floor(jfloat);
00218     klo = (int)floor(kfloat);
00219     if (VABS(pt[0] - xmin) < Vcompare) ilo = 0;
00220     if (VABS(pt[1] - ymin) < Vcompare) jlo = 0;
00221     if (VABS(pt[2] - zmin) < Vcompare) klo = 0;
00222     if (VABS(pt[0] - xmax) < Vcompare) ihi = nx-1;
00223     if (VABS(pt[1] - ymax) < Vcompare) jhi = ny-1;
00224     if (VABS(pt[2] - zmax) < Vcompare) khi = nz-1;
00225
00226     /* See if we're on the mesh */
00227     if ((ihi<nx) && (jhi<ny) && (khi<nz) &&
00228         (ilo>=0) && (jlo>=0) && (klo>=0)) {
00229
00230         dx = ifloat - (double)(ilo);
00231         dy = jfloat - (double)(jlo);
00232         dz = kfloat - (double)(klo);
00233         u = dx      *dy      *dz      *(thee->data[IJK(ihi,jhi,khi)])
00234           + dx      *(1.0-dy)*dz      *(thee->data[IJK(ihi,jlo,khi)])
00235           + dx      *dy      *(1.0-dz)*(thee->data[IJK(ihi,jhi,klo)])
00236           + dx      *(1.0-dy)*(1.0-dz)*(thee->data[IJK(ihi,jlo,klo)])
00237           + (1.0-dx)*dy      *dz      *(thee->data[IJK(ilo,jhi,khi)])
00238           + (1.0-dx)*(1.0-dy)*dz      *(thee->data[IJK(ilo,jlo,khi)])
00239           + (1.0-dx)*dy      *(1.0-dz)*(thee->data[IJK(ilo,jhi,klo)])
00240           + (1.0-dx)*(1.0-dy)*(1.0-dz)*(thee->data[IJK(ilo,jlo,klo)]);
00241
00242         *value = u;
00243
00244         if (isnan(u)) {
00245             Vnm_print(2, "Vgrid_value:  Got NaN!\n");
00246             Vnm_print(2, "Vgrid_value:  (x, y, z) = (%4.3f, %4.3f, %4.3f)\n",
00247                 pt[0], pt[1], pt[2]);
00248             Vnm_print(2, "Vgrid_value:  (ihi, jhi, khi) = (%d, %d, %d)\n",
00249                 ihi, jhi, khi);
00250             Vnm_print(2, "Vgrid_value:  (ilo, jlo, klo) = (%d, %d, %d)\n",
00251                 ilo, jlo, klo);

```



```

00252         ilo, jlo, klo);
00253     Vnm_print(2, "Vgrid_value:  (nx, ny, nz) = (%d, %d, %d)\n",
00254         nx, ny, nz);
00255     Vnm_print(2, "Vgrid_value:  (dx, dy, dz) = (%4.3f, %4.3f, %4.3f)\n",
00256         dx, dy, dz);
00257     Vnm_print(2, "Vgrid_value:  data[IJK(ihi,jhi,khi)] = %g\n",
00258         thee->data[IJK(ihi,jhi,khi)]);
00259     Vnm_print(2, "Vgrid_value:  data[IJK(ihi,jlo,khi)] = %g\n",
00260         thee->data[IJK(ihi,jlo,khi)]);
00261     Vnm_print(2, "Vgrid_value:  data[IJK(ihi,jhi,klo)] = %g\n",
00262         thee->data[IJK(ihi,jhi,klo)]);
00263     Vnm_print(2, "Vgrid_value:  data[IJK(ihi,jlo,klo)] = %g\n",
00264         thee->data[IJK(ihi,jlo,klo)]);
00265     Vnm_print(2, "Vgrid_value:  data[IJK(ilo,jhi,khi)] = %g\n",
00266         thee->data[IJK(ilo,jhi,khi)]);
00267     Vnm_print(2, "Vgrid_value:  data[IJK(ilo,jlo,khi)] = %g\n",
00268         thee->data[IJK(ilo,jlo,khi)]);
00269     Vnm_print(2, "Vgrid_value:  data[IJK(ilo,jhi,klo)] = %g\n",
00270         thee->data[IJK(ilo,jhi,klo)]);
00271     Vnm_print(2, "Vgrid_value:  data[IJK(ilo,jlo,klo)] = %g\n",
00272         thee->data[IJK(ilo,jlo,klo)]);
00273     }
00274     return 1;
00275
00276 } else {
00277
00278     *value = 0;
00279     return 0;
00280 }
00281
00282 return 0;
00283
00284 }
00285 }
00286
00287 /* ///////////////////////////////////////////////////////////////////
00288 // Routine:  Vgrid_curvature
00289 //
00290 //   Notes:  cflag=0 ==> Reduced Maximal Curvature
00291 //           cflag=1 ==> Mean Curvature (Laplace)
00292 //           cflag=2 ==> Gauss Curvature
00293 //           cflag=3 ==> True Maximal Curvature
00294 //
00295 // Authors:  Stephen Bond and Nathan Baker
00297 VPUBLIC int Vgrid_curvature(Vgrid *thee, double pt[3], int cflag,
00298     double *value) {
00299
00300     double hx, hy, hzed, curv;
00301     double dxx, dyy, dzz;
00302     double uleft, umid, uright, testpt[3];
00303
00304     if (thee == VNULL) {
00305         Vnm_print(2, "Vgrid_curvature:  Error -- got VNULL thee!\n");
00306         VASSERT(0);
00307     }
00308     if (!(thee->ctordata || thee->readdata)) {
00309         Vnm_print(2, "Vgrid_curvature:  Error -- no data available!\n");
00310         VASSERT(0);
00311     }
00312
00313     hx = thee->hx;
00314     hy = thee->hy;
00315     hzed = thee->hzed;
00316
00317     curv = 0.0;
00318
00319     testpt[0] = pt[0];
00320     testpt[1] = pt[1];
00321     testpt[2] = pt[2];
00322
00323     /* Compute 2nd derivative in the x-direction */
00324     VJMPERR1(Vgrid_value( thee, testpt, &umid));
00325     testpt[0] = pt[0] - hx;
00326     VJMPERR1(Vgrid_value( thee, testpt, &uleft));
00327     testpt[0] = pt[0] + hx;
00328     VJMPERR1(Vgrid_value( thee, testpt, &uright));
00329     testpt[0] = pt[0];
00330
00331     dxx = (uright - 2*umid + uleft)/(hx*hx);
00332
00333     /* Compute 2nd derivative in the y-direction */

```

```

00334     VJMPERR1(Vgrid_value( thee, testpt, &umid));
00335     testpt[1] = pt[1] - hy;
00336     VJMPERR1(Vgrid_value( thee, testpt, &uleft));
00337     testpt[1] = pt[1] + hy;
00338     VJMPERR1(Vgrid_value( thee, testpt, &uright));
00339     testpt[1] = pt[1];
00340
00341     dyy = (uright - 2*umid + uleft)/(hy*hy);
00342
00343     /* Compute 2nd derivative in the z-direction */
00344     VJMPERR1(Vgrid_value( thee, testpt, &umid));
00345     testpt[2] = pt[2] - hzed;
00346     VJMPERR1(Vgrid_value( thee, testpt, &uleft));
00347     testpt[2] = pt[2] + hzed;
00348     VJMPERR1(Vgrid_value( thee, testpt, &uright));
00349
00350     dzz = (uright - 2*umid + uleft)/(hzed*hzed);
00351
00352
00353     if ( cflag == 0 ) {
00354         curv = fabs(dxx);
00355         curv = ( curv > fabs(dyy) ) ? curv : fabs(dyy);
00356         curv = ( curv > fabs(dzz) ) ? curv : fabs(dzz);
00357     } else if ( cflag == 1 ) {
00358         curv = (dxx + dyy + dzz)/3.0;
00359     } else {
00360         Vnm_print(2, "Vgrid_curvature: support for cflag = %d not available!\n", cflag);
00361         VASSERT( 0 ); /* Feature Not Coded Yet! */
00362     }
00363
00364     *value = curv;
00365     return 1;
00366
00367     VERROR1:
00368     return 0;
00369
00370 }
00371
00372 /* ////////////////////////////////////////
00373 // Routine: Vgrid_gradient
00374 //
00375 // Authors: Nathan Baker and Stephen Bond
00377 VPUBLIC int Vgrid_gradient(Vgrid *thee, double pt[3], double grad[3]) {
00378
00379     double hx, hy, hzed;
00380     double uleft, umid, uright, testpt[3];
00381     int haveleft, haveright;
00382
00383     if (thee == VNULL) {
00384         Vnm_print(2, "Vgrid_gradient: Error -- got VNULL thee!\n");
00385         VASSERT(0);
00386     }
00387     if (!(thee->ctordata || thee->readdata)) {
00388         Vnm_print(2, "Vgrid_gradient: Error -- no data available!\n");
00389         VASSERT(0);
00390     }
00391
00392     hx = thee->hx;
00393     hy = thee->hy;
00394     hzed = thee->hzed;
00395
00396     /* Compute derivative in the x-direction */
00397     testpt[0] = pt[0];
00398     testpt[1] = pt[1];
00399     testpt[2] = pt[2];
00400     VJMPERR1( Vgrid_value( thee, testpt, &umid));
00401     testpt[0] = pt[0] - hx;
00402     if (Vgrid_value( thee, testpt, &uleft)) haveleft = 1;
00403     else haveleft = 0;
00404     testpt[0] = pt[0] + hx;
00405     if (Vgrid_value( thee, testpt, &uright)) haveright = 1;
00406     else haveright = 0;
00407     if (haveright && haveleft) grad[0] = (uright - uleft)/(2*hx);
00408     else if (haveright) grad[0] = (uright - umid)/hx;
00409     else if (haveleft) grad[0] = (umid - uleft)/hx;
00410     else VJMPERR1(0);
00411
00412     /* Compute derivative in the y-direction */
00413     testpt[0] = pt[0];
00414     testpt[1] = pt[1];
00415     testpt[2] = pt[2];

```

```

00416     VJMPERR1(Vgrid_value(thee, testpt, &umid));
00417     testpt[1] = pt[1] - hy;
00418     if (Vgrid_value(thee, testpt, &uleft)) haveleft = 1;
00419     else haveleft = 0;
00420     testpt[1] = pt[1] + hy;
00421     if (Vgrid_value(thee, testpt, &uright)) haveright = 1;
00422     else haveright = 0;
00423     if (haveright && haveleft) grad[1] = (uright - uleft)/(2*hy);
00424     else if (haveright) grad[1] = (uright - umid)/hy;
00425     else if (haveleft) grad[1] = (umid - uleft)/hy;
00426     else VJMPERR1(0);
00427
00428     /* Compute derivative in the z-direction */
00429     testpt[0] = pt[0];
00430     testpt[1] = pt[1];
00431     testpt[2] = pt[2];
00432     VJMPERR1(Vgrid_value(thee, testpt, &umid));
00433     testpt[2] = pt[2] - hzed;
00434     if (Vgrid_value(thee, testpt, &uleft)) haveleft = 1;
00435     else haveleft = 0;
00436     testpt[2] = pt[2] + hzed;
00437     if (Vgrid_value(thee, testpt, &uright)) haveright = 1;
00438     else haveright = 0;
00439     if (haveright && haveleft) grad[2] = (uright - uleft)/(2*hzed);
00440     else if (haveright) grad[2] = (uright - umid)/hzed;
00441     else if (haveleft) grad[2] = (umid - uleft)/hzed;
00442     else VJMPERR1(0);
00443
00444     return 1;
00445
00446     VERROR1:
00447     return 0;
00448
00449 }
00450
00451 /* ////////////////////////////////////// */
00452 // Routine:  Vgrid_readGZ
00453 //
00454 // Author:   David Gohara
00455 #ifdef HAVE_ZLIB
00456 #define off_t long
00457 #include "zlib.h"
00458 #endif
00459 #endif
00460 VPUBLIC int Vgrid_readGZ(Vgrid *thee, const char *fname) {
00461
00462 #ifdef HAVE_ZLIB
00463     size_t i, j, k, u;
00464     size_t len; // Temporary counter variable for loop conditionals
00465     size_t header, incr;
00466     double *temp;
00467     double dtmpl, dtmp2, dtmp3;
00468     gzFile infile;
00469     char line[VMAX_ARGLEN];
00470
00471     header = 0;
00472
00473     /* Check to see if the existing data is null and, if not, clear it out */
00474     if (thee->data != VNULL) {
00475         Vnm_print(1, "%s:  destroying existing data!\n", __func__);
00476         Vmem_free(thee->mem, thee->nx * thee->ny * thee->nz, sizeof(double),
00477                 (void **)&(thee->data));
00478     }
00479
00480     thee->readdata = 1;
00481     thee->ctordata = 0;
00482
00483     infile = gzopen(fname, "rb");
00484     if (infile == Z_NULL) {
00485         Vnm_print(2, "%s:  Problem opening compressed file %s\n", __func__, fname);
00486         return VRC_FAILURE;
00487     }
00488
00489     thee->hx = 0.0;
00490     thee->hy = 0.0;
00491     thee->hzed = 0.0;
00492
00493     //read data here
00494     while (header < 7) {
00495         if (gzgets(infile, line, VMAX_ARGLEN) == Z_NULL){
00496             return VRC_FAILURE;
00497         }

```

```

00498
00499 // Skip comments and newlines
00500 if(strncmp(line, "#", 1) == 0) continue;
00501 if(line[0] == '\n') continue;
00502
00503 switch (header) {
00504     case 0:
00505         sscanf(line, "object 1 class gridpositions counts %d %d %d",
00506             &(thee->nx), &(thee->ny), &(thee->nz));
00507         break;
00508     case 1:
00509         sscanf(line, "origin %lf %lf %lf",
00510             &(thee->xmin), &(thee->ymin), &(thee->zmin));
00511         break;
00512     case 2:
00513     case 3:
00514     case 4:
00515         sscanf(line, "delta %lf %lf %lf", &dtmpl, &dtmp2, &dtmp3);
00516         thee->hx += dtmpl;
00517         thee->hy += dtmp2;
00518         thee->hz += dtmp3;
00519         break;
00520     default:
00521         break;
00522 }
00523
00524 header++;
00525 }
00526
00527 /* Allocate space for the data */
00528 Vnm_print(0, "%s: allocating %d x %d x %d doubles for storage\n",
00529     __func__, thee->nx, thee->ny, thee->nz);
00530 len = thee->nx * thee->ny * thee->nz;
00531
00532 thee->data = VNULL;
00533 thee->data = Vmem_malloc(thee->mem, len, sizeof(double));
00534 if (thee->data == VNULL) {
00535     Vnm_print(2, "%s: Unable to allocate space for data!\n", __func__);
00536     return 0;
00537 }
00538
00539 /* Allocate a temporary buffer to store the compressed
00540 * data into (column major order). Add 2 to ensure the buffer is
00541 * big enough to take extra data on the final read loop.
00542 */
00543 temp = (double *)malloc(len * (2 * sizeof(double)));
00544
00545 for (i = 0; i < len; i += 3){
00546     memset(&line, 0, sizeof(line));
00547     gzgets(infile, line, VMAX_ARGLEN);
00548     sscanf(line, "%lf %lf %lf", &temp[i], &temp[i+1], &temp[i+2]);
00549 }
00550
00551 /* Now move the data to row major order */
00552 incr = 0;
00553 for (i=0; i<thee->nx; i++) {
00554     for (j=0; j<thee->ny; j++) {
00555         for (k=0; k<thee->nz; k++) {
00556             u = k*(thee->nx)*(thee->ny)+j*(thee->nx)+i;
00557             (thee->data)[u] = temp[incr++];
00558         }
00559     }
00560 }
00561
00562 /* calculate grid maxima */
00563 thee->xmax = thee->xmin + (thee->nx-1)*thee->hx;
00564 thee->ymax = thee->ymin + (thee->ny-1)*thee->hy;
00565 thee->zmax = thee->zmin + (thee->nz-1)*thee->hz;
00566
00567 /* Close off the socket */
00568 gzclose(infile);
00569 free(temp);
00570 #else
00571
00572 Vnm_print(0, "WARNING\n");
00573 Vnm_print(0, "Vgrid_readGZ: gzip read/write support is disabled in this build\n");
00574 Vnm_print(0, "Vgrid_readGZ: configure and compile without the --disable-zlib flag.\n");
00575 Vnm_print(0, "WARNING\n");
00576 #endif
00577 return VRC_SUCCESS;
00578 }

```

```

00579
00584 VPUBLIC int Vgrid_readDX(Vgrid *thee,
00585                          const char *iodev,
00586                          const char *iofmt,
00587                          const char *thost,
00588                          const char *fname
00589                          ) {
00590
00591     size_t i, j, k, itmp, u;
00592     double dtmp;
00593     char tok[VMAX_BUFSIZE];
00594     Vio *sock;
00595
00596     /* Check to see if the existing data is null and, if not, clear it out */
00597     if (thee->data != VNULL) {
00598         Vnm_print(1, "Vgrid_readDX: destroying existing data!\n");
00599         Vmem_free(thee->mem, (thee->nx*thee->ny*thee->nz), sizeof(double),
00600                  (void **)&(thee->data)); }
00601     thee->readdata = 1;
00602     thee->ctordata = 0;
00603
00604     /* Set up the virtual socket */
00605     sock = Vio_ctor(iodev, iofmt, thost, fname, "r");
00606     if (sock == VNULL) {
00607         Vnm_print(2, "Vgrid_readDX: Problem opening virtual socket %s\n",
00608                 fname);
00609         return 0;
00610     }
00611     if (Vio_accept(sock, 0) < 0) {
00612         Vnm_print(2, "Vgrid_readDX: Problem accepting virtual socket %s\n",
00613                 fname);
00614         return 0;
00615     }
00616
00617     Vio_setWhiteChars(sock, MCwhiteChars);
00618     Vio_setCommChars(sock, MCcommChars);
00619
00620     /* Read in the DX regular positions */
00621     /* Get "object" */
00622     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00623     VJMPERR1(!strcmp(tok, "object"));
00624     /* Get "1" */
00625     VJMPERR2(1 == Vio_scanf(sock, "%d", &itmp));
00626     /* Get "class" */
00627     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00628     VJMPERR1(!strcmp(tok, "class"));
00629     /* Get "gridpositions" */
00630     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00631     VJMPERR1(!strcmp(tok, "gridpositions"));
00632     /* Get "counts" */
00633     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00634     VJMPERR1(!strcmp(tok, "counts"));
00635     /* Get nx */
00636     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00637     VJMPERR1(1 == sscanf(tok, "%d", &(thee->nx)));
00638     /* Get ny */
00639     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00640     VJMPERR1(1 == sscanf(tok, "%d", &(thee->ny)));
00641     /* Get nz */
00642     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00643     VJMPERR1(1 == sscanf(tok, "%d", &(thee->nz)));
00644     Vnm_print(0, "Vgrid_readDX: Grid dimensions %d x %d x %d grid\n",
00645             thee->nx, thee->ny, thee->nz);
00646     /* Get "origin" */
00647     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00648     VJMPERR1(!strcmp(tok, "origin"));
00649     /* Get xmin */
00650     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00651     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->xmin)));
00652     /* Get ymin */
00653     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00654     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->ymin)));
00655     /* Get zmin */
00656     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00657     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->zmin)));
00658     Vnm_print(0, "Vgrid_readDX: Grid origin = (%g, %g, %g)\n",
00659             thee->xmin, thee->ymin, thee->zmin);
00660     /* Get "delta" */
00661     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00662     VJMPERR1(!strcmp(tok, "delta"));
00663     /* Get hx */

```

```

00664     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00665     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->hx)));
00666     /* Get 0.0 */
00667     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00668     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00669     VJMPERR1(dtmp == 0.0);
00670     /* Get 0.0 */
00671     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00672     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00673     VJMPERR1(dtmp == 0.0);
00674     /* Get "delta" */
00675     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00676     VJMPERR1(!strcmp(tok, "delta"));
00677     /* Get 0.0 */
00678     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00679     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00680     VJMPERR1(dtmp == 0.0);
00681     /* Get hy */
00682     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00683     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->hy)));
00684     /* Get 0.0 */
00685     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00686     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00687     VJMPERR1(dtmp == 0.0);
00688     /* Get "delta" */
00689     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00690     VJMPERR1(!strcmp(tok, "delta"));
00691     /* Get 0.0 */
00692     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00693     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00694     VJMPERR1(dtmp == 0.0);
00695     /* Get 0.0 */
00696     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00697     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00698     VJMPERR1(dtmp == 0.0);
00699     /* Get hz */
00700     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00701     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->hz)));
00702     Vnm_print(0, "Vgrid_readDX: Grid spacings = (%g, %g, %g)\n",
00703         thee->hx, thee->hy, thee->hz);
00704     /* Get "object" */
00705     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00706     VJMPERR1(!strcmp(tok, "object"));
00707     /* Get "2" */
00708     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00709     /* Get "class" */
00710     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00711     VJMPERR1(!strcmp(tok, "class"));
00712     /* Get "gridconnections" */
00713     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00714     VJMPERR1(!strcmp(tok, "gridconnections"));
00715     /* Get "counts" */
00716     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00717     VJMPERR1(!strcmp(tok, "counts"));
00718     /* Get the dimensions again */
00719     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00720     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00721     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00722     /* Get "object" */
00723     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00724     VJMPERR1(!strcmp(tok, "object"));
00725     /* Get # */
00726     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00727     /* Get "class" */
00728     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00729     VJMPERR1(!strcmp(tok, "class"));
00730     /* Get "array" */
00731     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00732     VJMPERR1(!strcmp(tok, "array"));
00733     /* Get "type" */
00734     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00735     VJMPERR1(!strcmp(tok, "type"));
00736     /* Get "double" */
00737     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00738     VJMPERR1(!strcmp(tok, "double"));
00739     /* Get "rank" */
00740     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00741     VJMPERR1(!strcmp(tok, "rank"));
00742     /* Get # */
00743     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00744     /* Get "items" */

```

```

00745     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00746     VJMPERR1(!strcmp(tok, "items"));
00747     /* Get # */
00748     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00749     VJMPERR1(1 == sscanf(tok, "%lu", &itmp));
00750     u = (size_t)thee->nx * thee->ny * thee->nz;
00751     VJMPERR1(u == itmp);
00752     /* Get "data" */
00753     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00754     VJMPERR1(!strcmp(tok, "data"));
00755     /* Get "follows" */
00756     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00757     VJMPERR1(!strcmp(tok, "follows"));
00758
00759     /* Allocate space for the data */
00760     Vnm_print(0, "Vgrid_readDX: allocating %d x %d x %d doubles for storage\n",
00761         thee->nx, thee->ny, thee->nz);
00762     thee->data = VNULL;
00763     thee->data = (double*)Vmem_malloc(thee->mem, u, sizeof(double));
00764     if (thee->data == VNULL) {
00765         Vnm_print(2, "Vgrid_readDX: Unable to allocate space for data!\n");
00766         return 0;
00767     }
00768
00769     for (i=0; i<thee->nx; i++) {
00770         for (j=0; j<thee->ny; j++) {
00771             for (k=0; k<thee->nz; k++) {
00772                 u = k*(thee->nx)*(thee->ny)+j*(thee->nx)+i;
00773                 VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00774                 VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00775                 (thee->data)[u] = dtmp;
00776             }
00777         }
00778     }
00779
00780     /* calculate grid maxima */
00781     thee->xmax = thee->xmin + (thee->nx-1)*thee->hx;
00782     thee->ymax = thee->ymin + (thee->ny-1)*thee->hy;
00783     thee->zmax = thee->zmin + (thee->nz-1)*thee->hz;
00784
00785     /* Close off the socket */
00786     Vio_acceptFree(sock);
00787     Vio_dtor(&sock);
00788
00789     return 1;
00790
00791 ERROR1:
00792     Vio_dtor(&sock);
00793     Vnm_print(2, "Vgrid_readDX: Format problem with input file <%s>\n",
00794         fname);
00795     return 0;
00796
00797 ERROR2:
00798     Vio_dtor(&sock);
00799     Vnm_print(2, "Vgrid_readDX: I/O problem with input file <%s>\n",
00800         fname);
00801     return 0;
00802
00803
00804
00805 }
00806
00807 /* ////////////////////////////////////// */
00808 // Routine: Vgrid_writeGZ
00809 //
00810 // Author: Nathan Baker
00812 VPUBLIC void Vgrid_writeGZ(Vgrid *thee, const char *iodev, const char *iofmt,
00813     const char *thost, const char *fname, char *title, double *pvec) {
00814
00815 #ifdef HAVE_ZLIB
00816     double xmin, ymin, zmin, hx, hy, hzed;
00817
00818     int nx, ny, nz, nxPART, nyPART, nzPART;
00819     int usepart, gotit;
00820     size_t icol, i, j, k, u;
00821     double x, y, z, xminPART, yminPART, zminPART;
00822
00823     size_t txyz;
00824     double txmin, tymin, tzmin;
00825
00826     char header[8196];

```

```

00827     char footer[8196];
00828     char line[80];
00829     char newline[] = "\n";
00830     gzFile outfile;
00831     char precFormat[VMAX_BUFSIZE];
00832
00833     if (thee == VNULL) {
00834         Vnm_print(2, "Vgrid_writeGZ: Error -- got VNULL thee!\n");
00835         VASSERT(0);
00836     }
00837     if (!(thee->ctordata || thee->readdata)) {
00838         Vnm_print(2, "Vgrid_writeGZ: Error -- no data available!\n");
00839         VASSERT(0);
00840     }
00841
00842     hx = thee->hx;
00843     hy = thee->hy;
00844     hzed = thee->hz;
00845     nx = thee->nx;
00846     ny = thee->ny;
00847     nz = thee->nz;
00848     xmin = thee->xmin;
00849     ymin = thee->ymin;
00850     zmin = thee->zmin;
00851
00852     if (pvec == VNULL) usepart = 0;
00853     else usepart = 1;
00854
00855     /* Set up the virtual socket */
00856     Vnm_print(0, "Vgrid_writeGZ: Opening file...\n");
00857     outfile = gzopen(fname, "wb");
00858
00859     if (usepart) {
00860         /* Get the lower corner and number of grid points for the local
00861          * partition */
00862         xminPART = VLARGE;
00863         yminPART = VLARGE;
00864         zminPART = VLARGE;
00865         nxPART = 0;
00866         nyPART = 0;
00867         nzPART = 0;
00868         /* First, search for the lower corner */
00869         for (k=0; k<nz; k++) {
00870             z = k*hzed + zmin;
00871             for (j=0; j<ny; j++) {
00872                 y = j*hy + ymin;
00873                 for (i=0; i<nx; i++) {
00874                     x = i*hx + xmin;
00875                     if (pvec[IJK(i,j,k)] > 0.0) {
00876                         if (x < xminPART) xminPART = x;
00877                         if (y < yminPART) yminPART = y;
00878                         if (z < zminPART) zminPART = z;
00879                     }
00880                 }
00881             }
00882         }
00883         /* Now search for the number of grid points in the z direction */
00884         for (k=0; k<nz; k++) {
00885             gotit = 0;
00886             for (j=0; j<ny; j++) {
00887                 for (i=0; i<nx; i++) {
00888                     if (pvec[IJK(i,j,k)] > 0.0) {
00889                         gotit = 1;
00890                         break;
00891                     }
00892                 }
00893                 if (gotit) break;
00894             }
00895             if (gotit) nzPART++;
00896         }
00897         /* Now search for the number of grid points in the y direction */
00898         for (j=0; j<ny; j++) {
00899             gotit = 0;
00900             for (k=0; k<nz; k++) {
00901                 for (i=0; i<nx; i++) {
00902                     if (pvec[IJK(i,j,k)] > 0.0) {
00903                         gotit = 1;
00904                         break;
00905                     }
00906                 }
00907                 if (gotit) break;

```



```

00908         }
00909         if (gotit) nyPART++;
00910     }
00911     /* Now search for the number of grid points in the x direction */
00912     for (i=0; i<nx; i++) {
00913         gotit = 0;
00914         for (k=0; k<nz; k++) {
00915             for (j=0; j<ny; j++) {
00916                 if (pvec[IJK(i,j,k)] > 0.0) {
00917                     gotit = 1;
00918                     break;
00919                 }
00920             }
00921             if (gotit) break;
00922         }
00923         if (gotit) nxPART++;
00924     }
00925
00926     if ((nxPART != nx) || (nyPART != ny) || (nzPART != nz)) {
00927         Vnm_print(0, "Vgrid_writeGZ: printing only subset of domain\n");
00928     }
00929
00930     txyz = (nxPART*nyPART*nzPART);
00931     txmin = xminPART;
00932     tymin = yminPART;
00933     tzmin = zminPART;
00934
00935 }else {
00936
00937     txyz = (nx*ny*nz);
00938     txmin = xmin;
00939     tymin = ymin;
00940     tzmin = zmin;
00941
00942 }
00943
00944 /* Write off the title (if we're not XDR) */
00945 sprintf(header,
00946         "# Data from %s\n" \
00947         "# \n" \
00948         "# %s\n" \
00949         "# \n" \
00950         "object 1 class gridpositions counts %i %i %i\n" \
00951         "origin %12.6e %12.6e %12.6e\n" \
00952         "delta %12.6e 0.000000e+00 0.000000e+00\n" \
00953         "delta 0.000000e+00 %12.6e 0.000000e+00\n" \
00954         "delta 0.000000e+00 0.000000e+00 %12.6e\n" \
00955         "object 2 class gridconnections counts %i %i %i\n" \
00956         "object 3 class array type double rank 0 items %lu data follows\n",
00957         PACKAGE_STRING,title,nx,ny,nz,txmin,tymin,tzmin,
00958         hx,hy,hzed,nx,ny,nz,txyz);
00959 gzwrite(outfile, header, strlen(header)*sizeof(char));
00960
00961 /* Now write the data */
00962 icol = 0;
00963 for (i=0; i<nx; i++) {
00964     for (j=0; j<ny; j++) {
00965         for (k=0; k<nz; k++) {
00966             u = k*(nx)*(ny)+j*(nx)+i;
00967             if (pvec[u] > 0.0) {
00968                 sprintf(line, "%12.6e ", thee->data[u]);
00969                 gzwrite(outfile, line, strlen(line)*sizeof(char));
00970                 icol++;
00971                 if (icol == 3) {
00972                     icol = 0;
00973                     gzwrite(outfile, newline, strlen(newline)*sizeof(char));
00974                 }
00975             }
00976         }
00977     }
00978 }
00979 if(icol < 3){
00980     char newline[] = "\n";
00981     gzwrite(outfile, newline, strlen(newline)*sizeof(char));
00982 }
00983
00984 /* Create the field */
00985 sprintf(footer, "attribute \"dep\" string \"positions\"\n" \
00986         "object \"regular positions regular connections\" class field\n" \
00987         "component \"positions\" value 1\n" \
00988         "component \"connections\" value 2\n" \

```

```

00989         "component \"data\" value 3\n");
00990         gzwrite(outfile, footer, strlen(footer)*sizeof(char));
00991
00992         gzclose(outfile);
00993     #else
00994
00995         Vnm_print(0, "WARNING\n");
00996         Vnm_print(0, "Vgrid_readGZ:  gzip read/write support is disabled in this build\n");
00997         Vnm_print(0, "Vgrid_readGZ:  configure and compile without the --disable-zlib flag.\n");
00998         Vnm_print(0, "WARNING\n");
00999     #endif
01000 }
01001
01002 /* ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
01003 // Routine:  Vgrid_writeDX
01004 //
01005 // Author:   Nathan Baker
01007 VPUBLIC void Vgrid_writeDX(Vgrid *thee, const char *iodev, const char *iofmt,
01008     const char *thost, const char *fname, char *title, double *pvec) {
01009
01010     double xmin, ymin, zmin, hx, hy, hzed;
01011     int nx, ny, nz, nxPART, nyPART, nzPART;
01012     int usepart, gotit;
01013     size_t icol, i, j, k, u;
01014     double x, y, z, xminPART, yminPART, zminPART;
01015     Vio *sock;
01016     char precFormat[VMAX_BUFSIZE];
01017
01018     if (thee == VNULL) {
01019         Vnm_print(2, "Vgrid_writeDX:  Error -- got VNULL thee!\n");
01020         VASSERT(0);
01021     }
01022     if (!(thee->ctordata || thee->readdata)) {
01023         Vnm_print(2, "Vgrid_writeDX:  Error -- no data available!\n");
01024         VASSERT(0);
01025     }
01026
01027     hx = thee->hx;
01028     hy = thee->hy;
01029     hzed = thee->hzed;
01030     nx = thee->nx;
01031     ny = thee->ny;
01032     nz = thee->nz;
01033     xmin = thee->xmin;
01034     ymin = thee->ymin;
01035     zmin = thee->zmin;
01036
01037     if (pvec == VNULL) usepart = 0;
01038     else usepart = 1;
01039
01040     /* Set up the virtual socket */
01041     Vnm_print(0, "Vgrid_writeDX:  Opening virtual socket...\n");
01042     sock = Vio_ctor(iodev, iofmt, thost, fname, "w");
01043     if (sock == VNULL) {
01044         Vnm_print(2, "Vgrid_writeDX:  Problem opening virtual socket %s\n",
01045             fname);
01046         return;
01047     }
01048     if (Vio_connect(sock, 0) < 0) {
01049         Vnm_print(2, "Vgrid_writeDX:  Problem connecting virtual socket %s\n",
01050             fname);
01051         return;
01052     }
01053
01054     Vio_setWhiteChars(sock, MCwhiteChars);
01055     Vio_setCommChars(sock, MCcommChars);
01056
01057     Vnm_print(0, "Vgrid_writeDX:  Writing to virtual socket...\n");
01058
01059     if (usepart) {
01060         /* Get the lower corner and number of grid points for the local
01061          * partition */
01062         xminPART = VLARGE;
01063         yminPART = VLARGE;
01064         zminPART = VLARGE;
01065         nxPART = 0;
01066         nyPART = 0;
01067         nzPART = 0;
01068         /* First, search for the lower corner */
01069         for (k=0; k<nz; k++) {
01070             z = k*hzed + zmin;

```

```

01071         for (j=0; j<ny; j++) {
01072             y = j*hy + ymin;
01073             for (i=0; i<nx; i++) {
01074                 x = i*hx + xmin;
01075                 if (pvec[IJK(i,j,k)] > 0.0) {
01076                     if (x < xminPART) xminPART = x;
01077                     if (y < yminPART) yminPART = y;
01078                     if (z < zminPART) zminPART = z;
01079                 }
01080             }
01081         }
01082     }
01083     /* Now search for the number of grid points in the z direction */
01084     for (k=0; k<nz; k++) {
01085         gotit = 0;
01086         for (j=0; j<ny; j++) {
01087             for (i=0; i<nx; i++) {
01088                 if (pvec[IJK(i,j,k)] > 0.0) {
01089                     gotit = 1;
01090                     break;
01091                 }
01092             }
01093             if (gotit) break;
01094         }
01095         if (gotit) nzPART++;
01096     }
01097     /* Now search for the number of grid points in the y direction */
01098     for (j=0; j<ny; j++) {
01099         gotit = 0;
01100         for (k=0; k<nz; k++) {
01101             for (i=0; i<nx; i++) {
01102                 if (pvec[IJK(i,j,k)] > 0.0) {
01103                     gotit = 1;
01104                     break;
01105                 }
01106             }
01107             if (gotit) break;
01108         }
01109         if (gotit) nyPART++;
01110     }
01111     /* Now search for the number of grid points in the x direction */
01112     for (i=0; i<nx; i++) {
01113         gotit = 0;
01114         for (k=0; k<nz; k++) {
01115             for (j=0; j<ny; j++) {
01116                 if (pvec[IJK(i,j,k)] > 0.0) {
01117                     gotit = 1;
01118                     break;
01119                 }
01120             }
01121             if (gotit) break;
01122         }
01123         if (gotit) nxPART++;
01124     }
01125
01126     if ((nxPART != nx) || (nyPART != ny) || (nzPART != nz)) {
01127         Vnm_print(0, "Vgrid_wroteDX: printing only subset of domain\n");
01128     }
01129
01130     /* Write off the title (if we're not XDR) */
01131     if (Vstring_strcasecmp(iofmt, "XDR") == 0) {
01132         Vnm_print(0, "Vgrid_wroteDX: Skipping comments for XDR format.\n");
01133     } else {
01134         Vnm_print(0, "Vgrid_wroteDX: Writing comments for %s format.\n",
01135             iofmt);
01136         Vio_printf(sock, "# Data from %s\n", PACKAGE_STRING);
01137         Vio_printf(sock, "# \n");
01138         Vio_printf(sock, "# %s\n", title);
01139         Vio_printf(sock, "# \n");
01140     }
01141
01142     /* Write off the DX regular positions */
01143     Vio_printf(sock, "object 1 class gridpositions counts %d %d %d\n",
01144         nxPART, nyPART, nzPART);
01145
01146     sprintf(precFormat, Vprecision, xminPART, yminPART, zminPART);
01147     Vio_printf(sock, "origin %s\n", precFormat);
01148     sprintf(precFormat, Vprecision, hx, 0.0, 0.0);
01149     Vio_printf(sock, "delta %s\n", precFormat);
01150     sprintf(precFormat, Vprecision, 0.0, hy, 0.0);
01151     Vio_printf(sock, "delta %s\n", precFormat);

```

```

01152     Vio_printf(sock, "delta %s\n", precFormat);
01153     sprintf(precFormat, Vprecision, 0.0, 0.0, hzed);
01154     Vio_printf(sock, "delta %s\n", precFormat);
01155
01156     /* Write off the DX regular connections */
01157     Vio_printf(sock, "object 2 class gridconnections counts %d %d %d\n",
01158         nxPART, nyPART, nzPART);
01159
01160     /* Write off the DX data */
01161     Vio_printf(sock, "object 3 class array type double rank 0 items %lu \
01162 data follows\n", (nxPART*nyPART*nzPART));
01163     icol = 0;
01164     for (i=0; i<nx; i++) {
01165         for (j=0; j<ny; j++) {
01166             for (k=0; k<nz; k++) {
01167                 u = k*(nx)*(ny)+j*(nx)+i;
01168                 if (pvec[u] > 0.0) {
01169                     Vio_printf(sock, "%12.6e ", thee->data[u]);
01170                     icol++;
01171                     if (icol == 3) {
01172                         icol = 0;
01173                         Vio_printf(sock, "\n");
01174                     }
01175                 }
01176             }
01177         }
01178     }
01179
01180     if (icol != 0) Vio_printf(sock, "\n");
01181
01182     /* Create the field */
01183     Vio_printf(sock, "attribute \"dep\" string \"positions\"\\n");
01184     Vio_printf(sock, "object \"regular positions regular connections\" \
01185 class field\\n");
01186     Vio_printf(sock, "component \"positions\" value 1\\n");
01187     Vio_printf(sock, "component \"connections\" value 2\\n");
01188     Vio_printf(sock, "component \"data\" value 3\\n");
01189
01190 } else {
01191     /* Write off the title (if we're not XDR) */
01192     if (Vstring_strcasecmp(iofmt, "XDR") == 0) {
01193         Vnm_print(0, "Vgrid_writeDX: Skipping comments for XDR format.\\n");
01194     } else {
01195         Vnm_print(0, "Vgrid_writeDX: Writing comments for %s format.\\n",
01196             iofmt);
01197         Vio_printf(sock, "# Data from %s\\n", PACKAGE_STRING);
01198         Vio_printf(sock, "# \\n");
01199         Vio_printf(sock, "# %s\\n", title);
01200         Vio_printf(sock, "# \\n");
01201     }
01202
01203     /* Write off the DX regular positions */
01204     Vio_printf(sock, "object 1 class gridpositions counts %d %d %d\\n",
01205         nx, ny, nz);
01206
01207     sprintf(precFormat, Vprecision, xmin, ymin, zmin);
01208     Vio_printf(sock, "origin %s\\n", precFormat);
01209     sprintf(precFormat, Vprecision, hx, 0.0, 0.0);
01210     Vio_printf(sock, "delta %s\\n", precFormat);
01211     sprintf(precFormat, Vprecision, 0.0, hy, 0.0);
01212     Vio_printf(sock, "delta %s\\n", precFormat);
01213     sprintf(precFormat, Vprecision, 0.0, 0.0, hzed);
01214     Vio_printf(sock, "delta %s\\n", precFormat);
01215
01216     /* Write off the DX regular connections */
01217     Vio_printf(sock, "object 2 class gridconnections counts %d %d %d\\n",
01218         nx, ny, nz);
01219
01220     /* Write off the DX data */
01221     Vio_printf(sock, "object 3 class array type double rank 0 items %lu \
01222 data follows\\n", (nx*ny*nz));
01223     icol = 0;
01224     for (i=0; i<nx; i++) {
01225         for (j=0; j<ny; j++) {
01226             for (k=0; k<nz; k++) {
01227                 u = k*(nx)*(ny)+j*(nx)+i;
01228                 Vio_printf(sock, "%12.6e ", thee->data[u]);
01229                 icol++;
01230                 if (icol == 3) {
01231                     icol = 0;
01232

```

```

01233             Vio_printf(sock, "\n");
01234         }
01235     }
01236 }
01237 }
01238 if (icol != 0) Vio_printf(sock, "\n");
01239
01240 /* Create the field */
01241 Vio_printf(sock, "attribute \"dep\" string \"positions\\n\\n\"");
01242 Vio_printf(sock, "object \"regular positions regular connections\\n \"
01243 class field\\n\"");
01244 Vio_printf(sock, "component \"positions\\n\" value 1\\n");
01245 Vio_printf(sock, "component \"connections\\n\" value 2\\n");
01246 Vio_printf(sock, "component \"data\\n\" value 3\\n");
01247 }
01248
01249 /* Close off the socket */
01250 Vio_connectFree(sock);
01251 Vio_dtor(&sock);
01252 }
01253
01254 /* ////////////////////////////////////////
01255 // Routine: Vgrid_writeUHBD
01256 // Author:  Nathan Baker
01257 VPUBLIC void Vgrid_writeUHBD(Vgrid *thee, const char *iodev, const char *iofmt,
01258 const char *thost, const char *fname, char *title, double *pvec) {
01259
01260     size_t u, icol, i, j, k;
01261     size_t gotit, nx, ny, nz;
01262     double xmin, ymin, zmin, hzed, hy, hx;
01263     Vio *sock;
01264
01265     if (thee == VNULL) {
01266         Vnm_print(2, "Vgrid_writeUHBD: Error -- got VNULL thee!\\n");
01267         VASSERT(0);
01268     }
01269     if (!(thee->ctordata || thee->readdata)) {
01270         Vnm_print(2, "Vgrid_writeUHBD: Error -- no data available!\\n");
01271         VASSERT(0);
01272     }
01273
01274     if ((thee->hx!=thee->hy) || (thee->hy!=thee->hzed)
01275         || (thee->hx!=thee->hzed)) {
01276         Vnm_print(2, "Vgrid_writeUHBD: can't write UHBD mesh with non-uniform \\
01277 spacing\\n");
01278         return;
01279     }
01280
01281     /* Set up the virtual socket */
01282     sock = Vio_ctor(iodev, iofmt, thost, fname, "w");
01283     if (sock == VNULL) {
01284         Vnm_print(2, "Vgrid_writeUHBD: Problem opening virtual socket %s\\n",
01285             fname);
01286         return;
01287     }
01288     if (Vio_connect(sock, 0) < 0) {
01289         Vnm_print(2, "Vgrid_writeUHBD: Problem connecting virtual socket %s\\n",
01290             fname);
01291         return;
01292     }
01293
01294     /* Get the lower corner and number of grid points for the local
01295      * partition */
01296     hx = thee->hx;
01297     hy = thee->hy;
01298     hzed = thee->hzed;
01299     nx = thee->nx;
01300     ny = thee->ny;
01301     nz = thee->nz;
01302     xmin = thee->xmin;
01303     ymin = thee->ymin;
01304     zmin = thee->zmin;
01305
01306     /* Let interested folks know that partition information is ignored */
01307     if (pvec != VNULL) {
01308         gotit = 0;
01309         for (i=0; i<(nx*ny*nz); i++) {
01310             if (pvec[i] == 0) {
01311                 gotit = 1;
01312                 break;
01313             }
01314         }

```

```

01315     }
01316     if (gotit) {
01317         Vnm_print(2, "Vgrid_writeUHBD:  IGNORING PARTITION INFORMATION!\n");
01318         Vnm_print(2, "Vgrid_writeUHBD:  This means I/O from parallel runs \
01319 will have significant overlap.\n");
01320     }
01321 }
01322
01323 /* Write out the header */
01324 Vio_printf(sock, "%72s\n", title);
01325 Vio_printf(sock, "%12.5e%12.5e%7d%7d%7d%7d\n", 1.0, 0.0, -1, 0,
01326 nz, 1, nz);
01327 Vio_printf(sock, "%7d%7d%7d%12.5e%12.5e%12.5e%12.5e\n", nx, ny, nz,
01328 hx, (xmin-hx), (ymin-hx), (zmin-hx));
01329 Vio_printf(sock, "%12.5e%12.5e%12.5e%12.5e\n", 0.0, 0.0, 0.0, 0.0);
01330 Vio_printf(sock, "%12.5e%12.5e%7d%7d", 0.0, 0.0, 0, 0);
01331
01332 /* Write out the entries */
01333 icol = 0;
01334 for (k=0; k<nz; k++) {
01335     Vio_printf(sock, "\n%7d%7d%7d\n", k+1, thee->nx, thee->ny);
01336     icol = 0;
01337     for (j=0; j<ny; j++) {
01338         for (i=0; i<nx; i++) {
01339             u = k*(nx)*(ny)+j*(nx)+i;
01340             icol++;
01341             Vio_printf(sock, " %12.5e", thee->data[u]);
01342             if (icol == 6) {
01343                 icol = 0;
01344                 Vio_printf(sock, "\n");
01345             }
01346         }
01347     }
01348 }
01349 if (icol != 0) Vio_printf(sock, "\n");
01350
01351 /* Close off the socket */
01352 Vio_connectFree(sock);
01353 Vio_dtor(&sock);
01354 }
01355
01356 VPUBLIC double Vgrid_integrate(Vgrid *thee) {
01357     size_t i, j, k;
01358     int nx, ny, nz;
01359     double sum, w;
01360
01361     if (thee == VNULL) {
01362         Vnm_print(2, "Vgrid_integrate:  Got VNULL thee!\n");
01363         VASSERT(0);
01364     }
01365
01366     nx = thee->nx;
01367     ny = thee->ny;
01368     nz = thee->nz;
01369
01370     sum = 0.0;
01371
01372     for (k=0; k<nz; k++) {
01373         w = 1.0;
01374         if ((k==0) || (k==(nz-1))) w = w * 0.5;
01375         for (j=0; j<ny; j++) {
01376             w = 1.0;
01377             if ((j==0) || (j==(ny-1))) w = w * 0.5;
01378             for (i=0; i<nx; i++) {
01379                 w = 1.0;
01380                 if ((i==0) || (i==(nx-1))) w = w * 0.5;
01381                 sum = sum + w*(thee->data[IJK(i,j,k)]);
01382             }
01383         }
01384     }
01385
01386     sum = sum*(thee->hx)*(thee->hy)*(thee->hz);
01387
01388     return sum;
01389
01390 }
01391
01392
01393
01394 VPUBLIC double Vgrid_normL1(Vgrid *thee) {
01395

```

```

01396     size_t i, j, k;
01397     int nx, ny, nz;
01398     double sum;
01399
01400     if (thee == VNULL) {
01401         Vnm_print(2, "Vgrid_normL1: Got VNULL thee!\n");
01402         VASSERT(0);
01403     }
01404
01405     nx = thee->nx;
01406     ny = thee->ny;
01407     nz = thee->nz;
01408
01409     sum = 0.0;
01410     for (k=0; k<nz; k++) {
01411         for (j=0; j<ny; j++) {
01412             for (i=0; i<nx; i++) {
01413                 sum = sum + VABS(thee->data[IJK(i,j,k)]);
01414             }
01415         }
01416     }
01417
01418     sum = sum*(thee->hx)*(thee->hy)*(thee->hz);
01419
01420     return sum;
01421 }
01422
01423
01424 VPUBLIC double Vgrid_normL2(Vgrid *thee) {
01425
01426     size_t i, j, k;
01427     int nx, ny, nz;
01428     double sum;
01429
01430     if (thee == VNULL) {
01431         Vnm_print(2, "Vgrid_normL2: Got VNULL thee!\n");
01432         VASSERT(0);
01433     }
01434
01435     nx = thee->nx;
01436     ny = thee->ny;
01437     nz = thee->nz;
01438
01439     sum = 0.0;
01440     for (k=0; k<nz; k++) {
01441         for (j=0; j<ny; j++) {
01442             for (i=0; i<nx; i++) {
01443                 sum = sum + VSQR(thee->data[IJK(i,j,k)]);
01444             }
01445         }
01446     }
01447
01448     sum = sum*(thee->hx)*(thee->hy)*(thee->hz);
01449
01450     return VSQRT(sum);
01451 }
01452
01453
01454 VPUBLIC double Vgrid_seminormH1(Vgrid *thee) {
01455
01456     size_t i, j, k;
01457     int nx, ny, nz, d;
01458     double pt[3], grad[3], sum, hx, hy, hzed, xmin, ymin, zmin;
01459
01460     if (thee == VNULL) {
01461         Vnm_print(2, "Vgrid_seminormH1: Got VNULL thee!\n");
01462         VASSERT(0);
01463     }
01464
01465     nx = thee->nx;
01466     ny = thee->ny;
01467     nz = thee->nz;
01468     hx = thee->hx;
01469     hy = thee->hy;
01470     hzed = thee->hz;
01471     xmin = thee->xmin;
01472     ymin = thee->ymin;
01473     zmin = thee->zmin;
01474
01475     sum = 0.0;
01476     for (k=0; k<nz; k++) {

```

```

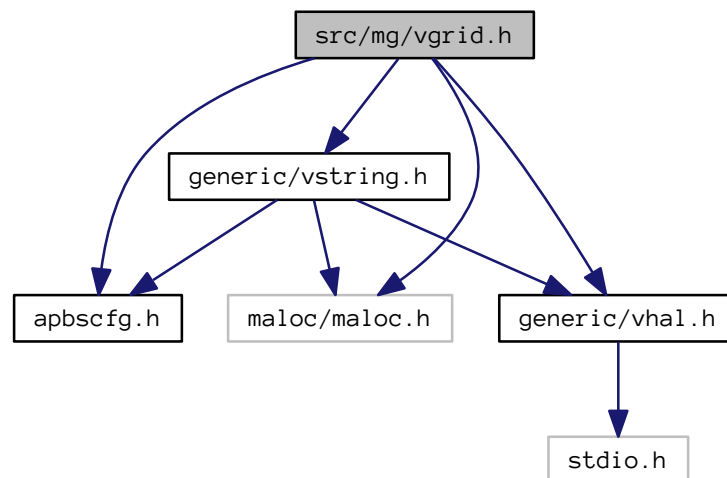
01477         pt[2] = k*hzed + zmin;
01478         for (j=0; j<ny; j++) {
01479             pt[1] = j*hy + ymin;
01480             for (i=0; i<nx; i++) {
01481                 pt[0] = i*hx + xmin;
01482                 VASSERT(Vgrid_gradient(thee, pt, grad));
01483                 for (d=0; d<3; d++) sum = sum + VSQR(grad[d]);
01484             }
01485         }
01486     }
01487
01488     sum = sum*(hx)*(hy)*(hzed);
01489
01490     if (VABS(sum) < VSMALL) sum = 0.0;
01491     else sum = VSQRT(sum);
01492
01493     return sum;
01494 }
01495 }
01496
01497 VPUBLIC double Vgrid_normH1(Vgrid *thee) {
01498     double sum = 0.0;
01499
01500     if (thee == VNULL) {
01501         Vnm_print(2, "Vgrid_normH1: Got VNULL thee!\n");
01502         VASSERT(0);
01503     }
01504
01505     sum = VSQR(Vgrid_seminormH1(thee)) + VSQR(Vgrid_normL2(thee));
01506
01507     return VSQRT(sum);
01508 }
01509
01510 }
01511
01512 VPUBLIC double Vgrid_normLinf(Vgrid *thee) {
01513     size_t i, j, k;
01514     int nx, ny, nz, gotval;
01515     double sum, val;
01516
01517     if (thee == VNULL) {
01518         Vnm_print(2, "Vgrid_normLinf: Got VNULL thee!\n");
01519         VASSERT(0);
01520     }
01521
01522     nx = thee->nx;
01523     ny = thee->ny;
01524     nz = thee->nz;
01525
01526     sum = 0.0;
01527     gotval = 0;
01528     for (k=0; k<nz; k++) {
01529         for (j=0; j<ny; j++) {
01530             for (i=0; i<nx; i++) {
01531                 val = VABS(thee->data[IJK(i,j,k)]);
01532                 if (!gotval) {
01533                     gotval = 1;
01534                     sum = val;
01535                 }
01536                 if (val > sum) sum = val;
01537             }
01538         }
01539     }
01540
01541     return sum;
01542 }
01543
01544 }
01545

```

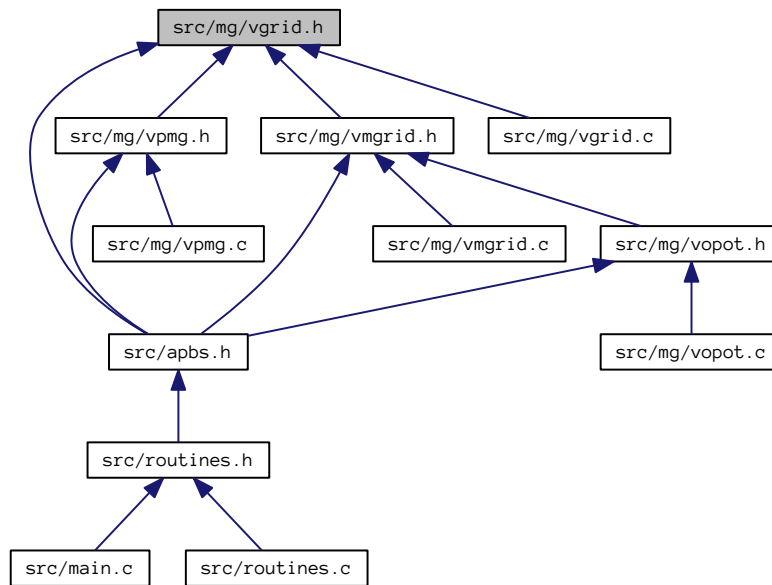
10.91 src/mg/vgrid.h File Reference

Potential oracle for Cartesian mesh data.


```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/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.

Macros

- #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)

FORTTRAN 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 *three`)

Get the H_1 norm (or energy norm) of the data. This returns the integral:

$$\|u\|_{H_1} = \left(\int_{\Omega} |\nabla u(x)|^2 dx + \int_{\Omega} |u(x)|^2 dx \right)^{1/2}$$

10.91.1 Detailed Description

Potential oracle for Cartesian mesh data.

Author

Nathan Baker and Steve Bond

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
```

```

* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [vgrid.h](#).

10.91.2 Function Documentation

10.91.2.1 `VEXTERNC void Vgrid_writeGZ (Vgrid * thee, const char * iodev, const char * iofmt, const char * thost, const char * fname, char * title, double * pvec)`

Write out OpenDX data in GZIP format.

Author

Dave Gohara

Parameters

<i>thee</i>	Object to hold new grid data
<i>iodev</i>	I/O device
<i>iofmt</i>	I/O format
<i>thost</i>	Remote host name
<i>fname</i>	File name
<i>title</i>	Data title
<i>pvec</i>	Masking vector (0 = not written)

Definition at line 812 of file [vgrid.c](#).

10.92 vgrid.h

```

00001
00062 #ifndef _VGRID_H_
00063 #define _VGRID_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "malloc/malloc.h"
00068
00069 #include "generic/vhal.h"
00070 #include "generic/vstring.h"
00071
00074 #define VGRID_DIGITS 6
00075
00081 struct sVgrid {
00082
00083     int nx;
00084     int ny;
00085     int nz;
00086     double hx;
00087     double hy;
00088     double hzed;
00089     double xmin;
00090     double ymin;
00091     double zmin;
00092     double xmax;
00093     double ymax;
00094     double zmax;
00095     double *data;
00096     int readdata;
00097     int ctordata;
```

```

00099     Vmem *mem;
00100 };
00101
00106 typedef struct sVgrid Vgrid;
00107
00108 #if !defined(VINLINE_VGRID)
00109
00117     VEXTERNC unsigned long int Vgrid_memChk(Vgrid *thee);
00118
00119 #else /* if defined(VINLINE_VGRID) */
00120
00128 #    define Vgrid_memChk(thee) (Vmem_bytes((thee)->vmem))
00129
00130 #endif /* if !defined(VINLINE_VPMG) */
00131
00149 VEXTERNC Vgrid* Vgrid_ctor(int nx, int ny, int nz,
00150                             double hx, double hy, double hzed,
00151                             double xmin, double ymin, double zmin,
00152                             double *data);
00153
00172 VEXTERNC int Vgrid_ctor2(Vgrid *thee, int nx, int ny, int nz,
00173                           double hx, double hy, double hzed,
00174                           double xmin, double ymin, double zmin,
00175                           double *data);
00176
00185 VEXTERNC int Vgrid_value(Vgrid *thee, double x[3], double *value);
00186
00192 VEXTERNC void Vgrid_dtor(Vgrid **thee);
00193
00199 VEXTERNC void Vgrid_dtor2(Vgrid *thee);
00200
00214 VEXTERNC int Vgrid_curvature(Vgrid *thee, double pt[3], int cflag,
00215                               double *curv);
00216
00225 VEXTERNC int Vgrid_gradient(Vgrid *thee, double pt[3], double grad[3] );
00226
00231 VEXTERNC int Vgrid_readGZ(
00232     Vgrid *thee,
00233     const char *fname
00234 );
00235
00239 VEXTERNC void Vgrid_writeGZ(
00240     Vgrid *thee,
00241     const char *iodev,
00242     const char *iofmt,
00243     const char *thost,
00244     const char *fname,
00245     char *title,
00246     double *pvec
00247 );
00248
00266 VEXTERNC void Vgrid_writeUHBD(Vgrid *thee, const char *iodev,
00267                                const char *iofmt, const char *thost, const char *fname, char *title,
00268                                double *pvec);
00269
00284 VEXTERNC void Vgrid_writeDX(Vgrid *thee, const char *iodev,
00285                                const char *iofmt, const char *thost, const char *fname, char *title,
00286                                double *pvec);
00287
00299 VEXTERNC int Vgrid_readDX(Vgrid *thee, const char *iodev, const char *iofmt,
00300                             const char *thost, const char *fname);
00301
00308 VEXTERNC double Vgrid_integrate(Vgrid *thee);
00309
00318 VEXTERNC double Vgrid_normL1(Vgrid *thee);
00319
00328 VEXTERNC double Vgrid_normL2(Vgrid *thee);
00329
00338 VEXTERNC double Vgrid_normLinf(Vgrid *thee);
00339
00349 VEXTERNC double Vgrid_seminormH1(Vgrid *thee);
00350
00361 VEXTERNC double Vgrid_normH1(Vgrid *thee);
00362
00363 #endif

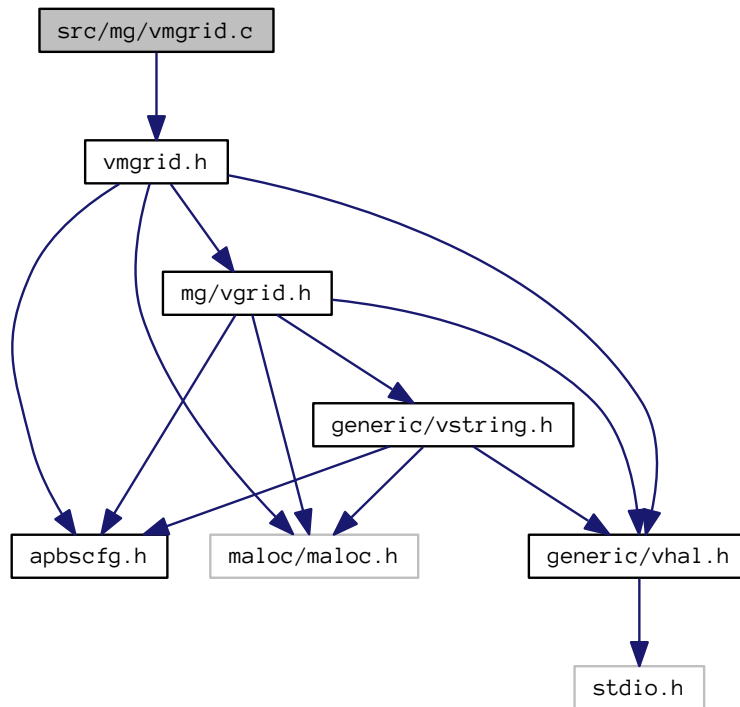
```

10.93 src/mg/vmgrid.c File Reference

Class Vmgrid methods.

```
#include "vmgrid.h"
```

Include dependency graph for vmgrid.c:



Functions

- `VPUBLIC Vmgrid * Vmgrid_ctor ()`
Construct Vmgrid object.
- `VPUBLIC int Vmgrid_ctor2 (Vmgrid *thee)`
Initialize Vmgrid object.
- `VPUBLIC void Vmgrid_dtor (Vmgrid **thee)`
Object destructor.
- `VPUBLIC void Vmgrid_dtor2 (Vmgrid *thee)`
FORTTRAN stub object destructor.
- `VPUBLIC int Vmgrid_value (Vmgrid *thee, double pt[3], double *value)`
Get potential value (from mesh or approximation) at a point.
- `VPUBLIC int Vmgrid_curvature (Vmgrid *thee, double pt[3], int cflag, double *value)`
Get second derivative values at a point.
- `VPUBLIC int Vmgrid_gradient (Vmgrid *thee, double pt[3], double grad[3])`

Get first derivative values at a point.

- `VPUBLIC int Vmgrid_addGrid (Vmgrid *thee, Vgrid *grid)`

Add a grid to the hierarchy.

10.93.1 Detailed Description

Class Vmgrid methods.

Author

Nathan Baker

Version

Id

[vmgrid.c](#) 1615 2010-10-20 19:16:35Z sobolevnm

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.
* 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.c](#).

10.94 vmgrid.c

```

00001
00049 #include "vmgrid.h"
00050
00051 VEMBED(rcsid="$Id: vmgrid.c 1615 2010-10-20 19:16:35Z sobolevnmr $")
00052
00053 /* ////////////////////////////////////////////////////////////////////
00054 // Routine: Vmgrid_ctor
00055 // Author: Nathan Baker
00057 VPUBLIC Vmgrid* Vmgrid_ctor() {
00058
00059     Vmgrid *thee = VNULL;
00060
00061     thee = Vmem_malloc(VNULL, 1, sizeof(Vmgrid));
00062     VASSERT(thee != VNULL);
00063     VASSERT(Vmgrid_ctor2(thee));
00064
00065     return thee;
00066 }
00067
00068 /* ////////////////////////////////////////////////////////////////////
00069 // Routine: Vmgrid_ctor2
00070 // Author: Nathan Baker
00072 VPUBLIC int Vmgrid_ctor2(Vmgrid *thee) {
00073
00074     int i;
00075
00076     if (thee == VNULL) return 0;
00077
00078     thee->ngrids = 0;
00079     for (i=0; i<VMGRIDMAX; i++) thee->grids[i] = VNULL;
00080
00081     return 1;
00082 }
00083
00084 /* ////////////////////////////////////////////////////////////////////
00085 // Routine: Vmgrid_dtor
00086 // Author: Nathan Baker
00088 VPUBLIC void Vmgrid_dtor(Vmgrid **thee) {
00089
00090     if ((*thee) != VNULL) {
00091         Vmgrid_dtor2(*thee);
00092         Vmem_free(VNULL, 1, sizeof(Vmgrid), (void **)thee);
00093         (*thee) = VNULL;
00094     }
00095 }
00096
00097 /* ////////////////////////////////////////////////////////////////////
00098 // Routine: Vmgrid_dtor2
00099 // Author: Nathan Baker
00101 VPUBLIC void Vmgrid_dtor2(Vmgrid *thee) { ; }
00102
00103 /* ////////////////////////////////////////////////////////////////////
00104 // Routine: Vmgrid_value
00105 // Author: Nathan Baker
00107 VPUBLIC int Vmgrid_value(Vmgrid *thee, double pt[3], double *value) {
00108
00109     int i, rc;
00110     double tvalue;
00111
00112     VASSERT(thee != VNULL);
00113
00114     for (i=0; i<thee->ngrids; i++) {
00115         rc = Vgrid_value(thee->grids[i], pt, &tvalue);
00116         if (rc) {
00117             *value = tvalue;
00118             return 1;
00119         }
00120     }
00121
00122     Vnm_print(2, "Vmgrid_value: Point (%g, %g, %g) not found in \
00123 hierarchy!\n", pt[0], pt[1], pt[2]);
00124
00125     return 0;
00126 }
00127
00128 /* ////////////////////////////////////////////////////////////////////
00129 // Routine: Vmgrid_curvature
00130 //

```

```

00131 // Notes: cflag=0 ==> Reduced Maximal Curvature
00132 //         cflag=1 ==> Mean Curvature (Laplace)
00133 //         cflag=2 ==> Gauss Curvature
00134 //         cflag=3 ==> True Maximal Curvature
00135 //
00136 // Authors: Nathan Baker
00138 VPUBLIC int Vmgrid_curvature(Vmgrid *thee, double pt[3], int cflag,
00139 double *value) {
00140
00141     int i, rc;
00142     double tvalue;
00143
00144     VASSERT(thee != VNULL);
00145
00146     for (i=0; i<thee->ngrids; i++) {
00147         rc = Vgrid_curvature(thee->grids[i], pt, cflag, &tvalue);
00148         if (rc) {
00149             *value = tvalue;
00150             return 1;
00151         }
00152     }
00153
00154     Vnm_print(2, "Vmgrid_curvature: Point (%g, %g, %g) not found in \
00155 hiearchy!\n", pt[0], pt[1], pt[2]);
00156
00157     return 0;
00158
00159 }
00160
00161 /* ////////////////////////////////////////
00162 // Routine: Vmgrid_gradient
00163 //
00164 // Authors: Nathan Baker
00166 VPUBLIC int Vmgrid_gradient(Vmgrid *thee, double pt[3], double grad[3]) {
00168
00169     int i, j, rc;
00170     double tgrad[3];
00171
00172     VASSERT(thee != VNULL);
00173
00174     for (i=0; i<thee->ngrids; i++) {
00175         rc = Vgrid_gradient(thee->grids[i], pt, tgrad);
00176         if (rc) {
00177             for (j=0; j<3; j++) grad[j] = tgrad[j];
00178             return 1;
00179         }
00180     }
00181
00182     Vnm_print(2, "Vmgrid_gradient: Point (%g, %g, %g) not found in \
00183 hiearchy!\n", pt[0], pt[1], pt[2]);
00184
00185     return 0;
00186
00187 }
00188
00189 /* ////////////////////////////////////////
00191 // Routine: Vmgrid_addGrid
00192 //
00193 // Authors: Nathan Baker
00195 VPUBLIC int Vmgrid_addGrid(Vmgrid *thee, Vgrid *grid) {
00196
00197     int i, j, rc;
00198     double tgrad[3];
00199
00200     VASSERT(thee != VNULL);
00201
00202     if (grid == VNULL) {
00203         Vnm_print(2, "Vmgrid_addGrid: Not adding VNULL grid!\n");
00204         return 0;
00205     }
00206
00207     if (thee->ngrids >= VMGRIDMAX) {
00208         Vnm_print(2, "Vmgrid_addGrid: Too many grids in hierarchy (max = \
00209 %d)!\n", VMGRIDMAX);
00210         Vnm_print(2, "Vmgrid_addGrid: Not adding grid!\n");
00211         return 0;
00212     }
00213
00214     thee->grids[thee->ngrids] = grid;

```

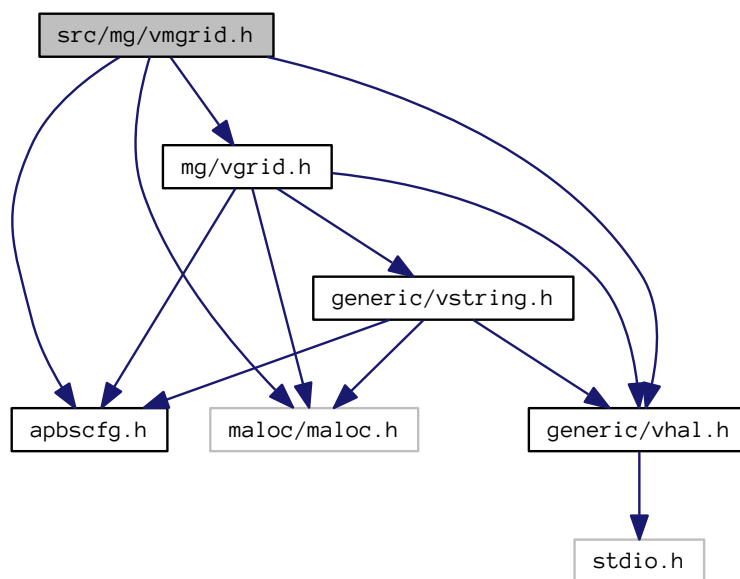
```
00215     (thee->ngrids)++;  
00216  
00217     return 1;  
00218  
00219 }
```

10.95 src/mg/vmgrid.h File Reference

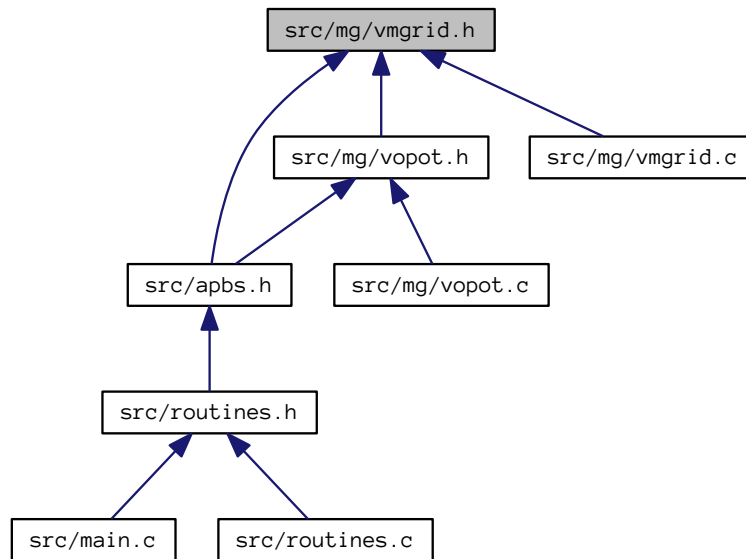
Multiresolution oracle for Cartesian mesh data.

```
#include "apbscfg.h"  
#include "malloc/malloc.h"  
#include "generic/vhal.h"  
#include "mg/vgrid.h"
```

Include dependency graph for vmgrid.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [sVmgrid](#)
Multiresolution oracle for Cartesian mesh data.

Macros

- `#define` [VMGRIDMAX](#) 20
The maximum number of levels in the grid hierarchy.

Typedefs

- typedef struct [sVmgrid](#) [Vmgrid](#)
Declaration of the Vmgrid class as the Vmgrid 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`)
FORTTRAN stub object destructor.
- VEXTERNC int `Vmgrid_addGrid` (`Vmgrid *thee`, `Vgrid *grid`)
Add a grid to the hierarchy.
- VEXTERNC int `Vmgrid_curvature` (`Vmgrid *thee`, double `pt[3]`, int `cflag`, double `*curv`)
Get second derivative values at a point.
- VEXTERNC int `Vmgrid_gradient` (`Vmgrid *thee`, double `pt[3]`, double `grad[3]`)
Get first derivative values at a point.
- VEXTERNC `Vgrid *` `Vmgrid_getGridByNum` (`Vmgrid *thee`, int `num`)
Get specific grid in hierarchy.
- VEXTERNC `Vgrid *` `Vmgrid_getGridByPoint` (`Vmgrid *thee`, double `pt[3]`)
Get grid in hierarchy which contains specified point or VNULL.

10.95.1 Detailed Description

Multiresolution oracle for Cartesian mesh data.

Author

Nathan Baker

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
```

```

* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vmgrid.h](#).

10.96 vmgrid.h

```

00001
00062 #ifndef _VMGRID_H_
00063 #define _VMGRID_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "malloc/malloc.h"
00068
00069 #include "generic/vhal.h"
00070 #include "mg/vgrid.h"
00071
00076 #define VMGRIDMAX 20
00077
00078
00084 struct sVmgrid {
00085     int ngrids;
00086     Vgrid *grids[VMGRIDMAX];
00092 };
00093
00098 typedef struct sVmgrid Vmgrid;
00099
00105 VEXTERNC Vmgrid* Vmgrid_ctor();
00106
00113 VEXTERNC int Vmgrid_ctor2(Vmgrid *thee);
00114
00123 VEXTERNC int Vmgrid_value(Vmgrid *thee, double x[3], double *value);
00124
00130 VEXTERNC void Vmgrid_dtor(Vmgrid **thee);
00131
00137 VEXTERNC void Vmgrid_dtor2(Vmgrid *thee);
00138
00151 VEXTERNC int Vmgrid_addGrid(Vmgrid *thee, Vgrid *grid);
00152
00153
00167 VEXTERNC int Vmgrid_curvature(Vmgrid *thee, double pt[3], int cflag,
00168     double *curv);
00169
00178 VEXTERNC int Vmgrid_gradient(Vmgrid *thee, double pt[3], double grad[3] );
00179
00187 VEXTERNC Vgrid* Vmgrid_getGridByNum(Vmgrid *thee, int num);
00188
00196 VEXTERNC Vgrid* Vmgrid_getGridByPoint(Vmgrid *thee, double pt[3]);
00197
00198 #endif
00199

```


Version**\$Id\$****Attention**

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vopot.c](#).**10.98 vopot.c**

```

00001
00057 #include "vopot.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 /* ////////////////////////////////////////
00062 // Routine:  Vopot_ctor
00063 // Author:   Nathan Baker

```



```

00065 VPUBLIC Vopot* Vopot_ctor(Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl) {
00066
00067     Vopot *thee = VNULL;
00068
00069     thee = Vmem_malloc(VNULL, 1, sizeof(Vopot));
00070     VASSERT(thee != VNULL);
00071     VASSERT(Vopot_ctor2(thee, mgrid, pbe, bcfl));
00072
00073     return thee;
00074 }
00075
00076 /* ////////////////////////////////////////////////////////////////////
00077 // Routine: Vopot_ctor2
00078 // Author:  Nathan Baker
00080 VPUBLIC int Vopot_ctor2(Vopot *thee, Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl) {
00081
00082     if (thee == VNULL) return 0;
00083     thee->bcfl = bcfl;
00084     thee->mgrid = mgrid;
00085     thee->pbe = pbe;
00086
00087     return 1;
00088 }
00089
00090 /* ////////////////////////////////////////////////////////////////////
00091 // Routine: Vopot_dtor
00092 // Author:  Nathan Baker
00094 VPUBLIC void Vopot_dtor(Vopot **thee) {
00095
00096     if ((*thee) != VNULL) {
00097         Vopot_dtor2(*thee);
00098         Vmem_free(VNULL, 1, sizeof(Vopot), (void **)thee);
00099         (*thee) = VNULL;
00100     }
00101 }
00102
00103 /* ////////////////////////////////////////////////////////////////////
00104 // Routine: Vopot_dtor2
00105 // Author:  Nathan Baker
00107 VPUBLIC void Vopot_dtor2(Vopot *thee) { return; }
00108
00109 /* ////////////////////////////////////////////////////////////////////
00110 // Routine: Vopot_pot
00111 // Author:  Nathan Baker
00113 #define IJK(i,j,k)  (((k)*(nx)*(ny))+((j)*(nx))+i))
00114 VPUBLIC int Vopot_pot(Vopot *thee, double pt[3], double *value) {
00115
00116     Vatom *atom;
00117     int i, iatom;
00118     double u, T, charge, eps_w, xkappa, dist, size, val, *position;
00119     Valist *alist;
00120
00121     VASSERT(thee != VNULL);
00122
00123     eps_w = Vpbe_getSolventDiel(thee->pbe);
00124     xkappa = (1.0e10)*Vpbe_getXkappa(thee->pbe);
00125     T = Vpbe_getTemperature(thee->pbe);
00126     alist = Vpbe_getValist(thee->pbe);
00127
00128     u = 0;
00129
00130     /* See if we're on the mesh */
00131     if (Vmgrid_value(thee->mgrid, pt, &u)) {
00132
00133         *value = u;
00134
00135     } else {
00136
00137         switch (thee->bcfl) {
00138
00139             case BCFL_ZERO:
00140                 u = 0;
00141                 break;
00142
00143             case BCFL_SDH:
00144                 size = (1.0e-10)*Vpbe_getSoluteRadius(thee->pbe);
00145                 position = Vpbe_getSoluteCenter(thee->pbe);
00146                 charge = Vunit_ec*Vpbe_getSoluteCharge(thee->pbe);
00147                 dist = 0;
00148                 for (i=0; i<3; i++)
00149                     dist += VSQR(position[i] - pt[i]);

```

```

00150         dist = (1.0e-10)*VSQRT(dist);
00151         val = (charge)/(4*VPI*Vunit_eps0*eps_w*dist);
00152         if (xkappa != 0.0)
00153             val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00154         val = val*Vunit_ec/(Vunit_kb*T);
00155         u = val;
00156         break;
00157
00158     case BCFL_MDH:
00159         u = 0;
00160         for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
00161             atom = Valist_getAtom(alist, iatom);
00162             position = Vatom_getPosition(atom);
00163             charge = Vunit_ec*Vatom_getCharge(atom);
00164             size = (1e-10)*Vatom_getRadius(atom);
00165             dist = 0;
00166             for (i=0; i<3; i++)
00167                 dist += VSQR(position[i] - pt[i]);
00168             dist = (1.0e-10)*VSQRT(dist);
00169             val = (charge)/(4*VPI*Vunit_eps0*eps_w*dist);
00170             if (xkappa != 0.0)
00171                 val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00172             val = val*Vunit_ec/(Vunit_kb*T);
00173             u = u + val;
00174         }
00175         break;
00176
00177     case BCFL_UNUSED:
00178         Vnm_print(2, "Vopot_pot: Invalid bcfl flag (%d)!\n",
00179             thee->bcfl);
00180         return 0;
00181
00182     case BCFL_FOCUS:
00183         Vnm_print(2, "Vopot_pot: Invalid bcfl flag (%d)!\n",
00184             thee->bcfl);
00185         return 0;
00186
00187     default:
00188         Vnm_print(2, "Vopot_pot: Bogus thee->bcfl flag (%d)!\n",
00189             thee->bcfl);
00190         return 0;
00191         break;
00192 }
00193
00194 *value = u;
00195 }
00196 }
00197
00198 return 1;
00199
00200 }
00201
00202 /* ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00203 // Routine: Vopot_curvature
00204 //
00205 // Notes: cflag=0 ==> Reduced Maximal Curvature
00206 //        cflag=1 ==> Mean Curvature (Laplace)
00207 //        cflag=2 ==> Gauss Curvature
00208 //        cflag=3 ==> True Maximal Curvature
00209 // If we are off the grid, we can still evaluate the Laplacian; assuming, we
00210 // are away from the molecular surface, it is simply equal to the DH factor.
00211 //
00212 // Authors: Nathan Baker
00213
00214 VPUBLIC int Vopot_curvature(Vopot *thee, double pt[3], int cflag,
00215     double *value) {
00216
00217     Vatom *atom;
00218     int i, iatom;
00219     double u, T, charge, eps_w, xkappa, dist, size, val, *position, zkappa2;
00220     Valist *alist;
00221
00222     VASSERT(thee != VNULL);
00223
00224     eps_w = Vpbe_getSolventDiel(thee->pbe);
00225     xkappa = (1.0e10)*Vpbe_getXkappa(thee->pbe);
00226     zkappa2 = Vpbe_getZkappa2(thee->pbe);
00227     T = Vpbe_getTemperature(thee->pbe);
00228     alist = Vpbe_getValist(thee->pbe);
00229
00230     u = 0;
00231

```

```

00232     if (Vmgrid_curvature(thee->mgrid, pt, cflag, value)) return 1;
00233     else if (cflag != 1) {
00234         Vnm_print(2, "Vopot_curvature: Off mesh!\n");
00235         return 1;
00236     } else {
00237
00238         switch (thee->bcfl) {
00239
00240             case BCFL_ZERO:
00241                 u = 0;
00242                 break;
00243
00244             case BCFL_SDH:
00245                 size = (1.0e-10)*Vpbe_getSoluteRadius(thee->pbe);
00246                 position = Vpbe_getSoluteCenter(thee->pbe);
00247                 charge = Vunit_ec*Vpbe_getSoluteCharge(thee->pbe);
00248                 dist = 0;
00249                 for (i=0; i<3; i++)
00250                     dist += VSQR(position[i] - pt[i]);
00251                 dist = (1.0e-10)*VSQRT(dist);
00252                 if (xkappa != 0.0)
00253                     u = zkappa2*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00254                 break;
00255
00256             case BCFL_MDH:
00257                 u = 0;
00258                 for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
00259                     atom = Valist_getAtom(alist, iatom);
00260                     position = Vatom_getPosition(atom);
00261                     charge = Vunit_ec*Vatom_getCharge(atom);
00262                     size = (1e-10)*Vatom_getRadius(atom);
00263                     dist = 0;
00264                     for (i=0; i<3; i++)
00265                         dist += VSQR(position[i] - pt[i]);
00266                     dist = (1.0e-10)*VSQRT(dist);
00267                     if (xkappa != 0.0)
00268                         val = zkappa2*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00269                     u = u + val;
00270                 }
00271                 break;
00272
00273             case BCFL_UNUSED:
00274                 Vnm_print(2, "Vopot_pot: Invalid bcfl (%d)!\n", thee->bcfl);
00275                 return 0;
00276
00277             case BCFL_FOCUS:
00278                 Vnm_print(2, "Vopot_pot: Invalid bcfl (%d)!\n", thee->bcfl);
00279                 return 0;
00280
00281             default:
00282                 Vnm_print(2, "Vopot_pot: Bogus thee->bcfl flag (%d)!\n",
00283                     thee->bcfl);
00284                 return 0;
00285                 break;
00286         }
00287
00288         *value = u;
00289     }
00290
00291     return 1;
00292 }
00293
00294
00295 /* ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
00296 // Routine: Vopot_gradient
00297 //
00298 // Authors: Nathan Baker
00300 VPUBLIC int Vopot_gradient(Vopot *thee, double pt[3], double grad[3]) {
00301
00302     Vatom *atom;
00303     int iatom;
00304     double T, charge, eps_w, xkappa, size, val, *position;
00305     double dx, dy, dz, dist;
00306     Valist *alist;
00307
00308     VASSERT(thee != VNULL);
00309
00310     eps_w = Vpbe_getSolventDiel(thee->pbe);
00311     xkappa = (1.0e10)*Vpbe_getXkappa(thee->pbe);
00312     T = Vpbe_getTemperature(thee->pbe);
00313     alist = Vpbe_getValist(thee->pbe);

```

```

00314
00315
00316     if (!Vmgrid_gradient(thee->mgrid, pt, grad)) {
00317
00318         switch (thee->bcfl) {
00319
00320             case BCFL_ZERO:
00321                 grad[0] = 0.0;
00322                 grad[1] = 0.0;
00323                 grad[2] = 0.0;
00324                 break;
00325
00326             case BCFL_SDH:
00327                 grad[0] = 0.0;
00328                 grad[1] = 0.0;
00329                 grad[2] = 0.0;
00330                 size = (1.0e-10)*Vpbe_getSoluteRadius(thee->pbe);
00331                 position = Vpbe_getSoluteCenter(thee->pbe);
00332                 charge = Vunit_ec*Vpbe_getSoluteCharge(thee->pbe);
00333                 dx = position[0] - pt[0];
00334                 dy = position[1] - pt[1];
00335                 dz = position[2] - pt[2];
00336                 dist = VSQR(dx) + VSQR(dy) + VSQR(dz);
00337                 dist = (1.0e-10)*VSQRT(dist);
00338                 val = (charge)/(4*VPI*Vunit_eps0*eps_w);
00339                 if (xkappa != 0.0)
00340                     val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00341                 val = val*Vunit_ec/(Vunit_kb*T);
00342                 grad[0] = val*dx/dist*(-1.0/dist/dist + xkappa/dist);
00343                 grad[1] = val*dy/dist*(-1.0/dist/dist + xkappa/dist);
00344                 grad[2] = val*dz/dist*(-1.0/dist/dist + xkappa/dist);
00345                 break;
00346
00347             case BCFL_MDH:
00348                 grad[0] = 0.0;
00349                 grad[1] = 0.0;
00350                 grad[2] = 0.0;
00351                 for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
00352                     atom = Valist_getAtom(alist, iatom);
00353                     position = Vatom_getPosition(atom);
00354                     charge = Vunit_ec*Vatom_getCharge(atom);
00355                     size = (1e-10)*Vatom_getRadius(atom);
00356                     dx = position[0] - pt[0];
00357                     dy = position[1] - pt[1];
00358                     dz = position[2] - pt[2];
00359                     dist = VSQR(dx) + VSQR(dy) + VSQR(dz);
00360                     dist = (1.0e-10)*VSQRT(dist);
00361                     val = (charge)/(4*VPI*Vunit_eps0*eps_w);
00362                     if (xkappa != 0.0)
00363                         val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00364                     val = val*Vunit_ec/(Vunit_kb*T);
00365                     grad[0] += (val*dx/dist*(-1.0/dist/dist + xkappa/dist));
00366                     grad[1] += (val*dy/dist*(-1.0/dist/dist + xkappa/dist));
00367                     grad[2] += (val*dz/dist*(-1.0/dist/dist + xkappa/dist));
00368                 }
00369                 break;
00370
00371             case BCFL_UNUSED:
00372                 Vnm_print(2, "Vopot: Invalid bcfl (%d)!\n", thee->bcfl);
00373                 return 0;
00374
00375             case BCFL_FOCUS:
00376                 Vnm_print(2, "Vopot: Invalid bcfl (%d)!\n", thee->bcfl);
00377                 return 0;
00378
00379             default:
00380                 Vnm_print(2, "Vopot_pot: Bogus thee->bcfl flag (%d)!\n",
00381                     thee->bcfl);
00382                 return 0;
00383                 break;
00384         }
00385
00386         return 1;
00387     }
00388
00389     return 1;
00390 }
00391
00392

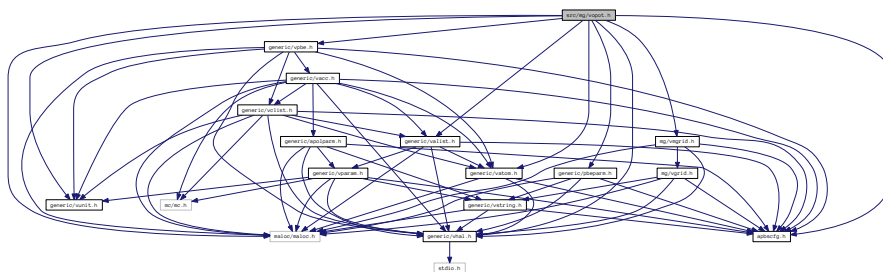
```

10.99 src/mg/vopot.h File Reference

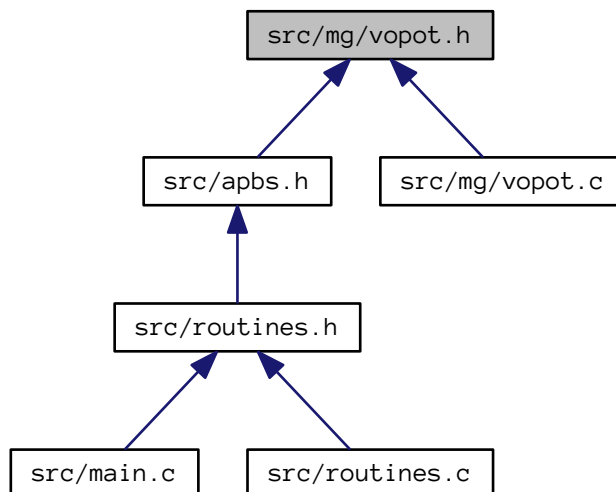
Potential oracle for Cartesian mesh data.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/pbeparm.h"
#include "generic/vatom.h"
#include "generic/valist.h"
#include "generic/vunit.h"
#include "generic/vpbe.h"
#include "mg/vmgrid.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)
FORTTRAN 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.

10.99.1 Detailed Description

Potential oracle for Cartesian mesh data.

Author

Nathan Baker

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
```

```

* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vopot.h](#).

10.100 vopot.h

```

00001
00062 #ifndef _VOPOT_H_
00063 #define _VOPOT_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "malloc/malloc.h"
00068
00069 #include "generic/vhal.h"
00070 #include "generic/pbeparm.h"
00071 #include "generic/vatom.h"
00072 #include "generic/valist.h"
00073 #include "generic/vunit.h"
00074 #include "generic/vpbe.h"
00075 #include "generic/pbeparm.h"
00076 #include "mg/vmgrid.h"
00077
00083 struct sVopot {
00084
00085     Vmgrid *mgrid;
00087     Vpbe *pbe;
00088     Vbcfl bcfl;
00090 };
00091
00096 typedef struct sVopot Vopot;
00097
00108 VEXTERNC Vopot* Vopot_ctor(Vmgrid *mgrid, Vpbe *
    pbe, Vbcfl bcfl);

```

```

00109
00121 VEXTERNC int Vopot_ctor2(Vopot *thee, Vmgrid *mgrid,
    Vpbe *pbe, Vbcfl bcfl);
00122
00131 VEXTERNC int Vopot_pot(Vopot *thee, double x[3], double *pot);
00132
00138 VEXTERNC void Vopot_dtor(Vopot **thee);
00139
00145 VEXTERNC void Vopot_dtor2(Vopot *thee);
00146
00160 VEXTERNC int Vopot_curvature(Vopot *thee, double pt[3], int cflag, double
00161     *curv);
00162
00171 VEXTERNC int Vopot_gradient(Vopot *thee, double pt[3], double grad[3] );
00172
00173
00174 #endif

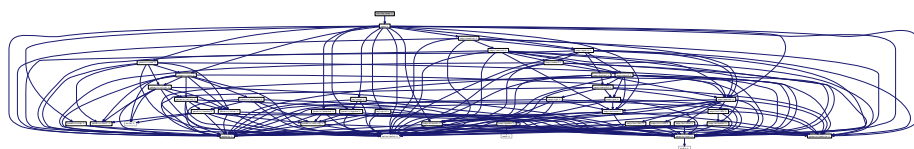
```

10.101 src/mg/vpmg.c File Reference

Class Vpmg methods.

```
#include "vpmg.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)
FORTTRAN 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)
FORTTRAN 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_PBType](#) pbetype, [PBEparm](#) *pbeparm)
Fill the specified array with accessibility values.
- VPRIVATE double [Vpmg_polarizEnergy](#) ([Vpmg](#) *thee, int extFlag)
Determines energy from polarizeable charge and interaction with fixed charges according to Rocchia et al.
- 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)
Vpmg_qmEnergy for SMPBE.
- 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)
Calculates charge-potential energy using summation over delta function positions (i.e. something like an Linf norm)
- 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)
Calculates charge-potential energy as integral over a volume.
- VPRIVATE void [Vpmg_splineSelect](#) (int srfrm, [Vacc](#) *acc, double *gpos, double win, double infrad, [Vatom](#) *atom, double *force)
Selects a spline based surface method from either VSM_SPLINE, VSM_SPLINE5 or VSM_SPLINE7.
- 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)
*Increment all boundary points by $pre1 * (charge/d) * (exp(-xkappa * (d-size)) / (1 + xkappa * size))$ to add the effect of the Debye-Huckel potential due to a single charge.*
- VPRIVATE void [bcCalcOrig](#) ([Vpmg](#) *thee)
- VPRIVATE int [gridPointsValid](#) (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])
This routine serves bcfl2. It returns (in tsr) the contraction independent portion of the Debye-Huckel potential tensor for a spherical ion with a central charge, dipole and quadrupole. See the code for an in depth description.
- 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)
Fill boundary condition arrays.
- VPRIVATE void [fillCoefMap](#) ([Vpmg](#) *thee)
Fill operator coefficient arrays from pre-calculated maps.

- VPRIVATE void [fillcoCoefMol](#) ([Vpmg](#) *thee)
Fill operator coefficient arrays from a molecular surface calculation.
- VPRIVATE void [fillcoCoefMolIon](#) ([Vpmg](#) *thee)
Fill ion (nonlinear) operator coefficient array from a molecular surface calculation.
- VPRIVATE void [fillcoCoefMolDiel](#) ([Vpmg](#) *thee)
Fill differential operator coefficient arrays from a molecular surface calculation.
- VPRIVATE void [fillcoCoefMolDielNoSmooth](#) ([Vpmg](#) *thee)
Fill differential operator coefficient arrays from a molecular surface calculation without smoothing.
- VPRIVATE void [fillcoCoefMolDielSmooth](#) ([Vpmg](#) *thee)
Fill differential operator coefficient arrays from a molecular surface calculation with smoothing.
- VPRIVATE void [fillcoCoefSpline](#) ([Vpmg](#) *thee)
Fill operator coefficient arrays from a spline-based surface calculation.
- VPRIVATE void [fillcoCoef](#) ([Vpmg](#) *thee)
Top-level driver to fill all operator coefficient arrays.
- VPRIVATE Vrc_Codes [fillcoCharge](#) ([Vpmg](#) *thee)
Top-level driver to fill source term charge array.
- VPRIVATE Vrc_Codes [fillcoChargeMap](#) ([Vpmg](#) *thee)
Fill source term charge array from a pre-calculated map.
- VPRIVATE void [fillcoChargeSpline1](#) ([Vpmg](#) *thee)
Fill source term charge array from linear interpolation.
- VPRIVATE double [bspline2](#) (double x)
Evaluate a cubic B-spline.
- VPRIVATE double [dbspline2](#) (double x)
Evaluate a cubic B-spline derivative.
- VPRIVATE void [fillcoChargeSpline2](#) ([Vpmg](#) *thee)
Fill source term charge array from cubic spline interpolation.
- 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](#) srfrm, [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](#) srfrm)
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](#) srfrm)
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)
Charge-field force due to a linear spline charge function.
- VPRIVATE void [qfForceSpline2](#) ([Vpmg](#) *thee, double *force, int atomID)
Charge-field force due to a cubic spline charge function.
- VPRIVATE void [qfForceSpline4](#) ([Vpmg](#) *thee, double *force, int atomID)
Charge-field force due to a quintic spline charge function.
- 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)

Mark the grid points inside a sphere with a particular value. This marks by resetting the the grid points inside the sphere to the specified value.

- VPRIVATE void **zlapSolve** (Vpmg *thee, double **solution, double **source, double **work1)
Calculate the solution to Poisson's equation with a simple Laplacian operator and zero-valued Dirichlet boundary conditions. Store the solution in thee->u.
- 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)
Return 2.5 plus difference of i - f.
- VPRIVATE double **bspline4** (double x)
Evaluate a 5th Order B-Spline (4th order polynomial)
- VPUBLIC double **db spline4** (double x)
Evaluate a 5th Order B-Spline derivative (4th order polynomial)
- VPUBLIC double **d2bspline4** (double x)
Evaluate the 2nd derivative of a 5th Order B-Spline.
- VPUBLIC double **d3bspline4** (double x)
Evaluate the 3rd derivative of a 5th Order B-Spline.
- VPUBLIC void **fillcoPermanentMultipole** (Vpmg *thee)
Fill source term charge array for the use of permanent multipoles.
- VPRIVATE void **fillcoCoefSpline4** (Vpmg *thee)
Fill operator coefficient arrays from a 7th order polynomial based surface calculation.
- VPUBLIC void **fillcoPermanentInduced** (Vpmg *thee)
- VPRIVATE void **fillcoCoefSpline3** (Vpmg *thee)
Fill operator coefficient arrays from a 5th order polynomial based surface calculation.
- VPRIVATE void **bcolcomp** (int *iparm, double *rparm, int *iwork, double *rwork, double *values, int *rowind, int *colptr, int *flag)
Build a column-compressed matrix in Harwell-Boeing format.
- VPRIVATE void **bcolcomp2** (int *iparm, double *rparm, int *nx, int *ny, int *nz, int *iz, int *ipc, double *rpc, double *ac, double *cc, double *values, int *rowind, int *colptr, int *flag)
Build a column-compressed matrix in Harwell-Boeing format.
- VPRIVATE void **bcolcomp3** (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *ac, double *cc, double *values, int *rowind, int *colptr, int *flag)
Build a column-compressed matrix in Harwell-Boeing format.
- VPRIVATE void **bcolcomp4** (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *oC, double *cc, double *oE, double *oN, double *uC, double *values, int *rowind, int *colptr, int *flag)
Build a column-compressed matrix in Harwell-Boeing format.
- VPRIVATE void **pcolcomp** (int *nrow, int *ncol, int *nnzero, double *values, int *rowind, int *colptr, char *path, char *title, char *mxtype)
Print a column-compressed matrix in Harwell-Boeing format.

10.101.1 Detailed Description

Class Vpmg methods.

Author

Nathan Baker

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* - Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of 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.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file [vpmg.c](#).

10.101.2 Function Documentation

10.101.2.1 VPRIVATE void bcCalc (Vpmg * *thee*)

Fill boundary condition arrays.

Author

Nathan Baker

Definition at line [4367](#) of file [vpmg.c](#).

10.101.2.2 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*)

Increment all boundary points by $pre1 * (charge/d) * (exp(-xkappa * (d-size)) / (1 + xkappa * size))$ to add the effect of the Debye-Huckel potential due to a single charge.

Author

Nathan Baker

Parameters

<i>apos</i>	Size of the ion
<i>charge</i>	Position of the ion
<i>xkappa</i>	Charge of the ion
<i>pre1</i>	Exponential screening factor
<i>gxcf</i>	Unit- and dielectric-dependent prefactor
<i>gycf</i>	Set to x-boundary values
<i>gzcf</i>	Set to y-boundary values
<i>xf</i>	Set to z-boundary values
<i>yf</i>	Boundary point x-coordinates
<i>zf</i>	Boundary point y-coordinates
<i>nx</i>	Boundary point z-coordinates
<i>ny</i>	Number of grid points in x-direction
<i>nz</i>	Number of grid points in y-direction Number of grid points in y-direction

Definition at line [2564](#) of file [vpmg.c](#).

10.101.2.3 VPRIVATE double bspline2 (double *x*)

Evaluate a cubic B-spline.

Author

Nathan Baker

Returns

Cubic B-spline value

Parameters

x	Position
-----	----------

Definition at line 5481 of file [vpmg.c](#).

10.101.2.4 VPRIVATE double bspline4 (double x)

Evaluate a 5th Order B-Spline (4th order polynomial)

Author

: Michael Schnieders

Returns

5th Order B-Spline

Parameters

x	Position
-----	----------

Definition at line 7121 of file [vpmg.c](#).

10.101.2.5 VPUBLIC double d2bspline4 (double x)

Evaluate the 2nd derivative of a 5th Order B-Spline.

Author

: Michael Schnieders

Returns

2nd derivative of a 5th Order B-Spline

Parameters

x	Position
-----	----------

Definition at line 7187 of file [vpmg.c](#).

10.101.2.6 VPUBLIC double d3bspline4 (double x)

Evaluate the 3rd derivative of a 5th Order B-Spline.

Author

: Michael Schnieders

Returns

3rd derivative of a 5th Order B-Spline

Parameters

x	Position
-----	----------

Definition at line 7214 of file [vpmg.c](#).

10.101.2.7 VPRIVATE double dbspline2 (double x)

Evaluate a cubic B-spline derivative.

Author

Nathan Baker

Returns

Cubic B-spline derivative

Parameters

x	Position
-----	----------

Definition at line 5497 of file [vpmg.c](#).

10.101.2.8 VPUBLIC double dbspline4 (double x)

Evaluate a 5th Order B-Spline derivative (4th order polynomial)

Author

: Michael Schnieders

Returns

5th Order B-Spline derivative

Parameters

x	Position
-----	----------

Definition at line 7155 of file [vpmg.c](#).

10.101.2.9 VPRIVATE Vrc_Codes fillCoCharge (Vpmg * *thee*)

Top-level driver to fill source term charge array.

Returns

Success/failure status

Author

Nathan Baker

Definition at line 5272 of file [vpmg.c](#).

10.101.2.10 VPRIVATE Vrc_Codes fillcoChargeMap (Vpmg * thee)

Fill source term charge array from a pre-calculated map.

Returns

Success/failure status

Author

Nathan Baker

Definition at line [5328](#) of file [vpmg.c](#).

10.101.2.11 VPRIVATE void fillcoChargeSpline1 (Vpmg * thee)

Fill source term charge array from linear interpolation.

Author

Nathan Baker

Definition at line [5376](#) of file [vpmg.c](#).

10.101.2.12 VPRIVATE void fillcoChargeSpline2 (Vpmg * thee)

Fill source term charge array from cubic spline interpolation.

Author

Nathan Baker

Definition at line [5513](#) of file [vpmg.c](#).

10.101.2.13 VPRIVATE void fillcoCoef (Vpmg * thee)

Top-level driver to fill all operator coefficient arrays.

Author

Nathan Baker

Definition at line [5232](#) of file [vpmg.c](#).

10.101.2.14 VPRIVATE void fillcoCoefMap (Vpmg * thee)

Fill operator coefficient arrays from pre-calculated maps.

Author

Nathan Baker

Definition at line [4474](#) of file [vpmg.c](#).

10.101.2.15 VPRIVATE void fillCoefMol (Vpmg * *thee*)

Fill operator coefficient arrays from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4597 of file [vpmg.c](#).

10.101.2.16 VPRIVATE void fillCoefMolDiel (Vpmg * *thee*)

Fill differential operator coefficient arrays from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4711 of file [vpmg.c](#).

10.101.2.17 VPRIVATE void fillCoefMolDielNoSmooth (Vpmg * *thee*)

Fill differential operator coefficient arrays from a molecular surface calculation without smoothing.

Author

Nathan Baker

Definition at line 4722 of file [vpmg.c](#).

10.101.2.18 VPRIVATE void fillCoefMolDielSmooth (Vpmg * *thee*)

Fill differential operator coefficient arrays from a molecular surface calculation with smoothing.

Molecular surface, dielectric smoothing following an implementation of Bruccoleri, et al. J Comput Chem 18 268-276 (1997).

This algorithm uses a 9 point harmonic smoothing technique - the point in question and all grid points 1/sqrt(2) grid spacings away.

Note

This uses *thee*->a1cf, *thee*->a2cf, *thee*->a3cf as temporary storage.

Author

Todd Dolinsky

Definition at line 4876 of file [vpmg.c](#).

10.101.2.19 VPRIVATE void fillCoefMollon (Vpmg * thee)

Fill ion (nonlinear) operator coefficient array from a molecular surface calculation.

Author

Nathan Baker

Definition at line [4613](#) of file [vpmg.c](#).

10.101.2.20 VPRIVATE void fillCoefSpline (Vpmg * thee)

Fill operator coefficient arrays from a spline-based surface calculation.

Author

Nathan Baker

Definition at line [5007](#) of file [vpmg.c](#).

10.101.2.21 VPRIVATE void fillCoefSpline3 (Vpmg * thee)

Fill operator coefficient arrays from a 5th order polynomial based surface calculation.

Author

Michael Schnieders

Definition at line [10415](#) of file [vpmg.c](#).

10.101.2.22 VPRIVATE void fillCoefSpline4 (Vpmg * thee)

Fill operator coefficient arrays from a 7th order polynomial based surface calculation.

Author

Michael Schnieders

Definition at line [9924](#) of file [vpmg.c](#).

10.101.2.23 VPUBLIC void fillCoPermanentMultipole (Vpmg * thee)

Fill source term charge array for the use of permanent multipoles.

Author

Michael Schnieders

Definition at line [7225](#) of file [vpmg.c](#).

10.101.2.24 VPRIVATE void markSphere (double *rtot*, double * *tpos*, int *nx*, int *ny*, int *nz*, double *hx*, double *hy*, double *hzd*, double *xmin*, double *ymin*, double *zmin*, double * *array*, double *markVal*)

Mark the grid points inside a sphere with a particular value. This marks by resetting the the grid points inside the sphere to the specified value.

Author

Nathan Baker

Parameters

<i>tpos</i>	Sphere radius
<i>nx</i>	Sphere position
<i>ny</i>	Number of grid points
<i>nz</i>	Number of grid points
<i>hx</i>	Number of grid points
<i>hy</i>	Grid spacing
<i>hz</i>	Grid spacing
<i>xmin</i>	Grid spacing
<i>ymin</i>	Grid lower corner
<i>zmin</i>	Grid lower corner
<i>array</i>	Grid lower corner
<i>markVal</i>	Grid values Value to mark with

Definition at line 6834 of file [vpmg.c](#).

10.101.2.25 VPRIVATE void multipolebc (double *r*, double *kappa*, double *eps_p*, double *eps_w*, double *rad*, double *tsr*[3])

This routine serves bcf12. It returns (in *tsr*) the contraction independent portion of the Debye-Huckel potential tensor for a spherical ion with a central charge, dipole and quadrupole. See the code for an in depth description.

Author

Michael Schnieders

Parameters

<i>kappa</i>	Distance to the boundary
<i>eps_p</i>	Exponential screening factor
<i>eps_w</i>	Solute dielectric
<i>rad</i>	Solvent dielectric
<i>tsr</i>	Radius of the sphere Contraction-independent portion of each tensor

Definition at line 3477 of file [vpmg.c](#).

10.101.2.26 VPRIVATE void qfForceSpline1 (Vpmg * *thee*, double * *force*, int *atomID*)

Charge-field force due to a linear spline charge function.

Author

Nathan Baker

Parameters

<i>atomID</i>	Set to force Valist atom ID
---------------	-----------------------------

Definition at line 6296 of file [vpmg.c](#).

10.101.2.27 VPRIVATE void qfForceSpline2 (Vpmg * *thee*, double * *force*, int *atomID*)

Charge-field force due to a cubic spline charge function.

Author

Nathan Baker

Parameters

<i>atomID</i>	Set to force Valist atom ID
---------------	-----------------------------

Definition at line 6433 of file [vpmg.c](#).

10.101.2.28 VPRIVATE void qfForceSpline4 (Vpmg * *thee*, double * *force*, int *atomID*)

Charge-field force due to a quintic spline charge function.

Author

Michael Schnieders

Parameters

<i>atomID</i>	Set to force Valist atom ID
---------------	-----------------------------

Definition at line 6546 of file [vpmg.c](#).

10.101.2.29 VPRIVATE double VFCHI4 (int *i*, double *f*)

Return 2.5 plus difference of i - f.

Author

Michael Schnieders

Returns

(2.5+((double)(i)-(f)))

Definition at line 7117 of file [vpmg.c](#).

10.101.2.30 VPRIVATE double Vpmg_polarizEnergy (Vpmg * *thee*, int *extFlag*)

Determines energy from polarizeable charge and interaction with fixed charges according to Rocchia et al.

Author

Nathan Baker

Returns

Energy in kT

Parameters

<i>extFlag</i>	If 1, add external energy contributions to result
----------------	---

Definition at line 1148 of file [vpmg.c](#).**10.101.2.31** `VPRIVATE double Vpmg_qfEnergyPoint (Vpmg * thee, int extFlag)`

Calculates charge-potential energy using summation over delta function positions (i.e. something like an Linf norm)

Author

Nathan Baker

Returns

Energy in kT

Parameters

<i>extFlag</i>	If 1, add external energy contributions to result
----------------	---

Definition at line 1704 of file [vpmg.c](#).**10.101.2.32** `VPRIVATE double Vpmg_qfEnergyVolume (Vpmg * thee, int extFlag)`

Calculates charge-potential energy as integral over a volume.

Author

Nathan Baker

Returns

Energy in kT

Parameters

<i>extFlag</i>	If 1, add external energy contributions to result
----------------	---

Definition at line 1861 of file [vpmg.c](#).**10.101.2.33** `VPUBLIC double Vpmg_qmEnergySMPBE (Vpmg * thee, int extFlag)`

Vpmg_qmEnergy for SMPBE.

Author

Vincent Chu

Definition at line 1490 of file [vpmg.c](#).

10.101.2.34 `VPRIVATE void Vpmg_splineSelect (int srfm, Vacc * acc, double * gpos, double win, double infrad, Vatom * atom, double * force)`

Selects a spline based surface method from either VSM_SPLINE, VSM_SPLINE5 or VSM_SPLINE7.

Author

David Gohara

Parameters

<i>acc</i>	Surface method, currently VSM_SPLINE, VSM_SPLINE5, or VSM_SPLINE7
<i>gpos</i>	Accessibility object
<i>win</i>	Position array -> array[3]
<i>infrad</i>	Spline window
<i>atom</i>	Inflation radius
<i>force</i>	Atom object Force array -> array[3]

Definition at line 1893 of file [vpmg.c](#).

10.101.2.35 `VPRIVATE void zlapSolve (Vpmg * thee, double ** solution, double ** source, double ** work1)`

Calculate the solution to Poisson's equation with a simple Laplacian operator and zero-valued Dirichlet boundary conditions. Store the solution in *thee*->u.

Author

Nathan Baker

Note

Vpmg_fillco must be called first

Parameters

<i>source</i>	Solution term vector
<i>work1</i>	Source term vector Work vector

Definition at line 6883 of file [vpmg.c](#).

10.102 vpmg.c

```

00001
00073 #include "vpmg.h"
00074
00075 VEMBED(rcsid="$Id$")
00076
00077 #if !defined(VINLINE_VPMG)
00078
00079 VPUBLIC unsigned long int Vpmg_memChk(Vpmg *thee) {
00080     if (thee == VNULL) return 0;
00081     return Vmem_bytes(thee->vmem);
00082 }
00083
00084 #endif /* if !defined(VINLINE_VPMG) */
00085
00086
00087 VPUBLIC void Vpmg_printColComp(Vpmg *thee, char path[72], char title[72],
00088     char mxtpe[3], int flag) {

```

```

00089
00090     int nn, nxm2, nym2, nzm2, ncol, nrow, nonz;
00091     double *nzval;
00092     int *colptr, *rowind;
00093
00094     /* Calculate the total number of unknowns */
00095     nxm2 = thee->pmgp->nx - 2;
00096     nym2 = thee->pmgp->ny - 2;
00097     nzm2 = thee->pmgp->nz - 2;
00098     nn = nxm2*ny2*nzm2;
00099     ncol = nn;
00100     nrow = nn;
00101
00102     /* Calculate the number of non-zero matrix entries:
00103      *   nn      nonzeros on diagonal
00104      *   nn-1    nonzeros on first off-diagonal
00105      *   nn-nx   nonzeros on second off-diagonal
00106      *   nn-nx*ny nonzeros on third off-diagonal
00107      *
00108      *   7*nn-2*nx*ny-2*nx-2 TOTAL non-zeros
00109      */
00110     nonz = 7*nn - 2*nxm2*ny2 - 2*nxm2 - 2;
00111     nzval = (double*)Vmem_malloc(thee->vmem, nonz, sizeof(double));
00112     rowind = (int*)Vmem_malloc(thee->vmem, nonz, sizeof(int));
00113     colptr = (int*)Vmem_malloc(thee->vmem, (ncol+1), sizeof(int));
00114
00115 #ifndef VAPBSQUIET
00116     Vnm_print(1, "Vpmg_printColComp: Allocated space for %d nonzeros\n",
00117         nonz);
00118 #endif
00119
00120     bcolcomp(thee->iparm, thee->rparm, thee->iwork, thee->
00121         rwork,
00122         nzval, rowind, colptr, &flag);
00123
00124 #if 0
00125     for (i=0; i<nn; i++) {
00126         Vnm_print(1, "nnz(%d) = %g\n", i, nzval[i]);
00127     }
00128 #endif
00129
00130     /* I do not understand why I need to pass nzval in this way, but it
00131      * works... */
00132     pcolcomp(&nrow, &ncol, &nonz, &(nzval[0]), rowind, colptr, path, title,
00133         mxtype);
00134
00135     Vmem_free(thee->vmem, (ncol+1), sizeof(int), (void **)&colptr);
00136     Vmem_free(thee->vmem, nonz, sizeof(int), (void **)&rowind);
00137     Vmem_free(thee->vmem, nonz, sizeof(double), (void **)&nzval);
00138
00139 }
00140
00141 VPUBLIC Vpmg* Vpmg_ctor(Vpmgp *pmgp, Vpbe *pbe, int focusFlag,
00142     Vpmg *pmgOLD, MGparm *mgparm, PBEparm_calcEnergy energyFlag) {
00143
00144     Vpmg *thee = VNULL;
00145
00146     thee = (Vpmg*)Vmem_malloc(VNULL, 1, sizeof(Vpmg) );
00147     VASSERT(thee != VNULL);
00148     VASSERT( Vpmg_ctor2(thee, pmgp, pbe, focusFlag, pmgOLD, mgparm,
00149         energyFlag) );
00150     return thee;
00151 }
00152
00153 VPUBLIC int Vpmg_ctor2(Vpmg *thee, Vpmgp *pmgp, Vpbe *pbe, int focusFlag,
00154     Vpmg *pmgOLD, MGparm *mgparm,
00155     PBEparm_calcEnergy energyFlag) {
00156     int i, j, nion;
00157     double ionConc[MAXION], ionQ[MAXION], ionRadii[MAXION], zkappa2, zks2;
00158     double ionstr, partMin[3], partMax[3];
00159     size_t size;
00160
00161     /* Get the parameters */
00162     VASSERT(pmgp != VNULL);
00163     VASSERT(pbe != VNULL);
00164     thee->pmgp = pmgp;
00165     thee->pbe = pbe;
00166
00167     /* Set up the memory */

```

```

00168     thee->vmem = Vmem_ctor("APBS:VPMG");
00169
00170
00171
00172     /* Initialize ion concentrations and valencies in PMG routines */
00173     zkappa2 = Vpbe_getZkappa2(thee->pbe);
00174     ionstr = Vpbe_getBulkIonicStrength(thee->pbe);
00175     if (ionstr > 0.0) zks2 = 0.5/ionstr;
00176     else zks2 = 0.0;
00177     Vpbe_getIons(thee->pbe, &nion, ionConc, ionRadii, ionQ);
00178
00179     /* TEMPORARY USEAQUA */
00180     /* Calculate storage requirements */
00181     if (mgparm->useAqua == 0) {
00182         Vpmgp_size(thee->pmgp);
00183     } else {
00184         VABORT_MSG0("Aqua is currently disabled");
00185     }
00186
00187     /* We need some additional storage if: nonlinear & newton OR cgmg */
00188     /* SMPBE Added - nonlin = 2 added since it mimics NPBE */
00189     if ( ( (thee->pmgp->nonlin == NONLIN_NPBE) || (thee->pmgp->
00190 nonlin == NONLIN_SMPBE) )
00191         && (thee->pmgp->meth == VSOL_Newton) ) || (thee->pmgp->
00192 meth == VSOL_CGMG) )
00193     {
00194         thee->pmgp->nrvk += (2*(thee->pmgp->nf));
00195     }
00196
00197     if (thee->pmgp->iinfo > 1) {
00198         Vnm_print(2, "Vpmg_ctor2: PMG chose nx = %d, ny = %d, nz = %d\n",
00199             thee->pmgp->nx, thee->pmgp->ny, thee->
00200 pmgp->nz);
00201         Vnm_print(2, "Vpmg_ctor2: PMG chose nlev = %d\n",
00202             thee->pmgp->nlev);
00203         Vnm_print(2, "Vpmg_ctor2: PMG chose nxc = %d, nyc = %d, nzc = %d\n",
00204             thee->pmgp->nxc, thee->pmgp->nyc, thee->
00205 pmgp->nzc);
00206         Vnm_print(2, "Vpmg_ctor2: PMG chose nf = %d, nc = %d\n",
00207             thee->pmgp->nf, thee->pmgp->nc);
00208         Vnm_print(2, "Vpmg_ctor2: PMG chose narr = %d, narrc = %d\n",
00209             thee->pmgp->narr, thee->pmgp->narrc);
00210         Vnm_print(2, "Vpmg_ctor2: PMG chose n_rpc = %d, n_iz = %d, n_ipc = %d\n",
00211             thee->pmgp->n_rpc, thee->pmgp->n_iz, thee->
00212 pmgp->n_ipc);
00213         Vnm_print(2, "Vpmg_ctor2: PMG chose nrwk = %d, niwk = %d\n",
00214             thee->pmgp->nrwk, thee->pmgp->niwk);
00215     }
00216
00217     /* Allocate boundary storage */
00218     thee->gxcf = (double *)Vmem_malloc(
00219         thee->vmem,
00220         10*(thee->pmgp->ny)*(thee->pmgp->nz),
00221         sizeof(double)
00222     );
00223     thee->gycf = (double *)Vmem_malloc(
00224         thee->vmem,
00225         10*(thee->pmgp->nx)*(thee->pmgp->nz),
00226         sizeof(double)
00227     );
00228     thee->gzcf = (double *)Vmem_malloc(
00229         thee->vmem,
00230         10*(thee->pmgp->nx)*(thee->pmgp->ny),
00231         sizeof(double)
00232     );
00233
00234
00235
00236
00237     /* Warn users if they are using BCFL_MAP that
00238     we do not include external energies */
00239     if (thee->pmgp->bcfl == BCFL_MAP)
00240         Vnm_print(2, "Vpmg_ctor2: \nWarning: External energies are not used in BCFL_MAP calculations!\n");
00241
00242     if (focusFlag) {
00243
00244         /* Overwrite any default or user-specified boundary condition

```



```

00245      * arguments; we are now committed to a calculation via focusing */
00246      if (thee->pmgp->bcfl != BCFL_FOCUS) {
00247          Vnm_print(2,
00248              "Vpmg_ctor2: reset boundary condition flag to BCFL_FOCUS!\n");
00249          thee->pmgp->bcfl = BCFL_FOCUS;
00250      }
00251      /* Fill boundaries */
00252      Vnm_print(0, "Vpmg_ctor2: Filling boundary with old solution!\n");
00253      focusFillBound(thee, pmgOLD);
00254
00255      /* Calculate energetic contributions from region outside focusing
00256      * domain */
00257      if (energyFlag != PCE_NO) {
00258          if (mgparm->type == MCT_PARALLEL) {
00259              for (j=0; j<3; j++) {
00260                  partMin[j] = mgparm->partDisjCenter[j]
00261                      - 0.5*mgparm->partDisjLength[j];
00262                  partMax[j] = mgparm->partDisjCenter[j]
00263                      + 0.5*mgparm->partDisjLength[j];
00264              }
00265              } else {
00266                  for (j=0; j<3; j++) {
00267                      partMin[j] = mgparm->center[j] - 0.5*mgparm->glen[j];
00268                      partMax[j] = mgparm->center[j] + 0.5*mgparm->glen[j];
00269                  }
00270              extEnergy(thee, pmgOLD, energyFlag, partMin, partMax,
00271                  mgparm->partDisjOwnSide);
00272          }
00273      } else {
00274          /* Ignore external energy contributions */
00275          thee->extQmEnergy = 0;
00276          thee->extDiEnergy = 0;
00277          thee->extQfEnergy = 0;
00278      }
00279
00280      /* Allocate partition vector storage */
00281      size = (thee->pmgp->nx)*(thee->pmgp->ny)*(thee->pmgp->nz);
00282      thee->pvec = (double *)Vmem_malloc(
00283          thee->vmem,
00284          size,
00285          sizeof(double)
00286      );
00287
00288      /* Allocate remaining storage */
00289      thee->iparm = (int *)Vmem_malloc(thee->vmem, 100, sizeof(int));
00290      thee->rparm = (double *)Vmem_malloc(thee->vmem, 100, sizeof(double));
00291      thee->iwork = (int *)Vmem_malloc(thee->vmem, thee->pmgp->
00292          niwk, sizeof(int));
00293      thee->rwork = (double *)Vmem_malloc(thee->vmem, thee->pmgp->
00294          nrwk, sizeof(double));
00295      thee->charge = (double *)Vmem_malloc(thee->vmem, thee->pmgp->
00296          narr, sizeof(double));
00297      thee->kappa = (double *)Vmem_malloc(thee->vmem, thee->pmgp->
00298          narr, sizeof(double));
00299      thee->pot = (double *)Vmem_malloc(thee->vmem, thee->pmgp->
00300          narr, sizeof(double));
00301      thee->epsx = (double *)Vmem_malloc(thee->vmem, thee->pmgp->
00302          narr, sizeof(double));
00303      thee->epsy = (double *)Vmem_malloc(thee->vmem, thee->pmgp->
00304          narr, sizeof(double));
00305      thee->epsz = (double *)Vmem_malloc(thee->vmem, thee->pmgp->
00306          narr, sizeof(double));
00307      thee->alcf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->
00308          narr, sizeof(double));
00309      thee->a2cf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->
00310          narr, sizeof(double));
00311      thee->a3cf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->
00312          narr, sizeof(double));
00313      thee->ccf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->
00314          narr, sizeof(double));
00315      thee->fcf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->
00316          narr, sizeof(double));
00317      thee->tcf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->
00318          narr, sizeof(double));

```

```

00312     thee->u      = (double *)Vmem_malloc(thee->vmem,  thee->pmgp->narr, sizeof(double));
00313     thee->xf      = (double *)Vmem_malloc(thee->vmem,  5*(thee->pmgp->nx), sizeof(double));
00314     thee->yf      = (double *)Vmem_malloc(thee->vmem,  5*(thee->pmgp->ny), sizeof(double));
00315     thee->zf      = (double *)Vmem_malloc(thee->vmem,  5*(thee->pmgp->nz), sizeof(double));
00316
00317
00318
00319     /* Packs parameters into the iparm and rparm arrays */
00320     Vpackmg(thee->iparm, thee->rparm, &(thee->pmgp->nrvk), &(thee->
pmgp->niwk),
00321             &(thee->pmgp->nx), &(thee->pmgp->ny), &(thee->pmgp->
nz),
00322             &(thee->pmgp->nlev), &(thee->pmgp->nul), &(thee->pmgp->
nu2),
00323             &(thee->pmgp->mgkey), &(thee->pmgp->itmax), &(thee->
pmgp->istop),
00324             &(thee->pmgp->ipcon), &(thee->pmgp->nonlin), &(thee->
pmgp->mgs moo),
00325             &(thee->pmgp->mgprol), &(thee->pmgp->mgcoar), &(thee->
pmgp->mgsolv),
00326             &(thee->pmgp->mgdisc), &(thee->pmgp->iinfo), &(thee->
pmgp->errtol),
00327             &(thee->pmgp->ipkey), &(thee->pmgp->omegal), &(thee->
pmgp->omegan),
00328             &(thee->pmgp->irite), &(thee->pmgp->iperf));
00329
00330
00331
00332     /* Currently for SMPBE type calculations we do not want to apply a scale
00333        factor to the ionConc */
00334     switch(pmgp->ipkey) {
00341
00342         case IPKEY_SMPBE:
00343
00344             Vmvpdefinitmpbe(&nion, ionQ, ionConc, &pbe->
smvolume, &pbe->smsize);
00345             break;
00346
00347
00348
00349         case IPKEY_NPBE:
00350
00351             /* Else adjust the ionConc by scaling factor zks2 */
00352             for (i=0; i<nion; i++)
00353                 ionConc[i] = zks2 * ionConc[i];
00354
00355             Vmvpdefinitnpbe(&nion, ionQ, ionConc);
00356             break;
00357
00358
00359
00360         case IPKEY_LPBE:
00361
00362             /* Else adjust the ionConc by scaling factor zks2 */
00363             for (i=0; i<nion; i++)
00364                 ionConc[i] = zks2 * ionConc[i];
00365
00366             Vmvpdefinitlpbe(&nion, ionQ, ionConc);
00367             break;
00368
00369
00370
00371         default:
00372             Vnm_print(2, "PMG: Warning: PBE structure not initialized!\n");
00373             /* Else adjust the ionConc by scaling factor zks2 */
00374             for (i=0; i<nion; i++)
00375                 ionConc[i] = zks2 * ionConc[i];
00376             break;
00377     }
00378
00379     /* Set the default chargeSrc for 5th order splines */
00380     thee->chargeSrc = mgparm->chgs;
00381
00382     /* Turn off restriction of observable calculations to a specific
00383        * partition */
00384     Vpmg_unsetPart(thee);
00385
00386     /* The coefficient arrays have not been filled */
00387     thee->filled = 0;
00388
00389

```

```

00390     /*
00391     * TODO: Move the dtor out of here. The current ctor is done in routines.c,
00392     *       This was originally moved out to kill a memory leak. The dtor has
00393     *       has been removed from initMG and placed back here to keep memory
00394     *       usage low. killMG has been modified accordingly.
00395     */
00396     Vpmg_dtor(&pmgOLD);
00397
00398     return 1;
00399 }
00400
00401 VPUBLIC int Vpmg_solve(Vpmg *thee) {
00402     int i,
00403         nx,
00404         ny,
00405         nz,
00406         n;
00407     double zkappa2;
00408
00409     nx = thee->pmgp->nx;
00410     ny = thee->pmgp->ny;
00411     nz = thee->pmgp->nz;
00412     n = nx*ny*nz;
00413
00414     if (!(thee->filled)) {
00415         Vnm_print(2, "Vpmg_solve: Need to call Vpmg_fillco()!\n");
00416         return 0;
00417     }
00418
00419     /* Fill the "true solution" array */
00420     for (i=0; i<n; i++) {
00421         thee->tcf[i] = 0.0;
00422     }
00423
00424     /* Fill the RHS array */
00425     for (i=0; i<n; i++) {
00426         thee->fcf[i] = thee->charge[i];
00427     }
00428
00429     /* Fill the operator coefficient array. */
00430     for (i=0; i<n; i++) {
00431         thee->a1cf[i] = thee->epsx[i];
00432         thee->a2cf[i] = thee->epsy[i];
00433         thee->a3cf[i] = thee->epsz[i];
00434     }
00435
00436     /* Fill the nonlinear coefficient array by multiplying the kappa
00437     * accessibility array (containing values between 0 and 1) by zkappa2. */
00438     zkappa2 = Vpbe_getZkappa2(thee->pbe);
00439     if (zkappa2 > VPMGSMALL) {
00440         for (i=0; i<n; i++) {
00441             thee->ccf[i] = zkappa2*thee->kappa[i];
00442         }
00443     } else {
00444         for (i=0; i<n; i++) {
00445             thee->ccf[i] = 0.0;
00446         }
00447     }
00448
00449     switch(thee->pmgp->meth) {
00450         /* CGMG (linear) */
00451         case VSOL_CGMG:
00452             if (thee->pmgp->iinfo > 1)
00453                 Vnm_print(2, "Driving with CGMGDRIV\n");
00454             VABORT_MSG0("CGMGDRIV is not currently supported");
00455             break;
00456
00457         /* Newton (nonlinear) */
00458         case VSOL_Newton:
00459             if (thee->pmgp->iinfo > 1)
00460                 Vnm_print(2, "Driving with NEWDRIV\n");
00461             Vnewdriv
00462                 (thee->iparm, thee->rparm, thee->iwork, thee->
00463                 rwork,
00464                 thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->
00465                 gycf,

```

```

00469             thee->gzcf, thee->alcf, thee->a2cf, thee->
00470             a3cf, thee->ccf,             thee->fcf, thee->tcf);
00471             break;
00472
00473             /* MG (linear/nonlinear) */
00474             case VSOL_MG:
00475
00476                 if (thee->pmgpg->iinfo > 1)
00477                     Vnm_print(2, "Driving with MGDRIV\n");
00478
00479                 Vmgdriv(thee->iparm, thee->rparm, thee->iwork, thee->
00480                 rwork,             thee->u, thee->xf, thee->yf, thee->
00481                 zf, thee->gxcf, thee->gycf,             thee->gzcf, thee->alcf, thee->
00482                 a2cf, thee->a3cf, thee->ccf,             thee->fcf, thee->tcf);
00483             break;
00484
00485             /* CGHS (linear/nonlinear) */
00486             case VSOL_CG:
00487
00488                 if (thee->pmgpg->iinfo > 1)
00489                     Vnm_print(2, "Driving with NCGHSDRIV\n");
00490
00491                 VABORT_MSG0("NCGHSDRIV is not currently supported");
00492                 break;
00493
00494             /* SOR (linear/nonlinear) */
00495             case VSOL_SOR:
00496
00497                 if (thee->pmgpg->iinfo > 1)
00498                     Vnm_print(2, "Driving with NSORDRIV\n");
00499
00500                 VABORT_MSG0("NSORDRIV is not currently supported");
00501                 break;
00502
00503             /* GSRB (linear/nonlinear) */
00504             case VSOL_RBGS:
00505
00506                 if (thee->pmgpg->iinfo > 1)
00507                     Vnm_print(2, "Driving with NGSRBDRIV\n");
00508
00509                 VABORT_MSG0("NGSRBDRIV is not currently supported");
00510                 break;
00511
00512             /* WJAC (linear/nonlinear) */
00513             case VSOL_WJ:
00514
00515                 if (thee->pmgpg->iinfo > 1)
00516                     Vnm_print(2, "Driving with NWJACDRIV\n");
00517
00518                 VABORT_MSG0("NWJACDRIV is not currently supported");
00519                 break;
00520
00521             /* RICH (linear/nonlinear) */
00522             case VSOL_Richardson:
00523
00524                 if (thee->pmgpg->iinfo > 1)
00525                     Vnm_print(2, "Driving with NRICHDRIV\n");
00526
00527                 VABORT_MSG0("NRICHDRIV is not currently supported");
00528                 break;
00529
00530             /* CGMG (linear) TEMPORARY USEAQUA */
00531             case VSOL_CGMGAqua:
00532
00533                 if (thee->pmgpg->iinfo > 1)
00534                     Vnm_print(2, "Driving with CGMGDRIVAQUA\n");
00535
00536                 VABORT_MSG0("CGMGDRIVAQUA is not currently supported");
00537                 break;
00538
00539             /* Newton (nonlinear) TEMPORARY USEAQUA */
00540             case VSOL_NewtonAqua:
00541
00542                 if (thee->pmgpg->iinfo > 1)
00543                     Vnm_print(2, "Driving with NEWDRIVAQUA\n");
00544
00545                 VABORT_MSG0("NEWDRIVAQUA is not currently supported");

```

```

00546         break;
00547
00548     /* Error handling */
00549     default:
00550         Vnm_print(2, "Vpmg_solve: invalid solver method key (%d)\n",
00551             thee->pmgp->key);
00552         return 0;
00553         break;
00554     }
00555
00556     return 1;
00557
00558 }
00559
00560
00561 VPUBLIC void Vpmg_dtor(Vpmg **thee) {
00562
00563     if ((*thee) != VNULL) {
00564         Vpmg_dtor2(*thee);
00565         Vmem_free(VNULL, 1, sizeof(Vpmg), (void **)thee);
00566         (*thee) = VNULL;
00567     }
00568
00569 }
00570
00571 VPUBLIC void Vpmg_dtor2(Vpmg *thee) {
00572
00573     /* Clean up the storage */
00574
00575     Vmem_free(thee->vmem, 100, sizeof(int),
00576         (void **)&(thee->iparm));
00577     Vmem_free(thee->vmem, 100, sizeof(double),
00578         (void **)&(thee->rparm));
00579     Vmem_free(thee->vmem, thee->pmgp->niwk, sizeof(int),
00580         (void **)&(thee->iwork));
00581     Vmem_free(thee->vmem, thee->pmgp->nrwk, sizeof(double),
00582         (void **)&(thee->rwork));
00583     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00584         (void **)&(thee->charge));
00585     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00586         (void **)&(thee->kappa));
00587     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00588         (void **)&(thee->pot));
00589     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00590         (void **)&(thee->epsx));
00591     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00592         (void **)&(thee->epsy));
00593     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00594         (void **)&(thee->epsz));
00595     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00596         (void **)&(thee->alcf));
00597     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00598         (void **)&(thee->a2cf));
00599     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00600         (void **)&(thee->a3cf));
00601     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00602         (void **)&(thee->ccf));
00603     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00604         (void **)&(thee->fcf));
00605     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00606         (void **)&(thee->tcf));
00607     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00608         (void **)&(thee->u));
00609     Vmem_free(thee->vmem, 5*(thee->pmgp->nx), sizeof(double),
00610         (void **)&(thee->xf));
00611     Vmem_free(thee->vmem, 5*(thee->pmgp->ny), sizeof(double),
00612         (void **)&(thee->yf));
00613     Vmem_free(thee->vmem, 5*(thee->pmgp->nz), sizeof(double),
00614         (void **)&(thee->zf));
00615     Vmem_free(thee->vmem, 10*(thee->pmgp->ny)*(thee->pmgp->nz), sizeof(double),
00616         (void **)&(thee->gxcf));
00617     Vmem_free(thee->vmem, 10*(thee->pmgp->nx)*(thee->pmgp->nz), sizeof(double),
00618         (void **)&(thee->gycf));
00619     Vmem_free(thee->vmem, 10*(thee->pmgp->nx)*(thee->pmgp->ny), sizeof(double),
00620         (void **)&(thee->gzcf));
00621     Vmem_free(thee->vmem, (thee->pmgp->nx)*(thee->pmgp->ny)*(thee->
pmgp->nz),
00622         sizeof(double), (void **)&(thee->pvec));
00623
00624     Vmem_dtor(&(thee->vmem));
00625 }

```

```

00626
00627 VPUBLIC void Vpmg_setPart(Vpmg *thee, double lowerCorner[3],
00628     double upperCorner[3], int bflags[6]) {
00629
00630     Valist *alist;
00631     Vatom *atom;
00632     int i, j, k, nx, ny, nz;
00633     double xmin, ymin, zmin, x, y, z, hx, hy, hzed, xok, yok, zok;
00634     double x0,x1,y0,y1,z0,z1;
00635
00636     nx = thee->pmgp->nx;
00637     ny = thee->pmgp->ny;
00638     nz = thee->pmgp->nz;
00639     hx = thee->pmgp->hx;
00640     hy = thee->pmgp->hy;
00641     hzed = thee->pmgp->hzed;
00642     xmin = thee->pmgp->xcent - 0.5*hx*(nx-1);
00643     ymin = thee->pmgp->ycent - 0.5*hy*(ny-1);
00644     zmin = thee->pmgp->zcent - 0.5*hzed*(nz-1);
00645
00646     xok = 0;
00647     yok = 0;
00648     zok = 0;
00649
00650     /* We need have called Vpmg_fillco first */
00651
00652     alist = thee->pbe->alist;
00653
00654     Vnm_print(0, "Vpmg_setPart: lower corner = (%g, %g, %g)\n",
00655         lowerCorner[0], lowerCorner[1], lowerCorner[2]);
00656     Vnm_print(0, "Vpmg_setPart: upper corner = (%g, %g, %g)\n",
00657         upperCorner[0], upperCorner[1], upperCorner[2]);
00658     Vnm_print(0, "Vpmg_setPart: actual minima = (%g, %g, %g)\n",
00659         xmin, ymin, zmin);
00660     Vnm_print(0, "Vpmg_setPart: actual maxima = (%g, %g, %g)\n",
00661         xmin+hx*(nx-1), ymin+hy*(ny-1), zmin+hzed*(nz-1));
00662     Vnm_print(0, "Vpmg_setPart: bflag[FRONT] = %d\n",
00663         bflags[VAPBS_FRONT]);
00664     Vnm_print(0, "Vpmg_setPart: bflag[BACK] = %d\n",
00665         bflags[VAPBS_BACK]);
00666     Vnm_print(0, "Vpmg_setPart: bflag[LEFT] = %d\n",
00667         bflags[VAPBS_LEFT]);
00668     Vnm_print(0, "Vpmg_setPart: bflag[RIGHT] = %d\n",
00669         bflags[VAPBS_RIGHT]);
00670     Vnm_print(0, "Vpmg_setPart: bflag[UP] = %d\n",
00671         bflags[VAPBS_UP]);
00672     Vnm_print(0, "Vpmg_setPart: bflag[DOWN] = %d\n",
00673         bflags[VAPBS_DOWN]);
00674
00675     /* Identify atoms as inside, outside, or on the border
00676     If on the border, use the bflags to determine if there
00677     is an adjacent processor - if so, this atom should be equally
00678     shared. */
00679
00680     for (i=0; i<Valist_getNumberAtoms(alist); i++) {
00681         atom = Valist_getAtom(alist, i);
00682
00683         if ((atom->position[0] < upperCorner[0]) &&
00684             (atom->position[0] > lowerCorner[0])) xok = 1;
00685         else {
00686             if ((VABS(atom->position[0] - lowerCorner[0]) < VPMGSMALL) &&
00687                 (bflags[VAPBS_LEFT] == 0)) xok = 1;
00688             else if ((VABS(atom->position[0] - lowerCorner[0]) <
00689                 VPMGSMALL) &&
00690                 (bflags[VAPBS_LEFT] == 1)) xok = 0.5;
00691             else if ((VABS(atom->position[0] - upperCorner[0]) <
00692                 VPMGSMALL) &&
00693                 (bflags[VAPBS_RIGHT] == 0)) xok = 1;
00694             else if ((VABS(atom->position[0] - upperCorner[0]) <
00695                 VPMGSMALL) &&
00696                 (bflags[VAPBS_RIGHT] == 1)) xok = 0.5;
00697             else xok = 0;
00698         }
00699         if ((atom->position[1] < upperCorner[1]) &&
00700             (atom->position[1] > lowerCorner[1])) yok = 1;
00701         else {
00702             if ((VABS(atom->position[1] - lowerCorner[1]) < VPMGSMALL) &&
00703                 (bflags[VAPBS_BACK] == 0)) yok = 1;
00704             else if ((VABS(atom->position[1] - lowerCorner[1]) <
00705                 VPMGSMALL) &&
00706                 (bflags[VAPBS_BACK] == 1)) yok = 0.5;

```

```

00703         else if ((VABS(atom->position[1] - upperCorner[1]) <
VPMGSMALL) &&
00704             (bflags[VAPBS_FRONT] == 0)) yok = 1;
00705         else if ((VABS(atom->position[1] - upperCorner[1]) <
VPMGSMALL) &&
00706             (bflags[VAPBS_FRONT] == 1)) yok = 0.5;
00707         else yok = 0;
00708     }
00709     if ((atom->position[2] < upperCorner[2]) &&
00710         (atom->position[2] > lowerCorner[2])) zok = 1;
00711     else {
00712         if ((VABS(atom->position[2] - lowerCorner[2]) < VPMGSMALL) &&
00713             (bflags[VAPBS_DOWN] == 0)) zok = 1;
00714         else if ((VABS(atom->position[2] - lowerCorner[2]) <
VPMGSMALL) &&
00715             (bflags[VAPBS_DOWN] == 1)) zok = 0.5;
00716         else if ((VABS(atom->position[2] - upperCorner[2]) <
VPMGSMALL) &&
00717             (bflags[VAPBS_UP] == 0)) zok = 1;
00718         else if ((VABS(atom->position[2] - upperCorner[2]) <
VPMGSMALL) &&
00719             (bflags[VAPBS_UP] == 1)) zok = 0.5;
00720         else zok = 0;
00721     }
00722     atom->partID = xok*yok*zok;
00723     /*
00724     Vnm_print(1, "DEBUG (%s, %d):  atom->position[0] - upperCorner[0] = %g\n",
00725         __FILE__, __LINE__, atom->position[0] - upperCorner[0]);
00726     Vnm_print(1, "DEBUG (%s, %d):  atom->position[0] - lowerCorner[0] = %g\n",
00727         __FILE__, __LINE__, atom->position[0] - lowerCorner[0]);
00728     Vnm_print(1, "DEBUG (%s, %d):  atom->position[1] - upperCorner[1] = %g\n",
00729         __FILE__, __LINE__, atom->position[1] - upperCorner[1]);
00730     Vnm_print(1, "DEBUG (%s, %d):  atom->position[1] - lowerCorner[1] = %g\n",
00731         __FILE__, __LINE__, atom->position[1] - lowerCorner[1]);
00732     Vnm_print(1, "DEBUG (%s, %d):  atom->position[2] - upperCorner[2] = %g\n",
00733         __FILE__, __LINE__, atom->position[2] - upperCorner[2]);
00734     Vnm_print(1, "DEBUG (%s, %d):  atom->position[2] - lowerCorner[0] = %g\n",
00735         __FILE__, __LINE__, atom->position[2] - lowerCorner[2]);
00736     Vnm_print(1, "DEBUG (%s, %d):  xok = %g, yok = %g, zok = %g\n",
00737         __FILE__, __LINE__, xok, yok, zok);
00738     */
00739 }
00740
00741 }
00742
00743 /* Load up pvec -
00744 For all points within h{axis}/2 of a border - use a gradient
00745 to determine the pvec weight.
00746 Points on the boundary depend on the presence of an adjacent
00747 processor. */
00748
00749 for (i=0; i<(nx*ny*nz); i++) three->pvec[i] = 0.0;
00750
00751 for (i=0; i<nx; i++) {
00752     xok = 0.0;
00753     x = i*hx + xmin;
00754     if ( (x < (upperCorner[0]-hx/2)) &&
00755         (x > (lowerCorner[0]+hx/2))
00756         ) xok = 1.0;
00757     else if ( (VABS(x - lowerCorner[0]) < VPMGSMALL) &&
00758         (bflags[VAPBS_LEFT] == 0)) xok = 1.0;
00759     else if ( (VABS(x - lowerCorner[0]) < VPMGSMALL) &&
00760         (bflags[VAPBS_LEFT] == 1)) xok = 0.5;
00761     else if ( (VABS(x - upperCorner[0]) < VPMGSMALL) &&
00762         (bflags[VAPBS_RIGHT] == 0)) xok = 1.0;
00763     else if ( (VABS(x - upperCorner[0]) < VPMGSMALL) &&
00764         (bflags[VAPBS_RIGHT] == 1)) xok = 0.5;
00765     else if ((x > (upperCorner[0] + hx/2)) || (x < (lowerCorner[0] - hx/2))) xok = 0.0;
00766     else if ((x < (upperCorner[0] + hx/2)) || (x > (lowerCorner[0] - hx/2))) {
00767         x0 = VMAX2(x - hx/2, lowerCorner[0]);
00768         x1 = VMIN2(x + hx/2, upperCorner[0]);
00769         xok = VABS(x1-x0)/hx;
00770
00771         if (xok < 0.0) {
00772             if (VABS(xok) < VPMGSMALL) xok = 0.0;
00773             else {
00774                 Vnm_print(2, "Vpmg_setPart:  fell off x-interval (%1.12E)!\n",
00775                     xok);
00776                 VASSERT(0);
00777             }
00778         }
00779     }
00780 }

```

```

00779         if (xok > 1.0) {
00780             if (VABS(xok - 1.0) < VPMGSMALL) xok = 1.0;
00781             else {
00782                 Vnm_print(2, "Vpmg_setPart: fell off x-interval (%1.12E)!\n",
00783                     xok);
00784                 VASSERT(0);
00785             }
00786         }
00787     } else xok = 0.0;
00788
00789     for (j=0; j<ny; j++) {
00790         yok = 0.0;
00791         y = j*hy + ymin;
00792         if ((y < (upperCorner[1]-hy/2)) && (y > (lowerCorner[1]+hy/2))) yok = 1.0;
00793         else if ((VABS(y - lowerCorner[1]) < VPMGSMALL) &&
00794             (bflags[VAPBS_BACK] == 0)) yok = 1.0;
00795         else if ((VABS(y - lowerCorner[1]) < VPMGSMALL) &&
00796             (bflags[VAPBS_BACK] == 1)) yok = 0.5;
00797         else if ((VABS(y - upperCorner[1]) < VPMGSMALL) &&
00798             (bflags[VAPBS_FRONT] == 0)) yok = 1.0;
00799         else if ((VABS(y - upperCorner[1]) < VPMGSMALL) &&
00800             (bflags[VAPBS_FRONT] == 1)) yok = 0.5;
00801         else if ((y > (upperCorner[1] + hy/2)) || (y < (lowerCorner[1] - hy/2))) yok=0.0;
00802         else if ((y < (upperCorner[1] + hy/2)) || (y > (lowerCorner[1] - hy/2))) {
00803             y0 = VMAX2(y - hy/2, lowerCorner[1]);
00804             y1 = VMIN2(y + hy/2, upperCorner[1]);
00805             yok = VABS(y1-y0)/hy;
00806         }
00807
00808         if (yok < 0.0) {
00809             if (VABS(yok) < VPMGSMALL) yok = 0.0;
00810             else {
00811                 Vnm_print(2, "Vpmg_setPart: fell off y-interval (%1.12E)!\n",
00812                     yok);
00813                 VASSERT(0);
00814             }
00815         }
00816         if (yok > 1.0) {
00817             if (VABS(yok - 1.0) < VPMGSMALL) yok = 1.0;
00818             else {
00819                 Vnm_print(2, "Vpmg_setPart: fell off y-interval (%1.12E)!\n",
00820                     yok);
00821                 VASSERT(0);
00822             }
00823         }
00824     }
00825     else yok=0.0;
00826
00827     for (k=0; k<nz; k++) {
00828         zok = 0.0;
00829         z = k*hzed + zmin;
00830         if ((z < (upperCorner[2]-hz/2)) && (z > (lowerCorner[2]+hz/2))) zok = 1.0;
00831         else if ((VABS(z - lowerCorner[2]) < VPMGSMALL) &&
00832             (bflags[VAPBS_DOWN] == 0)) zok = 1.0;
00833         else if ((VABS(z - lowerCorner[2]) < VPMGSMALL) &&
00834             (bflags[VAPBS_DOWN] == 1)) zok = 0.5;
00835         else if ((VABS(z - upperCorner[2]) < VPMGSMALL) &&
00836             (bflags[VAPBS_UP] == 0)) zok = 1.0;
00837         else if ((VABS(z - upperCorner[2]) < VPMGSMALL) &&
00838             (bflags[VAPBS_UP] == 1)) zok = 0.5;
00839         else if ((z > (upperCorner[2] + hz/2)) || (z < (lowerCorner[2] - hz/2))) zok=0.0;
00840         else if ((z < (upperCorner[2] + hz/2)) || (z > (lowerCorner[2] - hz/2))) {
00841             z0 = VMAX2(z - hz/2, lowerCorner[2]);
00842             z1 = VMIN2(z + hz/2, upperCorner[2]);
00843             zok = VABS(z1-z0)/hz;
00844         }
00845
00846         if (zok < 0.0) {
00847             if (VABS(zok) < VPMGSMALL) zok = 0.0;
00848             else {
00849                 Vnm_print(2, "Vpmg_setPart: fell off z-interval (%1.12E)!\n",
00850                     zok);
00851                 VASSERT(0);
00852             }
00853         }
00854         if (zok > 1.0) {
00855             if (VABS(zok - 1.0) < VPMGSMALL) zok = 1.0;
00856             else {
00857                 Vnm_print(2, "Vpmg_setPart: fell off z-interval (%1.12E)!\n",
00858                     zok);
00859                 VASSERT(0);
00860             }
00861         }
00862     }

```



```

00860         }
00861     }
00862     else zok = 0.0;
00863
00864     if (VABS(xok*yok*zok) < VPMGSMALL) thee->pvec[IJK(i,j,k)] = 0.0;
00865     else thee->pvec[IJK(i,j,k)] = xok*yok*zok;
00866
00867     }
00868 }
00869 }
00870 }
00871
00872 VPUBLIC void Vpmg_unsetPart(Vpmg *thee) {
00873
00874     int i, nx, ny, nz;
00875     Vatom *atom;
00876     Valist *alist;
00877
00878     VASSERT(thee != VNULL);
00879
00880     nx = thee->pmgp->nx;
00881     ny = thee->pmgp->ny;
00882     nz = thee->pmgp->nz;
00883     alist = thee->pbe->alist;
00884
00885     for (i=0; i<(nx*ny*nz); i++) thee->pvec[i] = 1;
00886     for (i=0; i<Valist_getNumberAtoms(alist); i++) {
00887         atom = Valist_getAtom(alist, i);
00888         atom->partID = 1;
00889     }
00890 }
00891
00892 VPUBLIC int Vpmg_fillArray(Vpmg *thee, double *vec, Vdata_Type type,
00893     double parm, Vhal_PBEType pbetype, PBEparm *pbeparm) {
00894
00895     Vacc *acc = VNULL;
00896     Vpbe *pbe = VNULL;
00897     Vgrid *grid = VNULL;
00898     Vatom *atoms = VNULL;
00899     Valist *alist = VNULL;
00900     double position[3], hx, hy, hzed, xmin, ymin, zmin;
00901     double grad[3], eps, epsp, epss, zmagic, u;
00902     int i, j, k, l, nx, ny, nz, ichop;
00903
00904     pbe = thee->pbe;
00905     acc = Vpbe_getVacc(pbe);
00906     nx = thee->pmgp->nx;
00907     ny = thee->pmgp->ny;
00908     nz = thee->pmgp->nz;
00909     hx = thee->pmgp->hx;
00910     hy = thee->pmgp->hy;
00911     hzed = thee->pmgp->hzed;
00912     xmin = thee->pmgp->xmin;
00913     ymin = thee->pmgp->ymin;
00914     zmin = thee->pmgp->zmin;
00915     epsp = Vpbe_getSoluteDiel(pbe);
00916     epss = Vpbe_getSolventDiel(pbe);
00917     zmagic = Vpbe_getZmagic(pbe);
00918
00919     if (!(thee->filled)) {
00920         Vnm_print(2, "Vpmg_fillArray: need to call Vpmg_fillco first!\n");
00921         return 0;
00922     }
00923
00924     switch (type) {
00925
00926     case VDT_CHARGE:
00927
00928         for (i=0; i<nx*ny*nz; i++) vec[i] = thee->charge[i]/zmagic;
00929         break;
00930
00931     case VDT_DIELX:
00932
00933         for (i=0; i<nx*ny*nz; i++) vec[i] = thee->epsx[i];
00934         break;
00935
00936     case VDT_DIELY:
00937
00938         for (i=0; i<nx*ny*nz; i++) vec[i] = thee->epsy[i];
00939         break;
00940

```

```

00941     case VDT_DIELZ:
00942
00943         for (i=0; i<nx*ny*nz; i++) vec[i] = thee->epsz[i];
00944         break;
00945
00946     case VDT_KAPPA:
00947
00948         for (i=0; i<nx*ny*nz; i++) vec[i] = thee->kappa[i];
00949         break;
00950
00951     case VDT_POT:
00952
00953         for (i=0; i<nx*ny*nz; i++) vec[i] = thee->u[i];
00954         break;
00955
00956     case VDT_ATOMPOT:
00957         alist = thee->pbe->alist;
00958         atoms = alist[pbeparm->molid-1].atoms;
00959         grid = Vgrid_ctor(nx, ny, nz, hx, hy,
00960                          hzed, xmin, ymin, zmin, thee->u);
00961         for (i=0; i<alist[pbeparm->molid-1].number; i++) {
00962             position[0] = atoms[i].position[0];
00963             position[1] = atoms[i].position[1];
00964             position[2] = atoms[i].position[2];
00965
00966             Vgrid_value(grid, position, &vec[i]);
00967         }
00968         Vgrid_dtor(&grid);
00969         break;
00970
00971     case VDT_SMOL:
00972
00973         for (k=0; k<nz; k++) {
00974             for (j=0; j<ny; j++) {
00975                 for (i=0; i<nx; i++) {
00976
00977                     position[0] = i*hx + xmin;
00978                     position[1] = j*hy + ymin;
00979                     position[2] = k*hzed + zmin;
00980
00981                     vec[IJK(i,j,k)] = (Vacc_molAcc(acc,position,parm));
00982                 }
00983             }
00984         }
00985         break;
00986
00987     case VDT_SSPL:
00988
00989         for (k=0; k<nz; k++) {
00990             for (j=0; j<ny; j++) {
00991                 for (i=0; i<nx; i++) {
00992
00993                     position[0] = i*hx + xmin;
00994                     position[1] = j*hy + ymin;
00995                     position[2] = k*hzed + zmin;
00996
00997                     vec[IJK(i,j,k)] = Vacc_splineAcc(acc,position,parm,0);
00998                 }
00999             }
01000         }
01001         break;
01002
01003     case VDT_VDW:
01004
01005         for (k=0; k<nz; k++) {
01006             for (j=0; j<ny; j++) {
01007                 for (i=0; i<nx; i++) {
01008
01009                     position[0] = i*hx + xmin;
01010                     position[1] = j*hy + ymin;
01011                     position[2] = k*hzed + zmin;
01012
01013                     vec[IJK(i,j,k)] = Vacc_vdwAcc(acc,position);
01014                 }
01015             }
01016         }
01017         break;
01018
01019     case VDT_IVDW:
01020
01021         for (k=0; k<nz; k++) {

```

```

01022         for (j=0; j<ny; j++) {
01023             for (i=0; i<nx; i++) {
01024
01025                 position[0] = i*hx + xmin;
01026                 position[1] = j*hy + ymin;
01027                 position[2] = k*hzed + zmin;
01028
01029                 vec[IJK(i,j,k)] = Vacc_ivdwAcc(acc,position,parm);
01030             }
01031         }
01032     }
01033     break;
01034
01035 case VDT_LAP:
01036
01037     grid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin,
01038         thee->u);
01039     for (k=0; k<nz; k++) {
01040         for (j=0; j<ny; j++) {
01041             for (i=0; i<nx; i++) {
01042
01043                 if ((k==0) || (k==(nz-1)) ||
01044                     (j==0) || (j==(ny-1)) ||
01045                     (i==0) || (i==(nx-1))) {
01046
01047                     vec[IJK(i,j,k)] = 0;
01048
01049                 } else {
01050                     position[0] = i*hx + xmin;
01051                     position[1] = j*hy + ymin;
01052                     position[2] = k*hzed + zmin;
01053                     VASSERT(Vgrid_curvature(grid,position, 1,
01054                         &(vec[IJK(i,j,k)])));
01055                 }
01056             }
01057         }
01058     }
01059     Vgrid_dtor(&grid);
01060     break;
01061
01062 case VDT_EDENS:
01063
01064     grid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin,
01065         thee->u);
01066     for (k=0; k<nz; k++) {
01067         for (j=0; j<ny; j++) {
01068             for (i=0; i<nx; i++) {
01069
01070                 position[0] = i*hx + xmin;
01071                 position[1] = j*hy + ymin;
01072                 position[2] = k*hzed + zmin;
01073                 VASSERT(Vgrid_gradient(grid, position, grad));
01074                 eps = epsp + (epss-epsp)*Vacc_molAcc(acc, position,
01075                     pbe->solventRadius);
01076                 vec[IJK(i,j,k)] = 0.0;
01077                 for (l=0; l<3; l++)
01078                     vec[IJK(i,j,k)] += eps*VSQR(grad[l]);
01079             }
01080         }
01081     }
01082     Vgrid_dtor(&grid);
01083     break;
01084
01085 case VDT_NDENS:
01086
01087     for (k=0; k<nz; k++) {
01088         for (j=0; j<ny; j++) {
01089             for (i=0; i<nx; i++) {
01090
01091                 position[0] = i*hx + xmin;
01092                 position[1] = j*hy + ymin;
01093                 position[2] = k*hzed + zmin;
01094                 vec[IJK(i,j,k)] = 0.0;
01095                 u = thee->u[IJK(i,j,k)];
01096                 if ( VABS(Vacc_ivdwAcc(acc,
01097                     position, pbe->maxIonRadius) - 1.0) < VSMALL) {
01098                     for (l=0; l<pbe->numIon; l++) {
01099                         double q = pbe->ionQ[l];
01100                         if (pbetype == PBE_NPBE || pbetype ==
01101                             PBE_SMPBE /* SMPBE Added */) {
01102                             vec[IJK(i,j,k)] += pbe->ionConc[l]*

```

```

Vcap_exp(-q*u, &ichop);
01102                                     } else if (pbetype == PBE_LPBE){
01103                                     vec[IJK(i,j,k)] += pbe->ionConc[l]*(1 - q*u + 0.5*q*q*u*u);
01104                                     }
01105                                     }
01106                                     }
01107                                     }
01108                                     }
01109                                     }
01110                                     break;
01111
01112     case VDT_QDENS:
01113         for (k=0; k<nz; k++) {
01114             for (j=0; j<ny; j++) {
01115                 for (i=0; i<nx; i++) {
01116                     position[0] = i*hx + xmin;
01117                     position[1] = j*hy + ymin;
01118                     position[2] = k*hzed + zmin;
01119                     vec[IJK(i,j,k)] = 0.0;
01120                     u = thee->u[IJK(i,j,k)];
01121                     if ( VABS(Vacc_ivdwAcc(acc,
01122                                     position, pbe->maxIonRadius) - 1.0) < VSMALL) {
01123                         for (l=0; l<pbe->numIon; l++) {
01124                             double q = pbe->ionQ[l];
01125                             if (pbetype == PBE_NPBE || pbetype == PBE_SMPBE /* SMPBE Added */) {
01126                                 } {
01127                                     vec[IJK(i,j,k)] += pbe->ionConc[l]*q*Vcap_exp(-q*u, &ichop);
01128                                 } else if (pbetype == PBE_LPBE) {
01129                                     vec[IJK(i,j,k)] += pbe->ionConc[l]*q*(1 - q*u + 0.5*q*q*u*u);
01130                                 }
01131                             }
01132                         }
01133                     }
01134                     break;
01135                 }
01136             }
01137             default:
01138                 Vnm_print(2, "main: Bogus data type (%d)!\n", type);
01139                 return 0;
01140                 break;
01141         }
01142     }
01143     return 1;
01144 }
01145
01146 }
01147
01148 VPRIVATE double Vpmg_polarizEnergy(Vpmg *thee,
01149                                     int extFlag
01150                                     ) {
01151
01152     int i,
01153         j,
01154         k,
01155         ijk,
01156         nx,
01157         ny,
01158         nz,
01159         iatom;
01160     double xmin,
01161         ymin,
01162         zmin,
01163         //x, // gcc: not used
01164         //y,
01165         //z,
01166         hx,
01167         hy,
01168         hzed,
01169         epsp,
01170         lap,
01171         pt[3],
01172         T,
01173         pre,
01174         polq,
01175         dist2,
01176         dist,
01177         energy,
01178         q,
01179         *charge,
01180         *pos,

```

```

01181         eps_w;
01182     Vgrid *potgrid;
01183     Vpbe *pbe;
01184     Valist *alist;
01185     Vatom *atom;
01186
01187     xmin = thee->pmgp->xmin;
01188     ymin = thee->pmgp->ymin;
01189     zmin = thee->pmgp->zmin;
01190     hx = thee->pmgp->hx;
01191     hy = thee->pmgp->hy;
01192     hzed = thee->pmgp->hzed;
01193     nx = thee->pmgp->nx;
01194     ny = thee->pmgp->ny;
01195     nz = thee->pmgp->nz;
01196     pbe = thee->pbe;
01197     epsp = Vpbe_getSoluteDiel(pbe);
01198     eps_w = Vpbe_getSolventDiel(pbe);
01199     alist = pbe->alist;
01200     charge = thee->charge;
01201
01202     /* Calculate the prefactor for Coulombic calculations */
01203     T = Vpbe_getTemperature(pbe);
01204     pre = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*
Vunit_kb*T);
01205     pre = pre*(1.0e10);
01206
01207     /* Set up Vgrid object with solution */
01208     potgrid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin, thee->
u);
01209
01210     /* Calculate polarization charge */
01211     energy = 0.0;
01212     for (i=1; i<(nx-1); i++) {
01213         pt[0] = xmin + hx*i;
01214         for (j=1; j<(ny-1); j++) {
01215             pt[1] = ymin + hy*j;
01216             for (k=1; k<(nz-1); k++) {
01217                 pt[2] = zmin + hzed*k;
01218
01219                 /* Calculate polarization charge */
01220                 VASSERT(Vgrid_curvature(potgrid, pt, 1, &lap));
01221                 ijk = IJK(i,j,k);
01222                 polq = charge[ijk] + epsp*lap*3.0;
01223
01224                 /* Calculate interaction energy with atoms */
01225                 if (VABS(polq) > VSMALL) {
01226                     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
01227                         atom = Valist_getAtom(alist, iatom);
01228                         q = Vatom_getCharge(atom);
01229                         pos = Vatom_getPosition(atom);
01230                         dist2 = VSQR(pos[0]-pt[0]) + VSQR(pos[1]-pt[1]) \
+ VSQR(pos[2]-pt[2]);
01231                         dist = VSQRT(dist2);
01232
01233                         if (dist < VSMALL) {
01234                             Vnm_print(2, "Vpmg_polarizEnergy: atom on grid point; ignoring!\n");
01235                         } else {
01236                             energy = energy + polq*q/dist;
01237                         }
01238                     }
01239                 }
01240             }
01241         }
01242     }
01243 }
01244
01245     return pre+energy;
01246 }
01247
01248 VPUBLIC double Vpmg_energy(Vpmg *thee,
01249     int extFlag
01250 ) {
01251
01252     double totEnergy = 0.0,
01253         dielEnergy = 0.0,
01254         qmEnergy = 0.0,
01255         qfEnergy = 0.0;
01256
01257     VASSERT(thee != VNULL);
01258
01259     if ((thee->pmgp->nonlin) && (Vpbe_getBulkIonicStrength(thee->

```

```

pbe) > 0.)) {
01260     Vnm_print(0, "Vpmg_energy: calculating full PBE energy\n");
01261     qmEnergy = Vpmg_qmEnergy(thee, extFlag);
01262     Vnm_print(0, "Vpmg_energy: qmEnergy = %1.12E kT\n", qmEnergy);
01263     qfEnergy = Vpmg_qfEnergy(thee, extFlag);
01264     Vnm_print(0, "Vpmg_energy: qfEnergy = %1.12E kT\n", qfEnergy);
01265     dielEnergy = Vpmg_dielEnergy(thee, extFlag);
01266     Vnm_print(0, "Vpmg_energy: dielEnergy = %1.12E kT\n", dielEnergy);
01267     totEnergy = qfEnergy - dielEnergy - qmEnergy;
01268 } else {
01269     Vnm_print(0, "Vpmg_energy: calculating only q-phi energy\n");
01270     qfEnergy = Vpmg_qfEnergy(thee, extFlag);
01271     Vnm_print(0, "Vpmg_energy: qfEnergy = %1.12E kT\n", qfEnergy);
01272     totEnergy = 0.5*qfEnergy;
01273 }
01274
01275 return totEnergy;
01276
01277 }
01278
01279 VPUBLIC double Vpmg_dielEnergy(Vpmg *thee,
01280                                int extFlag
01281                                ) {
01282
01283     double hx,
01284            hy,
01285            hzed,
01286            energy,
01287            nrgx,
01288            nrgy,
01289            nrgz,
01290            pvecx,
01291            pvecy,
01292            pvecz;
01293
01294     int i,
01295         j,
01296         k,
01297         nx,
01298         ny,
01299         nz;
01300
01301     VASSERT(thee != VNULL);
01302
01303     /* Get the mesh information */
01304     nx = thee->pmg->nx;
01305     ny = thee->pmg->ny;
01306     nz = thee->pmg->nz;
01307     hx = thee->pmg->hx;
01308     hy = thee->pmg->hy;
01309     hzed = thee->pmg->hzed;
01310
01311     energy = 0.0;
01312
01313     if (!thee->filled) {
01314         Vnm_print(2, "Vpmg_dielEnergy: Need to call Vpmg_fillco!\n");
01315         VASSERT(0);
01316     }
01317
01318     for (k=0; k<(nz-1); k++) {
01319         for (j=0; j<(ny-1); j++) {
01320             for (i=0; i<(nx-1); i++) {
01321                 pvecx = 0.5*(thee->pvec[IJK(i, j, k)]+thee->pvec[IJK(i+1, j, k)]);
01322                 pvecy = 0.5*(thee->pvec[IJK(i, j, k)]+thee->pvec[IJK(i, j+1, k)]);
01323                 pvecz = 0.5*(thee->pvec[IJK(i, j, k)]+thee->pvec[IJK(i, j, k+1)]);
01324                 nrgx = thee->epsx[IJK(i, j, k)]*pvecx
01325                     * VSQR((thee->u[IJK(i, j, k)]-thee->u[IJK(i+1, j, k)]) /hx);
01326                 nrgy = thee->epsy[IJK(i, j, k)]*pvecy
01327                     * VSQR((thee->u[IJK(i, j, k)]-thee->u[IJK(i, j+1, k)]) /hy);
01328                 nrgz = thee->epsz[IJK(i, j, k)]*pvecz
01329                     * VSQR((thee->u[IJK(i, j, k)]-thee->u[IJK(i, j, k+1)]) /hzed);
01330                 energy += (nrgx + nrgy + nrgz);
01331             }
01332         }
01333     }
01334
01335     energy = 0.5*energy*hx*hy*hzed;
01336     energy = energy/Vpbe_getZmagic(thee->pbe);
01337
01338     if (extFlag == 1) energy += (thee->extDiEnergy);
01339
01340     return energy;

```

```

01340 }
01341
01342 VPUBLIC double Vpmg_dielGradNorm(Vpmg *thee) {
01343
01344     double hx, hy, hzed, energy, nrgx, nrgy, nrgz, pvecx, pvecy, pvecz;
01345     int i, j, k, nx, ny, nz;
01346
01347     VASSERT(thee != VNULL);
01348
01349     /* Get the mesh information */
01350     nx = thee->pmgp->nx;
01351     ny = thee->pmgp->ny;
01352     nz = thee->pmgp->nz;
01353     hx = thee->pmgp->hx;
01354     hy = thee->pmgp->hy;
01355     hzed = thee->pmgp->hzed;
01356
01357     energy = 0.0;
01358
01359     if (!thee->filled) {
01360         Vnm_print(2, "Vpmg_dielGradNorm: Need to call Vpmg_fillco!\n");
01361         VASSERT(0);
01362     }
01363
01364     for (k=1; k<nz; k++) {
01365         for (j=1; j<ny; j++) {
01366             for (i=1; i<nx; i++) {
01367                 pvecx = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i-1,j,k)]);
01368                 pvecy = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j-1,k)]);
01369                 pvecz = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j,k-1)]);
01370                 nrgx = pvecx
01371                     * VSQR((thee->epsx[IJK(i,j,k)]-thee->epsx[IJK(i-1,j,k)])/(hx));
01372                 nrgy = pvecy
01373                     * VSQR((thee->epsy[IJK(i,j,k)]-thee->epsy[IJK(i,j-1,k)])/(hy));
01374                 nrgz = pvecz
01375                     * VSQR((thee->epsz[IJK(i,j,k)]-thee->epsz[IJK(i,j,k-1)])/(hzed));
01376                 energy += VSQRT(nrgx + nrgy + nrgz);
01377             }
01378         }
01379     }
01380
01381     energy = energy*hx*hy*hzed;
01382
01383     return energy;
01384 }
01385
01386 VPUBLIC double Vpmg_qmEnergy(Vpmg *thee,
01387                             int extFlag
01388                             ) {
01389
01390     double energy;
01391
01392     if (thee->pbe->ipkey == IPKEY_SMPBE) {
01393         energy = Vpmg_qmEnergySMPBE(thee, extFlag);
01394     } else {
01395         energy = Vpmg_qmEnergyNONLIN(thee, extFlag);
01396     }
01397
01398     return energy;
01399 }
01400
01401 VPRIVATE double Vpmg_qmEnergyNONLIN(Vpmg *thee,
01402                                     int extFlag
01403                                     ) {
01404
01405     double hx,
01406            hy,
01407            hzed,
01408            energy,
01409            ionConc[MAXION],
01410            ionRadii[MAXION],
01411            ionQ[MAXION],
01412            zkappa2,
01413            ionstr,
01414            zks2;
01415     int i, /* Loop variable */
01416         j,
01417         nx,
01418         ny,
01419         nz,
01420         nion,

```

```

01421         ichop,
01422         nchop,
01423         len; /* Stores number of iterations for loops to avoid multiple recalculations */
01424
01425     VASSERT(thee != VNULL);
01426
01427     /* Get the mesh information */
01428     nx = thee->pmgp->nx;
01429     ny = thee->pmgp->ny;
01430     nz = thee->pmgp->nz;
01431     hx = thee->pmgp->hx;
01432     hy = thee->pmgp->hy;
01433     hzed = thee->pmgp->hzed;
01434     zkappa2 = Vpbe_getZkappa2(thee->pbe);
01435     ionstr = Vpbe_getBulkIonicStrength(thee->pbe);
01436
01437     /* Bail if we're at zero ionic strength */
01438     if (zkappa2 < VSMALL) {
01439
01440     #ifndef VAPBSQUIET
01441         Vnm_print(0, "Vpmg_qmEnergy: Zero energy for zero ionic strength!\n");
01442     #endif
01443
01444         return 0.0;
01445     }
01446     zks2 = 0.5*zkappa2/ionstr;
01447
01448     if (!thee->filled) {
01449         Vnm_print(2, "Vpmg_qmEnergy: Need to call Vpmg_fillco()!\n");
01450         VASSERT(0);
01451     }
01452
01453     energy = 0.0;
01454     nchop = 0;
01455     Vpbe_getIons(thee->pbe, &nion, ionConc, ionRadii, ionQ);
01456     if (thee->pmgp->nonlin) {
01457         Vnm_print(0, "Vpmg_qmEnergy: Calculating nonlinear energy\n");
01458         for (i=0, len=nx*ny*nz; i<len; i++) {
01459             if (thee->pvec[i]*thee->kappa[i] > VSMALL) {
01460                 for (j=0; j<nion; j++) {
01461                     energy += (thee->pvec[i]*thee->kappa[i]*zks2
01462                             * ionConc[j]
01463                             * (Vcap_exp(-ionQ[j]*thee->u[i], &ichop)-1.0));
01464                     nchop += ichop;
01465                 }
01466             }
01467         }
01468         if (nchop > 0) {
01469             Vnm_print(2, "Vpmg_qmEnergy: Chopped EXP %d times!\n",nchop);
01470             Vnm_print(2, "\nERROR! Detected large potential values in energy evaluation! \nERROR! This
calculation failed -- please report to the APBS developers!\n\n");
01471             VASSERT(0);
01472         }
01473     } else {
01474         /* Zkappa2 OK here b/c LPBE approx */
01475         Vnm_print(0, "Vpmg_qmEnergy: Calculating linear energy\n");
01476         for (i=0, len=nx*ny*nz; i<len; i++) {
01477             if (thee->pvec[i]*thee->kappa[i] > VSMALL)
01478                 energy += (thee->pvec[i]*zkappa2*thee->kappa[i]*VSQR(thee->
u[i]));
01479         }
01480         energy = 0.5*energy;
01481     }
01482     energy = energy*hx*hy*hzed;
01483     energy = energy/Vpbe_getZmagic(thee->pbe);
01484
01485     if (extFlag == 1) energy += thee->extQmEnergy;
01486
01487     return energy;
01488 }
01489
01490 VPUBLIC double Vpmg_qmEnergySMPBE(Vpmg *thee,
01491                                 int extFlag
01492                                 ) {
01493
01494     double hx,
01495            hy,
01496            hzed,
01497            energy,
01498            ionConc[MAXION],
01499            ionRadii[MAXION],

```



```

01500         ionQ[MAXION],
01501         zkappa2,
01502         ionstr,
01503         zks2;
01504     int i,
01505         //j, // gcc: not used
01506         nx,
01507         ny,
01508         nz,
01509         nion,
01510         //ichop, // gcc: not used
01511         nchop,
01512         len; /* Loop variable */
01513
01514     /* SMPB Modification (vchu, 09/21/06) */
01515     /* variable declarations for SMPB energy terms */
01516     double a,
01517         k,
01518         z1,
01519         z2,
01520         z3,
01521         cb1,
01522         cb2,
01523         cb3,
01524         a1,
01525         a2,
01526         a3,
01527         c1,
01528         c2,
01529         c3,
01530         currEnergy,
01531         fracOccA,
01532         fracOccB,
01533         fracOccC,
01534         phi,
01535         gpark,
01536         denom;
01537     // Na; /* @todo remove if no conflicts are caused - This constant is already defined in
vpde.h. no need to redefine. */
01538     int ichop1,
01539         ichop2,
01540         ichop3;
01541
01542     VASSERT(thee != VNULL);
01543
01544     /* Get the mesh information */
01545     nx = thee->pmgp->nx;
01546     ny = thee->pmgp->ny;
01547     nz = thee->pmgp->nz;
01548     hx = thee->pmgp->hx;
01549     hy = thee->pmgp->hy;
01550     hzed = thee->pmgp->hzed;
01551     zkappa2 = Vpbe_getZkappa2(thee->pbe);
01552     ionstr = Vpbe_getBulkIonicStrength(thee->pbe);
01553
01554     /* Bail if we're at zero ionic strength */
01555     if (zkappa2 < VSMALL) {
01556
01557     #ifndef VAPBSQUIET
01558         Vnm_print(0, "Vpmg_qmEnergySMPBE: Zero energy for zero ionic strength!\n");
01559     #endif
01560
01561         return 0.0;
01562     }
01563     zks2 = 0.5*zkappa2/ionstr;
01564
01565     if (!thee->filled) {
01566         Vnm_print(2, "Vpmg_qmEnergySMPBE: Need to call Vpmg_fillco()!\n");
01567         VASSERT(0);
01568     }
01569
01570     energy = 0.0;
01571     nchop = 0;
01572     Vpbe_getIons(thee->pbe, &nion, ionConc, ionRadii, ionQ);
01573
01574     /* SMPB Modification (vchu, 09/21/06) */
01575     /* Extensive modification to the first part of the if statement
01576        where that handles the thee->pmgp->nonlin part. Basically, I've
01577        deleted all of the original code and written my own code that computes
01578        the electrostatic free energy in the SMPB framework. Definitely really hacky
01579        at this stage of the game, but gets the job done. The second part of the

```

```

01580         if statement (the part that handles linear poisson-boltzmann) has been deleted
01581         because there will be no linearized SMPB energy.. */
01582
01583     z1 = ionQ[0];
01584     z2 = ionQ[1];
01585     z3 = ionQ[2];
01586     cb1 = ionConc[0];
01587     cb2 = ionConc[1];
01588     cb3 = ionConc[2];
01589     a = thee->pbe->smvolume;
01590     k = thee->pbe->smsize;
01591
01592     // This constant is defined in vpde.h Do not need to redefine
01593     //Na = 6.022045000e-04; /* Converts from Molar to N/A^3 */
01594
01595     fracOccA = Na*cb1*VCUB(a);
01596     fracOccB = Na*cb2*VCUB(a);
01597     fracOccC = Na*cb3*VCUB(a);
01598
01599     phi = (fracOccA/k) + fracOccB + fracOccC;
01600
01601     if (thee->pmgp->nonlin) {
01602         Vnm_print(0, "Vpmg_qmEnergySMPBE: Calculating nonlinear energy using SMPB functional!\n");
01603         for (i=0, len=nx*ny*nz; i<len; i++) {
01604             if ((k-1) > VSMALL) && (thee->pvec[i]*thee->kappa[i] > VSMALL)) {
01605
01606                 a1 = Vcap_exp(-1.0*z1*thee->u[i], &ichop1);
01607                 a2 = Vcap_exp(-1.0*z2*thee->u[i], &ichop2);
01608                 a3 = Vcap_exp(-1.0*z3*thee->u[i], &ichop3);
01609
01610                 nchop += ichop1 + ichop2 + ichop3;
01611
01612                 gpark = (1 - phi + (fracOccA/k)*a1);
01613                 denom = VPOW(gpark, k) + VPOW(1-fracOccB-fracOccC, k-1)*(fracOccB*a2+fracOccC*a3);
01614
01615                 if (cb1 > VSMALL) {
01616                     c1 = Na*cb1*VPOW(gpark, k-1)*a1/denom;
01617                     if(c1 != c1) c1 = 0.;
01618                 } else c1 = 0.;
01619
01620                 if (cb2 > VSMALL) {
01621                     c2 = Na*cb2*VPOW(1-fracOccB-fracOccC,k-1)*a2/denom;
01622                     if(c2 != c2) c2 = 0.;
01623                 } else c2 = 0.;
01624
01625                 if (cb3 > VSMALL) {
01626                     c3 = Na*cb3*VPOW(1-fracOccB-fracOccC,k-1)*a3/denom;
01627                     if(c3 != c3) c3 = 0.;
01628                 } else c3 = 0.;
01629
01630                 currEnergy = k*VLOG((1-(c1*VCUB(a)/k)-c2*VCUB(a)-c3*VCUB(a))/(1-phi))
01631                     -(k-1)*VLOG((1-c2*VCUB(a)-c3*VCUB(a))/(1-phi+(fracOccA/k)));
01632
01633                 energy += thee->pvec[i]*thee->kappa[i]*currEnergy;
01634             } else if (thee->pvec[i]*thee->kappa[i] > VSMALL){
01635
01636                 a1 = Vcap_exp(-1.0*z1*thee->u[i], &ichop1);
01637                 a2 = Vcap_exp(-1.0*z2*thee->u[i], &ichop2);
01638                 a3 = Vcap_exp(-1.0*z3*thee->u[i], &ichop3);
01639
01640                 nchop += ichop1 + ichop2 + ichop3;
01641
01642                 gpark = (1 - phi + (fracOccA)*a1);
01643                 denom = gpark + (fracOccB*a2+fracOccC*a3);
01644
01645                 if (cb1 > VSMALL) {
01646                     c1 = Na*cb1*a1/denom;
01647                     if(c1 != c1) c1 = 0.;
01648                 } else c1 = 0.;
01649
01650                 if (cb2 > VSMALL) {
01651                     c2 = Na*cb2*a2/denom;
01652                     if(c2 != c2) c2 = 0.;
01653                 } else c2 = 0.;
01654
01655                 if (cb3 > VSMALL) {
01656                     c3 = Na*cb3*a3/denom;
01657                     if(c3 != c3) c3 = 0.;
01658                 } else c3 = 0.;
01659             }
01660         }
01661     }

```

```

01662             currEnergy = VLOG((1-c1*VCUB(a)-c2*VCUB(a)-c3*VCUB(a))/(1-fracOccA-fracOccB-fracOccC));
01663
01664             energy += thee->pvec[i]*thee->kappa[i]*currEnergy;
01665         }
01666     }
01667
01668     energy = -energy/VCUB(a);
01669
01670     if (nchop > 0) Vnm_print(2, "Vpmg_qmEnergySMPBE: Chopped EXP %d times!\n",
01671                             nchop);
01672
01673     } else {
01674         /* Zkappa2 OK here b/c LPBE approx */
01675         Vnm_print(0, "Vpmg_qmEnergySMPBE: ERROR: NO LINEAR ENERGY!! Returning 0!\n");
01676
01677         energy = 0.0;
01678     }
01679
01680     energy = energy*hx*hy*hzed;
01681
01682     if (extFlag == 1) energy += thee->extQmEnergy;
01683
01684     return energy;
01685 }
01686
01687 VPUBLIC double Vpmg_qfEnergy(Vpmg *thee,
01688                             int extFlag
01689                             ) {
01690
01691     double energy = 0.0;
01692
01693     VASSERT(thee != VNULL);
01694
01695     if ((thee->useChargeMap) || (thee->chargeMeth ==
01696     VCM_BSPL2)) {
01697         energy = Vpmg_qfEnergyVolume(thee, extFlag);
01698     } else {
01699         energy = Vpmg_qfEnergyPoint(thee, extFlag);
01700     }
01701
01702     return energy;
01703 }
01704
01705 VPRIVATE double Vpmg_qfEnergyPoint(Vpmg *thee,
01706                                    int extFlag
01707                                    ) {
01708
01709     int iatom, nx, ny, nz, ihi, ilo, jhi, jlo, khi, klo;
01710     double xmax, ymax, zmax, xmin, ymin, zmin, hx, hy, hzed, ifloat, jfloat;
01711     double charge, kfloat, dx, dy, dz, energy, uval, *position;
01712     double *u;
01713     double *pvec;
01714     Valist *alist;
01715     Vatom *atom;
01716     Vpbe *pbe;
01717
01718     pbe = thee->pbe;
01719     alist = pbe->alist;
01720     VASSERT(alist != VNULL);
01721
01722     /* Get the mesh information */
01723     nx = thee->pmgp->nx;
01724     ny = thee->pmgp->ny;
01725     nz = thee->pmgp->nz;
01726     hx = thee->pmgp->hx;
01727     hy = thee->pmgp->hy;
01728     hzed = thee->pmgp->hzed;
01729     xmax = thee->pmgp->xmax;
01730     ymax = thee->pmgp->ymax;
01731     zmax = thee->pmgp->zmax;
01732     xmin = thee->pmgp->xmin;
01733     ymin = thee->pmgp->ymin;
01734     zmin = thee->pmgp->zmin;
01735
01736     u = thee->u;
01737     pvec = thee->pvec;
01738
01739     energy = 0.0;
01740
01741     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {

```

```

01742     /* Get atomic information */
01743     atom = Valist_getAtom(alist, iatom);
01744
01745     position = Vatom_getPosition(atom);
01746     charge = Vatom_getCharge(atom);
01747
01748     /* Figure out which vertices we're next to */
01749     ifloat = (position[0] - xmin)/hx;
01750     jfloat = (position[1] - ymin)/hy;
01751     kfloat = (position[2] - zmin)/hz;
01752     ihi = (int)ceil(ifloat);
01753     ilo = (int)floor(ifloat);
01754     jhi = (int)ceil(jfloat);
01755     jlo = (int)floor(jfloat);
01756     khi = (int)ceil(kfloat);
01757     klo = (int)floor(kfloat);
01758
01759     if (atom->partID > 0) {
01760
01761         if ((ihi<nx) && (jhi<ny) && (khi<nz) &&
01762             (ilo>=0) && (jlo>=0) && (klo>=0)) {
01763
01764             /* Now get trilinear interpolation constants */
01765             dx = ifloat - (double)(ilo);
01766             dy = jfloat - (double)(jlo);
01767             dz = kfloat - (double)(klo);
01768             uval =
01769                 dx*dy*dz*u[IJK(ihi, jhi, khi)]
01770                 + dx*(1.0-dy)*dz*u[IJK(ihi, jlo, khi)]
01771                 + dx*dy*(1.0-dz)*u[IJK(ihi, jhi, klo)]
01772                 + dx*(1.0-dy)*(1.0-dz)*u[IJK(ihi, jlo, klo)]
01773                 + (1.0-dx)*dy*dz*u[IJK(ilo, jhi, khi)]
01774                 + (1.0-dx)*(1.0-dy)*dz*u[IJK(ilo, jlo, khi)]
01775                 + (1.0-dx)*dy*(1.0-dz)*u[IJK(ilo, jhi, klo)]
01776                 + (1.0-dx)*(1.0-dy)*(1.0-dz)*u[IJK(ilo, jlo, klo)];
01777             energy += (uval*charge*atom->partID);
01778         } else if (three->pmgp->bcfl != BCFL_FOCUS) {
01779             Vnm_print(2, "Vpmg_qfEnergy: Atom #%d at (%4.3f, %4.3f, \
01780 %4.3f) is off the mesh (ignoring)!\n",
01781                 iatom, position[0], position[1], position[2]);
01782         }
01783     }
01784 }
01785
01786 if (extFlag) energy += three->extQfEnergy;
01787
01788 return energy;
01789 }
01790
01791 VPUBLIC double Vpmg_qfAtomEnergy(Vpmg *three, Vatom *atom) {
01792
01793     int nx, ny, nz, ihi, ilo, jhi, jlo, khi, klo;
01794     double xmax, xmin, ymax, ymin, zmax, zmin, hx, hy, hz, ifloat, jfloat;
01795     double charge, kfloat, dx, dy, dz, energy, uval, *position;
01796     double *u;
01797
01798
01799     /* Get the mesh information */
01800     nx = three->pmgp->nx;
01801     ny = three->pmgp->ny;
01802     nz = three->pmgp->nz;
01803     hx = three->pmgp->hx;
01804     hy = three->pmgp->hy;
01805     hz = three->pmgp->hz;
01806     xmax = three->xf[nx-1];
01807     ymax = three->yf[ny-1];
01808     zmax = three->zf[nz-1];
01809     xmin = three->xf[0];
01810     ymin = three->yf[0];
01811     zmin = three->zf[0];
01812
01813     u = three->u;
01814
01815     energy = 0.0;
01816
01817
01818     position = Vatom_getPosition(atom);
01819     charge = Vatom_getCharge(atom);
01820
01821     /* Figure out which vertices we're next to */
01822     ifloat = (position[0] - xmin)/hx;

```

```

01823     jfloat = (position[1] - ymin)/hy;
01824     kfloat = (position[2] - zmin)/hz;
01825     ihi = (int)ceil(ifloat);
01826     ilo = (int)floor(ifloat);
01827     jhi = (int)ceil(jfloat);
01828     jlo = (int)floor(jfloat);
01829     khi = (int)ceil(kfloat);
01830     klo = (int)floor(kfloat);
01831
01832     if (atom->partID > 0) {
01833
01834         if ((ihi<nx) && (jhi<ny) && (khi<nz) &&
01835             (ilo>=0) && (jlo>=0) && (klo>=0)) {
01836
01837             /* Now get trilinear interpolation constants */
01838             dx = ifloat - (double)(ilo);
01839             dy = jfloat - (double)(jlo);
01840             dz = kfloat - (double)(klo);
01841             uval =
01842                 dx*dy*dz*u[IJK(ihi,jhi,khi)]
01843                 + dx*(1.0-dy)*dz*u[IJK(ihi,jlo,khi)]
01844                 + dx*dy*(1.0-dz)*u[IJK(ihi,jhi,klo)]
01845                 + dx*(1.0-dy)*(1.0-dz)*u[IJK(ihi,jlo,klo)]
01846                 + (1.0-dx)*dy*dz*u[IJK(ilo,jhi,khi)]
01847                 + (1.0-dx)*(1.0-dy)*dz*u[IJK(ilo,jlo,khi)]
01848                 + (1.0-dx)*dy*(1.0-dz)*u[IJK(ilo,jhi,klo)]
01849                 + (1.0-dx)*(1.0-dy)*(1.0-dz)*u[IJK(ilo,jlo,klo)];
01850             energy += (uval*charge*atom->partID);
01851         } else if (thee->pmgp->bcfl != BCFL_FOCUS) {
01852             Vnm_print(2, "Vpmg_qfAtomEnergy: Atom at (%4.3f, %4.3f, \
01853 %4.3f) is off the mesh (ignoring)!\n",
01854                 position[0], position[1], position[2]);
01855         }
01856     }
01857
01858     return energy;
01859 }
01860
01861 VPRIVATE double Vpmg_qfEnergyVolume(Vpmg *thee, int extFlag) {
01862     double hx, hy, hz, energy;
01863     int i, nx, ny, nz;
01864
01865     VASSERT(thee != VNULL);
01866
01867     /* Get the mesh information */
01868     nx = thee->pmgp->nx;
01869     ny = thee->pmgp->ny;
01870     nz = thee->pmgp->nz;
01871     hx = thee->pmgp->hx;
01872     hy = thee->pmgp->hy;
01873     hz = thee->pmgp->hz;
01874
01875     if (!thee->filled) {
01876         Vnm_print(2, "Vpmg_qfEnergyVolume: need to call Vpmg_fillco!\n");
01877         VASSERT(0);
01878     }
01879
01880     energy = 0.0;
01881     Vnm_print(0, "Vpmg_qfEnergyVolume: Calculating energy\n");
01882     for (i=0; i<(nx*ny*nz); i++) {
01883         energy += (thee->pvec[i]*thee->u[i]*thee->charge[i]);
01884     }
01885     energy = energy*hx*hy*hz/Vpbe_getZmagic(thee->pbe);
01886
01887     if (extFlag == 1) energy += thee->extQfEnergy;
01888
01889     return energy;
01890 }
01891
01892 VPRIVATE void Vpmg_splineSelect(int srfm, Vacc *acc, double *gpos, double win,
01893     double infrad, Vatom *atom, double *force) {
01894
01895     switch (srfm) {
01896     case VSM_SPLINE :
01897         Vacc_splineAccGradAtomNorm(acc, gpos, win, infrad, atom, force);
01898         break;
01899     case VSM_SPLINE3:
01900         Vacc_splineAccGradAtomNorm3(acc, gpos, win, infrad, atom, force);
01901         break;
01902     case VSM_SPLINE4 :

```

```

01904         Vacc_splineAccGradAtomNorm4(acc, gpos, win, infrad, atom, force);
01905         break;
01906     default:
01907         Vnm_print(2, "Vpmg_dbnbForce: Unknown surface method.\n");
01908         return;
01909     }
01910 }
01911 return;
01912 }
01913
01914 VPRIVATE void focusFillBound(Vpmg *thee,
01915                             Vpmg *pmgOLD
01916                             ) {
01917     Vpbe *pbe;
01918     double hxOLD,
01919           hyOLD,
01920           hzOLD,
01921           xminOLD,
01922           yminOLD,
01923           zminOLD,
01924           xmaxOLD,
01925           ymaxOLD,
01926           zmaxOLD,
01927           hxNEW,
01928           hyNEW,
01929           hzNEW,
01930           xminNEW,
01931           yminNEW,
01932           zminNEW,
01933           xmaxNEW,
01934           ymaxNEW,
01935           zmaxNEW,
01936           x,
01937           y,
01938           z,
01939           dx,
01940           dy,
01941           dz,
01942           ifloat,
01943           jfloat,
01944           kfloat,
01945           uval,
01946           eps_w,
01947           T,
01948           prel,
01949           xkappa,
01950           size,
01951           *apos,
01952           charge,
01953           //pos[3], // gcc: not used
01954           uvalMin,
01955           uvalMax,
01956           *data;
01957     int nxOLD,
01958         nyOLD,
01959         nzOLD,
01960         nxNEW,
01961         nyNEW,
01962         nzNEW,
01963         i,
01964         j,
01965         k,
01966         ihi,
01967         ilo,
01968         jhi,
01969         jlo,
01970         khi,
01971         klo,
01972         nx,
01973         ny,
01974         nz;
01975
01976     /* Calculate new problem dimensions */
01977     hxNEW = thee->pmgp->hx;
01978     hyNEW = thee->pmgp->hy;
01979     hzNEW = thee->pmgp->hz;
01980     nx = thee->pmgp->nx;
01981     ny = thee->pmgp->ny;
01982     nz = thee->pmgp->nz;
01983     nxNEW = thee->pmgp->nx;

```

```

01985     nyNEW = thee->pmgp->ny;
01986     nzNEW = thee->pmgp->nz;
01987     xminNEW = thee->pmgp->xcent - ((double) (nxNEW-1)*hxNEW)/2.0;
01988     xmaxNEW = thee->pmgp->xcent + ((double) (nxNEW-1)*hxNEW)/2.0;
01989     yminNEW = thee->pmgp->ycent - ((double) (nyNEW-1)*hyNEW)/2.0;
01990     ymaxNEW = thee->pmgp->ycent + ((double) (nyNEW-1)*hyNEW)/2.0;
01991     zminNEW = thee->pmgp->zcent - ((double) (nzNEW-1)*hzNEW)/2.0;
01992     zmaxNEW = thee->pmgp->zcent + ((double) (nzNEW-1)*hzNEW)/2.0;
01993
01994     if (pmgOLD != VNULL) {
01995         /* Relevant old problem parameters */
01996         hxOLD = pmgOLD->pmgp->hx;
01997         hyOLD = pmgOLD->pmgp->hy;
01998         hzOLD = pmgOLD->pmgp->hz;
01999         nxOLD = pmgOLD->pmgp->nx;
02000         nyOLD = pmgOLD->pmgp->ny;
02001         nzOLD = pmgOLD->pmgp->nz;
02002         xminOLD = pmgOLD->pmgp->xcent - ((double) (nxOLD-1)*hxOLD)/2.0;
02003         xmaxOLD = pmgOLD->pmgp->xcent + ((double) (nxOLD-1)*hxOLD)/2.0;
02004         yminOLD = pmgOLD->pmgp->ycent - ((double) (nyOLD-1)*hyOLD)/2.0;
02005         ymaxOLD = pmgOLD->pmgp->ycent + ((double) (nyOLD-1)*hyOLD)/2.0;
02006         zminOLD = pmgOLD->pmgp->zcent - ((double) (nzOLD-1)*hzOLD)/2.0;
02007         zmaxOLD = pmgOLD->pmgp->zcent + ((double) (nzOLD-1)*hzOLD)/2.0;
02008
02009         data = pmgOLD->u;
02010     } else {
02011         /* Relevant old problem parameters */
02012         hxOLD = thee->potMap->hx;
02013         hyOLD = thee->potMap->hy;
02014         hzOLD = thee->potMap->hz;
02015         nxOLD = thee->potMap->nx;
02016         nyOLD = thee->potMap->ny;
02017         nzOLD = thee->potMap->nz;
02018         xminOLD = thee->potMap->xmin;
02019         xmaxOLD = thee->potMap->xmax;
02020         yminOLD = thee->potMap->ymin;
02021         ymaxOLD = thee->potMap->ymax;
02022         zminOLD = thee->potMap->zmin;
02023         zmaxOLD = thee->potMap->zmax;
02024
02025         data = thee->potMap->data;
02026     }
02027     /* BOUNDARY CONDITION SETUP FOR POINTS OFF OLD MESH:
02028     * For each "atom" (only one for bcfl=1), we use the following formula to
02029     * calculate the boundary conditions:
02030     * 
$$g(x) = \frac{q e_c}{4\pi\epsilon_0\epsilon_w k_b T} \frac{\exp(-\kappa(d-a))}{1+\kappa a}$$

02031     * 
$$\frac{1}{d}$$

02032     * where  $d = ||x - x_0||$  (in m) and  $a$  is the size of the atom (in m).
02033     * We only need to evaluate some of these prefactors once:
02034     * 
$$\text{prel} = \frac{e_c}{4\pi\epsilon_0\epsilon_w k_b T}$$

02035     * which gives the potential as
02036     * 
$$g(x) = \text{prel} * q/d * \frac{\exp(-\kappa(d-a))}{1+\kappa a}$$

02037     */
02038     pbe = thee->pbe;
02039     eps_w = Vpbe_getSolventDiel(pbe); /* Dimensionless */
02040     T = Vpbe_getTemperature(pbe); /* K */
02041     prel = (Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T);
02042
02043     /* Finally, if we convert keep  $\kappa$  in  $\text{\AA}^{-1}$  and scale prel by
02044     * m/ $\text{\AA}$ , then we will only need to deal with distances and sizes in
02045     * Angstroms rather than meters. */
02046     xkappa = Vpbe_getXkappa(pbe); /*  $\text{\AA}^{-1}$  */
02047     prel = prel*(1.0e10);
02048     size = Vpbe_getSoluteRadius(pbe);
02049     apos = Vpbe_getSoluteCenter(pbe);
02050     charge = Vunit_ec*Vpbe_getSoluteCharge(pbe);
02051
02052     /* Check for rounding error */
02053     if (VABS(xminOLD-xminNEW) < VSMALL) xminNEW = xminOLD;
02054     if (VABS(xmaxOLD-xmaxNEW) < VSMALL) xmaxNEW = xmaxOLD;
02055     if (VABS(yminOLD-yminNEW) < VSMALL) yminNEW = yminOLD;
02056     if (VABS(ymaxOLD-ymaxNEW) < VSMALL) ymaxNEW = ymaxOLD;
02057     if (VABS(zminOLD-zminNEW) < VSMALL) zminNEW = zminOLD;
02058     if (VABS(zmaxOLD-zmaxNEW) < VSMALL) zmaxNEW = zmaxOLD;
02059
02060     /* Sanity check: make sure we're within the old mesh */
02061     Vnm_print(0, "VPMG::focusFillBound -- New mesh mins = %g, %g, %g\n",
02062         xminNEW, yminNEW, zminNEW);
02063     Vnm_print(0, "VPMG::focusFillBound -- New mesh maxs = %g, %g, %g\n",

```

```

02066         xmaxNEW, ymaxNEW, zmaxNEW);
02067 Vnm_print(0, "VPMG::focusFillBound -- Old mesh mins = %g, %g, %g\n",
02068           xminOLD, yminOLD, zminOLD);
02069 Vnm_print(0, "VPMG::focusFillBound -- Old mesh maxs = %g, %g, %g\n",
02070           xmaxOLD, ymaxOLD, zmaxOLD);
02071
02072 /* The following is obsolete; we'll substitute analytical boundary
02073  * condition values when the new mesh falls outside the old */
02074 if ((xmaxNEW>xmaxOLD) || (ymaxNEW>ymaxOLD) || (zmaxNEW>zmaxOLD) ||
02075     (xminOLD>xminNEW) || (yminOLD>yminNEW) || (zminOLD>zminNEW)) {
02076
02077     Vnm_print(2, "Vpmg::focusFillBound -- new mesh not contained in old!\n");
02078     Vnm_print(2, "Vpmg::focusFillBound -- old mesh min = (%g, %g, %g)\n",
02079               xminOLD, yminOLD, zminOLD);
02080     Vnm_print(2, "Vpmg::focusFillBound -- old mesh max = (%g, %g, %g)\n",
02081               xmaxOLD, ymaxOLD, zmaxOLD);
02082     Vnm_print(2, "Vpmg::focusFillBound -- new mesh min = (%g, %g, %g)\n",
02083               xminNEW, yminNEW, zminNEW);
02084     Vnm_print(2, "Vpmg::focusFillBound -- new mesh max = (%g, %g, %g)\n",
02085               xmaxNEW, ymaxNEW, zmaxNEW);
02086     fflush(stderr);
02087     VASSERT(0);
02088 }
02089
02090 uvalMin = VPMGSMALL;
02091 uvalMax = -VPMGSMALL;
02092
02093 /* Fill the "i" boundaries (dirichlet) */
02094 for (k=0; k<nzNEW; k++) {
02095     for (j=0; j<nyNEW; j++) {
02096         /* Low X face */
02097         x = xminNEW;
02098         y = yminNEW + j*hyNEW;
02099         z = zminNEW + k*hzNEW;
02100         if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&
02101             (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
02102             ifloat = (x - xminOLD)/hxOLD;
02103             jfloat = (y - yminOLD)/hyOLD;
02104             kfloat = (z - zminOLD)/hzOLD;
02105             ihi = (int)ceil(ifloat);
02106             if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02107             ilo = (int)floor(ifloat);
02108             if (ilo < 0) ilo = 0;
02109             jhi = (int)ceil(jfloat);
02110             if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02111             jlo = (int)floor(jfloat);
02112             if (jlo < 0) jlo = 0;
02113             khi = (int)ceil(kfloat);
02114             if (khi > (nzOLD-1)) khi = nzOLD-1;
02115             klo = (int)floor(kfloat);
02116             if (klo < 0) klo = 0;
02117             dx = ifloat - (double)(ilo);
02118             dy = jfloat - (double)(jlo);
02119             dz = kfloat - (double)(klo);
02120             nx = nxOLD; ny = nyOLD; nz = nzOLD;
02121             uval = dx*dy*dz*(data[IJK(ihi, jhi, khi)])
02122                   + dx*(1.0-dy)*dz*(data[IJK(ihi, jlo, khi)])
02123                   + dx*dy*(1.0-dz)*(data[IJK(ihi, jhi, klo)])
02124                   + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi, jlo, klo)])
02125                   + (1.0-dx)*dy*dz*(data[IJK(ilo, jhi, khi)])
02126                   + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo, jlo, khi)])
02127                   + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo, jhi, klo)])
02128                   + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo, jlo, klo)]);
02129             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02130         } else {
02131             Vnm_print(2, "focusFillBound (%s, %d): Off old mesh at %g, %g \
02132               %g!\n", __FILE__, __LINE__, x, y, z);
02133             Vnm_print(2, "focusFillBound (%s, %d): old mesh lower corner at \
02134               %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02135             Vnm_print(2, "focusFillBound (%s, %d): old mesh upper corner at \
02136               %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02137             VASSERT(0);
02138         }
02139         nx = nxNEW; ny = nyNEW; nz = nzNEW;
02140         thee->gxcf[IJKx(j,k,0)] = uval;
02141         if (uval < uvalMin) uvalMin = uval;
02142         if (uval > uvalMax) uvalMax = uval;
02143
02144         /* High X face */
02145         x = xmaxNEW;
02146         if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&

```



```

02147         (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
02148         ifloat = (x - xminOLD)/hxOLD;
02149         jfloat = (y - yminOLD)/hyOLD;
02150         kfloat = (z - zminOLD)/hzOLD;
02151         ihi = (int)ceil(ifloat);
02152         if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02153         ilo = (int)floor(ifloat);
02154         if (ilo < 0) ilo = 0;
02155         jhi = (int)ceil(jfloat);
02156         if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02157         jlo = (int)floor(jfloat);
02158         if (jlo < 0) jlo = 0;
02159         khi = (int)ceil(kfloat);
02160         if (khi > (nzOLD-1)) khi = nzOLD-1;
02161         klo = (int)floor(kfloat);
02162         if (klo < 0) klo = 0;
02163         dx = ifloat - (double)(ilo);
02164         dy = jfloat - (double)(jlo);
02165         dz = kfloat - (double)(klo);
02166         nx = nxOLD; ny = nyOLD; nz = nzOLD;
02167         uval = dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02168         + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02169         + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02170         + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02171         + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02172         + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02173         + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02174         + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02175         nx = nxNEW; ny = nyNEW; nz = nzNEW;
02176     } else {
02177         Vnm_print(2, "focusFillBound (%s, %d): Off old mesh at %g, %g \
02178             %g!\n", __FILE__, __LINE__, x, y, z);
02179         Vnm_print(2, "focusFillBound (%s, %d): old mesh lower corner at \
02180             %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02181         Vnm_print(2, "focusFillBound (%s, %d): old mesh upper corner at \
02182             %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02183         VASSERT(0);
02184     }
02185     nx = nxNEW; ny = nyNEW; nz = nzNEW;
02186     thee->gxcf[IJKx(j,k,1)] = uval;
02187     if(uval < uvalMin) uvalMin = uval;
02188     if(uval > uvalMax) uvalMax = uval;
02189
02190     /* Zero Neumann conditions */
02191     nx = nxNEW; ny = nyNEW; nz = nzNEW;
02192     thee->gxcf[IJKx(j,k,2)] = 0.0;
02193     nx = nxNEW; ny = nyNEW; nz = nzNEW;
02194     thee->gxcf[IJKx(j,k,3)] = 0.0;
02195 }
02196 }
02197
02198 /* Fill the "j" boundaries (dirichlet) */
02199 for (k=0; k<nzNEW; k++) {
02200     for (i=0; i<nxNEW; i++) {
02201         /* Low Y face */
02202         x = xminNEW + i*hxNEW;
02203         y = yminNEW;
02204         z = zminNEW + k*hzNEW;
02205         if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&
02206             (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
02207             ifloat = (x - xminOLD)/hxOLD;
02208             jfloat = (y - yminOLD)/hyOLD;
02209             kfloat = (z - zminOLD)/hzOLD;
02210             ihi = (int)ceil(ifloat);
02211             if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02212             ilo = (int)floor(ifloat);
02213             if (ilo < 0) ilo = 0;
02214             jhi = (int)ceil(jfloat);
02215             if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02216             jlo = (int)floor(jfloat);
02217             if (jlo < 0) jlo = 0;
02218             khi = (int)ceil(kfloat);
02219             if (khi > (nzOLD-1)) khi = nzOLD-1;
02220             klo = (int)floor(kfloat);
02221             if (klo < 0) klo = 0;
02222             dx = ifloat - (double)(ilo);
02223             dy = jfloat - (double)(jlo);
02224             dz = kfloat - (double)(klo);
02225             nx = nxOLD; ny = nyOLD; nz = nzOLD;
02226             uval = dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02227             + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])

```

```

02228         + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02229         + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02230         + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02231         + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02232         + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02233         + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02234     nx = nxNEW; ny = nyNEW; nz = nzNEW;
02235 } else {
02236     Vnm_print(2, "focusFillBound (%s, %d): Off old mesh at %g, %g \
02237               %g!\n", __FILE__, __LINE__, x, y, z);
02238     Vnm_print(2, "focusFillBound (%s, %d): old mesh lower corner at \
02239               %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02240     Vnm_print(2, "focusFillBound (%s, %d): old mesh upper corner at \
02241               %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02242     VASSERT(0);
02243 }
02244 nx = nxNEW; ny = nyNEW; nz = nzNEW;
02245 thee->gycf[IJKy(i,k,0)] = uval;
02246 if(uval < uvalMin) uvalMin = uval;
02247 if(uval > uvalMax) uvalMax = uval;
02248
02249 /* High Y face */
02250 y = ymaxNEW;
02251 if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&
02252     (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
02253     ifloat = (x - xminOLD)/hxOLD;
02254     jfloat = (y - yminOLD)/hyOLD;
02255     kfloat = (z - zminOLD)/hzOLD;
02256     ihi = (int)ceil(ifloat);
02257     if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02258     ilo = (int)floor(ifloat);
02259     if (ilo < 0) ilo = 0;
02260     jhi = (int)ceil(jfloat);
02261     if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02262     jlo = (int)floor(jfloat);
02263     if (jlo < 0) jlo = 0;
02264     khi = (int)ceil(kfloat);
02265     if (khi > (nzOLD-1)) khi = nzOLD-1;
02266     klo = (int)floor(kfloat);
02267     if (klo < 0) klo = 0;
02268     dx = ifloat - (double)(ilo);
02269     dy = jfloat - (double)(jlo);
02270     dz = kfloat - (double)(klo);
02271     nx = nxOLD; ny = nyOLD; nz = nzOLD;
02272     uval = dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02273         + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02274         + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02275         + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02276         + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02277         + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02278         + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02279         + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02280     nx = nxNEW; ny = nyNEW; nz = nzNEW;
02281 } else {
02282     Vnm_print(2, "focusFillBound (%s, %d): Off old mesh at %g, %g \
02283               %g!\n", __FILE__, __LINE__, x, y, z);
02284     Vnm_print(2, "focusFillBound (%s, %d): old mesh lower corner at \
02285               %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02286     Vnm_print(2, "focusFillBound (%s, %d): old mesh upper corner at \
02287               %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02288     VASSERT(0);
02289 }
02290 nx = nxNEW; ny = nyNEW; nz = nzNEW;
02291 thee->gycf[IJKy(i,k,1)] = uval;
02292 if(uval < uvalMin) uvalMin = uval;
02293 if(uval > uvalMax) uvalMax = uval;
02294
02295 /* Zero Neumann conditions */
02296 nx = nxNEW; ny = nyNEW; nz = nzNEW;
02297 thee->gycf[IJKy(i,k,2)] = 0.0;
02298 nx = nxNEW; ny = nyNEW; nz = nzNEW;
02299 thee->gycf[IJKy(i,k,3)] = 0.0;
02300 }
02301 }
02302
02303 /* Fill the "k" boundaries (dirichlet) */
02304 for (j=0; j<nyNEW; j++) {
02305     for (i=0; i<nxNEW; i++) {
02306         /* Low Z face */
02307         x = xminNEW + i*hxNEW;
02308         y = yminNEW + j*hyNEW;

```

```

02309     z = zminNEW;
02310     if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&
02311         (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
02312         ifloat = (x - xminOLD)/hxOLD;
02313         jfloat = (y - yminOLD)/hyOLD;
02314         kfloat = (z - zminOLD)/hzOLD;
02315         ihi = (int)ceil(ifloat);
02316         if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02317         ilo = (int)floor(ifloat);
02318         if (ilo < 0) ilo = 0;
02319         jhi = (int)ceil(jfloat);
02320         if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02321         jlo = (int)floor(jfloat);
02322         if (jlo < 0) jlo = 0;
02323         khi = (int)ceil(kfloat);
02324         if (khi > (nzOLD-1)) khi = nzOLD-1;
02325         klo = (int)floor(kfloat);
02326         if (klo < 0) klo = 0;
02327         dx = ifloat - (double)(ilo);
02328         dy = jfloat - (double)(jlo);
02329         dz = kfloat - (double)(klo);
02330         nx = nxOLD; ny = nyOLD; nz = nzOLD;
02331         uval = dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02332             + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02333             + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02334             + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02335             + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02336             + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02337             + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02338             + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02339         nx = nxNEW; ny = nyNEW; nz = nzNEW;
02340     } else {
02341         Vnm_print(2, "focusFillBound (%s, %d): Off old mesh at %g, %g \
02342             %g!\n", __FILE__, __LINE__, x, y, z);
02343         Vnm_print(2, "focusFillBound (%s, %d): old mesh lower corner at \
02344             %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02345         Vnm_print(2, "focusFillBound (%s, %d): old mesh upper corner at \
02346             %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02347         VASSERT(0);
02348     }
02349     nx = nxNEW; ny = nyNEW; nz = nzNEW;
02350     thee->gzcf[IJKz(i,j,0)] = uval;
02351     if(uval < uvalMin) uvalMin = uval;
02352     if(uval > uvalMax) uvalMax = uval;
02353
02354     /* High Z face */
02355     z = zmaxNEW;
02356     if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&
02357         (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
02358         ifloat = (x - xminOLD)/hxOLD;
02359         jfloat = (y - yminOLD)/hyOLD;
02360         kfloat = (z - zminOLD)/hzOLD;
02361         ihi = (int)ceil(ifloat);
02362         if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02363         ilo = (int)floor(ifloat);
02364         if (ilo < 0) ilo = 0;
02365         jhi = (int)ceil(jfloat);
02366         if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02367         jlo = (int)floor(jfloat);
02368         if (jlo < 0) jlo = 0;
02369         khi = (int)ceil(kfloat);
02370         if (khi > (nzOLD-1)) khi = nzOLD-1;
02371         klo = (int)floor(kfloat);
02372         if (klo < 0) klo = 0;
02373         dx = ifloat - (double)(ilo);
02374         dy = jfloat - (double)(jlo);
02375         dz = kfloat - (double)(klo);
02376         nx = nxOLD; ny = nyOLD; nz = nzOLD;
02377         uval = dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02378             + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02379             + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02380             + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02381             + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02382             + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02383             + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02384             + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02385         nx = nxNEW; ny = nyNEW; nz = nzNEW;
02386     } else {
02387         Vnm_print(2, "focusFillBound (%s, %d): Off old mesh at %g, %g \
02388             %g!\n", __FILE__, __LINE__, x, y, z);
02389         Vnm_print(2, "focusFillBound (%s, %d): old mesh lower corner at \

```

```

02390         %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02391     Vnm_print(2, "focusFillBound (%s, %d): old mesh upper corner at \
02392         %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02393     VASSERT(0);
02394 }
02395 nx = nxNEW; ny = nyNEW; nz = nzNEW;
02396 thee->gzcf[IJKz(i,j,1)] = uval;
02397 if(uval < uvalMin) uvalMin = uval;
02398 if(uval > uvalMax) uvalMax = uval;
02399
02400 /* Zero Neumann conditions */
02401 nx = nxNEW; ny = nyNEW; nz = nzNEW;
02402 thee->gzcf[IJKz(i,j,2)] = 0.0;
02403 nx = nxNEW; ny = nyNEW; nz = nzNEW;
02404 thee->gzcf[IJKz(i,j,3)] = 0.0;
02405 }
02406 }
02407
02408 VWARN_MSG0(
02409     uvalMin >= SINH_MIN && uvalMax <= SINH_MAX,
02410     "Unusually large potential values\n"
02411     "    detected on the focusing boundary!\n"
02412     "    Convergence not guaranteed for NPBE/NRPBE calculations!"
02413 );
02414 }
02415
02416 VPRIVATE void extEnergy(Vpmg *thee, Vpmg *pmgOLD, PBEparm_calcEnergy extFlag,
02417     double partMin[3], double partMax[3], int bflags[6]) {
02418
02419     Vatom *atom;
02420     double hxNEW, hyNEW, hzNEW;
02421     double lowerCorner[3], upperCorner[3];
02422     int nxNEW, nyNEW, nzNEW;
02423     int nxOLD, nyOLD, nzOLD;
02424     int i,j,k;
02425     double xmin, xmax, ymin, ymax, zmin, zmax;
02426     double hxOLD, hyOLD, hzOLD;
02427     double xval, yval, zval;
02428     double x,y,z;
02429     int nx, ny, nz;
02430
02431     /* Set the new external energy contribution to zero. Any external
02432      * contributions from higher levels will be included in the appropriate
02433      * energy function call. */
02434     thee->extQmEnergy = 0;
02435     thee->extQfEnergy = 0;
02436     thee->extDiEnergy = 0;
02437
02438     /* New problem dimensions */
02439     hxNEW = thee->pmgp->hx;
02440     hyNEW = thee->pmgp->hy;
02441     hzNEW = thee->pmgp->hz;
02442     nxNEW = thee->pmgp->nx;
02443     nyNEW = thee->pmgp->ny;
02444     nzNEW = thee->pmgp->nz;
02445     lowerCorner[0] = thee->pmgp->xcent - ((double) (nxNEW-1)*hxNEW)/2.0;
02446     upperCorner[0] = thee->pmgp->xcent + ((double) (nxNEW-1)*hxNEW)/2.0;
02447     lowerCorner[1] = thee->pmgp->ycent - ((double) (nyNEW-1)*hyNEW)/2.0;
02448     upperCorner[1] = thee->pmgp->ycent + ((double) (nyNEW-1)*hyNEW)/2.0;
02449     lowerCorner[2] = thee->pmgp->zcent - ((double) (nzNEW-1)*hzNEW)/2.0;
02450     upperCorner[2] = thee->pmgp->zcent + ((double) (nzNEW-1)*hzNEW)/2.0;
02451
02452     Vnm_print(0, "VPMG::extEnergy: energy flag = %d\n", extFlag);
02453
02454     /* Old problem dimensions */
02455     nxOLD = pmgOLD->pmgp->nx;
02456     nyOLD = pmgOLD->pmgp->ny;
02457     nzOLD = pmgOLD->pmgp->nz;
02458
02459     /* Create a partition based on the new problem dimensions */
02460     /* Vnm_print(1, "DEBUG (%s, %d): extEnergy calling Vpmg_setPart for old PMG.\n",
02461        __FILE__, __LINE__); */
02462     Vpmg_setPart(pmgOLD, lowerCorner, upperCorner, bflags);
02463
02464
02465     Vnm_print(0, "VPMG::extEnergy: Finding extEnergy dimensions...\n");
02466     Vnm_print(0, "VPMG::extEnergy Disj part lower corner = (%g, %g, %g)\n",
02467         partMin[0], partMin[1], partMin[2]);
02468     Vnm_print(0, "VPMG::extEnergy Disj part upper corner = (%g, %g, %g)\n",
02469         partMax[0], partMax[1], partMax[2]);
02470

```

```

02471      /* Find the old dimensions */
02472
02473      hxOLD = pmgOLD->pmgp->hx;
02474      hyOLD = pmgOLD->pmgp->hy;
02475      hzOLD = pmgOLD->pmgp->hz;
02476      xmin = pmgOLD->pmgp->xcent - 0.5*hxOLD*(nxOLD-1);
02477      ymin = pmgOLD->pmgp->ycent - 0.5*hyOLD*(nyOLD-1);
02478      zmin = pmgOLD->pmgp->zcent - 0.5*hzOLD*(nzOLD-1);
02479      xmax = xmin+hxOLD*(nxOLD-1);
02480      ymax = ymin+hyOLD*(nyOLD-1);
02481      zmax = zmin+hzOLD*(nzOLD-1);
02482
02483      Vnm_print(0,"VPMG::extEnergy      Old lower corner = (%g, %g, %g)\n",
02484                xmin, ymin, zmin);
02485      Vnm_print(0,"VPMG::extEnergy      Old upper corner = (%g, %g, %g)\n",
02486                xmax, ymax, zmax);
02487
02488      /* Flip the partition, but do not include any points that will
02489       be included by another processor */
02490
02491      nx = nxOLD;
02492      ny = nyOLD;
02493      nz = nzOLD;
02494
02495      for(i=0; i<nx; i++) {
02496          xval = 1;
02497          x = i*hxOLD + xmin;
02498          if (x < partMin[0] && bflags[VAPBS_LEFT] == 1) xval = 0;
02499          else if (x > partMax[0] && bflags[VAPBS_RIGHT] == 1) xval = 0;
02500
02501          for(j=0; j<ny; j++) {
02502              yval = 1;
02503              y = j*hyOLD + ymin;
02504              if (y < partMin[1] && bflags[VAPBS_BACK] == 1) yval = 0;
02505              else if (y > partMax[1] && bflags[VAPBS_FRONT] == 1) yval = 0;
02506
02507              for(k=0; k<nz; k++) {
02508                  zval = 1;
02509                  z = k*hzOLD + zmin;
02510                  if (z < partMin[2] && bflags[VAPBS_DOWN] == 1) zval = 0;
02511                  else if (z > partMax[2] && bflags[VAPBS_UP] == 1) zval = 0;
02512
02513                  if (pmgOLD->pvec[IJK(i,j,k)] > VSMALL) pmgOLD->pvec[IJK(i,j,k)] = 1.0;
02514                  pmgOLD->pvec[IJK(i,j,k)] = (1 - (pmgOLD->pvec[IJK(i,j,k)])) * (xval*yval*zval);
02515              }
02516          }
02517      }
02518
02519      for (i=0; i<Valist_getNumberAtoms(thee->pbe->alist); i++) {
02520          xval=1;
02521          yval=1;
02522          zval=1;
02523          atom = Valist_getAtom(thee->pbe->alist, i);
02524          x = atom->position[0];
02525          y = atom->position[1];
02526          z = atom->position[2];
02527          if (x < partMin[0] && bflags[VAPBS_LEFT] == 1) xval = 0;
02528          else if (x > partMax[0] && bflags[VAPBS_RIGHT] == 1) xval = 0;
02529          if (y < partMin[1] && bflags[VAPBS_BACK] == 1) yval = 0;
02530          else if (y > partMax[1] && bflags[VAPBS_FRONT] == 1) yval = 0;
02531          if (z < partMin[2] && bflags[VAPBS_DOWN] == 1) zval = 0;
02532          else if (z > partMax[2] && bflags[VAPBS_UP] == 1) zval = 0;
02533          if (atom->partID > VSMALL) atom->partID = 1.0;
02534          atom->partID = (1 - atom->partID) * (xval*yval*zval);
02535      }
02536
02537      /* Now calculate the energy on inverted subset of the domain */
02538      thee->extQmEnergy = Vpmg_qmEnergy(pmgOLD, 1);
02539      Vnm_print(0, "VPMG::extEnergy: extQmEnergy = %g kT\n", thee->extQmEnergy);
02540      thee->extQfEnergy = Vpmg_qfEnergy(pmgOLD, 1);
02541      Vnm_print(0, "VPMG::extEnergy: extQfEnergy = %g kT\n", thee->extQfEnergy);
02542      thee->extDiEnergy = Vpmg_dielEnergy(pmgOLD, 1);
02543      Vnm_print(0, "VPMG::extEnergy: extDiEnergy = %g kT\n", thee->extDiEnergy);
02544      Vpmg_unsetPart(pmgOLD);
02545  }
02546
02547  VPRIVATE double bcflisp(double size, double *apos, double charge,
02548                          double xkappa, double prel, double *pos) {
02549
02550      double dist, val;
02551

```

```

02552     dist = VSQRT(VSQR(pos[0]-apos[0]) + VSQR(pos[1]-apos[1])
02553               + VSQR(pos[2]-apos[2]));
02554     if (xkappa > VSMALL) {
02555         val = prel*(charge/dist)*VEXP(-xkappa*(dist-size))
02556         / (1+xkappa*size);
02557     } else {
02558         val = prel*(charge/dist);
02559     }
02560
02561     return val;
02562 }
02563
02564 VPRIVATE void bcfll(double size, double *apos, double charge,
02565                   double xkappa, double prel, double *gxcf, double *gycf, double *gzcf,
02566                   double *xf, double *yf, double *zf, int nx, int ny, int nz) {
02567
02568     int i, j, k;
02569     double dist, val;
02570     double gpos[3];
02571
02572     /* the "i" boundaries (dirichlet) */
02573     for (k=0; k<nz; k++) {
02574         gpos[2] = zf[k];
02575         for (j=0; j<ny; j++) {
02576             gpos[1] = yf[j];
02577             gpos[0] = xf[0];
02578             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02579                     + VSQR(gpos[2]-apos[2]));
02580             if (xkappa > VSMALL) {
02581                 val = prel*(charge/dist)*VEXP(-xkappa*(dist-size))
02582                 / (1+xkappa*size);
02583             } else {
02584                 val = prel*(charge/dist);
02585             }
02586             gxcf[IJKx(j,k,0)] += val;
02587             gpos[0] = xf[nx-1];
02588             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02589                     + VSQR(gpos[2]-apos[2]));
02590             if (xkappa > VSMALL) {
02591                 val = prel*(charge/dist)*VEXP(-xkappa*(dist-size))
02592                 / (1+xkappa*size);
02593             } else {
02594                 val = prel*(charge/dist);
02595             }
02596             gxcf[IJKx(j,k,1)] += val;
02597         }
02598     }
02599
02600     /* the "j" boundaries (dirichlet) */
02601     for (k=0; k<nz; k++) {
02602         gpos[2] = zf[k];
02603         for (i=0; i<nx; i++) {
02604             gpos[0] = xf[i];
02605             gpos[1] = yf[0];
02606             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02607                     + VSQR(gpos[2]-apos[2]));
02608             if (xkappa > VSMALL) {
02609                 val = prel*(charge/dist)*VEXP(-xkappa*(dist-size))
02610                 / (1+xkappa*size);
02611             } else {
02612                 val = prel*(charge/dist);
02613             }
02614             gycf[IJKy(i,k,0)] += val;
02615             gpos[1] = yf[ny-1];
02616             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02617                     + VSQR(gpos[2]-apos[2]));
02618             if (xkappa > VSMALL) {
02619                 val = prel*(charge/dist)*VEXP(-xkappa*(dist-size))
02620                 / (1+xkappa*size);
02621             } else {
02622                 val = prel*(charge/dist);
02623             }
02624             gycf[IJKy(i,k,1)] += val;
02625         }
02626     }
02627
02628     /* the "k" boundaries (dirichlet) */
02629     for (j=0; j<ny; j++) {
02630         gpos[1] = yf[j];
02631         for (i=0; i<nx; i++) {
02632             gpos[0] = xf[i];

```

```

02633         gpos[2] = zf[0];
02634         dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02635                     + VSQR(gpos[2]-apos[2]));
02636         if (xkappa > VSMALL) {
02637             val = prel*(charge/dist)*VEXP(-xkappa*(dist-size))
02638                 / (1+xkappa*size);
02639         } else {
02640             val = prel*(charge/dist);
02641         }
02642         gzcfc[IJKz(i,j,0)] += val;
02643         gpos[2] = zf[nz-1];
02644         dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02645                     + VSQR(gpos[2]-apos[2]));
02646         if (xkappa > VSMALL) {
02647             val = prel*(charge/dist)*VEXP(-xkappa*(dist-size))
02648                 / (1+xkappa*size);
02649         } else {
02650             val = prel*(charge/dist);
02651         }
02652         gzcfc[IJKz(i,j,1)] += val;
02653     }
02654 }
02655 }
02656
02657 VPRIVATE void bcf12(double size, double *apos,
02658                   double charge, double *dipole, double *quad,
02659                   double xkappa, double eps_p, double eps_w, double T,
02660                   double *gxycf, double *gyycf, double *gzycf,
02661                   double *xf, double *yf, double *zf,
02662                   int nx, int ny, int nz) {
02663
02664     int i, j, k;
02665     double val;
02666     double gpos[3], tensor[3];
02667     double ux, uy, uz, xr, yr, zr;
02668     double qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
02669     double dist, pre;
02670
02671     VASSERT(dipole != VNULL);
02672     ux = dipole[0];
02673     uy = dipole[1];
02674     uz = dipole[2];
02675     if (quad != VNULL) {
02676         /* The factor of 1/3 results from using a
02677          * traceless quadrupole definition. See, for example,
02678          * "The Theory of Intermolecular Forces" by A.J. Stone,
02679          * Chapter 3. */
02680         qxx = quad[0] / 3.0;
02681         qxy = quad[1] / 3.0;
02682         qxz = quad[2] / 3.0;
02683         qyx = quad[3] / 3.0;
02684         qyy = quad[4] / 3.0;
02685         qyz = quad[5] / 3.0;
02686         qzx = quad[6] / 3.0;
02687         qzy = quad[7] / 3.0;
02688         qzz = quad[8] / 3.0;
02689     } else {
02690         qxx = 0.0;
02691         qxy = 0.0;
02692         qxz = 0.0;
02693         qyx = 0.0;
02694         qyy = 0.0;
02695         qyz = 0.0;
02696         qzx = 0.0;
02697         qzy = 0.0;
02698         qzz = 0.0;
02699     }
02700
02701     pre = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*
02702          Vunit_kb*T);
02703     pre = pre*(1.0e10);
02704
02705     /* the "i" boundaries (dirichlet) */
02706     for (k=0; k<nz; k++) {
02707         gpos[2] = zf[k];
02708         for (j=0; j<ny; j++) {
02709             gpos[1] = yf[j];
02710             gpos[0] = xf[0];
02711             xr = gpos[0] - apos[0];
02712             yr = gpos[1] - apos[1];
02713             zr = gpos[2] - apos[2];

```

```

02713         dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02714         multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02715         val = pre*charge*tensor[0];
02716         val -= pre*ux*xr*tensor[1];
02717         val -= pre*uy*yr*tensor[1];
02718         val -= pre*uz*zr*tensor[1];
02719         val += pre*qxx*xr*xr*tensor[2];
02720         val += pre*qyy*yr*yr*tensor[2];
02721         val += pre*qzz*zr*zr*tensor[2];
02722         val += pre*2.0*qxy*xr*yr*tensor[2];
02723         val += pre*2.0*qxz*xr*zr*tensor[2];
02724         val += pre*2.0*qyz*yr*zr*tensor[2];
02725         gxcf[IJKx(j,k,0)] += val;
02726
02727         gpos[0] = xf[nx-1];
02728         xr = gpos[0] - apos[0];
02729         dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02730         multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02731         val = pre*charge*tensor[0];
02732         val -= pre*ux*xr*tensor[1];
02733         val -= pre*uy*yr*tensor[1];
02734         val -= pre*uz*zr*tensor[1];
02735         val += pre*qxx*xr*xr*tensor[2];
02736         val += pre*qyy*yr*yr*tensor[2];
02737         val += pre*qzz*zr*zr*tensor[2];
02738         val += pre*2.0*qxy*xr*yr*tensor[2];
02739         val += pre*2.0*qxz*xr*zr*tensor[2];
02740         val += pre*2.0*qyz*yr*zr*tensor[2];
02741         gxcf[IJKx(j,k,1)] += val;
02742     }
02743 }
02744
02745 /* the "j" boundaries (dirichlet) */
02746 for (k=0; k<nz; k++) {
02747     gpos[2] = zf[k];
02748     for (i=0; i<nx; i++) {
02749         gpos[0] = xf[i];
02750         gpos[1] = yf[0];
02751         xr = gpos[0] - apos[0];
02752         yr = gpos[1] - apos[1];
02753         zr = gpos[2] - apos[2];
02754         dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02755         multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02756         val = pre*charge*tensor[0];
02757         val -= pre*ux*xr*tensor[1];
02758         val -= pre*uy*yr*tensor[1];
02759         val -= pre*uz*zr*tensor[1];
02760         val += pre*qxx*xr*xr*tensor[2];
02761         val += pre*qyy*yr*yr*tensor[2];
02762         val += pre*qzz*zr*zr*tensor[2];
02763         val += pre*2.0*qxy*xr*yr*tensor[2];
02764         val += pre*2.0*qxz*xr*zr*tensor[2];
02765         val += pre*2.0*qyz*yr*zr*tensor[2];
02766         gycf[IJKy(i,k,0)] += val;
02767
02768         gpos[1] = yf[ny-1];
02769         yr = gpos[1] - apos[1];
02770         dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02771         multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02772         val = pre*charge*tensor[0];
02773         val -= pre*ux*xr*tensor[1];
02774         val -= pre*uy*yr*tensor[1];
02775         val -= pre*uz*zr*tensor[1];
02776         val += pre*qxx*xr*xr*tensor[2];
02777         val += pre*qyy*yr*yr*tensor[2];
02778         val += pre*qzz*zr*zr*tensor[2];
02779         val += pre*2.0*qxy*xr*yr*tensor[2];
02780         val += pre*2.0*qxz*xr*zr*tensor[2];
02781         val += pre*2.0*qyz*yr*zr*tensor[2];
02782         gycf[IJKy(i,k,1)] += val;
02783     }
02784 }
02785
02786 /* the "k" boundaries (dirichlet) */
02787 for (j=0; j<ny; j++) {
02788     gpos[1] = yf[j];
02789     for (i=0; i<nx; i++) {
02790         gpos[0] = xf[i];
02791         gpos[2] = zf[0];
02792         xr = gpos[0] - apos[0];
02793         yr = gpos[1] - apos[1];

```



```

02794         zr = gpos[2] - apos[2];
02795         dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02796         multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02797         val = pre*charge*tensor[0];
02798         val -= pre*ux*xr*tensor[1];
02799         val -= pre*uy*yr*tensor[1];
02800         val -= pre*uz*zr*tensor[1];
02801         val += pre*qxx*xr*xr*tensor[2];
02802         val += pre*qyy*yr*yr*tensor[2];
02803         val += pre*qzz*zr*zr*tensor[2];
02804         val += pre*2.0*qxy*xr*yr*tensor[2];
02805         val += pre*2.0*qxz*xr*zr*tensor[2];
02806         val += pre*2.0*qyz*yr*zr*tensor[2];
02807         gzcfc[IJKz(i,j,0)] += val;
02808
02809         gpos[2] = zf[nz-1];
02810         zr = gpos[2] - apos[2];
02811         dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02812         multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02813         val = pre*charge*tensor[0];
02814         val -= pre*ux*xr*tensor[1];
02815         val -= pre*uy*yr*tensor[1];
02816         val -= pre*uz*zr*tensor[1];
02817         val += pre*qxx*xr*xr*tensor[2];
02818         val += pre*qyy*yr*yr*tensor[2];
02819         val += pre*qzz*zr*zr*tensor[2];
02820         val += pre*2.0*qxy*xr*yr*tensor[2];
02821         val += pre*2.0*qxz*xr*zr*tensor[2];
02822         val += pre*2.0*qyz*yr*zr*tensor[2];
02823         gzcfc[IJKz(i,j,1)] += val;
02824     }
02825 }
02826 }
02827
02828 VPRIVATE void bcCalcOrig(Vpmg *thee) {
02829
02830     int nx, ny, nz;
02831     double size, *position, charge, xkappa, eps_w, T, prel;
02832     double *dipole, *quadrupole, debye, eps_p;
02833     double xr, yr, zr, qave, *apos;
02834     double sdhcharge, sdhdipole[3], traced[9], sdhquadrupole[9];
02835     int i, j, k, iatom;
02836     Vpbe *pbe;
02837     Vatom *atom;
02838     Valist *alist;
02839
02840     pbe = thee->pbe;
02841     alist = thee->pbe->alist;
02842     nx = thee->pmgp->nx;
02843     ny = thee->pmgp->ny;
02844     nz = thee->pmgp->nz;
02845
02846     /* Zero out the boundaries */
02847     /* the "i" boundaries (dirichlet) */
02848     for (k=0; k<nz; k++) {
02849         for (j=0; j<ny; j++) {
02850             thee->gxcfc[IJKx(j,k,0)] = 0.0;
02851             thee->gxcfc[IJKx(j,k,1)] = 0.0;
02852             thee->gxcfc[IJKx(j,k,2)] = 0.0;
02853             thee->gxcfc[IJKx(j,k,3)] = 0.0;
02854         }
02855     }
02856
02857     /* the "j" boundaries (dirichlet) */
02858     for (k=0; k<nz; k++) {
02859         for (i=0; i<nx; i++) {
02860             thee->gycfc[IJKy(i,k,0)] = 0.0;
02861             thee->gycfc[IJKy(i,k,1)] = 0.0;
02862             thee->gycfc[IJKy(i,k,2)] = 0.0;
02863             thee->gycfc[IJKy(i,k,3)] = 0.0;
02864         }
02865     }
02866
02867     /* the "k" boundaries (dirichlet) */
02868     for (j=0; j<ny; j++) {
02869         for (i=0; i<nx; i++) {
02870             thee->gzcfc[IJKz(i,j,0)] = 0.0;
02871             thee->gzcfc[IJKz(i,j,1)] = 0.0;
02872             thee->gzcfc[IJKz(i,j,2)] = 0.0;
02873             thee->gzcfc[IJKz(i,j,3)] = 0.0;
02874         }
02875     }

```

```

02875     }
02876
02877     /* For each "atom" (only one for bcfl=1), we use the following formula to
02878     * calculate the boundary conditions:
02879     *  $g(x) = \frac{q}{4\pi\epsilon_0\epsilon_w k_b T} \frac{\exp(-\kappa(d-a))}{1+\kappa a}$ 
02880     *  $\frac{1}{d}$ 
02881     * where  $d = ||x - x_0||$  (in m) and  $a$  is the size of the atom (in m).
02882     * We only need to evaluate some of these prefactors once:
02883     *  $prel = \frac{q}{4\pi\epsilon_0\epsilon_w k_b T}$ 
02884     * which gives the potential as
02885     *  $g(x) = prel * q/d * \frac{\exp(-\kappa(d-a))}{1+\kappa a}$ 
02886     */
02887     eps_w = Vpbe_getSolventDiel(pbe); /* Dimensionless */
02888     eps_p = Vpbe_getSoluteDiel(pbe); /* Dimensionless */
02889     T = Vpbe_getTemperature(pbe); /* K */
02890     prel = (Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T);
02891
02892     /* Finally, if we convert keep  $\kappa$  in  $\text{\AA}^{-1}$  and scale  $prel$  by
02893     * m/ $\text{\AA}$ , then we will only need to deal with distances and sizes in
02894     * Angstroms rather than meters. */
02895     kappa = Vpbe_getXkappa(pbe); /*  $\text{\AA}^{-1}$  */
02896     prel = prel*(1.0e10);
02897
02898     switch (thee->pmgp->bcfl) {
02899     /* If we have zero boundary conditions, we're done */
02900     case BCFL_ZERO:
02901         return;
02902
02903     /* For single DH sphere BC's, we only have one "atom" to deal with;
02904     * get its information and */
02905     case BCFL_SDH:
02906         size = Vpbe_getSoluteRadius(pbe);
02907         position = Vpbe_getSoluteCenter(pbe);
02908
02909         /*
02910         For AMOEBA SDH boundary conditions, we need to find the
02911         total monopole, dipole and traceless quadrupole moments
02912         of either the permanent multipoles, induced dipoles or
02913         non-local induced dipoles.
02914         */
02915
02916         sdhcharge = 0.0;
02917         for (i=0; i<3; i++) sdhdipole[i] = 0.0;
02918         for (i=0; i<9; i++) sdhquadrupole[i] = 0.0;
02919
02920         for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
02921             atom = Valist_getAtom(alist, iatom);
02922             apos = Vatom_getPosition(atom);
02923             xr = apos[0] - position[0];
02924             yr = apos[1] - position[1];
02925             zr = apos[2] - position[2];
02926             switch (thee->chargeSrc) {
02927             case VCM_CHARGE:
02928                 charge = Vatom_getCharge(atom);
02929                 sdhcharge += charge;
02930                 sdhdipole[0] += xr * charge;
02931                 sdhdipole[1] += yr * charge;
02932                 sdhdipole[2] += zr * charge;
02933                 traced[0] = xr*xr*charge;
02934                 traced[1] = xr*yr*charge;
02935                 traced[2] = xr*zr*charge;
02936                 traced[3] = yr*xr*charge;
02937                 traced[4] = yr*yr*charge;
02938                 traced[5] = yr*zr*charge;
02939                 traced[6] = zr*xr*charge;
02940                 traced[7] = zr*yr*charge;
02941                 traced[8] = zr*zr*charge;
02942                 gave = (traced[0] + traced[4] + traced[8]) / 3.0;
02943                 sdhquadrupole[0] += 1.5*(traced[0] - gave);
02944                 sdhquadrupole[1] += 1.5*(traced[1]);
02945                 sdhquadrupole[2] += 1.5*(traced[2]);
02946                 sdhquadrupole[3] += 1.5*(traced[3]);
02947                 sdhquadrupole[4] += 1.5*(traced[4] - gave);
02948                 sdhquadrupole[5] += 1.5*(traced[5]);
02949                 sdhquadrupole[6] += 1.5*(traced[6]);
02950                 sdhquadrupole[7] += 1.5*(traced[7]);
02951                 sdhquadrupole[8] += 1.5*(traced[8] - gave);
02952             #if defined(WITH_TINKER)
02953             case VCM_PERMANENT:
02954                 charge = Vatom_getCharge(atom);
02955             
```

```

02956         dipole = Vatom_getDipole(atom);
02957         quadrupole = Vatom_getQuadrupole(atom);
02958         sdhcharge += charge;
02959         sdhdipole[0] += xr * charge;
02960         sdhdipole[1] += yr * charge;
02961         sdhdipole[2] += zr * charge;
02962         traced[0] = xr*xr*charge;
02963         traced[1] = xr*yr*charge;
02964         traced[2] = xr*zr*charge;
02965         traced[3] = yr*xr*charge;
02966         traced[4] = yr*yr*charge;
02967         traced[5] = yr*zr*charge;
02968         traced[6] = zr*xr*charge;
02969         traced[7] = zr*yr*charge;
02970         traced[8] = zr*zr*charge;
02971         sdhdipole[0] += dipole[0];
02972         sdhdipole[1] += dipole[1];
02973         sdhdipole[2] += dipole[2];
02974         traced[0] += 2.0*xr*dipole[0];
02975         traced[1] += xr*dipole[1] + yr*dipole[0];
02976         traced[2] += xr*dipole[2] + zr*dipole[0];
02977         traced[3] += yr*dipole[0] + xr*dipole[1];
02978         traced[4] += 2.0*yr*dipole[1];
02979         traced[5] += yr*dipole[2] + zr*dipole[1];
02980         traced[6] += zr*dipole[0] + xr*dipole[2];
02981         traced[7] += zr*dipole[1] + yr*dipole[2];
02982         traced[8] += 2.0*zr*dipole[2];
02983         qave = (traced[0] + traced[4] + traced[8]) / 3.0;
02984         sdhquadrupole[0] += 1.5*(traced[0] - qave);
02985         sdhquadrupole[1] += 1.5*(traced[1]);
02986         sdhquadrupole[2] += 1.5*(traced[2]);
02987         sdhquadrupole[3] += 1.5*(traced[3]);
02988         sdhquadrupole[4] += 1.5*(traced[4] - qave);
02989         sdhquadrupole[5] += 1.5*(traced[5]);
02990         sdhquadrupole[6] += 1.5*(traced[6]);
02991         sdhquadrupole[7] += 1.5*(traced[7]);
02992         sdhquadrupole[8] += 1.5*(traced[8] - qave);
02993         sdhquadrupole[0] += quadrupole[0];
02994         sdhquadrupole[1] += quadrupole[1];
02995         sdhquadrupole[2] += quadrupole[2];
02996         sdhquadrupole[3] += quadrupole[3];
02997         sdhquadrupole[4] += quadrupole[4];
02998         sdhquadrupole[5] += quadrupole[5];
02999         sdhquadrupole[6] += quadrupole[6];
03000         sdhquadrupole[7] += quadrupole[7];
03001         sdhquadrupole[8] += quadrupole[8];
03002     case VCM_INDUCED:
03003         dipole = Vatom_getInducedDipole(atom);
03004         sdhdipole[0] += dipole[0];
03005         sdhdipole[1] += dipole[1];
03006         sdhdipole[2] += dipole[2];
03007         traced[0] = 2.0*xr*dipole[0];
03008         traced[1] = xr*dipole[1] + yr*dipole[0];
03009         traced[2] = xr*dipole[2] + zr*dipole[0];
03010         traced[3] = yr*dipole[0] + xr*dipole[1];
03011         traced[4] = 2.0*yr*dipole[1];
03012         traced[5] = yr*dipole[2] + zr*dipole[1];
03013         traced[6] = zr*dipole[0] + xr*dipole[2];
03014         traced[7] = zr*dipole[1] + yr*dipole[2];
03015         traced[8] = 2.0*zr*dipole[2];
03016         qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03017         sdhquadrupole[0] += 1.5*(traced[0] - qave);
03018         sdhquadrupole[1] += 1.5*(traced[1]);
03019         sdhquadrupole[2] += 1.5*(traced[2]);
03020         sdhquadrupole[3] += 1.5*(traced[3]);
03021         sdhquadrupole[4] += 1.5*(traced[4] - qave);
03022         sdhquadrupole[5] += 1.5*(traced[5]);
03023         sdhquadrupole[6] += 1.5*(traced[6]);
03024         sdhquadrupole[7] += 1.5*(traced[7]);
03025         sdhquadrupole[8] += 1.5*(traced[8] - qave);
03026     case VCM_NLINDUCED:
03027         dipole = Vatom_getNLInducedDipole(atom);
03028         sdhdipole[0] += dipole[0];
03029         sdhdipole[1] += dipole[1];
03030         sdhdipole[2] += dipole[2];
03031         traced[0] = 2.0*xr*dipole[0];
03032         traced[1] = xr*dipole[1] + yr*dipole[0];
03033         traced[2] = xr*dipole[2] + zr*dipole[0];
03034         traced[3] = yr*dipole[0] + xr*dipole[1];
03035         traced[4] = 2.0*yr*dipole[1];
03036         traced[5] = yr*dipole[2] + zr*dipole[1];

```

```

03037         traced[6] = zr*dipole[0] + xr*dipole[2];
03038         traced[7] = zr*dipole[1] + yr*dipole[2];
03039         traced[8] = 2.0*zr*dipole[2];
03040         qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03041         sdhquadrupole[0] += 1.5*(traced[0] - qave);
03042         sdhquadrupole[1] += 1.5*(traced[1]);
03043         sdhquadrupole[2] += 1.5*(traced[2]);
03044         sdhquadrupole[3] += 1.5*(traced[3]);
03045         sdhquadrupole[4] += 1.5*(traced[4] - qave);
03046         sdhquadrupole[5] += 1.5*(traced[5]);
03047         sdhquadrupole[6] += 1.5*(traced[6]);
03048         sdhquadrupole[7] += 1.5*(traced[7]);
03049         sdhquadrupole[8] += 1.5*(traced[8] - qave);
03050 #endif /* if defined(WITH_TINKER) */
03051     }
03052 }
03053 /* SDH dipole and traceless quadrupole values
03054 were checked against similar routines in TINKER
03055 for large proteins.
03056
03057 debye=4.8033324;
03058 printf("%6.3f, %6.3f, %6.3f\n", sdhdipole[0]*debye,
03059 sdhdipole[1]*debye, sdhdipole[2]*debye);
03060 printf("%6.3f\n", sdhquadrupole[0]*debye);
03061 printf("%6.3f %6.3f\n", sdhquadrupole[3]*debye,
03062 sdhquadrupole[4]*debye);
03063 printf("%6.3f %6.3f %6.3f\n", sdhquadrupole[6]*debye,
03064 sdhquadrupole[7]*debye, sdhquadrupole[8]*debye);
03065 */
03066
03067 bcf12(size, position, sdhcharge, sdhdipole, sdhquadrupole,
03068 xkappa, eps_p, eps_w, T, thee->gxcf, thee->gycf,
03069 thee->gzcf, thee->xf, thee->yf, thee->zf, nx, ny, nz);
03070 break;
03071
03072 case BCFL_MDH:
03073     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
03074         atom = Valist_getAtom(alist, iatom);
03075         position = Vatom_getPosition(atom);
03076         charge = Vunit_ec*Vatom_getCharge(atom);
03077         dipole = VNULL;
03078         quadrupole = VNULL;
03079         size = Vatom_getRadius(atom);
03080         switch (thee->chargeSrc)
03081         {
03082             case VCM_CHARGE:
03083                 ;
03084 #if defined(WITH_TINKER)
03085                 case VCM_PERMANENT:
03086                     dipole = Vatom_getDipole(atom);
03087                     quadrupole = Vatom_getQuadrupole(atom);
03088
03089                 case VCM_INDUCED:
03090                     dipole = Vatom_getInducedDipole(atom);
03091
03092                 case VCM_NLINDUCED:
03093                     dipole = Vatom_getNLInducedDipole(atom);
03094 #endif
03095             }
03096         bcf11(size, position, charge, xkappa, prel,
03097             thee->gxcf, thee->gycf, thee->gzcf,
03098             thee->xf, thee->yf, thee->zf, nx, ny, nz);
03099     }
03100     break;
03101
03102 case BCFL_UNUSED:
03103     Vnm_print(2, "bcCalc: Invalid bcf1 (%d)!\n", thee->pmgp->bcf1);
03104     VASSERT(0);
03105
03106 case BCFL_FOCUS:
03107     Vnm_print(2, "VPMG::bcCalc -- not appropriate for focusing!\n");
03108     VASSERT(0);
03109
03110 default:
03111     Vnm_print(2, "VPMG::bcCalc -- invalid boundary condition \
03112 flag (%d)!\n", thee->pmgp->bcf1);
03113     VASSERT(0);
03114 }
03115 }
03116
03117 /*

```

```

03118 Used by bcflnew
03119 */
03120 VPRIVATE int gridPointIsValid(int i, int j, int k, int nx, int ny, int nz){
03121
03122     int isValid = 0;
03123
03124     if((k==0) || (k==nz-1)){
03125         isValid = 1;
03126     }else if((j==0) || (j==ny-1)){
03127         isValid = 1;
03128     }else if((i==0) || (i==nx-1)){
03129         isValid = 1;
03130     }
03131
03132     return isValid;
03133 }
03134
03135 /*
03136 Used by bcflnew
03137 */
03138 #ifdef DEBUG_MAC_OSX_OCL
03139 #include "mach_chud.h"
03140 VPRIVATE void packAtomsOpenCL(float *ax, float *ay, float *az,
03141                               float *charge, float *size, Vpmg *thee){
03142
03143     int i;
03144     int natoms;
03145
03146     Vatom *atom = VNULL;
03147     Valist *alist = VNULL;
03148
03149     alist = thee->pbe->alist;
03150     natoms = Valist_getNumberAtoms(alist);
03151
03152     for(i=0;i<natoms;i++){
03153         atom = &(alist->atoms[i]);
03154         charge[i] = Vunit_ec*atom->charge;
03155         ax[i] = atom->position[0];
03156         ay[i] = atom->position[1];
03157         az[i] = atom->position[2];
03158         size[i] = atom->radius;
03159     }
03160 }
03161
03162 /*
03163 Used by bcflnew
03164 */
03165 VPRIVATE void packUnpackOpenCL(int nx, int ny, int nz, int ngrid,
03166                                float *gx, float *gy, float *gz, float *value,
03167                                Vpmg *thee, int pack){
03168
03169     int i,j,k,igrid;
03170     int x0,x1,y0,y1,z0,z1;
03171
03172     float gpos[3];
03173     double *xf, *yf, *zf;
03174     double *gxcf, *gycf, *gzcf;
03175
03176     xf = thee->xf;
03177     yf = thee->yf;
03178     zf = thee->zf;
03179
03180     gxcf = thee->gxcf;
03181     gycf = thee->gycf;
03182     gzcf = thee->gzcf;
03183
03184     igrid = 0;
03185     for(k=0;k<nz;k++){
03186         gpos[2] = zf[k];
03187         for(j=0;j<ny;j++){
03188             gpos[1] = yf[j];
03189             for(i=0;i<nx;i++){
03190                 gpos[0] = xf[i];
03191                 if(gridPointIsValid(i, j, k, nx, ny, nz)){
03192                     if(pack != 0){
03193                         gx[igrid] = gpos[0];
03194                         gy[igrid] = gpos[1];
03195                         gz[igrid] = gpos[2];
03196
03197                         value[igrid] = 0.0;
03198                     }else{

```

```

03199         x0 = IJKx(j,k,0);
03200         x1 = IJKx(j,k,1);
03201         y0 = IJKy(i,k,0);
03202         y1 = IJKy(i,k,1);
03203         z0 = IJKz(i,j,0);
03204         z1 = IJKz(i,j,1);
03205
03206         if(i==0){
03207             gxcf[x0] += value[igrid];
03208         }
03209         if(i==nx-1){
03210             gxcf[x1] += value[igrid];
03211         }
03212         if(j==0){
03213             gycf[y0] += value[igrid];
03214         }
03215         if(j==ny-1){
03216             gycf[y1] += value[igrid];
03217         }
03218         if(k==0){
03219             gzcf[z0] += value[igrid];
03220         }
03221         if(k==nz-1){
03222             gzcf[z1] += value[igrid];
03223         }
03224     }
03225
03226     igrid++;
03227 } //end is valid point
03228 } //end i
03229 } //end j
03230 } //end k
03231
03232 }
03233
03234 /*
03235 bcflnew is an optimized replacement for bcfl1. bcfl1 is still used when TINKER
03236 support is compiled in.
03237 bcflnew uses: packUnpack, packAtoms, gridPointIsValid
03238 */
03239 VPRIVATE void bcflnewOpenCL(Vpmg *thee){
03240
03241     int i,j,k, iatom, igrid;
03242     int x0, x1, y0, y1, z0, z1;
03243
03244     int nx, ny, nz;
03245     int natoms, ngrid, ngadj;
03246
03247     float dist, pre1, eps_w, eps_p, T, xkappa;
03248
03249     float *ax, *ay, *az;
03250     float *charge, *size, *val;
03251
03252     float *gx, *gy, *gz;
03253
03254     Vpbe *pbe = thee->pbe;
03255
03256     nx = thee->pmgp->nx;
03257     ny = thee->pmgp->ny;
03258     nz = thee->pmgp->nz;
03259
03260     eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
03261     eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
03262     T = Vpbe_getTemperature(pbe);              /* K */
03263     pre1 = ((Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T))*(1.0e10);
03264     xkappa = Vpbe_getXkappa(pbe);
03265
03266     natoms = Valist_getNumberAtoms(thee->pbe->alist);
03267     ngrid = 2*((nx*ny) + (ny*nz) + (nx*nz));
03268     ngadj = ngrid + (512 - (ngrid & 511));
03269
03270     ax = (float*)malloc(natoms * sizeof(float));
03271     ay = (float*)malloc(natoms * sizeof(float));
03272     az = (float*)malloc(natoms * sizeof(float));
03273
03274     charge = (float*)malloc(natoms * sizeof(float));
03275     size = (float*)malloc(natoms * sizeof(float));
03276
03277     gx = (float*)malloc(ngrid * sizeof(float));
03278     gy = (float*)malloc(ngrid * sizeof(float));
03279     gz = (float*)malloc(ngrid * sizeof(float));

```

```

03280
03281     val = (float*)malloc(ngrid * sizeof(float));
03282
03283     packAtomsOpenCL(ax, ay, az, charge, size, thee);
03284     packUnpackOpenCL(nx, ny, nz, ngrid, gx, gy, gz, val, thee, 1);
03285
03286     runMDHCL(ngrid, natoms, ngadj, ax, ay, az, gx, gy, gz, charge, size, xkappa, prel, val);
03287
03288     packUnpackOpenCL(nx, ny, nz, ngrid, gx, gy, gz, val, thee, 0);
03289
03290     free(ax);
03291     free(ay);
03292     free(az);
03293     free(charge);
03294     free(size);
03295
03296     free(gx);
03297     free(gy);
03298     free(gz);
03299     free(val);
03300 }
03301 #endif
03302
03303 VPRIVATE void packAtoms(double *ax, double *ay, double *az,
03304                        double *charge, double *size, Vpmg *thee){
03305
03306     int i;
03307     int natoms;
03308
03309     Vatom *atom = VNULL;
03310     Valist *alist = VNULL;
03311
03312     alist = thee->pbe->alist;
03313     natoms = Valist_getNumberAtoms(alist);
03314
03315     for(i=0; i<natoms; i++){
03316         atom = &(alist->atoms[i]);
03317         charge[i] = Vunit_ec*atom->charge;
03318         ax[i] = atom->position[0];
03319         ay[i] = atom->position[1];
03320         az[i] = atom->position[2];
03321         size[i] = atom->radius;
03322     }
03323 }
03324
03325 /*
03326  Used by bcflnew
03327 */
03328 VPRIVATE void packUnpack(int nx, int ny, int nz, int ngrid,
03329                          double *gx, double *gy, double *gz, double *value,
03330                          Vpmg *thee, int pack){
03331
03332     int i, j, k, igrd;
03333     int x0, x1, y0, y1, z0, z1;
03334
03335     double gpos[3];
03336     double *xf, *yf, *zf;
03337     double *gxcf, *gycf, *gzcf;
03338
03339     xf = thee->xf;
03340     yf = thee->yf;
03341     zf = thee->zf;
03342
03343     gxcf = thee->gxcf;
03344     gycf = thee->gycf;
03345     gzcf = thee->gzcf;
03346
03347     igrd = 0;
03348     for(k=0; k<nz; k++){
03349         gpos[2] = zf[k];
03350         for(j=0; j<ny; j++){
03351             gpos[1] = yf[j];
03352             for(i=0; i<nx; i++){
03353                 gpos[0] = xf[i];
03354                 if(gridPointIsValid(i, j, k, nx, ny, nz)){
03355                     if(pack != 0){
03356                         gx[igrd] = gpos[0];
03357                         gy[igrd] = gpos[1];
03358                         gz[igrd] = gpos[2];
03359
03360                         value[igrd] = 0.0;

```

```

03361         }else{
03362             x0 = IJKx(j,k,0);
03363             x1 = IJKx(j,k,1);
03364             y0 = IJKy(i,k,0);
03365             y1 = IJKy(i,k,1);
03366             z0 = IJKz(i,j,0);
03367             z1 = IJKz(i,j,1);
03368
03369             if(i==0){
03370                 gxcf[x0] += value[igrid];
03371             }
03372             if(i==nx-1){
03373                 gxcf[x1] += value[igrid];
03374             }
03375             if(j==0){
03376                 gycf[y0] += value[igrid];
03377             }
03378             if(j==ny-1){
03379                 gycf[y1] += value[igrid];
03380             }
03381             if(k==0){
03382                 gzcf[z0] += value[igrid];
03383             }
03384             if(k==nz-1){
03385                 gzcf[z1] += value[igrid];
03386             }
03387         }
03388
03389         igrid++;
03390     } //end is valid point
03391 } //end i
03392 } //end j
03393 } //end k
03394 }
03395 }
03396
03397 VPRIVATE void bcflnew(Vpmg *thee){
03398
03399     int i,j,k, iatom, igrid;
03400     int x0, x1, y0, y1, z0, z1;
03401
03402     int nx, ny, nz;
03403     int natoms, ngrid;
03404
03405     double dist, prel, eps_w, eps_p, T, xkappa;
03406
03407     double *ax, *ay, *az;
03408     double *charge, *size, *val;
03409
03410     double *gx, *gy, *gz;
03411
03412     Vpbe *pbe = thee->pbe;
03413
03414     nx = thee->pmgp->nx;
03415     ny = thee->pmgp->ny;
03416     nz = thee->pmgp->nz;
03417
03418     eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
03419     eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
03420     T = Vpbe_getTemperature(pbe);             /* K */
03421     prel = ((Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T))*(1.0e10);
03422     xkappa = Vpbe_getXkappa(pbe);
03423
03424     natoms = Valist_getNumberAtoms(thee->pbe->alist);
03425     ngrid = 2*((nx*ny) + (ny*nz) + (nx*nz));
03426
03427     ax = (double*)malloc(natoms * sizeof(double));
03428     ay = (double*)malloc(natoms * sizeof(double));
03429     az = (double*)malloc(natoms * sizeof(double));
03430
03431     charge = (double*)malloc(natoms * sizeof(double));
03432     size = (double*)malloc(natoms * sizeof(double));
03433
03434     gx = (double*)malloc(ngrid * sizeof(double));
03435     gy = (double*)malloc(ngrid * sizeof(double));
03436     gz = (double*)malloc(ngrid * sizeof(double));
03437
03438     val = (double*)malloc(ngrid * sizeof(double));
03439
03440     packAtoms(ax,ay,az,charge,size,thee);
03441     packUnpack(nx,ny,nz,ngrid,gx,gy,gz,val,thee,1);

```



```

03442
03443     if(xkappa > VSMALL){
03444 #pragma omp parallel for default(shared) private(igrid,iatom,dist)
03445         for(igrid=0;igrid<ngrid;igrid++){
03446             for(iatom=0; iatom<natoms; iatom++){
03447                 dist = VSQRT(VSQR(gx[igrid]-ax[iatom]) + VSQR(gy[igrid]-ay[iatom])
03448                     + VSQR(gz[igrid]-az[iatom]));
03449                 val[igrid] += prel*(charge[iatom]/dist)*VEXP(-xkappa*(dist-size[iatom]))
03450                     / (1+xkappa*size[iatom]);
03451             }
03452         }
03453     }else{
03454 #pragma omp parallel for default(shared) private(igrid,iatom,dist)
03455         for(igrid=0;igrid<ngrid;igrid++){
03456             for(iatom=0; iatom<natoms; iatom++){
03457                 dist = VSQRT(VSQR(gx[igrid]-ax[iatom]) + VSQR(gy[igrid]-ay[iatom])
03458                     + VSQR(gz[igrid]-az[iatom]));
03459                 val[igrid] += prel*(charge[iatom]/dist);
03460             }
03461         }
03462     }
03463     packUnpack(nx,ny,nz,ngrid,gx,gy,gz,val,thee,0);
03464
03465     free(ax);
03466     free(ay);
03467     free(az);
03468     free(charge);
03469     free(size);
03470
03471     free(gx);
03472     free(gy);
03473     free(gz);
03474     free(val);
03475 }
03476
03477 VPRIVATE void multipolebc(double r, double kappa, double eps_p,
03478     double eps_w, double rad, double tsr[3]) {
03479     double r2,r3,r5;
03480     double eps_r;
03481     double ka,ka2,ka3;
03482     double kr,kr2,kr3;
03483
03484     /*
03485     Below an attempt is made to explain the potential outside of a
03486     multipole located at the center of spherical cavity of dielectric
03487     eps_p, with dielectric eps_w outside (and possibly kappa > 0).
03488
03489     First, eps_p = 1.0
03490     eps_w = 1.0
03491     kappa = 0.0
03492
03493     The general form for the potential of a traceless multipole tensor
03494     of rank n in vacuum is:
03495
03496     
$$V(r) = (-1)^n \cdot u \cdot \nabla^n (1/r)$$

03497
03498     where
03499     u
03500     is a multipole of order n (3^n components)
03501     u · ∇n (1/r)
03502     is the contraction of u with the nth
03503     derivative of 1/r
03504
03505     for example, if n = 1, the dipole potential is
03506     V_vac(r) = (-1)*(ux*x + uy*y + uz*z)/r^3
03507
03508     This function returns the parts of V(r) for multipoles of
03509     order 0, 1 and 2 that are independent of the contraction.
03510
03511     For the vacuum example, this would be 1/r, -1/r^3 and 3/r^5
03512     respectively.
03513
03514     *** Note that this requires that the quadrupole is
03515     traceless. If not, the diagonal quadrupole potential changes
03516     from
03517     qaa * 3*a^2/r^5
03518     to
03519     qaa * (3*a^2/r^5 - 1/r^3a )
03520     where we sum over the trace; a = x, y and z.
03521
03522     (In other words, the -1/r^3 term cancels for a traceless quadrupole.
03523     qxx + qyy + qzz = 0

```

```

03523     such that
03524     -(qxx + qyy + qzz)/r^3 = 0
03525
03526     For quadrupole with trace:
03527     qxx + qyy + qzz != 0
03528     such that
03529     -(qxx + qyy + qzz)/r^3 != 0
03530 )
03531
03532     =====
03533
03534     eps_p != 1 or eps_w != 1
03535     kappa = 0.0
03536
03537     If the multipole is placed at the center of a sphere with
03538     dielectric eps_p in a solvent of dielectric eps_w, the potential
03539     outside the sphere is the solution to the Laplace equation:
03540
03541     V(r) = 1/eps_w * (2*n+1)*eps_r/(n*(n+1)*eps_r)
03542     * (-1)^n * u . n . Del^n (1/r)
03543     where
03544     eps_r = eps_w / eps_p
03545     is the ratio of solvent to solute dielectric
03546
03547     =====
03548
03549     kappa > 0
03550
03551     Finally, if the region outside the sphere is treated by the linearized
03552     PB equation with Debye-Huckel parameter kappa, the solution is:
03553
03554     V(r) = kappa/eps_w * alpha_n(kappa*a) * K_n(kappa*r) * r^(n+1)/a^n
03555     * (-1)^n * u . n . Del^n (1/r)
03556     where
03557     alpha_n(x) is [(2n + 1) / x] / [(n*K_n(x)/eps_r) - x*K_n'(x)]
03558     K_n(x) are modified spherical Bessel functions of the third kind.
03559     K_n'(x) is the derivative of K_n(x)
03560     */
03561
03562     eps_r = eps_w/eps_p;
03563     r2 = r*r;
03564     r3 = r2*r;
03565     r5 = r3*r2;
03566     tsr[0] = (1.0/eps_w)/r;
03567     tsr[1] = (1.0/eps_w)*(-1.0)/r3;
03568     tsr[2] = (1.0/eps_w)*(3.0)/r5;
03569     if (kappa < VSMALL) {
03570         tsr[1] = (3.0*eps_r)/(1.0 + 2.0*eps_r)*tsr[1];
03571         tsr[2] = (5.0*eps_r)/(2.0 + 3.0*eps_r)*tsr[2];
03572     } else {
03573         ka = kappa*rad;
03574         ka2 = ka*ka;
03575         ka3 = ka2*ka;
03576         kr = kappa*r;
03577         kr2 = kr*kr;
03578         kr3 = kr2*kr;
03579         tsr[0] = exp(ka-kr) / (1.0 + ka) * tsr[0];
03580         tsr[1] = 3.0*eps_r*exp(ka-kr)*(1.0 + kr) * tsr[1];
03581         tsr[1] = tsr[1] / (1.0 + ka + eps_r*(2.0 + 2.0*ka + ka2));
03582         tsr[2] = 5.0*eps_r*exp(ka-kr)*(3.0 + 3.0*kr + kr2) * tsr[2];
03583         tsr[2] = tsr[2]/(6.0+6.0*ka+2.0*ka2+eps_r*(9.0+9.0*ka+4.0*ka2+ka3));
03584     }
03585 }
03586
03587 VPRIVATE void bcf1_sdh(Vpmpg *thee){
03588
03589     int i,j,k,iatom;
03590     int nx, ny, nz;
03591
03592     double size, *position, charge, xkappa, eps_w, eps_p, T, pre, dist;
03593     double sdhcharge, sdhdipole[3], traced[9], sdhquadrupole[9];
03594     double *dipole, *quadrupole;
03595
03596     double val, *apos, gpos[3], tensor[3], qave;
03597     double ux, uy, uz, xr, yr, zr;
03598     double qxx,qxy,qxz,qyx,qyy,qyz,qzx,qzy,qzz;
03599
03600     double *xf, *yf, *zf;
03601     double *gxcf, *gycf, *gzcf;
03602
03603     Vpbe *pbe;

```

```

03604     Vatom *atom;
03605     Valist *alist;
03606
03607     pbe = thee->pbe;
03608     alist = thee->pbe->alist;
03609     nx = thee->pmgpp->nx;
03610     ny = thee->pmgpp->ny;
03611     nz = thee->pmgpp->nz;
03612
03613     xf = thee->xf;
03614     yf = thee->yf;
03615     zf = thee->zf;
03616
03617     gxcf = thee->gxcf;
03618     gycf = thee->gycf;
03619     gzcf = thee->gzcf;
03620
03621     /* For each "atom" (only one for bcfl=1), we use the following formula to
03622      * calculate the boundary conditions:
03623      *  $g(x) = \frac{q e_c}{4\pi\epsilon_0\epsilon_w k_b T}$ 
03624      *  $\frac{\exp(-\kappa(d-a))}{1+\kappa a}$ 
03625      *  $\frac{1}{d}$ 
03626      * where  $d = ||x - x_0||$  (in m) and  $a$  is the size of the atom (in m).
03627      * We only need to evaluate some of these prefactors once:
03628      *  $prel = \frac{e_c}{4\pi\epsilon_0\epsilon_w k_b T}$ 
03629      * which gives the potential as
03630      *  $g(x) = prel * q/d * \frac{\exp(-\kappa(d-a))}{1+\kappa a}$ 
03631      */
03632     eps_w = Vpbe_getSolventDiel(pbe); /* Dimensionless */
03633     eps_p = Vpbe_getSoluteDiel(pbe); /* Dimensionless */
03634     T = Vpbe_getTemperature(pbe); /* K */
03635
03636     pre = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*
Vunit_kb*T);
03637     pre = pre*(1.0e10);
03638
03639     /* Finally, if we convert keep  $\kappa$  in  $\text{\AA}^{-1}$  and scale  $prel$  by
03640      *  $m/\text{\AA}$ , then we will only need to deal with distances and sizes in
03641      * Angstroms rather than meters. */
03642     kappa = Vpbe_getKappa(pbe); /*  $\text{\AA}^{-1}$  */
03643
03644     /* Solute size and position */
03645     size = Vpbe_getSoluteRadius(pbe);
03646     position = Vpbe_getSoluteCenter(pbe);
03647
03648     sdhcharge = 0.0;
03649     for (i=0; i<3; i++) sdhdipole[i] = 0.0;
03650     for (i=0; i<9; i++) sdhquadrupole[i] = 0.0;
03651
03652     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
03653         atom = Valist_getAtom(alist, iatom);
03654         apos = Vatom_getPosition(atom);
03655         xr = apos[0] - position[0];
03656         yr = apos[1] - position[1];
03657         zr = apos[2] - position[2];
03658         switch (thee->chargeSrc) {
03659             case VCM_CHARGE:
03660                 charge = Vatom_getCharge(atom);
03661                 sdhcharge += charge;
03662                 sdhdipole[0] += xr * charge;
03663                 sdhdipole[1] += yr * charge;
03664                 sdhdipole[2] += zr * charge;
03665                 traced[0] = xr*xr*charge;
03666                 traced[1] = xr*yr*charge;
03667                 traced[2] = xr*zr*charge;
03668                 traced[3] = yr*xr*charge;
03669                 traced[4] = yr*yr*charge;
03670                 traced[5] = yr*zr*charge;
03671                 traced[6] = zr*xr*charge;
03672                 traced[7] = zr*yr*charge;
03673                 traced[8] = zr*zr*charge;
03674                 qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03675                 sdhquadrupole[0] += 1.5*(traced[0] - qave);
03676                 sdhquadrupole[1] += 1.5*(traced[1]);
03677                 sdhquadrupole[2] += 1.5*(traced[2]);
03678                 sdhquadrupole[3] += 1.5*(traced[3]);
03679                 sdhquadrupole[4] += 1.5*(traced[4] - qave);
03680                 sdhquadrupole[5] += 1.5*(traced[5]);
03681                 sdhquadrupole[6] += 1.5*(traced[6]);
03682                 sdhquadrupole[7] += 1.5*(traced[7]);
03683                 sdhquadrupole[8] += 1.5*(traced[8] - qave);

```

```

03684 #if defined(WITH_TINKER)
03685     case VCM_PERMANENT:
03686         charge = Vatom_getCharge(atom);
03687         dipole = Vatom_getDipole(atom);
03688         quadrupole = Vatom_getQuadrupole(atom);
03689         sdhcharge += charge;
03690         sdhdipole[0] += xr * charge;
03691         sdhdipole[1] += yr * charge;
03692         sdhdipole[2] += zr * charge;
03693         traced[0] = xr*xr*charge;
03694         traced[1] = xr*yr*charge;
03695         traced[2] = xr*zr*charge;
03696         traced[3] = yr*xr*charge;
03697         traced[4] = yr*yr*charge;
03698         traced[5] = yr*zr*charge;
03699         traced[6] = zr*xr*charge;
03700         traced[7] = zr*yr*charge;
03701         traced[8] = zr*zr*charge;
03702         sdhdipole[0] += dipole[0];
03703         sdhdipole[1] += dipole[1];
03704         sdhdipole[2] += dipole[2];
03705         traced[0] += 2.0*xr*dipole[0];
03706         traced[1] += xr*dipole[1] + yr*dipole[0];
03707         traced[2] += xr*dipole[2] + zr*dipole[0];
03708         traced[3] += yr*dipole[0] + xr*dipole[1];
03709         traced[4] += 2.0*yr*dipole[1];
03710         traced[5] += yr*dipole[2] + zr*dipole[1];
03711         traced[6] += zr*dipole[0] + xr*dipole[2];
03712         traced[7] += zr*dipole[1] + yr*dipole[2];
03713         traced[8] += 2.0*zr*dipole[2];
03714         gave = (traced[0] + traced[4] + traced[8]) / 3.0;
03715         sdhquadrupole[0] += 1.5*(traced[0] - gave);
03716         sdhquadrupole[1] += 1.5*(traced[1]);
03717         sdhquadrupole[2] += 1.5*(traced[2]);
03718         sdhquadrupole[3] += 1.5*(traced[3]);
03719         sdhquadrupole[4] += 1.5*(traced[4] - gave);
03720         sdhquadrupole[5] += 1.5*(traced[5]);
03721         sdhquadrupole[6] += 1.5*(traced[6]);
03722         sdhquadrupole[7] += 1.5*(traced[7]);
03723         sdhquadrupole[8] += 1.5*(traced[8] - gave);
03724         sdhquadrupole[0] += quadrupole[0];
03725         sdhquadrupole[1] += quadrupole[1];
03726         sdhquadrupole[2] += quadrupole[2];
03727         sdhquadrupole[3] += quadrupole[3];
03728         sdhquadrupole[4] += quadrupole[4];
03729         sdhquadrupole[5] += quadrupole[5];
03730         sdhquadrupole[6] += quadrupole[6];
03731         sdhquadrupole[7] += quadrupole[7];
03732         sdhquadrupole[8] += quadrupole[8];
03733     case VCM_INDUCED:
03734         dipole = Vatom_getInducedDipole(atom);
03735         sdhdipole[0] += dipole[0];
03736         sdhdipole[1] += dipole[1];
03737         sdhdipole[2] += dipole[2];
03738         traced[0] = 2.0*xr*dipole[0];
03739         traced[1] = xr*dipole[1] + yr*dipole[0];
03740         traced[2] = xr*dipole[2] + zr*dipole[0];
03741         traced[3] = yr*dipole[0] + xr*dipole[1];
03742         traced[4] = 2.0*yr*dipole[1];
03743         traced[5] = yr*dipole[2] + zr*dipole[1];
03744         traced[6] = zr*dipole[0] + xr*dipole[2];
03745         traced[7] = zr*dipole[1] + yr*dipole[2];
03746         traced[8] = 2.0*zr*dipole[2];
03747         gave = (traced[0] + traced[4] + traced[8]) / 3.0;
03748         sdhquadrupole[0] += 1.5*(traced[0] - gave);
03749         sdhquadrupole[1] += 1.5*(traced[1]);
03750         sdhquadrupole[2] += 1.5*(traced[2]);
03751         sdhquadrupole[3] += 1.5*(traced[3]);
03752         sdhquadrupole[4] += 1.5*(traced[4] - gave);
03753         sdhquadrupole[5] += 1.5*(traced[5]);
03754         sdhquadrupole[6] += 1.5*(traced[6]);
03755         sdhquadrupole[7] += 1.5*(traced[7]);
03756         sdhquadrupole[8] += 1.5*(traced[8] - gave);
03757     case VCM_NLINDUCED:
03758         dipole = Vatom_getNLInducedDipole(atom);
03759         sdhdipole[0] += dipole[0];
03760         sdhdipole[1] += dipole[1];
03761         sdhdipole[2] += dipole[2];
03762         traced[0] = 2.0*xr*dipole[0];
03763         traced[1] = xr*dipole[1] + yr*dipole[0];
03764         traced[2] = xr*dipole[2] + zr*dipole[0];

```

```

03765         traced[3] = yr*dipole[0] + xr*dipole[1];
03766         traced[4] = 2.0*yr*dipole[1];
03767         traced[5] = yr*dipole[2] + zr*dipole[1];
03768         traced[6] = zr*dipole[0] + xr*dipole[2];
03769         traced[7] = zr*dipole[1] + yr*dipole[2];
03770         traced[8] = 2.0*zr*dipole[2];
03771         gave = (traced[0] + traced[4] + traced[8]) / 3.0;
03772         sdhquadrupole[0] += 1.5*(traced[0] - gave);
03773         sdhquadrupole[1] += 1.5*(traced[1]);
03774         sdhquadrupole[2] += 1.5*(traced[2]);
03775         sdhquadrupole[3] += 1.5*(traced[3]);
03776         sdhquadrupole[4] += 1.5*(traced[4] - gave);
03777         sdhquadrupole[5] += 1.5*(traced[5]);
03778         sdhquadrupole[6] += 1.5*(traced[6]);
03779         sdhquadrupole[7] += 1.5*(traced[7]);
03780         sdhquadrupole[8] += 1.5*(traced[8] - gave);
03781 #endif /* if defined(WITH_TINKER) */
03782     }
03783 }
03784
03785 ux = sdhdipole[0];
03786 uy = sdhdipole[1];
03787 uz = sdhdipole[2];
03788
03789 /* The factor of 1/3 results from using a
03790 traceless quadrupole definition. See, for example,
03791 "The Theory of Intermolecular Forces" by A.J. Stone,
03792 Chapter 3. */
03793 qxx = sdhquadrupole[0] / 3.0;
03794 qxy = sdhquadrupole[1] / 3.0;
03795 qxz = sdhquadrupole[2] / 3.0;
03796 qyx = sdhquadrupole[3] / 3.0;
03797 qyy = sdhquadrupole[4] / 3.0;
03798 qyz = sdhquadrupole[5] / 3.0;
03799 qzx = sdhquadrupole[6] / 3.0;
03800 qzy = sdhquadrupole[7] / 3.0;
03801 qzz = sdhquadrupole[8] / 3.0;
03802
03803 for(k=0;k<nz;k++){
03804     gpos[2] = zf[k];
03805     for(j=0;j<ny;j++){
03806         gpos[1] = yf[j];
03807         for(i=0;i<nx;i++){
03808             gpos[0] = xf[i];
03809             if(gridPointIsValid(i, j, k, nx, ny, nz)){
03810                 xr = gpos[0] - position[0];
03811                 yr = gpos[1] - position[1];
03812                 zr = gpos[2] - position[2];
03813
03814                 dist = VSQR(VSQR(xr) + VSQR(yr) + VSQR(zr));
03815                 multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
03816
03817                 val = pre*sdhcharge*tensor[0];
03818                 val -= pre*ux*xr*tensor[1];
03819                 val -= pre*uy*yr*tensor[1];
03820                 val -= pre*uz*zr*tensor[1];
03821                 val += pre*qxx*xr*xr*tensor[2];
03822                 val += pre*qyy*yr*yr*tensor[2];
03823                 val += pre*qzz*zr*zr*tensor[2];
03824                 val += pre*2.0*qxy*xr*yr*tensor[2];
03825                 val += pre*2.0*qxz*xr*zr*tensor[2];
03826                 val += pre*2.0*qyz*yr*zr*tensor[2];
03827
03828                 if(i==0){
03829                     gxcf[IJKx(j,k,0)] = val;
03830                 }
03831                 if(i==nx-1){
03832                     gxcf[IJKx(j,k,1)] = val;
03833                 }
03834                 if(j==0){
03835                     gycf[IJKy(i,k,0)] = val;
03836                 }
03837                 if(j==ny-1){
03838                     gycf[IJKy(i,k,1)] = val;
03839                 }
03840                 if(k==0){
03841                     gzcf[IJKz(i,j,0)] = val;
03842                 }
03843                 if(k==nz-1){
03844                     gzcf[IJKz(i,j,1)] = val;
03845                 }

```

```

03846         } /* End grid point is valid */
03847     } /* End i loop */
03848 } /* End j loop */
03849 } /* End k loop */
03850
03851 }
03852
03853 VPRIVATE void bcfl_mdh(Vpmg *thee){
03854
03855     int i,j,k,iatom;
03856     int nx, ny, nz;
03857
03858     double val, *apos, gpos[3];
03859     double *dipole, *quadrupole;
03860     double size, charge, xkappa, eps_w, eps_p, T, pre1, dist;
03861
03862     double *xf, *yf, *zf;
03863     double *gxcf, *gycf, *gzcf;
03864
03865     Vpbe *pbe;
03866     Vatom *atom;
03867     Valist *alist;
03868
03869     pbe = thee->pbe;
03870     alist = thee->pbe->alist;
03871     nx = thee->pmgp->nx;
03872     ny = thee->pmgp->ny;
03873     nz = thee->pmgp->nz;
03874
03875     xf = thee->xf;
03876     yf = thee->yf;
03877     zf = thee->zf;
03878
03879     gxcf = thee->gxcf;
03880     gycf = thee->gycf;
03881     gzcf = thee->gzcf;
03882
03883     /* For each "atom" (only one for bcfl=1), we use the following formula to
03884     * calculate the boundary conditions:
03885     * 
$$g(x) = \frac{q}{4\pi\epsilon_0\epsilon_w k_b T} \frac{\exp(-xkappa(d-a))}{1+xkappa a}$$

03886     * where  $d = ||x - x_0||$  (in m) and  $a$  is the size of the atom (in m).
03887     * We only need to evaluate some of these prefactors once:
03888     *  $pre1 = \frac{q}{4\pi\epsilon_0\epsilon_w k_b T}$ 
03889     * which gives the potential as
03890     *  $g(x) = pre1 * q/d * \frac{\exp(-xkappa(d-a))}{1+xkappa a}$ 
03891     */
03892     eps_w = Vpbe_getSolventDiel(pbe); /* Dimensionless */
03893     eps_p = Vpbe_getSoluteDiel(pbe); /* Dimensionless */
03894     T = Vpbe_getTemperature(pbe); /* K */
03895     pre1 = (Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T);
03896
03897     /* Finally, if we convert keep xkappa in A-1 and scale pre1 by
03898     * m/A, then we will only need to deal with distances and sizes in
03899     * Angstroms rather than meters. */
03900     xkappa = Vpbe_getXkappa(pbe); /* A-1 */
03901     pre1 = pre1*(1.0e10);
03902
03903     /* Finally, if we convert keep xkappa in A-1 and scale pre1 by
03904     * m/A, then we will only need to deal with distances and sizes in
03905     * Angstroms rather than meters. */
03906     xkappa = Vpbe_getXkappa(pbe); /* A-1 */
03907
03908     for(k=0;k<nz;k++){
03909         gpos[2] = zf[k];
03910         for(j=0;j<ny;j++){
03911             gpos[1] = yf[j];
03912             for(i=0;i<nx;i++){
03913                 gpos[0] = xf[i];
03914                 if(gridPointIsValid(i, j, k, nx, ny, nz)){
03915                     val = 0.0;
03916
03917                     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
03918                         atom = Valist_getAtom(alist, iatom);
03919                         apos = Vatom_getPosition(atom);
03920                         charge = Vunit_ec*Vatom_getCharge(atom);
03921                         size = Vatom_getRadius(atom);
03922
03923                         dist = VSQRT(VSQRT(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])

```

```

03927             + VSQR(gpos[2]-apos[2]));
03928         if (xkappa > VSMALL) {
03929             val += prel*(charge/dist)*VEXP(-xkappa*(dist-size))
03930             / (1+xkappa*size);
03931         } else {
03932             val += prel*(charge/dist);
03933         }
03934     }
03935 }
03936
03937     if(i==0){
03938         gxcf[IJKx(j,k,0)] = val;
03939     }
03940     if(i==nx-1){
03941         gxcf[IJKx(j,k,1)] = val;
03942     }
03943     if(j==0){
03944         gycf[IJKy(i,k,0)] = val;
03945     }
03946     if(j==ny-1){
03947         gycf[IJKy(i,k,1)] = val;
03948     }
03949     if(k==0){
03950         gzcf[IJKz(i,j,0)] = val;
03951     }
03952     if(k==nz-1){
03953         gzcf[IJKz(i,j,1)] = val;
03954     }
03955     } /* End grid point is valid */
03956 } /* End i loop */
03957 } /* End j loop */
03958 } /* End k loop */
03959
03960 }
03961
03962 /* ////////////////////////////////////////
03963 // Routine:  bcfl_mem
03964 //
03965 // Purpose:  Increment all the boundary points by the
03966 //           analytic expression for a membrane system in
03967 //           the presence of a membrane potential. This
03968 //           Boundary flag should only be used for systems
03969 //           that explicitly have membranes in the dielectric
03970 //           and solvent maps.
03971 //
03972 //           There should be several input variables add to this
03973 //           function such as membrane potential, membrane thickness
03974 //           and height.
03975 //
03976 // Args:     apos is a 3-vector
03977 //
03978 // Author:   Michael Grabe
03979
03980 VPRIVATE void bcfl_mem(double zmem, double L, double eps_m, double eps_w,
03981 double V, double xkappa, double *gxcf, double *gycf, double *gzcf,
03982 double *xf, double *yf, double *zf, int nx, int ny, int nz) {
03983
03984     /* some definitions */
03985     /* L = total length of the membrane */
03986     /* xkappa = inverse Debye length */
03987     /* zmem = z value of membrane bottom (Cytoplasm) */
03988     /* V = electrical potential inside the cell */
03989     int i, j, k;
03990     double dist, val, z_low, z_high, z_shift;
03991     double A, B, C, D, edge_L, l;
03992     double G, z_0, z_rel;
03993     double gpos[3];
03994
03995     Vnm_print(0, "Here is the value of kappa: %f\n",xkappa);
03996     Vnm_print(0, "Here is the value of L: %f\n",L);
03997     Vnm_print(0, "Here is the value of zmem: %f\n",zmem);
04000     Vnm_print(0, "Here is the value of mdie: %f\n",eps_m);
04001     Vnm_print(0, "Here is the value of memv: %f\n",V);
04002
04003     /* no salt symmetric BC's at +/- infinity */
04004     // B=V/(edge_L - l*(1-eps_w/eps_m));
04005     // A=V + B*edge_L;
04006     // D=eps_w/eps_m*B;
04007     z_low = zmem; /* this defines the bottom of the membrane */
04008     z_high = zmem + L; /* this is the top of the membrane */
04009
04010     /*****

```

```

04011      /* proper boundary conditions for V = 0 extracellular */
04012      /* and psi=-V cytoplasm. */
04013      /* Implicit in this formulation is that the membrane */
04014      /* center be at z = 0 */
04015      /* **** */
04016
04017      l=L/2; /* half of the membrane length */
04018      z_0 = z_low + 1; /* center of the membrane */
04019      G=l*eps_w/eps_m*xkappa;
04020      A=-V/2*(1/(G+1))*exp(xkappa*l);
04021      B=V/2;
04022      C=-V/2*eps_w/eps_m*xkappa*(1/(G+1));
04023      D=-A;
04024      /* The analytic expression for the boundary conditions */
04025      /* had the cytoplasmic surface of the membrane set to zero. */
04026      /* This requires an off-set of the BC equations. */
04027
04028      /* the "i" boundaries (dirichlet) */
04029      for (k=0; k<nz; k++) {
04030          gpos[2] = zf[k];
04031          z_rel = gpos[2] - z_0; /* relative position for BCs */
04032
04033          for (j=0; j<ny; j++) {
04034
04035              if (gpos[2] <= z_low) { /* cytoplasmic */
04036
04037                  val = A*exp(xkappa*z_rel) + V;
04038                  gxcf[IJKx(j,k,0)] += val; /* assign low side BC */
04039                  gxcf[IJKx(j,k,1)] += val; /* assign high side BC */
04040
04041              }
04042
04043              else if (gpos[2] > z_low && gpos[2] <= z_high) { /* in membrane */
04044
04045                  val = B + C*z_rel;
04046                  gxcf[IJKx(j,k,0)] += val; /* assign low side BC */
04047                  gxcf[IJKx(j,k,1)] += val; /* assign high side BC */
04048
04049              }
04050
04051              else if (gpos[2] > z_high) { /* extracellular */
04052
04053                  val = D*exp(-xkappa*z_rel);
04054                  gxcf[IJKx(j,k,0)] += val; /* assign low side BC */
04055                  gxcf[IJKx(j,k,1)] += val; /* assign high side BC */
04056
04057              }
04058
04059          }
04060      }
04061
04062      /* the "j" boundaries (dirichlet) */
04063      for (k=0; k<nz; k++) {
04064          gpos[2] = zf[k];
04065          z_rel = gpos[2] - z_0;
04066          for (i=0; i<nix; i++) {
04067
04068              if (gpos[2] <= z_low) { /* cytoplasmic */
04069
04070                  val = A*exp(xkappa*z_rel) + V;
04071                  gycf[IJKy(i,k,0)] += val; /* assign low side BC */
04072                  gycf[IJKy(i,k,1)] += val; /* assign high side BC */
04073                  //printf("%f \n",val);
04074
04075              }
04076
04077              else if (gpos[2] > z_low && gpos[2] <= z_high) { /* in membrane */
04078
04079                  val = B + C*z_rel;
04080                  gycf[IJKy(i,k,0)] += val; /* assign low side BC */
04081                  gycf[IJKy(i,k,1)] += val; /* assign high side BC */
04082                  //printf("%f \n",val);
04083
04084              }
04085              else if (gpos[2] > z_high) { /* extracellular */
04086
04087                  val = D*exp(-xkappa*z_rel);
04088                  gycf[IJKy(i,k,0)] += val; /* assign low side BC */
04089                  gycf[IJKy(i,k,1)] += val; /* assign high side BC */
04090                  //printf("%f \n",val);
04091

```



```

04092     }
04093 }
04094 }
04095 }
04096
04097 /* the "k" boundaries (dirichlet) */
04098 for (j=0; j<ny; j++) {
04099     for (i=0; i<nx; i++) {
04100
04101         /* first assign the bottom boundary */
04102
04103         gpos[2] = zf[0];
04104         z_rel = gpos[2] - z_0;
04105
04106         if (gpos[2] <= z_low) {                                     /* cytoplasmic */
04107
04108             val = A*exp(xkappa*z_rel) + V;
04109             gzcf[IJKz(i,j,0)] += val;      /* assign low side BC */
04110             //printf("%f \n",val);
04111         }
04112
04113         else if (gpos[2] > z_low && gpos[2] <= z_high) { /* in membrane */
04114
04115             val = B + C*z_rel;
04116             gzcf[IJKz(i,j,0)] += val;      /* assign low side BC */
04117         }
04118
04119         else if (gpos[2] > z_high) {                                     /* extracellular */
04120
04121             val = D*exp(-xkappa*z_rel);
04122             gzcf[IJKz(i,j,0)] += val;      /* assign low side BC */
04123         }
04124
04125         /* now assign the top boundary */
04126
04127         gpos[2] = zf[nz-1];
04128         z_rel = gpos[2] - z_0;
04129
04130         if (gpos[2] <= z_low) {                                     /* cytoplasmic */
04131
04132             val = A*exp(xkappa*z_rel) + V;
04133             gzcf[IJKz(i,j,1)] += val;      /* assign high side BC */
04134         }
04135
04136         else if (gpos[2] > z_low && gpos[2] <= z_high) { /* in membrane */
04137
04138             val = B + C*z_rel;
04139             gzcf[IJKz(i,j,1)] += val;      /* assign high side BC */
04140         }
04141
04142         else if (gpos[2] > z_high) {                                     /* extracellular */
04143
04144             val = D*exp(-xkappa*z_rel);
04145             gzcf[IJKz(i,j,1)] += val;      /* assign high side BC */
04146             //printf("%f \n",val);
04147         }
04148     }
04149 }
04150 }
04151
04152 VPRIVATE void bcfl_map(Vpmg *thee){
04153
04154     Vpbe *pbe;
04155     double position[3], pot, hx, hy, hzed;
04156     int i, j, k, nx, ny, nz, rc;
04157
04158     VASSERT(thee != VNULL);
04159
04160     /* Mesh info */
04161     nx = thee->pmgp->nx;
04162     ny = thee->pmgp->ny;
04163     nz = thee->pmgp->nz;
04164     hx = thee->pmgp->hx;

```

```

04173     hy = thee->pmgp->hy;
04174     hzed = thee->pmgp->hzed;
04175
04176     /* Reset the potential array */
04177     for (i=0; i<(nx*ny*nz); i++) thee->pot[i] = 0.0;
04178
04179     /* Fill in the source term (atomic potentials) */
04180     Vnm_print(0, "Vpmg_fillco: filling in source term.\n");
04181     for (k=0; k<nz; k++) {
04182         for (j=0; j<ny; j++) {
04183             for (i=0; i<nx; i++) {
04184                 position[0] = thee->xf[i];
04185                 position[1] = thee->yf[j];
04186                 position[2] = thee->zf[k];
04187                 rc = Vgrid_value(thee->potMap, position, &pot);
04188                 if (!rc) {
04189                     Vnm_print(2, "fillcoChargeMap: Error -- fell off of potential map at (%g, %g, %g)!\n",
04190                             position[0], position[1], position[2]);
04191                     VASSERT(0);
04192                 }
04193                 thee->pot[IJK(i,j,k)] = pot;
04194             }
04195         }
04196     }
04197 }
04198 }
04199
04200 #if defined(WITH_TINKER)
04201 VPRIVATE void bcfl_mdh_tinker(Vpmg *thee) {
04202
04203     int i,j,k,iatom;
04204     int nx, ny, nz;
04205
04206     double val, *apos, gpos[3], tensor[9];
04207     double *dipole, *quadrupole;
04208     double size, charge, xkappa, eps_w, eps_p, T, pre1, dist;
04209
04210     double ux,uy,uz,xr,yr,zr;
04211     double qxx,qxy,qxz,qyx,qyy,qyz,qzx,qzy,qzz;
04212
04213     double *xf, *yf, *zf;
04214     double *gxcf, *gycf, *gzcf;
04215
04216     Vpbe *pbe;
04217     Vatom *atom;
04218     Valist *alist;
04219
04220     pbe = thee->pbe;
04221     alist = thee->pbe->alist;
04222     nx = thee->pmgp->nx;
04223     ny = thee->pmgp->ny;
04224     nz = thee->pmgp->nz;
04225
04226     xf = thee->xf;
04227     yf = thee->yf;
04228     zf = thee->zf;
04229
04230     gxcf = thee->gxcf;
04231     gycf = thee->gycf;
04232     gzcf = thee->gzcf;
04233
04234     /* For each "atom" (only one for bcfl=1), we use the following formula to
04235     * calculate the boundary conditions:
04236     * 
$$g(x) = \frac{q}{4\pi\epsilon_0\epsilon_w k_B T} \frac{\exp(-xkappa(d-a))}{1+xkappa a}$$

04237     * where  $d = ||x - x_0||$  (in m) and  $a$  is the size of the atom (in m).
04238     * We only need to evaluate some of these prefactors once:
04239     *  $pre1 = \frac{q}{4\pi\epsilon_0\epsilon_w k_B T}$ 
04240     * which gives the potential as
04241     * 
$$g(x) = pre1 * \frac{q}{d} \frac{\exp(-xkappa(d-a))}{1+xkappa a}$$

04242     */
04243     eps_w = Vpbe_getSolventDiel(pbe); /* Dimensionless */
04244     eps_p = Vpbe_getSoluteDiel(pbe); /* Dimensionless */
04245     T = Vpbe_getTemperature(pbe); /* K */
04246     pre1 = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*
04247             Vunit_kb*T);
04248
04249     /* Finally, if we convert keep xkappa in A-1 and scale pre1 by
04250     * m/A, then we will only need to deal with distances and sizes in
04251     * Angstroms rather than meters. */

```

```

04253     xkappa = Vpbe_getXkappa(pbe);                /* A-1 */
04254     prel = prel*(1.0e10);
04255
04256     /* Finally, if we convert keep xkappa in A-1 and scale prel by
04257      * m/A, then we will only need to deal with distances and sizes in
04258      * Angstroms rather than meters. */
04259     xkappa = Vpbe_getXkappa(pbe);                /* A-1 */
04260
04261     for(k=0;k<nz;k++){
04262         gpos[2] = zf[k];
04263         for(j=0;j<ny;j++){
04264             gpos[1] = yf[j];
04265             for(i=0;i<nx;i++){
04266                 gpos[0] = xf[i];
04267                 if(gridPointIsValid(i, j, k, nx, ny, nz)){
04268
04269                     val = 0.0;
04270
04271                     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
04272                         atom = Valist_getAtom(alist, iatom);
04273                         apos = Vatom_getPosition(atom);
04274                         size = Vatom_getRadius(atom);
04275
04276                         charge = 0.0;
04277
04278                         dipole = VNULL;
04279                         quadrupole = VNULL;
04280
04281                         if (three->chargeSrc == VCM_PERMANENT) {
04282                             charge = Vatom_getCharge(atom);
04283                             dipole = Vatom_getDipole(atom);
04284                             quadrupole = Vatom_getQuadrupole(atom);
04285                         } else if (three->chargeSrc == VCM_INDUCED) {
04286                             dipole = Vatom_getInducedDipole(atom);
04287                         } else {
04288                             dipole = Vatom_getNLInducedDipole(atom);
04289                         }
04290
04291                         ux = dipole[0];
04292                         uy = dipole[1];
04293                         uz = dipole[2];
04294
04295                         if (quadrupole != VNULL) {
04296                             /* The factor of 1/3 results from using a
04297                              * traceless quadrupole definition. See, for example,
04298                              * "The Theory of Intermolecular Forces" by A.J. Stone,
04299                              * Chapter 3. */
04300                             qxx = quadrupole[0] / 3.0;
04301                             qxy = quadrupole[1] / 3.0;
04302                             qxz = quadrupole[2] / 3.0;
04303                             qyx = quadrupole[3] / 3.0;
04304                             qyy = quadrupole[4] / 3.0;
04305                             qyz = quadrupole[5] / 3.0;
04306                             qzx = quadrupole[6] / 3.0;
04307                             qzy = quadrupole[7] / 3.0;
04308                             qzz = quadrupole[8] / 3.0;
04309                         } else {
04310                             qxx = 0.0;
04311                             qxy = 0.0;
04312                             qxz = 0.0;
04313                             qyx = 0.0;
04314                             qyy = 0.0;
04315                             qyz = 0.0;
04316                             qzx = 0.0;
04317                             qzy = 0.0;
04318                             qzz = 0.0;
04319                         }
04320
04321                         xr = gpos[0] - apos[0];
04322                         yr = gpos[1] - apos[1];
04323                         zr = gpos[2] - apos[2];
04324
04325                         dist = VSQR(VSQR(xr) + VSQR(yr) + VSQR(zr));
04326                         multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
04327
04328                         val += prel*charge*tensor[0];
04329                         val -= prel*ux*xr*tensor[1];
04330                         val -= prel*uy*yr*tensor[1];
04331                         val -= prel*uz*zr*tensor[1];
04332                         val += prel*qxx*xr*xr*tensor[2];
04333                         val += prel*qyy*yr*yr*tensor[2];

```

```

04334         val += prel*qzz*zr*zr*tensor[2];
04335         val += prel*2.0*qxy*xr*yr*tensor[2];
04336         val += prel*2.0*qxz*xr*zr*tensor[2];
04337         val += prel*2.0*qyz*yr*zr*tensor[2];
04338
04339     }
04340
04341     if(i==0){
04342         gxcf[IJKx(j,k,0)] = val;
04343     }
04344     if(i==nx-1){
04345         gxcf[IJKx(j,k,1)] = val;
04346     }
04347     if(j==0){
04348         gycf[IJKy(i,k,0)] = val;
04349     }
04350     if(j==ny-1){
04351         gycf[IJKy(i,k,1)] = val;
04352     }
04353     if(k==0){
04354         gzcf[IJKz(i,j,0)] = val;
04355     }
04356     if(k==nz-1){
04357         gzcf[IJKz(i,j,1)] = val;
04358     }
04359     } /* End grid point is valid */
04360 } /* End i loop */
04361 } /* End j loop */
04362 } /* End k loop */
04363
04364 }
04365 #endif
04366
04367 VPRIVATE void bcCalc(Vpmg *thee){
04368
04369     int i, j, k;
04370     int nx, ny, nz;
04371
04372     double zmem, eps_m, Lmem, memv, eps_w, xkappa;
04373
04374     nx = thee->pmgp->nx;
04375     ny = thee->pmgp->ny;
04376     nz = thee->pmgp->nz;
04377
04378     /* Zero out the boundaries */
04379     /* the "i" boundaries (dirichlet) */
04380     for (k=0; k<nz; k++) {
04381         for (j=0; j<ny; j++) {
04382             thee->gxcf[IJKx(j,k,0)] = 0.0;
04383             thee->gxcf[IJKx(j,k,1)] = 0.0;
04384             thee->gxcf[IJKx(j,k,2)] = 0.0;
04385             thee->gxcf[IJKx(j,k,3)] = 0.0;
04386         }
04387     }
04388
04389     /* the "j" boundaries (dirichlet) */
04390     for (k=0; k<nz; k++) {
04391         for (i=0; i<nx; i++) {
04392             thee->gycf[IJKy(i,k,0)] = 0.0;
04393             thee->gycf[IJKy(i,k,1)] = 0.0;
04394             thee->gycf[IJKy(i,k,2)] = 0.0;
04395             thee->gycf[IJKy(i,k,3)] = 0.0;
04396         }
04397     }
04398
04399     /* the "k" boundaries (dirichlet) */
04400     for (j=0; j<ny; j++) {
04401         for (i=0; i<nx; i++) {
04402             thee->gzcf[IJKz(i,j,0)] = 0.0;
04403             thee->gzcf[IJKz(i,j,1)] = 0.0;
04404             thee->gzcf[IJKz(i,j,2)] = 0.0;
04405             thee->gzcf[IJKz(i,j,3)] = 0.0;
04406         }
04407     }
04408
04409     switch (thee->pmgp->bcfl) {
04410         /* If we have zero boundary conditions, we're done */
04411         case BCFL_ZERO:
04412             return;
04413         case BCFL_SDH:
04414             bcfl_sdh(thee);

```

```

04415         break;
04416         case BCFL_MDH:
04417         #if defined(WITH_TINKER)
04418             bcfl_mdh_tinker(thee);
04419         #else
04420
04421         #ifdef DEBUG_MAC_OSX_OCL
04422         #include "mach_chud.h"
04423             uint64_t mbeg = mach_absolute_time();
04424
04425             /*
04426             * If OpenCL is available we use it, otherwise fall back to
04427             * normal route (CPU multithreaded w/ OpenMP)
04428             */
04429             if (kOpenCLAvailable == 1) bcflnewOpenCL(thee);
04430             else bcflnew(thee);
04431
04432             mets_(&mbeg, "MDH");
04433         #else
04434             /* bcfl_mdh(thee); */
04435             bcflnew(thee);
04436         #endif /* DEBUG_MAC_OSX_OCL */
04437
04438         #endif /* WITH_TINKER */
04439         break;
04440         case BCFL_MEM:
04441
04442             zmem = Vpbe_getzmem(thee->pbe);
04443             Lmem = Vpbe_getLmem(thee->pbe);
04444             eps_m = Vpbe_getmembraneDiel(thee->pbe);
04445             memv = Vpbe_getmemv(thee->pbe);
04446
04447             eps_w = Vpbe_getSolventDiel(thee->pbe);
04448             xkappa = Vpbe_getXkappa(thee->pbe);
04449
04450             bcfl_mem(zmem, Lmem, eps_m, eps_w, memv, xkappa,
04451                 thee->gxcf, thee->gycf, thee->gzcf,
04452                 thee->xf, thee->yf, thee->zf, nx, ny, nz);
04453             break;
04454         case BCFL_UNUSED:
04455             Vnm_print(2, "bcCalc: Invalid bcfl (%d)!\n", thee->pmgp->bcfl);
04456             VASSERT(0);
04457             break;
04458         case BCFL_FOCUS:
04459             Vnm_print(2, "VPMG::bcCalc -- not appropriate for focusing!\n");
04460             VASSERT(0);
04461             break;
04462         case BCFL_MAP:
04463             bcfl_map(thee);
04464             focusFillBound(thee, VNULL);
04465             break;
04466         default:
04467             Vnm_print(2, "VPMG::bcCalc -- invalid boundary condition \
04468                 flag (%d)!\n", thee->pmgp->bcfl);
04469             VASSERT(0);
04470             break;
04471     }
04472 }
04473
04474 VPRIVATE void fillCoefMap(Vpmg *thee) {
04475
04476     Vpbe *pbe;
04477     double ionstr, position[3], tkappa, eps, pot, hx, hy, hzed;
04478     int i, j, k, nx, ny, nz;
04479     double kappamax;
04480     VASSERT(thee != VNULL);
04481
04482     /* Get PBE info */
04483     pbe = thee->pbe;
04484     ionstr = Vpbe_getBulkIonicStrength(pbe);
04485
04486     /* Mesh info */
04487     nx = thee->pmgp->nx;
04488     ny = thee->pmgp->ny;
04489     nz = thee->pmgp->nz;
04490     hx = thee->pmgp->hx;
04491     hy = thee->pmgp->hy;
04492     hzed = thee->pmgp->hzed;
04493
04494     if ((!thee->useDielXMap) || (!thee->useDielYMap)
04495         || (!thee->useDielZMap) || (!thee->useKappaMap) && (ionstr>

```

```

VPMGSMALL))) {
04496
04497     Vnm_print(2, "fillcoCoefMap: You need to use all coefficient maps!\n");
04498     VASSERT(0);
04499 }
04500
04501 /* Scale the kappa map to values between 0 and 1
04502    Thus get the maximum value in the map - this
04503    is theoretically unnecessary, but a good check.*/
04504 kappamax = -1.00;
04505 for (k=0; k<nz; k++) {
04506     for (j=0; j<ny; j++) {
04507         for (i=0; i<nx; i++) {
04508             if (ionstr > VPMGSMALL) {
04509                 position[0] = thee->xf[i];
04510                 position[1] = thee->yf[j];
04511                 position[2] = thee->zf[k];
04512                 if (!Vgrid_value(thee->kappaMap, position, &tkappa)) {
04513                     Vnm_print(2, "Vpmg_fillco: Off kappaMap at:\n");
04514                     Vnm_print(2, "Vpmg_fillco: (x,y,z) = (%g,%g %g)\n",
04515                             position[0], position[1], position[2]);
04516                     VASSERT(0);
04517                 }
04518                 if (tkappa > kappamax) {
04519                     kappamax = tkappa;
04520                 }
04521                 if (tkappa < 0.0){
04522                     Vnm_print(2, "Vpmg_fillcoCoefMap: Kappa map less than 0\n");
04523                     Vnm_print(2, "Vpmg_fillcoCoefMap: at (x,y,z) = (%g,%g %g)\n",
04524                             position[0], position[1], position[2]);
04525                     VASSERT(0);
04526                 }
04527             }
04528         }
04529     }
04530 }
04531 }
04532
04533 if (kappamax > 1.0){
04534     Vnm_print(2, "Vpmg_fillcoCoefMap: Maximum Kappa value\n");
04535     Vnm_print(2, "%g is greater than 1 - will scale appropriately!\n",
04536             kappamax);
04537 }
04538 else {
04539     kappamax = 1.0;
04540 }
04541
04542 for (k=0; k<nz; k++) {
04543     for (j=0; j<ny; j++) {
04544         for (i=0; i<nx; i++) {
04545             if (ionstr > VPMGSMALL) {
04546                 position[0] = thee->xf[i];
04547                 position[1] = thee->yf[j];
04548                 position[2] = thee->zf[k];
04549                 if (!Vgrid_value(thee->kappaMap, position, &tkappa)) {
04550                     Vnm_print(2, "Vpmg_fillco: Off kappaMap at:\n");
04551                     Vnm_print(2, "Vpmg_fillco: (x,y,z) = (%g,%g %g)\n",
04552                             position[0], position[1], position[2]);
04553                     VASSERT(0);
04554                 }
04555                 if (tkappa < VPMGSMALL) tkappa = 0.0;
04556                 thee->kappa[IJK(i,j,k)] = (tkappa / kappamax);
04557             }
04558         }
04559     }
04560     position[0] = thee->xf[i] + 0.5*hx;
04561     position[1] = thee->yf[j];
04562     position[2] = thee->zf[k];
04563     if (!Vgrid_value(thee->dielXMap, position, &eps)) {
04564         Vnm_print(2, "Vpmg_fillco: Off dielXMap at:\n");
04565         Vnm_print(2, "Vpmg_fillco: (x,y,z) = (%g,%g %g)\n",
04566                 position[0], position[1], position[2]);
04567         VASSERT(0);
04568     }
04569     thee->epsx[IJK(i,j,k)] = eps;
04570
04571     position[0] = thee->xf[i];
04572     position[1] = thee->yf[j] + 0.5*hy;
04573     position[2] = thee->zf[k];
04574     if (!Vgrid_value(thee->dielYMap, position, &eps)) {
04575         Vnm_print(2, "Vpmg_fillco: Off dielYMap at:\n");

```

```

04576         Vnm_print(2, "Vpmg_fillco: (x,y,z) = (%g,%g %g)\n",
04577                     position[0], position[1], position[2]);
04578         VASSERT(0);
04579     }
04580     thee->epsy[IJK(i,j,k)] = eps;
04581
04582     position[0] = thee->xf[i];
04583     position[1] = thee->yf[j];
04584     position[2] = thee->zf[k] + 0.5*hzed;
04585     if (!Vgrid_value(thee->dielZMap, position, &eps)) {
04586         Vnm_print(2, "Vpmg_fillco: Off dielZMap at:\n");
04587         Vnm_print(2, "Vpmg_fillco: (x,y,z) = (%g,%g %g)\n",
04588                 position[0], position[1], position[2]);
04589         VASSERT(0);
04590     }
04591     thee->epsz[IJK(i,j,k)] = eps;
04592 }
04593 }
04594 }
04595 }
04596
04597 VPRIVATE void fillcoCoefMol(Vpmg *thee) {
04598
04599     if (thee->useDielXMap || thee->useDielYMap || thee->
04600         useDielZMap ||
04601         thee->useKappaMap) {
04602         fillcoCoefMap(thee);
04603     } else {
04604         fillcoCoefMolDiel(thee);
04605         fillcoCoefMolIon(thee);
04606     }
04607 }
04608
04609 }
04610
04611 }
04612
04613 VPRIVATE void fillcoCoefMolIon(Vpmg *thee) {
04614
04615     Vacc *acc;
04616     Valist *alist;
04617     Vpbe *pbe;
04618     Vatom *atom;
04619     double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr;
04620     double xlen, ylen, zlen, irad;
04621     double hx, hy, hzed, *apos, arad;
04622     int i, nx, ny, nz, iatom;
04623     Vsurf_Meth surfMeth;
04624
04625     VASSERT(thee != VNULL);
04626     surfMeth = thee->surfMeth;
04627
04628     /* Get PBE info */
04629     pbe = thee->pbe;
04630     acc = pbe->acc;
04631     alist = pbe->alist;
04632     irad = Vpbe_getMaxIonRadius(pbe);
04633     ionstr = Vpbe_getBulkIonicStrength(pbe);
04634
04635     /* Mesh info */
04636     nx = thee->pmgp->nx;
04637     ny = thee->pmgp->ny;
04638     nz = thee->pmgp->nz;
04639     hx = thee->pmgp->hx;
04640     hy = thee->pmgp->hy;
04641     hzed = thee->pmgp->hzed;
04642
04643     /* Define the total domain size */
04644     xlen = thee->pmgp->xlen;
04645     ylen = thee->pmgp->ylen;
04646     zlen = thee->pmgp->zlen;
04647
04648     /* Define the min/max dimensions */
04649     xmin = thee->pmgp->xcent - (xlen/2.0);
04650     ymin = thee->pmgp->ycent - (ylen/2.0);
04651     zmin = thee->pmgp->zcent - (zlen/2.0);
04652     xmax = thee->pmgp->xcent + (xlen/2.0);
04653     ymax = thee->pmgp->ycent + (ylen/2.0);
04654     zmax = thee->pmgp->zcent + (zlen/2.0);
04655

```

```

04656  /* This is a floating point parameter related to the non-zero nature of the
04657  * bulk ionic strength. If the ionic strength is greater than zero; this
04658  * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
04659  * Otherwise, this parameter is set to 0.0 */
04660  if (ionstr > VPMGSMALL) ionmask = 1.0;
04661  else ionmask = 0.0;
04662
04663  /* Reset the kappa array, marking everything accessible */
04664  for (i=0; i<(nx*ny*nz); i++) thee->kappa[i] = ionmask;
04665
04666  if (ionstr < VPMGSMALL) return;
04667
04668  /* Loop through the atoms and set kappa = 0.0 (inaccessible) if a point
04669  * is inside the ion-inflated van der Waals radii */
04670  for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
04671
04672      atom = Valist_getAtom(alist, iatom);
04673      apos = Vatom_getPosition(atom);
04674      arad = Vatom_getRadius(atom);
04675
04676      if (arad > VSMALL) {
04677
04678          /* Make sure we're on the grid */
04679          if ((apos[0]<(xmin-irad-arad)) || (apos[0]>(xmax+irad+arad)) || \
04680              (apos[1]<(ymin-irad-arad)) || (apos[1]>(ymax+irad+arad)) || \
04681              (apos[2]<(zmin-irad-arad)) || (apos[2]>(zmax+irad+arad))) {
04682              if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
04683                  (thee->pmgp->bcfl != BCFL_MAP)) {
04684                  Vnm_print(2,
04685                      "Vpmg_fillco: Atom #%d at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):\n",
04686                      iatom, apos[0], apos[1], apos[2]);
04687                  Vnm_print(2, "Vpmg_fillco:  xmin = %g, xmax = %g\n",
04688                      xmin, xmax);
04689                  Vnm_print(2, "Vpmg_fillco:  ymin = %g, ymax = %g\n",
04690                      ymin, ymax);
04691                  Vnm_print(2, "Vpmg_fillco:  zmin = %g, zmax = %g\n",
04692                      zmin, zmax);
04693              }
04694              fflush(stderr);
04695          } else { /* if we're on the mesh */
04696
04697              /* Mark ions */
04698              markSphere((irad+arad), apos,
04699                      nx, ny, nz,
04700                      hx, hy, hzed,
04701                      xmin, ymin, zmin,
04702                      thee->kappa, 0.0);
04703
04704              } /* endif (on the mesh) */
04705          }
04706      } /* endfor (over all atoms) */
04707  }
04708
04709 }
04710
04711 VPRIVATE void fillcoCoefMolDiel(Vpmg *thee) {
04712
04713     /* Always call NoSmooth to fill the epsilon arrays */
04714     fillcoCoefMolDielNoSmooth(thee);
04715
04716     /* Call the smoothing algorithm as needed */
04717     if (thee->surfMeth == VSM_MOLSMOOTH) {
04718         fillcoCoefMolDielSmooth(thee);
04719     }
04720 }
04721
04722 VPRIVATE void fillcoCoefMolDielNoSmooth(Vpmg *thee) {
04723
04724     Vacc *acc;
04725     VaccSurf *asurf;
04726     Valist *alist;
04727     Vpbe *pbe;
04728     Vatom *atom;
04729     double xmin, xmax, ymin, ymax, zmin, zmax;
04730     double xlen, ylen, zlen, position[3];
04731     double srad, epsw, epsp, deps, area;
04732     double hx, hy, hzed, *apos, arad;
04733     int i, nx, ny, nz, ntot, iatom, ipt;
04734
04735     /* Get PBE info */
04736     pbe = thee->pbe;

```



```

04737     acc = pbe->acc;
04738     alist = pbe->alist;
04739     srad = Vpbe_getSolventRadius(pbe);
04740     epsw = Vpbe_getSolventDiel(pbe);
04741     epsp = Vpbe_getSoluteDiel(pbe);
04742
04743     /* Mesh info */
04744     nx = thee->pmgp->nx;
04745     ny = thee->pmgp->ny;
04746     nz = thee->pmgp->nz;
04747     hx = thee->pmgp->hx;
04748     hy = thee->pmgp->hy;
04749     hzed = thee->pmgp->hzed;
04750
04751     /* Define the total domain size */
04752     xlen = thee->pmgp->xlen;
04753     ylen = thee->pmgp->ylen;
04754     zlen = thee->pmgp->zlen;
04755
04756     /* Define the min/max dimensions */
04757     xmin = thee->pmgp->xcent - (xlen/2.0);
04758     ymin = thee->pmgp->ycent - (ylen/2.0);
04759     zmin = thee->pmgp->zcent - (zlen/2.0);
04760     xmax = thee->pmgp->xcent + (xlen/2.0);
04761     ymax = thee->pmgp->ycent + (ylen/2.0);
04762     zmax = thee->pmgp->zcent + (zlen/2.0);
04763
04764     /* Reset the arrays */
04765     ntot = nx*ny*nz;
04766     for (i=0; i<ntot; i++) {
04767         thee->epsx[i] = epsw;
04768         thee->epsy[i] = epsw;
04769         thee->epsz[i] = epsw;
04770     }
04771
04772     /* Loop through the atoms and set a{123}cf = 0.0 (inaccessible)
04773      * if a point is inside the solvent-inflated van der Waals radii */
04774     #pragma omp parallel for default(shared) private(iatom,atom,apos,arad)
04775     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
04776
04777         atom = Valist_getAtom(alist, iatom);
04778         apos = Vatom_getPosition(atom);
04779         arad = Vatom_getRadius(atom);
04780
04781         /* Make sure we're on the grid */
04782         if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
04783             (apos[1]<=ymin) || (apos[1]>=ymax) || \
04784             (apos[2]<=zmin) || (apos[2]>=zmax)) {
04785             if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
04786                 (thee->pmgp->bcfl != BCFL_MAP)) {
04787                 Vnm_print(2, "Vpmg_fillco: Atom #d at (%4.3f, %4.3f,\
04788 %4.3f) is off the mesh (ignoring):\n",
04789                     iatom, apos[0], apos[1], apos[2]);
04790                 Vnm_print(2, "Vpmg_fillco: xmin = %g, xmax = %g\n",
04791                     xmin, xmax);
04792                 Vnm_print(2, "Vpmg_fillco: ymin = %g, ymax = %g\n",
04793                     ymin, ymax);
04794                 Vnm_print(2, "Vpmg_fillco: zmin = %g, zmax = %g\n",
04795                     zmin, zmax);
04796             }
04797             fflush(stderr);
04798
04799         } else { /* if we're on the mesh */
04800
04801             if (arad > VSMALL) {
04802                 /* Mark x-shifted dielectric */
04803                 markSphere((arad+srad), apos,
04804                     nx, ny, nz,
04805                     hx, hy, hzed,
04806                     (xmin+0.5*hx), ymin, zmin,
04807                     thee->epsx, epsp);
04808
04809                 /* Mark y-shifted dielectric */
04810                 markSphere((arad+srad), apos,
04811                     nx, ny, nz,
04812                     hx, hy, hzed,
04813                     xmin, (ymin+0.5*hy), zmin,
04814                     thee->epsy, epsp);
04815
04816                 /* Mark z-shifted dielectric */
04817                 markSphere((arad+srad), apos,

```

```

04818             nx, ny, nz,
04819             hx, hy, hzed,
04820             xmin, ymin, (zmin+0.5*hzed),
04821             thee->epsz, epsp);
04822         }
04823     } /* endif (on the mesh) */
04824 } /* endfor (over all atoms) */
04825
04826 area = Vacc_SASA(acc, srاد);
04827
04828 /* We only need to do the next step for non-zero solvent radii */
04829 if (srاد > VSMALL) {
04830     /* Now loop over the solvent accessible surface points */
04831
04832 #pragma omp parallel for default(shared) private(iatom,atom,area,asurf,ipt,position)
04833     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
04834         atom = Valist_getAtom(alist, iatom);
04835         area = Vacc_atomSASA(acc, srاد, atom);
04836         if (area > 0.0 ) {
04837             asurf = Vacc_atomsASPoints(acc, srاد, atom);
04838
04839             /* Use each point on the SAS to reset the solvent accessibility */
04840             /* TODO: Make sure we're not still wasting time here. */
04841             for (ipt=0; ipt<(asurf->npts); ipt++) {
04842                 position[0] = asurf->xpts[ipt];
04843                 position[1] = asurf->ypts[ipt];
04844                 position[2] = asurf->zpts[ipt];
04845
04846                 /* Mark x-shifted dielectric */
04847                 markSphere(srاد, position,
04848                     nx, ny, nz,
04849                     hx, hy, hzed,
04850                     (xmin+0.5*nx), ymin, zmin,
04851                     thee->epsx, epsw);
04852
04853                 /* Mark y-shifted dielectric */
04854                 markSphere(srاد, position,
04855                     nx, ny, nz,
04856                     hx, hy, hzed,
04857                     xmin, (ymin+0.5*ny), zmin,
04858                     thee->epsy, epsw);
04859
04860                 /* Mark z-shifted dielectric */
04861                 markSphere(srاد, position,
04862                     nx, ny, nz,
04863                     hx, hy, hzed,
04864                     xmin, ymin, (zmin+0.5*hzed),
04865                     thee->epsz, epsw);
04866             }
04867         }
04868     }
04869 }
04870
04871 VPRIVATE void fillcoCoefMolDielSmooth(Vpmg *thee) {
04872     /* This function smoothes using a 9 point method based on
04873     Bruccoleri, et al. J Comput Chem 18 268-276 (1997). The nine points
04874     used are the shifted grid point and the 8 points that are 1/sqrt(2)
04875     grid spacings away. The harmonic mean of the 9 points is then used to
04876     find the overall dielectric value for the point in question. The use of
04877     this function assumes that the non-smoothed values were placed in the
04878     dielectric arrays by the fillcoCoefMolDielNoSmooth function.*/
04879
04880     Vpbe *pbe;
04881     double frac, epsw;
04882     int i, j, k, nx, ny, nz, numpts;
04883
04884     /* Mesh info */
04885     nx = thee->pmgp->nx;
04886     ny = thee->pmgp->ny;
04887     nz = thee->pmgp->nz;
04888
04889     pbe = thee->pbe;
04890     epsw = Vpbe_getSolventDiel(pbe);
04891
04892     /* Copy the existing diel arrays to work arrays */

```

```

04899     for (i=0; i<(nx*ny*nz); i++) {
04900         thee->a1cf[i] = thee->epsx[i];
04901         thee->a2cf[i] = thee->epsy[i];
04902         thee->a3cf[i] = thee->epsz[i];
04903         thee->epsx[i] = epsw;
04904         thee->epsy[i] = epsw;
04905         thee->epsz[i] = epsw;
04906     }
04907
04908     /* Smooth the dielectric values */
04909     for (i=0; i<nx; i++) {
04910         for (j=0; j<ny; j++) {
04911             for (k=0; k<nz; k++) {
04912
04913                 /* Get the 8 points that are 1/sqrt(2) grid spacings away */
04914
04915                 /* Points for the X-shifted array */
04916                 frac = 1.0/thee->a1cf[IJK(i,j,k)];
04917                 frac += 1.0/thee->a2cf[IJK(i,j,k)];
04918                 frac += 1.0/thee->a3cf[IJK(i,j,k)];
04919                 numpts = 3;
04920
04921                 if (j > 0) {
04922                     frac += 1.0/thee->a2cf[IJK(i,j-1,k)];
04923                     numpts += 1;
04924                 }
04925                 if (k > 0) {
04926                     frac += 1.0/thee->a3cf[IJK(i,j,k-1)];
04927                     numpts += 1;
04928                 }
04929                 if (i < (nx-1)){
04930                     frac += 1.0/thee->a2cf[IJK(i+1,j,k)];
04931                     frac += 1.0/thee->a3cf[IJK(i+1,j,k)];
04932                     numpts += 2;
04933                     if (j > 0) {
04934                         frac += 1.0/thee->a2cf[IJK(i+1,j-1,k)];
04935                         numpts += 1;
04936                     }
04937                     if (k > 0) {
04938                         frac += 1.0/thee->a3cf[IJK(i+1,j,k-1)];
04939                         numpts += 1;
04940                     }
04941                 }
04942                 thee->epsx[IJK(i,j,k)] = numpts/frac;
04943
04944                 /* Points for the Y-shifted array */
04945                 frac = 1.0/thee->a2cf[IJK(i,j,k)];
04946                 frac += 1.0/thee->a1cf[IJK(i,j,k)];
04947                 frac += 1.0/thee->a3cf[IJK(i,j,k)];
04948                 numpts = 3;
04949
04950                 if (i > 0) {
04951                     frac += 1.0/thee->a1cf[IJK(i-1,j,k)];
04952                     numpts += 1;
04953                 }
04954                 if (k > 0) {
04955                     frac += 1.0/thee->a3cf[IJK(i,j,k-1)];
04956                     numpts += 1;
04957                 }
04958                 if (j < (ny-1)){
04959                     frac += 1.0/thee->a1cf[IJK(i,j+1,k)];
04960                     frac += 1.0/thee->a3cf[IJK(i,j+1,k)];
04961                     numpts += 2;
04962                     if (i > 0) {
04963                         frac += 1.0/thee->a1cf[IJK(i-1,j+1,k)];
04964                         numpts += 1;
04965                     }
04966                     if (k > 0) {
04967                         frac += 1.0/thee->a3cf[IJK(i,j+1,k-1)];
04968                         numpts += 1;
04969                     }
04970                 }
04971                 thee->epsy[IJK(i,j,k)] = numpts/frac;
04972
04973                 /* Points for the Z-shifted array */
04974                 frac = 1.0/thee->a3cf[IJK(i,j,k)];
04975                 frac += 1.0/thee->a1cf[IJK(i,j,k)];
04976                 frac += 1.0/thee->a2cf[IJK(i,j,k)];
04977                 numpts = 3;
04978
04979                 if (i > 0) {

```

```

04980         frac += 1.0/thee->alcf[IJK(i-1,j,k)];
04981         numpts += 1;
04982     }
04983     if (j > 0) {
04984         frac += 1.0/thee->a2cf[IJK(i,j-1,k)];
04985         numpts += 1;
04986     }
04987     if (k < (nz-1)){
04988         frac += 1.0/thee->alcf[IJK(i,j,k+1)];
04989         frac += 1.0/thee->a2cf[IJK(i,j,k+1)];
04990         numpts += 2;
04991         if (i > 0) {
04992             frac += 1.0/thee->alcf[IJK(i-1,j,k+1)];
04993             numpts += 1;
04994         }
04995         if (j > 0) {
04996             frac += 1.0/thee->a2cf[IJK(i,j-1,k+1)];
04997             numpts += 1;
04998         }
04999     }
05000     thee->epsz[IJK(i,j,k)] = numpts/frac;
05001 }
05002 }
05003 }
05004 }
05005
05006
05007 VPRIVATE void fillCoefSpline(Vpmg *thee) {
05008     Valist *alist;
05009     Vpbe *pbe;
05010     Vatom *atom;
05011     double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr, dist2;
05012     double xlen, ylen, zlen, position[3], itot, stot, ictot, ictot2, sctot;
05013     double irad, dx, dy, dz, epsw, epsp, w2i;
05014     double hx, hy, hzed, *apos, arad, sctot2;
05015     double dx2, dy2, dz2, stot2, itot2, rtot, rtot2, splineWin, w3i;
05016     double dist, value, sm, sm2;
05017     int i, j, k, nx, ny, nz, iatom;
05018     int imin, imax, jmin, jmax, kmin, kmax;
05019
05020     VASSERT(thee != VNULL);
05021     splineWin = thee->splineWin;
05022     w2i = 1.0/(splineWin*splineWin);
05023     w3i = 1.0/(splineWin*splineWin*splineWin);
05024
05025     /* Get PBE info */
05026     pbe = thee->pbe;
05027     alist = pbe->alist;
05028     irad = Vpbe_getMaxIonRadius(pbe);
05029     ionstr = Vpbe_getBulkIonicStrength(pbe);
05030     epsw = Vpbe_getSolventDiel(pbe);
05031     epsp = Vpbe_getSoluteDiel(pbe);
05032
05033     /* Mesh info */
05034     nx = thee->pmgp->nx;
05035     ny = thee->pmgp->ny;
05036     nz = thee->pmgp->nz;
05037     hx = thee->pmgp->hx;
05038     hy = thee->pmgp->hy;
05039     hzed = thee->pmgp->hzed;
05040
05041     /* Define the total domain size */
05042     xlen = thee->pmgp->xlen;
05043     ylen = thee->pmgp->ylen;
05044     zlen = thee->pmgp->zlen;
05045
05046     /* Define the min/max dimensions */
05047     xmin = thee->pmgp->xcent - (xlen/2.0);
05048     ymin = thee->pmgp->ycent - (ylen/2.0);
05049     zmin = thee->pmgp->zcent - (zlen/2.0);
05050     xmax = thee->pmgp->xcent + (xlen/2.0);
05051     ymax = thee->pmgp->ycent + (ylen/2.0);
05052     zmax = thee->pmgp->zcent + (zlen/2.0);
05053
05054     /* This is a floating point parameter related to the non-zero nature of the
05055      * bulk ionic strength. If the ionic strength is greater than zero; this
05056      * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
05057      * Otherwise, this parameter is set to 0.0 */
05058     if (ionstr > VPMGSMALL) ionmask = 1.0;
05059     else ionmask = 0.0;
05060

```

```

05061
05062 /* Reset the kappa, epsx, epsy, and epsz arrays */
05063 for (i=0; i<(nx*ny*nz); i++) {
05064     thee->kappa[i] = 1.0;
05065     thee->epsx[i] = 1.0;
05066     thee->epsy[i] = 1.0;
05067     thee->epsz[i] = 1.0;
05068 }
05069
05070 /* Loop through the atoms and do assign the dielectric */
05071 for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
05072
05073     atom = Valist_getAtom(alist, iatom);
05074     apos = Vatom_getPosition(atom);
05075     arad = Vatom_getRadius(atom);
05076
05077     /* Make sure we're on the grid */
05078     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
05079         (apos[1]<=ymin) || (apos[1]>=ymax) || \
05080         (apos[2]<=zmin) || (apos[2]>=zmax)) {
05081         if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
05082             (thee->pmgp->bcfl != BCFL_MAP)) {
05083             Vnm_print(2, "Vpmg_fillco: Atom #d at (%4.3f, %4.3f,\n
05084 %4.3f) is off the mesh (ignoring):\n",
05085                 iatom, apos[0], apos[1], apos[2]);
05086             Vnm_print(2, "Vpmg_fillco:      xmin = %g, xmax = %g\n",
05087                 xmin, xmax);
05088             Vnm_print(2, "Vpmg_fillco:      ymin = %g, ymax = %g\n",
05089                 ymin, ymax);
05090             Vnm_print(2, "Vpmg_fillco:      zmin = %g, zmax = %g\n",
05091                 zmin, zmax);
05092         }
05093         fflush(stderr);
05094     } else if (arad > VPMGSMALL) { /* if we're on the mesh */
05095
05096         /* Convert the atom position to grid reference frame */
05097         position[0] = apos[0] - xmin;
05098         position[1] = apos[1] - ymin;
05099         position[2] = apos[2] - zmin;
05100
05101         /* MARK ION ACCESSIBILITY AND DIELECTRIC VALUES FOR LATER
05102          * ASSIGNMENT (Steps #1-3) */
05103         itot = irad + arad + splineWin;
05104         itot2 = VSQR(itot);
05105         ictot = VMAX2(0, (irad + arad - splineWin));
05106         ictot2 = VSQR(ictot);
05107         stot = arad + splineWin;
05108         stot2 = VSQR(stot);
05109         sctot = VMAX2(0, (arad - splineWin));
05110         sctot2 = VSQR(sctot);
05111
05112         /* We'll search over grid points which are in the greater of
05113          * these two radii */
05114         rtot = VMAX2(itot, stot);
05115         rtot2 = VMAX2(itot2, stot2);
05116         dx = rtot + 0.5*hx;
05117         dy = rtot + 0.5*hy;
05118         dz = rtot + 0.5*hzed;
05119         imin = VMAX2(0, (int)floor((position[0] - dx)/hx));
05120         imax = VMIN2(nx-1, (int)ceil((position[0] + dx)/hx));
05121         jmin = VMAX2(0, (int)floor((position[1] - dy)/hy));
05122         jmax = VMIN2(ny-1, (int)ceil((position[1] + dy)/hy));
05123         kmin = VMAX2(0, (int)floor((position[2] - dz)/hzed));
05124         kmax = VMIN2(nz-1, (int)ceil((position[2] + dz)/hzed));
05125         for (i=imin; i<=imax; i++) {
05126             dx2 = VSQR(position[0] - hx*i);
05127             for (j=jmin; j<=jmax; j++) {
05128                 dy2 = VSQR(position[1] - hy*j);
05129                 for (k=kmin; k<=kmax; k++) {
05130                     dz2 = VSQR(position[2] - k*hzed);
05131
05132                     /* ASSIGN CCF */
05133                     if ((thee->kappa[IJK(i,j,k)] > VPMGSMALL) {
05134                         dist2 = dz2 + dy2 + dx2;
05135                         if (dist2 >= itot2) {
05136                             ;
05137                         }
05138                     }
05139                     if (dist2 <= ictot2) {
05140                         thee->kappa[IJK(i,j,k)] = 0.0;
05141                     }
05142                 }
05143             }
05144         }
05145     }
05146 }

```

```

05142         if ((dist2 < itot2) && (dist2 > ictot2)) {
05143             dist = VSQRT(dist2);
05144             sm = dist - (arad + irad) + splineWin;
05145             sm2 = VSQR(sm);
05146             value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
05147             thee->kappa[IJK(i,j,k)] *= value;
05148         }
05149     }
05150
05151     /* ASSIGN A1CF */
05152     if (thee->epsx[IJK(i,j,k)] > VPMGSMALL) {
05153         dist2 = dz2+dy2+VSQR(position[0]-(i+0.5)*hx);
05154         if (dist2 >= stot2) {
05155             thee->epsx[IJK(i,j,k)] *= 1.0;
05156         }
05157         if (dist2 <= sctot2) {
05158             thee->epsx[IJK(i,j,k)] = 0.0;
05159         }
05160         if ((dist2 > sctot2) && (dist2 < stot2)) {
05161             dist = VSQRT(dist2);
05162             sm = dist - arad + splineWin;
05163             sm2 = VSQR(sm);
05164             value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
05165             thee->epsx[IJK(i,j,k)] *= value;
05166         }
05167     }
05168
05169     /* ASSIGN A2CF */
05170     if (thee->epsy[IJK(i,j,k)] > VPMGSMALL) {
05171         dist2 = dz2+dx2+VSQR(position[1]-(j+0.5)*hy);
05172         if (dist2 >= stot2) {
05173             thee->epsy[IJK(i,j,k)] *= 1.0;
05174         }
05175         if (dist2 <= sctot2) {
05176             thee->epsy[IJK(i,j,k)] = 0.0;
05177         }
05178         if ((dist2 > sctot2) && (dist2 < stot2)) {
05179             dist = VSQRT(dist2);
05180             sm = dist - arad + splineWin;
05181             sm2 = VSQR(sm);
05182             value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
05183             thee->epsy[IJK(i,j,k)] *= value;
05184         }
05185     }
05186
05187     /* ASSIGN A3CF */
05188     if (thee->epsz[IJK(i,j,k)] > VPMGSMALL) {
05189         dist2 = dy2+dx2+VSQR(position[2]-(k+0.5)*hzed);
05190         if (dist2 >= stot2) {
05191             thee->epsz[IJK(i,j,k)] *= 1.0;
05192         }
05193         if (dist2 <= sctot2) {
05194             thee->epsz[IJK(i,j,k)] = 0.0;
05195         }
05196         if ((dist2 > sctot2) && (dist2 < stot2)) {
05197             dist = VSQRT(dist2);
05198             sm = dist - arad + splineWin;
05199             sm2 = VSQR(sm);
05200             value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
05201             thee->epsz[IJK(i,j,k)] *= value;
05202         }
05203     }
05204
05205     } /* k loop */
05206 } /* j loop */
05207 } /* i loop */
05208 } /* endif (on the mesh) */
05209 } /* endfor (over all atoms) */
05210
05211 Vnm_print(0, "Vpmg_fillco: filling coefficient arrays\n");
05212 /* Interpret markings and fill the coefficient arrays */
05213 for (k=0; k<nz; k++) {
05214     for (j=0; j<ny; j++) {
05215         for (i=0; i<nx; i++) {
05216
05217             thee->kappa[IJK(i,j,k)] = ionmask*thee->kappa[IJK(i,j,k)];
05218             thee->epsx[IJK(i,j,k)] = (epsw-epsp)*thee->epsx[IJK(i,j,k)]
05219                 + epsp;
05220             thee->epsy[IJK(i,j,k)] = (epsw-epsp)*thee->epsy[IJK(i,j,k)]
05221                 + epsp;
05222

```

```

05223         thee->epsz[IJK(i,j,k)] = (epsw-epsp)*thee->epsz[IJK(i,j,k)]
05224         + epsp;
05225
05226     } /* i loop */
05227 } /* j loop */
05228 } /* k loop */
05229
05230 }
05231
05232 VPRIVATE void fillcoCoef(Vpmg *thee) {
05233
05234     VASSERT(thee != VNULL);
05235
05236     if (thee->useDielXMap || thee->useDielYMap ||
05237         thee->useDielZMap || thee->useKappaMap) {
05238         fillcoCoefMap(thee);
05239         return;
05240     }
05241
05242     switch(thee->surfMeth) {
05243     case VSM_MOL:
05244         Vnm_print(0, "fillcoCoef: Calling fillcoCoefMol...\n");
05245         fillcoCoefMol(thee);
05246         break;
05247     case VSM_MOLSMOOTH:
05248         Vnm_print(0, "fillcoCoef: Calling fillcoCoefMol...\n");
05249         fillcoCoefMol(thee);
05250         break;
05251     case VSM_SPLINE:
05252         Vnm_print(0, "fillcoCoef: Calling fillcoCoefSpline...\n");
05253         fillcoCoefSpline(thee);
05254         break;
05255     case VSM_SPLINE3:
05256         Vnm_print(0, "fillcoCoef: Calling fillcoCoefSpline3...\n");
05257         fillcoCoefSpline3(thee);
05258         break;
05259     case VSM_SPLINE4:
05260         Vnm_print(0, "fillcoCoef: Calling fillcoCoefSpline4...\n");
05261         fillcoCoefSpline4(thee);
05262         break;
05263     default:
05264         Vnm_print(2, "fillcoCoef: Invalid surfMeth (%d)!\n",
05265             thee->surfMeth);
05266         VASSERT(0);
05267         break;
05268     }
05269 }
05270
05271
05272 VPRIVATE Vrc_Codes fillcoCharge(Vpmg *thee) {
05273
05274     Vrc_Codes rc;
05275
05276     VASSERT(thee != VNULL);
05277
05278     if (thee->useChargeMap) {
05279         return fillcoChargeMap(thee);
05280     }
05281
05282     switch(thee->chargeMeth) {
05283     case VCM_TRIL:
05284         Vnm_print(0, "fillcoCharge: Calling fillcoChargeSpline1...\n");
05285         fillcoChargeSpline1(thee);
05286         break;
05287     case VCM_BSPL2:
05288         Vnm_print(0, "fillcoCharge: Calling fillcoChargeSpline2...\n");
05289         fillcoChargeSpline2(thee);
05290         break;
05291     case VCM_BSPL4:
05292         switch (thee->chargeSrc) {
05293         case VCM_CHARGE:
05294             Vnm_print(0, "fillcoCharge: Calling fillcoPermanentMultipole...\n");
05295             fillcoPermanentMultipole(thee);
05296             break;
05297 #if defined(WITH_TINKER)
05298         case VCM_PERMANENT:
05299             Vnm_print(0, "fillcoCharge: Calling fillcoPermanentMultipole...\n");
05300             fillcoPermanentMultipole(thee);
05301             break;
05302         case VCM_INDUCED:
05303             Vnm_print(0, "fillcoCharge: Calling fillcoInducedDipole...\n");

```

```

05304         fillcoInducedDipole(thee);
05305         break;
05306     case VCM_NLINDUCED:
05307         Vnm_print(0, "fillcoCharge: Calling fillcoNLInducedDipole...\n");
05308         fillcoNLInducedDipole(thee);
05309         break;
05310 #endif /* if defined(WITH_TINKER) */
05311     default:
05312         Vnm_print(2, "fillcoCharge: Invalid chargeSource (%d)!\n",
05313             thee->chargeSrc);
05314         return VRC_FAILURE;
05315         break;
05316     }
05317     break;
05318     default:
05319         Vnm_print(2, "fillcoCharge: Invalid chargeMeth (%d)!\n",
05320             thee->chargeMeth);
05321         return VRC_FAILURE;
05322         break;
05323     }
05324 }
05325 return VRC_SUCCESS;
05326 }
05327
05328 VPRIVATE Vrc_Codes fillcoChargeMap(Vpmg *thee) {
05329
05330     Vpbe *pbe;
05331     double position[3], charge, zmagic, hx, hy, hzed;
05332     int i, j, k, nx, ny, nz, rc;
05333
05334     VASSERT(thee != VNULL);
05335
05336     /* Get PBE info */
05337     pbe = thee->pbe;
05338     zmagic = Vpbe_getZmagic(pbe);
05339
05340     /* Mesh info */
05341     nx = thee->pmgp->nx;
05342     ny = thee->pmgp->ny;
05343     nz = thee->pmgp->nz;
05344     hx = thee->pmgp->hx;
05345     hy = thee->pmgp->hy;
05346     hzed = thee->pmgp->hzed;
05347
05348     /* Reset the charge array */
05349     for (i=0; i<(nx*ny*nz); i++) thee->charge[i] = 0.0;
05350
05351     /* Fill in the source term (atomic charges) */
05352     Vnm_print(0, "Vpmg_fillco: filling in source term.\n");
05353     for (k=0; k<nz; k++) {
05354         for (j=0; j<ny; j++) {
05355             for (i=0; i<nx; i++) {
05356                 position[0] = thee->xf[i];
05357                 position[1] = thee->yf[j];
05358                 position[2] = thee->zf[k];
05359                 rc = Vgrid_value(thee->chargeMap, position, &charge);
05360                 if (!rc) {
05361                     Vnm_print(2, "fillcoChargeMap: Error -- fell off of charge map at (%g, %g, %g)!\n",
05362                         position[0], position[1], position[2]);
05363                     return VRC_FAILURE;
05364                 }
05365                 /* Scale the charge to internal units */
05366                 charge = charge*zmagic;
05367                 thee->charge[IJK(i,j,k)] = charge;
05368             }
05369         }
05370     }
05371 }
05372
05373 return VRC_SUCCESS;
05374 }
05375
05376 VPRIVATE void fillcoChargeSpline1(Vpmg *thee) {
05377
05378     Valist *alist;
05379     Vpbe *pbe;
05380     Vatom *atom;
05381     double xmin, xmax, ymin, ymax, zmin, zmax;
05382     double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;
05383     double charge, dx, dy, dz, zmagic, hx, hy, hzed, *apos;
05384     int i, nx, ny, nz, iatom, ihi, ilo, jhi, jlo, khi, klo;

```



```

05385
05386
05387     VASSERT(thee != VNULL);
05388
05389     /* Get PBE info */
05390     pbe = thee->pbe;
05391     alist = pbe->alist;
05392     zmagic = Vpbe_getZmagic(pbe);
05393
05394     /* Mesh info */
05395     nx = thee->pmgp->nx;
05396     ny = thee->pmgp->ny;
05397     nz = thee->pmgp->nz;
05398     hx = thee->pmgp->hx;
05399     hy = thee->pmgp->hy;
05400     hzed = thee->pmgp->hzed;
05401
05402     /* Define the total domain size */
05403     xlen = thee->pmgp->xlen;
05404     ylen = thee->pmgp->ylen;
05405     zlen = thee->pmgp->zlen;
05406
05407     /* Define the min/max dimensions */
05408     xmin = thee->pmgp->xcent - (xlen/2.0);
05409     ymin = thee->pmgp->ycent - (ylen/2.0);
05410     zmin = thee->pmgp->zcent - (zlen/2.0);
05411     xmax = thee->pmgp->xcent + (xlen/2.0);
05412     ymax = thee->pmgp->ycent + (ylen/2.0);
05413     zmax = thee->pmgp->zcent + (zlen/2.0);
05414
05415     /* Reset the charge array */
05416     for (i=0; i<(nx*ny*nz); i++) thee->charge[i] = 0.0;
05417
05418     /* Fill in the source term (atomic charges) */
05419     Vnm_print(0, "Vpmg_fillco: filling in source term.\n");
05420     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
05421
05422         atom = Valist_getAtom(alist, iatom);
05423         apos = Vatom_getPosition(atom);
05424         charge = Vatom_getCharge(atom);
05425
05426         /* Make sure we're on the grid */
05427         if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
05428             (apos[1]<=ymin) || (apos[1]>=ymax) || \
05429             (apos[2]<=zmin) || (apos[2]>=zmax)) {
05430             if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
05431                 (thee->pmgp->bcfl != BCFL_MAP)) {
05432                 Vnm_print(2, "Vpmg_fillco: Atom #d at (%4.3f, %4.3f, \
05433 %4.3f) is off the mesh (ignoring):\n",
05434                     iatom, apos[0], apos[1], apos[2]);
05435                 Vnm_print(2, "Vpmg_fillco:      xmin = %g, xmax = %g\n",
05436                     xmin, xmax);
05437                 Vnm_print(2, "Vpmg_fillco:      ymin = %g, ymax = %g\n",
05438                     ymin, ymax);
05439                 Vnm_print(2, "Vpmg_fillco:      zmin = %g, zmax = %g\n",
05440                     zmin, zmax);
05441             }
05442             fflush(stderr);
05443         } else {
05444
05445             /* Convert the atom position to grid reference frame */
05446             position[0] = apos[0] - xmin;
05447             position[1] = apos[1] - ymin;
05448             position[2] = apos[2] - zmin;
05449
05450             /* Scale the charge to be a delta function */
05451             charge = charge*zmagic/(hx*hy*hzed);
05452
05453             /* Figure out which vertices we're next to */
05454             ifloat = position[0]/hx;
05455             jfloat = position[1]/hy;
05456             kfloat = position[2]/hzed;
05457
05458             ihi = (int)ceil(ifloat);
05459             ilo = (int)floor(ifloat);
05460             jhi = (int)ceil(jfloat);
05461             jlo = (int)floor(jfloat);
05462             khi = (int)ceil(kfloat);
05463             klo = (int)floor(kfloat);
05464
05465             /* Now assign fractions of the charge to the nearby verts */

```

```

05466         dx = ifloat - (double)(ilo);
05467         dy = jfloat - (double)(jlo);
05468         dz = kfloat - (double)(klo);
05469         thee->charge[IJK(ihi,jhi,khi)] += (dx*dy*dz*charge);
05470         thee->charge[IJK(ihi,jlo,khi)] += (dx*(1.0-dy)*dz*charge);
05471         thee->charge[IJK(ihi,jhi,klo)] += (dx*dy*(1.0-dz)*charge);
05472         thee->charge[IJK(ihi,jlo,klo)] += (dx*(1.0-dy)*(1.0-dz)*charge);
05473         thee->charge[IJK(ilo,jhi,khi)] += ((1.0-dx)*dy*dz *charge);
05474         thee->charge[IJK(ilo,jlo,khi)] += ((1.0-dx)*(1.0-dy)*dz *charge);
05475         thee->charge[IJK(ilo,jhi,klo)] += ((1.0-dx)*dy*(1.0-dz)*charge);
05476         thee->charge[IJK(ilo,jlo,klo)] += ((1.0-dx)*(1.0-dy)*(1.0-dz)*charge);
05477     } /* endif (on the mesh) */
05478 } /* endfor (each atom) */
05479 }
05480
05481 VPRIVATE double bspline2(double x) {
05482
05483     double m2m, m2, m3;
05484
05485     if ((x >= 0.0) && (x <= 2.0)) m2m = 1.0 - VABS(x - 1.0);
05486     else m2m = 0.0;
05487     if ((x >= 1.0) && (x <= 3.0)) m2 = 1.0 - VABS(x - 2.0);
05488     else m2 = 0.0;
05489
05490     if ((x >= 0.0) && (x <= 3.0)) m3 = 0.5*x*m2m + 0.5*(3.0-x)*m2;
05491     else m3 = 0.0;
05492
05493     return m3;
05494 }
05495
05496 VPRIVATE double dbspline2(double x) {
05497
05498     double m2m, m2, dm3;
05499
05500     if ((x >= 0.0) && (x <= 2.0)) m2m = 1.0 - VABS(x - 1.0);
05501     else m2m = 0.0;
05502     if ((x >= 1.0) && (x <= 3.0)) m2 = 1.0 - VABS(x - 2.0);
05503     else m2 = 0.0;
05504
05505     dm3 = m2m - m2;
05506
05507     return dm3;
05508 }
05509
05510 }
05511
05512 VPRIVATE void fillcoChargeSpline2(Vpmg *thee) {
05513
05514     Valist *alist;
05515     Vpbe *pbe;
05516     Vatom *atom;
05517
05518     double xmin, xmax, ymin, ymax, zmin, zmax, zmagic;
05519     double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;
05520     double charge, hx, hy, hzed, *apos, mx, my, mz;
05521     int i, ii, jj, kk, nx, ny, nz, iatom;
05522     int im2, im1, ip1, ip2, jm2, jm1, jpl, jp2, km2, km1, kp1, kp2;
05523
05524     VASSERT(thee != VNULL);
05525
05526     /* Get PBE info */
05527     pbe = thee->pbe;
05528     alist = pbe->alist;
05529     zmagic = Vpbe_getZmagic(pbe);
05530
05531     /* Mesh info */
05532     nx = thee->pmgp->nx;
05533     ny = thee->pmgp->ny;
05534     nz = thee->pmgp->nz;
05535     hx = thee->pmgp->hx;
05536     hy = thee->pmgp->hy;
05537     hzed = thee->pmgp->hzed;
05538
05539     /* Define the total domain size */
05540     xlen = thee->pmgp->xlen;
05541     ylen = thee->pmgp->ylen;
05542     zlen = thee->pmgp->zlen;
05543
05544     /* Define the min/max dimensions */
05545     xmin = thee->pmgp->xcen - (xlen/2.0);

```

```

05547     ymin = thee->pmgp->ycent - (ylen/2.0);
05548     zmin = thee->pmgp->zcent - (zlen/2.0);
05549     xmax = thee->pmgp->xcent + (xlen/2.0);
05550     ymax = thee->pmgp->ycent + (ylen/2.0);
05551     zmax = thee->pmgp->zcent + (zlen/2.0);
05552
05553     /* Reset the charge array */
05554     for (i=0; i<(nx*ny*nz); i++) thee->charge[i] = 0.0;
05555
05556     /* Fill in the source term (atomic charges) */
05557     Vnm_print(0, "Vpmg_fillco: filling in source term.\n");
05558     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
05559
05560         atom = Valist_getAtom(alist, iatom);
05561         apos = Vatom_getPosition(atom);
05562         charge = Vatom_getCharge(atom);
05563
05564         /* Make sure we're on the grid */
05565         if ((apos[0]<=(xmin-hx)) || (apos[0]>=(xmax+hx)) || \
05566             (apos[1]<=(ymin-hy)) || (apos[1]>=(ymax+hy)) || \
05567             (apos[2]<=(zmin-hzed)) || (apos[2]>=(zmax+hzed))) {
05568             if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
05569                 (thee->pmgp->bcfl != BCFL_MAP)) {
05570                 Vnm_print(2, "Vpmg_fillco: Atom #d at (%4.3f, %4.3f, \
05571 %4.3f) is off the mesh (for cubic splines!!) (ignoring this atom):\n",
05572                     iatom, apos[0], apos[1], apos[2]);
05573                 Vnm_print(2, "Vpmg_fillco:      xmin = %g, xmax = %g\n",
05574                     xmin, xmax);
05575                 Vnm_print(2, "Vpmg_fillco:      ymin = %g, ymax = %g\n",
05576                     ymin, ymax);
05577                 Vnm_print(2, "Vpmg_fillco:      zmin = %g, zmax = %g\n",
05578                     zmin, zmax);
05579             }
05580             fflush(stderr);
05581         } else {
05582
05583             /* Convert the atom position to grid reference frame */
05584             position[0] = apos[0] - xmin;
05585             position[1] = apos[1] - ymin;
05586             position[2] = apos[2] - zmin;
05587
05588             /* Scale the charge to be a delta function */
05589             charge = charge*zmagic/(hx*hy*hzed);
05590
05591             /* Figure out which vertices we're next to */
05592             ifloat = position[0]/hx;
05593             jfloat = position[1]/hy;
05594             kfloat = position[2]/hzed;
05595
05596             ip1 = (int)ceil(ifloat);
05597             ip2 = ip1 + 1;
05598             im1 = (int)floor(ifloat);
05599             im2 = im1 - 1;
05600             jp1 = (int)ceil(jfloat);
05601             jp2 = jp1 + 1;
05602             jm1 = (int)floor(jfloat);
05603             jm2 = jm1 - 1;
05604             kp1 = (int)ceil(kfloat);
05605             kp2 = kp1 + 1;
05606             km1 = (int)floor(kfloat);
05607             km2 = km1 - 1;
05608
05609             /* This step shouldn't be necessary, but it saves nasty debugging
05610              * later on if something goes wrong */
05611             ip2 = VMIN2(ip2,nx-1);
05612             ip1 = VMIN2(ip1,nx-1);
05613             im1 = VMAX2(im1,0);
05614             im2 = VMAX2(im2,0);
05615             jp2 = VMIN2(jp2,ny-1);
05616             jp1 = VMIN2(jp1,ny-1);
05617             jm1 = VMAX2(jm1,0);
05618             jm2 = VMAX2(jm2,0);
05619             kp2 = VMIN2(kp2,nz-1);
05620             kp1 = VMIN2(kp1,nz-1);
05621             km1 = VMAX2(km1,0);
05622             km2 = VMAX2(km2,0);
05623
05624             /* Now assign fractions of the charge to the nearby verts */
05625             for (ii=im2; ii<=ip2; ii++) {
05626                 mx = bspline2(VFCHI(ii,ifloat));
05627                 for (jj=jm2; jj<=jp2; jj++) {

```

```

05628         my = bspline2(VFCHI(jj,jfloat));
05629         for (kk=km2; kk<=kp2; kk++) {
05630             mz = bspline2(VFCHI(kk,kfloat));
05631             thee->charge[IJK(ii,jj,kk)] += (charge*mx*my*mz);
05632         }
05633     }
05634 }
05635
05636     } /* endif (on the mesh) */
05637 } /* endfor (each atom) */
05638 }
05639
05640 VPUBLIC int Vpmg_fillco(Vpmg *thee,
05641     Vsurf_Meth surfMeth,
05642     double splineWin,
05643     Vchrg_Meth chargeMeth,
05644     int useDielXMap,
05645     Vgrid *dielXMap,
05646     int useDielYMap,
05647     Vgrid *dielYMap,
05648     int useDielZMap,
05649     Vgrid *dielZMap,
05650     int useKappaMap,
05651     Vgrid *kappaMap,
05652     int usePotMap,
05653     Vgrid *potMap,
05654     int useChargeMap,
05655     Vgrid *chargeMap
05656 ) {
05657
05658     Vpbe *pbe;
05659     double xmin,
05660         xmax,
05661         ymin,
05662         ymax,
05663         zmin,
05664         zmax,
05665         xlen,
05666         ylen,
05667         zlen,
05668         hx,
05669         hy,
05670         hzed,
05671         epsw,
05672         epsp,
05673         ionstr;
05674     int i,
05675         nx,
05676         ny,
05677         nz,
05678         islap;
05679     Vrc_Codes rc;
05680
05681     if (thee == VNULL) {
05682         Vnm_print(2, "Vpmg_fillco: got NULL thee!\n");
05683         return 0;
05684     }
05685
05686     thee->surfMeth = surfMeth;
05687     thee->splineWin = splineWin;
05688     thee->chargeMeth = chargeMeth;
05689     thee->useDielXMap = useDielXMap;
05690     if (thee->useDielXMap) thee->dielXMap = dielXMap;
05691     thee->useDielYMap = useDielYMap;
05692     if (thee->useDielYMap) thee->dielYMap = dielYMap;
05693     thee->useDielZMap = useDielZMap;
05694     if (thee->useDielZMap) thee->dielZMap = dielZMap;
05695     thee->useKappaMap = useKappaMap;
05696     if (thee->useKappaMap) thee->kappaMap = kappaMap;
05697     thee->usePotMap = usePotMap;
05698     if (thee->usePotMap) thee->potMap = potMap;
05699     thee->useChargeMap = useChargeMap;
05700     if (thee->useChargeMap) thee->chargeMap = chargeMap;
05701
05702     /* Get PBE info */
05703     pbe = thee->pbe;
05704     ionstr = Vpbe_getBulkIonicStrength(pbe);
05705     epsw = Vpbe_getSolventDiel(pbe);
05706     epsp = Vpbe_getSoluteDiel(pbe);
05707
05708     /* Mesh info */

```

```

05709     nx = thee->pmgp->nx;
05710     ny = thee->pmgp->ny;
05711     nz = thee->pmgp->nz;
05712     hx = thee->pmgp->hx;
05713     hy = thee->pmgp->hy;
05714     hzed = thee->pmgp->hzed;
05715
05716     /* Define the total domain size */
05717     xlen = thee->pmgp->xlen;
05718     ylen = thee->pmgp->ylen;
05719     zlen = thee->pmgp->zlen;
05720
05721     /* Define the min/max dimensions */
05722     xmin = thee->pmgp->xcent - (xlen/2.0);
05723     thee->pmgp->xmin = xmin;
05724     ymin = thee->pmgp->ycent - (ylen/2.0);
05725     thee->pmgp->ymin = ymin;
05726     zmin = thee->pmgp->zcent - (zlen/2.0);
05727     thee->pmgp->zmin = zmin;
05728     xmax = thee->pmgp->xcent + (xlen/2.0);
05729     thee->pmgp->xmax = xmax;
05730     ymax = thee->pmgp->ycent + (ylen/2.0);
05731     thee->pmgp->ymax = ymax;
05732     zmax = thee->pmgp->zcent + (zlen/2.0);
05733     thee->pmgp->zmax = zmax;
05734     thee->rparm[2] = xmin;
05735     thee->rparm[3] = xmax;
05736     thee->rparm[4] = ymin;
05737     thee->rparm[5] = ymax;
05738     thee->rparm[6] = zmin;
05739     thee->rparm[7] = zmax;
05740
05741     /* This is a flag that gets set if the operator is a simple Laplacian;
05742      * i.e., in the case of a homogenous dielectric and zero ionic strength
05743      * The operator cannot be a simple Laplacian if maps are read in. */
05744     if (thee->useDielsXMap || thee->useDielsYMap || thee->
useDielsZMap ||
05745         thee->useKappaMap || thee->usePotMap){
05746         islap = 0;
05747     } else if ( (ionstr < VPMGSMALL) && (VABS(epsps-epsw) < VPMGSMALL) ){
05748         islap = 1;
05749     } else {
05750         islap = 0;
05751     }
05752
05753     /* Fill the mesh point coordinate arrays */
05754     for (i=0; i<nx; i++) thee->xf[i] = xmin + i*hx;
05755     for (i=0; i<ny; i++) thee->yf[i] = ymin + i*hy;
05756     for (i=0; i<nz; i++) thee->zf[i] = zmin + i*hzed;
05757
05758     /* Reset the tcf array */
05759     for (i=0; i<(nx*ny*nz); i++) thee->tcf[i] = 0.0;
05760
05761     /* Fill in the source term (atomic charges) */
05762     Vnm_print(0, "Vpmg_fillco: filling in source term.\n");
05763     rc = fillcoCharge(thee);
05764     switch(rc) {
05765     case VRC_SUCCESS:
05766         break;
05767     case VRC_WARNING:
05768         Vnm_print(2, "Vpmg_fillco: non-fatal errors while filling charge map!\n");
05769         break;
05770     case VRC_FAILURE:
05771         Vnm_print(2, "Vpmg_fillco: fatal errors while filling charge map!\n");
05772         return 0;
05773         break;
05774     }
05775
05776     /* THE FOLLOWING NEEDS TO BE DONE IF WE'RE NOT USING A SIMPLE LAPLACIAN
05777      * OPERATOR */
05778     if (!islap) {
05779         Vnm_print(0, "Vpmg_fillco: marking ion and solvent accessibility.\n");
05780         fillcoCoef(thee);
05781         Vnm_print(0, "Vpmg_fillco: done filling coefficient arrays\n");
05782     } else { /* else (!islap) ==> It's a Laplacian operator! */
05783
05784         for (i=0; i<(nx*ny*nz); i++) {
05785             thee->kappa[i] = 0.0;
05786             thee->epsx[i] = epsps;
05787             thee->epsy[i] = epsps;

```

```

05789         thee->epsz[i] = epsp;
05790     }
05791
05792     } /* endif (!islap) */
05793
05794     /* Fill the boundary arrays (except when focusing, bcfl = 4) */
05795     if (thee->pmgp->bcfl != BCFL_FOCUS) {
05796         Vnm_print(0, "Vpmg_fillco: filling boundary arrays\n");
05797         bcCalc(thee);
05798         Vnm_print(0, "Vpmg_fillco: done filling boundary arrays\n");
05799     }
05800
05801     thee->filled = 1;
05802
05803     return 1;
05804 }
05805
05806
05807 VPUBLIC int Vpmg_force(Vpmg *thee, double *force, int atomID,
05808     Vsurf_Meth srfrm, Vchrg_Meth chgm) {
05809
05810     int rc = 1;
05811     double qfF[3]; /* Charge-field force */
05812     double dbF[3]; /* Dielectric boundary force */
05813     double ibF[3]; /* Ion boundary force */
05814     double npF[3]; /* Non-polar boundary force */
05815
05816     VASSERT(thee != VNULL);
05817
05818     rc = rc && Vpmg_dbForce(thee, qfF, atomID, srfrm);
05819     rc = rc && Vpmg_ibForce(thee, dbF, atomID, srfrm);
05820     rc = rc && Vpmg_qfForce(thee, ibF, atomID, chgm);
05821
05822     force[0] = qfF[0] + dbF[0] + ibF[0];
05823     force[1] = qfF[1] + dbF[1] + ibF[1];
05824     force[2] = qfF[2] + dbF[2] + ibF[2];
05825
05826     return rc;
05827
05828 }
05829
05830 VPUBLIC int Vpmg_ibForce(Vpmg *thee, double *force, int atomID,
05831     Vsurf_Meth srfrm) {
05832
05833     Valist *alist;
05834     Vacc *acc;
05835     Vpbe *pbe;
05836     Vatom *atom;
05837
05838     double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
05839     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;
05840     double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
05841     double izmagic;
05842     int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
05843
05844     /* For nonlinear forces */
05845     int ichop, nchop, nion, m;
05846     double ionConc[MAXION], ionRadii[MAXION], ionQ[MAXION], ionstr;
05847
05848     VASSERT(thee != VNULL);
05849
05850     acc = thee->pbe->acc;
05851     atom = Valist_getAtom(thee->pbe->alist, atomID);
05852     apos = Vatom_getPosition(atom);
05853     arad = Vatom_getRadius(atom);
05854
05855     /* Reset force */
05856     force[0] = 0.0;
05857     force[1] = 0.0;
05858     force[2] = 0.0;
05859
05860     /* Check surface definition */
05861     if ((srfrm != VSM_SPLINE) && (srfrm != VSM_SPLINE3) && (srfrm !=
05862         VSM_SPLINE4)) {
05862         Vnm_print(2, "Vpmg_ibForce: Forces *must* be calculated with \
05863 spline-based surfaces!\n");
05864         Vnm_print(2, "Vpmg_ibForce: Skipping ionic boundary force \
05865 calculation!\n");
05866         return 0;
05867     }
05868

```

```

05869      /* If we aren't in the current position, then we're done */
05870      if (atom->partID == 0) return 1;
05871
05872      /* Get PBE info */
05873      pbe = thee->pbe;
05874      acc = pbe->acc;
05875      alist = pbe->alist;
05876      irad = Vpbe_getMaxIonRadius(pbe);
05877      zkappa2 = Vpbe_getZkappa2(pbe);
05878      izmagic = 1.0/Vpbe_getZmagic(pbe);
05879
05880      ionstr = Vpbe_getBulkIonicStrength(pbe);
05881      Vpbe_getIons(pbe, &nion, ionConc, ionRadii, ionQ);
05882
05883      /* Mesh info */
05884      nx = thee->pmgp->nx;
05885      ny = thee->pmgp->ny;
05886      nz = thee->pmgp->nz;
05887      hx = thee->pmgp->hx;
05888      hy = thee->pmgp->hy;
05889      hzed = thee->pmgp->hzed;
05890      xlen = thee->pmgp->xlen;
05891      ylen = thee->pmgp->ylen;
05892      zlen = thee->pmgp->zlen;
05893      xmin = thee->pmgp->xmin;
05894      ymin = thee->pmgp->ymin;
05895      zmin = thee->pmgp->zmin;
05896      xmax = thee->pmgp->xmax;
05897      ymax = thee->pmgp->ymax;
05898      zmax = thee->pmgp->zmax;
05899
05900      /* Sanity check: there is no force if there is zero ionic strength */
05901      if (zkappa2 < VPMGSMALL) {
05902 #ifndef VAPBSQUIET
05903         Vnm_print(2, "Vpmg_ibForce: No force for zero ionic strength!\n");
05904 #endif
05905         return 1;
05906     }
05907
05908      /* Make sure we're on the grid */
05909      if ((apos[0] <= xmin) || (apos[0] >= xmax) || \
05910          (apos[1] <= ymin) || (apos[1] >= ymax) || \
05911          (apos[2] <= zmin) || (apos[2] >= zmax)) {
05912         if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
05913             (thee->pmgp->bcfl != BCFL_MAP)) {
05914             Vnm_print(2, "Vpmg_ibForce: Atom #%d at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):\n",
05915                 atom, apos[0], apos[1], apos[2]);
05916             Vnm_print(2, "Vpmg_ibForce:      xmin = %g, xmax = %g\n",
05917                 xmin, xmax);
05918             Vnm_print(2, "Vpmg_ibForce:      ymin = %g, ymax = %g\n",
05919                 ymin, ymax);
05920             Vnm_print(2, "Vpmg_ibForce:      zmin = %g, zmax = %g\n",
05921                 zmin, zmax);
05922         }
05923         fflush(stderr);
05924     } else {
05925
05926         /* Convert the atom position to grid reference frame */
05927         position[0] = apos[0] - xmin;
05928         position[1] = apos[1] - ymin;
05929         position[2] = apos[2] - zmin;
05930
05931         /* Integrate over points within this atom's (inflated) radius */
05932         rtot = (irad + arad + thee->splineWin);
05933         rtot2 = VSQR(rtot);
05934         dx = rtot + 0.5*hx;
05935         imin = VMAX2(0, (int)ceil((position[0] - dx)/hx));
05936         imax = VMIN2(nx-1, (int)floor((position[0] + dx)/hx));
05937         for (i=imin; i<=imax; i++) {
05938             dx2 = VSQR(position[0] - hx*i);
05939             if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
05940             else dy = 0.5*hy;
05941             jmin = VMAX2(0, (int)ceil((position[1] - dy)/hy));
05942             jmax = VMIN2(ny-1, (int)floor((position[1] + dy)/hy));
05943             for (j=jmin; j<=jmax; j++) {
05944                 dy2 = VSQR(position[1] - hy*j);
05945                 if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
05946                 else dz = 0.5*hzed;
05947                 kmin = VMAX2(0, (int)ceil((position[2] - dz)/hzed));
05948                 kmax = VMIN2(nz-1, (int)floor((position[2] + dz)/hzed));
05949                 for (k=kmin; k<=kmax; k++) {

```

```

05950         dz2 = VSQR(k*hzed - position[2]);
05951         /* See if grid point is inside ivdw radius and set kappa
05952          * accordingly (do spline assignment here) */
05953         if ((dz2 + dy2 + dx2) <= rtot2) {
05954             gpos[0] = i*hx + xmin;
05955             gpos[1] = j*hy + ymin;
05956             gpos[2] = k*hzed + zmin;
05957
05958             /* Select the correct function based on the surface definition
05959              * (now including the 7th order polynomial) */
05960             Vpmg_splineSelect(srfm,acc, gpos,thee->
splineWin, irad, atom, tgrad);
05961
05962             if (thee->pmgp->nonlin) {
05963                 /* Nonlinear forces */
05964                 fmag = 0.0;
05965                 nchop = 0;
05966                 for (m=0; m<nion; m++) {
05967                     fmag += (thee->kappa[IJK(i,j,k)])*ionConc[m]*(
Vcap_exp(-ionQ[m]*thee->u[IJK(i,j,k)], &ichop)-1.0)/ionstr;
05968                     nchop += ichop;
05969                 }
05970                 /* if (nchop > 0) Vnm_print(2, "Vpmg_ibForece: Chopped EXP %d
times!\n", nchop);*/
05971                 force[0] += (zkappa2*fmag*tgrad[0]);
05972                 force[1] += (zkappa2*fmag*tgrad[1]);
05973                 force[2] += (zkappa2*fmag*tgrad[2]);
05974             } else {
05975                 /* Use of bulk factor (zkappa2) OK here becuase
05976                  * LPBE force approximation */
05977                 /* NAB -- did we forget a kappa factor here??? */
05978                 fmag = VSQR(thee->u[IJK(i,j,k)])*(thee->kappa[IJK(i,j,k)]);
05979                 force[0] += (zkappa2*fmag*tgrad[0]);
05980                 force[1] += (zkappa2*fmag*tgrad[1]);
05981                 force[2] += (zkappa2*fmag*tgrad[2]);
05982             }
05983         }
05984     } /* k loop */
05985 } /* j loop */
05986 } /* i loop */
05987 }
05988 force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
05989 force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
05990 force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
05991
05992 return 1;
05993 }
05994
05995 VPUBLIC int Vpmg_dbForce(Vpmg *thee, double *dbForce, int atomID,
05996 Vsrf_Meth srfm) {
05997
05998     Vacc *acc;
05999     Vpbe *pbe;
06000     Vatom *atom;
06001
06002     double *apos, position[3], arad, srad, hx, hy, hzed, izmagic, deps, depsi;
06003     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2, epsp;
06004     double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
06005     double *u, Hxijk, Hyijk, Hzijk, Hximljk, Hyijmlk, Hzijkml;
06006     double dHxijk[3], dHyijk[3], dHzijk[3], dHximljk[3], dHyijmlk[3];
06007     double dHzijkml[3];
06008     int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
06009
06010     VASSERT(thee != VNULL);
06011     if (!thee->filled) {
06012         Vnm_print(2, "Vpmg_dbForce: Need to callVpmg_fillco!\n");
06013         return 0;
06014     }
06015
06016     acc = thee->pbe->acc;
06017     atom = Valist_getAtom(thee->pbe->alist, atomID);
06018     apos = Vatom_getPosition(atom);
06019     arad = Vatom_getRadius(atom);
06020     srad = Vpbe_getSolventRadius(thee->pbe);
06021
06022     /* Reset force */
06023     dbForce[0] = 0.0;
06024     dbForce[1] = 0.0;
06025     dbForce[2] = 0.0;
06026
06027     /* Check surface definition */

```



```

06028     if ((srfm != VSM_SPLINE) && (srfm!=VSM_SPLINE3) && (srfm!=
VSM_SPLINE4)) {
06029         Vnm_print(2, "Vpmg_dbForce: Forces *must* be calculated with \
06030 spline-based surfaces!\n");
06031         Vnm_print(2, "Vpmg_dbForce: Skipping dielectric/apolar boundary \
06032 force calculation!\n");
06033         return 0;
06034     }
06035
06036
06037     /* If we aren't in the current position, then we're done */
06038     if (atom->partID == 0) return 1;
06039
06040     /* Get PBE info */
06041     pbe = thee->pbe;
06042     acc = pbe->acc;
06043     epsp = Vpbe_getSoluteDiel(pbe);
06044     epsw = Vpbe_getSolventDiel(pbe);
06045     kT = Vpbe_getTemperature(pbe) * (1e-3) * Vunit_Na *
Vunit_kb;
06046     izmagic = 1.0/Vpbe_getZmagic(pbe);
06047
06048     /* Mesh info */
06049     nx = thee->pmgp->nx;
06050     ny = thee->pmgp->ny;
06051     nz = thee->pmgp->nz;
06052     hx = thee->pmgp->hx;
06053     hy = thee->pmgp->hy;
06054     hzed = thee->pmgp->hzd;
06055     xlen = thee->pmgp->xlen;
06056     ylen = thee->pmgp->ylen;
06057     zlen = thee->pmgp->zlen;
06058     xmin = thee->pmgp->xmin;
06059     ymin = thee->pmgp->ymin;
06060     zmin = thee->pmgp->zmin;
06061     xmax = thee->pmgp->xmax;
06062     ymax = thee->pmgp->ymax;
06063     zmax = thee->pmgp->zmax;
06064     u = thee->u;
06065
06066     /* Sanity check: there is no force if there is zero ionic strength */
06067     if (VABS(epsp-epsw) < VPMGSMALL) {
06068         Vnm_print(0, "Vpmg_dbForce: No force for uniform dielectric!\n");
06069         return 1;
06070     }
06071     deps = (epsw - epsp);
06072     depsi = 1.0/deps;
06073     rtot = (arad + thee->splineWin + srاد);
06074
06075     /* Make sure we're on the grid */
06076     /* Grid checking modified by Matteo Rotter */
06077     if ((apos[0]<=xmin + rtot) || (apos[0]>=xmax - rtot) || \
06078         (apos[1]<=ymin + rtot) || (apos[1]>=ymax - rtot) || \
06079         (apos[2]<=zmin + rtot) || (apos[2]>=zmax - rtot)) {
06080         if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
06081             (thee->pmgp->bcfl != BCFL_MAP)) {
06082             Vnm_print(2, "Vpmg_dbForce: Atom #%d at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):\n",
06083                 atomID, apos[0], apos[1], apos[2]);
06084             Vnm_print(2, "Vpmg_dbForce:      xmin = %g, xmax = %g\n",
06085                 xmin, xmax);
06086             Vnm_print(2, "Vpmg_dbForce:      ymin = %g, ymax = %g\n",
06087                 ymin, ymax);
06088             Vnm_print(2, "Vpmg_dbForce:      zmin = %g, zmax = %g\n",
06089                 zmin, zmax);
06090         }
06091         fflush(stderr);
06092     } else {
06093
06094         /* Convert the atom position to grid reference frame */
06095         position[0] = apos[0] - xmin;
06096         position[1] = apos[1] - ymin;
06097         position[2] = apos[2] - zmin;
06098
06099         /* Integrate over points within this atom's (inflated) radius */
06100         rtot2 = VSQR(rtot);
06101         dx = rtot/hx;
06102         imin = (int) floor((position[0]-rtot)/hx);
06103         if (imin < 1) {
06104             Vnm_print(2, "Vpmg_dbForce: Atom %d off grid!\n", atomID);
06105             return 0;
06106         }

```

```

06107         imax = (int)ceil((position[0]+rtot)/hx);
06108         if (imax > (nx-2)) {
06109             Vnm_print(2, "Vpmg_dbForce: Atom %d off grid!\n", atomID);
06110             return 0;
06111         }
06112         jmin = (int)floor((position[1]-rtot)/hy);
06113         if (jmin < 1) {
06114             Vnm_print(2, "Vpmg_dbForce: Atom %d off grid!\n", atomID);
06115             return 0;
06116         }
06117         jmax = (int)ceil((position[1]+rtot)/hy);
06118         if (jmax > (ny-2)) {
06119             Vnm_print(2, "Vpmg_dbForce: Atom %d off grid!\n", atomID);
06120             return 0;
06121         }
06122         kmin = (int)floor((position[2]-rtot)/hz);
06123         if (kmin < 1) {
06124             Vnm_print(2, "Vpmg_dbForce: Atom %d off grid!\n", atomID);
06125             return 0;
06126         }
06127         kmax = (int)ceil((position[2]+rtot)/hz);
06128         if (kmax > (nz-2)) {
06129             Vnm_print(2, "Vpmg_dbForce: Atom %d off grid!\n", atomID);
06130             return 0;
06131         }
06132         for (i=imin; i<=imax; i++) {
06133             for (j=jmin; j<=jmax; j++) {
06134                 for (k=kmin; k<=kmax; k++) {
06135                     /* i, j, k */
06136                     gpos[0] = (i+0.5)*hx + xmin;
06137                     gpos[1] = j*hy + ymin;
06138                     gpos[2] = k*hz + zmin;
06139                     Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
06140
06141                     /* Select the correct function based on the surface definition
06142                      * (now including the 7th order polynomial) */
06143                     Vpmg_splineSelect(srfm, acc, gpos, thee->
splineWin, 0., atom, dHxijk);
06144                     /*
06145                      switch (srfm) {
06146                          case VSM_SPLINE :
06147                              Vacc_splineAccGradAtomNorm(acc, gpos, thee->splineWin, 0.,
06148                                                          atom, dHxijk);
06149                              break;
06150                          case VSM_SPLINE4 :
06151                              Vacc_splineAccGradAtomNorm4(acc, gpos, thee->splineWin, 0.,
06152                                                          atom, dHxijk);
06153                              break;
06154                          default:
06155                              Vnm_print(2, "Vpmg_dbnbForce: Unknown surface method.\n");
06156                              return;
06157                      }
06158                      */
06159                     for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
06160                     gpos[0] = i*hx + xmin;
06161                     gpos[1] = (j+0.5)*hy + ymin;
06162                     gpos[2] = k*hz + zmin;
06163                     Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
06164
06165                     /* Select the correct function based on the surface definition
06166                      * (now including the 7th order polynomial) */
06167                     Vpmg_splineSelect(srfm, acc, gpos, thee->
splineWin, 0., atom, dHyijk);
06168                     for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
06169                     gpos[0] = i*hx + xmin;
06170                     gpos[1] = j*hy + ymin;
06171                     gpos[2] = (k+0.5)*hz + zmin;
06172                     Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
06173
06174                     /* Select the correct function based on the surface definition
06175                      * (now including the 7th order polynomial) */
06176                     Vpmg_splineSelect(srfm, acc, gpos, thee->
splineWin, 0., atom, dHzijk);
06177                     for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
06178                     /* i-1, j, k */
06179                     gpos[0] = (i-0.5)*hx + xmin;
06180                     gpos[1] = j*hy + ymin;
06181                     gpos[2] = k*hz + zmin;
06182                     Hximljk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;

```

```

06185
06186 /* Select the correct function based on the surface definition
06187 * (now including the 7th order polynomial) */
06188 Vpmg_splineSelect(srfm,acc, gpos, thee->
    splineWin, 0.,atom, dHximljk);
06189
06190 for (l=0; l<3; l++) dHximljk[l] *= Hximljk;
06191 /* i,j-1,k */
06192 gpos[0] = i*hx + xmin;
06193 gpos[1] = (j-0.5)*hy + ymin;
06194 gpos[2] = k*hzed + zmin;
06195 Hyijmlk = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
06196
06197 /* Select the correct function based on the surface definition
06198 * (now including the 7th order polynomial) */
06199 Vpmg_splineSelect(srfm,acc, gpos, thee->
    splineWin, 0.,atom, dHyijmlk);
06200
06201 for (l=0; l<3; l++) dHyijmlk[l] *= Hyijmlk;
06202 /* i,j,k-1 */
06203 gpos[0] = i*hx + xmin;
06204 gpos[1] = j*hy + ymin;
06205 gpos[2] = (k-0.5)*hzed + zmin;
06206 Hzijkml = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
06207
06208 /* Select the correct function based on the surface definition
06209 * (now including the 7th order polynomial) */
06210 Vpmg_splineSelect(srfm,acc, gpos, thee->
    splineWin, 0.,atom, dHzijkml);
06211
06212 for (l=0; l<3; l++) dHzijkml[l] *= Hzijkml;
06213 /* *** CALCULATE DIELECTRIC BOUNDARY FORCES *** */
06214 dbFmag = u[IJK(i,j,k)];
06215 tgrad[0] =
06216     (dHxijk[0] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
06217      + dHximljk[0] * (u[IJK(i-1,j,k)]-u[IJK(i,j,k)])) /VSQR(hx)
06218     + (dHyijk[0] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
06219      + dHyijmlk[0] * (u[IJK(i,j-1,k)]-u[IJK(i,j,k)])) /VSQR(hy)
06220     + (dHzij[0] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
06221      + dHzijkml[0] * (u[IJK(i,j,k-1)]-u[IJK(i,j,k)])) /VSQR(hzed);
06222 tgrad[1] =
06223     (dHxijk[1] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
06224      + dHximljk[1] * (u[IJK(i-1,j,k)]-u[IJK(i,j,k)])) /VSQR(hx)
06225     + (dHyijk[1] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
06226      + dHyijmlk[1] * (u[IJK(i,j-1,k)]-u[IJK(i,j,k)])) /VSQR(hy)
06227     + (dHzij[1] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
06228      + dHzijkml[1] * (u[IJK(i,j,k-1)]-u[IJK(i,j,k)])) /VSQR(hzed);
06229 tgrad[2] =
06230     (dHxijk[2] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
06231      + dHximljk[2] * (u[IJK(i-1,j,k)]-u[IJK(i,j,k)])) /VSQR(hx)
06232     + (dHyijk[2] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
06233      + dHyijmlk[2] * (u[IJK(i,j-1,k)]-u[IJK(i,j,k)])) /VSQR(hy)
06234     + (dHzij[2] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
06235      + dHzijkml[2] * (u[IJK(i,j,k-1)]-u[IJK(i,j,k)])) /VSQR(hzed);
06236 dbForce[0] += (dbFmag*tgrad[0]);
06237 dbForce[1] += (dbFmag*tgrad[1]);
06238 dbForce[2] += (dbFmag*tgrad[2]);
06239
06240     } /* k loop */
06241     } /* j loop */
06242 } /* i loop */
06243
06244 dbForce[0] = -dbForce[0]*hx*hy*hzed*deps*0.5*izmagic;
06245 dbForce[1] = -dbForce[1]*hx*hy*hzed*deps*0.5*izmagic;
06246 dbForce[2] = -dbForce[2]*hx*hy*hzed*deps*0.5*izmagic;
06247 }
06248
06249 return 1;
06250 }
06251
06252 VPUBLIC int Vpmg_qfForce(Vpmg *thee, double *force, int atomID,
06253     Vchrg_Meth chgm) {
06254
06255     double tforce[3];
06256
06257     /* Reset force */
06258     force[0] = 0.0;
06259     force[1] = 0.0;
06260     force[2] = 0.0;
06261
06262     /* Check surface definition */

```

```

06263     if (chgm != VCM_BSPL2) {
06264         Vnm_print(2, "Vpmg_qfForce: It is recommended that forces be \
06265 calculated with the\n");
06266         Vnm_print(2, "Vpmg_qfForce: cubic spline charge discretization \
06267 scheme\n");
06268     }
06269
06270     switch (chgm) {
06271         case VCM_TRIL:
06272             qfForceSpline1(thee, tforce, atomID);
06273             break;
06274         case VCM_BSPL2:
06275             qfForceSpline2(thee, tforce, atomID);
06276             break;
06277         case VCM_BSPL4:
06278             qfForceSpline4(thee, tforce, atomID);
06279             break;
06280         default:
06281             Vnm_print(2, "Vpmg_qfForce: Undefined charge discretization \
06282 method (%d)\n", chgm);
06283             Vnm_print(2, "Vpmg_qfForce: Forces not calculated!\n");
06284             return 0;
06285     }
06286
06287     /* Assign forces */
06288     force[0] = tforce[0];
06289     force[1] = tforce[1];
06290     force[2] = tforce[2];
06291
06292     return 1;
06293 }
06294
06295
06296 VPRIVATE void qfForceSpline1(Vpmg *thee, double *force, int atomID) {
06297     Vatom *atom;
06298
06299     double *apos, position[3], hx, hy, hzed;
06300     double xmin, ymin, zmin, xmax, ymax, zmax;
06301     double dx, dy, dz;
06302     double *u, charge, ifloat, jfloat, kfloat;
06303     int nx, ny, nz, ihi, ilo, jhi, jlo, khi, klo;
06304
06305     VASSERT(thee != VNULL);
06306
06307     atom = Valist_getAtom(thee->pbe->alist, atomID);
06308     apos = Vatom_getPosition(atom);
06309     charge = Vatom_getCharge(atom);
06310
06311     /* Reset force */
06312     force[0] = 0.0;
06313     force[1] = 0.0;
06314     force[2] = 0.0;
06315
06316     /* If we aren't in the current position, then we're done */
06317     if (atom->partID == 0) return;
06318
06319     /* Mesh info */
06320     nx = thee->pmgp->nx;
06321     ny = thee->pmgp->ny;
06322     nz = thee->pmgp->nz;
06323     hx = thee->pmgp->hx;
06324     hy = thee->pmgp->hy;
06325     hzed = thee->pmgp->hzed;
06326     xmin = thee->pmgp->xmin;
06327     ymin = thee->pmgp->ymin;
06328     zmin = thee->pmgp->zmin;
06329     xmax = thee->pmgp->xmax;
06330     ymax = thee->pmgp->ymax;
06331     zmax = thee->pmgp->zmax;
06332     u = thee->u;
06333
06334     /* Make sure we're on the grid */
06335     if ((apos[0] <= xmin) || (apos[0] >= xmax) || (apos[1] <= ymin) || \
06336         (apos[1] >= ymax) || (apos[2] <= zmin) || (apos[2] >= zmax)) {
06337         if ((thee->pmgp->bconf != BCFL_FOCUS) &&
06338             (thee->pmgp->bconf != BCFL_MAP)) {
06339             Vnm_print(2, "Vpmg_qfForce: Atom # %d at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):\n",
06340 atomID, apos[0], apos[1], apos[2]);
06341             Vnm_print(2, "Vpmg_qfForce:      xmin = %g, xmax = %g\n", xmin, xmax);
06342             Vnm_print(2, "Vpmg_qfForce:      ymin = %g, ymax = %g\n", ymin, ymax);

```

```

06343         Vnm_print(2, "Vpmg_qfForce:      zmin = %g, zmax = %g\n", zmin, zmax);
06344     }
06345     fflush(stderr);
06346 } else {
06347
06348     /* Convert the atom position to grid coordinates */
06349     position[0] = apos[0] - xmin;
06350     position[1] = apos[1] - ymin;
06351     position[2] = apos[2] - zmin;
06352     ifloat = position[0]/hx;
06353     jfloat = position[1]/hy;
06354     kfloat = position[2]/hzed;
06355     ihi = (int)ceil(ifloat);
06356     ilo = (int)floor(ifloat);
06357     jhi = (int)ceil(jfloat);
06358     jlo = (int)floor(jfloat);
06359     khi = (int)ceil(kfloat);
06360     klo = (int)floor(kfloat);
06361     VASSERT((ihi < nx) && (ihi >=0));
06362     VASSERT((ilo < nx) && (ilo >=0));
06363     VASSERT((jhi < ny) && (jhi >=0));
06364     VASSERT((jlo < ny) && (jlo >=0));
06365     VASSERT((khi < nz) && (khi >=0));
06366     VASSERT((klo < nz) && (klo >=0));
06367     dx = ifloat - (double)(ilo);
06368     dy = jfloat - (double)(jlo);
06369     dz = kfloat - (double)(klo);
06370
06371
06372 #if 0
06373     Vnm_print(1, "Vpmg_qfForce: (DEBUG) u ~ %g\n",
06374         dx      *dy      *dz      *u[IJK(ihi,jhi,khi)]
06375         +dx      *dy      *(1-dz) *u[IJK(ihi,jhi,klo)]
06376         +dx      *(1-dy) *dz      *u[IJK(ihi,jlo,khi)]
06377         +dx      *(1-dy) *(1-dz) *u[IJK(ihi,jlo,klo)]
06378         + (1-dx) *dy      *dz      *u[IJK(ilo,jhi,khi)]
06379         + (1-dx) *dy      *(1-dz) *u[IJK(ilo,jhi,klo)]
06380         + (1-dx) *(1-dy) *dz      *u[IJK(ilo,jlo,khi)]
06381         + (1-dx) *(1-dy) *(1-dz) *u[IJK(ilo,jlo,klo)]);
06382 #endif
06383
06384
06385     if ((dx > VPMGSMALL) && (VABS(1.0-dx) > VPMGSMALL)) {
06386         force[0] =
06387             -charge*(dy      *dz      *u[IJK(ihi,jhi,khi)]
06388                 + dy      *(1-dz) *u[IJK(ihi,jhi,klo)]
06389                 + (1-dy) *dz      *u[IJK(ihi,jlo,khi)]
06390                 + (1-dy) *(1-dz) *u[IJK(ihi,jlo,klo)]
06391                 - dy      *dz      *u[IJK(ilo,jhi,khi)]
06392                 - dy      *(1-dz) *u[IJK(ilo,jhi,klo)]
06393                 - (1-dy) *dz      *u[IJK(ilo,jlo,khi)]
06394                 - (1-dy) *(1-dz) *u[IJK(ilo,jlo,klo)])/hx;
06395     } else {
06396         force[0] = 0;
06397         Vnm_print(0,
06398             "Vpmg_qfForce: Atom %d on x gridline; zero x-force\n", atomID);
06399     }
06400     if ((dy > VPMGSMALL) && (VABS(1.0-dy) > VPMGSMALL)) {
06401         force[1] =
06402             -charge*(dx      *dz      *u[IJK(ihi,jhi,khi)]
06403                 + dx      *(1-dz) *u[IJK(ihi,jhi,klo)]
06404                 - dx      *dz      *u[IJK(ihi,jlo,khi)]
06405                 - dx      *(1-dz) *u[IJK(ihi,jlo,klo)]
06406                 + (1-dx) *dz      *u[IJK(ilo,jhi,khi)]
06407                 + (1-dx) *(1-dz) *u[IJK(ilo,jhi,klo)]
06408                 - (1-dx) *dz      *u[IJK(ilo,jlo,khi)]
06409                 - (1-dx) *(1-dz) *u[IJK(ilo,jlo,klo)])/hy;
06410     } else {
06411         force[1] = 0;
06412         Vnm_print(0,
06413             "Vpmg_qfForce: Atom %d on y gridline; zero y-force\n", atomID);
06414     }
06415     if ((dz > VPMGSMALL) && (VABS(1.0-dz) > VPMGSMALL)) {
06416         force[2] =
06417             -charge*(dy      *dx      *u[IJK(ihi,jhi,khi)]
06418                 - dy      *dx      *u[IJK(ihi,jhi,klo)]
06419                 + (1-dy) *dx      *u[IJK(ihi,jlo,khi)]
06420                 - (1-dy) *dx      *u[IJK(ihi,jlo,klo)]
06421                 + dy      *(1-dx) *u[IJK(ilo,jhi,khi)]
06422                 - dy      *(1-dx) *u[IJK(ilo,jhi,klo)]
06423                 + (1-dy) *(1-dx) *u[IJK(ilo,jlo,khi)]

```

```

06424         - (1-dy)*(1-dx)*u[IJK(ilo,jlo,klo)])/hzd;
06425     } else {
06426         force[2] = 0;
06427         Vnm_print(0,
06428             "Vpmg_qfForce: Atom %d on z gridline; zero z-force\n", atomID);
06429     }
06430 }
06431 }
06432
06433 VPRIVATE void qfForceSpline2(Vpmg *thee, double *force, int atomID) {
06434
06435     Vatom *atom;
06436
06437     double *apos, position[3], hx, hy, hzed;
06438     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
06439     double mx, my, mz, dmX, dmy, dmz;
06440     double *u, charge, ifloat, jfloat, kfloat;
06441     int nx, ny, nz, im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1;
06442     int kp1, kp2, ii, jj, kk;
06443
06444     VASSERT(thee != VNULL);
06445
06446     atom = Valist_getAtom(thee->pbe->alist, atomID);
06447     apos = Vatom_getPosition(atom);
06448     charge = Vatom_getCharge(atom);
06449
06450     /* Reset force */
06451     force[0] = 0.0;
06452     force[1] = 0.0;
06453     force[2] = 0.0;
06454
06455     /* If we aren't in the current position, then we're done */
06456     if (atom->partID == 0) return;
06457
06458     /* Mesh info */
06459     nx = thee->pmgp->nx;
06460     ny = thee->pmgp->ny;
06461     nz = thee->pmgp->nz;
06462     hx = thee->pmgp->hx;
06463     hy = thee->pmgp->hy;
06464     hzed = thee->pmgp->hzd;
06465     xlen = thee->pmgp->xlen;
06466     ylen = thee->pmgp->ylen;
06467     zlen = thee->pmgp->zlen;
06468     xmin = thee->pmgp->xmin;
06469     ymin = thee->pmgp->ymin;
06470     zmin = thee->pmgp->zmin;
06471     xmax = thee->pmgp->xmax;
06472     ymax = thee->pmgp->ymax;
06473     zmax = thee->pmgp->zmax;
06474     u = thee->u;
06475
06476     /* Make sure we're on the grid */
06477     if ((apos[0] <= (xmin+hx)) || (apos[0] >= (xmax-hx)) \
06478         || (apos[1] <= (ymin+hy)) || (apos[1] >= (ymax-hy)) \
06479         || (apos[2] <= (zmin+hzed)) || (apos[2] >= (zmax-hzed))) {
06480         if ((thee->pmgp->bconf != BCFL_FOCUS) &&
06481             (thee->pmgp->bconf != BCFL_MAP)) {
06482             Vnm_print(2, "qfForceSpline2: Atom #%d off the mesh \
06483                 (ignoring)\n", atomID);
06484         }
06485         fflush(stderr);
06486
06487     } else {
06488
06489         /* Convert the atom position to grid coordinates */
06490         position[0] = apos[0] - xmin;
06491         position[1] = apos[1] - ymin;
06492         position[2] = apos[2] - zmin;
06493         ifloat = position[0]/hx;
06494         jfloat = position[1]/hy;
06495         kfloat = position[2]/hzed;
06496         ip1 = (int)ceil(ifloat);
06497         ip2 = ip1 + 1;
06498         im1 = (int)floor(ifloat);
06499         im2 = im1 - 1;
06500         jp1 = (int)ceil(jfloat);
06501         jp2 = jp1 + 1;
06502         jm1 = (int)floor(jfloat);
06503         jm2 = jm1 - 1;
06504         kp1 = (int)ceil(kfloat);

```

```

06505         kp2 = kp1 + 1;
06506         km1 = (int)floor(kfloat);
06507         km2 = km1 - 1;
06508
06509         /* This step shouldn't be necessary, but it saves nasty debugging
06510          * later on if something goes wrong */
06511         ip2 = VMIN2(ip2,nx-1);
06512         ip1 = VMIN2(ip1,nx-1);
06513         im1 = VMAX2(im1,0);
06514         im2 = VMAX2(im2,0);
06515         jp2 = VMIN2(jp2,ny-1);
06516         jp1 = VMIN2(jp1,ny-1);
06517         jm1 = VMAX2(jm1,0);
06518         jm2 = VMAX2(jm2,0);
06519         kp2 = VMIN2(kp2,nz-1);
06520         kp1 = VMIN2(kp1,nz-1);
06521         km1 = VMAX2(km1,0);
06522         km2 = VMAX2(km2,0);
06523
06524
06525         for (ii=im2; ii<=ip2; ii++) {
06526             mx = bspline2(VFCHI(ii,ifloat));
06527             dmx = dbspline2(VFCHI(ii,ifloat));
06528             for (jj=jm2; jj<=jp2; jj++) {
06529                 my = bspline2(VFCHI(jj,jfloat));
06530                 dmy = dbspline2(VFCHI(jj,jfloat));
06531                 for (kk=km2; kk<=kp2; kk++) {
06532                     mz = bspline2(VFCHI(kk,kfloat));
06533                     dmz = dbspline2(VFCHI(kk,kfloat));
06534
06535                     force[0] += (charge*dmx*my*mz*u[IJK(ii,jj,kk)])/hx;
06536                     force[1] += (charge*mx*dmy*mz*u[IJK(ii,jj,kk)])/hy;
06537                     force[2] += (charge*mx*my*dmz*u[IJK(ii,jj,kk)])/hz;
06538
06539                 }
06540             }
06541         }
06542     }
06543 }
06544 }
06545
06546 VPRIVATE void qfForceSpline4(Vpmg *thee, double *force, int atomID) {
06547
06548     Vatom *atom;
06549     double f, c, *u, *apos, position[3];
06550
06551     /* Grid variables */
06552     int nx,ny,nz;
06553     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
06554     double hx, hy, hzed, ifloat, jfloat, kfloat;
06555
06556     /* B-spline weights */
06557     double mx, my, mz, dmx, dmy, dmz;
06558     double mi, mj, mk;
06559
06560     /* Loop indeces */
06561     int i, j, k, ii, jj, kk;
06562     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
06563
06564     /* field */
06565     double e[3];
06566
06567     VASSERT(thee != VNULL);
06568     VASSERT(thee->filled);
06569
06570     atom = Valist_getAtom(thee->pbe->alist, atomID);
06571     apos = Vatom_getPosition(atom);
06572     c = Vatom_getCharge(atom);
06573
06574     for (i=0;i<3;i++){
06575         e[i] = 0.0;
06576     }
06577
06578     /* Mesh info */
06579     nx = thee->pmgp->nx;
06580     ny = thee->pmgp->ny;
06581     nz = thee->pmgp->nz;
06582     hx = thee->pmgp->hx;
06583     hy = thee->pmgp->hy;
06584     hzed = thee->pmgp->hz;
06585     xlen = thee->pmgp->xlen;

```

```

06586     ylen = thee->pmgp->ylen;
06587     zlen = thee->pmgp->zlen;
06588     xmin = thee->pmgp->xmin;
06589     ymin = thee->pmgp->ymin;
06590     zmin = thee->pmgp->zmin;
06591     xmax = thee->pmgp->xmax;
06592     ymax = thee->pmgp->ymax;
06593     zmax = thee->pmgp->zmax;
06594     u = thee->u;
06595
06596     /* Make sure we're on the grid */
06597     if ((apos[0]<=(xmin+2*hx)) || (apos[0]>=(xmax-2*hx)) \
06598         || (apos[1]<=(ymin+2*hy)) || (apos[1]>=(ymax-2*hy)) \
06599         || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
06600         Vnm_print(2, "qfForceSpline4: Atom off the mesh \
06601             (ignoring) %6.3f %6.3f %6.3f\n", apos[0], apos[1], apos[2]);
06602         fflush(stderr);
06603     } else {
06604
06605         /* Convert the atom position to grid coordinates */
06606         position[0] = apos[0] - xmin;
06607         position[1] = apos[1] - ymin;
06608         position[2] = apos[2] - zmin;
06609         ifloat = position[0]/hx;
06610         jfloat = position[1]/hy;
06611         kfloat = position[2]/hzed;
06612         ip1 = (int)ceil(ifloat);
06613         ip2 = ip1 + 2;
06614         im1 = (int)floor(ifloat);
06615         im2 = im1 - 2;
06616         jp1 = (int)ceil(jfloat);
06617         jp2 = jp1 + 2;
06618         jm1 = (int)floor(jfloat);
06619         jm2 = jm1 - 2;
06620         kp1 = (int)ceil(kfloat);
06621         kp2 = kp1 + 2;
06622         km1 = (int)floor(kfloat);
06623         km2 = km1 - 2;
06624
06625         /* This step shouldn't be necessary, but it saves nasty debugging
06626            * later on if something goes wrong */
06627         ip2 = VMIN2(ip2,nx-1);
06628         ip1 = VMIN2(ip1,nx-1);
06629         im1 = VMAX2(im1,0);
06630         im2 = VMAX2(im2,0);
06631         jp2 = VMIN2(jp2,ny-1);
06632         jp1 = VMIN2(jp1,ny-1);
06633         jm1 = VMAX2(jm1,0);
06634         jm2 = VMAX2(jm2,0);
06635         kp2 = VMIN2(kp2,nz-1);
06636         kp1 = VMIN2(kp1,nz-1);
06637         km1 = VMAX2(km1,0);
06638         km2 = VMAX2(km2,0);
06639
06640         for (ii=im2; ii<=ip2; ii++) {
06641             mi = VFCHI4(ii,ifloat);
06642             mx = bspline4(mi);
06643             dmx = dbspline4(mi);
06644             for (jj=jm2; jj<=jp2; jj++) {
06645                 mj = VFCHI4(jj,jfloat);
06646                 my = bspline4(mj);
06647                 dmy = dbspline4(mj);
06648                 for (kk=km2; kk<=kp2; kk++) {
06649                     mk = VFCHI4(kk,kfloat);
06650                     mz = bspline4(mk);
06651                     dmz = dbspline4(mk);
06652                     f = u[IJK(ii,jj,kk)];
06653                     /* Field */
06654                     e[0] += f*dmx*my*mz/hx;
06655                     e[1] += f*mx*dmy*mz/hy;
06656                     e[2] += f*mx*my*dmz/hzed;
06657                 }
06658             }
06659         }
06660     }
06661
06662     /* Monopole Force */
06663     force[0] = e[0]*c;
06664     force[1] = e[1]*c;
06665     force[2] = e[2]*c;
06666

```



```

06667 }
06668
06669 VPRIVATE void markFrac(
06670     double rtot, double *tpos,
06671     int nx, int ny, int nz,
06672     double hx, double hy, double hzed,
06673     double xmin, double ymin, double zmin,
06674     double *xarray, double *yarray, double *zarray) {
06675
06676     int i, j, k, imin, imax, jmin, jmax, kmin, kmax;
06677     double dx, dx2, dy, dy2, dz, dz2, a000, a001, a010, a100, r2;
06678     double x, xp, xm, y, yp, ym, zp, z, zm, xspan, yspan, zspan;
06679     double rtot2, pos[3];
06680
06681     /* Convert to grid reference frame */
06682     pos[0] = tpos[0] - xmin;
06683     pos[1] = tpos[1] - ymin;
06684     pos[2] = tpos[2] - zmin;
06685
06686     rtot2 = VSQR(rtot);
06687
06688     xspan = rtot + 2*hx;
06689     imin = VMAX2(0, (int)ceil((pos[0] - xspan)/hx));
06690     imax = VMIN2(nx-1, (int)floor((pos[0] + xspan)/hx));
06691     for (i=imin; i<=imax; i++) {
06692         x = hx*i;
06693         dx2 = VSQR(pos[0] - x);
06694         if (rtot2 > dx2) {
06695             yspan = VSQRT(rtot2 - dx2) + 2*hy;
06696         } else {
06697             yspan = 2*hy;
06698         }
06699         jmin = VMAX2(0, (int)ceil((pos[1] - yspan)/hy));
06700         jmax = VMIN2(ny-1, (int)floor((pos[1] + yspan)/hy));
06701         for (j=jmin; j<=jmax; j++) {
06702             y = hy*j;
06703             dy2 = VSQR(pos[1] - y);
06704             if (rtot2 > (dx2+dy2)) {
06705                 zspan = VSQRT(rtot2-dx2-dy2) + 2*hzed;
06706             } else {
06707                 zspan = 2*hzed;
06708             }
06709             kmin = VMAX2(0, (int)ceil((pos[2] - zspan)/hzed));
06710             kmax = VMIN2(nz-1, (int)floor((pos[2] + zspan)/hzed));
06711             for (k=kmin; k<=kmax; k++) {
06712                 z = hzed*k;
06713                 dz2 = VSQR(pos[2] - z);
06714
06715                 r2 = dx2 + dy2 + dz2;
06716
06717                 /* We need to determine the inclusion value a000 at (i,j,k) */
06718                 if (r2 < rtot2) a000 = 1.0;
06719                 else a000 = 0.0;
06720
06721                 /* We need to evaluate the values of x which intersect the
06722                  * sphere and determine if these are in the interval
06723                  * [(i,j,k), (i+1,j,k)] */
06724                 if (r2 < (rtot2 - hx*hx)) a100 = 1.0;
06725                 else if (r2 > (rtot2 + hx*hx)) a100 = 0.0;
06726                 else if (rtot2 > (dy2 + dz2)) {
06727                     dx = VSQRT(rtot2 - dy2 - dz2);
06728                     xm = pos[0] - dx;
06729                     xp = pos[0] + dx;
06730                     if ((xm < x+hx) && (xm > x)) {
06731                         a100 = (xm - x)/hx;
06732                     } else if ((xp < x+hx) && (xp > x)) {
06733                         a100 = (xp - x)/hx;
06734                     }
06735                 } else a100 = 0.0;
06736
06737                 /* We need to evaluate the values of y which intersect the
06738                  * sphere and determine if these are in the interval
06739                  * [(i,j,k), (i,j+1,k)] */
06740                 if (r2 < (rtot2 - hy*hy)) a010 = 1.0;
06741                 else if (r2 > (rtot2 + hy*hy)) a010 = 0.0;
06742                 else if (rtot2 > (dx2 + dz2)) {
06743                     dy = VSQRT(rtot2 - dx2 - dz2);
06744                     ym = pos[1] - dy;
06745                     yp = pos[1] + dy;
06746                     if ((ym < y+hy) && (ym > y)) {
06747                         a010 = (ym - y)/hy;

```

```

06748         } else if ((yp < y+hy) && (yp > y)) {
06749             a010 = (yp - y)/hy;
06750         }
06751     } else a010 = 0.0;
06752
06753     /* We need to evaluate the values of y which intersect the
06754      * sphere and determine if these are in the interval
06755      * [(i,j,k), (i,j,k+1)] */
06756     if (r2 < (rtot2 - hzed*hzed)) a001 = 1.0;
06757     else if (r2 > (rtot2 + hzed*hzed)) a001 = 0.0;
06758     else if (rtot2 > (dx2 + dy2)) {
06759         dz = VSQRT(rtot2 - dx2 - dy2);
06760         zm = pos[2] - dz;
06761         zp = pos[2] + dz;
06762         if ((zm < z+hzed) && (zm > z)) {
06763             a001 = (zm - z)/hzed;
06764         } else if ((zp < z+hzed) && (zp > z)) {
06765             a001 = (zp - z)/hzed;
06766         }
06767     } else a001 = 0.0;
06768
06769     if (a100 < xarray[IJK(i,j,k)]) xarray[IJK(i,j,k)] = a100;
06770     if (a010 < yarray[IJK(i,j,k)]) yarray[IJK(i,j,k)] = a010;
06771     if (a001 < zarray[IJK(i,j,k)]) zarray[IJK(i,j,k)] = a001;
06772
06773     } /* k loop */
06774 } /* j loop */
06775 } /* i loop */
06776 }
06777
06778 /*
06779
06780 NOTE: This is the original version of the markSphere function. It's in here
06781 for reference and in case a reversion to the original code is needed.
06782 D. Gohara (2/14/08)
06783 */
06784 /*
06785 VPRIVATE void markSphere(
06786     double rtot, double *tpos,
06787     int nx, int ny, int nz,
06788     double hx, double hy, double hzed,
06789     double xmin, double ymin, double zmin,
06790     double *array, double markVal) {
06791
06792     int i, j, k, imin, imax, jmin, jmax, kmin, kmax;
06793     double dx, dx2, dy, dy2, dz, dz2;
06794     double rtot2, pos[3];
06795
06796     // Convert to grid reference frame
06797     pos[0] = tpos[0] - xmin;
06798     pos[1] = tpos[1] - ymin;
06799     pos[2] = tpos[2] - zmin;
06800
06801     rtot2 = VSQR(rtot);
06802
06803     dx = rtot + 0.5*hx;
06804     imin = VMAX2(0, (int)ceil((pos[0] - dx)/hx));
06805     imax = VMIN2(nx-1, (int)floor((pos[0] + dx)/hx));
06806     for (i=imin; i<=imax; i++) {
06807         dx2 = VSQR(pos[0] - hx*i);
06808         if (rtot2 > dx2) {
06809             dy = VSQRT(rtot2 - dx2) + 0.5*hy;
06810         } else {
06811             dy = 0.5*hy;
06812         }
06813         jmin = VMAX2(0, (int)ceil((pos[1] - dy)/hy));
06814         jmax = VMIN2(ny-1, (int)floor((pos[1] + dy)/hy));
06815         for (j=jmin; j<=jmax; j++) {
06816             dy2 = VSQR(pos[1] - hy*j);
06817             if (rtot2 > (dx2+dy2)) {
06818                 dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
06819             } else {
06820                 dz = 0.5*hzed;
06821             }
06822             kmin = VMAX2(0, (int)ceil((pos[2] - dz)/hzed));
06823             kmax = VMIN2(nz-1, (int)floor((pos[2] + dz)/hzed));
06824             for (k=kmin; k<=kmax; k++) {
06825                 dz2 = VSQR(k*hzed - pos[2]);
06826                 if ((dz2 + dy2 + dx2) <= rtot2) {
06827                     array[IJK(i,j,k)] = markVal;
06828                 }

```

```

06829         } // k loop
06830     } // j loop
06831 } // i loop
06832 }
06833 */
06834 VPRIVATE void markSphere(double rtot, double *tpos,
06835     int nx, int ny, int nz,
06836     double hx, double hy, double hz,
06837     double xmin, double ymin, double zmin,
06838     double *array, double markVal) {
06839
06840     int i, j, k;
06841     double fi,fj,fk;
06842     int imin, imax;
06843     int jmin, jmax;
06844     int kmin, kmax;
06845     double dx2, dy2, dz2;
06846     double xrange, yrange, zrange;
06847     double rtot2, posx, posy, posz;
06848
06849     /* Convert to grid reference frame */
06850     posx = tpos[0] - xmin;
06851     posy = tpos[1] - ymin;
06852     posz = tpos[2] - zmin;
06853
06854     rtot2 = VSQR(rtot);
06855
06856     xrange = rtot + 0.5 * hx;
06857     yrange = rtot + 0.5 * hy;
06858     zrange = rtot + 0.5 * hz;
06859
06860     imin = VMAX2(0, (int)ceil((posx - xrange)/hx));
06861     jmin = VMAX2(0, (int)ceil((posy - yrange)/hy));
06862     kmin = VMAX2(0, (int)ceil((posz - zrange)/hz));
06863
06864     imax = VMIN2(nx-1, (int)floor((posx + xrange)/hx));
06865     jmax = VMIN2(ny-1, (int)floor((posy + yrange)/hy));
06866     kmax = VMIN2(nz-1, (int)floor((posz + zrange)/hz));
06867
06868     for (i=imin,fi=imin; i<=imax; i++, fi+=1.) {
06869         dx2 = VSQR(posx - hx*fi);
06870         for (j=jmin,fj=jmin; j<=jmax; j++, fj+=1.) {
06871             dy2 = VSQR(posy - hy*fj);
06872             if((dx2 + dy2) > rtot2) continue;
06873             for (k=kmin,fk=kmin; k<=kmax; k++, fk+=1.) {
06874                 dz2 = VSQR(posz - hz*fk);
06875                 if ((dz2 + dy2 + dx2) <= rtot2) {
06876                     array[IJK(i,j,k)] = markVal;
06877                 }
06878             }
06879         }
06880     }
06881 }
06882
06883 VPRIVATE void zlapSolve(
06884     Vpmg *thee,
06885     double **solution,
06886     double **source,
06887     double **work1
06888 ) {
06889
06890     /* NOTE: this is an incredibly inefficient algorithm. The next
06891      * improvement is to focus on only non-zero entries in the source term.
06892      * The best improvement is to use a fast sine transform */
06893
06894     int n, nx, ny, nz, i, j, k, kx, ky, kz;
06895     double hx, hy, hzed, wx, wy, wz, xlen, ylen, zlen;
06896     double phix, phixpl, phixml, phiy, phiyml, phiypl, phiz, phizml, phizpl;
06897     double norm, coef, proj, eigx, eigy, eigz;
06898     double ihx2, ihy2, ihzed2;
06899     double *u, *f, *phi;
06900
06901     /* Snarf grid parameters */
06902     nx = thee->pmgp->nx;
06903     ny = thee->pmgp->ny;
06904     nz = thee->pmgp->nz;
06905     n = nx*ny*nz;
06906     hx = thee->pmgp->hx;
06907     ihx2 = 1.0/hx/hx;
06908     hy = thee->pmgp->hy;
06909     ihy2 = 1.0/hy/hy;

```

```

06910     hzed = thee->pmgp->hzed;
06911     ihzed2 = 1.0/hzed/hzed;
06912     xlen = thee->pmgp->xlen;
06913     ylen = thee->pmgp->ylen;
06914     zlen = thee->pmgp->zlen;
06915
06916     /* Set solution and source array pointers */
06917     u = *solution;
06918     f = *source;
06919     phi = *work1;
06920
06921     /* Zero out the solution vector */
06922     for (i=0; i<n; i++) thee->u[i] = 0.0;
06923
06924     /* Iterate through the wavenumbers */
06925     for (kx=1; kx<(nx-1); kx++) {
06926
06927         wx = (VPI*(double)kx)/((double)nx - 1.0);
06928         eigx = 2.0*ihx2*(1.0 - cos(wx));
06929
06930         for (ky=1; ky<(ny-1); ky++) {
06931
06932             wy = (VPI*(double)ky)/((double)ny - 1.0);
06933             eigy = 2.0*ihy2*(1.0 - cos(wy));
06934
06935             for (kz=1; kz<(nz-1); kz++) {
06936
06937                 wz = (VPI*(double)kz)/((double)nz - 1.0);
06938                 eigz = 2.0*ihzed2*(1.0 - cos(wz));
06939
06940                 /* Calculate the basis function.
06941                  * We could calculate each basis function as
06942                  *   phix(i) = sin(wx*i)
06943                  *   phiy(j) = sin(wy*j)
06944                  *   phiz(k) = sin(wz*k)
06945                  * However, this is likely to be very expensive.
06946                  * Therefore, we can use the fact that
06947                  *   phix(i+1) = (2-hx*hx*eigx)*phix(i) - phix(i-1)
06948                  * */
06949                 for (i=1; i<(nx-1); i++) {
06950                     if (i == 1) {
06951                         phix = sin(wx*(double)i);
06952                         phixml = 0.0;
06953                     } else {
06954                         phixpl = (2.0-hx*hx*eigx)*phix - phixml;
06955                         phixml = phix;
06956                         phix = phixpl;
06957                     }
06958                     /* phix = sin(wx*(double)i); */
06959                     for (j=1; j<(ny-1); j++) {
06960                         if (j == 1) {
06961                             phiy = sin(wy*(double)j);
06962                             phiyml = 0.0;
06963                         } else {
06964                             phiypl = (2.0-hy*hy*eigy)*phiy - phiyml;
06965                             phiyml = phiy;
06966                             phiy = phiypl;
06967                         }
06968                         /* phiy = sin(wy*(double)j); */
06969                         for (k=1; k<(nz-1); k++) {
06970                             if (k == 1) {
06971                                 phiz = sin(wz*(double)k);
06972                                 phizml = 0.0;
06973                             } else {
06974                                 phizpl = (2.0-hzed*hzed*eigz)*phiz - phizml;
06975                                 phizml = phiz;
06976                                 phiz = phizpl;
06977                             }
06978                             /* phiz = sin(wz*(double)k); */
06979                             phi[IJK(i,j,k)] = phix*phiy*phiz;
06980
06981                         }
06982                     }
06983                 }
06984             }
06985
06986             /* Calculate the projection of the source function on this
06987              * basis function */
06988             proj = 0.0;
06989             for (i=1; i<(nx-1); i++) {
06990                 for (j=1; j<(ny-1); j++) {

```

```

06991         for (k=1; k<(nz-1); k++) {
06992
06993             proj += f[IJK(i,j,k)]*phi[IJK(i,j,k)];
06994
06995             } /* k loop */
06996         } /* j loop */
06997     } /* i loop */
06998
06999     /* Assemble the coefficient to weight the contribution of this
07000      * basis function to the solution */
07001     /* The first contribution is the projection */
07002     coef = proj;
07003     /* The second contribution is the eigenvalue */
07004     coef = coef/(eigx + eigy + eigz);
07005     /* The third contribution is the normalization factor */
07006     coef = (8.0/xlen/ylen/zlen)*coef;
07007     /* The fourth contribution is from scaling the diagonal */
07008     /* coef = hx*hy*hzed*coef; */
07009
07010     /* Evaluate the basis function at each grid point */
07011     for (i=1; i<(nx-1); i++) {
07012         for (j=1; j<(ny-1); j++) {
07013             for (k=1; k<(nz-1); k++) {
07014
07015                 u[IJK(i,j,k)] += coef*phi[IJK(i,j,k)];
07016
07017             } /* k loop */
07018         } /* j loop */
07019     } /* i loop */
07020
07021     } /* kz loop */
07022     } /* ky loop */
07023 } /* kx loop */
07024 }
07025 }
07026
07027 VPUBLIC int Vpmg_solveLaplace(Vpmg *thee) {
07028
07029     int i, j, k, ijk, nx, ny, nz, n, dilo, dihi, djlo, djhi, dklo, dkhi;
07030     double hx, hy, hzed, epsw, iepsw, scal, scalx, scaly, scalz;
07031
07032     nx = thee->pmgp->nx;
07033     ny = thee->pmgp->ny;
07034     nz = thee->pmgp->nz;
07035     n = nx*ny*nz;
07036     hx = thee->pmgp->hx;
07037     hy = thee->pmgp->hy;
07038     hzed = thee->pmgp->hzed;
07039     epsw = Vpbe_getSolventDiel(thee->pbe);
07040     iepsw = 1.0/epsw;
07041     scal = hx*hy*hzed;
07042     scalx = hx*hy/hzed;
07043     scaly = hx*hzed/hy;
07044     scalz = hx*hy/hzed;
07045
07046     if (!(thee->filled)) {
07047         Vnm_print(2, "Vpmg_solve: Need to call Vpmg_fillco()\n");
07048         return 0;
07049     }
07050
07051     /* Load boundary conditions into the RHS array */
07052     for (i=1; i<(nx-1); i++) {
07053
07054         if (i == 1) dilo = 1;
07055         else dilo = 0;
07056         if (i == nx-2) dihi = 1;
07057         else dihi = 0;
07058
07059         for (j=1; j<(ny-1); j++) {
07060
07061             if (j == 1) djlo = 1;
07062             else djlo = 0;
07063             if (j == ny-2) djhi = 1;
07064             else djhi = 0;
07065
07066             for (k=1; k<(nz-1); k++) {
07067
07068                 if (k == 1) dklo = 1;
07069                 else dklo = 0;
07070                 if (k == nz-2) dkhi = 1;
07071                 else dkhi = 0;

```

```

07072
07073         thee->fcf[IJK(i,j,k)] = \
07074             iepsw*scal*thee->charge[IJK(i,j,k)] \
07075             + dilo*scalx*thee->gxcf[IJKx(j,k,0)] \
07076             + dihi*scalx*thee->gxcf[IJKx(j,k,1)] \
07077             + djlo*scaly*thee->gycf[IJKy(i,k,0)] \
07078             + djhi*scaly*thee->gycf[IJKy(i,k,1)] \
07079             + dklo*scalz*thee->gzcf[IJKz(i,j,0)] \
07080             + dkhi*scalz*thee->gzcf[IJKz(i,j,1)] ;
07081
07082     }
07083 }
07084
07085 }
07086
07087 /* Solve */
07088 zlapSolve( thee, &(thee->u), &(thee->fcf), &(thee->tcf) );
07089
07090 /* Add boundary conditions to solution */
07091 /* i faces */
07092 for (j=0; j<ny; j++) {
07093     for (k=0; k<nz; k++) {
07094         thee->u[IJK(0,j,k)] = thee->gxcf[IJKx(j,k,0)];
07095         thee->u[IJK(nx-1,j,k)] = thee->gycf[IJKx(j,k,1)];
07096     }
07097 }
07098 /* j faces */
07099 for (i=0; i<nx; i++) {
07100     for (k=0; k<nz; k++) {
07101         thee->u[IJK(i,0,k)] = thee->gycf[IJKy(i,k,0)];
07102         thee->u[IJK(i,ny-1,k)] = thee->gycf[IJKy(i,k,1)];
07103     }
07104 }
07105 /* k faces */
07106 for (i=0; i<nx; i++) {
07107     for (j=0; j<ny; j++) {
07108         thee->u[IJK(i,j,0)] = thee->gzcf[IJKz(i,j,0)];
07109         thee->u[IJK(i,j,nz-1)] = thee->gzcf[IJKz(i,j,1)];
07110     }
07111 }
07112
07113 return 1;
07114
07115 }
07116
07117 VPRIVATE double VFCHI4(int i, double f) {
07118     return (2.5+((double) (i)-(f)));
07119 }
07120
07121 VPRIVATE double bspline4(double x) {
07122
07123     double m, m2;
07124     static double one6 = 1.0/6.0;
07125     static double one8 = 1.0/8.0;
07126     static double one24 = 1.0/24.0;
07127     static double thirteen24 = 13.0/24.0;
07128     static double fortyseven24 = 47.0/24.0;
07129     static double seventeen24 = 17.0/24.0;
07130
07131     if ((x > 0.0) && (x <= 1.0)){
07132         m = x*x;
07133         return one24*m*m;
07134     } else if ((x > 1.0) && (x <= 2.0)){
07135         m = x - 1.0;
07136         m2 = m*m;
07137         return -one8 + one6*x + m2*(0.25 + one6*m - one6*m2);
07138     } else if ((x > 2.0) && (x <= 3.0)){
07139         m = x - 2.0;
07140         m2 = m*m;
07141         return -thirteen24 + 0.5*x + m2*(-0.25 - 0.5*m + 0.25*m2);
07142     } else if ((x > 3.0) && (x <= 4.0)){
07143         m = x - 3.0;
07144         m2 = m*m;
07145         return fortyseven24 - 0.5*x + m2*(-0.25 + 0.5*m - one6*m2);
07146     } else if ((x > 4.0) && (x <= 5.0)){
07147         m = x - 4.0;
07148         m2 = m*m;
07149         return seventeen24 - one6*x + m2*(0.25 - one6*m + one24*m2);
07150     } else {
07151         return 0.0;
07152     }
07153 }

```

```

07154
07155 VPUBLIC double dbspline4(double x) {
07156
07157     double m, m2;
07158     static double one6 = 1.0/6.0;
07159     static double one3 = 1.0/3.0;
07160     static double two3 = 2.0/3.0;
07161     static double thirteen6 = 13.0/6.0;
07162
07163     if ((x > 0.0) && (x <= 1.0)){
07164         m2 = x*x;
07165         return one6*x*m2;
07166     } else if ((x > 1.0) && (x <= 2.0)){
07167         m = x - 1.0;
07168         m2 = m*m;
07169         return -one3 + 0.5*x + m2*(0.5 - two3*m);
07170     } else if ((x > 2.0) && (x <= 3.0)){
07171         m = x - 2.0;
07172         m2 = m*m;
07173         return 1.5 - 0.5*x + m2*(-1.5 + m);
07174     } else if ((x > 3.0) && (x <= 4.0)){
07175         m = x - 3.0;
07176         m2 = m*m;
07177         return 1.0 - 0.5*x + m2*(1.5 - two3*m);
07178     } else if ((x > 4.0) && (x <= 5.0)){
07179         m = x - 4.0;
07180         m2 = m*m;
07181         return -thirteen6 + 0.5*x + m2*(-0.5 + one6*m);
07182     } else {
07183         return 0.0;
07184     }
07185 }
07186
07187 VPUBLIC double d2bspline4(double x) {
07188
07189     double m, m2;
07190
07191     if ((x > 0.0) && (x <= 1.0)){
07192         return 0.5*x*x;
07193     } else if ((x > 1.0) && (x <= 2.0)){
07194         m = x - 1.0;
07195         m2 = m*m;
07196         return -0.5 + x - 2.0*m2;
07197     } else if ((x > 2.0) && (x <= 3.0)){
07198         m = x - 2.0;
07199         m2 = m*m;
07200         return 5.5 - 3.0*x + 3.0*m2;
07201     } else if ((x > 3.0) && (x <= 4.0)){
07202         m = x - 3.0;
07203         m2 = m*m;
07204         return -9.5 + 3.0*x - 2.0*m2;
07205     } else if ((x > 4.0) && (x <= 5.0)){
07206         m = x - 4.0;
07207         m2 = m*m;
07208         return 4.5 - x + 0.5*m2;
07209     } else {
07210         return 0.0;
07211     }
07212 }
07213
07214 VPUBLIC double d3bspline4(double x) {
07215
07216     if ((x > 0.0) && (x <= 1.0)) return x;
07217     else if ((x > 1.0) && (x <= 2.0)) return 5.0 - 4.0 * x;
07218     else if ((x > 2.0) && (x <= 3.0)) return -15.0 + 6.0 * x;
07219     else if ((x > 3.0) && (x <= 4.0)) return 15.0 - 4.0 * x;
07220     else if ((x > 4.0) && (x <= 5.0)) return x - 5.0;
07221     else return 0.0;
07222 }
07223 }
07224
07225 VPUBLIC void fillcoPermanentMultipole(Vpmg *three) {
07226
07227     Valist *alist;
07228     Vpbe *pbe;
07229     Vatom *atom;
07230     /* Conversions */
07231     double zmagic, f;
07232     /* Grid */
07233     double xmin, xmax, ymin, ymax, zmin, zmax;
07234     double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;

```

```

07235 double hx, hy, hzed, *apos;
07236 /* Multipole */
07237 double charge, *dipole,*quad;
07238 double c,ux,uy,uz,qxx,qyx,qyy,qzx,qzy,qzz,qave;
07239 /* B-spline weights */
07240 double mx,my,mz,dmx,dmy,dmz,d2mx,d2my,d2mz;
07241 double mi,mj,mk;
07242 /* Loop variables */
07243 int i, ii, jj, kk, nx, ny, nz, iatom;
07244 int im2, im1, ip1, ip2, jm2, jm1, jpl, jp2, km2, km1, kpl, kp2;
07245
07246 /* sanity check */
07247 double mir,mjr,mkr,mr2;
07248 double debye,mc,mux,muy,muz,mqxx,mqyx,mqyy,mqzx,mqzy,mqzz;
07249
07250 VASSERT(thee != VNULL);
07251
07252 /* Get PBE info */
07253 pbe = thee->pbe;
07254 alist = pbe->alist;
07255 zmagic = Vpbe_getZmagic(pbe);
07256
07257 /* Mesh info */
07258 nx = thee->pmgp->nx;
07259 ny = thee->pmgp->ny;
07260 nz = thee->pmgp->nz;
07261 hx = thee->pmgp->hx;
07262 hy = thee->pmgp->hy;
07263 hzed = thee->pmgp->hzed;
07264
07265 /* Conversion */
07266 f = zmagic/(hx*hy*hzed);
07267
07268 /* Define the total domain size */
07269 xlen = thee->pmgp->xlen;
07270 ylen = thee->pmgp->ylen;
07271 zlen = thee->pmgp->zlen;
07272
07273 /* Define the min/max dimensions */
07274 xmin = thee->pmgp->xcent - (xlen/2.0);
07275 ymin = thee->pmgp->ycent - (ylen/2.0);
07276 zmin = thee->pmgp->zcent - (zlen/2.0);
07277 xmax = thee->pmgp->xcent + (xlen/2.0);
07278 ymax = thee->pmgp->ycent + (ylen/2.0);
07279 zmax = thee->pmgp->zcent + (zlen/2.0);
07280
07281 /* Fill in the source term (permanent atomic multipoles) */
07282 Vnm_print(0, "fillcoPermanentMultipole: filling in source term.\n");
07283 for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
07284
07285     atom = Valist_getAtom(alist, iatom);
07286     apos = Vatom_getPosition(atom);
07287
07288     c = Vatom_getCharge(atom)*f;
07289
07290 #if defined(WITH_TINKER)
07291     dipole = Vatom_getDipole(atom);
07292     ux = dipole[0]/hx*f;
07293     uy = dipole[1]/hy*f;
07294     uz = dipole[2]/hzed*f;
07295     quad = Vatom_getQuadrupole(atom);
07296     qxx = (1.0/3.0)*quad[0]/(hx*hx)*f;
07297     qyx = (2.0/3.0)*quad[3]/(hx*hy)*f;
07298     qyy = (1.0/3.0)*quad[4]/(hy*hy)*f;
07299     qzx = (2.0/3.0)*quad[6]/(hzed*hx)*f;
07300     qzy = (2.0/3.0)*quad[7]/(hzed*hy)*f;
07301     qzz = (1.0/3.0)*quad[8]/(hzed*hzed)*f;
07302 #else
07303     ux = 0.0;
07304     uy = 0.0;
07305     uz = 0.0;
07306     qxx = 0.0;
07307     qyx = 0.0;
07308     qyy = 0.0;
07309     qzx = 0.0;
07310     qzy = 0.0;
07311     qzz = 0.0;
07312 #endif /* if defined(WITH_TINKER) */
07313
07314 /* check
07315 mc = 0.0;

```



```

07316     mux = 0.0;
07317     muy = 0.0;
07318     muz = 0.0;
07319     mqxx = 0.0;
07320     mqyx = 0.0;
07321     mqyy = 0.0;
07322     mqzx = 0.0;
07323     mqzy = 0.0;
07324     mqzz = 0.0; */
07325
07326     /* Make sure we're on the grid */
07327     if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx)) || \
07328         (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy)) || \
07329         (apos[2]<=(zmin-2*hz)) || (apos[2]>=(zmax+2*hz))) {
07330         Vnm_print(2, "fillcoPermanentMultipole: Atom #%d at (%4.3f, %4.3f, %4.3f) is off the mesh
07331         (ignoring this atom):\n", iatom, apos[0], apos[1], apos[2]);
07332         Vnm_print(2, "fillcoPermanentMultipole: xmin = %g, xmax = %g\n", xmin, xmax);
07333         Vnm_print(2, "fillcoPermanentMultipole: ymin = %g, ymax = %g\n", ymin, ymax);
07334         Vnm_print(2, "fillcoPermanentMultipole: zmin = %g, zmax = %g\n", zmin, zmax);
07335         fflush(stderr);
07336     } else {
07337
07338         /* Convert the atom position to grid reference frame */
07339         position[0] = apos[0] - xmin;
07340         position[1] = apos[1] - ymin;
07341         position[2] = apos[2] - zmin;
07342
07343         /* Figure out which vertices we're next to */
07344         ifloat = position[0]/hx;
07345         jfloat = position[1]/hy;
07346         kfloat = position[2]/hz;
07347
07348         ip1 = (int)ceil(ifloat);
07349         ip2 = ip1 + 2;
07350         im1 = (int)floor(ifloat);
07351         im2 = im1 - 2;
07352         jp1 = (int)ceil(jfloat);
07353         jp2 = jp1 + 2;
07354         jm1 = (int)floor(jfloat);
07355         jm2 = jm1 - 2;
07356         kp1 = (int)ceil(kfloat);
07357         kp2 = kp1 + 2;
07358         km1 = (int)floor(kfloat);
07359         km2 = km1 - 2;
07360
07361         /* This step shouldn't be necessary, but it saves nasty debugging
07362         * later on if something goes wrong */
07363         ip2 = VMIN2(ip2,nx-1);
07364         ip1 = VMIN2(ip1,nx-1);
07365         im1 = VMAX2(im1,0);
07366         im2 = VMAX2(im2,0);
07367         jp2 = VMIN2(jp2,ny-1);
07368         jp1 = VMIN2(jp1,ny-1);
07369         jm1 = VMAX2(jm1,0);
07370         jm2 = VMAX2(jm2,0);
07371         kp2 = VMIN2(kp2,nz-1);
07372         kp1 = VMIN2(kp1,nz-1);
07373         km1 = VMAX2(km1,0);
07374         km2 = VMAX2(km2,0);
07375
07376         /* Now assign fractions of the charge to the nearby verts */
07377         for (ii=im2; ii<=ip2; ii++) {
07378             mi = VFCH14(ii,ifloat);
07379             mx = bspline4(mi);
07380             dmx = dbspline4(mi);
07381             d2mx = d2bspline4(mi);
07382             for (jj=jm2; jj<=jp2; jj++) {
07383                 mj = VFCH14(jj,jfloat);
07384                 my = bspline4(mj);
07385                 dmy = dbspline4(mj);
07386                 d2my = d2bspline4(mj);
07387                 for (kk=km2; kk<=kp2; kk++) {
07388                     mk = VFCH14(kk,kfloat);
07389                     mz = bspline4(mk);
07390                     dmz = dbspline4(mk);
07391                     d2mz = d2bspline4(mk);
07392                     charge = mx*my*mz*c -
07393                         dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz +
07394                         d2mx*my*mz*qxx +
07395                         dmx*dmy*mz*qyx + mx*d2my*mz*qyy +
07396                         dmx*my*dmz*qzx + mx*dmy*dmz*qzy + mx*my*d2mz*qzz;

```

```

07396         thee->charge[IJK(ii,jj,kk)] += charge;
07397
07398         /* sanity check - recalculate traceless multipoles
07399            from the grid charge distribution for this
07400            site.
07401
07402            mir = (mi - 2.5) * hx;
07403            mjr = (mj - 2.5) * hy;
07404            mkr = (mk - 2.5) * hzed;
07405            mr2 = mir*mir+mjr*mjr+mkr*mkr;
07406            mc += charge;
07407            mux += mir * charge;
07408            muy += mjr * charge;
07409            muz += mkr * charge;
07410            mqxx += (1.5*mir*mir - 0.5*mr2) * charge;
07411            mqyx += 1.5*mjr*mir * charge;
07412            mqyy += (1.5*mjr*mjr - 0.5*mr2) * charge;
07413            mqzx += 1.5*mkr*mir * charge;
07414            mqzy += 1.5*mkr*mjr * charge;
07415            mqzz += (1.5*mkr*mkr - 0.5*mr2) * charge;
07416            */
07417     }
07418 }
07419 }
07420 } /* endif (on the mesh) */
07421
07422 /* print out the Grid vs. Ideal Point Multipole. */
07423
07424 /*
07425 debye = 4.8033324;
07426 mc = mc/f;
07427 mux = mux/f*debye;
07428 muy = muy/f*debye;
07429 muz = muz/f*debye;
07430 mqxx = mqxx/f*debye;
07431 mqyy = mqyy/f*debye;
07432 mqzz = mqzz/f*debye;
07433 mqyx = mqyx/f*debye;
07434 mqzx = mqzx/f*debye;
07435 mqzy = mqzy/f*debye;
07436
07437 printf(" Grid v. Actual Permanent Multipole for Site %i\n",iatom);
07438 printf(" G: %10.6f\n",mc);
07439 printf(" A: %10.6f\n\n",c/f);
07440 printf(" G: %10.6f %10.6f %10.6f\n",mux,muy,muz);
07441 printf(" A: %10.6f %10.6f %10.6f\n\n",
07442        (ux * hx / f) * debye,
07443        (uy * hy / f) * debye,
07444        (uz * hzed / f) * debye);
07445 printf(" G: %10.6f\n",mqxx);
07446 printf(" A: %10.6f\n",quad[0]*debye);
07447 printf(" G: %10.6f %10.6f\n",mqyx,mqyy);
07448 printf(" A: %10.6f %10.6f\n",quad[3]*debye,quad[4]*debye);
07449 printf(" G: %10.6f %10.6f %10.6f\n",mqzx,mqzy,mqzz);
07450 printf(" A: %10.6f %10.6f %10.6f\n\n",
07451        quad[6]*debye,quad[7]*debye,quad[8]*debye); */
07452
07453 } /* endfor (each atom) */
07454 }
07455
07456 #if defined(WITH_TINKER)
07457
07458 VPUBLIC void fillcoInducedDipole(Vpmg *thee) {
07459
07460     Valist *alist;
07461     Vpbe *pbe;
07462     Vatom *atom;
07463     /* Conversions */
07464     double zmagic, f;
07465     /* Grid */
07466     double xmin, xmax, ymin, ymax, zmin, zmax;
07467     double xlen, ylen, zlen, ifloat, jfloat, kfloat;
07468     double hx, hy, hzed, *apos, position[3];
07469     /* B-spline weights */
07470     double mx, my, mz, dmx, dmy, dmz;
07471     /* Dipole */
07472     double charge, *dipole, ux,uy,uz;
07473     double mi,mj,mk;
07474     /* Loop indeces */
07475     int i, ii, jj, kk, nx, ny, nz, iatom;
07476     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;

```

```

07477
07478     double debye;
07479     double mux,muy,muz;
07480     double mir,mjr,mkr;
07481
07482     VASSERT(thee != VNULL);
07483
07484     /* Get PBE info */
07485     pbe = thee->pbe;
07486     alist = pbe->alist;
07487     zmagic = Vpbe_getZmagic(pbe);
07488
07489     /* Mesh info */
07490     nx = thee->pmgp->nx;
07491     ny = thee->pmgp->ny;
07492     nz = thee->pmgp->nz;
07493     hx = thee->pmgp->hx;
07494     hy = thee->pmgp->hy;
07495     hzed = thee->pmgp->hzed;
07496
07497     /* Conversion */
07498     f = zmagic/(hx*hy*hzed);
07499
07500     /* Define the total domain size */
07501     xlen = thee->pmgp->xlen;
07502     ylen = thee->pmgp->ylen;
07503     zlen = thee->pmgp->zlen;
07504
07505     /* Define the min/max dimensions */
07506     xmin = thee->pmgp->xcent - (xlen/2.0);
07507     ymin = thee->pmgp->ycent - (ylen/2.0);
07508     zmin = thee->pmgp->zcent - (zlen/2.0);
07509     xmax = thee->pmgp->xcent + (xlen/2.0);
07510     ymax = thee->pmgp->ycent + (ylen/2.0);
07511     zmax = thee->pmgp->zcent + (zlen/2.0);
07512
07513     /* Fill in the source term (induced dipoles) */
07514     Vnm_print(0, "fillcoInducedDipole: filling in the source term.\n");
07515     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
07516
07517         atom = Valist_getAtom(alist, iatom);
07518         apos = Vatom_getPosition(atom);
07519
07520         dipole = Vatom_getInducedDipole(atom);
07521         ux = dipole[0]/hx*f;
07522         uy = dipole[1]/hy*f;
07523         uz = dipole[2]/hzed*f;
07524
07525         mux = 0.0;
07526         muy = 0.0;
07527         muz = 0.0;
07528
07529         /* Make sure we're on the grid */
07530         if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx)) || \
07531             (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy)) || \
07532             (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
07533             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]);
07534             Vnm_print(2, "fillcoInducedDipole: xmin = %g, xmax = %g\n", xmin, xmax);
07535             Vnm_print(2, "fillcoInducedDipole: ymin = %g, ymax = %g\n", ymin, ymax);
07536             Vnm_print(2, "fillcoInducedDipole: zmin = %g, zmax = %g\n", zmin, zmax);
07537             fflush(stderr);
07538         } else {
07539
07540             /* Convert the atom position to grid reference frame */
07541             position[0] = apos[0] - xmin;
07542             position[1] = apos[1] - ymin;
07543             position[2] = apos[2] - zmin;
07544
07545             /* Figure out which vertices we're next to */
07546             ifloat = position[0]/hx;
07547             jfloat = position[1]/hy;
07548             kfloat = position[2]/hzed;
07549
07550             ip1 = (int)ceil(ifloat);
07551             ip2 = ip1 + 2;
07552             im1 = (int)floor(ifloat);
07553             im2 = im1 - 2;
07554             jp1 = (int)ceil(jfloat);
07555             jp2 = jp1 + 2;
07556             jml = (int)floor(jfloat);

```

```

07557         jm2 = jm1 - 2;
07558         kp1 = (int)ceil(kfloat);
07559         kp2 = kp1 + 2;
07560         km1 = (int)floor(kfloat);
07561         km2 = km1 - 2;
07562
07563         /* This step shouldn't be necessary, but it saves nasty debugging
07564          * later on if something goes wrong */
07565         ip2 = VMIN2(ip2,nx-1);
07566         ip1 = VMIN2(ip1,nx-1);
07567         im1 = VMAX2(im1,0);
07568         im2 = VMAX2(im2,0);
07569         jp2 = VMIN2(jp2,ny-1);
07570         jp1 = VMIN2(jp1,ny-1);
07571         jm1 = VMAX2(jm1,0);
07572         jm2 = VMAX2(jm2,0);
07573         kp2 = VMIN2(kp2,nz-1);
07574         kp1 = VMIN2(kp1,nz-1);
07575         km1 = VMAX2(km1,0);
07576         km2 = VMAX2(km2,0);
07577
07578         /* Now assign fractions of the dipole to the nearby verts */
07579         for (ii=im2; ii<=ip2; ii++) {
07580             mi = VFCHI4(ii,ifloat);
07581             mx = bspline4(mi);
07582             dmx = dbspline4(mi);
07583             for (jj=jm2; jj<=jp2; jj++) {
07584                 mj = VFCHI4(jj,jfloat);
07585                 my = bspline4(mj);
07586                 dmy = dbspline4(mj);
07587                 for (kk=km2; kk<=kp2; kk++) {
07588                     mk = VFCHI4(kk,kfloat);
07589                     mz = bspline4(mk);
07590                     dmz = dbspline4(mk);
07591                     charge = -dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz;
07592                     thee->charge[IJK(ii,jj,kk)] += charge;
07593
07594                     /*
07595                     mir = (mi - 2.5) * hx;
07596                     mjr = (mj - 2.5) * hy;
07597                     mkr = (mk - 2.5) * hzed;
07598                     mux += mir * charge;
07599                     muy += mjr * charge;
07600                     muz += mkr * charge;
07601                     */
07602                 }
07603             }
07604         }
07605     } /* endif (on the mesh) */
07606
07607     /* check
07608     debye = 4.8033324;
07609     mux = mux/f*debye;
07610     muy = muy/f*debye;
07611     muz = muz/f*debye;
07612
07613     printf(" Grid v. Actual Induced Dipole for Site %i\n",iatom);
07614     printf(" G: %10.6f %10.6f %10.6f\n",mux,muy,muz);
07615     printf(" A: %10.6f %10.6f %10.6f\n\n",
07616            (ux * hx / f) * debye,
07617            (uy * hy / f) * debye,
07618            (uz * hzed / f) * debye);
07619     */
07620
07621 } /* endfor (each atom) */
07622 }
07623
07624 VPUBLIC void fillcoNLInducedDipole(Vpmg *thee) {
07625     Valist *alist;
07626     Vpbe *pbe;
07627     Vatom *atom;
07628     /* Conversions */
07629     double zmagic, f;
07630     /* Grid */
07631     double xmin, xmax, ymin, ymax, zmin, zmax;
07632     double xlen, ylen, zlen, ifloat, jfloat, kfloat;
07633     double hx, hy, hzed, *apos, position[3];
07634     /* B-spline weights */
07635     double mx, my, mz, dmx, dmy, dmz;
07636     /* Dipole */

```

```

07638     double charge, *dipole, ux,uy,uz;
07639     double mi,mj,mk;
07640     /* Loop indeces */
07641     int i, ii, jj, kk, nx, ny, nz, iatom;
07642     int im2, im1, ip1, ip2, jm2, jm1, jpl, jp2, km2, km1, kpl, kp2;
07643
07644     /* sanity check
07645     double debye;
07646     double mux,muy,muz;
07647     double mir,mjr,mkr;
07648     */
07649
07650     VASSERT(thee != VNULL);
07651
07652     /* Get PBE info */
07653     pbe = thee->pbe;
07654     alist = pbe->alist;
07655     zmagic = Vpbe_getZmagic(pbe);
07656
07657     /* Mesh info */
07658     nx = thee->pmgp->nx;
07659     ny = thee->pmgp->ny;
07660     nz = thee->pmgp->nz;
07661     hx = thee->pmgp->hx;
07662     hy = thee->pmgp->hy;
07663     hzed = thee->pmgp->hzed;
07664
07665     /* Conversion */
07666     f = zmagic/(hx*hy*hzed);
07667
07668     /* Define the total domain size */
07669     xlen = thee->pmgp->xlen;
07670     ylen = thee->pmgp->ylen;
07671     zlen = thee->pmgp->zlen;
07672
07673     /* Define the min/max dimensions */
07674     xmin = thee->pmgp->xcent - (xlen/2.0);
07675     ymin = thee->pmgp->ycent - (ylen/2.0);
07676     zmin = thee->pmgp->zcent - (zlen/2.0);
07677     xmax = thee->pmgp->xcent + (xlen/2.0);
07678     ymax = thee->pmgp->ycent + (ylen/2.0);
07679     zmax = thee->pmgp->zcent + (zlen/2.0);
07680
07681     /* Fill in the source term (non-local induced dipoles) */
07682     Vnm_print(0, "fillcoNLInducedDipole: filling in source term.\n");
07683     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
07684
07685         atom = Valist_getAtom(alist, iatom);
07686         apos = Vatom_getPosition(atom);
07687
07688         dipole = Vatom_getNLInducedDipole(atom);
07689         ux = dipole[0]/hx*f;
07690         uy = dipole[1]/hy*f;
07691         uz = dipole[2]/hzed*f;
07692
07693         /*
07694         mux = 0.0;
07695         muy = 0.0;
07696         muz = 0.0;
07697         */
07698
07699         /* Make sure we're on the grid */
07700         if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx)) || \
07701             (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy)) || \
07702             (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
07703             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]);
07704             Vnm_print(2, "fillcoNLInducedDipole: xmin = %g, xmax = %g\n", xmin, xmax);
07705             Vnm_print(2, "fillcoNLInducedDipole: ymin = %g, ymax = %g\n", ymin, ymax);
07706             Vnm_print(2, "fillcoNLInducedDipole: zmin = %g, zmax = %g\n", zmin, zmax);
07707             fflush(stderr);
07708         } else {
07709
07710             /* Convert the atom position to grid reference frame */
07711             position[0] = apos[0] - xmin;
07712             position[1] = apos[1] - ymin;
07713             position[2] = apos[2] - zmin;
07714
07715             /* Figure out which vertices we're next to */
07716             ifloat = position[0]/hx;
07717             jfloat = position[1]/hy;

```

```

07718         kfloat = position[2]/hzd;
07719
07720         ip1 = (int)ceil(ifloat);
07721         ip2 = ip1 + 2;
07722         im1 = (int)floor(ifloat);
07723         im2 = im1 - 2;
07724         jpl = (int)ceil(jfloat);
07725         jp2 = jpl + 2;
07726         jml = (int)floor(jfloat);
07727         jm2 = jml - 2;
07728         kpl = (int)ceil(kfloat);
07729         kp2 = kpl + 2;
07730         kml = (int)floor(kfloat);
07731         km2 = kml - 2;
07732
07733         /* This step shouldn't be necessary, but it saves nasty debugging
07734          * later on if something goes wrong */
07735         ip2 = VMIN2(ip2,nx-1);
07736         ip1 = VMIN2(ip1,nx-1);
07737         im1 = VMAX2(im1,0);
07738         im2 = VMAX2(im2,0);
07739         jp2 = VMIN2(jp2,ny-1);
07740         jpl = VMIN2(jpl,ny-1);
07741         jml = VMAX2(jml,0);
07742         jm2 = VMAX2(jm2,0);
07743         kp2 = VMIN2(kp2,nz-1);
07744         kpl = VMIN2(kpl,nz-1);
07745         kml = VMAX2(kml,0);
07746         km2 = VMAX2(km2,0);
07747
07748         /* Now assign fractions of the non local induced dipole
07749          * to the nearby verts */
07750         for (ii=im2; ii<=ip2; ii++) {
07751             mi = VFCHI4(ii,ifloat);
07752             mx = bspline4(mi);
07753             dmx = dbspline4(mi);
07754             for (jj=jm2; jj<=jp2; jj++) {
07755                 mj = VFCHI4(jj,jfloat);
07756                 my = bspline4(mj);
07757                 dmy = dbspline4(mj);
07758                 for (kk=km2; kk<=kp2; kk++) {
07759                     mk = VFCHI4(kk,kfloat);
07760                     mz = bspline4(mk);
07761                     dmz = dbspline4(mk);
07762                     charge = -dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz;
07763                     thee->charge[IJK(ii,jj,kk)] += charge;
07764
07765                     /*
07766                     mir = (mi - 2.5) * hx;
07767                     mjr = (mj - 2.5) * hy;
07768                     mkr = (mk - 2.5) * hzed;
07769                     mux += mir * charge;
07770                     muy += mjr * charge;
07771                     muz += mkr * charge;
07772                     */
07773                 }
07774             }
07775         }
07776     } /* endif (on the mesh) */
07777
07778     /*
07779     debye = 4.8033324;
07780     mux = mux/f*debye;
07781     muy = muy/f*debye;
07782     muz = muz/f*debye;
07783
07784     printf(" Grid v. Actual Non-Local Induced Dipole for Site %i\n",iatom);
07785     printf(" G: %10.6f %10.6f %10.6f\n",mux,muy,muz);
07786     printf(" A: %10.6f %10.6f %10.6f\n",
07787            (ux * hx / f) * debye,
07788            (uy * hy / f) * debye,
07789            (uz * hzed / f) * debye); */
07790
07791     } /* endfor (each atom) */
07792 }
07793
07794 VPUBLIC double Vpmg_qfPermanentMultipoleEnergy(
07795     Vpmg *thee, int atomID) {
07796     double *u;
07797     Vatom *atom;

```

```

07798      /* Grid variables */
07799      int nx, ny, nz;
07800      double xmax, xmin, ymax, ymin, zmax, zmin;
07801      double hx, hy, hzed, ifloat, jfloat, kfloat;
07802      double mi, mj, mk;
07803      double *position;
07804      /* B-spline weights */
07805      double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz;
07806      /* Loop indeces */
07807      int ip1, ip2, im1, im2, jp1, jp2, jm1, jm2, kp1, kp2, km1, km2;
07808      int i, j, ii, jj, kk;
07809      /* Potential, field, field gradient and multipole components */
07810      double pot, rfe[3], rfde[3][3], energy;
07811      double f, charge, *dipole, *quad;
07812      double qxx, qyx, qyy, qzx, qzy, qzz;
07813
07814
07815      VASSERT(thee != VNULL);
07816      VASSERT(thee->filled);
07817
07818      /* Get the mesh information */
07819      nx = thee->pmgp->nx;
07820      ny = thee->pmgp->ny;
07821      nz = thee->pmgp->nz;
07822      hx = thee->pmgp->hx;
07823      hy = thee->pmgp->hy;
07824      hzed = thee->pmgp->hzed;
07825      xmax = thee->xf[nx-1];
07826      ymax = thee->yf[ny-1];
07827      zmax = thee->zf[nz-1];
07828      xmin = thee->xf[0];
07829      ymin = thee->yf[0];
07830      zmin = thee->zf[0];
07831
07832      u = thee->u;
07833
07834      atom = Valist_getAtom(thee->pbe->alist, atomID);
07835
07836      /* Currently all atoms must be in the same partition. */
07837
07838      VASSERT(atom->partID != 0);
07839
07840      /* Convert the atom position to grid coordinates */
07841
07842      position = Vatom_getPosition(atom);
07843      ifloat = (position[0] - xmin)/hx;
07844      jfloat = (position[1] - ymin)/hy;
07845      kfloat = (position[2] - zmin)/hzed;
07846      ip1 = (int)ceil(ifloat);
07847      ip2 = ip1 + 2;
07848      im1 = (int)floor(ifloat);
07849      im2 = im1 - 2;
07850      jp1 = (int)ceil(jfloat);
07851      jp2 = jp1 + 2;
07852      jm1 = (int)floor(jfloat);
07853      jm2 = jm1 - 2;
07854      kp1 = (int)ceil(kfloat);
07855      kp2 = kp1 + 2;
07856      km1 = (int)floor(kfloat);
07857      km2 = km1 - 2;
07858
07859      /* This step shouldn't be necessary, but it saves nasty debugging
07860       * later on if something goes wrong */
07861      ip2 = VMIN2(ip2, nx-1);
07862      ip1 = VMIN2(ip1, nx-1);
07863      im1 = VMAX2(im1, 0);
07864      im2 = VMAX2(im2, 0);
07865      jp2 = VMIN2(jp2, ny-1);
07866      jp1 = VMIN2(jp1, ny-1);
07867      jm1 = VMAX2(jm1, 0);
07868      jm2 = VMAX2(jm2, 0);
07869      kp2 = VMIN2(kp2, nz-1);
07870      kp1 = VMIN2(kp1, nz-1);
07871      km1 = VMAX2(km1, 0);
07872      km2 = VMAX2(km2, 0);
07873
07874      /* Initialize observables to zero */
07875      energy = 0.0;
07876      pot = 0.0;
07877      for (i=0; i<3; i++){
07878          rfe[i] = 0.0;

```

```

07879         for (j=0;j<3;j++){
07880             rfde[i][j] = 0.0;
07881         }
07882     }
07883
07884     for (ii=im2; ii<=ip2; ii++) {
07885         mi = VFCHI4(ii,ifloat);
07886         mx = bspline4(mi);
07887         dmx = dbspline4(mi);
07888         d2mx = d2bspline4(mi);
07889         for (jj=jm2; jj<=jp2; jj++) {
07890             mj = VFCHI4(jj,jfloat);
07891             my = bspline4(mj);
07892             dmy = dbspline4(mj);
07893             d2my = d2bspline4(mj);
07894             for (kk=km2; kk<=kp2; kk++) {
07895                 mk = VFCHI4(kk,kfloat);
07896                 mz = bspline4(mk);
07897                 dmz = dbspline4(mk);
07898                 d2mz = d2bspline4(mk);
07899                 f = u[IJK(ii,jj,kk)];
07900                 /* potential */
07901                 pot += f*mx*my*mz;
07902                 /* field */
07903                 rfe[0] += f*dmx*my*mz/hx;
07904                 rfe[1] += f*mx*dmy*mz/hy;
07905                 rfe[2] += f*mx*my*dmz/hzed;
07906                 /* field gradient */
07907                 rfde[0][0] += f*d2mx*my*mz/(hx*hx);
07908                 rfde[1][0] += f*dmx*dmy*mz/(hy*hx);
07909                 rfde[1][1] += f*mx*d2my*mz/(hy*hy);
07910                 rfde[2][0] += f*dmx*my*dmz/(hx*hzed);
07911                 rfde[2][1] += f*mx*dmy*dmz/(hy*hzed);
07912                 rfde[2][2] += f*mx*my*d2mz/(hzed*hzed);
07913             }
07914         }
07915     }
07916
07917     charge = Vatom_getCharge(atom);
07918     dipole = Vatom_getDipole(atom);
07919     quad = Vatom_getQuadrupole(atom);
07920     qxx = quad[0]/3.0;
07921     qyx = quad[3]/3.0;
07922     qyy = quad[4]/3.0;
07923     qzx = quad[6]/3.0;
07924     qzy = quad[7]/3.0;
07925     qzz = quad[8]/3.0;
07926
07927     energy = pot * charge
07928             - rfe[0] * dipole[0]
07929             - rfe[1] * dipole[1]
07930             - rfe[2] * dipole[2]
07931             + rfde[0][0]*qxx
07932             + 2.0*rfde[1][0]*qyx + rfde[1][1]*qyy
07933             + 2.0*rfde[2][0]*qzx + 2.0*rfde[2][1]*qzy + rfde[2][2]*qzz;
07934
07935     return energy;
07936 }
07937
07938 VPUBLIC void Vpmg_fieldSpline4(Vpmg *thee, int atomID, double field[3]) {
07939
07940     Vatom *atom;
07941     double *u, f;
07942     /* Grid variables */
07943     int nx, ny, nz;
07944     double xmax, xmin, ymax, ymin, zmax, zmin;
07945     double hx, hy, hzed, ifloat, jfloat, kfloat;
07946     double *apos, position[3];
07947     /* B-Spline weights */
07948     double mx, my, mz, dmx, dmy, dmz;
07949     double mi, mj, mk;
07950     /* Loop indeces */
07951     int ip1,ip2,im1,im2,jp1,jp2,jm1,jm2,kp1,kp2,km1,km2;
07952     int i,j,ii,jj,kk;
07953
07954
07955     VASSERT (thee != VNULL);
07956
07957     /* Get the mesh information */
07958     nx = thee->pmgp->nx;
07959     ny = thee->pmgp->ny;

```



```

07960     nz = thee->pmgp->nz;
07961     hx = thee->pmgp->hx;
07962     hy = thee->pmgp->hy;
07963     hzed = thee->pmgp->hzed;
07964     xmax = thee->xf[nx-1];
07965     ymax = thee->yf[ny-1];
07966     zmax = thee->zf[nz-1];
07967     xmin = thee->xf[0];
07968     ymin = thee->yf[0];
07969     zmin = thee->zf[0];
07970
07971     u = thee->u;
07972
07973     atom = Valist_getAtom(thee->pbe->alist, atomID);
07974
07975     /* Currently all atoms must be in the same partition. */
07976
07977     VASSERT (atom->partID != 0);
07978
07979     /* Convert the atom position to grid coordinates */
07980
07981     apos = Vatom_getPosition(atom);
07982     position[0] = apos[0] - xmin;
07983     position[1] = apos[1] - ymin;
07984     position[2] = apos[2] - zmin;
07985     ifloat = position[0]/hx;
07986     jfloat = position[1]/hy;
07987     kfloat = position[2]/hzed;
07988     ip1 = (int)ceil(ifloat);
07989     ip2 = ip1 + 2;
07990     im1 = (int)floor(ifloat);
07991     im2 = im1 - 2;
07992     jp1 = (int)ceil(jfloat);
07993     jp2 = jp1 + 2;
07994     jm1 = (int)floor(jfloat);
07995     jm2 = jm1 - 2;
07996     kp1 = (int)ceil(kfloat);
07997     kp2 = kp1 + 2;
07998     km1 = (int)floor(kfloat);
07999     km2 = km1 - 2;
08000
08001     /* This step shouldn't be necessary, but it saves nasty debugging
08002      * later on if something goes wrong */
08003     ip2 = VMIN2(ip2, nx-1);
08004     ip1 = VMIN2(ip1, nx-1);
08005     im1 = VMAX2(im1, 0);
08006     im2 = VMAX2(im2, 0);
08007     jp2 = VMIN2(jp2, ny-1);
08008     jp1 = VMIN2(jp1, ny-1);
08009     jm1 = VMAX2(jm1, 0);
08010     jm2 = VMAX2(jm2, 0);
08011     kp2 = VMIN2(kp2, nz-1);
08012     kp1 = VMIN2(kp1, nz-1);
08013     km1 = VMAX2(km1, 0);
08014     km2 = VMAX2(km2, 0);
08015
08016     for (i=0; i<3; i++){
08017         field[i] = 0.0;
08018     }
08019
08020     for (ii=im2; ii<=ip2; ii++) {
08021         mi = VFCHI4(ii, ifloat);
08022         mx = bspline4(mi);
08023         dmx = dbspline4(mi);
08024         for (jj=jm2; jj<=jp2; jj++) {
08025             mj = VFCHI4(jj, jfloat);
08026             my = bspline4(mj);
08027             dmy = dbspline4(mj);
08028             for (kk=km2; kk<=kp2; kk++) {
08029                 mk = VFCHI4(kk, kfloat);
08030                 mz = bspline4(mk);
08031                 dmz = dbspline4(mk);
08032                 f = u[IJK(ii, jj, kk)];
08033
08034                 field[0] += f*dmx*my*mz/hx;
08035                 field[1] += f*mx*dmy*mz/hy;
08036                 field[2] += f*mx*my*dmz/hzed;
08037             }
08038         }
08039     }
08040 }

```

```

08041
08042 VPUBLIC void Vpmg_qfPermanentMultipoleForce(Vpmg *thee, int atomID,
08043                                             double force[3], double torque[3]) {
08044
08045     Vatom *atom;
08046     double f, *u, *apos, position[3];
08047
08048     /* Grid variables */
08049     int nx,ny,nz;
08050     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
08051     double hx, hy, hzed, ifloat, jfloat, kfloat;
08052
08053     /* B-spline weights */
08054     double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz, d3mx, d3my, d3mz;
08055     double mi, mj, mk;
08056
08057     /* Loop indeces */
08058     int i, j, k, ii, jj, kk;
08059     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
08060
08061     /* Potential, field, field gradient and 2nd field gradient */
08062     double pot, e[3], de[3][3], d2e[3][3][3];
08063
08064     /* Permanent multipole components */
08065     double *dipole, *quad;
08066     double c, ux, uy, uz, qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
08067
08068     VASSERT(thee != VNULL);
08069     VASSERT(thee->filled);
08070
08071     atom = Valist_getAtom(thee->pbe->alist, atomID);
08072
08073     /* Currently all atoms must be in the same partition. */
08074
08075     VASSERT(atom->partID != 0);
08076
08077     apos = Vatom_getPosition(atom);
08078
08079     c = Vatom_getCharge(atom);
08080     dipole = Vatom_getDipole(atom);
08081     ux = dipole[0];
08082     uy = dipole[1];
08083     uz = dipole[2];
08084     quad = Vatom_getQuadrupole(atom);
08085     qxx = quad[0]/3.0;
08086     qxy = quad[1]/3.0;
08087     qxz = quad[2]/3.0;
08088     qyx = quad[3]/3.0;
08089     qyy = quad[4]/3.0;
08090     qyz = quad[5]/3.0;
08091     qzx = quad[6]/3.0;
08092     qzy = quad[7]/3.0;
08093     qzz = quad[8]/3.0;
08094
08095     /* Initialize observables */
08096     pot = 0.0;
08097     for (i=0; i<3; i++){
08098         e[i] = 0.0;
08099         for (j=0; j<3; j++){
08100             de[i][j] = 0.0;
08101             for (k=0; k<3; k++){
08102                 d2e[i][j][k] = 0.0;
08103             }
08104         }
08105     }
08106
08107     /* Mesh info */
08108     nx = thee->pmgp->nx;
08109     ny = thee->pmgp->ny;
08110     nz = thee->pmgp->nz;
08111     hx = thee->pmgp->hx;
08112     hy = thee->pmgp->hy;
08113     hzed = thee->pmgp->hzed;
08114     xlen = thee->pmgp->xlen;
08115     ylen = thee->pmgp->ylen;
08116     zlen = thee->pmgp->zlen;
08117     xmin = thee->pmgp->xmin;
08118     ymin = thee->pmgp->ymin;
08119     zmin = thee->pmgp->zmin;
08120     xmax = thee->pmgp->xmax;
08121     ymax = thee->pmgp->ymax;

```

```

08122     zmax = thee->pmpg->zmax;
08123     u = thee->u;
08124
08125     /* Make sure we're on the grid */
08126     if ((apos[0] <= (xmin+2*hx)) || (apos[0] >= (xmax-2*hx)) \
08127         || (apos[1] <= (ymin+2*hy)) || (apos[1] >= (ymax-2*hy)) \
08128         || (apos[2] <= (zmin+2*hzed)) || (apos[2] >= (zmax-2*hzed))) {
08129         Vnm_print(2, "qfPermanentMultipoleForce: Atom off the mesh (ignoring) %6.3f %6.3f %6.3f\n", apos[0
08130     ], apos[1], apos[2]);
08131         fflush(stderr);
08132     } else {
08133
08134         /* Convert the atom position to grid coordinates */
08135         position[0] = apos[0] - xmin;
08136         position[1] = apos[1] - ymin;
08137         position[2] = apos[2] - zmin;
08138         ifloat = position[0]/hx;
08139         jfloat = position[1]/hy;
08140         kfloat = position[2]/hzed;
08141         ip1 = (int)ceil(ifloat);
08142         ip2 = ip1 + 2;
08143         im1 = (int)floor(ifloat);
08144         im2 = im1 - 2;
08145         jp1 = (int)ceil(jfloat);
08146         jp2 = jp1 + 2;
08147         jm1 = (int)floor(jfloat);
08148         jm2 = jm1 - 2;
08149         kp1 = (int)ceil(kfloat);
08150         kp2 = kp1 + 2;
08151         km1 = (int)floor(kfloat);
08152         km2 = km1 - 2;
08153
08154         /* This step shouldn't be necessary, but it saves nasty debugging
08155          * later on if something goes wrong */
08156         ip2 = VMIN2(ip2, nx-1);
08157         ip1 = VMIN2(ip1, nx-1);
08158         im1 = VMAX2(im1, 0);
08159         im2 = VMAX2(im2, 0);
08160         jp2 = VMIN2(jp2, ny-1);
08161         jp1 = VMIN2(jp1, ny-1);
08162         jm1 = VMAX2(jm1, 0);
08163         jm2 = VMAX2(jm2, 0);
08164         kp2 = VMIN2(kp2, nz-1);
08165         kp1 = VMIN2(kp1, nz-1);
08166         km1 = VMAX2(km1, 0);
08167         km2 = VMAX2(km2, 0);
08168
08169         for (ii=im2; ii<=ip2; ii++) {
08170             mi = VFCHI4(ii, ifloat);
08171             mx = bspline4(mi);
08172             dmx = dbspline4(mi);
08173             d2mx = d2bspline4(mi);
08174             d3mx = d3bspline4(mi);
08175             for (jj=jm2; jj<=jp2; jj++) {
08176                 mj = VFCHI4(jj, jfloat);
08177                 my = bspline4(mj);
08178                 dmy = dbspline4(mj);
08179                 d2my = d2bspline4(mj);
08180                 d3my = d3bspline4(mj);
08181                 for (kk=km2; kk<=kp2; kk++) {
08182                     mk = VFCHI4(kk, kfloat);
08183                     mz = bspline4(mk);
08184                     dmz = dbspline4(mk);
08185                     d2mz = d2bspline4(mk);
08186                     d3mz = d3bspline4(mk);
08187                     f = u[IJK(ii, jj, kk)];
08188                     /* Potential */
08189                     pot += f*mx*my*mz;
08190                     /* Field */
08191                     e[0] += f*dmx*my*mz/hx;
08192                     e[1] += f*mx*dmy*mz/hy;
08193                     e[2] += f*mx*my*dmz/hzed;
08194                     /* Field gradient */
08195                     de[0][0] += f*d2mx*my*mz/(hx*hx);
08196                     de[1][0] += f*dmx*dmy*mz/(hy*hx);
08197                     de[1][1] += f*mx*d2my*mz/(hy*hy);
08198                     de[2][0] += f*dmx*my*dmz/(hx*hzed);
08199                     de[2][1] += f*mx*dmy*dmz/(hy*hzed);
08200                     de[2][2] += f*mx*my*d2mz/(hzed*hzed);
08201                     /* 2nd Field Gradient
08202                     VxVxVa */

```

```

08202         d2e[0][0][0] += f*d3mx*my*mz / (hx*hx*hx);
08203         d2e[0][0][1] += f*d2mx*dmy*mz / (hx*hy*hx);
08204         d2e[0][0][2] += f*d2mx*my*dmz / (hx*hx*hzed);
08205         /* VyVxVa */
08206         d2e[1][0][0] += f*d2mx*dmy*mz / (hx*hx*hy);
08207         d2e[1][0][1] += f*dmx*d2my*mz / (hx*hy*hy);
08208         d2e[1][0][2] += f*dmx*dmy*dmz / (hx*hy*hzed);
08209         /* VyVyVa */
08210         d2e[1][1][0] += f*dmx*d2my*mz / (hx*hy*hy);
08211         d2e[1][1][1] += f*mx*d3my*mz / (hy*hy*hy);
08212         d2e[1][1][2] += f*mx*d2my*dmz / (hy*hy*hzed);
08213         /* VzVxVa */
08214         d2e[2][0][0] += f*d2mx*my*dmz / (hx*hx*hzed);
08215         d2e[2][0][1] += f*dmx*dmy*dmz / (hx*hy*hzed);
08216         d2e[2][0][2] += f*dmx*my*d2mz / (hx*hzed*hzed);
08217         /* VzVyVa */
08218         d2e[2][1][0] += f*dmx*dmy*dmz / (hx*hy*hzed);
08219         d2e[2][1][1] += f*mx*d2my*dmz / (hy*hy*hzed);
08220         d2e[2][1][2] += f*mx*dmy*d2mz / (hy*hzed*hzed);
08221         /* VzVzVa */
08222         d2e[2][2][0] += f*dmx*my*d2mz / (hx*hzed*hzed);
08223         d2e[2][2][1] += f*mx*dmy*d2mz / (hy*hzed*hzed);
08224         d2e[2][2][2] += f*mx*my*d3mz / (hzed*hzed*hzed);
08225     }
08226 }
08227 }
08228 }
08229
08230 /* Monopole Force */
08231 force[0] = e[0]*c;
08232 force[1] = e[1]*c;
08233 force[2] = e[2]*c;
08234
08235 /* Dipole Force */
08236 force[0] -= de[0][0]*ux+de[1][0]*uy+de[2][0]*uz;
08237 force[1] -= de[1][0]*ux+de[1][1]*uy+de[2][1]*uz;
08238 force[2] -= de[2][0]*ux+de[2][1]*uy+de[2][2]*uz;
08239
08240 /* Quadrupole Force */
08241 force[0] += d2e[0][0][0]*qxx
08242           + d2e[1][0][0]*qyx*2.0+d2e[1][1][0]*qyy
08243           + d2e[2][0][0]*qzx*2.0+d2e[2][1][0]*qzy*2.0+d2e[2][2][0]*qzz;
08244 force[1] += d2e[0][0][1]*qxx
08245           + d2e[1][0][1]*qyx*2.0+d2e[1][1][1]*qyy
08246           + d2e[2][0][1]*qzx*2.0+d2e[2][1][1]*qzy*2.0+d2e[2][2][1]*qzz;
08247 force[2] += d2e[0][0][2]*qxx
08248           + d2e[1][0][2]*qyx*2.0+d2e[1][1][2]*qyy
08249           + d2e[2][0][2]*qzx*2.0+d2e[2][1][2]*qzy*2.0+d2e[2][2][2]*qzz;
08250
08251 /* Dipole Torque */
08252 torque[0] = uy * e[2] - uz * e[1];
08253 torque[1] = uz * e[0] - ux * e[2];
08254 torque[2] = ux * e[1] - uy * e[0];
08255 /* Quadrupole Torque */
08256 de[0][1] = de[1][0];
08257 de[0][2] = de[2][0];
08258 de[1][2] = de[2][1];
08259 torque[0] -= 2.0*(qyx*de[0][2] + qyy*de[1][2] + qyz*de[2][2]
08260               - qzx*de[0][1] - qzy*de[1][1] - qzz*de[2][1]);
08261 torque[1] -= 2.0*(qzx*de[0][0] + qzy*de[1][0] + qzz*de[2][0]
08262               - qxx*de[0][2] - qxy*de[1][2] - qxz*de[2][2]);
08263 torque[2] -= 2.0*(qxx*de[0][1] + qxy*de[1][1] + qxz*de[2][1]
08264               - qyx*de[0][0] - qyy*de[1][0] - qyz*de[2][0]);
08265
08266
08267 /* printf(" qPhi Force %f %f %f\n", force[0], force[1], force[2]);
08268          printf(" qPhi Torque %f %f %f\n", torque[0], torque[1], torque[2]); */
08269 }
08270
08271 VPUBLIC void Vpmg_ibPermanentMultipoleForce(Vpmg *thee, int atomID,
08272                                             double force[3]) {
08273
08274     Valist *alist;
08275     Vacc *acc;
08276     Vpbe *pbe;
08277     Vatom *atom;
08278     Vsurf_Meth srfm;
08279
08280     /* Grid variables */
08281     double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
08282     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;

```

```

08283     double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
08284     double izmagic;
08285     int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
08286
08287     VASSERT(thee != VNULL);
08288
08289     /* Nonlinear PBE is not implemented for AMOEBA */
08290     VASSERT(!thee->pmgp->nonlin);
08291
08292     acc = thee->pbe->acc;
08293     srfrm = thee->surfMeth;
08294     atom = Valist_getAtom(thee->pbe->alist, atomID);
08295
08296     /* Currently all atoms must be in the same partition. */
08297
08298     VASSERT(atom->partID != 0);
08299     apos = Vatom_getPosition(atom);
08300     arad = Vatom_getRadius(atom);
08301
08302     /* Reset force */
08303     force[0] = 0.0;
08304     force[1] = 0.0;
08305     force[2] = 0.0;
08306
08307     /* Get PBE info */
08308     pbe = thee->pbe;
08309     acc = pbe->acc;
08310     alist = pbe->alist;
08311     irad = Vpbe_getMaxIonRadius(pbe);
08312     zkappa2 = Vpbe_getZkappa2(pbe);
08313     izmagic = 1.0/Vpbe_getZmagic(pbe);
08314
08315     /* Should be a check for this further up. */
08316     VASSERT(zkappa2 > VPMGSMALL);
08317
08318     /* Mesh info */
08319     nx = thee->pmgp->nx;
08320     ny = thee->pmgp->ny;
08321     nz = thee->pmgp->nz;
08322     hx = thee->pmgp->hx;
08323     hy = thee->pmgp->hy;
08324     hzed = thee->pmgp->hzed;
08325     xlen = thee->pmgp->xlen;
08326     ylen = thee->pmgp->ylen;
08327     zlen = thee->pmgp->zlen;
08328     xmin = thee->pmgp->xmin;
08329     ymin = thee->pmgp->ymin;
08330     zmin = thee->pmgp->zmin;
08331     xmax = thee->pmgp->xmax;
08332     ymax = thee->pmgp->ymax;
08333     zmax = thee->pmgp->zmax;
08334
08335     /* Make sure we're on the grid */
08336     if ((apos[0] <= xmin) || (apos[0] >= xmax) || \
08337         (apos[1] <= ymin) || (apos[1] >= ymax) || \
08338         (apos[2] <= zmin) || (apos[2] >= zmax)) {
08339         Vnm_print(2, "ibPermanentMultipoleForce: Atom %d at (%4.3f, %4.3f, %4.3f) is off the mesh\n", atomID, apos[0], apos[1], apos[2]);
08340         Vnm_print(2, "ibPermanentMultipoleForce: xmin = %g, xmax = %g\n", xmin, xmax);
08341         Vnm_print(2, "ibPermanentMultipoleForce: ymin = %g, ymax = %g\n", ymin, ymax);
08342         Vnm_print(2, "ibPermanentMultipoleForce: zmin = %g, zmax = %g\n", zmin, zmax);
08343         fflush(stderr);
08344     } else {
08345
08346         /* Convert the atom position to grid reference frame */
08347         position[0] = apos[0] - xmin;
08348         position[1] = apos[1] - ymin;
08349         position[2] = apos[2] - zmin;
08350
08351         /* Integrate over points within this atom's (inflated) radius */
08352         rtot = (irad + arad + thee->splineWin);
08353         rtot2 = VSQR(rtot);
08354         dx = rtot + 0.5*hx;
08355         imin = VMAX2(0, (int)ceil((position[0] - dx)/hx));
08356         imax = VMIN2(nx-1, (int)floor((position[0] + dx)/hx));
08357         for (i=imin; i<=imax; i++) {
08358             dx2 = VSQR(position[0] - hx*i);
08359             if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
08360             else dy = 0.5*hy;
08361             jmin = VMAX2(0, (int)ceil((position[1] - dy)/hy));
08362             jmax = VMIN2(ny-1, (int)floor((position[1] + dy)/hy));

```

```

08363         for (j=jmin; j<=jmax; j++) {
08364             dy2 = VSQR(position[1] - hy*j);
08365             if (rtot2 > (dx2+dy2)) dz = VSQR(rtot2-dx2-dy2)+0.5*hzed;
08366             else dz = 0.5*hzed;
08367             kmin = VMAX2(0, (int)ceil((position[2] - dz)/hzed));
08368             kmax = VMIN2(nz-1, (int)floor((position[2] + dz)/hzed));
08369             for (k=kmin; k<=kmax; k++) {
08370                 dz2 = VSQR(k*hzed - position[2]);
08371                 /* See if grid point is inside ivdw radius and set ccf
08372                  * accordingly (do spline assignment here) */
08373                 if ((dz2 + dy2 + dx2) <= rtot2) {
08374                     gpos[0] = i*hx + xmin;
08375                     gpos[1] = j*hy + ymin;
08376                     gpos[2] = k*hzed + zmin;
08377                     Vpmg_splineSelect(srffm, acc, gpos, thee->
splineWin, irad, atom, tgrad);
08378                     fmag = VSQR(thee->u[IJK(i,j,k)])*thee->kappa[IJK(i,j,k)];
08379                     force[0] += (zkappa2*fmag*tgrad[0]);
08380                     force[1] += (zkappa2*fmag*tgrad[1]);
08381                     force[2] += (zkappa2*fmag*tgrad[2]);
08382                 }
08383             } /* k loop */
08384         } /* j loop */
08385     } /* i loop */
08386 }
08387
08388 force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
08389 force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
08390 force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
08391
08392 }
08393
08394 VPUBLIC void Vpmg_dbPermanentMultipoleForce(Vpmg *thee, int atomID,
08395 double force[3]) {
08396
08397     Vacc *acc;
08398     Vpbe *pbe;
08399     Vatom *atom;
08400     Vsurf_Meth srffm;
08401
08402     double *apos, position[3], arad, hx, hy, hzed, izmagic, deps, depsi;
08403     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2, epsp;
08404     double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
08405     double *u, Hxijk, Hyijk, Hzijk, Hximljk, Hyijmlk, Hzijkm1;
08406     double dHxijk[3], dHyijk[3], dHzijk[3], dHximljk[3], dHyijmlk[3];
08407     double dHzijkm1[3];
08408     int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
08409
08410     VASSERT(thee != VNULL);
08411
08412     acc = thee->pbe->acc;
08413     srffm = thee->surfMeth;
08414     atom = Valist_getAtom(thee->pbe->alist, atomID);
08415
08416     /* Currently all atoms must be in the same partition. */
08417
08418     VASSERT(atom->partID != 0);
08419     arad = Vatom_getRadius(atom);
08420     apos = Vatom_getPosition(atom);
08421
08422     /* Reset force */
08423     force[0] = 0.0;
08424     force[1] = 0.0;
08425     force[2] = 0.0;
08426
08427     /* Get PBE info */
08428     pbe = thee->pbe;
08429     acc = pbe->acc;
08430     epsp = Vpbe_getSoluteDiel(pbe);
08431     epsw = Vpbe_getSolventDiel(pbe);
08432     kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*
Vunit_kb;
08433     izmagic = 1.0/Vpbe_getZmagic(pbe);
08434
08435
08436     deps = (epsw - epsp);
08437     depsi = 1.0/deps;
08438
08439     VASSERT(VABS(deps) > VPMGSMALL);
08440
08441     /* Mesh info */

```

```

08442     nx = thee->pmgp->nx;
08443     ny = thee->pmgp->ny;
08444     nz = thee->pmgp->nz;
08445     hx = thee->pmgp->hx;
08446     hy = thee->pmgp->hy;
08447     hzed = thee->pmgp->hzed;
08448     xlen = thee->pmgp->xlen;
08449     ylen = thee->pmgp->ylen;
08450     zlen = thee->pmgp->zlen;
08451     xmin = thee->pmgp->xmin;
08452     ymin = thee->pmgp->ymin;
08453     zmin = thee->pmgp->zmin;
08454     xmax = thee->pmgp->xmax;
08455     ymax = thee->pmgp->ymax;
08456     zmax = thee->pmgp->zmax;
08457     u = thee->u;
08458
08459     /* Make sure we're on the grid */
08460     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
08461         (apos[1]<=ymin) || (apos[1]>=ymax) || \
08462         (apos[2]<=zmin) || (apos[2]>=zmax)) {
08463         Vnm_print(2, "dbPermanentMultipoleForce: Atom at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):
08464         \n", apos[0], apos[1], apos[2]);
08465         Vnm_print(2, "dbPermanentMultipoleForce: xmin = %g, xmax = %g\n", xmin, xmax);
08466         Vnm_print(2, "dbPermanentMultipoleForce: ymin = %g, ymax = %g\n", ymin, ymax);
08467         Vnm_print(2, "dbPermanentMultipoleForce: zmin = %g, zmax = %g\n", zmin, zmax);
08468         fflush(stderr);
08469     } else {
08470         /* Convert the atom position to grid reference frame */
08471         position[0] = apos[0] - xmin;
08472         position[1] = apos[1] - ymin;
08473         position[2] = apos[2] - zmin;
08474
08475         /* Integrate over points within this atom's (inflated) radius */
08476         rtot = (arad + thee->splineWin);
08477         rtot2 = VSQR(rtot);
08478         dx = rtot/hx;
08479         imin = (int)floor((position[0]-rtot)/hx);
08480         if (imin < 1) {
08481             Vnm_print(2, "dbPermanentMultipoleForce: Atom off grid!\n");
08482             return;
08483         }
08484         imax = (int)ceil((position[0]+rtot)/hx);
08485         if (imax > (nx-2)) {
08486             Vnm_print(2, "dbPermanentMultipoleForce: Atom off grid!\n");
08487             return;
08488         }
08489         jmin = (int)floor((position[1]-rtot)/hy);
08490         if (jmin < 1) {
08491             Vnm_print(2, "dbPermanentMultipoleForce: Atom off grid!\n");
08492             return;
08493         }
08494         jmax = (int)ceil((position[1]+rtot)/hy);
08495         if (jmax > (ny-2)) {
08496             Vnm_print(2, "dbPermanentMultipoleForce: Atom off grid!\n");
08497             return;
08498         }
08499         kmin = (int)floor((position[2]-rtot)/hzed);
08500         if (kmin < 1) {
08501             Vnm_print(2, "dbPermanentMultipoleForce: Atom off grid!\n");
08502             return;
08503         }
08504         kmax = (int)ceil((position[2]+rtot)/hzed);
08505         if (kmax > (nz-2)) {
08506             Vnm_print(2, "dbPermanentMultipoleForce: Atom off grid!\n");
08507             return;
08508         }
08509         for (i=imin; i<=imax; i++) {
08510             for (j=jmin; j<=jmax; j++) {
08511                 for (k=kmin; k<=kmax; k++) {
08512                     /* i,j,k */
08513                     gpos[0] = (i+0.5)*hx + xmin;
08514                     gpos[1] = j*hy + ymin;
08515                     gpos[2] = k*hzed + zmin;
08516                     Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
08517                     Vpmg_splineSelect(srfm, acc, gpos, thee->
08518                                     splineWin, 0.,
08519                                     atom, dHxijk);
08520                     for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
08521                     gpos[0] = i*hx + xmin;

```

```

08521         gpos[1] = (j+0.5)*hy + ymin;
08522         gpos[2] = k*hzed + zmin;
08523         Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
08524         Vpmg_splineSelect(srfm, acc, gpos, thee->
splineWin, 0.,
08525             atom, dHyijk);
08526         for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
08527         gpos[0] = i*hx + xmin;
08528         gpos[1] = j*hy + ymin;
08529         gpos[2] = (k+0.5)*hzed + zmin;
08530         Hziijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
08531         Vpmg_splineSelect(srfm, acc, gpos, thee->
splineWin, 0.,
08532             atom, dHziijk);
08533         for (l=0; l<3; l++) dHziijk[l] *= Hziijk;
08534         /* i-1,j,k */
08535         gpos[0] = (i-0.5)*hx + xmin;
08536         gpos[1] = j*hy + ymin;
08537         gpos[2] = k*hzed + zmin;
08538         Hximljk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
08539         Vpmg_splineSelect(srfm, acc, gpos, thee->
splineWin, 0.,
08540             atom, dHximljk);
08541         for (l=0; l<3; l++) dHximljk[l] *= Hximljk;
08542         /* i,j-1,k */
08543         gpos[0] = i*hx + xmin;
08544         gpos[1] = (j-0.5)*hy + ymin;
08545         gpos[2] = k*hzed + zmin;
08546         Hyijmlk = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
08547         Vpmg_splineSelect(srfm, acc, gpos, thee->
splineWin, 0.,
08548             atom, dHyijmlk);
08549         for (l=0; l<3; l++) dHyijmlk[l] *= Hyijmlk;
08550         /* i,j,k-1 */
08551         gpos[0] = i*hx + xmin;
08552         gpos[1] = j*hy + ymin;
08553         gpos[2] = (k-0.5)*hzed + zmin;
08554         Hzijkm1 = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
08555         Vpmg_splineSelect(srfm, acc, gpos, thee->
splineWin, 0.,
08556             atom, dHzijkm1);
08557         for (l=0; l<3; l++) dHzijkm1[l] *= Hzijkm1;
08558         dbFmag = u[IJK(i,j,k)];
08559         tgrad[0] =
08560             (dHxijk[0] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)]))
08561             + dHximljk[0] * (u[IJK(i-1,j,k)]-u[IJK(i,j,k)])) /VSQR(hx)
08562             + (dHyijk[0] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)]))
08563             + dHyijmlk[0] * (u[IJK(i,j-1,k)]-u[IJK(i,j,k)])) /VSQR(hy)
08564             + (dHziijk[0] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)]))
08565             + dHzijkm1[0] * (u[IJK(i,j,k-1)]-u[IJK(i,j,k)])) /VSQR(hzed);
08566         tgrad[1] =
08567             (dHxijk[1] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)]))
08568             + dHximljk[1] * (u[IJK(i-1,j,k)]-u[IJK(i,j,k)])) /VSQR(hx)
08569             + (dHyijk[1] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)]))
08570             + dHyijmlk[1] * (u[IJK(i,j-1,k)]-u[IJK(i,j,k)])) /VSQR(hy)
08571             + (dHziijk[1] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)]))
08572             + dHzijkm1[1] * (u[IJK(i,j,k-1)]-u[IJK(i,j,k)])) /VSQR(hzed);
08573         tgrad[2] =
08574             (dHxijk[2] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)]))
08575             + dHximljk[2] * (u[IJK(i-1,j,k)]-u[IJK(i,j,k)])) /VSQR(hx)
08576             + (dHyijk[2] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)]))
08577             + dHyijmlk[2] * (u[IJK(i,j-1,k)]-u[IJK(i,j,k)])) /VSQR(hy)
08578             + (dHziijk[2] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)]))
08579             + dHzijkm1[2] * (u[IJK(i,j,k-1)]-u[IJK(i,j,k)])) /VSQR(hzed);
08580         force[0] += (dbFmag*tgrad[0]);
08581         force[1] += (dbFmag*tgrad[1]);
08582         force[2] += (dbFmag*tgrad[2]);
08583     } /* k loop */
08584 } /* j loop */
08585 } /* i loop */
08586 force[0] = -force[0]*hx*hy*hzed*deps*0.5*izmagic;
08587 force[1] = -force[1]*hx*hy*hzed*deps*0.5*izmagic;
08588 force[2] = -force[2]*hx*hy*hzed*deps*0.5*izmagic;
08589 }
08590 }
08591
08592 VPUBLIC void Vpmg_qfDirectPolForce(Vpmg *thee, Vgrid* perm,
Vgrid *induced,
08593     int atomID, double force[3], double torque[3]) {
08594
08595     Vatom *atom;

```



```

08596     Vpbe *pbe;
08597     double f, fp, *u, *up, *apos, position[3];
08598
08599     /* Grid variables */
08600     int nx,ny,nz;
08601     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
08602     double hx, hy, hzed, ifloat, jfloat, kfloat;
08603
08604     /* B-spline weights */
08605     double mx, my, mz, dmxx, dmy, dmz, d2mx, d2my, d2mz, d3mx, d3my, d3mz;
08606     double mi, mj, mk;
08607
08608     /* Loop indeces */
08609     int i, j, k, ii, jj, kk;
08610     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
08611
08612     /* Permanent potential, field, field gradient and 2nd field gradient */
08613     double pot, e[3], de[3][3], d2e[3][3][3];
08614     /* Induced dipole field */
08615     double dep[3][3];
08616
08617     /* Permanent multipole components */
08618     double *dipole, *quad;
08619     double c, ux, uy, uz, qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
08620     double uix, uiy, uiz;
08621
08622     VASSERT(thee != VNULL);
08623     VASSERT(induced != VNULL); /* the potential due to permanent multipoles */
08624     VASSERT(induced != VNULL); /* the potential due to local induced dipoles */
08625     VASSERT(thee->pbe != VNULL);
08626     VASSERT(thee->pbe->alist != VNULL);
08627
08628     atom = Valist_getAtom(thee->pbe->alist, atomID);
08629     VASSERT(atom->partID != 0); /* all atoms must be in the same partition */
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_getInducedDipole(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 = induced->data;
08683     up = perm->data;
08684
08685     /* Make sure we're on the grid */
08686     if ((apos[0] <= (xmin+2*hx)) || (apos[0] >= (xmax-2*hx)) \
08687         || (apos[1] <= (ymin+2*hy)) || (apos[1] >= (ymax-2*hy)) \
08688         || (apos[2] <= (zmin+2*hzed)) || (apos[2] >= (zmax-2*hzed))) {
08689         Vnm_print(2, "qfDirectPolForce: Atom off the mesh (ignoring) %6.3f %6.3f %6.3f\n", apos[0], apos[1
], apos[2]);
08690         fflush(stderr);
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             dmxx = d2bspline4(mi);
08732             d2mx = d3bspline4(mi);
08733             d3mx = d3bspline4(mi);
08734             for (jj=jm2; jj<=jp2; jj++) {
08735                 mj = VFCHI4(jj, jfloat);
08736                 my = bspline4(mj);
08737                 dmy = d2bspline4(mj);
08738                 d2my = d3bspline4(mj);
08739                 d3my = d3bspline4(mj);
08740                 for (kk=km2; kk<=kp2; kk++) {
08741                     mk = VFCHI4(kk, kfloat);
08742                     mz = bspline4(mk);
08743                     dmz = d2bspline4(mk);
08744                     d2mz = d3bspline4(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*dmxx*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);

```

```

08757         de[1][0] += f*dmx*dmy*mz/(hy*hx);
08758         de[1][1] += f*mx*d2my*mz/(hy*hy);
08759         de[2][0] += f*dmx*my*dmz/(hx*hzed);
08760         de[2][1] += f*mx*dmy*dmz/(hy*hzed);
08761         de[2][2] += f*mx*my*d2mz/(hzed*hzed);
08762         /* The gradient of the (permanent) field */
08763         dep[0][0] += fp*d2mx*my*mz/(hx*hx);
08764         dep[1][0] += fp*dmx*dmy*mz/(hy*hx);
08765         dep[1][1] += fp*mx*d2my*mz/(hy*hy);
08766         dep[2][0] += fp*dmx*my*dmz/(hx*hzed);
08767         dep[2][1] += fp*mx*dmy*dmz/(hy*hzed);
08768         dep[2][2] += fp*mx*my*d2mz/(hzed*hzed);
08769         /* The 2nd gradient of the field
08770         VxVxVa */
08771         d2e[0][0][0] += f*d3mx*my*mz/(hx*hx*hx);
08772         d2e[0][0][1] += f*d2mx*dmy*mz/(hx*hy*hx);
08773         d2e[0][0][2] += f*d2mx*my*dmz/(hx*hx*hzed);
08774         /* VyVxVa */
08775         d2e[1][0][0] += f*d2mx*dmy*mz/(hx*hx*hy);
08776         d2e[1][0][1] += f*dmx*d2my*mz/(hx*hy*hy);
08777         d2e[1][0][2] += f*dmx*dmy*dmz/(hx*hy*hzed);
08778         /* VyVyVa */
08779         d2e[1][1][0] += f*dmx*d2my*mz/(hx*hy*hy);
08780         d2e[1][1][1] += f*mx*d3my*mz/(hy*hy*hy);
08781         d2e[1][1][2] += f*mx*d2my*dmz/(hy*hy*hzed);
08782         /* VzVxVa */
08783         d2e[2][0][0] += f*d2mx*my*dmz/(hx*hx*hzed);
08784         d2e[2][0][1] += f*dmx*dmy*dmz/(hx*hy*hzed);
08785         d2e[2][0][2] += f*dmx*my*d2mz/(hx*hzed*hzed);
08786         /* VzVyVa */
08787         d2e[2][1][0] += f*dmx*dmy*dmz/(hx*hy*hzed);
08788         d2e[2][1][1] += f*mx*d2my*dmz/(hy*hy*hzed);
08789         d2e[2][1][2] += f*mx*dmy*d2mz/(hy*hzed*hzed);
08790         /* VzVzVa */
08791         d2e[2][2][0] += f*dmx*my*d2mz/(hx*hzed*hzed);
08792         d2e[2][2][1] += f*mx*dmy*d2mz/(hy*hzed*hzed);
08793         d2e[2][2][2] += f*mx*my*d3mz/(hzed*hzed*hzed);
08794     }
08795 }
08796 }
08797 }
08798
08799 /* force on permanent multipole due to induced reaction field */
08800
08801 /* Monopole Force */
08802 force[0] = e[0]*c;
08803 force[1] = e[1]*c;
08804 force[2] = e[2]*c;
08805
08806 /* Dipole Force */
08807 force[0] -= de[0][0]*ux+de[1][0]*uy+de[2][0]*uz;
08808 force[1] -= de[1][0]*ux+de[1][1]*uy+de[2][1]*uz;
08809 force[2] -= de[2][0]*ux+de[2][1]*uy+de[2][2]*uz;
08810
08811 /* Quadrupole Force */
08812 force[0] += d2e[0][0][0]*qxx
08813         + d2e[1][0][0]*qyx*2.0+d2e[1][1][0]*qyy
08814         + d2e[2][0][0]*qzx*2.0+d2e[2][1][0]*qzy*2.0+d2e[2][2][0]*qzz;
08815 force[1] += d2e[0][0][1]*qxx
08816         + d2e[1][0][1]*qyx*2.0+d2e[1][1][1]*qyy
08817         + d2e[2][0][1]*qzx*2.0+d2e[2][1][1]*qzy*2.0+d2e[2][2][1]*qzz;
08818 force[2] += d2e[0][0][2]*qxx
08819         + d2e[1][0][2]*qyx*2.0+d2e[1][1][2]*qyy
08820         + d2e[2][0][2]*qzx*2.0+d2e[2][1][2]*qzy*2.0+d2e[2][2][2]*qzz;
08821
08822 /* torque on permanent mulitpole due to induced reaction field */
08823
08824 /* Dipole Torque */
08825 torque[0] = uy * e[2] - uz * e[1];
08826 torque[1] = uz * e[0] - ux * e[2];
08827 torque[2] = ux * e[1] - uy * e[0];
08828
08829 /* Quadrupole Torque */
08830 /* Tx = -2.0*(Sum_a (Qya*dEaz) + Sum_b (Qzb*dEby))
08831    Ty = -2.0*(Sum_a (Qza*dEax) + Sum_b (Qxb*dEbz))
08832    Tz = -2.0*(Sum_a (Qxa*dEay) + Sum_b (Qyb*dEbx)) */
08833 de[0][1] = de[1][0];
08834 de[0][2] = de[2][0];
08835 de[1][2] = de[2][1];
08836 torque[0] -= 2.0*(qyx*de[0][2] + qyy*de[1][2] + qyz*de[2][2]
08837         - qzx*de[0][1] - qzy*de[1][1] - qzz*de[2][1]);

```

```

08838 torque[1] -= 2.0*(qzx*de[0][0] + qzy*de[1][0] + qzz*de[2][0]
08839             - qxx*de[0][2] - qxy*de[1][2] - qxz*de[2][2]);
08840 torque[2] -= 2.0*(qxx*de[0][1] + qxy*de[1][1] + qxz*de[2][1]
08841             - qyx*de[0][0] - qyy*de[1][0] - qyz*de[2][0]);
08842
08843 /* force on induced dipole due to permanent reaction field */
08844
08845 force[0] -= dep[0][0]*uix+dep[1][0]*uiy+dep[2][0]*uiz;
08846 force[1] -= dep[1][0]*uix+dep[1][1]*uiy+dep[2][1]*uiz;
08847 force[2] -= dep[2][0]*uix+dep[2][1]*uiy+dep[2][2]*uiz;
08848
08849 force[0] = 0.5 * force[0];
08850 force[1] = 0.5 * force[1];
08851 force[2] = 0.5 * force[2];
08852 torque[0] = 0.5 * torque[0];
08853 torque[1] = 0.5 * torque[1];
08854 torque[2] = 0.5 * torque[2];
08855
08856 /* printf(" qPhi Force %f %f %f\n", force[0], force[1], force[2]);
08857    printf(" qPhi Torque %f %f %f\n", torque[0], torque[1], torque[2]); */
08858 }
08859
08860 VPUBLIC void Vpmg_qfNLDirectPolForce(Vpmg *thee,
    Vgrid *perm, Vgrid *nlInduced,
08861                                     int atomID, double force[3], double torque[3]) {
08862
08863     Vatom *atom;
08864     double *apos, *dipole, *quad, position[3], hx, hy, hzed;
08865     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
08866     double pot, e[3], de[3][3], dep[3][3], d2e[3][3][3];
08867     double mx, my, mz, dmx, dmy, dmz, mi, mj, mk;
08868     double d2mx, d2my, d2mz, d3mx, d3my, d3mz;
08869     double *u, *up, charge, ifloat, jfloat, kfloat;
08870     double f, fp, c, ux, uy, uz, qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
08871     double uix, uiy, uiz;
08872     int i,j,k,nx, ny, nz, im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1;
08873     int kp1, kp2, ii, jj, kk;
08874
08875     VASSERT(thee != VNULL);
08876     VASSERT(perm != VNULL); /* potential due to permanent multipoles. */
08877     VASSERT(nlInduced != VNULL); /* potential due to non-local induced dipoles */
08878     VASSERT(!thee->pmgp->nonlin); /* Nonlinear PBE is not implemented for AMOEBA */
08879
08880     atom = Valist_getAtom(thee->pbe->alist, atomID);
08881     VASSERT(atom->partID != 0); /* Currently all atoms must be in the same partition. */
08882     apos = Vatom_getPosition(atom);
08883
08884     c = Vatom_getCharge(atom);
08885     dipole = Vatom_getDipole(atom);
08886     ux = dipole[0];
08887     uy = dipole[1];
08888     uz = dipole[2];
08889     quad = Vatom_getQuadrupole(atom);
08890     qxx = quad[0]/3.0;
08891     qxy = quad[1]/3.0;
08892     qxz = quad[2]/3.0;
08893     qyx = quad[3]/3.0;
08894     qyy = quad[4]/3.0;
08895     qyz = quad[5]/3.0;
08896     qzx = quad[6]/3.0;
08897     qzy = quad[7]/3.0;
08898     qzz = quad[8]/3.0;
08899
08900     dipole = Vatom_getNLInducedDipole(atom);
08901     uix = dipole[0];
08902     uiy = dipole[1];
08903     uiz = dipole[2];
08904
08905     /* Reset Field Gradients */
08906     pot = 0.0;
08907     for (i=0;i<3;i++){
08908         e[i] = 0.0;
08909         for (j=0;j<3;j++){
08910             de[i][j] = 0.0;
08911             dep[i][j] = 0.0;
08912             for (k=0;k<3;k++){
08913                 d2e[i][j][k] = 0.0;
08914             }
08915         }
08916     }
08917

```

```

08918      /* Mesh info */
08919      nx = thee->pmgp->nx;
08920      ny = thee->pmgp->ny;
08921      nz = thee->pmgp->nz;
08922      hx = thee->pmgp->hx;
08923      hy = thee->pmgp->hy;
08924      hzed = thee->pmgp->hzed;
08925      xlen = thee->pmgp->xlen;
08926      ylen = thee->pmgp->ylen;
08927      zlen = thee->pmgp->zlen;
08928      xmin = thee->pmgp->xmin;
08929      ymin = thee->pmgp->ymin;
08930      zmin = thee->pmgp->zmin;
08931      xmax = thee->pmgp->xmax;
08932      ymax = thee->pmgp->ymax;
08933      zmax = thee->pmgp->zmax;
08934      u = nlInduced->data;
08935      up = perm->data;
08936
08937
08938      /* Make sure we're on the grid */
08939      if ((apos[0] <= (xmin+2*hx)) || (apos[0] >= (xmax-2*hx)) \
08940          || (apos[1] <= (ymin+2*hy)) || (apos[1] >= (ymax-2*hy)) \
08941          || (apos[2] <= (zmin+2*hzed)) || (apos[2] >= (zmax-2*hzed))) {
08942          Vnm_print(2, "qfNLDirectMultipoleForce: Atom off the mesh (ignoring) %6.3f %6.3f %6.3f\n", apos[0]
, apos[1], apos[2]);
08943      } else {
08944
08945          /* Convert the atom position to grid coordinates */
08946          position[0] = apos[0] - xmin;
08947          position[1] = apos[1] - ymin;
08948          position[2] = apos[2] - zmin;
08949          ifloat = position[0]/hx;
08950          jfloat = position[1]/hy;
08951          kfloat = position[2]/hzed;
08952          ip1 = (int)ceil(ifloat);
08953          ip2 = ip1 + 2;
08954          im1 = (int)floor(ifloat);
08955          im2 = im1 - 2;
08956          jp1 = (int)ceil(jfloat);
08957          jp2 = jp1 + 2;
08958          jm1 = (int)floor(jfloat);
08959          jm2 = jm1 - 2;
08960          kp1 = (int)ceil(kfloat);
08961          kp2 = kp1 + 2;
08962          km1 = (int)floor(kfloat);
08963          km2 = km1 - 2;
08964
08965          /* This step shouldn't be necessary, but it saves nasty debugging
08966             * later on if something goes wrong */
08967          ip2 = VMIN2(ip2, nx-1);
08968          ip1 = VMIN2(ip1, nx-1);
08969          im1 = VMAX2(im1, 0);
08970          im2 = VMAX2(im2, 0);
08971          jp2 = VMIN2(jp2, ny-1);
08972          jp1 = VMIN2(jp1, ny-1);
08973          jm1 = VMAX2(jm1, 0);
08974          jm2 = VMAX2(jm2, 0);
08975          kp2 = VMIN2(kp2, nz-1);
08976          kp1 = VMIN2(kp1, nz-1);
08977          km1 = VMAX2(km1, 0);
08978          km2 = VMAX2(km2, 0);
08979
08980          for (ii=im2; ii<=ip2; ii++) {
08981              mi = VFCHI4(ii, ifloat);
08982              mx = bspline4(mi);
08983              dmx = dbspline4(mi);
08984              d2mx = d2bspline4(mi);
08985              d3mx = d3bspline4(mi);
08986              for (jj=jm2; jj<=jp2; jj++) {
08987                  mj = VFCHI4(jj, jfloat);
08988                  my = bspline4(mj);
08989                  dmy = dbspline4(mj);
08990                  d2my = d2bspline4(mj);
08991                  d3my = d3bspline4(mj);
08992                  for (kk=km2; kk<=kp2; kk++) {
08993                      mk = VFCHI4(kk, kfloat);
08994                      mz = bspline4(mk);
08995                      dmz = dbspline4(mk);
08996                      d2mz = d2bspline4(mk);
08997                      d3mz = d3bspline4(mk);

```

```

08998         f = u[IJK(ii,jj,kk)];
08999         fp = up[IJK(ii,jj,kk)];
09000         /* The potential */
09001         pot += f*mx*my*mz;
09002         /* The field */
09003         e[0] += f*dmx*my*mz/hx;
09004         e[1] += f*mx*dmy*mz/hy;
09005         e[2] += f*mx*my*dmz/hzed;
09006         /* The gradient of the field */
09007         de[0][0] += f*d2mx*my*mz/(hx*hx);
09008         de[1][0] += f*dmx*dmy*mz/(hy*hx);
09009         de[1][1] += f*mx*d2my*mz/(hy*hy);
09010         de[2][0] += f*dmx*my*dmz/(hx*hzed);
09011         de[2][1] += f*mx*dmy*dmz/(hy*hzed);
09012         de[2][2] += f*mx*my*d2mz/(hzed*hzed);
09013         /* The gradient of the (permanent) field */
09014         dep[0][0] += fp*d2mx*my*mz/(hx*hx);
09015         dep[1][0] += fp*dmx*dmy*mz/(hy*hx);
09016         dep[1][1] += fp*mx*d2my*mz/(hy*hy);
09017         dep[2][0] += fp*dmx*my*dmz/(hx*hzed);
09018         dep[2][1] += fp*mx*dmy*dmz/(hy*hzed);
09019         dep[2][2] += fp*mx*my*d2mz/(hzed*hzed);
09020         /* The 2nd gradient of the field */
09021         /* VxVxVa */
09022         d2e[0][0][0] += f*d3mx*my*mz/(hx*hx*hx);
09023         d2e[0][0][1] += f*d2mx*dmy*mz/(hx*hy*hx);
09024         d2e[0][0][2] += f*d2mx*my*dmz/(hx*hx*hzed);
09025         /* VyVxVa */
09026         d2e[1][0][0] += f*d2mx*dmy*mz/(hx*hx*hy);
09027         d2e[1][0][1] += f*dmx*d2my*mz/(hx*hy*hy);
09028         d2e[1][0][2] += f*dmx*dmy*dmz/(hx*hy*hzed);
09029         /* VyVyVa */
09030         d2e[1][1][0] += f*dmx*d2my*mz/(hx*hy*hy);
09031         d2e[1][1][1] += f*mx*d3my*mz/(hy*hy*hy);
09032         d2e[1][1][2] += f*mx*d2my*dmz/(hy*hy*hzed);
09033         /* VzVxVa */
09034         d2e[2][0][0] += f*d2mx*my*dmz/(hx*hx*hzed);
09035         d2e[2][0][1] += f*dmx*dmy*dmz/(hx*hy*hzed);
09036         d2e[2][0][2] += f*dmx*my*d2mz/(hx*hzed*hzed);
09037         /* VzVyVa */
09038         d2e[2][1][0] += f*dmx*dmy*dmz/(hx*hy*hzed);
09039         d2e[2][1][1] += f*mx*d2my*dmz/(hy*hy*hzed);
09040         d2e[2][1][2] += f*mx*dmy*d2mz/(hy*hzed*hzed);
09041         /* VzVzVa */
09042         d2e[2][2][0] += f*dmx*my*d2mz/(hx*hzed*hzed);
09043         d2e[2][2][1] += f*mx*dmy*d2mz/(hy*hzed*hzed);
09044         d2e[2][2][2] += f*mx*my*d3mz/(hzed*hzed*hzed);
09045     }
09046 }
09047 }
09048 }
09049
09050 /* force on permanent multipole due to non-local induced reaction field */
09051
09052 /* Monopole Force */
09053 force[0] = e[0]*c;
09054 force[1] = e[1]*c;
09055 force[2] = e[2]*c;
09056
09057 /* Dipole Force */
09058 force[0] -= de[0][0]*ux+de[1][0]*uy+de[2][0]*uz;
09059 force[1] -= de[1][0]*ux+de[1][1]*uy+de[2][1]*uz;
09060 force[2] -= de[2][0]*ux+de[2][1]*uy+de[2][2]*uz;
09061
09062 /* Quadrupole Force */
09063 force[0] += d2e[0][0][0]*qxx
09064         + d2e[1][0][0]*qyx*2.0+d2e[1][1][0]*qyy
09065         + d2e[2][0][0]*qzx*2.0+d2e[2][1][0]*qzy*2.0+d2e[2][2][0]*qzz;
09066 force[1] += d2e[0][0][1]*qxx
09067         + d2e[1][0][1]*qyx*2.0+d2e[1][1][1]*qyy
09068         + d2e[2][0][1]*qzx*2.0+d2e[2][1][1]*qzy*2.0+d2e[2][2][1]*qzz;
09069 force[2] += d2e[0][0][2]*qxx
09070         + d2e[1][0][2]*qyx*2.0+d2e[1][1][2]*qyy
09071         + d2e[2][0][2]*qzx*2.0+d2e[2][1][2]*qzy*2.0+d2e[2][2][2]*qzz;
09072
09073 /* torque on permanent mulitpole due to non-local induced reaction field */
09074
09075 /* Dipole Torque */
09076 torque[0] = uy * e[2] - uz * e[1];
09077 torque[1] = uz * e[0] - ux * e[2];
09078 torque[2] = ux * e[1] - uy * e[0];

```

```

09079
09080 /* Quadrupole Torque */
09081 /* Tx = -2.0*(Sum_a (Qya*dEaz) + Sum_b (Qzb*dEby))
09082    Ty = -2.0*(Sum_a (Qza*dEax) + Sum_b (Qxb*dEbz))
09083    Tz = -2.0*(Sum_a (Qxa*dEay) + Sum_b (Qyb*dEbx)) */
09084 de[0][1] = de[1][0];
09085 de[0][2] = de[2][0];
09086 de[1][2] = de[2][1];
09087 torque[0] -= 2.0*(qyx*de[0][2] + qyy*de[1][2] + qyz*de[2][2]
09088    - qzx*de[0][1] - qzy*de[1][1] - qzz*de[2][1]);
09089 torque[1] -= 2.0*(qzx*de[0][0] + qzy*de[1][0] + qzz*de[2][0]
09090    - qxx*de[0][2] - qxy*de[1][2] - qxz*de[2][2]);
09091 torque[2] -= 2.0*(qxx*de[0][1] + qxy*de[1][1] + qxz*de[2][1]
09092    - qyx*de[0][0] - qyy*de[1][0] - qyz*de[2][0]);
09093
09094 /* force on non-local induced dipole due to permanent reaction field */
09095
09096 force[0] -= dep[0][0]*uix+dep[1][0]*uiy+dep[2][0]*uiz;
09097 force[1] -= dep[1][0]*uix+dep[1][1]*uiy+dep[2][1]*uiz;
09098 force[2] -= dep[2][0]*uix+dep[2][1]*uiy+dep[2][2]*uiz;
09099
09100 force[0] = 0.5 * force[0];
09101 force[1] = 0.5 * force[1];
09102 force[2] = 0.5 * force[2];
09103 torque[0] = 0.5 * torque[0];
09104 torque[1] = 0.5 * torque[1];
09105 torque[2] = 0.5 * torque[2];
09106
09107 /* printf(" qPhi Force %f %f %f\n", force[0], force[1], force[2]);
09108    printf(" qPhi Torque %f %f %f\n", torque[0], torque[1], torque[2]); */
09109 }
09110
09111 VPUBLIC void Vpmg_ibDirectPolForce(Vpmg *thee, Vgrid *perm,
Vgrid *induced,
09112 int atomID, double force[3]) {
09113
09114 Vatom *atom;
09115 Valist *alist;
09116 Vacc *acc;
09117 Vpbe *pbe;
09118 Vsurf_Meth srfm;
09119
09120 double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
09121 double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;
09122 double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
09123 double izmagic;
09124 int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
09125
09126 VASSERT(thee != VNULL);
09127 VASSERT(perm != VNULL); /* potential due to permanent multipoles.*/
09128 VASSERT(induced != VNULL); /* potential due to induced dipoles. */
09129 VASSERT (!thee->pmgp->nonlin); /* Nonlinear PBE is not implemented for AMOEBA */
09130
09131 acc = thee->pbe->acc;
09132 srfm = thee->surfMeth;
09133 atom = Valist_getAtom(thee->pbe->alist, atomID);
09134 VASSERT(atom->partID != 0); /* Currently all atoms must be in the same partition. */
09135 apos = Vatom_getPosition(atom);
09136 arad = Vatom_getRadius(atom);
09137
09138 /* Reset force */
09139 force[0] = 0.0;
09140 force[1] = 0.0;
09141 force[2] = 0.0;
09142
09143 /* Get PBE info */
09144 pbe = thee->pbe;
09145 acc = pbe->acc;
09146 alist = pbe->alist;
09147 irad = Vpbe_getMaxIonRadius(pbe);
09148 zkappa2 = Vpbe_getZkappa2(pbe);
09149 izmagic = 1.0/Vpbe_getZmagic(pbe);
09150
09151 VASSERT (zkappa2 > VPMGSMALL); /* It is ok to run AMOEBA with no ions, but this is checked for
higher up in the driver. */
09152
09153 /* Mesh info */
09154 nx = induced->nx;
09155 ny = induced->ny;
09156 nz = induced->nz;
09157 hx = induced->hx;

```

```

09158     hy = induced->hy;
09159     hzed = induced->hzed;
09160     xmin = induced->xmin;
09161     ymin = induced->ymin;
09162     zmin = induced->zmin;
09163     xmax = induced->xmax;
09164     ymax = induced->ymax;
09165     zmax = induced->zmax;
09166     xlen = xmax-xmin;
09167     ylen = ymax-ymin;
09168     zlen = zmax-zmin;
09169
09170     /* Make sure we're on the grid */
09171     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
09172         (apos[1]<=ymin) || (apos[1]>=ymax) || \
09173         (apos[2]<=zmin) || (apos[2]>=zmax)) {
09174         Vnm_print(2, "Vpmg_ibForce: Atom at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):\n",
09175             apos[0], apos[1], apos[2]);
09176         Vnm_print(2, "Vpmg_ibForce:      xmin = %g, xmax = %g\n", xmin, xmax);
09177         Vnm_print(2, "Vpmg_ibForce:      ymin = %g, ymax = %g\n", ymin, ymax);
09178         Vnm_print(2, "Vpmg_ibForce:      zmin = %g, zmax = %g\n", zmin, zmax);
09179         fflush(stderr);
09180     } else {
09181
09182         /* Convert the atom position to grid reference frame */
09183         position[0] = apos[0] - xmin;
09184         position[1] = apos[1] - ymin;
09185         position[2] = apos[2] - zmin;
09186
09187         /* Integrate over points within this atom's (inflated) radius */
09188         rtot = (irad + arad + thee->splineWin);
09189         rtot2 = VSQR(rtot);
09190         dx = rtot + 0.5*hx;
09191         imin = VMAX2(0, (int)ceil((position[0] - dx)/hx));
09192         imax = VMIN2(nx-1, (int)floor((position[0] + dx)/hx));
09193         for (i=imin; i<=imax; i++) {
09194             dx2 = VSQR(position[0] - hx*i);
09195             if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
09196             else dy = 0.5*hy;
09197             jmin = VMAX2(0, (int)ceil((position[1] - dy)/hy));
09198             jmax = VMIN2(ny-1, (int)floor((position[1] + dy)/hy));
09199             for (j=jmin; j<=jmax; j++) {
09200                 dy2 = VSQR(position[1] - hy*j);
09201                 if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
09202                 else dz = 0.5*hzed;
09203                 kmin = VMAX2(0, (int)ceil((position[2] - dz)/hzed));
09204                 kmax = VMIN2(nz-1, (int)floor((position[2] + dz)/hzed));
09205                 for (k=kmin; k<=kmax; k++) {
09206                     dz2 = VSQR(k*hzed - position[2]);
09207                     /* See if grid point is inside ivdw radius and set ccf
09208                        * accordingly (do spline assignment here) */
09209                     if ((dz2 + dy2 + dx2) <= rtot2) {
09210                         gpos[0] = i*hx + xmin;
09211                         gpos[1] = j*hy + ymin;
09212                         gpos[2] = k*hzed + zmin;
09213                         Vpmg_splineSelect(srffm, acc, gpos, thee->
09214 splineWin, irad,
09215                             atom, tgrad);
09216                         fmag = induced->data[IJK(i,j,k)];
09217                         fmag *= perm->data[IJK(i,j,k)];
09218                         fmag *= thee->kappa[IJK(i,j,k)];
09219                         force[0] += (zkappa2*fmag*tgrad[0]);
09220                         force[1] += (zkappa2*fmag*tgrad[1]);
09221                         force[2] += (zkappa2*fmag*tgrad[2]);
09222                     } /* k loop */
09223                 } /* j loop */
09224             } /* i loop */
09225         }
09226
09227         force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
09228         force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
09229         force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
09230
09231     }
09232
09233 VPUBLIC void Vpmg_ibNLDirectPolForce(Vpmg *thee,
09234     Vgrid *perm, Vgrid *nlInduced,
09235     int atomID, double force[3]) {
09236     Vpmg_ibDirectPolForce(thee, perm, nlInduced, atomID, force);

```



```

09237
09238 VPUBLIC void Vpmg_dbDirectPolForce(Vpmg *thee, Vgrid *perm,
    Vgrid *induced,
09239                                     int atomID, double force[3]) {
09240
09241     Vatom *atom;
09242     Vacc *acc;
09243     Vpbe *pbe;
09244     Vsurf_Meth srfm;
09245
09246     double *apos, position[3], arad, hx, hy, hzed, izmagic, deps, depsi;
09247     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2, epsp;
09248     double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
09249     double *u, *up, Hxijk, Hyijk, Hzijk, Hximljk, Hyijmlk, Hzijkm1;
09250     double dHxijk[3], dHyijk[3], dHzijk[3], dHximljk[3], dHyijmlk[3];
09251     double dHzijkm1[3];
09252     int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
09253
09254     VASSERT(thee != VNULL);
09255     VASSERT(perm != VNULL); /* permanent multipole PMG solution. */
09256     VASSERT(induced != VNULL); /* potential due to induced dipoles. */
09257
09258     acc = thee->pbe->acc;
09259     atom = Valist_getAtom(thee->pbe->alist, atomID);
09260     VASSERT (atom->partID != 0); /* Currently all atoms must be in the same partition. */
09261     apos = Vatom_getPosition(atom);
09262     arad = Vatom_getRadius(atom);
09263
09264     /* Reset force */
09265     force[0] = 0.0;
09266     force[1] = 0.0;
09267     force[2] = 0.0;
09268
09269     /* Get PBE info */
09270     pbe = thee->pbe;
09271     acc = pbe->acc;
09272     srfm = thee->surfMeth;
09273     epsp = Vpbe_getSoluteDiel(pbe);
09274     epsw = Vpbe_getSolventDiel(pbe);
09275     kT = Vpbe_getTemperature(pbe) * (1e-3) * Vunit_Na*
    Vunit_kb;
09276     izmagic = 1.0/Vpbe_getZmagic(pbe);
09277
09278     deps = (epsw - epsp);
09279     depsi = 1.0/deps;
09280     VASSERT(VABS(deps) > VPMGSMALL);
09281
09282     /* Mesh info */
09283     nx = thee->pmgp->nx;
09284     ny = thee->pmgp->ny;
09285     nz = thee->pmgp->nz;
09286     hx = thee->pmgp->hx;
09287     hy = thee->pmgp->hy;
09288     hzed = thee->pmgp->hzed;
09289     xlen = thee->pmgp->xlen;
09290     ylen = thee->pmgp->ylen;
09291     zlen = thee->pmgp->zlen;
09292     xmin = thee->pmgp->xmin;
09293     ymin = thee->pmgp->ymin;
09294     zmin = thee->pmgp->zmin;
09295     xmax = thee->pmgp->xmax;
09296     ymax = thee->pmgp->ymax;
09297     zmax = thee->pmgp->zmax;
09298     /* If the permanent and induced potentials are flipped the
09299        results are exactly the same. */
09300     u = induced->data;
09301     up = perm->data;
09302
09303     /* Make sure we're on the grid */
09304     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
09305         (apos[1]<=ymin) || (apos[1]>=ymax) || \
09306         (apos[2]<=zmin) || (apos[2]>=zmax)) {
09307         Vnm_print(2, "Vpmg_dbDirectPolForce: Atom at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):\n"
    , apos[0], apos[1], apos[2]);
09308         Vnm_print(2, "Vpmg_dbDirectPolForce: xmin = %g, xmax = %g\n", xmin, xmax);
09309         Vnm_print(2, "Vpmg_dbDirectPolForce: ymin = %g, ymax = %g\n", ymin, ymax);
09310         Vnm_print(2, "Vpmg_dbDirectPolForce: zmin = %g, zmax = %g\n", zmin, zmax);
09311         fflush(stderr);
09312     } else {
09313
09314         /* Convert the atom position to grid reference frame */

```

```

09315     position[0] = apos[0] - xmin;
09316     position[1] = apos[1] - ymin;
09317     position[2] = apos[2] - zmin;
09318
09319     /* Integrate over points within this atom's (inflated) radius */
09320     rtot = (arad + thee->splineWin);
09321     rtot2 = VSQR(rtot);
09322     dx = rtot/hx;
09323     imin = (int)floor((position[0]-rtot)/hx);
09324     if (imin < 1) {
09325         Vnm_print(2, "Vpmg_dbDirectPolForce: Atom %d off grid!\n", atomID);
09326         return;
09327     }
09328     imax = (int)ceil((position[0]+rtot)/hx);
09329     if (imax > (nx-2)) {
09330         Vnm_print(2, "Vpmg_dbDirectPolForce: Atom %d off grid!\n", atomID);
09331         return;
09332     }
09333     jmin = (int)floor((position[1]-rtot)/hy);
09334     if (jmin < 1) {
09335         Vnm_print(2, "Vpmg_dbDirectPolForce: Atom %d off grid!\n", atomID);
09336         return;
09337     }
09338     jmax = (int)ceil((position[1]+rtot)/hy);
09339     if (jmax > (ny-2)) {
09340         Vnm_print(2, "Vpmg_dbDirectPolForce: Atom %d off grid!\n", atomID);
09341         return;
09342     }
09343     kmin = (int)floor((position[2]-rtot)/hz);
09344     if (kmin < 1) {
09345         Vnm_print(2, "Vpmg_dbDirectPolForce: Atom %d off grid!\n", atomID);
09346         return;
09347     }
09348     kmax = (int)ceil((position[2]+rtot)/hz);
09349     if (kmax > (nz-2)) {
09350         Vnm_print(2, "Vpmg_dbDirectPolForce: Atom %d off grid!\n", atomID);
09351         return;
09352     }
09353     for (i=imin; i<=imax; i++) {
09354         for (j=jmin; j<=jmax; j++) {
09355             for (k=kmin; k<=kmax; k++) {
09356                 /* i,j,k */
09357                 gpos[0] = (i+0.5)*hx + xmin;
09358                 gpos[1] = j*hy + ymin;
09359                 gpos[2] = k*hz + zmin;
09360                 Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
09361                 Vpmg_splineSelect(srffm, acc, gpos, thee->
splineWin, 0.,
09362                                 atom, dHxijk);
09363                 for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
09364                 gpos[0] = i*hx + xmin;
09365                 gpos[1] = (j+0.5)*hy + ymin;
09366                 gpos[2] = k*hz + zmin;
09367                 Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
09368                 Vpmg_splineSelect(srffm, acc, gpos, thee->
splineWin, 0.,
09369                                 atom, dHyijk);
09370                 for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
09371                 gpos[0] = i*hx + xmin;
09372                 gpos[1] = j*hy + ymin;
09373                 gpos[2] = (k+0.5)*hz + zmin;
09374                 Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
09375                 Vpmg_splineSelect(srffm, acc, gpos, thee->
splineWin, 0.,
09376                                 atom, dHzijk);
09377                 for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
09378                 /* i-1,j,k */
09379                 gpos[0] = (i-0.5)*hx + xmin;
09380                 gpos[1] = j*hy + ymin;
09381                 gpos[2] = k*hz + zmin;
09382                 Hximljk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
09383                 Vpmg_splineSelect(srffm, acc, gpos, thee->
splineWin, 0.,
09384                                 atom, dHximljk);
09385                 for (l=0; l<3; l++) dHximljk[l] *= Hximljk;
09386                 /* i,j-1,k */
09387                 gpos[0] = i*hx + xmin;
09388                 gpos[1] = (j-0.5)*hy + ymin;
09389                 gpos[2] = k*hz + zmin;
09390                 Hyijmlk = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
09391                 Vpmg_splineSelect(srffm, acc, gpos, thee->

```

```

splineWin, 0.,
09392         atom, dHyijm1k);
09393     for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;
09394     /* i,j,k-1 */
09395     gpos[0] = i*hx + xmin;
09396     gpos[1] = j*hy + ymin;
09397     gpos[2] = (k-0.5)*hz + zmin;
09398     Hzijkm1 = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
09399     Vpmg_splineSelect(srfm, acc, gpos, thee->
splineWin, 0.,
09400         atom, dHzijkm1);
09401     for (l=0; l<3; l++) dHzijkm1[l] *= Hzijkm1;
09402
09403     dbFmag = up[IJK(i,j,k)];
09404     tgrad[0] =
09405         (dHxijk[0] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09406          + dHxim1jk[0] * (u[IJK(i-1,j,k)]-u[IJK(i,j,k)])) /VSQR(hx)
09407          + (dHyijk[0] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09408           + dHyijm1k[0] * (u[IJK(i,j-1,k)]-u[IJK(i,j,k)])) /VSQR(hy)
09409           + (dHzijkm1[0] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
09410            + dHzijkm1[0] * (u[IJK(i,j,k-1)]-u[IJK(i,j,k)])) /VSQR(hzed);
09411     tgrad[1] =
09412         (dHxijk[1] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09413          + dHxim1jk[1] * (u[IJK(i-1,j,k)]-u[IJK(i,j,k)])) /VSQR(hx)
09414          + (dHyijk[1] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09415           + dHyijm1k[1] * (u[IJK(i,j-1,k)]-u[IJK(i,j,k)])) /VSQR(hy)
09416           + (dHzijkm1[1] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
09417            + dHzijkm1[1] * (u[IJK(i,j,k-1)]-u[IJK(i,j,k)])) /VSQR(hzed);
09418     tgrad[2] =
09419         (dHxijk[2] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09420          + dHxim1jk[2] * (u[IJK(i-1,j,k)]-u[IJK(i,j,k)])) /VSQR(hx)
09421          + (dHyijk[2] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09422           + dHyijm1k[2] * (u[IJK(i,j-1,k)]-u[IJK(i,j,k)])) /VSQR(hy)
09423           + (dHzijkm1[2] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
09424            + dHzijkm1[2] * (u[IJK(i,j,k-1)]-u[IJK(i,j,k)])) /VSQR(hzed);
09425     force[0] += (dbFmag*tgrad[0]);
09426     force[1] += (dbFmag*tgrad[1]);
09427     force[2] += (dbFmag*tgrad[2]);
09428
09429     } /* k loop */
09430     } /* j loop */
09431     } /* i loop */
09432
09433     force[0] = -force[0]*hx*hy*hz*deps*0.5*izmagic;
09434     force[1] = -force[1]*hx*hy*hz*deps*0.5*izmagic;
09435     force[2] = -force[2]*hx*hy*hz*deps*0.5*izmagic;
09436
09437 }
09438 }
09439
09440 VPUBLIC void Vpmg_dbNLDirectPolForce(Vpmg *thee,
Vgrid *perm, Vgrid *nlInduced,
09441         int atomID, double force[3]) {
09442     Vpmg_dbDirectPolForce(thee, perm, nlInduced, atomID, force);
09443 }
09444
09445 VPUBLIC void Vpmg_qfMutualPolForce(Vpmg *thee, Vgrid *induced,
Vgrid *nlinduced, int atomID, double force[3]) {
09446
09447     Vatom *atom;
09448     double *apos, *dipole, position[3], hx, hy, hzed;
09449     double *u, *unl;
09450     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
09451     double de[3][3], denl[3][3];
09452     double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz, mi, mj, mk;
09453     double ifloat, jfloat, kfloat;
09454     double f, fnl, uix, uiy, uiz, uixnl, uiynl, uiznl;
09455     int i,j,k,nx, ny, nz, im2, im1, ip1, ip2, jm2, jml, jpl, jp2, km2, kml;
09456     int kpl, kp2, ii, jj, kk;
09457
09458     VASSERT(thee != VNULL); /* PMG object with PBE info. */
09459     VASSERT(induced != VNULL); /* potential due to induced dipoles. */
09460     VASSERT(nlinduced != VNULL); /* potential due to non-local induced dipoles. */
09461     atom = Valist_getAtom(thee->pbe->alist, atomID);
09462     VASSERT(atom->partID != 0); /* all atoms must be in the same partition. */
09463     apos = Vatom_getPosition(atom);
09464     dipole = Vatom_getInducedDipole(atom);
09465     uix = dipole[0];
09466     uiy = dipole[1];
09467     uiz = dipole[2];
09468     dipole = Vatom_getNLDipole(atom);
09469

```

```

09470     uixnl = dipole[0];
09471     uiynl = dipole[1];
09472     uiznl = dipole[2];
09473     u = induced->data;
09474     unl = nlinduced->data;
09475
09476     for (i=0;i<3;i++){
09477         for (j=0;j<3;j++){
09478             de[i][j] = 0.0;
09479             denl[i][j] = 0.0;
09480         }
09481     }
09482
09483     /* Mesh info */
09484     nx = induced->nx;
09485     ny = induced->ny;
09486     nz = induced->nz;
09487     hx = induced->hx;
09488     hy = induced->hy;
09489     hzed = induced->hzed;
09490     xmin = induced->xmin;
09491     ymin = induced->ymin;
09492     zmin = induced->zmin;
09493     xmax = induced->xmax;
09494     ymax = induced->ymax;
09495     zmax = induced->zmax;
09496     xlen = xmax-xmin;
09497     ylen = ymax-ymin;
09498     zlen = zmax-zmin;
09499
09500     /* If we aren't in the current position, then we're done */
09501     if (atom->partID == 0) return;
09502
09503     /* Make sure we're on the grid */
09504     if ((apos[0]<=(xmin+2*hx)) || (apos[0]>=(xmax-2*hx)) \
09505         || (apos[1]<=(ymin+2*hy)) || (apos[1]>=(ymax-2*hy)) \
09506         || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
09507         Vnm_print(2, "qfMutualPolForce: Atom off the mesh (ignoring) %6.3f %6.3f %6.3f\n", apos[0], apos[1
09508     ], apos[2]);
09509         fflush(stderr);
09510     } else {
09511         /* Convert the atom position to grid coordinates */
09512         position[0] = apos[0] - xmin;
09513         position[1] = apos[1] - ymin;
09514         position[2] = apos[2] - zmin;
09515         ifloat = position[0]/hx;
09516         jfloat = position[1]/hy;
09517         kfloat = position[2]/hzed;
09518         ip1 = (int)ceil(ifloat);
09519         ip2 = ip1 + 2;
09520         im1 = (int)floor(ifloat);
09521         im2 = im1 - 2;
09522         jp1 = (int)ceil(jfloat);
09523         jp2 = jp1 + 2;
09524         jm1 = (int)floor(jfloat);
09525         jm2 = jm1 - 2;
09526         kp1 = (int)ceil(kfloat);
09527         kp2 = kp1 + 2;
09528         km1 = (int)floor(kfloat);
09529         km2 = km1 - 2;
09530
09531         /* This step shouldn't be necessary, but it saves nasty debugging
09532          * later on if something goes wrong */
09533         ip2 = VMIN2(ip2,nx-1);
09534         ip1 = VMIN2(ip1,nx-1);
09535         im1 = VMAX2(im1,0);
09536         im2 = VMAX2(im2,0);
09537         jp2 = VMIN2(jp2,ny-1);
09538         jp1 = VMIN2(jp1,ny-1);
09539         jm1 = VMAX2(jm1,0);
09540         jm2 = VMAX2(jm2,0);
09541         kp2 = VMIN2(kp2,nz-1);
09542         kp1 = VMIN2(kp1,nz-1);
09543         km1 = VMAX2(km1,0);
09544         km2 = VMAX2(km2,0);
09545
09546         for (ii=im2; ii<=ip2; ii++) {
09547             mi = VFCHI4(ii,ifloat);
09548             mx = bspline4(mi);
09549             dmx = dbspline4(mi);

```

```

09550         d2mx = d2bspline4(mi);
09551     for (jj=jm2; jj<=jp2; jj++) {
09552         mj = VFCHI4(jj,jfloat);
09553         my = bspline4(mj);
09554         dmy = dbspline4(mj);
09555         d2my = d2bspline4(mj);
09556         for (kk=km2; kk<=kp2; kk++) {
09557             mk = VFCHI4(kk,kfloat);
09558             mz = bspline4(mk);
09559             dmz = dbspline4(mk);
09560             d2mz = d2bspline4(mk);
09561             f = u[IJK(ii,jj,kk)];
09562             fnl = unl[IJK(ii,jj,kk)];
09563
09564             /* The gradient of the reaction field
09565              due to induced dipoles */
09566             de[0][0] += f*d2mx*my*mz/(hx*hx);
09567             de[1][0] += f*dmx*dmy*mz/(hy*hx);
09568             de[1][1] += f*mx*d2my*mz/(hy*hy);
09569             de[2][0] += f*dmx*my*dmz/(hx*hzed);
09570             de[2][1] += f*mx*dmy*dmz/(hy*hzed);
09571             de[2][2] += f*mx*my*d2mz/(hzed*hzed);
09572
09573             /* The gradient of the reaction field
09574              due to non-local induced dipoles */
09575             denl[0][0] += fnl*d2mx*my*mz/(hx*hx);
09576             denl[1][0] += fnl*dmx*dmy*mz/(hy*hx);
09577             denl[1][1] += fnl*mx*d2my*mz/(hy*hy);
09578             denl[2][0] += fnl*dmx*my*dmz/(hx*hzed);
09579             denl[2][1] += fnl*mx*dmy*dmz/(hy*hzed);
09580             denl[2][2] += fnl*mx*my*d2mz/(hzed*hzed);
09581         }
09582     }
09583 }
09584 }
09585
09586 /* mutual polarization force */
09587 force[0] = -(de[0][0]*uixnl + de[1][0]*uiynl + de[2][0]*uiznl);
09588 force[1] = -(de[1][0]*uixnl + de[1][1]*uiynl + de[2][1]*uiznl);
09589 force[2] = -(de[2][0]*uixnl + de[2][1]*uiynl + de[2][2]*uiznl);
09590 force[0] -= denl[0][0]*uix + denl[1][0]*uiy + denl[2][0]*uiz;
09591 force[1] -= denl[1][0]*uix + denl[1][1]*uiy + denl[2][1]*uiz;
09592 force[2] -= denl[2][0]*uix + denl[2][1]*uiy + denl[2][2]*uiz;
09593
09594 force[0] = 0.5 * force[0];
09595 force[1] = 0.5 * force[1];
09596 force[2] = 0.5 * force[2];
09597
09598 }
09599
09600 VPUBLIC void Vpmg_ibMutualPolForce(Vpmg *thee, Vgrid *induced,
09601     Vgrid *nlininduced,
09602     int atomID, double force[3]) {
09603     Vatom *atom;
09604     Valist *alist;
09605     Vacc *acc;
09606     Vpbe *pbe;
09607     Vsurf_Meth srfm;
09608
09609     double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
09610     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;
09611     double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
09612     double izmagic;
09613     int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
09614
09615     VASSERT(thee != VNULL); /* We need a PMG object with PBE info. */
09616     VASSERT(induced != VNULL); /* We need the potential due to induced dipoles. */
09617     VASSERT(nlininduced != VNULL); /* We need the potential due to non-local induced dipoles. */
09618     VASSERT (!thee->pmgp->nonlin); /* Nonlinear PBE is not implemented for AMOEBA */
09619
09620     atom = Valist_getAtom(thee->pbe->alist, atomID);
09621     VASSERT (atom->partID != 0); /* Currently all atoms must be in the same partition. */
09622
09623     acc = thee->pbe->acc;
09624     srfm = thee->surfMeth;
09625     apos = Vatom_getPosition(atom);
09626     arad = Vatom_getRadius(atom);
09627
09628     /* Reset force */
09629     force[0] = 0.0;

```

```

09630     force[1] = 0.0;
09631     force[2] = 0.0;
09632
09633     /* If we aren't in the current position, then we're done */
09634     if (atom->partID == 0) return;
09635
09636     /* Get PBE info */
09637     pbe = thee->pbe;
09638     acc = pbe->acc;
09639     alist = pbe->alist;
09640     irad = Vpbe_getMaxIonRadius(pbe);
09641     zkappa2 = Vpbe_getZkappa2(pbe);
09642     izmagic = 1.0/Vpbe_getZmagic(pbe);
09643
09644     VASSERT (zkappa2 > VPMGSMALL); /* Should be a check for this further up.*/
09645
09646     /* Mesh info */
09647     nx = induced->nx;
09648     ny = induced->ny;
09649     nz = induced->nz;
09650     hx = induced->hx;
09651     hy = induced->hy;
09652     hzed = induced->hzed;
09653     xmin = induced->xmin;
09654     ymin = induced->ymin;
09655     zmin = induced->zmin;
09656     xmax = induced->xmax;
09657     ymax = induced->ymax;
09658     zmax = induced->zmax;
09659     xlen = xmax-xmin;
09660     ylen = ymax-ymin;
09661     zlen = zmax-zmin;
09662
09663     /* Make sure we're on the grid */
09664     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
09665         (apos[1]<=ymin) || (apos[1]>=ymax) || \
09666         (apos[2]<=zmin) || (apos[2]>=zmax)) {
09667         Vnm_print(2, "Vpmg_ibMutalPolForce: Atom at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):\n",
09668             apos[0], apos[1], apos[2]);
09669         Vnm_print(2, "Vpmg_ibMutalPolForce: xmin = %g, xmax = %g\n", xmin, xmax);
09670         Vnm_print(2, "Vpmg_ibMutalPolForce: ymin = %g, ymax = %g\n", ymin, ymax);
09671         Vnm_print(2, "Vpmg_ibMutalPolForce: zmin = %g, zmax = %g\n", zmin, zmax);
09672         fflush(stderr);
09673     } else {
09674
09675         /* Convert the atom position to grid reference frame */
09676         position[0] = apos[0] - xmin;
09677         position[1] = apos[1] - ymin;
09678         position[2] = apos[2] - zmin;
09679
09680         /* Integrate over points within this atom's (inflated) radius */
09681         rtot = (irad + arad + thee->splineWin);
09682         rtot2 = VSQR(rtot);
09683         dx = rtot + 0.5*hx;
09684         imin = VMAX2(0, (int)ceil((position[0] - dx)/hx));
09685         imax = VMIN2(nx-1, (int)floor((position[0] + dx)/hx));
09686         for (i=imin; i<=imax; i++) {
09687             dx2 = VSQR(position[0] - hx*i);
09688             if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
09689             else dy = 0.5*hy;
09690             jmin = VMAX2(0, (int)ceil((position[1] - dy)/hy));
09691             jmax = VMIN2(ny-1, (int)floor((position[1] + dy)/hy));
09692             for (j=jmin; j<=jmax; j++) {
09693                 dy2 = VSQR(position[1] - hy*j);
09694                 if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
09695                 else dz = 0.5*hzed;
09696                 kmin = VMAX2(0, (int)ceil((position[2] - dz)/hzed));
09697                 kmax = VMIN2(nz-1, (int)floor((position[2] + dz)/hzed));
09698                 for (k=kmin; k<=kmax; k++) {
09699                     dz2 = VSQR(k*hzed - position[2]);
09700                     /* See if grid point is inside ivdw radius and set ccf
09701                        * accordingly (do spline assignment here) */
09702                     if ((dz2 + dy2 + dx2) <= rtot2) {
09703                         gpos[0] = i*hx + xmin;
09704                         gpos[1] = j*hy + ymin;
09705                         gpos[2] = k*hzed + zmin;
09706                         Vpmg_splineSelect(srffm, acc, gpos, thee->
09707                             splineWin, irad,
09708                             atom, tgrad);
09709                         fmag = induced->data[IJK(i,j,k)];
09710                         fmag *= nlinduced->data[IJK(i,j,k)];

```

```

09709             fmag *= thee->kappa[IJK(i,j,k)];
09710             force[0] += (zkappa2*fmag*tgrad[0]);
09711             force[1] += (zkappa2*fmag*tgrad[1]);
09712             force[2] += (zkappa2*fmag*tgrad[2]);
09713         }
09714     } /* k loop */
09715 } /* j loop */
09716 } /* i loop */
09717 }
09718
09719 force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
09720 force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
09721 force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
09722 }
09723
09724 VPUBLIC void Vpmg_dbMutualPolForce(Vpmg *thee, Vgrid *induced,
09725                                   Vgrid *nlinduced, int atomID,
09726                                   double force[3]) {
09727
09728     Vatom *atom;
09729     Vacc *acc;
09730     Vpbe *pbe;
09731     Vsurf_Meth srfm;
09732
09733     double *apos, position[3], arad, hx, hy, hzed, izmagic, deps, depsi;
09734     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2, epsp;
09735     double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
09736     double *u, *unl, Hxijk, Hyijk, Hzijk, Hximljk, Hyijmlk, Hzijkm1;
09737     double dHxijk[3], dHyijk[3], dHzijk[3], dHximljk[3], dHyijmlk[3];
09738     double dHzijkml[3];
09739     int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
09740
09741     VASSERT(thee != VNULL); /* PMG object with PBE info. */
09742     VASSERT(induced != VNULL); /* potential due to induced dipoles.*/
09743     VASSERT(nlinduced != VNULL); /* potential due to non-local induced dipoles.*/
09744
09745     acc = thee->pbe->acc;
09746     srfm = thee->surfMeth;
09747     atom = Valist_getAtom(thee->pbe->alist, atomID);
09748     VASSERT (atom->partID != 0); /* all atoms must be in the same partition.*/
09749     apos = Vatom_getPosition(atom);
09750     arad = Vatom_getRadius(atom);
09751
09752     /* Reset force */
09753     force[0] = 0.0;
09754     force[1] = 0.0;
09755     force[2] = 0.0;
09756
09757     /* Get PBE info */
09758     pbe = thee->pbe;
09759     acc = pbe->acc;
09760     epsp = Vpbe_getSoluteDiel(pbe);
09761     epsw = Vpbe_getSolventDiel(pbe);
09762     kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*
Vunit_kb;
09763     izmagic = 1.0/Vpbe_getZmagic(pbe);
09764
09765     deps = (epsw - epsp);
09766     depsi = 1.0/deps;
09767     VASSERT (VABS(deps) > VPMGSMALL);
09768
09769     /* Mesh info */
09770     nx = thee->pmgp->nx;
09771     ny = thee->pmgp->ny;
09772     nz = thee->pmgp->nz;
09773     hx = thee->pmgp->hx;
09774     hy = thee->pmgp->hy;
09775     hzed = thee->pmgp->hzed;
09776     xlen = thee->pmgp->xlen;
09777     ylen = thee->pmgp->ylen;
09778     zlen = thee->pmgp->zlen;
09779     xmin = thee->pmgp->xmin;
09780     ymin = thee->pmgp->ymin;
09781     zmin = thee->pmgp->zmin;
09782     xmax = thee->pmgp->xmax;
09783     ymax = thee->pmgp->ymax;
09784     zmax = thee->pmgp->zmax;
09785     u = induced->data;
09786     unl = nlinduced->data;
09787
09788     /* Make sure we're on the grid */

```

```

09789     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
09790         (apos[1]<=ymin) || (apos[1]>=ymax) || \
09791         (apos[2]<=zmin) || (apos[2]>=zmax)) {
09792         Vnm_print(2, "Vpmg_dbMutualPolForce: Atom at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):\n",
apos[0], apos[1], apos[2]);
09793         Vnm_print(2, "Vpmg_dbMutualPolForce:      xmin = %g, xmax = %g\n", xmin, xmax);
09794         Vnm_print(2, "Vpmg_dbMutualPolForce:      ymin = %g, ymax = %g\n", ymin, ymax);
09795         Vnm_print(2, "Vpmg_dbMutualPolForce:      zmin = %g, zmax = %g\n", zmin, zmax);
09796         fflush(stderr);
09797     } else {
09798
09799         /* Convert the atom position to grid reference frame */
09800         position[0] = apos[0] - xmin;
09801         position[1] = apos[1] - ymin;
09802         position[2] = apos[2] - zmin;
09803
09804         /* Integrate over points within this atom's (inflated) radius */
09805         rtot = (arad + thee->splineWin);
09806         rtot2 = VSQR(rtot);
09807         dx = rtot/hx;
09808         imin = (int)floor((position[0]-rtot)/hx);
09809         if (imin < 1) {
09810             Vnm_print(2, "Vpmg_dbMutualPolForce: Atom %d off grid!\n", atomID);
09811             return;
09812         }
09813         imax = (int)ceil((position[0]+rtot)/hx);
09814         if (imax > (nx-2)) {
09815             Vnm_print(2, "Vpmg_dbMutualPolForce: Atom %d off grid!\n", atomID);
09816             return;
09817         }
09818         jmin = (int)floor((position[1]-rtot)/hy);
09819         if (jmin < 1) {
09820             Vnm_print(2, "Vpmg_dbMutualPolForce: Atom %d off grid!\n", atomID);
09821             return;
09822         }
09823         jmax = (int)ceil((position[1]+rtot)/hy);
09824         if (jmax > (ny-2)) {
09825             Vnm_print(2, "Vpmg_dbMutualPolForce: Atom %d off grid!\n", atomID);
09826             return;
09827         }
09828         kmin = (int)floor((position[2]-rtot)/hz);
09829         if (kmin < 1) {
09830             Vnm_print(2, "Vpmg_dbMutualPolForce: Atom %d off grid!\n", atomID);
09831             return;
09832         }
09833         kmax = (int)ceil((position[2]+rtot)/hz);
09834         if (kmax > (nz-2)) {
09835             Vnm_print(2, "Vpmg_dbMutualPolForce: Atom %d off grid!\n", atomID);
09836             return;
09837         }
09838         for (i=imin; i<=imax; i++) {
09839             for (j=jmin; j<=jmax; j++) {
09840                 for (k=kmin; k<=kmax; k++) {
09841                     /* i,j,k */
09842                     gpos[0] = (i+0.5)*hx + xmin;
09843                     gpos[1] = j*hy + ymin;
09844                     gpos[2] = k*hz + zmin;
09845                     Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
09846                     Vpmg_splineSelect(srffm, acc, gpos, thee->
splineWin, 0.,
09847                                     atom, dHxijk);
09848                     for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
09849                     gpos[0] = i*hx + xmin;
09850                     gpos[1] = (j+0.5)*hy + ymin;
09851                     gpos[2] = k*hz + zmin;
09852                     Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
09853                     Vpmg_splineSelect(srffm, acc, gpos, thee->
splineWin, 0.,
09854                                     atom, dHyijk);
09855                     for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
09856                     gpos[0] = i*hx + xmin;
09857                     gpos[1] = j*hy + ymin;
09858                     gpos[2] = (k+0.5)*hz + zmin;
09859                     Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
09860                     Vpmg_splineSelect(srffm, acc, gpos, thee->
splineWin, 0.,
09861                                     atom, dHzijk);
09862                     for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
09863                     /* i-1,j,k */
09864                     gpos[0] = (i-0.5)*hx + xmin;
09865                     gpos[1] = j*hy + ymin;

```



```

09866         gpos[2] = k*hzed + zmin;
09867         Hximljk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
09868         Vpmg_splineSelect(srfm, acc, gpos, thee->
splineWin, 0.,
09869             atom, dHximljk);
09870         for (l=0; l<3; l++) dHximljk[l] *= Hximljk;
09871         /* i,j-1,k */
09872         gpos[0] = i*hx + xmin;
09873         gpos[1] = (j-0.5)*hy + ymin;
09874         gpos[2] = k*hzed + zmin;
09875         Hyijm1k = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
09876         Vpmg_splineSelect(srfm, acc, gpos, thee->
splineWin, 0.,
09877             atom, dHyijm1k);
09878         for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;
09879         /* i,j,k-1 */
09880         gpos[0] = i*hx + xmin;
09881         gpos[1] = j*hy + ymin;
09882         gpos[2] = (k-0.5)*hzed + zmin;
09883         Hzijkm1 = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
09884         Vpmg_splineSelect(srfm, acc, gpos, thee->
splineWin, 0.,
09885             atom, dHzijkm1);
09886         for (l=0; l<3; l++) dHzijkm1[l] *= Hzijkm1;
09887         dbFmag = unl[IJK(i,j,k)];
09888         tgrad[0] =
09889             (dHxijk[0] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09890             + dHximljk[0] * (u[IJK(i-1,j,k)]-u[IJK(i,j,k)])) /VSQR(hx)
09891             + (dHyijk[0] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09892             + dHyijm1k[0] * (u[IJK(i,j-1,k)]-u[IJK(i,j,k)])) /VSQR(hy)
09893             + (dHzijk[0] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
09894             + dHzijkm1[0] * (u[IJK(i,j,k-1)]-u[IJK(i,j,k)])) /VSQR(hzed);
09895         tgrad[1] =
09896             (dHxijk[1] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09897             + dHximljk[1] * (u[IJK(i-1,j,k)]-u[IJK(i,j,k)])) /VSQR(hx)
09898             + (dHyijk[1] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09899             + dHyijm1k[1] * (u[IJK(i,j-1,k)]-u[IJK(i,j,k)])) /VSQR(hy)
09900             + (dHzijk[1] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
09901             + dHzijkm1[1] * (u[IJK(i,j,k-1)]-u[IJK(i,j,k)])) /VSQR(hzed);
09902         tgrad[2] =
09903             (dHxijk[2] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09904             + dHximljk[2] * (u[IJK(i-1,j,k)]-u[IJK(i,j,k)])) /VSQR(hx)
09905             + (dHyijk[2] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09906             + dHyijm1k[2] * (u[IJK(i,j-1,k)]-u[IJK(i,j,k)])) /VSQR(hy)
09907             + (dHzijk[2] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
09908             + dHzijkm1[2] * (u[IJK(i,j,k-1)]-u[IJK(i,j,k)])) /VSQR(hzed);
09909         force[0] += (dbFmag*tgrad[0]);
09910         force[1] += (dbFmag*tgrad[1]);
09911         force[2] += (dbFmag*tgrad[2]);
09912     } /* k loop */
09913 } /* j loop */
09914 } /* i loop */
09915
09916 force[0] = -force[0]*hx*hy*hzed*deps*0.5*izmagic;
09917 force[1] = -force[1]*hx*hy*hzed*deps*0.5*izmagic;
09918 force[2] = -force[2]*hx*hy*hzed*deps*0.5*izmagic;
09919 }
09920 }
09921
09922 #endif /* if defined(WITH_TINKER) */
09923
09924 VPRIVATE void fillCoefSpline4(Vpmg *thee) {
09925     Valist *alist;
09926     Vpbe *pbe;
09927     Vatom *atom;
09928     double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr, dist2;
09929     double xlen, ylen, zlen, position[3], itot, stot, ictot, ictot2, sctot;
09930     double irad, dx, dy, dz, epsw, epsp, w2i;
09931     double hx, hy, hzed, *apos, arad, sctot2;
09932     double dx2, dy2, dz2, stot2, itot2, rtot, rtot2, splineWin;
09933     double dist, value, denom, sm, sm2, sm3, sm4, sm5, sm6, sm7;
09934     double e, e2, e3, e4, e5, e6, e7;
09935     double b, b2, b3, b4, b5, b6, b7;
09936     double c0, c1, c2, c3, c4, c5, c6, c7;
09937     double ic0, ic1, ic2, ic3, ic4, ic5, ic6, ic7;
09938     int i, j, k, nx, ny, nz, iatom;
09939     int imin, imax, jmin, jmax, kmin, kmax;
09940
09941     VASSERT(thee != VNULL);
09942     splineWin = thee->splineWin;
09943

```

```

09944
09945 /* Get PBE info */
09946 pbe = thee->pbe;
09947 alist = pbe->alist;
09948 irad = Vpbe_getMaxIonRadius(pbe);
09949 ionstr = Vpbe_getBulkIonicStrength(pbe);
09950 epsw = Vpbe_getSolventDiel(pbe);
09951 epsp = Vpbe_getSoluteDiel(pbe);
09952
09953 /* Mesh info */
09954 nx = thee->pmgp->nx;
09955 ny = thee->pmgp->ny;
09956 nz = thee->pmgp->nz;
09957 hx = thee->pmgp->hx;
09958 hy = thee->pmgp->hy;
09959 hzed = thee->pmgp->hzed;
09960
09961 /* Define the total domain size */
09962 xlen = thee->pmgp->xlen;
09963 ylen = thee->pmgp->ylen;
09964 zlen = thee->pmgp->zlen;
09965
09966 /* Define the min/max dimensions */
09967 xmin = thee->pmgp->xcent - (xlen/2.0);
09968 ymin = thee->pmgp->ycent - (ylen/2.0);
09969 zmin = thee->pmgp->zcent - (zlen/2.0);
09970 xmax = thee->pmgp->xcent + (xlen/2.0);
09971 ymax = thee->pmgp->ycent + (ylen/2.0);
09972 zmax = thee->pmgp->zcent + (zlen/2.0);
09973
09974 /* This is a floating point parameter related to the non-zero nature of the
09975  * bulk ionic strength. If the ionic strength is greater than zero; this
09976  * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
09977  * Otherwise, this parameter is set to 0.0 */
09978 if (ionstr > VPMGSMALL) ionmask = 1.0;
09979 else ionmask = 0.0;
09980
09981 /* Reset the kappa, epsx, epsy, and epsz arrays */
09982 for (i=0; i<(nx*ny*nz); i++) {
09983     thee->kappa[i] = 1.0;
09984     thee->epsx[i] = 1.0;
09985     thee->epsy[i] = 1.0;
09986     thee->epsz[i] = 1.0;
09987 }
09988
09989 /* Loop through the atoms and do assign the dielectric */
09990 for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
09991
09992     atom = Valist_getAtom(alist, iatom);
09993     apos = Vatom_getPosition(atom);
09994     arad = Vatom_getRadius(atom);
09995
09996     b = arad - splineWin;
09997     e = arad + splineWin;
09998     e2 = e * e;
09999     e3 = e2 * e;
10000     e4 = e3 * e;
10001     e5 = e4 * e;
10002     e6 = e5 * e;
10003     e7 = e6 * e;
10004     b2 = b * b;
10005     b3 = b2 * b;
10006     b4 = b3 * b;
10007     b5 = b4 * b;
10008     b6 = b5 * b;
10009     b7 = b6 * b;
10010     denom = e7 - 7.0*b*e6 + 21.0*b2*e5 - 35.0*e4*b3
10011             + 35.0*e3*b4 - 21.0*b5*e2 + 7.0*e*b6 - b7;
10012     c0 = b4*(35.0*e3 - 21.0*b*e2 + 7*e*b2 - b3)/denom;
10013     c1 = -140.0*b3*e3/denom;
10014     c2 = 210.0*e2*b2*(e + b)/denom;
10015     c3 = -140.0*e*b*(e2 + 3.0*b*e + b2)/denom;
10016     c4 = 35.0*(e3 + 9.0*b*e2 + 9.0*e*b2 + b3)/denom;
10017     c5 = -84.0*(e2 + 3.0*b*e + b2)/denom;
10018     c6 = 70.0*(e + b)/denom;
10019     c7 = -20.0/denom;
10020
10021     b = irad + arad - splineWin;
10022     e = irad + arad + splineWin;
10023     e2 = e * e;
10024     e3 = e2 * e;

```

```

10025     e4 = e3 * e;
10026     e5 = e4 * e;
10027     e6 = e5 * e;
10028     e7 = e6 * e;
10029     b2 = b * b;
10030     b3 = b2 * b;
10031     b4 = b3 * b;
10032     b5 = b4 * b;
10033     b6 = b5 * b;
10034     b7 = b6 * b;
10035     denom = e7 - 7.0*b*e6 + 21.0*b2*e5 - 35.0*e4*b3
10036             + 35.0*e3*b4 - 21.0*b5*e2 + 7.0*e*b6 - b7;
10037     ic0 = b4*(35.0*e3 - 21.0*b*e2 + 7*e*b2 - b3)/denom;
10038     ic1 = -140.0*b3*e3/denom;
10039     ic2 = 210.0*e2*b2*(e + b)/denom;
10040     ic3 = -140.0*e*b*(e2 + 3.0*b*e + b2)/denom;
10041     ic4 = 35.0*(e3 + 9.0*b*e2 + 9.0*e*b2 + b3)/denom;
10042     ic5 = -84.0*(e2 + 3.0*b*e + b2)/denom;
10043     ic6 = 70.0*(e + b)/denom;
10044     ic7 = -20.0/denom;
10045
10046     /* Make sure we're on the grid */
10047     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
10048         (apos[1]<=ymin) || (apos[1]>=ymax) || \
10049         (apos[2]<=zmin) || (apos[2]>=zmax)) {
10050         if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
10051             (thee->pmgp->bcfl != BCFL_MAP)) {
10052             Vnm_print(2, "Vpmg_fillco: Atom #d at (%4.3f, %4.3f, \
10053 %4.3f) is off the mesh (ignoring):\n",
10054                 iatom, apos[0], apos[1], apos[2]);
10055             Vnm_print(2, "Vpmg_fillco:   xmin = %g, xmax = %g\n",
10056                 xmin, xmax);
10057             Vnm_print(2, "Vpmg_fillco:   ymin = %g, ymax = %g\n",
10058                 ymin, ymax);
10059             Vnm_print(2, "Vpmg_fillco:   zmin = %g, zmax = %g\n",
10060                 zmin, zmax);
10061         }
10062         fflush(stderr);
10063
10064     } else if (arad > VPMGSMALL) { /* if we're on the mesh */
10065
10066         /* Convert the atom position to grid reference frame */
10067         position[0] = apos[0] - xmin;
10068         position[1] = apos[1] - ymin;
10069         position[2] = apos[2] - zmin;
10070
10071         /* MARK ION ACCESSIBILITY AND DIELECTRIC VALUES FOR LATER
10072          * ASSIGNMENT (Steps #1-3) */
10073         itot = irad + arad + splineWin;
10074         itot2 = VSQR(itot);
10075         ictot = VMAX2(0, (irad + arad - splineWin));
10076         ictot2 = VSQR(ictot);
10077         stot = arad + splineWin;
10078         stot2 = VSQR(stot);
10079         sctot = VMAX2(0, (arad - splineWin));
10080         sctot2 = VSQR(sctot);
10081
10082         /* We'll search over grid points which are in the greater of
10083          * these two radii */
10084         rtot = VMAX2(itot, stot);
10085         rtot2 = VMAX2(itot2, stot2);
10086         dx = rtot + 0.5*hx;
10087         dy = rtot + 0.5*hy;
10088         dz = rtot + 0.5*hzed;
10089         imin = VMAX2(0, (int)floor((position[0] - dx)/hx));
10090         imax = VMIN2(nx-1, (int)ceil((position[0] + dx)/hx));
10091         jmin = VMAX2(0, (int)floor((position[1] - dy)/hy));
10092         jmax = VMIN2(ny-1, (int)ceil((position[1] + dy)/hy));
10093         kmin = VMAX2(0, (int)floor((position[2] - dz)/hzed));
10094         kmax = VMIN2(nz-1, (int)ceil((position[2] + dz)/hzed));
10095         for (i=imin; i<=imax; i++) {
10096             dx2 = VSQR(position[0] - hx*i);
10097             for (j=jmin; j<=jmax; j++) {
10098                 dy2 = VSQR(position[1] - hy*j);
10099                 for (k=kmin; k<=kmax; k++) {
10100                     dz2 = VSQR(position[2] - k*hzed);
10101
10102                     /* ASSIGN CCF */
10103                     if (thee->kappa[IJK(i,j,k)] > VPMGSMALL) {
10104                         dist2 = dz2 + dy2 + dx2;
10105                         if (dist2 >= itot2) {

```

```

10106         ;
10107     }
10108     if (dist2 <= ictot2) {
10109         thee->kappa[IJK(i,j,k)] = 0.0;
10110     }
10111     if ((dist2 < itot2) && (dist2 > ictot2)) {
10112         dist = VSQRT(dist2);
10113         sm = dist;
10114         sm2 = dist2;
10115         sm3 = sm2 * sm;
10116         sm4 = sm3 * sm;
10117         sm5 = sm4 * sm;
10118         sm6 = sm5 * sm;
10119         sm7 = sm6 * sm;
10120         value = ic0 + ic1*sm + ic2*sm2 + ic3*sm3
10121             + ic4*sm4 + ic5*sm5 + ic6*sm6 + ic7*sm7;
10122         if (value > 1.0) {
10123             value = 1.0;
10124         } else if (value < 0.0){
10125             value = 0.0;
10126         }
10127         thee->kappa[IJK(i,j,k)] *= value;
10128     }
10129 }
10130
10131 /* ASSIGN A1CF */
10132 if (thee->epsx[IJK(i,j,k)] > VPMGSMALL) {
10133     dist2 = dz2+dy2+VSQR(position[0]-(i+0.5)*hx);
10134     if (dist2 >= stot2) {
10135         thee->epsx[IJK(i,j,k)] *= 1.0;
10136     }
10137     if (dist2 <= sctot2) {
10138         thee->epsx[IJK(i,j,k)] = 0.0;
10139     }
10140     if ((dist2 > sctot2) && (dist2 < stot2)) {
10141         dist = VSQRT(dist2);
10142         sm = dist;
10143         sm2 = VSQR(sm);
10144         sm3 = sm2 * sm;
10145         sm4 = sm3 * sm;
10146         sm5 = sm4 * sm;
10147         sm6 = sm5 * sm;
10148         sm7 = sm6 * sm;
10149         value = c0 + c1*sm + c2*sm2 + c3*sm3
10150             + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
10151         if (value > 1.0) {
10152             value = 1.0;
10153         } else if (value < 0.0){
10154             value = 0.0;
10155         }
10156         thee->epsx[IJK(i,j,k)] *= value;
10157     }
10158 }
10159
10160 /* ASSIGN A2CF */
10161 if (thee->epsy[IJK(i,j,k)] > VPMGSMALL) {
10162     dist2 = dz2+dx2+VSQR(position[1]-(j+0.5)*hy);
10163     if (dist2 >= stot2) {
10164         thee->epsy[IJK(i,j,k)] *= 1.0;
10165     }
10166     if (dist2 <= sctot2) {
10167         thee->epsy[IJK(i,j,k)] = 0.0;
10168     }
10169     if ((dist2 > sctot2) && (dist2 < stot2)) {
10170         dist = VSQRT(dist2);
10171         sm = dist;
10172         sm2 = VSQR(sm);
10173         sm3 = sm2 * sm;
10174         sm4 = sm3 * sm;
10175         sm5 = sm4 * sm;
10176         sm6 = sm5 * sm;
10177         sm7 = sm6 * sm;
10178         value = c0 + c1*sm + c2*sm2 + c3*sm3
10179             + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
10180         if (value > 1.0) {
10181             value = 1.0;
10182         } else if (value < 0.0){
10183             value = 0.0;
10184         }
10185         thee->epsy[IJK(i,j,k)] *= value;
10186     }

```

```

10187     }
10188
10189     /* ASSIGN A3CF */
10190     if (thee->epsz[IJK(i,j,k)] > VPMGSMALL) {
10191         dist2 = dy2+dx2+VSQR(position[2]-(k+0.5)*hzed);
10192         if (dist2 >= stot2) {
10193             thee->epsz[IJK(i,j,k)] *= 1.0;
10194         }
10195         if (dist2 <= sctot2) {
10196             thee->epsz[IJK(i,j,k)] = 0.0;
10197         }
10198         if ((dist2 > sctot2) && (dist2 < stot2)) {
10199             dist = VSQRT(dist2);
10200             sm = dist;
10201             sm2 = dist2;
10202             sm3 = sm2 * sm;
10203             sm4 = sm3 * sm;
10204             sm5 = sm4 * sm;
10205             sm6 = sm5 * sm;
10206             sm7 = sm6 * sm;
10207             value = c0 + c1*sm + c2*sm2 + c3*sm3
10208                   + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
10209             if (value > 1.0) {
10210                 value = 1.0;
10211             } else if (value < 0.0) {
10212                 value = 0.0;
10213             }
10214             thee->epsz[IJK(i,j,k)] *= value;
10215         }
10216     }
10217
10218     } /* k loop */
10219 } /* j loop */
10220 } /* i loop */
10221 } /* endif (on the mesh) */
10222 } /* endfor (over all atoms) */
10223
10224 Vnm_print(0, "Vpmg_fillco: filling coefficient arrays\n");
10225 /* Interpret markings and fill the coefficient arrays */
10226 for (k=0; k<nz; k++) {
10227     for (j=0; j<ny; j++) {
10228         for (i=0; i<nx; i++) {
10229
10230             thee->kappa[IJK(i,j,k)] = ionmask*thee->kappa[IJK(i,j,k)];
10231             thee->epsx[IJK(i,j,k)] = (epsw-epsp)*thee->epsx[IJK(i,j,k)]
10232                                     + epsp;
10233             thee->epsy[IJK(i,j,k)] = (epsw-epsp)*thee->epsy[IJK(i,j,k)]
10234                                     + epsp;
10235             thee->epsz[IJK(i,j,k)] = (epsw-epsp)*thee->epsz[IJK(i,j,k)]
10236                                     + epsp;
10237
10238         } /* i loop */
10239     } /* j loop */
10240 } /* k loop */
10241 }
10242
10243 }
10244
10245 VPUBLIC void fillcoPermanentInduced(Vpmg *thee) {
10246
10247     Valist *alist;
10248     Vpbe *pbe;
10249     Vatom *atom;
10250     /* Conversions */
10251     double zmagic, f;
10252     /* Grid */
10253     double xmin, xmax, ymin, ymax, zmin, zmax;
10254     double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;
10255     double hx, hy, hzed, *apos;
10256     /* Multipole */
10257     double charge, *dipole, *quad;
10258     double c, ux, uy, uz, qxx, qyx, qyy, qzx, qzy, qzz, qave;
10259     /* B-spline weights */
10260     double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz;
10261     double mi, mj, mk;
10262     /* Loop variables */
10263     int i, ii, jj, kk, nx, ny, nz, iatom;
10264     int im2, im1, ip1, ip2, jm2, jm1, jpl, jp2, km2, km1, kp1, kp2;
10265
10266     VASSERT(thee != VNULL);
10267

```

```

10268      /* Get PBE info */
10269      pbe = thee->pbe;
10270      alist = pbe->alist;
10271      zmagic = Vpbe_getZmagic(pbe);
10272
10273      /* Mesh info */
10274      nx = thee->pmgp->nx;
10275      ny = thee->pmgp->ny;
10276      nz = thee->pmgp->nz;
10277      hx = thee->pmgp->hx;
10278      hy = thee->pmgp->hy;
10279      hzed = thee->pmgp->hzed;
10280
10281      /* Conversion */
10282      f = zmagic/(hx*hy*hzed);
10283
10284      /* Define the total domain size */
10285      xlen = thee->pmgp->xlen;
10286      ylen = thee->pmgp->ylen;
10287      zlen = thee->pmgp->zlen;
10288
10289      /* Define the min/max dimensions */
10290      xmin = thee->pmgp->xcent - (xlen/2.0);
10291      ymin = thee->pmgp->ycent - (ylen/2.0);
10292      zmin = thee->pmgp->zcent - (zlen/2.0);
10293      xmax = thee->pmgp->xcent + (xlen/2.0);
10294      ymax = thee->pmgp->ycent + (ylen/2.0);
10295      zmax = thee->pmgp->zcent + (zlen/2.0);
10296
10297      /* Fill in the source term (permanent atomic multipoles
10298      and induced dipoles) */
10299      Vnm_print(0, "fillcoPermanentInduced: filling in source term.\n");
10300      for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
10301
10302          atom = Valist_getAtom(alist, iatom);
10303          apos = Vatom_getPosition(atom);
10304
10305          c = Vatom_getCharge(atom)*f;
10306
10307      #if defined(WITH_TINKER)
10308          dipole = Vatom_getDipole(atom);
10309          ux = dipole[0]/hx*f;
10310          uy = dipole[1]/hy*f;
10311          uz = dipole[2]/hzed*f;
10312          dipole = Vatom_getInducedDipole(atom);
10313          ux = ux + dipole[0]/hx*f;
10314          uy = uy + dipole[1]/hy*f;
10315          uz = uz + dipole[2]/hzed*f;
10316          quad = Vatom_getQuadrupole(atom);
10317          qxx = (1.0/3.0)*quad[0]/(hx*hx)*f;
10318          qyx = (2.0/3.0)*quad[3]/(hx*hy)*f;
10319          qyy = (1.0/3.0)*quad[4]/(hy*hy)*f;
10320          qzx = (2.0/3.0)*quad[6]/(hzed*hx)*f;
10321          qzy = (2.0/3.0)*quad[7]/(hzed*hy)*f;
10322          qzz = (1.0/3.0)*quad[8]/(hzed*hzed)*f;
10323      #else
10324          ux = 0.0;
10325          uy = 0.0;
10326          uz = 0.0;
10327          qxx = 0.0;
10328          qyx = 0.0;
10329          qyy = 0.0;
10330          qzx = 0.0;
10331          qzy = 0.0;
10332          qzz = 0.0;
10333      #endif /* if defined(WITH_TINKER) */
10334
10335      /* Make sure we're on the grid */
10336      if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx)) || \
10337          (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy)) || \
10338          (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
10339          Vnm_print(2, "fillcoPermanentMultipole: Atom #%d at (%4.3f, %4.3f, %4.3f) is off the mesh
10340      (ignoring this atom):\n", iatom, apos[0], apos[1], apos[2]);
10341          Vnm_print(2, "fillcoPermanentMultipole: xmin = %g, xmax = %g\n", xmin, xmax);
10342          Vnm_print(2, "fillcoPermanentMultipole: ymin = %g, ymax = %g\n", ymin, ymax);
10343          Vnm_print(2, "fillcoPermanentMultipole: zmin = %g, zmax = %g\n", zmin, zmax);
10344          fflush(stderr);
10345      } else {
10346          /* Convert the atom position to grid reference frame */
10347          position[0] = apos[0] - xmin;

```

```

10348     position[1] = apos[1] - ymin;
10349     position[2] = apos[2] - zmin;
10350
10351     /* Figure out which vertices we're next to */
10352     ifloat = position[0]/hx;
10353     jfloat = position[1]/hy;
10354     kfloat = position[2]/hz;
10355
10356     ip1 = (int)ceil(ifloat);
10357     ip2 = ip1 + 2;
10358     im1 = (int)floor(ifloat);
10359     im2 = im1 - 2;
10360     jp1 = (int)ceil(jfloat);
10361     jp2 = jp1 + 2;
10362     jm1 = (int)floor(jfloat);
10363     jm2 = jm1 - 2;
10364     kp1 = (int)ceil(kfloat);
10365     kp2 = kp1 + 2;
10366     km1 = (int)floor(kfloat);
10367     km2 = km1 - 2;
10368
10369     /* This step shouldn't be necessary, but it saves nasty debugging
10370      * later on if something goes wrong */
10371     ip2 = VMIN2(ip2,nx-1);
10372     ip1 = VMIN2(ip1,nx-1);
10373     im1 = VMAX2(im1,0);
10374     im2 = VMAX2(im2,0);
10375     jp2 = VMIN2(jp2,ny-1);
10376     jp1 = VMIN2(jp1,ny-1);
10377     jm1 = VMAX2(jm1,0);
10378     jm2 = VMAX2(jm2,0);
10379     kp2 = VMIN2(kp2,nz-1);
10380     kp1 = VMIN2(kp1,nz-1);
10381     km1 = VMAX2(km1,0);
10382     km2 = VMAX2(km2,0);
10383
10384     /* Now assign fractions of the charge to the nearby verts */
10385     for (ii=im2; ii<=ip2; ii++) {
10386         mi = VFCHI4(ii,ifloat);
10387         mx = bspline4(mi);
10388         dmx = dbspline4(mi);
10389         d2mx = d2bspline4(mi);
10390         for (jj=jm2; jj<=jp2; jj++) {
10391             mj = VFCHI4(jj,jfloat);
10392             my = bspline4(mj);
10393             dmy = dbspline4(mj);
10394             d2my = d2bspline4(mj);
10395             for (kk=km2; kk<=kp2; kk++) {
10396                 mk = VFCHI4(kk,kfloat);
10397                 mz = bspline4(mk);
10398                 dmz = dbspline4(mk);
10399                 d2mz = d2bspline4(mk);
10400                 charge = mx*my*mz*c -
10401                     dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz +
10402                     d2mx*my*mz*qxx +
10403                     dmx*dmy*mz*qyx + mx*d2my*mz*qyy +
10404                     dmx*my*dmz*qzx + mx*dmy*dmz*qzy + mx*my*d2mz*qzz;
10405                 thee->charge[IJK(ii,jj,kk)] += charge;
10406             }
10407         }
10408     }
10409 }
10410 } /* endif (on the mesh) */
10411
10412 } /* endfor (each atom) */
10413 }
10414
10415 VPRIVATE void fillCoefSpline3(Vpmg *thee) {
10416     Valist *alist;
10417     Vpbe *pbe;
10418     Vatom *atom;
10419     double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr, dist2;
10420     double xlen, ylen, zlen, position[3], itot, stot, ictot, ictot2, sctot;
10421     double irad, dx, dy, dz, epsw, epsp, w2i;
10422     double hx, hy, hzed, *apos, arad, sctot2;
10423     double dx2, dy2, dz2, stot2, itot2, rtot, rtot2, splineWin;
10424     double dist, value, denom, sm, sm2, sm3, sm4, sm5;
10425     double e, e2, e3, e4, e5;
10426     double b, b2, b3, b4, b5;
10427     double c0, c1, c2, c3, c4, c5;

```

```

10429     double ic0, ic1, ic2, ic3, ic4, ic5;
10430     int i, j, k, nx, ny, nz, iatom;
10431     int imin, imax, jmin, jmax, kmin, kmax;
10432
10433     VASSERT(thee != VNULL);
10434     splineWin = thee->splineWin;
10435
10436     /* Get PBE info */
10437     pbe = thee->pbe;
10438     alist = pbe->alist;
10439     irad = Vpbe_getMaxIonRadius(pbe);
10440     ionstr = Vpbe_getBulkIonicStrength(pbe);
10441     epsw = Vpbe_getSolventDiel(pbe);
10442     epsp = Vpbe_getSoluteDiel(pbe);
10443
10444     /* Mesh info */
10445     nx = thee->pmgp->nx;
10446     ny = thee->pmgp->ny;
10447     nz = thee->pmgp->nz;
10448     hx = thee->pmgp->hx;
10449     hy = thee->pmgp->hy;
10450     hzed = thee->pmgp->hzed;
10451
10452     /* Define the total domain size */
10453     xlen = thee->pmgp->xlen;
10454     ylen = thee->pmgp->ylen;
10455     zlen = thee->pmgp->zlen;
10456
10457     /* Define the min/max dimensions */
10458     xmin = thee->pmgp->xcent - (xlen/2.0);
10459     ymin = thee->pmgp->ycent - (ylen/2.0);
10460     zmin = thee->pmgp->zcent - (zlen/2.0);
10461     xmax = thee->pmgp->xcent + (xlen/2.0);
10462     ymax = thee->pmgp->ycent + (ylen/2.0);
10463     zmax = thee->pmgp->zcent + (zlen/2.0);
10464
10465     /* This is a floating point parameter related to the non-zero nature of the
10466      * bulk ionic strength. If the ionic strength is greater than zero; this
10467      * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
10468      * Otherwise, this parameter is set to 0.0 */
10469     if (ionstr > VPMGSMALL) ionmask = 1.0;
10470     else ionmask = 0.0;
10471
10472     /* Reset the kappa, epsx, epsy, and epsz arrays */
10473     for (i=0; i<(nx*ny*nz); i++) {
10474         thee->kappa[i] = 1.0;
10475         thee->epsx[i] = 1.0;
10476         thee->epsy[i] = 1.0;
10477         thee->epsz[i] = 1.0;
10478     }
10479
10480     /* Loop through the atoms and do assign the dielectric */
10481     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
10482
10483         atom = Valist_getAtom(alist, iatom);
10484         apos = Vatom_getPosition(atom);
10485         arad = Vatom_getRadius(atom);
10486
10487         b = arad - splineWin;
10488         e = arad + splineWin;
10489         e2 = e * e;
10490         e3 = e2 * e;
10491         e4 = e3 * e;
10492         e5 = e4 * e;
10493         b2 = b * b;
10494         b3 = b2 * b;
10495         b4 = b3 * b;
10496         b5 = b4 * b;
10497         denom = pow((e - b), 5.0);
10498         c0 = -10.0*e2*b3 + 5.0*e*b4 - b5;
10499         c1 = 30.0*e2*b2;
10500         c2 = -30.0*(e2*b + e*b2);
10501         c3 = 10.0*(e2 + 4.0*e*b + b2);
10502         c4 = -15.0*(e + b);
10503         c5 = 6;
10504         c0 = c0/denom;
10505         c1 = c1/denom;
10506         c2 = c2/denom;
10507         c3 = c3/denom;
10508         c4 = c4/denom;
10509         c5 = c5/denom;

```



```

10510
10511     b = irad + arad - splineWin;
10512     e = irad + arad + splineWin;
10513     e2 = e * e;
10514     e3 = e2 * e;
10515     e4 = e3 * e;
10516     e5 = e4 * e;
10517     b2 = b * b;
10518     b3 = b2 * b;
10519     b4 = b3 * b;
10520     b5 = b4 * b;
10521     denom = pow((e - b), 5.0);
10522     ic0 = -10.0*e2*b3 + 5.0*e*b4 - b5;
10523     ic1 = 30.0*e2*b2;
10524     ic2 = -30.0*(e2*b + e*b2);
10525     ic3 = 10.0*(e2 + 4.0*e*b + b2);
10526     ic4 = -15.0*(e + b);
10527     ic5 = 6;
10528     ic0 = c0/denom;
10529     ic1 = c1/denom;
10530     ic2 = c2/denom;
10531     ic3 = c3/denom;
10532     ic4 = c4/denom;
10533     ic5 = c5/denom;
10534
10535     /* Make sure we're on the grid */
10536     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
10537         (apos[1]<=ymin) || (apos[1]>=ymax) || \
10538         (apos[2]<=zmin) || (apos[2]>=zmax)) {
10539         if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
10540             (thee->pmgp->bcfl != BCFL_MAP)) {
10541             Vnm_print(2, "Vpmg_fillco: Atom #%d at (%4.3f, %4.3f, \
10542 %4.3f) is off the mesh (ignoring):\n",
10543                 iatom, apos[0], apos[1], apos[2]);
10544             Vnm_print(2, "Vpmg_fillco:      xmin = %g, xmax = %g\n",
10545                 xmin, xmax);
10546             Vnm_print(2, "Vpmg_fillco:      ymin = %g, ymax = %g\n",
10547                 ymin, ymax);
10548             Vnm_print(2, "Vpmg_fillco:      zmin = %g, zmax = %g\n",
10549                 zmin, zmax);
10550         }
10551         fflush(stderr);
10552
10553     } else if (arad > VPMGSMALL) { /* if we're on the mesh */
10554
10555         /* Convert the atom position to grid reference frame */
10556         position[0] = apos[0] - xmin;
10557         position[1] = apos[1] - ymin;
10558         position[2] = apos[2] - zmin;
10559
10560         /* MARK ION ACCESSIBILITY AND DIELECTRIC VALUES FOR LATER
10561          * ASSIGNMENT (Steps #1-3) */
10562         itot = irad + arad + splineWin;
10563         itot2 = VSQR(itot);
10564         ictot = VMAX2(0, (irad + arad - splineWin));
10565         ictot2 = VSQR(ictot);
10566         stot = arad + splineWin;
10567         stot2 = VSQR(stot);
10568         sctot = VMAX2(0, (arad - splineWin));
10569         sctot2 = VSQR(sctot);
10570
10571         /* We'll search over grid points which are in the greater of
10572          * these two radii */
10573         rtot = VMAX2(itot, stot);
10574         rtot2 = VMAX2(itot2, stot2);
10575         dx = rtot + 0.5*hx;
10576         dy = rtot + 0.5*hy;
10577         dz = rtot + 0.5*hzed;
10578         imin = VMAX2(0, (int)floor((position[0] - dx)/hx));
10579         imax = VMIN2(nx-1, (int)ceil((position[0] + dx)/hx));
10580         jmin = VMAX2(0, (int)floor((position[1] - dy)/hy));
10581         jmax = VMIN2(ny-1, (int)ceil((position[1] + dy)/hy));
10582         kmin = VMAX2(0, (int)floor((position[2] - dz)/hzed));
10583         kmax = VMIN2(nz-1, (int)ceil((position[2] + dz)/hzed));
10584         for (i=imin; i<=imax; i++) {
10585             dx2 = VSQR(position[0] - hx*i);
10586             for (j=jmin; j<=jmax; j++) {
10587                 dy2 = VSQR(position[1] - hy*j);
10588                 for (k=kmin; k<=kmax; k++) {
10589                     dz2 = VSQR(position[2] - k*hzed);
10590

```

```

10591      /* ASSIGN CCF */
10592      if (thee->kappa[IJK(i,j,k)] > VPMGSMALL) {
10593          dist2 = dz2 + dy2 + dx2;
10594          if (dist2 >= itot2) {
10595              ;
10596          }
10597          if (dist2 <= ictot2) {
10598              thee->kappa[IJK(i,j,k)] = 0.0;
10599          }
10600          if ((dist2 < itot2) && (dist2 > ictot2)) {
10601              dist = VSQRT(dist2);
10602              sm = dist;
10603              sm2 = dist2;
10604              sm3 = sm2 * sm;
10605              sm4 = sm3 * sm;
10606              sm5 = sm4 * sm;
10607              value = ic0 + ic1*sm + ic2*sm2 + ic3*sm3
10608                  + ic4*sm4 + ic5*sm5;
10609              if (value > 1.0) {
10610                  value = 1.0;
10611              } else if (value < 0.0){
10612                  value = 0.0;
10613              }
10614              thee->kappa[IJK(i,j,k)] *= value;
10615          }
10616      }
10617
10618      /* ASSIGN A1CF */
10619      if (thee->epsx[IJK(i,j,k)] > VPMGSMALL) {
10620          dist2 = dz2+dy2+VSQR(position[0]-(i+0.5)*hx);
10621          if (dist2 >= stot2) {
10622              thee->epsx[IJK(i,j,k)] *= 1.0;
10623          }
10624          if (dist2 <= sctot2) {
10625              thee->epsx[IJK(i,j,k)] = 0.0;
10626          }
10627          if ((dist2 > sctot2) && (dist2 < stot2)) {
10628              dist = VSQRT(dist2);
10629              sm = dist;
10630              sm2 = VSQR(sm);
10631              sm3 = sm2 * sm;
10632              sm4 = sm3 * sm;
10633              sm5 = sm4 * sm;
10634              value = c0 + c1*sm + c2*sm2 + c3*sm3
10635                  + c4*sm4 + c5*sm5;
10636              if (value > 1.0) {
10637                  value = 1.0;
10638              } else if (value < 0.0){
10639                  value = 0.0;
10640              }
10641              thee->epsx[IJK(i,j,k)] *= value;
10642          }
10643      }
10644
10645      /* ASSIGN A2CF */
10646      if (thee->epsy[IJK(i,j,k)] > VPMGSMALL) {
10647          dist2 = dz2+dx2+VSQR(position[1]-(j+0.5)*hy);
10648          if (dist2 >= stot2) {
10649              thee->epsy[IJK(i,j,k)] *= 1.0;
10650          }
10651          if (dist2 <= sctot2) {
10652              thee->epsy[IJK(i,j,k)] = 0.0;
10653          }
10654          if ((dist2 > sctot2) && (dist2 < stot2)) {
10655              dist = VSQRT(dist2);
10656              sm = dist;
10657              sm2 = VSQR(sm);
10658              sm3 = sm2 * sm;
10659              sm4 = sm3 * sm;
10660              sm5 = sm4 * sm;
10661              value = c0 + c1*sm + c2*sm2 + c3*sm3
10662                  + c4*sm4 + c5*sm5;
10663              if (value > 1.0) {
10664                  value = 1.0;
10665              } else if (value < 0.0){
10666                  value = 0.0;
10667              }
10668              thee->epsy[IJK(i,j,k)] *= value;
10669          }
10670      }
10671

```

```

10672             /* ASSIGN A3CF */
10673             if (thee->epsz[IJK(i,j,k)] > VPMGSMALL) {
10674                 dist2 = dy2+dx2+VSQR(position[2]-(k+0.5)*hzed);
10675                 if (dist2 >= stot2) {
10676                     thee->epsz[IJK(i,j,k)] *= 1.0;
10677                 }
10678                 if (dist2 <= sctot2) {
10679                     thee->epsz[IJK(i,j,k)] = 0.0;
10680                 }
10681                 if ((dist2 > sctot2) && (dist2 < stot2)) {
10682                     dist = VSQR(dist2);
10683                     sm = dist;
10684                     sm2 = dist2;
10685                     sm3 = sm2 * sm;
10686                     sm4 = sm3 * sm;
10687                     sm5 = sm4 * sm;
10688                     value = c0 + c1*sm + c2*sm2 + c3*sm3
10689                             + c4*sm4 + c5*sm5;
10690                     if (value > 1.0) {
10691                         value = 1.0;
10692                     } else if (value < 0.0) {
10693                         value = 0.0;
10694                     }
10695                     thee->epsz[IJK(i,j,k)] *= value;
10696                 }
10697             }
10698         } /* k loop */
10699     } /* j loop */
10700 } /* i loop */
10701 } /* endif (on the mesh) */
10702 } /* endfor (over all atoms) */
10703
10704 Vnm_print(0, "Vpmg_fillco: filling coefficient arrays\n");
10705 /* Interpret markings and fill the coefficient arrays */
10706 for (k=0; k<nz; k++) {
10707     for (j=0; j<ny; j++) {
10708         for (i=0; i<nx; i++) {
10709             thee->kappa[IJK(i,j,k)] = ionmask*thee->kappa[IJK(i,j,k)];
10710             thee->epsx[IJK(i,j,k)] = (epsw-epsp)*thee->epsx[IJK(i,j,k)]
10711                 + epsp;
10712             thee->epsy[IJK(i,j,k)] = (epsw-epsp)*thee->epsy[IJK(i,j,k)]
10713                 + epsp;
10714             thee->epsz[IJK(i,j,k)] = (epsw-epsp)*thee->epsz[IJK(i,j,k)]
10715                 + epsp;
10716         } /* i loop */
10717     } /* j loop */
10718 } /* k loop */
10719 }
10720
10721 VPRIVATE void bcolcomp(int *iparm, double *rparm, int *iwork, double *rwork,
10722     double *values, int *rowind, int *colptr, int *flag) {
10723     int nrow, ncol, nnzero, i;
10724     int nxc, nyc, nzc, nf, nc, narr, narrc, n_rpc;
10725     int n_iz, n_ipc, iretot, iintot;
10726     int nrw, niwk, nx, ny, nz, nlev, ierror, maxlev, mxlv;
10727     int mgcoar, mgdisc, mgsolv;
10728     int k_iz;
10729     int k_ipc, k_rpc, k_ac, k_cc, k_fc, k_pc;
10730
10731     WARN_UNTESTED;
10732
10733     // Decode some parameters
10734     nrw = VAT(iparm, 1);
10735     niwk = VAT(iparm, 2);
10736     nx = VAT(iparm, 3);
10737     ny = VAT(iparm, 4);
10738     nz = VAT(iparm, 5);
10739     nlev = VAT(iparm, 6);
10740
10741     // Some checks on input
10742     mxlv = Vmaxlev(nx, ny, nz);
10743
10744     // Basic grid sizes, etc.
10745     mgcoar = VAT(iparm, 18);
10746     mgdisc = VAT(iparm, 19);
10747     mgsolv = VAT(iparm, 21);

```

```

10753     Vmgsz(&mgscoar, &mgsdisc, &mgsolv,
10754           &nx, &ny, &nz,
10755           &nlev,
10756           &nxc, &nyc, &nzc,
10757           &nf, &nc,
10758           &narr, &narrc,
10759           &n_rpc, &n_iz, &n_ipc,
10760           &ioretot, &iintot);
10761
10762     // Split up the integer work array
10763     k_iz = 1;
10764     k_ipc = k_iz + n_iz;
10765
10766     // Split up the real work array
10767     k_rpc = 1;
10768     k_cc = k_rpc + n_rpc;
10769     k_fc = k_cc + narr;
10770     k_pc = k_fc + narr;
10771     k_ac = k_pc + 27*narrc;
10772
10773     bcolcomp2(iparm, rparam,
10774              &nx, &ny, &nz, RAT(iwork, k_iz),
10775              RAT(iwork, k_ipc), RAT(rwork, k_rpc),
10776              RAT(rwork, k_ac), RAT(rwork, k_cc),
10777              values, rowind, colptr, flag);
10778 }
10779
10780 VPRIVATE void bcolcomp2(int *iparm, double *rparam,
10781                          int *nx, int *ny, int *nz,
10782                          int *iz, int *ipc, double *rpc,
10783                          double *ac, double *cc, double *values,
10784                          int *rowind, int *colptr, int *flag) {
10785
10786     int nlev = 1;
10787     int lev = VAT(iparm, 6);
10788
10789     MAT2(iz, 50, nlev);
10790
10791     WARN_UNTESTED;
10792
10793     /*
10794     * Build the multigrid data structure in iz
10795     * THIS MAY HAVE BEEN DONE ALREADY, BUT IT'S OK TO DO IT AGAIN,
10796     * RIGHT?
10797     * call buildstr (nx,ny,nz,nlev,iz)
10798     *
10799     * We're interested in the finest level
10800     */
10801     bcolcomp3(nx, ny, nz,
10802              RAT(ipc, VAT2(iz, 5, lev)), RAT(rpc, VAT2(iz, 6, lev)),
10803              RAT(ac, VAT2(iz, 7, lev)), RAT(cc, VAT2(iz, 1, lev)),
10804              values, rowind, colptr, flag);
10805 }
10806
10807 /*****
10808 * Routine: bcolcomp3
10809 * Purpose: Build a column-compressed matrix in Harwell-Boeing format
10810 * Args:   flag 0 ==> Use Poisson operator only
10811 *         1 ==> Use linearization of full operator around current
10812 *         solution
10813 * Author:  Nathan Baker (mostly ripped off from Harwell-Boeing format
10814 *         documentation)
10815 *****/
10816 VPRIVATE void bcolcomp3(int *nx, int *ny, int *nz,
10817                          int *ipc, double *rpc,
10818                          double *ac, double *cc,
10819                          double *values, int *rowind, int *colptr, int *flag) {
10820
10821     MAT2(ac, *nx * *ny * *nz, 1);
10822
10823     WARN_UNTESTED;
10824
10825     bcolcomp4(nx, ny, nz,
10826              ipc, rpc,
10827              RAT2(ac, 1, 1), cc,
10828              RAT2(ac, 1, 2), RAT2(ac, 1, 3), RAT2(ac, 1, 4),
10829              values, rowind, colptr, flag);
10830 }
10831
10832
10833

```

```

10834 /*****
10835  * Routine:  bcolcomp4
10836  * Purpose:  Build a column-compressed matrix in Harwell-Boeing format
10837  * Args:    flag    0 ==> Use Poisson operator only
10838  *           1 ==> Use linearization of full operator around current
10839  *           solution
10840  * Author:   Nathan Baker (mostly ripped off from Harwell-Boeing format
10841  *           documentation)
10842  *****/
10843 VPRIVATE void bcolcomp4(int *nx, int *ny, int *nz,
10844                          int *ipc, double *rpc,
10845                          double *oC, double *cc, double *oE, double *oN, double *uC,
10846                          double *values, int *rowind, int *colptr, int *flag) {
10847
10848     int nxm2, nym2, nzm2;
10849     int ii, jj, kk, ll;
10850     int i, j, k, l;
10851     int inonz, irow, nn, nrow, ncol, nonz, irow, n;
10852
10853     int doit;
10854
10855     MAT3(oE, *nx, *ny, *nz);
10856     MAT3(oN, *nx, *ny, *nz);
10857     MAT3(uC, *nx, *ny, *nz);
10858     MAT3(cc, *nx, *ny, *nz);
10859     MAT3(oC, *nx, *ny, *nz);
10860
10861     WARN_UNTESTED;
10862
10863     // Get some column, row, and nonzero information
10864     n = *nx * *ny * *nz;
10865     nxm2 = *nx - 2;
10866     nym2 = *ny - 2;
10867     nzm2 = *nz - 2;
10868     nn = nxm2 * nym2 * nzm2;
10869     ncol = nn;
10870     nrow = nn;
10871     nonz = 7 * nn - 2 * nxm2 * nym2 - 2 * nxm2 - 2;
10872
10873     // Intialize some pointers
10874     inonz = 1;
10875
10876     /*
10877     * Run over the dimensions of the matrix (non-zero only in the interior
10878     * of the mesh
10879     */
10880     for (k=2; k<=*nz-1; k++) {
10881         // Offset the index to the output grid index
10882         kk = k - 1;
10883
10884         for (j=2; j<=*ny-1; j++) {
10885             // Offset the index to the output grid index
10886             jj = j - 1;
10887
10888             for (i=2; i<=*nx-1; i++) {
10889                 // Offset the index to the output grid index
10890                 ii = i - 1;
10891
10892                 // Get the output (i,j,k) row number in natural ordering
10893                 ll = (kk - 1) * nxm2 * nym2 + (jj - 1) * nxm2 + (ii - 1) + 1;
10894                 l = (k - 1) * *nx * *ny + (j - 1) * *nx + (i - 1) + 1;
10895
10896                 // Store where this column starts
10897                 VAT(colptr,ll) = inonz;
10898
10899                 // SUB-DIAGONAL 3
10900                 irow = ll - nxm2 * nym2;
10901                 irow = 1 - *nx * *ny;
10902
10903                 doit = (irow >= 1) && (irow <= nn);
10904                 doit = doit && (irow >= 1) && (irow <= n);
10905
10906                 if (doit) {
10907                     VAT(values, inonz) = -VAT3(uC, i, j, k-1);
10908                     VAT(rowind, inonz) = irow;
10909                     inonz++;
10910                 }
10911
10912
10913
10914                 // SUB-DIAGONAL 2

```

```

10915         iirow = ll - nxm2;
10916         irow = 1 - *nx;
10917
10918         doit = (iirow >= 1) && (iirow <= nn);
10919         doit = doit && (irow >= 1) && (irow <= n);
10920
10921         if (doit) {
10922             VAT(values, inonz) = -VAT3(oN, i, j-1, k);
10923             VAT(rowind, inonz) = iirow;
10924             inonz++;
10925         }
10926
10927
10928
10929         // SUB-DIAGONAL 1
10930         iirow = ll - 1;
10931         irow = 1 - 1;
10932
10933         doit = (iirow >= 1) && (iirow <= nn);
10934         doit = doit && (irow <= 1) && (irow <= n);
10935         if (doit) {
10936             VAT(values, inonz) = -VAT3(oE, i-1, j, k);
10937             VAT(rowind, inonz) = iirow;
10938             inonz++;
10939         }
10940
10941
10942
10943         // DIAGONAL
10944         iirow = ll;
10945         irow = 1;
10946
10947         if (*flag == 0) {
10948             VAT(values, inonz) = VAT3(oC, i, j, k);
10949         } else if (*flag == 1) {
10950             VAT(values, inonz) = VAT3(oC, i, j, k)
10951                               + VAT3(cc, i, j, k);
10952         } else {
10953             VABORT_MSG0("PMGF1");
10954         }
10955
10956         VAT(rowind, inonz) = iirow;
10957         inonz++;
10958
10959         // SUPER-DIAGONAL 1
10960         iirow = ll + 1;
10961         irow = 1 + 1;
10962         doit = (iirow >= 1) && (iirow <= nn);
10963         doit = doit && (irow >= 1) && (irow <= n);
10964         if (doit) {
10965             VAT(values, inonz) = -VAT3(oE, i, j, k);
10966             VAT(rowind, inonz) = iirow;
10967             inonz++;
10968         }
10969
10970
10971
10972         // SUPER-DIAGONAL 2
10973         iirow = ll + nxm2;
10974         irow = 1 + *nx;
10975         doit = (iirow >= 1) && (iirow <= nn);
10976         doit = doit && (irow >= 1) && (irow <= n);
10977         if (doit) {
10978             VAT(values, inonz) = -VAT3(oN, i, j, k);
10979             VAT(rowind, inonz) = iirow;
10980             inonz++;
10981         }
10982
10983
10984
10985         // SUPER-DIAGONAL 3
10986         iirow = ll + nxm2 * nym2;
10987         irow = 1 + *nx * *ny;
10988         doit = (iirow >= 1) && (iirow <= nn);
10989         doit = doit && (irow >= 1) && (irow <= n);
10990         if (doit) {
10991             VAT(values, inonz) = -VAT3(uC, i, j, k);
10992             VAT(rowind, inonz) = iirow;
10993             inonz++;
10994         }
10995     }

```

```

10996     }
10997 }
10998
10999 VAT(colptr, ncol + 1) = inonz;
11000
11001 if (inonz != (nonz + 1)) {
11002     VABORT_MSG2("BCOLCOMP4:  ERROR -- INONZ = %d, NONZ = %d", inonz, nonz);
11003 }
11004 }
11005
11006
11007
11008 VPRIVATE void pcolcomp(int *nrow, int *ncol, int *nnzero,
11009     double *values, int *rowind, int *colptr,
11010     char *path, char *title, char *mxttype) {
11011
11012     char key[] = "key";
11013     char ptrfmt[] = "(10I8)";
11014     char indfmt[] = "(10I8)";
11015     char valfmt[] = "(5E15.8)";
11016     char rhsfmt[] = "(5E15.8)";
11017
11018     int i, totcrd, ptrcrd, indcrd, valcrd, neltvl, rhscrd;
11019
11020     FILE *outFile;
11021
11022     WARN_UNTESTED;
11023
11024     // Open the file for reading
11025     outFile = fopen(path, "w");
11026
11027     // Set some default values
11028     ptrcrd = (int)(*ncol / 10 + 1) - 1;
11029     indcrd = (int)(*nnzero / 10 + 1) - 1;
11030     valcrd = (int)(*nnzero / 10 + 1) - 1;
11031     totcrd = ptrcrd + indcrd + valcrd;
11032     rhscrd = 0;
11033     neltvl = 0;
11034
11035     // Print the header
11036     fprintf(outFile, "%72s%8s\n",
11037         title, key);
11038     fprintf(outFile, "%14d%14d%14d%14d%14d\n",
11039         totcrd, ptrcrd, indcrd, valcrd, rhscrd);
11040     fprintf(outFile, "%3s\n", mxttype);
11041     fprintf(outFile, "      %14d%14d%14d%14d\n",
11042         *nrow, *ncol, *nnzero, neltvl);
11043     fprintf(outFile, "%16s%16s%20s%20s\n",
11044         ptrfmt, indfmt, valfmt, rhsfmt);
11045
11046     // Write the matrix structure
11047     for (i=1; i<=*ncol+1; i++)
11048         fprintf(outFile, "%8d", VAT(colptr, i));
11049     fprintf(outFile, "\n");
11050
11051     for (i=1; i<=*nnzero; i++)
11052         fprintf(outFile, "%8d", VAT(rowind, i));
11053     fprintf(outFile, "\n");
11054
11055     // Write out the values
11056     if (valcrd > 0) {
11057         for (i=1; i<=*nnzero; i++)
11058             fprintf(outFile, "%15.8e", VAT(values, i));
11059         fprintf(outFile, "\n");
11060     }
11061
11062     // Close the file
11063     fclose (outFile);
11064 }

```

10.103 src/mg/vpmg.h File Reference

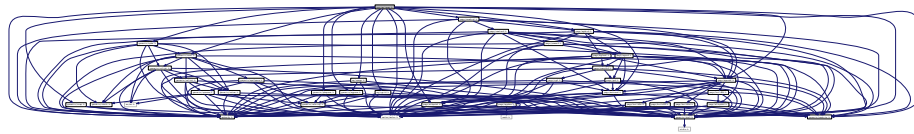
Contains declarations for class Vpmg.

```

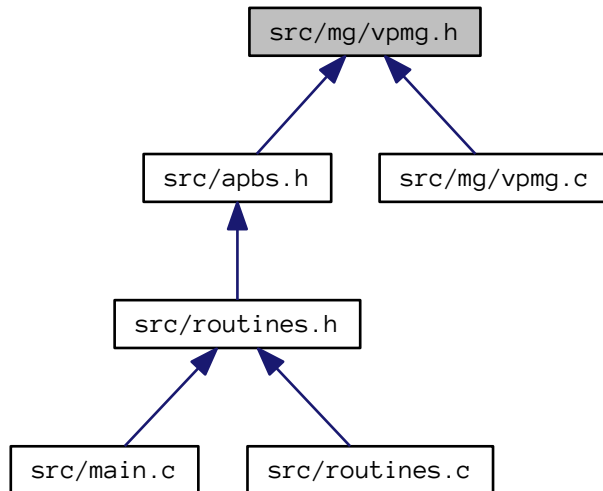
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/vacc.h"
#include "generic/vcap.h"
#include "generic/vpbe.h"
#include "generic/mgparm.h"
#include "generic/pbeparm.h"
#include "generic/vmatrix.h"
#include "pmgc/mgdrvd.h"
#include "pmgc/newdrvd.h"
#include "pmgc/mgsubd.h"
#include "pmgc/mikpckd.h"
#include "pmgc/matvecd.h"
#include "mg/vpmgp.h"
#include "mg/vgrid.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.

Macros

- `#define VPMGMAXPART 2000`
- `#define VCUB(x) ((x)*(x)*(x))`
- `#define VLOG(x) (log(x))`
- `#define IJK(i, j, k) (((k)*(nx)*(ny))+((j)*(nx))+i)`
- `#define IJKx(j, k, i) (((i)*(ny)*(nz))+((k)*(ny))+j)`
- `#define IJKy(i, k, j) (((j)*(nx)*(nz))+((k)*(nx))+i)`
- `#define IJKz(i, j, k) (((k)*(nx)*(ny))+((j)*(nx))+i)`
- `#define VFCHI(iint, iflt) (1.5+((double)(iint)-(iflt)))`

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)
FORTTRAN stub constructor for the Vpmg class (uses previously-allocated memory)
- VEXTERNC void [Vpmg_dtor](#) ([Vpmg](#) **thee)
Object destructor.
- VEXTERNC void [Vpmg_dtor2](#) ([Vpmg](#) *thee)
FORTTRAN 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` srfrm, `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` srfrm)
- 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` srfrm)
- 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` pbe-type, `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 dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.*

 - VEXTERNC void `Vpmg_ibDirectPolForce` (`Vpmg *thee`, `Vgrid` *perm, `Vgrid` *induced, int atomID, double force[3])
- Ionic boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.*

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

 - VEXTERNC void `Vpmg_dbDirectPolForce` (`Vpmg *thee`, `Vgrid` *perm, `Vgrid` *induced, int atomID, double force[3])

Dielectric boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.

- VEXTERNC void [Vpmpg_dbNLDirectPolForce](#) ([Vpmpg](#) *thee, [Vgrid](#) *perm, [Vgrid](#) *nllInduced, int atomID, double force[3])

Dielectric boundary direct polarization force between permanent multipoles and non-local induced dipoles. Keep in mind that the "non-local" induced dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

- VEXTERNC void [Vpmpg_qfMutualPolForce](#) ([Vpmpg](#) *thee, [Vgrid](#) *induced, [Vgrid](#) *nllInduced, int atomID, double force[3])

Mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

- VEXTERNC void [Vpmpg_ibMutualPolForce](#) ([Vpmpg](#) *thee, [Vgrid](#) *induced, [Vgrid](#) *nllInduced, int atomID, double force[3])

Ionic boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

- VEXTERNC void [Vpmpg_dbMutualPolForce](#) ([Vpmpg](#) *thee, [Vgrid](#) *induced, [Vgrid](#) *nllInduced, int atomID, double force[3])

Dielectric boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

- VEXTERNC void [Vpmpg_printColComp](#) ([Vpmpg](#) *thee, char path[72], char title[72], char mxtype[3], int flag)

Print out a column-compressed sparse matrix in Harwell-Boeing format.

- VPRIVATE void [bcolcomp](#) (int *iparm, double *rparm, int *iwork, double *rwork, double *values, int *rowind, int *colptr, int *flag)

Build a column-compressed matrix in Harwell-Boeing format.

- VPRIVATE void [bcolcomp2](#) (int *iparm, double *rparm, int *nx, int *ny, int *nz, int *iz, int *ipc, double *rpc, double *ac, double *cc, double *values, int *rowind, int *colptr, int *flag)

Build a column-compressed matrix in Harwell-Boeing format.

- VPRIVATE void [bcolcomp3](#) (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *ac, double *cc, double *values, int *rowind, int *colptr, int *flag)

Build a column-compressed matrix in Harwell-Boeing format.

- VPRIVATE void [bcolcomp4](#) (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *oC, double *cc, double *oE, double *oN, double *uC, double *values, int *rowind, int *colptr, int *flag)

Build a column-compressed matrix in Harwell-Boeing format.

- VPRIVATE void [pcolcomp](#) (int *nrow, int *ncol, int *nnzero, double *values, int *rowind, int *colptr, char *path, char *title, char *mxtype)

Print a column-compressed matrix in Harwell-Boeing format.

- VPRIVATE double [bspline2](#) (double x)

Evaluate a cubic B-spline.

- VPRIVATE double [dbspline2](#) (double x)

Evaluate a cubic B-spline derivative.

- VPRIVATE double [VFCHI4](#) (int i, double f)

Return 2.5 plus difference of i - f.

- VPRIVATE double [bspline4](#) (double x)

Evaluate a 5th Order B-Spline (4th order polynomial)

- VPRIVATE double [dbspline4](#) (double x)

Evaluate a 5th Order B-Spline derivative (4th order polynomial)

- VPRIVATE double [d2bspline4](#) (double x)

Evaluate the 2nd derivative of a 5th Order B-Spline.

- VPRIVATE double [d3bspline4](#) (double x)

- Evaluate the 3rd derivative of a 5th Order B-Spline.*
- VPRIVATE double [Vpmg_polarizEnergy](#) ([Vpmg](#) *thee, int extFlag)
Determines energy from polarizeable charge and interaction with fixed charges according to Rocchia et al.
 - VPRIVATE double [Vpmg_qfEnergyPoint](#) ([Vpmg](#) *thee, int extFlag)
Calculates charge-potential energy using summation over delta function positions (i.e. something like an Linf norm)
 - VPRIVATE double [Vpmg_qfEnergyVolume](#) ([Vpmg](#) *thee, int extFlag)
Calculates charge-potential energy as integral over a volume.
 - VPRIVATE void [Vpmg_splineSelect](#) (int srfrm, [Vacc](#) *acc, double *gpos, double win, double infrad, [Vatom](#) *atom, double *force)
Selects a spline based surface method from either VSM_SPLINE, VSM_SPLINE5 or VSM_SPLINE7.
 - 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)
*Increment all boundary points by $pre1 * (charge/d) * (exp(-xkappa * (d-size)) / (1 + xkappa * size))$ to add the effect of the Debye-Huckel potential due to a single charge.*
 - VPRIVATE void [multipolebc](#) (double r, double kappa, double eps_p, double eps_w, double rad, double tsr[3])
This routine serves bcfl2. It returns (in tsr) the contraction independent portion of the Debye-Huckel potential tensor for a spherical ion with a central charge, dipole and quadrupole. See the code for an in depth description.
 - VPRIVATE void [bcCalc](#) ([Vpmg](#) *thee)
Fill boundary condition arrays.
 - VPRIVATE void [fillcoCoef](#) ([Vpmg](#) *thee)
Top-level driver to fill all operator coefficient arrays.
 - VPRIVATE void [fillcoCoefMap](#) ([Vpmg](#) *thee)
Fill operator coefficient arrays from pre-calculated maps.
 - VPRIVATE void [fillcoCoefMol](#) ([Vpmg](#) *thee)
Fill operator coefficient arrays from a molecular surface calculation.
 - VPRIVATE void [fillcoCoefMollon](#) ([Vpmg](#) *thee)
Fill ion (nonlinear) operator coefficient array from a molecular surface calculation.
 - VPRIVATE void [fillcoCoefMolDiel](#) ([Vpmg](#) *thee)
Fill differential operator coefficient arrays from a molecular surface calculation.
 - VPRIVATE void [fillcoCoefMolDielNoSmooth](#) ([Vpmg](#) *thee)
Fill differential operator coefficient arrays from a molecular surface calculation without smoothing.
 - VPRIVATE void [fillcoCoefMolDielSmooth](#) ([Vpmg](#) *thee)
Fill differential operator coefficient arrays from a molecular surface calculation with smoothing.
 - VPRIVATE void [fillcoCoefSpline](#) ([Vpmg](#) *thee)
Fill operator coefficient arrays from a spline-based surface calculation.
 - VPRIVATE void [fillcoCoefSpline3](#) ([Vpmg](#) *thee)
Fill operator coefficient arrays from a 5th order polynomial based surface calculation.
 - VPRIVATE void [fillcoCoefSpline4](#) ([Vpmg](#) *thee)
Fill operator coefficient arrays from a 7th order polynomial based surface calculation.
 - VPRIVATE Vrc_Codes [fillcoCharge](#) ([Vpmg](#) *thee)
Top-level driver to fill source term charge array.
 - VPRIVATE Vrc_Codes [fillcoChargeMap](#) ([Vpmg](#) *thee)
Fill source term charge array from a pre-calculated map.
 - VPRIVATE void [fillcoChargeSpline1](#) ([Vpmg](#) *thee)
Fill source term charge array from linear interpolation.
 - VPRIVATE void [fillcoChargeSpline2](#) ([Vpmg](#) *thee)
Fill source term charge array from cubic spline interpolation.
 - VPRIVATE void [fillcoPermanentMultipole](#) ([Vpmg](#) *thee)

- Fill source term charge array for the use of permanent multipoles.*
- VPRIVATE void [fillcolInducedDipole](#) ([Vpmg](#) *thee)
- Fill source term charge array for use of induced dipoles.*
- VPRIVATE void [fillcoNLInducedDipole](#) ([Vpmg](#) *thee)
- Fill source term charge array for non-local induced dipoles.*
- VPRIVATE void [qfForceSpline1](#) ([Vpmg](#) *thee, double *force, int atomID)
- Charge-field force due to a linear spline charge function.*
- VPRIVATE void [qfForceSpline2](#) ([Vpmg](#) *thee, double *force, int atomID)
- Charge-field force due to a cubic spline charge function.*
- VPRIVATE void [qfForceSpline4](#) ([Vpmg](#) *thee, double *force, int atomID)
- Charge-field force due to a quintic spline charge function.*
- VPRIVATE void [zlapSolve](#) ([Vpmg](#) *thee, double **solution, double **source, double **work1)
- Calculate the solution to Poisson's equation with a simple Laplacian operator and zero-valued Dirichlet boundary conditions. Store the solution in thee->u.*
- VPRIVATE void [markSphere](#) (double rtot, double *tpos, int nx, int ny, int nz, double hx, double hy, double hzed, double xmin, double ymin, double zmin, double *array, double markVal)
- Mark the grid points inside a sphere with a particular value. This marks by resetting the the grid points inside the sphere to the specified value.*
- VPRIVATE double [Vpmg_qmEnergySMPBE](#) ([Vpmg](#) *thee, int extFlag)
- Vpmg_qmEnergy for SMPBE.*
- VPRIVATE double [Vpmg_qmEnergyNONLIN](#) ([Vpmg](#) *thee, int extFlag)

10.103.1 Detailed Description

Contains declarations for class [Vpmg](#).

Version

\$Id\$

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
```

```

* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* - Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of 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.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vpmg.h](#).

10.103.2 Function Documentation

10.103.2.1 VPRIVATE void bcCalc (Vpmg * thee)

Fill boundary condition arrays.

Author

Nathan Baker

Definition at line [4367](#) of file [vpmg.c](#).

10.103.2.2 **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*)

Increment all boundary points by $\text{pre1} * (\text{charge}/d) * (\exp(-xkappa * (d - \text{size})) / (1 + xkappa * \text{size}))$ to add the effect of the Debye-Huckel potential due to a single charge.

Author

Nathan Baker

Parameters

<i>apos</i>	Size of the ion
<i>charge</i>	Position of the ion
<i>xkappa</i>	Charge of the ion
<i>pre1</i>	Exponential screening factor
<i>gxcf</i>	Unit- and dielectric-dependent prefactor
<i>gycf</i>	Set to x-boundary values
<i>gzcf</i>	Set to y-boundary values
<i>xf</i>	Set to z-boundary values
<i>yf</i>	Boundary point x-coordinates
<i>zf</i>	Boundary point y-coordinates
<i>nx</i>	Boundary point z-coordinates
<i>ny</i>	Number of grid points in x-direction
<i>nz</i>	Number of grid points in y-direction Number of grid points in y-direction

Definition at line 2564 of file [vpmpg.c](#).

10.103.2.3 **VPRIVATE** double bspline2 (double *x*)

Evaluate a cubic B-spline.

Author

Nathan Baker

Returns

Cubic B-spline value

Parameters

<i>x</i>	Position
----------	----------

Definition at line 5481 of file [vpmpg.c](#).

10.103.2.4 **VPRIVATE** double bspline4 (double *x*)

Evaluate a 5th Order B-Spline (4th order polynomial)

Author

: Michael Schnieders

Returns

5th Order B-Spline

Parameters

x	Position
-----	----------

Definition at line 7121 of file [vpmg.c](#).

10.103.2.5 **VPRIVATE** double d2bspline4 (double x)

Evaluate the 2nd derivative of a 5th Order B-Spline.

Author

: Michael Schnieders

Returns

2nd derivative of a 5th Order B-Spline

Parameters

x	Position
-----	----------

Definition at line 7187 of file [vpmg.c](#).

10.103.2.6 **VPRIVATE** double d3bspline4 (double x)

Evaluate the 3rd derivative of a 5th Order B-Spline.

Author

: Michael Schnieders

Returns

3rd derivative of a 5th Order B-Spline

Parameters

x	Position
-----	----------

Definition at line 7214 of file [vpmg.c](#).

10.103.2.7 **VPRIVATE** double dbspline2 (double x)

Evaluate a cubic B-spline derivative.

Author

Nathan Baker

Returns

Cubic B-spline derivative

Parameters

x	Position
-----	----------

Definition at line 5497 of file [vpmg.c](#).

10.103.2.8 VPRIVATE double dbspline4 (double x)

Evaluate a 5th Order B-Spline derivative (4th order polynomial)

Author

: Michael Schnieders

Returns

5th Order B-Spline derivative

Parameters

x	Position
-----	----------

Definition at line 7155 of file [vpmg.c](#).

10.103.2.9 VPRIVATE Vrc_Codes fillcoCharge (Vpmg * *thee*)

Top-level driver to fill source term charge array.

Returns

Success/failure status

Author

Nathan Baker

Definition at line 5272 of file [vpmg.c](#).

10.103.2.10 VPRIVATE Vrc_Codes fillcoChargeMap (Vpmg * *thee*)

Fill source term charge array from a pre-calculated map.

Returns

Success/failure status

Author

Nathan Baker

Definition at line 5328 of file [vpmg.c](#).

10.103.2.11 **VPRIVATE** void fillcoChargeSpline1 (**Vpmg** * *thee*)

Fill source term charge array from linear interpolation.

Author

Nathan Baker

Definition at line [5376](#) of file [vpmg.c](#).

10.103.2.12 **VPRIVATE** void fillcoChargeSpline2 (**Vpmg** * *thee*)

Fill source term charge array from cubic spline interpolation.

Author

Nathan Baker

Definition at line [5513](#) of file [vpmg.c](#).

10.103.2.13 **VPRIVATE** void fillcoCoef (**Vpmg** * *thee*)

Top-level driver to fill all operator coefficient arrays.

Author

Nathan Baker

Definition at line [5232](#) of file [vpmg.c](#).

10.103.2.14 **VPRIVATE** void fillcoCoefMap (**Vpmg** * *thee*)

Fill operator coefficient arrays from pre-calculated maps.

Author

Nathan Baker

Definition at line [4474](#) of file [vpmg.c](#).

10.103.2.15 **VPRIVATE** void fillcoCoefMol (**Vpmg** * *thee*)

Fill operator coefficient arrays from a molecular surface calculation.

Author

Nathan Baker

Definition at line [4597](#) of file [vpmg.c](#).

10.103.2.16 `VPRIVATE void fillCoefMolDiel (Vpmg * thee)`

Fill differential operator coefficient arrays from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4711 of file [vpmg.c](#).

10.103.2.17 `VPRIVATE void fillCoefMolDielNoSmooth (Vpmg * thee)`

Fill differential operator coefficient arrays from a molecular surface calculation without smoothing.

Author

Nathan Baker

Definition at line 4722 of file [vpmg.c](#).

10.103.2.18 `VPRIVATE void fillCoefMolDielSmooth (Vpmg * thee)`

Fill differential operator coefficient arrays from a molecular surface calculation with smoothing.

Molecular surface, dielectric smoothing following an implementation of Bruccoleri, et al. J Comput Chem 18 268-276 (1997).

This algorithm uses a 9 point harmonic smoothing technique - the point in question and all grid points 1/sqrt(2) grid spacings away.

Note

This uses thee->a1cf, thee->a2cf, thee->a3cf as temporary storage.

Author

Todd Dolinsky

Definition at line 4876 of file [vpmg.c](#).

10.103.2.19 `VPRIVATE void fillCoefMollon (Vpmg * thee)`

Fill ion (nonlinear) operator coefficient array from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4613 of file [vpmg.c](#).

10.103.2.20 **VPRIVATE** void fillCoefSpline (**Vpmg** * *thee*)

Fill operator coefficient arrays from a spline-based surface calculation.

Author

Nathan Baker

Definition at line 5007 of file [vpmg.c](#).

10.103.2.21 **VPRIVATE** void fillCoefSpline3 (**Vpmg** * *thee*)

Fill operator coefficient arrays from a 5th order polynomial based surface calculation.

Author

Michael Schnieders

Definition at line 10415 of file [vpmg.c](#).

10.103.2.22 **VPRIVATE** void fillCoefSpline4 (**Vpmg** * *thee*)

Fill operator coefficient arrays from a 7th order polynomial based surface calculation.

Author

Michael Schnieders

Definition at line 9924 of file [vpmg.c](#).

10.103.2.23 **VPRIVATE** void fillCoInducedDipole (**Vpmg** * *thee*)

Fill source term charge array for use of induced dipoles.

Author

Michael Schnieders

10.103.2.24 **VPRIVATE** void fillCoNLInducedDipole (**Vpmg** * *thee*)

Fill source term charge array for non-local induced dipoles.

Author

Michael Schnieders

10.103.2.25 **VPRIVATE** void fillCoPermanentMultipole (**Vpmg** * *thee*)

Fill source term charge array for the use of permanent multipoles.

Author

Michael Schnieders

Definition at line 7225 of file [vpmg.c](#).

10.103.2.26 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*)

Mark the grid points inside a sphere with a particular value. This marks by resetting the the grid points inside the sphere to the specified value.

Author

Nathan Baker

Parameters

<i>tpos</i>	Sphere radius
<i>nx</i>	Sphere position
<i>ny</i>	Number of grid points
<i>nz</i>	Number of grid points
<i>hx</i>	Number of grid points
<i>hy</i>	Grid spacing
<i>hz</i>	Grid spacing
<i>xmin</i>	Grid spacing
<i>ymin</i>	Grid lower corner
<i>zmin</i>	Grid lower corner
<i>array</i>	Grid lower corner
<i>markVal</i>	Grid values Value to mark with

Definition at line 6834 of file [vpmg.c](#).

10.103.2.27 VPRIVATE void multipolebc (double *r*, double *kappa*, double *eps_p*, double *eps_w*, double *rad*, double *tsr*[3])

This routine serves bcf12. It returns (in *tsr*) the contraction independent portion of the Debye-Huckel potential tensor for a spherical ion with a central charge, dipole and quadrupole. See the code for an in depth description.

Author

Michael Schnieders

Parameters

<i>kappa</i>	Distance to the boundary
<i>eps_p</i>	Exponential screening factor
<i>eps_w</i>	Solute dielectric
<i>rad</i>	Solvent dielectric
<i>tsr</i>	Radius of the sphere Contraction-independent portion of each tensor

Definition at line 3477 of file [vpmg.c](#).

10.103.2.28 VPRIVATE void qfForceSpline1 (Vpmg * *thee*, double * *force*, int *atomID*)

Charge-field force due to a linear spline charge function.

Author

Nathan Baker

Parameters

<i>atomID</i>	Set to force Valist atom ID
---------------	-----------------------------

Definition at line 6296 of file [vpmg.c](#).

10.103.2.29 VPRIVATE void qfForceSpline2 (Vpmg * *thee*, double * *force*, int *atomID*)

Charge-field force due to a cubic spline charge function.

Author

Nathan Baker

Parameters

<i>atomID</i>	Set to force Valist atom ID
---------------	-----------------------------

Definition at line 6433 of file [vpmg.c](#).

10.103.2.30 VPRIVATE void qfForceSpline4 (Vpmg * *thee*, double * *force*, int *atomID*)

Charge-field force due to a quintic spline charge function.

Author

Michael Schnieders

Parameters

<i>atomID</i>	Set to force Valist atom ID
---------------	-----------------------------

Definition at line 6546 of file [vpmg.c](#).

10.103.2.31 VPRIVATE double VFCHI4 (int *i*, double *f*)

Return 2.5 plus difference of i - f.

Author

Michael Schnieders

Returns

(2.5+((double)(i)-(f)))

Definition at line 7117 of file [vpmg.c](#).

10.103.2.32 VPRIVATE double Vpmg_polarizEnergy (Vpmg * *thee*, int *extFlag*)

Determines energy from polarizeable charge and interaction with fixed charges according to Rocchia et al.

Author

Nathan Baker

Returns

Energy in kT

Parameters

<i>extFlag</i>	If 1, add external energy contributions to result
----------------	---

Definition at line 1148 of file [vpmg.c](#).**10.103.2.33** VPRIVATE double Vpmg_qfEnergyPoint (Vpmg * *thee*, int *extFlag*)

Calculates charge-potential energy using summation over delta function positions (i.e. something like an Linf norm)

Author

Nathan Baker

Returns

Energy in kT

Parameters

<i>extFlag</i>	If 1, add external energy contributions to result
----------------	---

Definition at line 1704 of file [vpmg.c](#).**10.103.2.34** VPRIVATE double Vpmg_qfEnergyVolume (Vpmg * *thee*, int *extFlag*)

Calculates charge-potential energy as integral over a volume.

Author

Nathan Baker

Returns

Energy in kT

Parameters

<i>extFlag</i>	If 1, add external energy contributions to result
----------------	---

Definition at line 1861 of file [vpmg.c](#).**10.103.2.35** VPRIVATE double Vpmg_qmEnergySMPBE (Vpmg * *thee*, int *extFlag*)

Vpmg_qmEnergy for SMPBE.

Author

Vincent Chu

Definition at line 1490 of file [vpmg.c](#).

10.103.2.36 `VPRIVATE void Vpmg_splineSelect (int srfm, Vacc * acc, double * gpos, double win, double infrad, Vatom * atom, double * force)`

Selects a spline based surface method from either VSM_SPLINE, VSM_SPLINE5 or VSM_SPLINE7.

Author

David Gohara

Parameters

<i>acc</i>	Surface method, currently VSM_SPLINE, VSM_SPLINE5, or VSM_SPLINE7
<i>gpos</i>	Accessibility object
<i>win</i>	Position array -> array[3]
<i>infrad</i>	Spline window
<i>atom</i>	Inflation radius
<i>force</i>	Atom object Force array -> array[3]

Definition at line 1893 of file [vpmg.c](#).

10.103.2.37 `VPRIVATE void zlapSolve (Vpmg * thee, double ** solution, double ** source, double ** work1)`

Calculate the solution to Poisson's equation with a simple Laplacian operator and zero-valued Dirichlet boundary conditions. Store the solution in *thee*->u.

Author

Nathan Baker

Note

Vpmg_fillco must be called first

Parameters

<i>source</i>	Solution term vector
<i>work1</i>	Source term vector Work vector

Definition at line 6883 of file [vpmg.c](#).

10.104 vpmg.h

```

00001
00080 #ifndef _VPMG_H_
00081 #define _VPMG_H_
00082
00083 #include "apbscfg.h"
00084
00085 #include "malloc/malloc.h"
00086
00087 #include "generic/vhal.h"
00088 #include "generic/vacc.h"
00089 #include "generic/vcap.h"
00090 #include "generic/vpbe.h"
00091 #include "generic/mgparm.h"
00092 #include "generic/pbeparm.h"
00093 #include "generic/vmatrix.h"
00094 #include "pmgc/mgdrv.h"
00095 #include "pmgc/newdrv.h"

```



```

00096 #include "pmgc/mgsubd.h"
00097 #include "pmgc/mikpckd.h"
00098 #include "pmgc/matvecd.h"
00099 #include "mg/vpmgp.h"
00100 #include "mg/vgrid.h"
00101
00105 #define VPMGMAXPART 2000
00106
00116 struct sVpmg {
00117     Vmem *vmem;
00119     Vpmgp *pmgp;
00120     Vpbe *pbe;
00122 #ifdef BURY_FORTRAN
00123     Vpde *pde;
00124     Vmgdriver *mgdriver;
00125 #endif
00126
00127     double *epsx;
00128     double *epsy;
00129     double *epsz;
00130     double *kappa;
00131     double *pot;
00132     double *charge;
00134     int *iparm;
00135     double *rparm;
00136     int *iwork;
00137     double *rwork;
00138     double *alcf;
00140     double *a2cf;
00142     double *a3cf;
00144     double *ccf;
00145     double *fcf;
00146     double *tcf;
00147     double *u;
00148     double *xf;
00149     double *yf;
00150     double *zf;
00151     double *gxcf;
00152     double *gycf;
00153     double *gzcf;
00154     double *pvec;
00155     double extDiEnergy;
00157     double extQmEnergy;
00159     double extQfEnergy;
00161     double extNpEnergy;
00163     Vsurf_Meth surfMeth;
00164     double splineWin;
00165     Vchrg_Meth chargeMeth;
00166     Vchrg_Src chargeSrc;
00168     int filled;
00170     int useDielXMap;
00172     Vgrid *dielXMap;
00173     int useDielYMap;
00175     Vgrid *dielYMap;
00176     int useDielZMap;
00178     Vgrid *dielZMap;
00179     int useKappaMap;
00181     Vgrid *kappaMap;
00182     int usePotMap;
00184     Vgrid *potMap;
00186     int useChargeMap;
00188     Vgrid *chargeMap;
00189 };
00190
00195 typedef struct sVpmg Vpmg;
00196
00197 /* //////////////////////////////////////
00200 #if !defined(VINLINE_VPMG)
00201
00208     VEXTERNC unsigned long int Vpmg_memChk(
00209         Vpmg *thee
00210     );
00211
00212 #else /* if defined(VINLINE_VPMG) */
00213
00214 #    define Vpmg_memChk(thee) (Vmem_bytes((thee)->vmem))
00215
00216 #endif /* if !defined(VINLINE_VPMG) */
00217
00218 /* //////////////////////////////////////

```

```
00221
00226 VEXTERNC Vpmg* Vpmg_ctor(
00227     Vpmgp *parms,
00228     Vpbe *pbe,
00229     int focusFlag,
00230     Vpmg *pmgOLD,
00231     MGparm *mgparm,
00232     PBEparm_calcEnergy energyFlag
00233 );
00234
00242 VEXTERNC int Vpmg_ctor2(
00243     Vpmg *thee,
00244     Vpmgp *parms,
00245     Vpbe *pbe,
00246     int focusFlag,
00247     Vpmg *pmgOLD,
00249     MGparm *mgparm,
00251     PBEparm_calcEnergy energyFlag
00254 );
00255
00260 VEXTERNC void Vpmg_dtor(
00261     Vpmg **thee
00263 );
00264
00269 VEXTERNC void Vpmg_dtor2(
00270     Vpmg *thee
00271 );
00272
00281 VEXTERNC int Vpmg_fillco(
00282     Vpmg *thee,
00283     Vsurf_Meth surfMeth,
00284     double splineWin,
00286     Vchrg_Meth chargeMeth,
00287     int useDielXMap,
00288     Vgrid *dielXMap,
00289     int useDielYMap,
00290     Vgrid *dielYMap,
00291     int useDielZMap,
00292     Vgrid *dielZMap,
00293     int useKappaMap,
00294     Vgrid *kappaMap,
00295     int usePotMap,
00296     Vgrid *potMap,
00297     int useChargeMap,
00298     Vgrid *chargeMap
00299 );
00300
00306 VEXTERNC int Vpmg_solve(
00307     Vpmg *thee
00308 );
00309
00321 VEXTERNC int Vpmg_solveLaplace(
00322     Vpmg *thee
00323 );
00324
00334 VEXTERNC double Vpmg_energy(
00335     Vpmg *thee,
00336     int extFlag
00340 );
00341
00359 VEXTERNC double Vpmg_qfEnergy(
00360     Vpmg *thee,
00361     int extFlag
00365 );
00366
00386 VEXTERNC double Vpmg_qfAtomEnergy(
00387     Vpmg *thee,
00388     Vatom *atom
00389 );
00390
00415 VEXTERNC double Vpmg_qmEnergy(
00416     Vpmg *thee,
00417     int extFlag
00421 );
00422
00423
00442 VEXTERNC double Vpmg_dielEnergy(
00443     Vpmg *thee,
00444     int extFlag
00448 );
00449
```

```
00450
00467 VEXTERNC double Vpmg_dielGradNorm(
00468     Vpmg *thee
00469 );
00470
00482 VEXTERNC int Vpmg_force(
00483     Vpmg *thee,
00484     double *force,
00486     int atomID,
00487     Vsurf_Meth srfm,
00488     Vchrg_Meth chgm
00489 );
00490
00502 VEXTERNC int Vpmg_qfForce(
00503     Vpmg *thee,
00504     double *force,
00506     int atomID,
00507     Vchrg_Meth chgm
00508 );
00509
00521 VEXTERNC int Vpmg_dbForce(
00522     Vpmg *thee,
00523     double *dbForce,
00525     int atomID,
00526     Vsurf_Meth srfm
00527 );
00528
00540 VEXTERNC int Vpmg_ibForce(
00541     Vpmg *thee,
00542     double *force,
00544     int atomID,
00545     Vsurf_Meth srfm
00546 );
00547
00553 VEXTERNC void Vpmg_setPart(
00554     Vpmg *thee,
00555     double lowerCorner[3],
00556     double upperCorner[3],
00557     int bflags[6]
00561 );
00562
00567 VEXTERNC void Vpmg_unsetPart(
00568     Vpmg *thee
00569 );
00570
00576 VEXTERNC int Vpmg_fillArray(
00577     Vpmg *thee,
00578     double *vec,
00580     Vdata_Type type,
00581     double parm,
00582     Vhal_PBEType pbetype,
00583     PBEparm * pbeparm
00584 );
00585
00591 VPUBLIC void Vpmg_fieldSpline4(
00592     Vpmg *thee,
00593     int atomID,
00594     double field[3]
00595 );
00596
00604 VEXTERNC double Vpmg_qfPermanentMultipoleEnergy(
00605     Vpmg *thee,
00606     int atomID
00607 );
00608
00614 VEXTERNC void Vpmg_qfPermanentMultipoleForce(
00615     Vpmg *thee,
00616     int atomID,
00617     double force[3],
00618     double torque[3]
00619 );
00620
00625 VEXTERNC void Vpmg_ibPermanentMultipoleForce(
00626     Vpmg *thee,
00627     int atomID,
00628     double force[3]
00629 );
00630
00635 VEXTERNC void Vpmg_dbPermanentMultipoleForce(
00636     Vpmg *thee,
00637     int atomID,
```

```
00638         double force[3]
00639     );
00640
00641 VEXTERNC void Vpmg_qfDirectPolForce(
00642     Vpmg *thee,
00643     Vgrid *perm,
00644     Vgrid *induced,
00645     int atomID,
00646     double force[3],
00647     double torque[3]
00648 );
00649
00650 VEXTERNC void Vpmg_qfNLDirectPolForce(
00651     Vpmg *thee,
00652     Vgrid *perm,
00653     Vgrid *nlInduced,
00654     int atomID,
00655     double force[3],
00656     double torque[3]
00657 );
00658
00659 VEXTERNC void Vpmg_ibDirectPolForce(
00660     Vpmg *thee,
00661     Vgrid *perm,
00662     Vgrid *induced,
00663     int atomID,
00664     double force[3]
00665 );
00666
00667 VEXTERNC void Vpmg_ibNLDirectPolForce(
00668     Vpmg *thee,
00669     Vgrid *perm,
00670     Vgrid *nlInduced,
00671     int atomID,
00672     double force[3]
00673 );
00674
00675 VEXTERNC void Vpmg_dbDirectPolForce(
00676     Vpmg *thee,
00677     Vgrid *perm,
00678     Vgrid *induced,
00679     int atomID,
00680     double force[3]
00681 );
00682
00683 VEXTERNC void Vpmg_dbNLDirectPolForce(
00684     Vpmg *thee,
00685     Vgrid *perm,
00686     Vgrid *nlInduced,
00687     int atomID,
00688     double force[3]
00689 );
00690
00691 VEXTERNC void Vpmg_qfMutualPolForce(
00692     Vpmg *thee,
00693     Vgrid *induced,
00694     Vgrid *nlInduced,
00695     int atomID,
00696     double force[3]
00697 );
00698
00699 VEXTERNC void Vpmg_ibMutualPolForce(
00700     Vpmg *thee,
00701     Vgrid *induced,
00702     Vgrid *nlInduced,
00703     int atomID,
00704     double force[3]
00705 );
00706
00707 VEXTERNC void Vpmg_dbMutualPolForce(
00708     Vpmg *thee,
00709     Vgrid *induced,
00710     Vgrid *nlInduced,
00711     int atomID,
00712     double force[3]
00713 );
00714
00715 VEXTERNC void Vpmg_printColComp(
00716     Vpmg *thee,
00717     char path[72],
00718     char title[72],
```

```
00789     char mxttype[3],
00797     int flag
00801     );
00802
00803
00804
00811 VPRIVATE void bcolcomp(
00812     int *iparm, ///< @todo Document
00813     double *rparm, ///< @todo Document
00814     int *iwork, ///< @todo Document
00815     double *rwork, ///< @todo Document
00816     double *values, ///< @todo Document
00817     int *rowind, ///< @todo Document
00818     int *colptr, ///< @todo Document
00819     int *flag
00824     );
00825
00826
00827
00834 VPRIVATE void bcolcomp2(
00835     int *iparm, ///< @todo Document
00836     double *rparm, ///< @todo Document
00837     int *nx, ///< @todo Document
00838     int *ny, ///< @todo Document
00839     int *nz, ///< @todo Document
00840     int *iz, ///< @todo Document
00841     int *ipc, ///< @todo Document
00842     double *rpc, ///< @todo Document
00843     double *ac, ///< @todo Document
00844     double *cc, ///< @todo Document
00845     double *values, ///< @todo Document
00846     int *rowind, ///< @todo Document
00847     int *colptr, ///< @todo Document
00848     int *flag
00853     );
00854
00855
00856
00863 VPRIVATE void bcolcomp3(
00864     int *nx, ///< @todo Document
00865     int *ny, ///< @todo Document
00866     int *nz, ///< @todo Document
00867     int *ipc, ///< @todo Document
00868     double *rpc, ///< @todo Document
00869     double *ac, ///< @todo Document
00870     double *cc, ///< @todo Document
00871     double *values, ///< @todo Document
00872     int *rowind, ///< @todo Document
00873     int *colptr, ///< @todo Document
00874     int *flag ///< @todo Document
00875     );
00876
00877
00878
00885 VPRIVATE void bcolcomp4(
00886     int *nx, ///< @todo Document
00887     int *ny, ///< @todo Document
00888     int *nz, ///< @todo Document
00889     int *ipc, ///< @todo Document
00890     double *rpc, ///< @todo Document
00891     double *oC, ///< @todo Document
00892     double *cc, ///< @todo Document
00893     double *oE, ///< @todo Document
00894     double *oN, ///< @todo Document
00895     double *uC, ///< @todo Document
00896     double *values, ///< @todo Document
00897     int *rowind, ///< @todo Document
00898     int *colptr, ///< @todo Document
00899     int *flag ///< @todo Document
00900     );
00901
00902
00903
00910 VPRIVATE void pcolcomp(
00911     int *nrow, ///< @todo Document
00912     int *ncol, ///< @todo Document
00913     int *nnzero, ///< @todo Document
00914     double *values, ///< @todo Document
00915     int *rowind, ///< @todo Document
00916     int *colptr, ///< @todo Document
00917     char *path, ///< @todo Document
```

```
00918      char   *title, ///< @todo Document
00919      char   *mxttype ///< @todo Document
00920      );
00921
00922
00923
00924 /* ////////////////////////////////////////
00925 // Internal routines
00926
00927
00933 VPRIVATE double bspline2(
00934     double x
00935 );
00936
00942 VPRIVATE double dbspline2(
00943     double x
00944 );
00945
00951 VPRIVATE double VFCHI4(
00952     int i,
00953     double f
00954 );
00955
00961 VPRIVATE double bspline4(
00962     double x
00963 );
00964
00970 VPRIVATE double dbspline4(
00971     double x
00972 );
00973
00979 VPRIVATE double d2bspline4(
00980     double x
00981 );
00982
00988 VPRIVATE double d3bspline4(
00989     double x
00990 );
00991
00998 VPRIVATE double Vpmg_polarizEnergy(
00999     Vpmg *thee,
01000     int extFlag
01001 );
01002
01009 VPRIVATE double Vpmg_qfEnergyPoint(
01010     Vpmg *thee,
01011     int extFlag
01012 );
01013
01014
01020 VPRIVATE double Vpmg_qfEnergyVolume(
01021     Vpmg *thee,
01022     int extFlag
01023 );
01024
01025
01031 VPRIVATE void Vpmg_splineSelect(
01032     int srfm,
01033     Vacc *acc,
01034     double *gpos,
01035     double win,
01036     double infrad,
01037     Vatom *atom,
01038     double *force
01039 );
01040
01041
01047 VPRIVATE void focusFillBound(
01048     Vpmg *thee,
01049     Vpmg *pmg
01050 );
01051
01058 VPRIVATE void bcfl1(
01059     double size,
01060     double *apos,
01061     double charge,
01062     double xkappa,
01063     double prel,
01064     double *gxcf,
01065     double *gycf,
01066     double *gzcf,
01067     double *xf,
01068     double *yf,
01069     double *zf,
01070     int nx,
01071     int ny,
```

```
01072         int nz
01073     );
01074
01080 VPRIVATE void bcfl12(
01081     double size,
01082     double *apos,
01083     double charge,
01084     double *dipole,
01085     double *quad,
01086     double xkappa,
01087     double eps_p,
01088     double eps_w,
01089     double T,
01090     double *gxcf,
01091     double *gycf,
01092     double *gzcf,
01093     double *xf,
01094     double *yf,
01095     double *zf,
01096     int nx,
01097     int ny,
01098     int nz
01099 );
01100
01109 VPRIVATE void multipolebc(
01110     double r,
01111     double kappa,
01112     double eps_p,
01113     double eps_w,
01114     double rad,
01115     double tsr[3]
01116 );
01117
01126 VPRIVATE double bcfl1sp(
01127     double size,
01128     double *apos,
01129     double charge,
01130     double xkappa,
01131     double prel,
01132     double *pos
01133 );
01134
01139 VPRIVATE void bcCalc(
01140     Vpmg *thee
01141 );
01142
01147 VPRIVATE void fillcoCoef(
01148     Vpmg *thee
01149 );
01150
01155 VPRIVATE void fillcoCoefMap(
01156     Vpmg *thee
01157 );
01158
01164 VPRIVATE void fillcoCoefMol(
01165     Vpmg *thee
01166 );
01167
01173 VPRIVATE void fillcoCoefMolIon(
01174     Vpmg *thee
01175 );
01176
01182 VPRIVATE void fillcoCoefMolDiel(
01183     Vpmg *thee
01184 );
01185
01191 VPRIVATE void fillcoCoefMolDielNoSmooth(
01192     Vpmg *thee
01193 );
01194
01208 VPRIVATE void fillcoCoefMolDielSmooth(
01209     Vpmg *thee
01210 );
01211
01217 VPRIVATE void fillcoCoefSpline(
01218     Vpmg *thee
01219 );
01220
01226 VPRIVATE void fillcoCoefSpline3(
01227     Vpmg *thee
01228 );
```

```
01229
01235 VPRIVATE void fillcoCoefSpline4(
01236     Vpmg *thee
01237 );
01238
01244 VPRIVATE Vrc_Codes fillcoCharge(
01245     Vpmg *thee
01246 );
01247
01253 VPRIVATE Vrc_Codes fillcoChargeMap(
01254     Vpmg *thee
01255 );
01256
01261 VPRIVATE void fillcoChargeSpline1(
01262     Vpmg *thee
01263 );
01264
01269 VPRIVATE void fillcoChargeSpline2(
01270     Vpmg *thee
01271 );
01272
01277 VPRIVATE void fillcoPermanentMultipole(
01278     Vpmg *thee
01279 );
01280
01285 VPRIVATE void fillcoInducedDipole(
01286     Vpmg *thee
01287 );
01288
01294 VPRIVATE void fillcoNLInducedDipole(
01295     Vpmg *thee
01296 );
01297
01304 VPRIVATE void extEnergy(
01305     Vpmg *thee,
01306     Vpmg *pmgOLD,
01307     PBEparm_calcEnergy extFlag,
01308     double partMin[3],
01309     double partMax[3],
01310     int bflags[6]
01311 );
01312
01317 VPRIVATE void qfForceSpline1(
01318     Vpmg *thee,
01319     double *force,
01320     int atomID
01321 );
01322
01327 VPRIVATE void qfForceSpline2(
01328     Vpmg *thee,
01329     double *force,
01330     int atomID
01331 );
01332
01337 VPRIVATE void qfForceSpline4(
01338     Vpmg *thee,
01339     double *force,
01340     int atomID
01341 );
01342
01343
01351 VPRIVATE void zlapSolve(
01352     Vpmg *thee,
01353     double **solution,
01354     double **source,
01355     double **work1
01356 );
01357
01364 VPRIVATE void markSphere(
01365     double rtot,
01366     double *tpos,
01367     int nx,
01368     int ny,
01369     int nz,
01370     double hx,
01371     double hy,
01372     double hzed,
01373     double xmin,
01374     double ymin,
01375     double zmin,
01376     double *array,
```



```

01377     double markVal
01378     );
01379
01384 VPRIVATE double Vpmg_qmEnergySMPBE(Vpmg *thee, int extFlag);
01385 VPRIVATE double Vpmg_qmEnergyNONLIN(Vpmg *thee, int extFlag);
01386
01387
01388
01389 // Additional macros and definitions. May not be needed
01390
01391 // Added by Vincent Chu 9/13/06 for SMPB
01392 #define VCUB(x)      ((x)*(x)*(x))
01393 #define VLOG(x)      (log(x))
01394
01395 #define IJK(i,j,k)   (((k)*(nx)*(ny))+((j)*(nx))+((i)))
01396 #define IJKx(j,k,i) (((i)*(ny)*(nz))+((k)*(ny))+((j)))
01397 #define IJKy(i,k,j) (((j)*(nx)*(nz))+((k)*(nx))+((i)))
01398 #define IJKz(i,j,k) (((k)*(nx)*(ny))+((j)*(nx))+((i)))
01399 #define VFCHI(iint,iflt) (1.5+((double)(iint)-(iflt)))
01400
01401
01402 #endif /* ifndef _VPMG_H_ */
01403

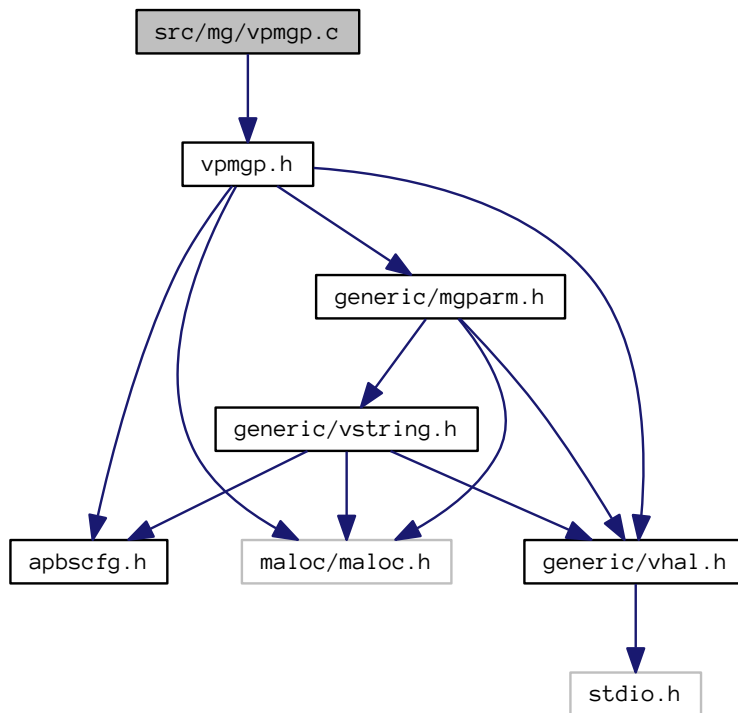
```

10.105 src/mg/vpmgp.c File Reference

Class Vpmgp methods.

```
#include "vpmgp.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)
FORTTRAN 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)
FORTTRAN stub for object destructor.
- VPUBLIC void [Vpmgp_size](#) ([Vpmgp](#) *thee)
Determine array sizes and parameters for multigrid solver.
- VPRIVATE int **coarsenThis** (int 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.

10.105.1 Detailed Description

Class Vpmgp methods.

Author

Nathan Baker

Version

\$Id\$

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
```

```

* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vpmgp.c](#).

10.106 vpmgp.c

```

00001
00057 #include "vpmgp.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 /* ////////////////////////////////////////
00062 // Class Vpmgp: Inlineable methods
00064 #if !defined(VINLINE_VACC)
00065 #endif /* if !defined(VINLINE_VACC) */
00066
00067 /* ////////////////////////////////////////
00068 // Class Vpmgp: Non-inlineable methods
00070
00071 /* ////////////////////////////////////////
00072 // Routine: Vpmgp_ctor
00073 //
00074 // Author: Nathan Baker
00076 VPUBLIC Vpmgp* Vpmgp_ctor(MGparm *mgparm) {
00077
00078     Vpmgp *thee = VNULL;
00079
00080     /* Set up the structure */
00081     thee = (Vpmgp*)Vmem_malloc(VNULL, 1, sizeof(Vpmgp) );
00082     VASSERT( thee != VNULL);
00083     VASSERT(Vpmgp_ctor2(thee,mgparm));
00084
00085     return thee;
00086 }
00087
00088 /* ////////////////////////////////////////
00089 // Routine: Vpmgp_ctor2
00090 //
00091 // Author: Nathan Baker
00093 VPUBLIC int Vpmgp_ctor2(Vpmgp *thee,MGparm *mgparm) {
00094
00095     /* Specified parameters */
00096     thee->nx = mgparm->dime[0];
00097     thee->ny = mgparm->dime[1];
00098     thee->nz = mgparm->dime[2];
00099     thee->hx = mgparm->grid[0];
00100     thee->hy = mgparm->grid[1];
00101     thee->hz = mgparm->grid[2];
00102     thee->xlen = ((double) (mgparm->dime[0]-1))*mgparm->grid[0];
00103     thee->ylen = ((double) (mgparm->dime[1]-1))*mgparm->grid[1];
00104     thee->zlen = ((double) (mgparm->dime[2]-1))*mgparm->grid[2];
00105     thee->nlev = mgparm->nlev;
00106

```

```

00107     thee->nonlin = mgparm->nonlintype;
00108     thee->meth = mgparm->method;
00109
00110 #ifdef DEBUG_MAC_OSX_OCL
00111 #include "mach_chud.h"
00112     if (kOpenCLAvailable)
00113         thee->meth = 4;
00114 #endif
00115
00116     if (thee->nonlin == NONLIN_LPBE) thee->ipkey = IPKEY_LPBE; /* LPBE case */
00117     else if (thee->nonlin == NONLIN_SMPBE) thee->ipkey = IPKEY_SMPBE; /* SMPBE case */
00118     else thee->ipkey = IPKEY_NPBE; /* NPBE standard case */
00119
00120     /* Default parameters */
00121     if (mgparm->setetol) { /* If etol is set by the user in APBS input file, then use this custom-defined
        etol */
00122         thee->errtol = mgparm->etol;
00123         Vnm_print(1, " Error tolerance (etol) is now set to user-defined \
00124 value: %g \n", thee->errtol);
00125         Vnm_print(0, "Error tolerance (etol) is now set to user-defined \
00126 value: %g \n", thee->errtol);
00127     } else thee->errtol = 1.0e-6; /* Here are a few comments. Mike had this set to
        * 1e-9; conventional wisdom sets this at 1e-6 for
00128     * the PBE; Ray Luo sets this at 1e-3 for his
00129     * accelerated PBE (for dynamics, etc.) */
00130
00131     thee->itmax = 200;
00132     thee->istop = 1;
00133     thee->iinfo = 1; /* I'd recommend either 1 (for debugging LPBE) or 2 (for debugging NPBE),
        higher values give too much output */
00134
00135     thee->bcfl = BCFL_SDH;
00136     thee->key = 0;
00137     thee->iperf = 0;
00138     thee->mgcoar = 2;
00139     thee->mgkey = 0;
00140     thee->nu1 = 2;
00141     thee->nu2 = 2;
00142     thee->mgprol = 0;
00143     thee->mgdisc = 0;
00144     thee->omegal = 19.4e-1;
00145     thee->omegan = 9.0e-1;
00146     thee->ipcon = 3;
00147     thee->irite = 8;
00148     thee->xcen = 0.0;
00149     thee->ycen = 0.0;
00150     thee->zcen = 0.0;
00151
00152     /* Default value for all APBS runs */
00153     thee->mgsmoo = 1;
00154     if (thee->nonlin == NONLIN_NPBE || thee->nonlin == NONLIN_SMPBE) {
00155         /* SMPBE Added - SMPBE needs to mimic NPBE */
00156         Vnm_print(0, "Vpmp_ctor2: Using meth = 1, mgsolv = 0\n");
00157         thee->mgsolv = 0;
00158     } else {
00159         /* Most rigorous (good for testing) */
00160         Vnm_print(0, "Vpmp_ctor2: Using meth = 2, mgsolv = 1\n");
00161         thee->mgsolv = 1;
00162     }
00163
00164     /* TEMPORARY USEAQUA */
00165     /* If we are using aqua, our solution method is either VSOL_CGMAqua or VSOL_NewtonAqua
        * so we need to temporarily override the mgsolve value and set it to 0
00166     */
00167     if (mgparm->useAqua == 1) thee->mgsolv = 0;
00168
00169     return 1;
00170 }
00171
00172
00173 /* ////////////////////////////////////////
00174 // Routine: Vpmgp_dtor
00175 //
00176 // Author: Nathan Baker
00177 VPUBLIC void Vpmgp_dtor(Vpmgp **thee) {
00178
00179     if ((*thee) != VNULL) {
00180         Vpmgp_dtor2(*thee);
00181         Vmem_free(VNULL, 1, sizeof(Vpmgp), (void **)thee);
00182         (*thee) = VNULL;
00183     }
00184 }
00185
00186 }

```

```

00187
00188 /* ///////////////////////////////////////////////////////////////////////////
00189 // Routine: Vpmgp_dtor2
00190 //
00191 // Author: Nathan Baker
00193 VPUBLIC void Vpmgp_dtor2(Vpmgp *thee) { ; }
00194
00195
00196 VPUBLIC void Vpmgp_size(
00197     Vpmgp *thee
00198 )
00199 {
00200
00201     int num_nf = 0;
00202     int num_narr = 2;
00203     int num_narrc = 27;
00204     int nxf, nyf, nzf, level, num_nf_oper, num_narrc_oper, n_band, nc_band, num_band, iretot;
00205
00206     thee->nf = thee->nx * thee->ny * thee->nz;
00207     thee->narr = thee->nf;
00208     nxf = thee->nx;
00209     nyf = thee->ny;
00210     nzf = thee->nz;
00211     thee->nxc = thee->nx;
00212     thee->nyc = thee->ny;
00213     thee->nzc = thee->nz;
00214
00215     for (level=2; level<=thee->nlev; level++) {
00216         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 passed by reference... */
00217         nxf = thee->nxc;
00218         nyf = thee->nyc;
00219         nzf = thee->nzc;
00220         thee->narr = thee->narr + (nxf * nyf * nzf);
00221     }
00222
00223     thee->nc = thee->nxc * thee->nyc * thee->nzc;
00224     thee->narrc = thee->narr - thee->nf;
00225
00226     /* Box or FEM discretization on fine grid? */
00227     switch (thee->mgdisc) { /* NAB TO-DO: This needs to be changed into an enumeration */
00228     case 0:
00229         num_nf_oper = 4;
00230         break;
00231     case 1:
00232         num_nf_oper = 14;
00233         break;
00234     default:
00235         Vnm_print(2, "Vpmgp_size: Invalid mgdisc value (%d)!\n", thee->mgdisc);
00236         VASSERT(0);
00237     }
00238
00239     /* Galerkin or standard coarsening? */
00240     switch (thee->mgcoar) { /* NAB TO-DO: This needs to be changed into an enumeration */
00241     case 0:
00242         if (thee->mgdisc != 0) {
00243             Vnm_print(2, "Vpmgp_size: Invalid mgcoar value (%d); must be used with mgdisc 0!\n", thee->
mgcoar);
00244             VASSERT(0);
00245         }
00246         num_narrc_oper = 4;
00247         break;
00248     case 1:
00249         if (thee->mgdisc != 0) {
00250             Vnm_print(2, "Vpmgp_size: Invalid mgcoar value (%d); must be used with mgdisc 0!\n", thee->
mgcoar);
00251             VASSERT(0);
00252         }
00253         num_narrc_oper = 14;
00254         break;
00255     case 2:
00256         num_narrc_oper = 14;
00257         break;
00258     default:
00259         Vnm_print(2, "Vpmgp_size: Invalid mgcoar value (%d)!\n", thee->mgcoar);
00260         VASSERT(0);
00261     }
00262
00263     /* LINPACK storage on coarse grid */
00264     switch (thee->mgsovl) { /* NAB TO-DO: This needs to be changed into an enumeration */
00265     case 0:

```

```

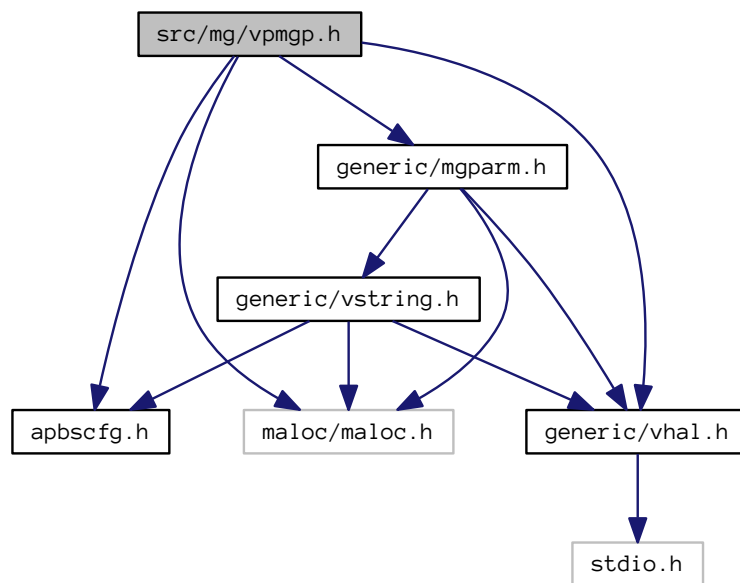
00266         n_band = 0;
00267         break;
00268     case 1:
00269         if ( ( (thee->mgcoar == 0) || (thee->mgcoar == 1)) && (thee->mgdisc == 0) ) {
00270             num_band = 1 + (thee->nxc-2)*(thee->nyc-2);
00271         } else {
00272             num_band = 1 + (thee->nxc-2)*(thee->nyc-2) + (thee->nxc-2) + 1;
00273         }
00274         nc_band = (thee->nxc-2)*(thee->nyc-2)*(thee->nzc-2);
00275         n_band = nc_band * num_band;
00276         break;
00277     default:
00278         Vnm_print(2, "Vpmgp_size: Invalid mgsolv value (%d)!\n", thee->mgsolv);
00279         VASSERT(0);
00280     }
00281
00282     /* Real storage parameters */
00283     thee->n_rpc = 100*(thee->nlev+1);
00284
00285     /* Resulting total required for real storage */
00286     thee->n_rwk = num_narr*thee->narr + (size_t)(num_nf + num_nf_oper)*thee->nf + (size_t)(num_narrc +
num_narrc_oper)*thee->narrc + n_band + thee->n_rpc;
00287
00288     /* Integer storage parameters */
00289     thee->n_iz = 50*(thee->nlev+1);
00290     thee->n_ipc = 100*(thee->nlev+1);
00291     thee->niwk = thee->n_iz + thee->n_ipc;
00292 }
00293
00294 VPRIVATE int coarsenThis(int nOld) {
00295     int nOut;
00296
00297     nOut = (nOld - 1) / 2 + 1;
00298
00299     if (((nOut-1)*2) != (nOld-1)) {
00300         Vnm_print(2, "Vpmgp_makeCoarse: Warning! The grid dimensions you have chosen are not consistent
with the nlev you have specified!\n");
00301         Vnm_print(2, "Vpmgp_makeCoarse: This calculation will only work if you are running with mg-dummy
type.\n");
00302     }
00303     if (nOut < 1) {
00304         Vnm_print(2, "D'oh! You coarsened the grid below zero! How did you do that?\n");
00305         VASSERT(0);
00306     }
00307
00308     return nOut;
00309 }
00310
00311 VPUBLIC void Vpmgp_makeCoarse(
00312     int numLevel,
00313     int nxOld,
00314     int nyOld,
00315     int nzOld,
00316     int *nxNew,
00317     int *nyNew,
00318     int *nzNew
00319 )
00320 {
00321     int nxtmp, nytmp, nztmp, iLevel;
00322
00323     for (iLevel=0; iLevel<numLevel; iLevel++) {
00324         nxtmp = *nxNew;
00325         nytmp = *nyNew;
00326         nztmp = *nzNew;
00327         *nxNew = coarsenThis(nxtmp);
00328         *nyNew = coarsenThis(nytmp);
00329         *nzNew = coarsenThis(nztmp);
00330     }
00331 }
00332
00333
00334 }

```

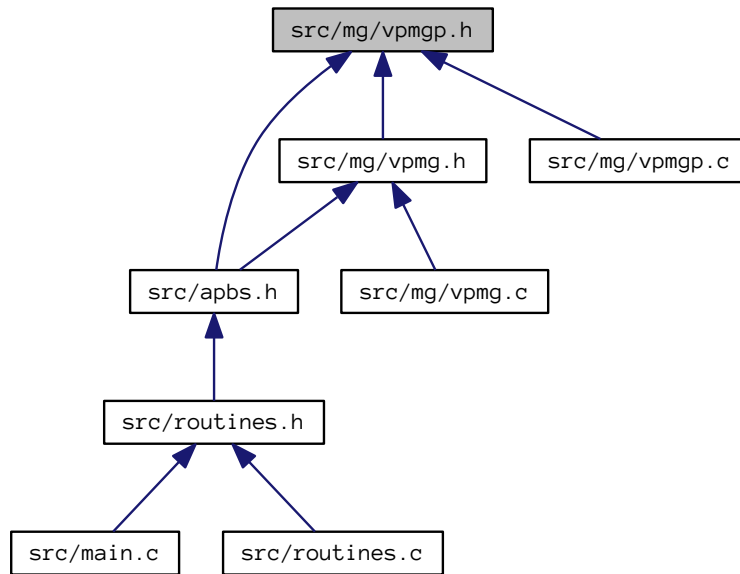
10.107 src/mg/vpmgp.h File Reference

Contains declarations for class Vpmgp.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/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)
FORTTRAN 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)
FORTTRAN 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.

10.107.1 Detailed Description

Contains declarations for class Vpmgp.

Version

\$Id\$

Author

Nathan A. Baker

Note

Variables and many default values taken directly from PMG

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
```

```

* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vpmgmp.h](#).

10.108 vpmgmp.h

```

00001
00064 #ifndef _VPMGMP_H_
00065 #define _VPMGMP_H_
00066
00067 #include "apbscfg.h"
00068
00069 #include "malloc/malloc.h"
00070
00071 #include "generic/vhal.h"
00072 #include "generic/mgparm.h"
00073
00080 struct sVpmgmp {
00081
00082     /* ***** USER-SPECIFIED PARAMETERS ***** */
00083     int nx;
00084     int ny;
00085     int nz;
00086     int nlev;
00087     double hx;
00088     double hy;
00089     double hzed;
00090     int nonlin;
00095     /* ***** DERIVED PARAMETERS ***** */
00096     int nxc;
00097     int nyc;
00098     int nzc;
00099     int nf;
00100     int nc;
00101     int narrc;
00102     int n_rpc;
00103     int n_iz;
00104     int n_ipc;
00106     size_t nrw;
00107     int niwk;
00108     int narr;
00109     int ipkey;
00117     /* ***** PARAMETERS WITH DEFAULT VALUES ***** */
00118     double xcent;
00119     double ycent;
00120     double zcent;
00121     double errtol;
00122     int itmax;
00123     int istop;
00130     int iinfo;
00135     Vbcfl bcfl;
00136     int key;
00139     int iperf;
00144     int meth;
00155     int mgkey;
00158     int nul;
00159     int nu2;
00160     int mgsmoo;
00166     int mgprol;
00170     int mgcoar;
00174     int mgsolv;
00177     int mgdisc;
00180     double omegal;
00181     double omegan;
00182     int irite;
00183     int ipcon;
00189     double xlen;
00190     double ylen;
00191     double zlen;

```

```

00192     double xmin;
00193     double ymin;
00194     double zmin;
00195     double xmax;
00196     double ymax;
00197     double zmax;
00198 };
00199
00204 typedef struct sVpmgp Vpmgp;
00205
00206 /* ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////// */
00207 // Class Vpmgp: Inlineable methods (vpmgp.c)
00208
00209 #if !defined(VINLINE_VPMGP)
00210 #else /* if defined(VINLINE_VPMGP) */
00211 #endif /* if !defined(VINLINE_VPMGP) */
00212
00213 /* ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////// */
00214 // Class Vpmgp: Non-Inlineable methods (vpmgp.c)
00215
00216 #if !defined(VINLINE_VPMGP)
00217 #else /* if defined(VINLINE_VPMGP) */
00218 #endif /* if !defined(VINLINE_VPMGP) */
00219
00224 VEXTERNC Vpmgp* Vpmgp_ctor(MGparm *mgparm);
00225
00226 VEXTERNC int Vpmgp_ctor2(Vpmgp *thee, MGparm *mgparm);
00227
00228 VEXTERNC void Vpmgp_dtor(Vpmgp **thee);
00229
00230 VEXTERNC void Vpmgp_dtor2(Vpmgp *thee);
00231
00232 VEXTERNC void Vpmgp_size(
00233     Vpmgp *thee
00234 );
00235
00236 VEXTERNC void Vpmgp_makeCoarse(
00237     int numLevel,
00238     int nxOld,
00239     int nyOld,
00240     int nzOld,
00241     int *nxNew,
00242     int *nyNew,
00243     int *nzNew
00244 );
00245
00246 #endif /* ifndef _VPMGP_H_ */

```

10.109 src/routines.c File Reference

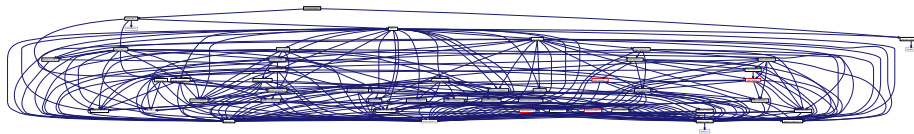
Supporting routines for APBS front end.

```

#include "routines.h"
#include "geoflow/cpbconcz2.h"

```

Include dependency graph for routines.c:



Functions

- VPUBLIC void [startVio](#) ()
Wrapper to start MALOC Vio layer.
- VPUBLIC [Vparam](#) * [loadParameter](#) ([NOsh](#) *nosh)
Loads and returns parameter object.

- VPUBLIC int [loadMolecules](#) (NOsh *nosh, Vparam *param, Valist *alist[NOSH_MAXMOL])
Load the molecules given in NOsh into atom lists.
- VPUBLIC void [killMolecules](#) (NOsh *nosh, Valist *alist[NOSH_MAXMOL])
Destroy the loaded molecules.
- VPUBLIC int [loadDielMaps](#) (NOsh *nosh, Vgrid *dielXMap[NOSH_MAXMOL], Vgrid *dielYMap[NOSH_MAXMOL], Vgrid *dielZMap[NOSH_MAXMOL])
Load the dielectric maps given in NOsh into grid objects.
- VPUBLIC void [killDielMaps](#) (NOsh *nosh, Vgrid *dielXMap[NOSH_MAXMOL], Vgrid *dielYMap[NOSH_MAXMOL], Vgrid *dielZMap[NOSH_MAXMOL])
Destroy the loaded dielectric.
- VPUBLIC int [loadKappaMaps](#) (NOsh *nosh, Vgrid *map[NOSH_MAXMOL])
Load the kappa maps given in NOsh into grid objects.
- VPUBLIC void [killKappaMaps](#) (NOsh *nosh, Vgrid *map[NOSH_MAXMOL])
Destroy the loaded kappa maps.
- VPUBLIC int [loadPotMaps](#) (NOsh *nosh, Vgrid *map[NOSH_MAXMOL])
Load the potential maps given in NOsh into grid objects.
- VPUBLIC void [killPotMaps](#) (NOsh *nosh, Vgrid *map[NOSH_MAXMOL])
Destroy the loaded potential maps.
- VPUBLIC int [loadChargeMaps](#) (NOsh *nosh, Vgrid *map[NOSH_MAXMOL])
Load the charge maps given in NOsh into grid objects.
- VPUBLIC void [killChargeMaps](#) (NOsh *nosh, Vgrid *map[NOSH_MAXMOL])
Destroy the loaded charge maps.
- VPUBLIC void [printPBEPARM](#) (PBEParm *pbeparm)
Print out generic PBE params loaded from input.
- VPUBLIC void [printMGPARAM](#) (MGparm *mgparm, double realCenter[3])
Print out MG-specific params loaded from input.
- VPUBLIC int [initMG](#) (int icalc, NOsh *nosh, MGparm *mgparm, PBEParm *pbeparm, double realCenter[3], Vpbe *pbe[NOSH_MAXCALC], Valist *alist[NOSH_MAXMOL], Vgrid *dielXMap[NOSH_MAXMOL], Vgrid *dielYMap[NOSH_MAXMOL], Vgrid *dielZMap[NOSH_MAXMOL], Vgrid *kappaMap[NOSH_MAXMOL], Vgrid *chargeMap[NOSH_MAXMOL], Vpmgp *pmgp[NOSH_MAXCALC], Vpmg *pmg[NOSH_MAXCALC], Vgrid *potMap[NOSH_MAXMOL])
Initialize an MG calculation.
- VPUBLIC void [killMG](#) (NOsh *nosh, Vpbe *pbe[NOSH_MAXCALC], Vpmgp *pmgp[NOSH_MAXCALC], Vpmg *pmg[NOSH_MAXCALC])
Kill structures initialized during an MG calculation.
- VPUBLIC int [solveMG](#) (NOsh *nosh, Vpmg *pmg, MGparm_CalcType type)
Solve the PBE with MG.
- VPUBLIC int [setPartMG](#) (NOsh *nosh, MGparm *mgparm, Vpmg *pmg)
Set MG partitions for calculating observables and performing I/O.
- VPUBLIC int [energyMG](#) (NOsh *nosh, int icalc, Vpmg *pmg, int *nenergy, double *totEnergy, double *qfEnergy, double *qmEnergy, double *dielEnergy)
Calculate electrostatic energies from MG solution.
- VPUBLIC int [forceMG](#) (Vmem *mem, NOsh *nosh, PBEParm *pbeparm, MGparm *mgparm, Vpmg *pmg, int *nforce, AtomForce **atomForce, Valist *alist[NOSH_MAXMOL])
Calculate forces from MG solution.
- VPUBLIC void [killEnergy](#) ()
Kill arrays allocated for energies.
- VPUBLIC void [killForce](#) (Vmem *mem, NOsh *nosh, int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC])

Free memory from MG force calculation.

- VPUBLIC int [writematMG](#) (int rank, [NOsh](#) *nosh, [PBEparm](#) *pbeparm, [Vpmg](#) *pmg)

Write out operator matrix from MG calculation to file.

- VPUBLIC void [storeAtomEnergy](#) ([Vpmg](#) *pmg, int icalc, double **atomEnergy, int *nenergy)

Store energy in arrays for future use.

- VPUBLIC int [writedataFlat](#) ([NOsh](#) *nosh, [Vcom](#) *com, const char *fname, double totEnergy[[NOSH_MAXCALC](#)], double qfEnergy[[NOSH_MAXCALC](#)], double qmEnergy[[NOSH_MAXCALC](#)], double dielEnergy[[NOSH_MAXCALC](#)], int nenergy[[NOSH_MAXCALC](#)], double *atomEnergy[[NOSH_MAXCALC](#)], int nforce[[NOSH_MAXCALC](#)], [AtomForce](#) *atomForce[[NOSH_MAXCALC](#)])

Write out information to a flat file.

- VPUBLIC int [writedataXML](#) ([NOsh](#) *nosh, [Vcom](#) *com, const char *fname, double totEnergy[[NOSH_MAXCALC](#)], double qfEnergy[[NOSH_MAXCALC](#)], double qmEnergy[[NOSH_MAXCALC](#)], double dielEnergy[[NOSH_MAXCALC](#)], int nenergy[[NOSH_MAXCALC](#)], double *atomEnergy[[NOSH_MAXCALC](#)], int nforce[[NOSH_MAXCALC](#)], [AtomForce](#) *atomForce[[NOSH_MAXCALC](#)])

Write out information to an XML file.

- VPUBLIC int [writedataMG](#) (int rank, [NOsh](#) *nosh, [PBEparm](#) *pbeparm, [Vpmg](#) *pmg)

Write out observables from MG calculation to file.

- VPUBLIC double [returnEnergy](#) ([Vcom](#) *com, [NOsh](#) *nosh, double totEnergy[[NOSH_MAXCALC](#)], int iprint)

Access net local energy.

- VPUBLIC int [printEnergy](#) ([Vcom](#) *com, [NOsh](#) *nosh, double totEnergy[[NOSH_MAXCALC](#)], int iprint)

Combine and pretty-print energy data (deprecated...see printElecEnergy)

- VPUBLIC int [printElecEnergy](#) ([Vcom](#) *com, [NOsh](#) *nosh, double totEnergy[[NOSH_MAXCALC](#)], int iprint)

Combine and pretty-print energy data.

- VPUBLIC int [printApolEnergy](#) ([NOsh](#) *nosh, int iprint)

Combine and pretty-print energy data.

- VPUBLIC int [printForce](#) ([Vcom](#) *com, [NOsh](#) *nosh, int nforce[[NOSH_MAXCALC](#)], [AtomForce](#) *atomForce[[NOSH_MAXCALC](#)], int iprint)

Combine and pretty-print force data (deprecated...see printElecForce)

- VPUBLIC int [printElecForce](#) ([Vcom](#) *com, [NOsh](#) *nosh, int nforce[[NOSH_MAXCALC](#)], [AtomForce](#) *atomForce[[NOSH_MAXCALC](#)], int iprint)

Combine and pretty-print force data.

- VPUBLIC int [printApolForce](#) ([Vcom](#) *com, [NOsh](#) *nosh, int nforce[[NOSH_MAXCALC](#)], [AtomForce](#) *atomForce[[NOSH_MAXCALC](#)], int iprint)

Combine and pretty-print force data.

- VPUBLIC void [killFE](#) ([NOsh](#) *nosh, [Vpbe](#) *pbe[[NOSH_MAXCALC](#)], [Vfetk](#) *fetk[[NOSH_MAXCALC](#)], [Gem](#) *gm[[NOSH_MAXMOL](#)])

Kill structures initialized during an FE calculation.

- VPUBLIC [Vrc_Codes](#) [initFE](#) (int icalc, [NOsh](#) *nosh, [FEMparm](#) *feparm, [PBEparm](#) *pbeparm, [Vpbe](#) *pbe[[NOSH_MAXCALC](#)], [Valist](#) *alist[[NOSH_MAXMOL](#)], [Vfetk](#) *fetk[[NOSH_MAXCALC](#)])

Initialize FE solver objects.

- VPUBLIC void [printFEPARM](#) (int icalc, [NOsh](#) *nosh, [FEMparm](#) *feparm, [Vfetk](#) *fetk[[NOSH_MAXCALC](#)])

Print out FE-specific params loaded from input.

- VPUBLIC int [partFE](#) (int icalc, [NOsh](#) *nosh, [FEMparm](#) *feparm, [Vfetk](#) *fetk[[NOSH_MAXCALC](#)])

Partition mesh (if applicable)

- VPUBLIC int [preRefineFE](#) (int icalc, [FEMparm](#) *feparm, [Vfetk](#) *fetk[[NOSH_MAXCALC](#)])

Pre-refine mesh before solve.

- VPUBLIC int [solveFE](#) (int icalc, [PBEparm](#) *pbeparm, [FEMparm](#) *feparm, [Vfetk](#) *fetk[[NOSH_MAXCALC](#)])

Solve-estimate-refine.

- VPUBLIC int [energyFE](#) ([NOsh](#) *nosh, int icalc, [Vfetk](#) *fetk[[NOSH_MAXCALC](#)], int *nenergy, double *totEnergy, double *qfEnergy, double *qmEnergy, double *dielEnergy)
Calculate electrostatic energies from FE solution.
- VPUBLIC int [postRefineFE](#) (int icalc, [FEMparm](#) *feparm, [Vfetk](#) *fetk[[NOSH_MAXCALC](#)])
Estimate error, mark mesh, and refine mesh after solve.
- VPUBLIC int [writedataFE](#) (int rank, [NOsh](#) *nosh, [PBEparm](#) *pbeparm, [Vfetk](#) *fetk)
Write FEM data to files.
- VPUBLIC int [initAPOL](#) ([NOsh](#) *nosh, Vmem *mem, [Vparam](#) *param, [APOLparm](#) *apolparm, int *nforce, [AtomForce](#) **atomForce, [Valist](#) *alist)
Upperlevel routine to the non-polar energy and force routines.
- VPUBLIC int [energyAPOL](#) ([APOLparm](#) *apolparm, double sasa, double sav, double atomsasa[], double atomwcaEnergy[], int numatoms)
Calculate non-polar energies.
- VPUBLIC int [forceAPOL](#) ([Vacc](#) *acc, Vmem *mem, [APOLparm](#) *apolparm, int *nforce, [AtomForce](#) **atomForce, [Valist](#) *alist, [Vclist](#) *clist)
Calculate non-polar forces.
- VPUBLIC int [initGEOFLOW](#) (int icalc, [NOsh](#) *nosh, [GEOFLOWparm](#) *bemparm, [PBEparm](#) *pbeparm, [Vpbe](#) *pbe[[NOSH_MAXCALC](#)])
Initialize an GEOFLOW calculation.
- VPUBLIC void [killGEOFLOW](#) ([NOsh](#) *nosh, [Vpbe](#) *pbe[[NOSH_MAXCALC](#)])
Kill structures initialized during an GEOFLOW calculation.
- VPUBLIC int [solveGEOFLOW](#) ([Valist](#) *molecules[[NOSH_MAXMOL](#)], [NOsh](#) *nosh, [PBEparm](#) *pbeparm, [APOLparm](#) *apolparm, [GEOFLOWparm](#) *parm, [GEOFLOWparm_CalcType](#) type)
Solve the PBE with GEOFLOW.
- VPUBLIC int [setPartGEOFLOW](#) ([NOsh](#) *nosh, [GEOFLOWparm](#) *GEOFLOWparm)
Set GEOFLOW partitions for calculating observables and performing I/O.
- VPUBLIC int [energyGEOFLOW](#) ([NOsh](#) *nosh, int icalc, int *nenergy, double *totEnergy, double *qfEnergy, double *qmEnergy, double *dielEnergy)
Calculate electrostatic energies from GEOFLOW solution.
- VPUBLIC int [forceGEOFLOW](#) ([NOsh](#) *nosh, [PBEparm](#) *pbeparm, [GEOFLOWparm](#) *parm, int *nforce, [AtomForce](#) **atomForce, [Valist](#) *alist[[NOSH_MAXMOL](#)])
Calculate forces from GEOFLOW solution.
- VPUBLIC void [printGEOFLOWPARAM](#) ([GEOFLOWparm](#) *parm)
Print out GEOFLOW-specific params loaded from input.
- VPUBLIC int [writedataGEOFLOW](#) (int rank, [NOsh](#) *nosh, [PBEparm](#) *pbeparm)
Write out observables from GEOFLOW calculation to file.
- VPUBLIC int [writematGEOFLOW](#) (int rank, [NOsh](#) *nosh, [PBEparm](#) *pbeparm)
Write out operator matrix from GEOFLOW calculation to file.

10.109.1 Detailed Description

Supporting routines for APBS front end.

Author

Nathan Baker

Version

\$Id\$

Attention

```

* APBS -- Adaptive Poisson-Boltzmann Solver
*
*   Nathan A. Baker (nathan.baker@pnnl.gov)
*   Pacific Northwest National Laboratory
*
*   Additional contributing authors listed in the code documentation.
*
*   Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
*   Pacific Northwest National Laboratory, operated by Battelle Memorial
*   Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
*   Portions Copyright (c) 2002-2010, Washington University in St. Louis.
*   Portions Copyright (c) 2002-2010, Nathan A. Baker.
*   Portions Copyright (c) 1999-2002, The Regents of the University of
*   California.
*   Portions Copyright (c) 1995, Michael Holst.
*   All rights reserved.
*
*   Redistribution and use in source and binary forms, with or without
*   modification, are permitted provided that the following conditions are met:
*
*   Redistributions of source code must retain the above copyright notice, this
*   list of conditions and the following disclaimer.
*
*   Redistributions in binary form must reproduce the above copyright notice,
*   this list of conditions and the following disclaimer in the documentation
*   and/or other materials provided with the distribution.
*
*   Neither the name of the developer nor the names of its contributors may be
*   used to endorse or promote products derived from this software without
*   specific prior written permission.
*
*   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
*   AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
*   IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
*   ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
*   LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
*   CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
*   SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
*   INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
*   CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
*   ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
*   THE POSSIBILITY OF SUCH DAMAGE.
*

```

Definition in file [routines.c](#).

10.110 routines.c

```

00001
00054 #include "routines.h"
00055 #include "geoflow/cpbconcz2.h"
00056 #include "EMBED(rcsid="$Id$")
00057
00058 VPUBLIC void startVio() { Vio_start(); }
00059
00060 VPUBLIC Vparam* loadParameter(NOsh *nosh) {
00061
00062     Vparam *param = VNULL;

```

```

00063
00064     if (nosh->gotparm) {
00065         param = Vparam_ctor();
00066         switch (nosh->parmfmt) {
00067             case NPF_FLAT:
00068                 Vnm_tprint( 1, "Reading parameter data from %s.\n",
00069                     nosh->parmpath);
00070                 if (Vparam_readFlatFile(param, "FILE", "ASC", VNULL,
00071                     nosh->parmpath) != 1) {
00072                     Vnm_tprint(2, "Error reading parameter file (%s)!\n", nosh->
00073                         parmpath);
00074                     return VNULL;
00075                 }
00076                 break;
00077             case NPF_XML:
00078                 Vnm_tprint( 1, "Reading parameter data from %s.\n",
00079                     nosh->parmpath);
00080                 if (Vparam_readXMLFile(param, "FILE", "ASC", VNULL,
00081                     nosh->parmpath) != 1) {
00082                     Vnm_tprint(2, "Error reading parameter file (%s)!\n", nosh->
00083                         parmpath);
00084                     return VNULL;
00085                 }
00086                 break;
00087             default:
00088                 Vnm_tprint(2, "Error! Undefined parameter file type (%d)!\n", nosh->
00089                     parmfmt);
00090                 return VNULL;
00091         } /* switch parmfmt */
00092     }
00093     return param;
00094 }
00095 VPUBLIC int loadMolecules(NOSH *nosh, Vparam *param,
00096     Valist *alist[NOSH_MAXMOL]) {
00097     int i;
00098     int use_params = 0;
00099     Vrc_Codes rc;
00100
00101     Vio *sock = VNULL;
00102
00103     Vnm_tprint( 1, "Got paths for %d molecules\n", nosh->nmol);
00104     if (nosh->nmol <= 0) {
00105         Vnm_tprint(2, "You didn't specify any molecules (correctly)!\n");
00106         Vnm_tprint(2, "Bailing out!\n");
00107         return 0;
00108     }
00109
00110     if (nosh->gotparm) {
00111         if (param == VNULL) {
00112             Vnm_tprint(2, "Error! You don't have a valid parameter object!\n");
00113             return 0;
00114         }
00115         use_params = 1;
00116     }
00117
00118     for (i=0; i<nosh->nmol; i++) {
00119         if(alist[i] == VNULL){
00120             alist[i] = Valist_ctor();
00121         }else{
00122             alist[i] = VNULL;
00123             alist[i] = Valist_ctor();
00124         }
00125
00126         switch (nosh->molfmt[i]) {
00127             case NMF_PQR:
00128                 /* Print out a warning to the user letting them know that we are overriding PQR
00129                     values for charge, radius and epsilon */
00130                 if (use_params) {
00131                     Vnm_print(2, "\nWARNING!! Radius/charge information from PQR file %s\n", nosh->
00132                         molpath[i]);
00133                     Vnm_print(2, "will be replaced with data from parameter file (%s)!\n", nosh->
00134                         parmpath);
00135                 }
00136                 Vnm_tprint( 1, "Reading PQR-format atom data from %s.\n",
00137                     nosh->molpath[i]);
00138                 sock = Vio_ctor("FILE", "ASC", VNULL, nosh->molpath[i], "r");
00139                 if (sock == VNULL) {

```



```
00138         Vnm_print(2, "Problem opening virtual socket %s!\n",
00139                 nosh->molpath[i]);
00140         return 0;
00141     }
00142     if (Vio_accept(sock, 0) < 0) {
00143         Vnm_print(2, "Problem accepting virtual socket %s!\n",
00144                 nosh->molpath[i]);
00145         return 0;
00146     }
00147     if (use_params) {
00148         rc = Valist_readPQR(alist[i], param, sock);
00149     } else {
00150         rc = Valist_readPQR(alist[i], VNULL, sock);
00151     }
00152     if (rc == 0) return 0;
00153
00154     Vio_acceptFree(sock);
00155     Vio_dtor(&sock);
00156     break;
00157 case NMF_PDB:
00158     /* Load parameters */
00159     if (!nosh->gotparm) {
00160         Vnm_tprint(2, "Nosh: Error! Can't read PDB without specifying PARM file!\n");
00161         return 0;
00162     }
00163     Vnm_tprint(1, "Reading PDB-format atom data from %s.\n",
00164             nosh->molpath[i]);
00165     sock = Vio_ctor("FILE", "ASC", VNULL, nosh->molpath[i], "r");
00166     if (sock == VNULL) {
00167         Vnm_print(2, "Problem opening virtual socket %s!\n",
00168                 nosh->molpath[i]);
00169         return 0;
00170     }
00171     if (Vio_accept(sock, 0) < 0) {
00172         Vnm_print(2, "Problem accepting virtual socket %s!\n", nosh->
00173                 molpath[i]);
00174         return 0;
00175     }
00176     rc = Valist_readPDB(alist[i], param, sock);
00177     /* If we are looking for an atom/residue that does not exist
00178      * then abort and return 0 */
00179     if (rc == 0)
00180         return 0;
00181
00182     Vio_acceptFree(sock);
00183     Vio_dtor(&sock);
00184     break;
00185 case NMF_XML:
00186     Vnm_tprint(1, "Reading XML-format atom data from %s.\n",
00187             nosh->molpath[i]);
00188     sock = Vio_ctor("FILE", "ASC", VNULL, nosh->molpath[i], "r");
00189     if (sock == VNULL) {
00190         Vnm_print(2, "Problem opening virtual socket %s!\n",
00191                 nosh->molpath[i]);
00192         return 0;
00193     }
00194     if (Vio_accept(sock, 0) < 0) {
00195         Vnm_print(2, "Problem accepting virtual socket %s!\n",
00196                 nosh->molpath[i]);
00197         return 0;
00198     }
00199     if (use_params) {
00200         rc = Valist_readXML(alist[i], param, sock);
00201     } else {
00202         rc = Valist_readXML(alist[i], VNULL, sock);
00203     }
00204     if (rc == 0)
00205         return 0;
00206
00207     Vio_acceptFree(sock);
00208     Vio_dtor(&sock);
00209     break;
00210 default:
00211     Vnm_tprint(2, "Nosh: Error! Undefined molecule file type \
00212 (%d)!\n", nosh->molfmt[i]);
00213     return 0;
00214 } /* switch molfmt */
00215
00216 if (rc != 1) {
00217     Vnm_tprint(2, "Error while reading molecule from %s\n",
00218             nosh->molpath[i]);
```

```

00218         return 0;
00219     }
00220
00221     Vnm_tprint( 1, "  %d atoms\n", Valist_getNumberAtoms(alist[i]));
00222     Vnm_tprint( 1, "  Centered at (%4.3e, %4.3e, %4.3e)\n",
00223         alist[i]->center[0], alist[i]->center[1],
00224         alist[i]->center[2]);
00225     Vnm_tprint( 1, "  Net charge %3.2e e\n", alist[i]->charge);
00226
00227 }
00228
00229 return 1;
00230
00231 }
00232
00233 VPUBLIC void killMolecules(NOsh *nosh, Valist *alist[
    NOSH_MAXMOL]) {
00234
00235     int i;
00236
00237 #ifndef VAPBSQUIET
00238     Vnm_tprint( 1, "Destroying %d molecules\n", nosh->nmol);
00239 #endif
00240
00241     for (i=0; i<nosh->nmol; i++)
00242         Valist_dtor(&(alist[i]));
00243
00244 }
00245
00250 VPUBLIC int loadDielMaps(NOsh *nosh,
00251     Vgrid *dielXMap[NOSH_MAXMOL],
00252     Vgrid *dielYMap[NOSH_MAXMOL],
00253     Vgrid *dielZMap[NOSH_MAXMOL]
00254 ) {
00255
00256     int i,
00257         ii,
00258         nx,
00259         ny,
00260         nz;
00261     double sum,
00262         hx,
00263         hy,
00264         hzed,
00265         xmin,
00266         ymin,
00267         zmin;
00268
00269     // Check to be sure we have dielectric map paths; if not, return.
00270     if (nosh->ndiel > 0)
00271         Vnm_tprint( 1, "Got paths for %d dielectric map sets\n",
00272             nosh->ndiel);
00273     else
00274         return 1;
00275
00276     // For each dielectric map path, read the data and calculate needed values.
00277     for (i=0; i<nosh->ndiel; i++) {
00278         Vnm_tprint( 1, "Reading x-shifted dielectric map data from \
00279 %s:\n", nosh->dielXpath[i]);
00280         dielXMap[i] = Vgrid_ctor(0, 0, 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, VNULL);
00281
00282         // Determine the format and read data if the format is valid.
00283         switch (nosh->dielfmt[i]) {
00284             // OpenDX (Data Explorer) format
00285             case VDF_DX:
00286                 if (Vgrid_readDX(dielXMap[i], "FILE", "ASC", VNULL,
00287                     nosh->dielXpath[i] != 1) {
00288                     Vnm_tprint( 2, "Fatal error while reading from %s\n",
00289                         nosh->dielXpath[i]);
00290                     return 0;
00291                 }
00292
00293                 // Set grid sizes
00294                 nx = dielXMap[i]->nx;
00295                 ny = dielXMap[i]->ny;
00296                 nz = dielXMap[i]->nz;
00297
00298                 // Set spacings
00299                 hx = dielXMap[i]->hx;
00300                 hy = dielXMap[i]->hy;
00301                 hzed = dielXMap[i]->hzed;

```

```

00302
00303         // Set minimum lower corner
00304         xmin = dielXMap[i]->xmin;
00305         ymin = dielXMap[i]->ymin;
00306         zmin = dielXMap[i]->zmin;
00307         Vnm_tprint(1, "   %d x %d x %d grid\n", nx, ny, nz);
00308         Vnm_tprint(1, "   (%g, %g, %g) A spacings\n", hx, hy, hzed);
00309         Vnm_tprint(1, "   (%g, %g, %g) A lower corner\n",
00310             xmin, ymin, zmin);
00311         sum = 0;
00312         for (ii=0; ii<(nx*ny*nz); ii++)
00313             sum += (dielXMap[i]->data[ii]);
00314         sum = sum*hx*hy*hzed;
00315         Vnm_tprint(1, "   Volume integral = %3.2e A^3\n", sum);
00316         break;
00317     // Binary file (GZip)
00318     case VDF_GZ:
00319         if (Vgrid_readGZ(dielXMap[i], nosh->dielXpath[i]) != 1) {
00320             Vnm_tprint( 2, "Fatal error while reading from %s\n",
00321                 nosh->dielXpath[i]);
00322             return 0;
00323         }
00324
00325         // Set grid sizes
00326         nx = dielXMap[i]->nx;
00327         ny = dielXMap[i]->ny;
00328         nz = dielXMap[i]->nz;
00329
00330         // Set spacings
00331         hx = dielXMap[i]->hx;
00332         hy = dielXMap[i]->hy;
00333         hzed = dielXMap[i]->hzed;
00334
00335         // Set minimum lower corner
00336         xmin = dielXMap[i]->xmin;
00337         ymin = dielXMap[i]->ymin;
00338         zmin = dielXMap[i]->zmin;
00339         Vnm_tprint(1, "   %d x %d x %d grid\n", nx, ny, nz);
00340         Vnm_tprint(1, "   (%g, %g, %g) A spacings\n", hx, hy, hzed);
00341         Vnm_tprint(1, "   (%g, %g, %g) A lower corner\n",
00342             xmin, ymin, zmin);
00343         sum = 0;
00344         for (ii=0; ii<(nx*ny*nz); ii++)
00345             sum += (dielXMap[i]->data[ii]);
00346         sum = sum*hx*hy*hzed;
00347         Vnm_tprint(1, "   Volume integral = %3.2e A^3\n", sum);
00348         break;
00349     // UHBD format
00350     case VDF_UHBD:
00351         Vnm_tprint( 2, "UHBD input not supported yet!\n");
00352         return 0;
00353     // AVS UCD format
00354     case VDF_AVS:
00355         Vnm_tprint( 2, "AVS input not supported yet!\n");
00356         return 0;
00357     // FEtk MC Simplex Format (MCSF)
00358     case VDF_MCSF:
00359         Vnm_tprint( 2, "MCSF input not supported yet!\n");
00360         return 0;
00361     default:
00362         Vnm_tprint( 2, "Invalid data format (%d)!\n",
00363             nosh->dielfmt[i]);
00364         return 0;
00365 }
00366 Vnm_tprint( 1, "Reading y-shifted dielectric map data from \
00367 %s:\n", nosh->dielYpath[i]);
00368 dielYMap[i] = Vgrid_ctor(0, 0, 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, VNULL);
00369
00370 // Determine the format and read data if the format is valid.
00371 switch (nosh->dielfmt[i]) {
00372     // OpenDX (Data Explorer) format
00373     case VDF_DX:
00374         if (Vgrid_readDX(dielYMap[i], "FILE", "ASC", VNULL,
00375             nosh->dielYpath[i]) != 1) {
00376             Vnm_tprint( 2, "Fatal error while reading from %s\n",
00377                 nosh->dielYpath[i]);
00378             return 0;
00379         }
00380
00381         // Read grid
00382         nx = dielYMap[i]->nx;

```

```

00383         ny = dielYMap[i]->ny;
00384         nz = dielYMap[i]->nz;
00385
00386         // Read spacings
00387         hx = dielYMap[i]->hx;
00388         hy = dielYMap[i]->hy;
00389         hzed = dielYMap[i]->hzed;
00390
00391         // Read minimum lower corner
00392         xmin = dielYMap[i]->xmin;
00393         ymin = dielYMap[i]->ymin;
00394         zmin = dielYMap[i]->zmin;
00395         Vnm_tprint(1, "  %d x %d x %d grid\n", nx, ny, nz);
00396         Vnm_tprint(1, "  (%g, %g, %g) A spacings\n", hx, hy, hzed);
00397         Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
00398                 xmin, ymin, zmin);
00399         sum = 0;
00400         for (ii=0; ii<(nx*ny*nz); ii++)
00401             sum += (dielYMap[i]->data[ii]);
00402         sum = sum*hx*hy*hzed;
00403         Vnm_tprint(1, "  Volume integral = %3.2e A^3\n", sum);
00404         break;
00405     // Binary file (GZip) format
00406     case VDF_GZ:
00407         if (Vgrid_readGZ(dielYMap[i], nosh->dielYpath[i]) != 1) {
00408             Vnm_tprint( 2, "Fatal error while reading from %s\n",
00409                     nosh->dielYpath[i]);
00410             return 0;
00411         }
00412
00413         // Read grid
00414         nx = dielYMap[i]->nx;
00415         ny = dielYMap[i]->ny;
00416         nz = dielYMap[i]->nz;
00417
00418         // Read spacings
00419         hx = dielYMap[i]->hx;
00420         hy = dielYMap[i]->hy;
00421         hzed = dielYMap[i]->hzed;
00422
00423         // Read minimum lower corner
00424         xmin = dielYMap[i]->xmin;
00425         ymin = dielYMap[i]->ymin;
00426         zmin = dielYMap[i]->zmin;
00427         Vnm_tprint(1, "  %d x %d x %d grid\n", nx, ny, nz);
00428         Vnm_tprint(1, "  (%g, %g, %g) A spacings\n", hx, hy, hzed);
00429         Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
00430                 xmin, ymin, zmin);
00431         sum = 0;
00432         for (ii=0; ii<(nx*ny*nz); ii++)
00433             sum += (dielYMap[i]->data[ii]);
00434         sum = sum*hx*hy*hzed;
00435         Vnm_tprint(1, "  Volume integral = %3.2e A^3\n", sum);
00436         break;
00437     // UHBD format
00438     case VDF_UHBD:
00439         Vnm_tprint( 2, "UHBD input not supported yet!\n");
00440         return 0;
00441     // AVS UCD format
00442     case VDF_AVS:
00443         Vnm_tprint( 2, "AVS input not supported yet!\n");
00444         return 0;
00445     // FETk MC Simplex Format (MCSF)
00446     case VDF_MCSF:
00447         Vnm_tprint( 2, "MCSF input not supported yet!\n");
00448         return 0;
00449     default:
00450         Vnm_tprint( 2, "Invalid data format (%d)!\n",
00451                 nosh->dielfmt[i]);
00452         return 0;
00453 }
00454
00455 Vnm_tprint( 1, "Reading z-shifted dielectric map data from \
00456 %s:\n", nosh->dielZpath[i]);
00457 dielZMap[i] = Vgrid_ctor(0, 0, 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, VNULL);
00458
00459 // Determine the format and read data if the format is valid.
00460 switch (nosh->dielfmt[i]) {
00461     // OpenDX (Data Explorer) format
00462     case VDF_DX:
00463         if (Vgrid_readDX(dielZMap[i], "FILE", "ASC", VNULL,

```

```

00464             nosh->dielZpath[i]) != 1) {
00465         Vnm_tprint( 2, "Fatal error while reading from %s\n",
00466             nosh->dielZpath[i]);
00467         return 0;
00468     }
00469
00470     // Read grid
00471     nx = dielZMap[i]->nx;
00472     ny = dielZMap[i]->ny;
00473     nz = dielZMap[i]->nz;
00474
00475     // Read spacings
00476     hx = dielZMap[i]->hx;
00477     hy = dielZMap[i]->hy;
00478     hzed = dielZMap[i]->hzed;
00479
00480     // Read minimum lower corner
00481     xmin = dielZMap[i]->xmin;
00482     ymin = dielZMap[i]->ymin;
00483     zmin = dielZMap[i]->zmin;
00484     Vnm_tprint(1, " %d x %d x %d grid\n",
00485         nx, ny, nz);
00486     Vnm_tprint(1, " (%g, %g, %g) A spacings\n",
00487         hx, hy, hzed);
00488     Vnm_tprint(1, " (%g, %g, %g) A lower corner\n",
00489         xmin, ymin, zmin);
00490     sum = 0;
00491     for (ii=0; ii<(nx*ny*nz); ii++) sum += (dielZMap[i]->data[ii]);
00492     sum = sum*hx*hy*hzed;
00493     Vnm_tprint(1, " Volume integral = %3.2e A^3\n", sum);
00494     break;
00495 // Binary file (GZip) format
00496 case VDF_GZ:
00497     if (Vgrid_readGZ(dielZMap[i], nosh->dielZpath[i]) != 1) {
00498         Vnm_tprint( 2, "Fatal error while reading from %s\n",
00499             nosh->dielZpath[i]);
00500         return 0;
00501     }
00502
00503     // Read grid
00504     nx = dielZMap[i]->nx;
00505     ny = dielZMap[i]->ny;
00506     nz = dielZMap[i]->nz;
00507
00508     // Read spacings
00509     hx = dielZMap[i]->hx;
00510     hy = dielZMap[i]->hy;
00511     hzed = dielZMap[i]->hzed;
00512
00513     // Read minimum lower corner
00514     xmin = dielZMap[i]->xmin;
00515     ymin = dielZMap[i]->ymin;
00516     zmin = dielZMap[i]->zmin;
00517     Vnm_tprint(1, " %d x %d x %d grid\n",
00518         nx, ny, nz);
00519     Vnm_tprint(1, " (%g, %g, %g) A spacings\n",
00520         hx, hy, hzed);
00521     Vnm_tprint(1, " (%g, %g, %g) A lower corner\n",
00522         xmin, ymin, zmin);
00523     sum = 0;
00524     for (ii=0; ii<(nx*ny*nz); ii++) sum += (dielZMap[i]->data[ii]);
00525     sum = sum*hx*hy*hzed;
00526     Vnm_tprint(1, " Volume integral = %3.2e A^3\n", sum);
00527     break;
00528 // UHBD format
00529 case VDF_UHBD:
00530     Vnm_tprint( 2, "UHBD input not supported yet!\n");
00531     return 0;
00532 // AVS UCD format
00533 case VDF_AVS:
00534     Vnm_tprint( 2, "AVS input not supported yet!\n");
00535     return 0;
00536 // FEtk MC Simplex Format (MCSF)
00537 case VDF_MCSF:
00538     Vnm_tprint( 2, "MCSF input not supported yet!\n");
00539     return 0;
00540 default:
00541     Vnm_tprint( 2, "Invalid data format (%d)!\n",
00542         nosh->dielfmt[i]);
00543     return 0;
00544 }

```

```

00545     }
00546
00547     return 1;
00548 }
00549
00550 VPUBLIC void killDielMaps(Nosh *nosh,
00551                          Vgrid *dielXMap[NOSH_MAXMOL],
00552                          Vgrid *dielYMap[NOSH_MAXMOL],
00553                          Vgrid *dielZMap[NOSH_MAXMOL]) {
00554
00555     int i;
00556
00557     if (nosh->ndiel > 0) {
00558 #ifndef VAPBSQUIET
00559         Vnm_tprint( 1, "Destroying %d dielectric map sets\n",
00560                   nosh->ndiel);
00561 #endif
00562         for (i=0; i<nosh->ndiel; i++) {
00563             Vgrid_dtor(&(dielXMap[i]));
00564             Vgrid_dtor(&(dielYMap[i]));
00565             Vgrid_dtor(&(dielZMap[i]));
00566         }
00567     }
00568     else return;
00569 }
00570
00571 VPUBLIC int loadKappaMaps(Nosh *nosh,
00572                          Vgrid *map[NOSH_MAXMOL]
00573                          ) {
00574
00575     int i,
00576         ii,
00577         len;
00578     double sum;
00579
00580     if (nosh->nkappa > 0)
00581         Vnm_tprint( 1, "Got paths for %d kappa maps\n", nosh->nkappa);
00582     else return 1;
00583
00584     for (i=0; i<nosh->nkappa; i++) {
00585         Vnm_tprint( 1, "Reading kappa map data from %s:\n",
00586                   nosh->kappapath[i]);
00587         map[i] = Vgrid_ctor(0, 0, 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, VNULL);
00588
00589         // Determine the format and read data if the format is valid.
00590         switch (nosh->kappafmt[i]) {
00591             // OpenDX (Data Explorer) format
00592             case VDF_DX:
00593                 if (Vgrid_readDX(map[i], "FILE", "ASC", VNULL,
00594                                nosh->kappapath[i]) != 1) {
00595                     Vnm_tprint( 2, "Fatal error while reading from %s\n",
00596                               nosh->kappapath[i]);
00597                     return 0;
00598                 }
00599                 Vnm_tprint(1, "  %d x %d x %d grid\n",
00600                           map[i]->nx, map[i]->ny, map[i]->nz);
00601                 Vnm_tprint(1, "  (%g, %g, %g) A spacings\n",
00602                           map[i]->hx, map[i]->hy, map[i]->hz);
00603                 Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
00604                           map[i]->xmin, map[i]->ymin, map[i]->zmin);
00605                 sum = 0;
00606                 for (ii = 0, len = map[i]->nx * map[i]->ny * map[i]->nz;
00607                     ii < len;
00608                     ii++)
00609                     {
00610                         sum += (map[i]->data[ii]);
00611                     }
00612                 sum = sum*map[i]->hx*map[i]->hy*map[i]->hz;
00613                 Vnm_tprint(1, "  Volume integral = %3.2e A^3\n", sum);
00614                 break;
00615             // UHBD format
00616             case VDF_UHBD:
00617                 Vnm_tprint( 2, "UHBD input not supported yet!\n");
00618                 return 0;
00619             // FETk MC Simplex Format (MCSF)
00620             case VDF_MCSF:
00621                 Vnm_tprint( 2, "MCSF input not supported yet!\n");
00622                 return 0;
00623             // AVS UCD format
00624             case VDF_AVS:

```

```

00629         Vnm_tprint( 2, "AVS input not supported yet!\n");
00630         return 0;
00631     // Binary file (GZip) format
00632     case VDF_GZ:
00633         if (Vgrid_readGZ(map[i], nosh->kappapath[i]) != 1) {
00634             Vnm_tprint( 2, "Fatal error while reading from %s\n",
00635                 nosh->kappapath[i]);
00636             return 0;
00637         }
00638         Vnm_tprint(1, "  %d x %d x %d grid\n",
00639             map[i]->nx, map[i]->ny, map[i]->nz);
00640         Vnm_tprint(1, "  (%g, %g, %g) A spacings\n",
00641             map[i]->hx, map[i]->hy, map[i]->hz);
00642         Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
00643             map[i]->xmin, map[i]->ymin, map[i]->zmin);
00644         sum = 0;
00645         for (ii=0, len=map[i]->nx*map[i]->ny*map[i]->nz; ii<len; ii++) {
00646             sum += (map[i]->data[ii]);
00647         }
00648         sum = sum*map[i]->hx*map[i]->hy*map[i]->hz;
00649         Vnm_tprint(1, "  Volume integral = %3.2e A^3\n", sum);
00650         break;
00651     default:
00652         Vnm_tprint( 2, "Invalid data format (%d)!\n",
00653             nosh->kappafmt[i]);
00654         return 0;
00655     }
00656 }
00657
00658 return 1;
00659
00660 }
00661
00662 VPUBLIC void killKappaMaps(NOsh *nosh, Vgrid *map[
    NOSH_MAXMOL]) {
00663     int i;
00664
00665     if (nosh->nkappa > 0) {
00666 #ifndef VAPBSQUIET
00667         Vnm_tprint( 1, "Destroying %d kappa maps\n", nosh->nkappa);
00668 #endif
00669         for (i=0; i<nosh->nkappa; i++) Vgrid_dtor(&(map[i]));
00670     }
00671     else return;
00672
00673 }
00674 }
00675
00676 VPUBLIC int loadPotMaps(NOsh *nosh,
    Vgrid *map[NOSH_MAXMOL]
    ) {
00677     int i,
00678         ii,
00679         len;
00680     double sum;
00681
00682     if (nosh->npot > 0)
00683         Vnm_tprint( 1, "Got paths for %d potential maps\n", nosh->npot);
00684     else return 1;
00685
00686     for (i=0; i<nosh->npot; i++) {
00687         Vnm_tprint( 1, "Reading potential map data from %s:\n",
00688             nosh->potpath[i]);
00689         map[i] = Vgrid_ctor(0, 0, 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, VNULL);
00690         switch (nosh->potfmt[i]) {
00691             // OpenDX (Data Explorer) format
00692             case VDF_DX:
00693                 // Binary file (GZip) format
00694                 case VDF_GZ:
00695                     if (nosh->potfmt[i] == VDF_DX) {
00696                         if (Vgrid_readDX(map[i], "FILE", "ASC", VNULL,
00697                             nosh->potpath[i]) != 1) {
00698                             Vnm_tprint( 2, "Fatal error while reading from %s\n",
00699                                 nosh->potpath[i]);
00700                             return 0;
00701                         }
00702                     }
00703                     else {
00704                         if (Vgrid_readGZ(map[i], nosh->potpath[i]) != 1) {
00705                             Vnm_tprint( 2, "Fatal error while reading from %s\n",
00706                                 nosh->potpath[i]);
00707                         }
00708                     }
00709                 }
00710             }
00711         }

```

```

00712         return 0;
00713     }
00714 }
00715 Vnm_tprint(1, " %d x %d x %d grid\n",
00716     map[i]->nx, map[i]->ny, map[i]->nz);
00717 Vnm_tprint(1, " (%g, %g, %g) A spacings\n",
00718     map[i]->hx, map[i]->hy, map[i]->hz);
00719 Vnm_tprint(1, " (%g, %g, %g) A lower corner\n",
00720     map[i]->xmin, map[i]->ymin, map[i]->zmin);
00721 sum = 0;
00722 for (ii=0, len=map[i]->nx*map[i]->ny*map[i]->nz; ii<len; ii++) {
00723     sum += (map[i]->data[ii]);
00724 }
00725 sum = sum*map[i]->hx*map[i]->hy*map[i]->hz;
00726 Vnm_tprint(1, " Volume integral = %3.2e A^3\n", sum);
00727 break;
00728 // UHBD format
00729 case VDF_UHBD:
00730     Vnm_tprint( 2, "UHBD input not supported yet!\n");
00731     return 0;
00732 // FETk MC Simplex Format (MCSF)
00733 case VDF_MCSF:
00734     Vnm_tprint( 2, "MCSF input not supported yet!\n");
00735     return 0;
00736 // AVS UCD format
00737 case VDF_AVS:
00738     Vnm_tprint( 2, "AVS input not supported yet!\n");
00739     return 0;
00740 default:
00741     Vnm_tprint( 2, "Invalid data format (%d)!\n",
00742         nosh->potfmt[i]);
00743     return 0;
00744 }
00745 }
00746
00747 return 1;
00748 }
00749 }
00750
00751 VPUBLIC void killPotMaps(NOsh *nosh,
00752     Vgrid *map[NOSH_MAXMOL]
00753 ) {
00754
00755     int i;
00756
00757     if (nosh->npot > 0) {
00758 #ifndef VAPBSQUIET
00759         Vnm_tprint( 1, "Destroying %d potential maps\n", nosh->npot);
00760 #endif
00761         for (i=0; i<nosh->npot; i++) Vgrid_dtor(&(map[i]));
00762     }
00763     else return;
00764 }
00765 }
00766
00770 VPUBLIC int loadChargeMaps(NOsh *nosh,
00771     Vgrid *map[NOSH_MAXMOL]
00772 ) {
00773
00774     int i,
00775         ii,
00776         len;
00777     double sum;
00778
00779     if (nosh->ncharge > 0)
00780         Vnm_tprint( 1, "Got paths for %d charge maps\n", nosh->ncharge);
00781     else return 1;
00782
00783     for (i=0; i<nosh->ncharge; i++) {
00784         Vnm_tprint( 1, "Reading charge map data from %s:\n",
00785             nosh->chargepath[i]);
00786         map[i] = Vgrid_ctor(0, 0, 0, 0.0, 0.0, 0.0, 0.0, 0.0, VNULL);
00787
00788         // Determine data format and read data
00789         switch (nosh->chargefmt[i]) {
00790             case VDF_DX:
00791                 if (Vgrid_readDX(map[i], "FILE", "ASC", VNULL,
00792                     nosh->chargepath[i]) != 1) {
00793                     Vnm_tprint( 2, "Fatal error while reading from %s\n",
00794                         nosh->chargepath[i]);
00795                     return 0;

```



```

00796         }
00797         Vnm_tprint(1, "  %d x %d x %d grid\n",
00798             map[i]->nx, map[i]->ny, map[i]->nz);
00799         Vnm_tprint(1, "    (%g, %g, %g) A spacings\n",
00800             map[i]->hx, map[i]->hy, map[i]->hz);
00801         Vnm_tprint(1, "    (%g, %g, %g) A lower corner\n",
00802             map[i]->xmin, map[i]->ymin, map[i]->zmin);
00803         sum = 0;
00804         for (ii=0, len=map[i]->nx*map[i]->ny*map[i]->nz; ii<len; ii++) {
00805             sum += (map[i]->data[ii]);
00806         }
00807         sum = sum*map[i]->hx*map[i]->hy*map[i]->hz;
00808         Vnm_tprint(1, "  Charge map integral = %3.2e e\n", sum);
00809         break;
00810     case VDF_UHBD:
00811         Vnm_tprint( 2, "UHBD input not supported yet!\n");
00812         return 0;
00813     case VDF_AVS:
00814         Vnm_tprint( 2, "AVS input not supported yet!\n");
00815         return 0;
00816     case VDF_MCSF:
00817         Vnm_tprint(2, "MCSF input not supported yet!\n");
00818         return 0;
00819     case VDF_GZ:
00820         if (Vgrid_readGZ(map[i], nosh->chargepath[i]) != 1) {
00821             Vnm_tprint( 2, "Fatal error while reading from %s\n",
00822                 nosh->chargepath[i]);
00823             return 0;
00824         }
00825         Vnm_tprint(1, "  %d x %d x %d grid\n",
00826             map[i]->nx, map[i]->ny, map[i]->nz);
00827         Vnm_tprint(1, "    (%g, %g, %g) A spacings\n",
00828             map[i]->hx, map[i]->hy, map[i]->hz);
00829         Vnm_tprint(1, "    (%g, %g, %g) A lower corner\n",
00830             map[i]->xmin, map[i]->ymin, map[i]->zmin);
00831         sum = 0;
00832         for (ii=0, len=map[i]->nx*map[i]->ny*map[i]->nz; ii<len; ii++) {
00833             sum += (map[i]->data[ii]);
00834         }
00835         sum = sum*map[i]->hx*map[i]->hy*map[i]->hz;
00836         Vnm_tprint(1, "  Charge map integral = %3.2e e\n", sum);
00837         break;
00838     default:
00839         Vnm_tprint( 2, "Invalid data format (%d)!\n",
00840             nosh->kappafmt[i]);
00841         return 0;
00842     }
00843 }
00844
00845 return 1;
00846
00847 }
00848
00849 VPUBLIC void killChargeMaps(Nosh *nosh,
00850     Vgrid *map[NOSH_MAXMOL]
00851 ) {
00852
00853     int i;
00854
00855     if (nosh->ncharge > 0) {
00856 #ifndef VAPBSQUIET
00857         Vnm_tprint( 1, "Destroying %d charge maps\n", nosh->ncharge);
00858 #endif
00859         for (i=0; i<nosh->ncharge; i++) Vgrid_dtor(&(map[i]));
00860     }
00861
00862     else return;
00863
00864 }
00865
00866
00867 VPUBLIC void printPBEPARM(PBEParm *pbeparm) {
00868
00869     int i;
00870     double ionstr = 0.0;
00871
00872     for (i=0; i<pbeparm->nion; i++)
00873         ionstr += 0.5*(VSQR(pbeparm->ionq[i])*pbeparm->ionc[i]);
00874
00875     Vnm_tprint( 1, "  Molecule ID: %d\n", pbeparm->molid);
00876     switch (pbeparm->pctype) {

```

```

00877     case PBE_NPBE:
00878         Vnm_tprint( 1, " Nonlinear traditional PBE\n");
00879         break;
00880     case PBE_LPBE:
00881         Vnm_tprint( 1, " Linearized traditional PBE\n");
00882         break;
00883     case PBE_NRPBE:
00884         Vnm_tprint( 1, " Nonlinear regularized PBE\n");
00885         Vnm_tprint( 2, " ** Sorry, but Nathan broke the nonlinear regularized PBE implementation. **\n
");
00886         Vnm_tprint( 2, " ** Please let us know if you are interested in using it. **\n");
00887         VASSERT(0);
00888         break;
00889     case PBE_LRPBE:
00890         Vnm_tprint( 1, " Linearized regularized PBE\n");
00891         break;
00892     case PBE_SMPBE: /* SMPBE Added */
00893         Vnm_tprint( 1, " Nonlinear Size-Modified PBE\n");
00894         break;
00895     default:
00896         Vnm_tprint(2, " Unknown PBE type (%d)!\n", pbeparm->pbetype);
00897         break;
00898     }
00899     if (pbeparm->bcfl == BCFL_ZERO) {
00900         Vnm_tprint( 1, " Zero boundary conditions\n");
00901     } else if (pbeparm->bcfl == BCFL_SDH) {
00902         Vnm_tprint( 1, " Single Debye-Huckel sphere boundary \
00903 conditions\n");
00904     } else if (pbeparm->bcfl == BCFL_MDH) {
00905         Vnm_tprint( 1, " Multiple Debye-Huckel sphere boundary \
00906 conditions\n");
00907     } else if (pbeparm->bcfl == BCFL_FOCUS) {
00908         Vnm_tprint( 1, " Boundary conditions from focusing\n");
00909     } else if (pbeparm->bcfl == BCFL_MAP) {
00910         Vnm_tprint( 1, " Boundary conditions from potential map\n");
00911     } else if (pbeparm->bcfl == BCFL_MEM) {
00912         Vnm_tprint( 1, " Membrane potential boundary conditions.\n");
00913     }
00914     Vnm_tprint( 1, " %d ion species (%4.3f M ionic strength):\n",
00915         pbeparm->nion, ionstr);
00916     for (i=0; i<pbeparm->nion; i++) {
00917         Vnm_tprint( 1, " %4.3f A-radius, %4.3f e-charge, \
00918 %4.3f M concentration\n",
00919             pbeparm->ionr[i], pbeparm->ionq[i], pbeparm->ionc[i]);
00920     }
00921
00922     if (pbeparm->pbetype == PBE_SMPBE) { /* SMPBE Added */
00923         Vnm_tprint( 1, " Lattice spacing: %4.3f A (SMPBE) \n", pbeparm->
smvolume);
00924         Vnm_tprint( 1, " Relative size parameter: %4.3f (SMPBE) \n", pbeparm->
smsize);
00925     }
00926
00927     Vnm_tprint( 1, " Solute dielectric: %4.3f\n", pbeparm->pdie);
00928     Vnm_tprint( 1, " Solvent dielectric: %4.3f\n", pbeparm->sdie);
00929     switch (pbeparm->srfm) {
00930     case 0:
00931         Vnm_tprint( 1, " Using \"molecular\" surface \
00932 definition; no smoothing\n");
00933         Vnm_tprint( 1, " Solvent probe radius: %4.3f A\n",
00934             pbeparm->srad);
00935         break;
00936     case 1:
00937         Vnm_tprint( 1, " Using \"molecular\" surface definition;\
00938 harmonic average smoothing\n");
00939         Vnm_tprint( 1, " Solvent probe radius: %4.3f A\n",
00940             pbeparm->srad);
00941         break;
00942     case 2:
00943         Vnm_tprint( 1, " Using spline-based surface definition;\
00944 window = %4.3f\n", pbeparm->swin);
00945         break;
00946     default:
00947         break;
00948     }
00949     Vnm_tprint( 1, " Temperature: %4.3f K\n", pbeparm->temp);
00950     if (pbeparm->calcenergy != PCE_NO) Vnm_tprint( 1, " Electrostatic \
00951 energies will be calculated\n");
00952     if (pbeparm->calcforce == PCF_TOTAL) Vnm_tprint( 1, " Net solvent \
00953 forces will be calculated \n");
00954     if (pbeparm->calcforce == PCF_COMPS) Vnm_tprint( 1, " All-atom \

```

```
00955 solvent forces will be calculated\n");
00956     for (i=0; i<pbeparm->numwrite; i++) {
00957         switch (pbeparm->writetype[i]) {
00958             case VDT_CHARGE:
00959                 Vnm_tprint(1, " Charge distribution to be written to ");
00960                 break;
00961             case VDT_POT:
00962                 Vnm_tprint(1, " Potential to be written to ");
00963                 break;
00964             case VDT_SMOL:
00965                 Vnm_tprint(1, " Molecular solvent accessibility \
00966 to be written to ");
00967                 break;
00968             case VDT_SSPL:
00969                 Vnm_tprint(1, " Spline-based solvent accessibility \
00970 to be written to ");
00971                 break;
00972             case VDT_VDW:
00973                 Vnm_tprint(1, " van der Waals solvent accessibility \
00974 to be written to ");
00975                 break;
00976             case VDT_IVDW:
00977                 Vnm_tprint(1, " Ion accessibility to be written to ");
00978                 break;
00979             case VDT_LAP:
00980                 Vnm_tprint(1, " Potential Laplacian to be written to ");
00981                 break;
00982             case VDT_EDENS:
00983                 Vnm_tprint(1, " Energy density to be written to ");
00984                 break;
00985             case VDT_NDENS:
00986                 Vnm_tprint(1, " Ion number density to be written to ");
00987                 break;
00988             case VDT_QDENS:
00989                 Vnm_tprint(1, " Ion charge density to be written to ");
00990                 break;
00991             case VDT_DIELX:
00992                 Vnm_tprint(1, " X-shifted dielectric map to be written \
00993 to ");
00994                 break;
00995             case VDT_DIELY:
00996                 Vnm_tprint(1, " Y-shifted dielectric map to be written \
00997 to ");
00998                 break;
00999             case VDT_DIELZ:
01000                 Vnm_tprint(1, " Z-shifted dielectric map to be written \
01001 to ");
01002                 break;
01003             case VDT_KAPPA:
01004                 Vnm_tprint(1, " Kappa map to be written to ");
01005                 break;
01006             case VDT_ATOMPOT:
01007                 Vnm_tprint(1, " Atom potentials to be written to ");
01008                 break;
01009             default:
01010                 Vnm_tprint(2, " Invalid data type for writing!\n");
01011                 break;
01012         }
01013         switch (pbeparm->writefmt[i]) {
01014             case VDF_DX:
01015                 Vnm_tprint(1, "%.s%.s\n", pbeparm->writestem[i], "dx");
01016                 break;
01017             case VDF_GZ:
01018                 Vnm_tprint(1, "%.s%.s\n", pbeparm->writestem[i], "dx.gz");
01019                 break;
01020             case VDF_UHBD:
01021                 Vnm_tprint(1, "%.s%.s\n", pbeparm->writestem[i], "grd");
01022                 break;
01023             case VDF_AVS:
01024                 Vnm_tprint(1, "%.s%.s\n", pbeparm->writestem[i], "ucd");
01025                 break;
01026             case VDF_MCSF:
01027                 Vnm_tprint(1, "%.s%.s\n", pbeparm->writestem[i], "mcsf");
01028                 break;
01029             case VDF_FLAT:
01030                 Vnm_tprint(1, "%.s%.s\n", pbeparm->writestem[i], "txt");
01031                 break;
01032             default:
01033                 Vnm_tprint(2, " Invalid format for writing!\n");
01034                 break;
01035         }
```

```

01036
01037     }
01038
01039 }
01040
01041 VPUBLIC void printMGPARAM(MGparam *mgparam, double realCenter[3]) {
01042     switch (mgparam->chgm) {
01043         case 0:
01044             Vnm_tprint(1, " Using linear spline charge discretization.\n");
01045             break;
01046         case 1:
01047             Vnm_tprint(1, " Using cubic spline charge discretization.\n");
01048             break;
01049         default:
01050             break;
01051     }
01052     if (mgparam->type == MCT_PARALLEL) {
01053         Vnm_tprint(1, " Partition overlap fraction = %g\n",
01054             mgparam->ofrac);
01055         Vnm_tprint(1, " Processor array = %d x %d x %d\n",
01056             mgparam->pdime[0], mgparam->pdime[1], mgparam->pdime[2]);
01057     }
01058     Vnm_tprint(1, " Grid dimensions: %d x %d x %d\n",
01059         mgparam->dime[0], mgparam->dime[1], mgparam->dime[2]);
01060     Vnm_tprint(1, " Grid spacings: %4.3f x %4.3f x %4.3f\n",
01061         mgparam->grid[0], mgparam->grid[1], mgparam->grid[2]);
01062     Vnm_tprint(1, " Grid lengths: %4.3f x %4.3f x %4.3f\n",
01063         mgparam->glen[0], mgparam->glen[1], mgparam->glen[2]);
01064     Vnm_tprint(1, " Grid center: (%4.3f, %4.3f, %4.3f)\n",
01065         realCenter[0], realCenter[1], realCenter[2]);
01066     Vnm_tprint(1, " Multigrid levels: %d\n", mgparam->nlev);
01067 }
01068
01069 }
01070
01071 VPUBLIC int initMG(int icalc,
01072     NOSH *nosh, MGparam *mgparam,
01073     PBEparam *pbeparm,
01074     double realCenter[3],
01075     Vpbe *pbe[NOSH_MAXCALC],
01076     Valist *alist[NOSH_MAXMOL],
01077     Vgrid *dielXMap[NOSH_MAXMOL],
01078     Vgrid *dielYMap[NOSH_MAXMOL],
01079     Vgrid *dielZMap[NOSH_MAXMOL],
01080     Vgrid *kappaMap[NOSH_MAXMOL],
01081     Vgrid *chargeMap[NOSH_MAXMOL],
01082     Vpmgp *pmgp[NOSH_MAXCALC],
01083     Vpmg *pmg[NOSH_MAXCALC],
01084     Vgrid *potMap[NOSH_MAXMOL]
01085 ) {
01086     int j,
01087         focusFlag,
01088         iatom;
01089     size_t bytesTotal,
01090         highWater;
01091     double sparm,
01092         iparm,
01093         q;
01094     Vatom *atom = VNULL;
01095     Vgrid *theDielXMap = VNULL,
01096         *theDielYMap = VNULL,
01097         *theDielZMap = VNULL;
01098     Vgrid *theKappaMap = VNULL,
01099         *thePotMap = VNULL,
01100         *theChargeMap = VNULL;
01101     Valist *myalist = VNULL;
01102
01103     Vnm_tstart(APBS_TIMER_SETUP, "Setup timer");
01104
01105     /* Update the grid center */
01106     for (j=0; j<3; j++) realCenter[j] = mgparam->center[j];
01107
01108     /* Check for completely-neutral molecule */
01109     q = 0;
01110     myalist = alist[pbeparm->molid-1];
01111     for (iatom=0; iatom<Valist_getNumberAtoms(myalist); iatom++) {
01112         atom = Valist_getAtom(myalist, iatom);
01113         q += VSQR(Vatom_getCharge(atom));
01114     }
01115     /* D. Gohara 10/22/09 - disabled

```

```

01120     if (q < (1e-6)) {
01121         Vnm_tprint(2, "Molecule #%d is uncharged!\n", pbeparm->molid);
01122         Vnm_tprint(2, "Sum square charge = %g!\n", q);
01123         return 0;
01124     }
01125     */
01126
01127     /* Set up PBE object */
01128     Vnm_tprint(0, "Setting up PBE object...\n");
01129     if (pbeparm->srfm == VSM_SPLINE) {
01130         sparm = pbeparm->swin;
01131     } else {
01132         sparm = pbeparm->sradi;
01133     }
01134     if (pbeparm->nion > 0) {
01135         iparm = pbeparm->ionr[0];
01136     } else {
01137         iparm = 0.0;
01138     }
01139     if (pbeparm->bcfl == BCFL_FOCUS) {
01140         if (icalc == 0) {
01141             Vnm_tprint(2, "Can't focus first calculation!\n");
01142             return 0;
01143         }
01144         focusFlag = 1;
01145     } else {
01146         focusFlag = 0;
01147     }
01148
01149     // Construct Vpbe object
01150     pbe[icalc] = Vpbe_ctor(myalist, pbeparm->nion,
01151                          pbeparm->ionc, pbeparm->ionr, pbeparm->ionq,
01152                          pbeparm->temp, pbeparm->pdie,
01153                          pbeparm->sdie, sparm, focusFlag, pbeparm->sdens,
01154                          pbeparm->zmemb, pbeparm->lmemb, pbeparm->mdie,
01155                          pbeparm->memv);
01156
01157     /* Set up PDE object */
01158     Vnm_tprint(0, "Setting up PDE object...\n");
01159     switch (pbeparm->pbetype) {
01160     case PBE_NPBE:
01161         /* TEMPORARY USEAQUA */
01162         mgparm->nonlntype = NONLIN_NPBE;
01163         mgparm->method = (mgparm->useAqua == 1) ? VSOL_NewtonAqua : VSOL_Newton;
01164         pmgp[icalc] = Vpmgp_ctor(mgparm);
01165         break;
01166     case PBE_LPBE:
01167         /* TEMPORARY USEAQUA */
01168         mgparm->nonlntype = NONLIN_LPBE;
01169         mgparm->method = (mgparm->useAqua == 1) ? VSOL_CGMGAqua : VSOL_MG;
01170         pmgp[icalc] = Vpmgp_ctor(mgparm);
01171         break;
01172     case PBE_LRPBE:
01173         Vnm_tprint(2, "Sorry, LRPBE isn't supported with the MG solver!\n");
01174         return 0;
01175     case PBE_NRPBE:
01176         Vnm_tprint(2, "Sorry, NRPBE isn't supported with the MG solver!\n");
01177         return 0;
01178     case PBE_SMPBE: /* SMPBE Added */
01179         mgparm->nonlntype = NONLIN_SMPBE;
01180         pmgp[icalc] = Vpmgp_ctor(mgparm);
01181
01182         /* Copy Code */
01183         pbe[icalc]->smsize = pbeparm->smsize;
01184         pbe[icalc]->smvolume = pbeparm->smvolume;
01185         pbe[icalc]->ipkey = pmgp[icalc]->ipkey;
01186
01187         break;
01188     default:
01189         Vnm_tprint(2, "Error! Unknown PBE type (%d)!\n", pbeparm->pbetype);
01190         return 0;
01191     }
01192     Vnm_tprint(0, "Setting PDE center to local center...\n");
01193     pmgp[icalc]->bcfl = pbeparm->bcfl;
01194     pmgp[icalc]->xcent = realCenter[0];
01195     pmgp[icalc]->ycent = realCenter[1];
01196     pmgp[icalc]->zcent = realCenter[2];
01197
01198     if (pbeparm->bcfl == BCFL_FOCUS) {
01199         if (icalc == 0) {
01200             Vnm_tprint(2, "Can't focus first calculation!\n");

```

```

01201         return 0;
01202     }
01203     /* Focusing requires the previous calculation in order to setup the
01204     current run... */
01205     pmg[icalc] = Vpmg_ctor(pmgp[icalc], pbe[icalc], 1, pmg[icalc-1],
01206                          mgparm, pbeparm->calcenergy);
01207     /* ...however, it should be done with the previous calculation now, so
01208     we should be able to destroy it here. */
01209     /* Vpmg_dtor(&(pmg[icalc-1])); */
01210 } else {
01211     if (icalc>0) Vpmg_dtor(&(pmg[icalc-1]));
01212     pmg[icalc] = Vpmg_ctor(pmgp[icalc], pbe[icalc], 0, VNULL, mgparm,
PCE_NO);
01213 }
01214 if (icalc>0) {
01215     Vpmgp_dtor(&(pmgp[icalc-1]));
01216     Vpbe_dtor(&(pbe[icalc-1]));
01217 }
01218 if (pbeparm->useDielMap) {
01219     if ((pbeparm->dielMapID-1) < nosh->ndiel) {
01220         theDielXMap = dielXMap[pbeparm->dielMapID-1];
01221     } else {
01222         Vnm_print(2, "Error! %d is not a valid dielectric map ID!\n",
01223                 pbeparm->dielMapID);
01224         return 0;
01225     }
01226 }
01227 if (pbeparm->useDielMap) {
01228     if ((pbeparm->dielMapID-1) < nosh->ndiel) {
01229         theDielYMap = dielYMap[pbeparm->dielMapID-1];
01230     } else {
01231         Vnm_print(2, "Error! %d is not a valid dielectric map ID!\n",
01232                 pbeparm->dielMapID);
01233         return 0;
01234     }
01235 }
01236 if (pbeparm->useDielMap) {
01237     if ((pbeparm->dielMapID-1) < nosh->ndiel) {
01238         theDielZMap = dielZMap[pbeparm->dielMapID-1];
01239     } else {
01240         Vnm_print(2, "Error! %d is not a valid dielectric map ID!\n",
01241                 pbeparm->dielMapID);
01242         return 0;
01243     }
01244 }
01245 if (pbeparm->useKappaMap) {
01246     if ((pbeparm->kappaMapID-1) < nosh->nkappa) {
01247         theKappaMap = kappaMap[pbeparm->kappaMapID-1];
01248     } else {
01249         Vnm_print(2, "Error! %d is not a valid kappa map ID!\n",
01250                 pbeparm->kappaMapID);
01251         return 0;
01252     }
01253 }
01254 if (pbeparm->usePotMap) {
01255     if ((pbeparm->potMapID-1) < nosh->npot) {
01256         thePotMap = potMap[pbeparm->potMapID-1];
01257     } else {
01258         Vnm_print(2, "Error! %d is not a valid potential map ID!\n",
01259                 pbeparm->potMapID);
01260         return 0;
01261     }
01262 }
01263 if (pbeparm->useChargeMap) {
01264     if ((pbeparm->chargeMapID-1) < nosh->ncharge) {
01265         theChargeMap = chargeMap[pbeparm->chargeMapID-1];
01266     } else {
01267         Vnm_print(2, "Error! %d is not a valid charge map ID!\n",
01268                 pbeparm->chargeMapID);
01269         return 0;
01270     }
01271 }
01272
01273 if (pbeparm->bconf == BCFL_MAP && thePotMap == VNULL) {
01274     Vnm_print(2, "Warning: You specified 'bcfl map' in the input file, but no potential map was found.
\n");
01275     Vnm_print(2, "        You must specify 'usemap pot' statement in the APBS input file!\n");
01276     Vnm_print(2, "Bailing out ...!\n");
01277     return 0;
01278 }
01279

```

```

01280 // Initialize calculation coefficients
01281 if (!Vpmg_fillco(pmg[icalc],
01282                 pbeparm->srfm, pbeparm->swin, mgparm->chgm,
01283                 pbeparm->useDielMap, theDielXMap,
01284                 pbeparm->useDielMap, theDielYMap,
01285                 pbeparm->useDielMap, theDielZMap,
01286                 pbeparm->useKappaMap, theKappaMap,
01287                 pbeparm->usePotMap, thePotMap,
01288                 pbeparm->useChargeMap, theChargeMap)) {
01289     Vnm_print(2, "initMG: problems setting up coefficients (fillco)!\n");
01290     return 0;
01291 }
01292
01293 /* Print a few derived parameters */
01294 #ifndef VAPBSQUIET
01295 Vnm_tprint(1, " Debye length:  %g A\n", Vpbe_getDeblen(pbe[icalc]));
01296 #endif
01297
01298 /* Setup time statistics */
01299 Vnm_tstop(APBS_TIMER_SETUP, "Setup timer");
01300
01301 /* Memory statistics */
01302 bytesTotal = Vmem_bytesTotal();
01303 highWater = Vmem_highWaterTotal();
01304
01305 #ifndef VAPBSQUIET
01306 Vnm_tprint( 1, " Current memory usage:  %4.3f MB total, \
01307 %4.3f MB high water\n", (double)(bytesTotal)/(1024.*1024.),
01308                         (double)(highWater)/(1024.*1024.));
01309 #endif
01310
01311 return 1;
01312 }
01313 }
01314
01315 VPUBLIC void killMG(NOsh *nosh, Vpbe *pbe[NOSH_MAXCALC],
01316                  Vpmgp *pmgp[NOSH_MAXCALC], Vpmg *pmg[NOSH_MAXCALC]) {
01317     int i;
01318
01319 #ifndef VAPBSQUIET
01320     Vnm_tprint(1, "Destroying multigrid structures.\n");
01321 #endif
01322
01323 /*
01324  * There appears to be a relationship (or this is a bug in Linux, can't tell
01325  * at the moment, since Linux is the only OS that seems to be affected)
01326  * between one of the three object types: Vpbe, Vpmg or Vpmgp that requires
01327  * deallocations to be performed in a specific order. This results in a
01328  * bug some of the time when freeing Vpmg objects below. Therefore it
01329  * appears to be important to release the Vpmg structs BEFORE the Vpmgp structs .
01330  */
01331 Vpmg_dtor(&(pmg[nosh->ncalc-1]));
01332
01333 for(i=0;i<nosh->ncalc;i++){
01334     Vpbe_dtor(&(pbe[i]));
01335     Vpmgp_dtor(&(pmgp[i]));
01336 }
01337 }
01338 }
01339 }
01340
01341 VPUBLIC int solveMG(NOsh *nosh,
01342                  Vpmg *pmg,
01343                  MGparm_CalcType type
01344                  ) {
01345     int nx,
01346         ny,
01347         nz,
01348         i;
01349
01350     if (nosh != VNULL) {
01351         if (nosh->bogus) return 1;
01352     }
01353
01354     Vnm_tstart(APBS_TIMER_SOLVER, "Solver timer");
01355
01356     if (type != MCT_DUMMY) {
01357         if (!Vpmg_solve(pmg)) {
01358             Vnm_print(2, " Error during PDE solution!\n");

```

```

01361         return 0;
01362     }
01363 } else {
01364     Vnm_tprint( 1, " Skipping solve for mg-dummy run; zeroing \
01365 solution array\n");
01366     nx = pmg->pmgp->nx;
01367     ny = pmg->pmgp->ny;
01368     nz = pmg->pmgp->nz;
01369     for (i=0; i<nx*ny*nz; i++) pmg->u[i] = 0.0;
01370 }
01371 Vnm_tstop(APBS_TIMER_SOLVER, "Solver timer");
01372
01373 return 1;
01374 }
01375 }
01376
01377 VPUBLIC int setPartMG(Nosh *nosh,
01378                     MGparm *mgparm,
01379                     Vpmg *pmg
01380                     ) {
01381
01382     int j;
01383     double partMin[3],
01384            partMax[3];
01385
01386     if (nosh->bogus) return 1;
01387
01388     if (mgparm->type == MCT_PARALLEL) {
01389         for (j=0; j<3; j++) {
01390             partMin[j] = mgparm->partDisjCenter[j] - 0.5*mgparm->
01391 partDisjLength[j];
01392             partMax[j] = mgparm->partDisjCenter[j] + 0.5*mgparm->
01393 partDisjLength[j];
01394         }
01395         Vnm_tprint(1, "setPartMG (%s, %d): Disj part center = (%g, %g, %g)\n",
01396                     __FILE__, __LINE__,
01397                     mgparm->partDisjCenter[0],
01398                     mgparm->partDisjCenter[1],
01399                     mgparm->partDisjCenter[2]
01400                     );
01401         Vnm_tprint(1, "setPartMG (%s, %d): Disj part lower corner = (%g, %g, %g)\n",
01402                     __FILE__, __LINE__, partMin[0], partMin[1], partMin[2]);
01403         Vnm_tprint(1, "setPartMG (%s, %d): Disj part upper corner = (%g, %g, %g)\n",
01404                     __FILE__, __LINE__,
01405                     partMax[0], partMax[1], partMax[2]);
01406     } else {
01407         for (j=0; j<3; j++) {
01408             partMin[j] = mgparm->center[j] - 0.5*mgparm->glen[j];
01409             partMax[j] = mgparm->center[j] + 0.5*mgparm->glen[j];
01410         }
01411     }
01412     /* Vnm_print(1, "DEBUG (%s, %d): setPartMG calling setPart with upper corner \
01413 %g %g %g and lower corner %g %g %g\n", __FILE__, __LINE__,
01414                 partMin[0], partMin[1], partMin[2],
01415                 partMax[0], partMax[1], partMax[2]); */
01416     Vpmg_setPart(pmg, partMin, partMax, mgparm->partDisjOwnSide);
01417
01418     return 1;
01419 }
01420 }
01421 }
01422
01423 VPUBLIC int energyMG(Nosh *nosh,
01424                    int icalc,
01425                    Vpmg *pmg,
01426                    int *nenergy,
01427                    double *totEnergy,
01428                    double *qfEnergy,
01429                    double *qmEnergy,
01430                    double *dielEnergy
01431                    ) {
01432
01433     Valist *alist;
01434     Vatom *atom;
01435     int i,
01436         extEnergy;
01437     double tenergy;
01438     MGparm *mgparm;
01439     PBEParm *pbeparm;

```



```

01440
01441     mgparm = nosh->calc[icalc]->mgparm;
01442     pbeparm = nosh->calc[icalc]->pbeparm;
01443
01444     Vnm_tstart(APBS_TIMER_ENERGY, "Energy timer");
01445     extEnergy = 1;
01446
01447     if (pbeparm->calcenergy == PCE_TOTAL) {
01448         *nenergy = 1;
01449         /* Some processors don't count */
01450         if (nosh->bogus == 0) {
01451             *totEnergy = Vpmg_energy(pmg, extEnergy);
01452 #ifndef VAPBSQUIET
01453             Vnm_tprint( 1, " Total electrostatic energy = %1.12E kJ/mol\n",
01454                 Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*( *totEnergy));
01455 #endif
01456         } else *totEnergy = 0;
01457     } else if (pbeparm->calcenergy == PCE_COMPS) {
01458         *nenergy = 1;
01459         *totEnergy = Vpmg_energy(pmg, extEnergy);
01460         *qfEnergy = Vpmg_qfEnergy(pmg, extEnergy);
01461         *qmEnergy = Vpmg_qmEnergy(pmg, extEnergy);
01462         *dielEnergy = Vpmg_dielEnergy(pmg, extEnergy);
01463 #ifndef VAPBSQUIET
01464         Vnm_tprint( 1, " Total electrostatic energy = %1.12E \
01465 kJ/mol\n", Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*( *totEnergy));
01466         Vnm_tprint( 1, " Fixed charge energy = %g kJ/mol\n",
01467             0.5*Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*( *qfEnergy));
01468         Vnm_tprint( 1, " Mobile charge energy = %g kJ/mol\n",
01469             Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*( *qmEnergy));
01470         Vnm_tprint( 1, " Dielectric energy = %g kJ/mol\n",
01471             Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*( *dielEnergy));
01472         Vnm_tprint( 1, " Per-atom energies:\n");
01473 #endif
01474         alist = pmg->pbe->alist;
01475         for (i=0; i<Valist_getNumberAtoms(alist); i++) {
01476             atom = Valist_getAtom(alist, i);
01477             tenergy = Vpmg_qfAtomEnergy(pmg, atom);
01478 #ifndef VAPBSQUIET
01479             Vnm_tprint( 1, " Atom %d: %1.12E kJ/mol\n", i,
01480                 0.5*Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*tenergy);
01481 #endif
01482         }
01483     } else *nenergy = 0;
01484
01485     Vnm_tstop(APBS_TIMER_ENERGY, "Energy timer");
01486
01487     return 1;
01488 }
01489
01490 VPUBLIC int forceMG(Vmem *mem,
01491     NOSH *nosh,
01492     PBEParm *pbeparm,
01493     MGparm *mgparm,
01494     Vpmg *pmg,
01495     int *nforce,
01496     AtomForce **atomForce,
01497     Valist *alist[NOSH_MAXMOL]
01498 ) {
01499
01500     int j,
01501         k;
01502     double qfForce[3],
01503         dbForce[3],
01504         ibForce[3];
01505
01506     Vnm_tstart(APBS_TIMER_FORCE, "Force timer");
01507
01508 #ifndef VAPBSQUIET
01509     Vnm_tprint( 1, " Calculating forces...\n");
01510 #endif
01511
01512     if (pbeparm->calcforce == PCF_TOTAL) {
01513         *nforce = 1;
01514         *atomForce = (AtomForce *)Vmem_malloc(mem, 1, sizeof(AtomForce));
01515         /* Clear out force arrays */
01516         for (j=0; j<3; j++) {
01517             (*atomForce)[0].qfForce[j] = 0;
01518             (*atomForce)[0].ibForce[j] = 0;
01519             (*atomForce)[0].dbForce[j] = 0;
01520         }

```

```

01521         for (j=0;j<Valist_getNumberAtoms(alist[pbeparm->
molid-1]);j++) {
01522             if (nosh->bogus == 0) {
01523                 VASSERT(Vpmg_qfForce(pmg, qfForce, j, mgparm->chgm));
01524                 VASSERT(Vpmg_ibForce(pmg, ibForce, j, pbeparm->srfm));
01525                 VASSERT(Vpmg_dbForce(pmg, dbForce, j, pbeparm->srfm));
01526             } else {
01527                 for (k=0; k<3; k++) {
01528                     qfForce[k] = 0;
01529                     ibForce[k] = 0;
01530                     dbForce[k] = 0;
01531                 }
01532             }
01533             for (k=0; k<3; k++) {
01534                 (*atomForce)[0].qfForce[k] += qfForce[k];
01535                 (*atomForce)[0].ibForce[k] += ibForce[k];
01536                 (*atomForce)[0].dbForce[k] += dbForce[k];
01537             }
01538         }
01539 #ifndef VAPBSQUIET
01540         Vnm_tprint( 1, " Printing net forces for molecule %d (kJ/mol/A)\n",
01541                     pbeparm->molid);
01542         Vnm_tprint( 1, " Legend:\n");
01543         Vnm_tprint( 1, "   qf -- fixed charge force\n");
01544         Vnm_tprint( 1, "   db -- dielectric boundary force\n");
01545         Vnm_tprint( 1, "   ib -- ionic boundary force\n");
01546         Vnm_tprint( 1, "   qf %4.3e %4.3e %4.3e\n",
01547                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(atomForce)[0].qfForce[0],
01548                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(atomForce)[0].qfForce[1],
01549                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(atomForce)[0].qfForce[2]);
01550         Vnm_tprint( 1, "   ib %4.3e %4.3e %4.3e\n",
01551                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(atomForce)[0].ibForce[0],
01552                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(atomForce)[0].ibForce[1],
01553                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(atomForce)[0].ibForce[2]);
01554         Vnm_tprint( 1, "   db %4.3e %4.3e %4.3e\n",
01555                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(atomForce)[0].dbForce[0],
01556                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(atomForce)[0].dbForce[1],
01557                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(atomForce)[0].dbForce[2]);
01558 #endif
01559     } else if (pbeparm->calcforce == PCF_COMPS) {
01560         *nforce = Valist_getNumberAtoms(alist[pbeparm->
molid-1]);
01561         *atomForce = (AtomForce *)Vmem_malloc(mem, *nforce,
01562                                                 sizeof(AtomForce));
01563 #ifndef VAPBSQUIET
01564         Vnm_tprint( 1, " Printing per-atom forces for molecule %d (kJ/mol/A)\n",
01565                     pbeparm->molid);
01566         Vnm_tprint( 1, " Legend:\n");
01567         Vnm_tprint( 1, "   tot n -- total force for atom n\n");
01568         Vnm_tprint( 1, "   qf n -- fixed charge force for atom n\n");
01569         Vnm_tprint( 1, "   db n -- dielectric boundary force for atom n\n");
01570         Vnm_tprint( 1, "   ib n -- ionic boundary force for atom n\n");
01571 #endif
01572         for (j=0;j<Valist_getNumberAtoms(alist[pbeparm->
molid-1]);j++) {
01573             if (nosh->bogus == 0) {
01574                 VASSERT(Vpmg_qfForce(pmg, (*atomForce)[j].qfForce, j,
01575                                     mgparm->chgm));
01576                 VASSERT(Vpmg_ibForce(pmg, (*atomForce)[j].ibForce, j,
01577                                     pbeparm->srfm));
01578                 VASSERT(Vpmg_dbForce(pmg, (*atomForce)[j].dbForce, j,
01579                                     pbeparm->srfm));
01580             } else {
01581                 for (k=0; k<3; k++) {
01582                     (*atomForce)[j].qfForce[k] = 0;
01583                     (*atomForce)[j].ibForce[k] = 0;
01584                     (*atomForce)[j].dbForce[k] = 0;
01585                 }
01586             }
01587 #ifndef VAPBSQUIET
01588             Vnm_tprint( 1, "mgF tot %d %4.3e %4.3e %4.3e\n", j,
01589                         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01590                         *((atomForce)[j].qfForce[0]+(atomForce)[j].ibForce[0]+
01591                         (atomForce)[j].dbForce[0]),
01592                         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01593                         *((atomForce)[j].qfForce[1]+(atomForce)[j].ibForce[1]+
01594                         (atomForce)[j].dbForce[1]),
01595                         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01596                         *((atomForce)[j].qfForce[2]+(atomForce)[j].ibForce[2]+
01597                         (atomForce)[j].dbForce[2]));
01598             Vnm_tprint( 1, "mgF qf %d %4.3e %4.3e %4.3e\n", j,

```

```

01599         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01600         *(*atomForce)[j].qfForce[0],
01601         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01602         *(*atomForce)[j].qfForce[1],
01603         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01604         *(*atomForce)[j].qfForce[2]);
01605     Vnm_tprint(1, "mgF ib %d %4.3e %4.3e %4.3e\n", j,
01606         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01607         *(*atomForce)[j].ibForce[0],
01608         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01609         *(*atomForce)[j].ibForce[1],
01610         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01611         *(*atomForce)[j].ibForce[2]);
01612     Vnm_tprint(1, "mgF db %d %4.3e %4.3e %4.3e\n", j,
01613         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01614         *(*atomForce)[j].dbForce[0],
01615         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01616         *(*atomForce)[j].dbForce[1],
01617         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01618         *(*atomForce)[j].dbForce[2]);
01619 #endif
01620     }
01621     } else *nforce = 0;
01622
01623     Vnm_tstop(APBS_TIMER_FORCE, "Force timer");
01624
01625     return 1;
01626 }
01627
01628 VPUBLIC void killEnergy() {
01629
01630 #ifndef VAPBSQUIET
01631     Vnm_tprint(1, "No energy arrays to destroy.\n");
01632 #endif
01633 }
01634 }
01635
01636 VPUBLIC void killForce(Vmem *mem, NOsh *nosh, int nforce[
    NOSH_MAXCALC],
01637         AtomForce *atomForce[NOSH_MAXCALC]) {
01638
01639     int i;
01640
01641 #ifndef VAPBSQUIET
01642     Vnm_tprint(1, "Destroying force arrays.\n");
01643 #endif
01644
01645     for (i=0; i<nosh->ncalc; i++) {
01646
01647         if (nforce[i] > 0) Vmem_free(mem, nforce[i], sizeof(AtomForce),
01648             (void *)&(atomForce[i]));
01649
01650     }
01651 }
01652
01653 VPUBLIC int writematMG(int rank, NOsh *nosh, PBEparm *pbeparm,
    Vpmg *pmg) {
01654
01655     char writematstem[VMAX_ARGLEN];
01656     char outpath[VMAX_ARGLEN];
01657     char mxttype[3];
01658     int strlenmax;
01659
01660     if (nosh->bogus) return 1;
01661
01662 #ifdef HAVE_MPI_H
01663     strlenmax = VMAX_ARGLEN-14;
01664     if (strlen(pbeparm->writematstem) > strlenmax) {
01665         Vnm_tprint(2, " Matrix name (%s) too long (%d char max)!\n",
01666             pbeparm->writematstem, strlenmax);
01667         Vnm_tprint(2, " Not writing matrix!\n");
01668         return 0;
01669     }
01670     sprintf(writematstem, "%s-PE%d", pbeparm->writematstem, rank);
01671 #else
01672     strlenmax = (int)(VMAX_ARGLEN)-1;
01673     if ((int)strlen(pbeparm->writematstem) > strlenmax) {
01674         Vnm_tprint(2, " Matrix name (%s) too long (%d char max)!\n",
01675             pbeparm->writematstem, strlenmax);
01676         Vnm_tprint(2, " Not writing matrix!\n");
01677         return 0;

```

```

01678     }
01679     if(nosh->ispara == 1){
01680         sprintf(writematstem, "%s-PE%d", pbeparm->writematstem,nosh->
proc_rank);
01681     }else{
01682         sprintf(writematstem, "%s", pbeparm->writematstem);
01683     }
01684 #endif
01685
01686     if (pbeparm->writemat == 1) {
01687         strlenmax = VMAX_ARGLEN-5;
01688         if ((int)strlen(pbeparm->writematstem) > strlenmax) {
01689             Vnm_tprint(2, " Matrix name (%s) too long (%d char max)!\n",
pbeparm->writematstem, strlenmax);
01690             Vnm_tprint(2, " Not writing matrix!\n");
01691             return 0;
01692         }
01693         sprintf(outpath, "%s.%s", writematstem, "mat");
01694         mxtype[0] = 'R';
01695         mxtype[1] = 'S';
01696         mxtype[2] = 'A';
01697         /* Poisson operator only */
01698         if (pbeparm->writematflag == 0) {
01699             Vnm_tprint( 1, " Writing Poisson operator matrix \
01700 to %s...\n", outpath);
01701
01702             /* Linearization of Poisson-Boltzmann operator around solution */
01703         } else if (pbeparm->writematflag == 1) {
01704             Vnm_tprint( 1, " Writing linearization of full \
01705 Poisson-Boltzmann operator matrix to %s...\n", outpath);
01706
01707         } else {
01708             Vnm_tprint( 2, " Bogus matrix specification\
01709 (%d)!\n", pbeparm->writematflag);
01710             return 0;
01711         }
01712
01713         Vnm_tprint(0, " Printing operator...\n");
01714         //Vpmg_printColComp(pmg, outpath, outpath, mxtype,
01715         // pbeparm->writematflag);
01716         return 0;
01717     }
01718
01719     return 1;
01720 }
01721
01722 }
01723
01724 VPUBLIC void storeAtomEnergy(Vpmg *pmg, int icalc, double **atomEnergy,
int *nenergy){
01725
01726     Vatom *atom;
01727     Valist *alist;
01728     int i;
01729
01730     alist = pmg->pbe->alist;
01731     *nenergy = Valist_getNumberAtoms(alist);
01732     *atomEnergy = (double *)Vmem_malloc(pmg->vmem, *nenergy, sizeof(double));
01733
01734     for (i=0; i<*nenergy; i++) {
01735         atom = Valist_getAtom(alist, i);
01736         (*atomEnergy)[i] = Vpmg_qfAtomEnergy(pmg, atom);
01737     }
01738 }
01739 }
01740
01741 VPUBLIC int writedataFlat(
01742     NOsh *nosh,
01743     Vcom *com,
01744     const char *fname,
01745     double totEnergy[NOSH_MAXCALC],
01746     double qfEnergy[NOSH_MAXCALC],
01747     double qmEnergy[NOSH_MAXCALC],
01748     double dielEnergy[NOSH_MAXCALC],
01749     int nenergy[NOSH_MAXCALC],
01750     double *atomEnergy[NOSH_MAXCALC],
01751     int nforce[NOSH_MAXCALC],
01752     AtomForce *atomForce[NOSH_MAXCALC]) {
01753
01754     FILE *file;
01755     time_t now;
01756     int ielec, icalc, i, j;
01757     char *timestring = VNULL;

```

```
01758     PBeparm *pbeparm = VNULL;
01759     MGparm *mgparm = VNULL;
01760     double conversion, ltenergy, gtenergy, scalar;
01761
01762     if (nosh->bogus) return 1;
01763
01764     /* Initialize some variables */
01765
01766     icalc = 0;
01767
01768     file = fopen(fname, "w");
01769     if (file == VNULL) {
01770         Vnm_print(2, "writedataFlat: Problem opening virtual socket %s\n",
01771             fname);
01772         return 0;
01773     }
01774
01775     /* Strip the newline character from the date */
01776
01777     now = time(VNULL);
01778     timestring = ctime(&now);
01779     fprintf(file, "%s\n", timestring);
01780
01781     for (ielec=0; ielec<nosh->nelec; ielec++) { /* elec loop */
01782
01783         /* Initialize per-elec pointers */
01784
01785         mgparm = nosh->calc[icalc]->mgparm;
01786         pbeparm = nosh->calc[icalc]->pbeparm;
01787
01788         /* Convert from kT/e to kJ/mol */
01789         conversion = Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na;
01790
01791         fprintf(file, "elec");
01792         if (Vstring_strcasecmp(nosh->elecname[ielec], "") != 0) {
01793             fprintf(file, " name %s\n", nosh->elecname[ielec]);
01794         } else fprintf(file, "\n");
01795
01796         switch (mgparm->type) {
01797             case MCT_DUMMY:
01798                 fprintf(file, "    mg-dummy\n");
01799                 break;
01800             case MCT_MANUAL:
01801                 fprintf(file, "    mg-manual\n");
01802                 break;
01803             case MCT_AUTO:
01804                 fprintf(file, "    mg-auto\n");
01805                 break;
01806             case MCT_PARALLEL:
01807                 fprintf(file, "    mg-para\n");
01808                 break;
01809             default:
01810                 break;
01811         }
01812
01813         fprintf(file, "    mol %d\n", pbeparm->molid);
01814         fprintf(file, "    dime %d %d %d\n", mgparm->dime[0], mgparm->dime[1],\
01815             mgparm->dime[2]);
01816
01817         switch (pbeparm->pbetype) {
01818             case PBE_NPBE:
01819                 fprintf(file, "    npbe\n");
01820                 break;
01821             case PBE_LPBE:
01822                 fprintf(file, "    lpbe\n");
01823                 break;
01824             default:
01825                 break;
01826         }
01827
01828         if (pbeparm->nion > 0) {
01829             for (i=0; i<pbeparm->nion; i++) {
01830                 fprintf(file, "    ion %4.3f %4.3f %4.3f\n",
01831                     pbeparm->ionr[i], pbeparm->ionq[i], pbeparm->
01832                     ionc[i]);
01833             }
01834         }
01835         fprintf(file, "    pdie %4.3f\n", pbeparm->pdie);
01836         fprintf(file, "    sdie %4.3f\n", pbeparm->sdie);
01837     }
```

```

01838     switch (pbeparm->srfm) {
01839     case 0:
01840         fprintf(file, "    srfm mol\n");
01841         fprintf(file, "    srad %4.3f\n", pbeparm->srad);
01842         break;
01843     case 1:
01844         fprintf(file, "    srfm smol\n");
01845         fprintf(file, "    srad %4.3f\n", pbeparm->srad);
01846         break;
01847     case 2:
01848         fprintf(file, "    srfm spl2\n");
01849         fprintf(file, "    srad %4.3f\n", pbeparm->srad);
01850         break;
01851     default:
01852         break;
01853     }
01854
01855     switch (pbeparm->bcbf) {
01856     case BCFL_ZERO:
01857         fprintf(file, "    bcbf zero\n");
01858         break;
01859     case BCFL_SDH:
01860         fprintf(file, "    bcbf sdh\n");
01861         break;
01862     case BCFL_MDH:
01863         fprintf(file, "    bcbf mdh\n");
01864         break;
01865     case BCFL_FOCUS:
01866         fprintf(file, "    bcbf focus\n");
01867         break;
01868     case BCFL_MAP:
01869         fprintf(file, "    bcbf map\n");
01870         break;
01871     case BCFL_MEM:
01872         fprintf(file, "    bcbf mem\n");
01873         break;
01874     default:
01875         break;
01876     }
01877
01878     fprintf(file, "    temp %4.3f\n", pbeparm->temp);
01879
01880     for (; icalc<=nosh->elec2calc[ielec]; icalc++){ /* calc loop */
01881
01882         /* Reinitialize per-calc pointers */
01883         mgparm = nosh->calc[icalc]->mgparm;
01884         pbeparm = nosh->calc[icalc]->pbeparm;
01885
01886         fprintf(file, "    calc\n");
01887         fprintf(file, "        id %i\n", (icalc+1));
01888         fprintf(file, "        grid %4.3f %4.3f %4.3f\n",
01889             mgparm->grid[0], mgparm->grid[1], mgparm->grid[2]);
01890         fprintf(file, "        glen %4.3f %4.3f %4.3f\n",
01891             mgparm->glen[0], mgparm->glen[1], mgparm->glen[2]);
01892
01893         if (pbeparm->calcenergy == PCE_TOTAL) {
01894             fprintf(file, "            totEnergy %1.12E kJ/mol\n",
01895                 (totEnergy[icalc]*conversion));
01896         } if (pbeparm->calcenergy == PCE_COMPS) {
01897             fprintf(file, "            totEnergy %1.12E kJ/mol\n",
01898                 (totEnergy[icalc]*conversion));
01899             fprintf(file, "            qfEnergy %1.12E kJ/mol\n",
01900                 (0.5*qfEnergy[icalc]*conversion));
01901             fprintf(file, "            qmEnergy %1.12E kJ/mol\n",
01902                 (qmEnergy[icalc]*conversion));
01903             fprintf(file, "            dielEnergy %1.12E kJ/mol\n",
01904                 (dielEnergy[icalc]*conversion));
01905             for (i=0; i<nenergy[icalc]; i++){
01906                 fprintf(file, "                atom %i %1.12E kJ/mol\n", i,
01907                     (0.5*atomEnergy[icalc][i]*conversion));
01908             }
01909         }
01910     }
01911
01912     if (pbeparm->calcforce == PCF_TOTAL) {
01913         fprintf(file, "            qfForce %1.12E %1.12E %1.12E kJ/mol/A\n",
01914             (atomForce[icalc][0].qfForce[0]*conversion),
01915             (atomForce[icalc][0].qfForce[1]*conversion),
01916             (atomForce[icalc][0].qfForce[2]*conversion));
01917         fprintf(file, "            ibForce %1.12E %1.12E %1.12E kJ/mol/A\n",
01918             (atomForce[icalc][0].ibForce[0]*conversion),

```

```

01919                (atomForce[icalc][0].ibForce[1]*conversion),
01920                (atomForce[icalc][0].ibForce[2]*conversion));
01921        fprintf(file, "          dbForce %1.12E %1.12E %1.12E kJ/mol/A\n",
01922                (atomForce[icalc][0].dbForce[0]*conversion),
01923                (atomForce[icalc][0].dbForce[1]*conversion),
01924                (atomForce[icalc][0].dbForce[2]*conversion));
01925    }
01926    fprintf(file, "      end\n");
01927  }
01928
01929    fprintf(file, "end\n");
01930  }
01931
01932  /* Handle print energy statements */
01933
01934  for (i=0; i<nosh->nprint; i++) {
01935
01936    if (nosh->printwhat[i] == NPT_ENERGY) {
01937
01938      fprintf(file, "print energy");
01939      fprintf(file, " %d", nosh->printcalc[i][0]+1);
01940
01941      for (j=1; j<nosh->printnarg[i]; j++) {
01942        if (nosh->printop[i][j-1] == 0) fprintf(file, " +");
01943        else if (nosh->printop[i][j-1] == 1) fprintf(file, " -");
01944        fprintf(file, " %d", nosh->printcalc[i][j]+1);
01945      }
01946
01947      fprintf(file, "\n");
01948      icalc = nosh->elec2calc[nosh->printcalc[i][0]];
01949
01950      ltenergy = Vunit_kb * (1e-3) * Vunit_Na * \
01951        nosh->calc[icalc]->pbeparm->temp * totEnergy[icalc];
01952
01953      for (j=1; j<nosh->printnarg[i]; j++) {
01954        icalc = nosh->elec2calc[nosh->printcalc[i][j]];
01955        /* Add or subtract? */
01956        if (nosh->printop[i][j-1] == 0) scalar = 1.0;
01957        else if (nosh->printop[i][j-1] == 1) scalar = -1.0;
01958        /* Accumulate */
01959        ltenergy += (scalar * Vunit_kb * (1e-3) * Vunit_Na *
01960          nosh->calc[icalc]->pbeparm->temp * totEnergy[icalc]);
01961
01962        Vcom_reduce(com, &ltenergy, &gtenergy, 1, 2, 0);
01963      }
01964
01965      fprintf(file, "      localEnergy %1.12E kJ/mol\n", \
01966              ltenergy);
01967      fprintf(file, "      globalEnergy %1.12E kJ/mol\nend\n", \
01968              gtenergy);
01969    }
01970  }
01971
01972  fclose(file);
01973
01974  return 1;
01975 }
01976
01977 VPUBLIC int writedataXML(Nosh *nosh, Vcom *com, const char *fname,
01978                        double totEnergy[NOSH_MAXCALC],
01979                        double qfEnergy[NOSH_MAXCALC],
01980                        double qmEnergy[NOSH_MAXCALC],
01981                        double dielEnergy[NOSH_MAXCALC],
01982                        int nenergy[NOSH_MAXCALC],
01983                        double *atomEnergy[NOSH_MAXCALC],
01984                        int nforce[NOSH_MAXCALC],
01985                        AtomForce *atomForce[NOSH_MAXCALC]) {
01986
01987  FILE *file;
01988  time_t now;
01989  int ielec, icalc, i, j;
01990  char *timestring = VNULL;
01991  char *c = VNULL;
01992  PBeparm *pbeparm = VNULL;
01993  MGparm *mgparm = VNULL;
01994  double conversion, ltenergy, gtenergy, scalar;
01995
01996  if (nosh->bogus) return 1;
01997
01998  /* Initialize some variables */
01999

```

```

02000     icalc = 0;
02001
02002     file = fopen(fname, "w");
02003     if (file == VNULL) {
02004         Vnm_print(2, "writedataXML: Problem opening virtual socket %s\n",
02005             fname);
02006         return 0;
02007     }
02008
02009     fprintf(file, "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
02010     fprintf(file, "<APBS>\n");
02011
02012     /* Strip the newline character from the date */
02013
02014     now = time(VNULL);
02015     timestring = ctime(&now);
02016     for(c = timestring; *c != '\n'; c++);
02017     *c = '\0';
02018     fprintf(file, "    <date>%s</date>\n", timestring);
02019
02020     for (ielec=0; ielec<nosh->nelec;ielec++){ /* elec loop */
02021
02022         /* Initialize per-elec pointers */
02023
02024         mgparm = nosh->calc[icalc]->mgparm;
02025         pbeparm = nosh->calc[icalc]->pbeparm;
02026
02027         /* Convert from kT/e to kJ/mol */
02028         conversion = Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na;
02029
02030         fprintf(file, "    <elec>\n");
02031         if (Vstring_strcasecmp(nosh->elecname[ielec], "") != 0) {
02032             fprintf(file, "        <name>%s</name>\n", nosh->elecname[ielec]);
02033         }
02034
02035         switch (mgparm->type) {
02036             case MCT_DUMMY:
02037                 fprintf(file, "            <type>mg-dummy</type>\n");
02038                 break;
02039             case MCT_MANUAL:
02040                 fprintf(file, "            <type>mg-manual</type>\n");
02041                 break;
02042             case MCT_AUTO:
02043                 fprintf(file, "            <type>mg-auto</type>\n");
02044                 break;
02045             case MCT_PARALLEL:
02046                 fprintf(file, "            <type>mg-para</type>\n");
02047                 break;
02048             default:
02049                 break;
02050         }
02051
02052         fprintf(file, "        <molid>%d</molid>\n", pbeparm->molid);
02053         fprintf(file, "        <nx>%d</nx>\n", mgparm->dime[0]);
02054         fprintf(file, "        <ny>%d</ny>\n", mgparm->dime[1]);
02055         fprintf(file, "        <nz>%d</nz>\n", mgparm->dime[2]);
02056
02057         switch (pbeparm->pbetype) {
02058             case PBE_NPBE:
02059                 fprintf(file, "            <pbe>npbe</pbe>\n");
02060                 break;
02061             case PBE_LPBE:
02062                 fprintf(file, "            <pbe>lpbe</pbe>\n");
02063                 break;
02064             default:
02065                 break;
02066         }
02067
02068         if (pbeparm->nion > 0) {
02069             for (i=0; i<pbeparm->nion; i++) {
02070                 fprintf(file, "                <ion>\n");
02071                 fprintf(file, "                    <radius>%4.3f A</radius>\n",
02072                     pbeparm->ionr[i]);
02073                 fprintf(file, "                    <charge>%4.3f A</charge>\n",
02074                     pbeparm->ionq[i]);
02075                 fprintf(file, "                    <concentration>%4.3f M</concentration>\n",
02076                     pbeparm->ionc[i]);
02077                 fprintf(file, "                </ion>\n");
02078             }
02079         }
02080     }

```



```

02081
02082     fprintf(file,"          <pdie>%4.3f</pdie>\n", pbeparm->pdie);
02083     fprintf(file,"          <sdie>%4.3f</sdie>\n", pbeparm->sdie);
02084
02085     switch (pbeparm->srfm) {
02086     case 0:
02087         fprintf(file,"          <srfm>mol</srfm>\n");
02088         fprintf(file,"          <srاد>%4.3f</srاد>\n", pbeparm->srاد);
02089         break;
02090     case 1:
02091         fprintf(file,"          <srfm>smol</srfm>\n");
02092         fprintf(file,"          <srاد>%4.3f</srاد>\n", pbeparm->srاد);
02093         break;
02094     case 2:
02095         fprintf(file,"          <srfm>spl2</srfm>\n");
02096         break;
02097     default:
02098         break;
02099     }
02100
02101     switch (pbeparm->bcfl) {
02102     case BCFL_ZERO:
02103         fprintf(file,"          <bcfl>zero</bcfl>\n");
02104         break;
02105     case BCFL_SDH:
02106         fprintf(file,"          <bcfl>sdh</bcfl>\n");
02107         break;
02108     case BCFL_MDH:
02109         fprintf(file,"          <bcfl>mdh</bcfl>\n");
02110         break;
02111     case BCFL_FOCUS:
02112         fprintf(file,"          <bcfl>focus</bcfl>\n");
02113         break;
02114     case BCFL_MAP:
02115         fprintf(file,"          <bcfl>map</bcfl>\n");
02116         break;
02117     case BCFL_MEM:
02118         fprintf(file,"          <bcfl>mem</bcfl>\n");
02119         break;
02120     default:
02121         break;
02122     }
02123
02124     fprintf(file,"          <temp>%4.3f K</temp>\n", pbeparm->temp);
02125
02126     for (; icalc<=nosh->elec2calc[ielec]; icalc++){ /* calc loop */
02127
02128         /* Reinitialize per-calc pointers */
02129         mgparm = nosh->calc[icalc]->mgparm;
02130         pbeparm = nosh->calc[icalc]->pbeparm;
02131
02132         fprintf(file,"          <calc>\n");
02133         fprintf(file,"          <id>%i</id>\n", (icalc+1));
02134         fprintf(file,"          <hx>%4.3f A</hx>\n", mgparm->grid[0]);
02135         fprintf(file,"          <hy>%4.3f A</hy>\n", mgparm->grid[1]);
02136         fprintf(file,"          <hz>%4.3f A</hz>\n", mgparm->grid[2]);
02137         fprintf(file,"          <xlen>%4.3f A</xlen>\n", mgparm->glen[0]);
02138         fprintf(file,"          <ylen>%4.3f A</ylen>\n", mgparm->glen[1]);
02139         fprintf(file,"          <zlen>%4.3f A</zlen>\n", mgparm->glen[2]);
02140
02141         if (pbeparm->calcenergy == PCE_TOTAL) {
02142             fprintf(file,"          <totEnergy>%1.12E kJ/mol</totEnergy>\n",
02143                 (totEnergy[icalc]*conversion));
02144         } else if (pbeparm->calcenergy == PCE_COMPS) {
02145             fprintf(file,"          <totEnergy>%1.12E kJ/mol</totEnergy>\n",
02146                 (totEnergy[icalc]*conversion));
02147             fprintf(file,"          <qfEnergy>%1.12E kJ/mol</qfEnergy>\n",
02148                 (0.5*qfEnergy[icalc]*conversion));
02149             fprintf(file,"          <qmEnergy>%1.12E kJ/mol</qmEnergy>\n",
02150                 (qmEnergy[icalc]*conversion));
02151             fprintf(file,"          <dielEnergy>%1.12E kJ/mol</dielEnergy>\n",
02152                 (dielEnergy[icalc]*conversion));
02153             for (i=0; i<nenergy[icalc]; i++){
02154                 fprintf(file,"          <atom>\n");
02155                 fprintf(file,"          <id>%i</id>\n", i+1);
02156                 fprintf(file,"          <energy>%1.12E kJ/mol</energy>\n",
02157                     (0.5*atomEnergy[icalc][i]*conversion));
02158                 fprintf(file,"          </atom>\n");
02159             }
02160         }
02161     }

```

```

02162
02163     if (pbeparm->calcforce == PCF_TOTAL) {
02164         fprintf(file, "          <qfforce_x>%1.12E</qfforce_x>\n",
02165             atomForce[icalc][0].qfForce[0]*conversion);
02166         fprintf(file, "          <qfforce_y>%1.12E</qfforce_y>\n",
02167             atomForce[icalc][0].qfForce[1]*conversion);
02168         fprintf(file, "          <qfforce_z>%1.12E</qfforce_z>\n",
02169             atomForce[icalc][0].qfForce[2]*conversion);
02170         fprintf(file, "          <ibforce_x>%1.12E</ibforce_x>\n",
02171             atomForce[icalc][0].ibForce[0]*conversion);
02172         fprintf(file, "          <ibforce_y>%1.12E</ibforce_y>\n",
02173             atomForce[icalc][0].ibForce[1]*conversion);
02174         fprintf(file, "          <ibforce_z>%1.12E</ibforce_z>\n",
02175             atomForce[icalc][0].ibForce[2]*conversion);
02176         fprintf(file, "          <dbforce_x>%1.12E</dbforce_x>\n",
02177             atomForce[icalc][0].dbForce[0]*conversion);
02178         fprintf(file, "          <dbforce_y>%1.12E</dbforce_y>\n",
02179             atomForce[icalc][0].dbForce[1]*conversion);
02180         fprintf(file, "          <dbforce_z>%1.12E</dbforce_z>\n",
02181             atomForce[icalc][0].dbForce[2]*conversion);
02182     }
02183     fprintf(file, "          </calc>\n");
02184 }
02185
02186     fprintf(file, "      </elec>\n");
02187 }
02188
02189 /* Handle print energy statements */
02190
02191 for (i=0; i<nosh->nprint; i++) {
02192     if (nosh->printwhat[i] == NPT_ENERGY) {
02193         fprintf(file, "      <printEnergy>\n");
02194         fprintf(file, "          <equation>%d", nosh->printcalc[i][0]+1);
02195
02196         for (j=1; j<nosh->printnarg[i]; j++) {
02197             if (nosh->printop[i][j-1] == 0) fprintf(file, " +");
02198             else if (nosh->printop[i][j-1] == 1) fprintf(file, " -");
02199             fprintf(file, " %d", nosh->printcalc[i][j]+1);
02200         }
02201
02202         fprintf(file, " </equation>\n");
02203         icalc = nosh->elec2calc[nosh->printcalc[i][0]];
02204
02205         ltenergy = Vunit_kb * (1e-3) * Vunit_Na * \
02206             nosh->calc[icalc]->pbeparm->temp * totEnergy[icalc];
02207
02208         for (j=1; j<nosh->printnarg[i]; j++) {
02209             icalc = nosh->elec2calc[nosh->printcalc[i][j]];
02210             /* Add or subtract? */
02211             if (nosh->printop[i][j-1] == 0) scalar = 1.0;
02212             else if (nosh->printop[i][j-1] == 1) scalar = -1.0;
02213             /* Accumulate */
02214             ltenergy += (scalar * Vunit_kb * (1e-3) * Vunit_Na * \
02215                 nosh->calc[icalc]->pbeparm->temp * totEnergy[icalc]);
02216         }
02217         Vcom_reduce(com, &ltenergy, &gtenergy, 1, 2, 0);
02218         fprintf(file, "          <localEnergy>%1.12E kJ/mol</localEnergy>\n", \
02219             ltenergy);
02220         fprintf(file, "          <globalEnergy>%1.12E kJ/mol</globalEnergy>\n", \
02221             gtenergy);
02222     }
02223     fprintf(file, "      </printEnergy>\n");
02224 }
02225
02226 /* Add ending tags and close the file */
02227 fprintf(file, "</APBS>\n");
02228 fclose(file);
02229
02230 return 1;
02231 }
02232
02233 VPUBLIC int writedataMG(int rank,
02234     NOSH *nosh,
02235     PBEPARM *pbeparm,
02236     VPMG *pmg)
02237 {

```

```

02243     char writestem[VMAX_ARGLEN];
02244     char outpath[VMAX_ARGLEN];
02245     char title[72];
02246     int i,
02247         nx,
02248         ny,
02249         nz,
02250         natoms;
02251     double hx,
02252            hy,
02253            hzed,
02254            xcent,
02255            ycent,
02256            zcent,
02257            xmin,
02258            ymin,
02259            zmin;
02260
02261     Vgrid *grid;
02262     Vio *sock;
02263
02264     if (nosh->bogus) return 1;
02265
02266     for (i=0; i<pbeparm->numwrite; i++) {
02267
02268         nx = pmg->pmgp->nx;
02269         ny = pmg->pmgp->ny;
02270         nz = pmg->pmgp->nz;
02271         hx = pmg->pmgp->hx;
02272         hy = pmg->pmgp->hy;
02273         hzed = pmg->pmgp->hzed;
02274
02275         switch (pbeparm->writetype[i]) {
02276
02277             case VDT_CHARGE:
02278
02279                 Vnm_tprint(1, " Writing charge distribution to ");
02280                 xcent = pmg->pmgp->xcent;
02281                 ycent = pmg->pmgp->ycent;
02282                 zcent = pmg->pmgp->zcent;
02283                 xmin = xcent - 0.5*(nx-1)*hx;
02284                 ymin = ycent - 0.5*(ny-1)*hy;
02285                 zmin = zcent - 0.5*(nz-1)*hzed;
02286                 VASSERT(Vpmg_fillArray(pmg, pmg->rwork,
VDT_CHARGE, 0.0,
02287                                     pbeparm->pbetype, pbeparm));
02288                 sprintf(title, "CHARGE DISTRIBUTION (e)");
02289                 break;
02290
02291             case VDT_POT:
02292
02293                 Vnm_tprint(1, " Writing potential to ");
02294                 xcent = pmg->pmgp->xcent;
02295                 ycent = pmg->pmgp->ycent;
02296                 zcent = pmg->pmgp->zcent;
02297                 xmin = xcent - 0.5*(nx-1)*hx;
02298                 ymin = ycent - 0.5*(ny-1)*hy;
02299                 zmin = zcent - 0.5*(nz-1)*hzed;
02300                 VASSERT(Vpmg_fillArray(pmg, pmg->rwork,
VDT_POT, 0.0,
02301                                     pbeparm->pbetype, pbeparm));
02302                 sprintf(title, "POTENTIAL (kT/e)");
02303                 break;
02304
02305             case VDT_SMOL:
02306
02307                 Vnm_tprint(1, " Writing molecular accessibility to ");
02308                 xcent = pmg->pmgp->xcent;
02309                 ycent = pmg->pmgp->ycent;
02310                 zcent = pmg->pmgp->zcent;
02311                 xmin = xcent - 0.5*(nx-1)*hx;
02312                 ymin = ycent - 0.5*(ny-1)*hy;
02313                 zmin = zcent - 0.5*(nz-1)*hzed;
02314                 VASSERT(Vpmg_fillArray(pmg, pmg->rwork,
VDT_SMOL,
02315                                     pbeparm->srad, pbeparm->pbetype, pbeparm));
02316                 sprintf(title,
02317                         "SOLVENT ACCESSIBILITY -- MOLECULAR (%4.3f PROBE)",
02318                         pbeparm->srad);
02319                 break;
02320

```

```

02321         case VDT_SSPL:
02322
02323             Vnm_tprint(1, " Writing spline-based accessibility to ");
02324             xcent = pmg->pmgp->xcent;
02325             ycent = pmg->pmgp->ycent;
02326             zcent = pmg->pmgp->zcent;
02327             xmin = xcent - 0.5*(nx-1)*hx;
02328             ymin = ycent - 0.5*(ny-1)*hy;
02329             zmin = zcent - 0.5*(nz-1)*hzed;
02330             VASSERT(Vpmg_fillArray(pmg, pmg->rwork,
VDT_SSPL,
02331                                     pbeparm->swin, pbeparm->pbetype, pbeparm));
02332             sprintf(title,
02333                     "SOLVENT ACCESSIBILITY -- SPLINE (%4.3f WINDOW)",
02334                     pbeparm->swin);
02335             break;
02336
02337         case VDT_VDW:
02338
02339             Vnm_tprint(1, " Writing van der Waals accessibility to ");
02340             xcent = pmg->pmgp->xcent;
02341             ycent = pmg->pmgp->ycent;
02342             zcent = pmg->pmgp->zcent;
02343             xmin = xcent - 0.5*(nx-1)*hx;
02344             ymin = ycent - 0.5*(ny-1)*hy;
02345             zmin = zcent - 0.5*(nz-1)*hzed;
02346             VASSERT(Vpmg_fillArray(pmg, pmg->rwork,
VDT_VDW, 0.0,
02347                                     pbeparm->pbetype, pbeparm));
02348             sprintf(title, "SOLVENT ACCESSIBILITY -- VAN DER WAALS");
02349             break;
02350
02351         case VDT_IVDW:
02352
02353             Vnm_tprint(1, " Writing ion accessibility to ");
02354             xcent = pmg->pmgp->xcent;
02355             ycent = pmg->pmgp->ycent;
02356             zcent = pmg->pmgp->zcent;
02357             xmin = xcent - 0.5*(nx-1)*hx;
02358             ymin = ycent - 0.5*(ny-1)*hy;
02359             zmin = zcent - 0.5*(nz-1)*hzed;
02360             VASSERT(Vpmg_fillArray(pmg, pmg->rwork,
VDT_IVDW,
02361                                     pmg->pbe->maxIonRadius, pbeparm->
pbetype, pbeparm));
02362             sprintf(title,
02363                     "ION ACCESSIBILITY -- SPLINE (%4.3f RADIUS)",
02364                     pmg->pbe->maxIonRadius);
02365             break;
02366
02367         case VDT_LAP:
02368
02369             Vnm_tprint(1, " Writing potential Laplacian to ");
02370             xcent = pmg->pmgp->xcent;
02371             ycent = pmg->pmgp->ycent;
02372             zcent = pmg->pmgp->zcent;
02373             xmin = xcent - 0.5*(nx-1)*hx;
02374             ymin = ycent - 0.5*(ny-1)*hy;
02375             zmin = zcent - 0.5*(nz-1)*hzed;
02376             VASSERT(Vpmg_fillArray(pmg, pmg->rwork,
VDT_LAP, 0.0,
02377                                     pbeparm->pbetype, pbeparm));
02378             sprintf(title,
02379                     "POTENTIAL LAPLACIAN (kT/e/A^2)");
02380             break;
02381
02382         case VDT_EDENS:
02383
02384             Vnm_tprint(1, " Writing energy density to ");
02385             xcent = pmg->pmgp->xcent;
02386             ycent = pmg->pmgp->ycent;
02387             zcent = pmg->pmgp->zcent;
02388             xmin = xcent - 0.5*(nx-1)*hx;
02389             ymin = ycent - 0.5*(ny-1)*hy;
02390             zmin = zcent - 0.5*(nz-1)*hzed;
02391             VASSERT(Vpmg_fillArray(pmg, pmg->rwork,
VDT_EDENS, 0.0,
02392                                     pbeparm->pbetype, pbeparm));
02393             sprintf(title, "ENERGY DENSITY (kT/e/A)^2");
02394             break;
02395

```

```

02396         case VDT_NDENS:
02397
02398             Vnm_tprint(1, " Writing number density to ");
02399             xcent = pmg->pmgp->xcent;
02400             ycent = pmg->pmgp->ycent;
02401             zcent = pmg->pmgp->zcent;
02402             xmin = xcent - 0.5*(nx-1)*hx;
02403             ymin = ycent - 0.5*(ny-1)*hy;
02404             zmin = zcent - 0.5*(nz-1)*hzed;
02405             VASSERT(Vpmg_fillArray(pmg, pmg->rwork,
VDT_NDENS, 0.0,
02406                                     pbeparm->pbetype, pbeparm));
02407             sprintf(title,
02408                     "ION NUMBER DENSITY (M)");
02409             break;
02410         case VDT_QDENS:
02411
02412             Vnm_tprint(1, " Writing charge density to ");
02413             xcent = pmg->pmgp->xcent;
02414             ycent = pmg->pmgp->ycent;
02415             zcent = pmg->pmgp->zcent;
02416             xmin = xcent - 0.5*(nx-1)*hx;
02417             ymin = ycent - 0.5*(ny-1)*hy;
02418             zmin = zcent - 0.5*(nz-1)*hzed;
02419             VASSERT(Vpmg_fillArray(pmg, pmg->rwork,
VDT_QDENS, 0.0,
02420                                     pbeparm->pbetype, pbeparm));
02421             sprintf(title,
02422                     "ION CHARGE DENSITY (e_c * M)");
02423             break;
02424         case VDT_DIELX:
02425
02426             Vnm_tprint(1, " Writing x-shifted dielectric map to ");
02427             xcent = pmg->pmgp->xcent + 0.5*hx;
02428             ycent = pmg->pmgp->ycent;
02429             zcent = pmg->pmgp->zcent;
02430             xmin = xcent - 0.5*(nx-1)*hx;
02431             ymin = ycent - 0.5*(ny-1)*hy;
02432             zmin = zcent - 0.5*(nz-1)*hzed;
02433             VASSERT(Vpmg_fillArray(pmg, pmg->rwork,
VDT_DIELX, 0.0,
02434                                     pbeparm->pbetype, pbeparm));
02435             sprintf(title,
02436                     "X-SHIFTED DIELECTRIC MAP");
02437             break;
02438         case VDT_DIELY:
02439
02440             Vnm_tprint(1, " Writing y-shifted dielectric map to ");
02441             xcent = pmg->pmgp->xcent;
02442             ycent = pmg->pmgp->ycent + 0.5*hy;
02443             zcent = pmg->pmgp->zcent;
02444             xmin = xcent - 0.5*(nx-1)*hx;
02445             ymin = ycent - 0.5*(ny-1)*hy;
02446             zmin = zcent - 0.5*(nz-1)*hzed;
02447             VASSERT(Vpmg_fillArray(pmg, pmg->rwork,
VDT_DIELY, 0.0,
02448                                     pbeparm->pbetype, pbeparm));
02449             sprintf(title,
02450                     "Y-SHIFTED DIELECTRIC MAP");
02451             break;
02452         case VDT_DIELZ:
02453
02454             Vnm_tprint(1, " Writing z-shifted dielectric map to ");
02455             xcent = pmg->pmgp->xcent;
02456             ycent = pmg->pmgp->ycent;
02457             zcent = pmg->pmgp->zcent + 0.5*hzed;
02458             xmin = xcent - 0.5*(nx-1)*hx;
02459             ymin = ycent - 0.5*(ny-1)*hy;
02460             zmin = zcent - 0.5*(nz-1)*hzed;
02461             VASSERT(Vpmg_fillArray(pmg, pmg->rwork,
VDT_DIELZ, 0.0,
02462                                     pbeparm->pbetype, pbeparm));
02463             sprintf(title,
02464                     "Z-SHIFTED DIELECTRIC MAP");
02465             break;
02466         case VDT_KAPPA:

```

```

02472
02473         Vnm_tprint(1, " Writing kappa map to ");
02474         xcent = pmg->pmgp->xcent;
02475         ycent = pmg->pmgp->ycent;
02476         zcent = pmg->pmgp->zcent;
02477         xmin = xcent - 0.5*(nx-1)*hx;
02478         ymin = ycent - 0.5*(ny-1)*hy;
02479         zmin = zcent - 0.5*(nz-1)*hzed;
02480         VASSERT(Vpmg_fillArray(pmg, pmg->rwork,
VDT_KAPPA, 0.0,
                                pbeparm->pbetype, pbeparm));
02481         sprintf(title,
02482                 "KAPPA MAP");
02483         break;
02484
02485     case VDT_ATOMPOT:
02486
02487         Vnm_tprint(1, " Writing atom potentials to ");
02488         xcent = pmg->pmgp->xcent;
02489         ycent = pmg->pmgp->ycent;
02490         zcent = pmg->pmgp->zcent;
02491         xmin = xcent - 0.5*(nx-1)*hx;
02492         ymin = ycent - 0.5*(ny-1)*hy;
02493         zmin = zcent - 0.5*(nz-1)*hzed;
02494         VASSERT(Vpmg_fillArray(pmg, pmg->rwork,
VDT_ATOMPOT, 0.0,
                                pbeparm->pbetype, pbeparm));
02496         sprintf(title,
02497                 "ATOM POTENTIALS");
02498         break;
02499     default:
02500
02501         Vnm_tprint(2, "Invalid data type for writing!\n");
02502         return 0;
02503     }
02504 }
02505
02506 #ifdef HAVE_MPI_H
02507     sprintf(writestem, "%s-PE%d", pbeparm->writestem[i], rank);
02508 #else
02509     if(nosh->ispara){
02510         sprintf(writestem, "%s-PE%d", pbeparm->writestem[i], nosh->
proc_rank);
02511     }else{
02512         sprintf(writestem, "%s", pbeparm->writestem[i]);
02513     }
02514 #endif
02515
02516     switch (pbeparm->writefmt[i]) {
02517     case VDF_DX:
02518         sprintf(outpath, "%.s", writestem, "dx");
02519         Vnm_tprint(1, "%s\n", outpath);
02520         grid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin,
02521                         pmg->rwork);
02522         Vgrid_writeDX(grid, "FILE", "ASC", VNULL, outpath, title,
02523                      pmg->pvec);
02524         Vgrid_dtor(&grid);
02525         break;
02526
02527     case VDF_AVS:
02528         sprintf(outpath, "%.s", writestem, "ucd");
02529         Vnm_tprint(1, "%s\n", outpath);
02530         Vnm_tprint(2, "Sorry, AVS format isn't supported for \
02531 uniform meshes yet!\n");
02532         break;
02533
02534     case VDF_MCSF:
02535         sprintf(outpath, "%.s", writestem, "mcsf");
02536         Vnm_tprint(1, "%s\n", outpath);
02537         Vnm_tprint(2, "Sorry, MCSF format isn't supported for \
02538 uniform meshes yet!\n");
02539         break;
02540
02541     case VDF_UHBD:
02542         sprintf(outpath, "%.s", writestem, "grd");
02543         Vnm_tprint(1, "%s\n", outpath);
02544         grid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin,
02545                         pmg->rwork);
02546         Vgrid_writeUHBD(grid, "FILE", "ASC", VNULL, outpath, title,
02547                        pmg->pvec);
02548     }
02549

```

```

02550         Vgrid_dtor(&grid);
02551         break;
02552
02553     case VDF_GZ:
02554         sprintf(outpath, "%.s", writestem, "dx.gz");
02555         Vnm_tprint(1, "%s\n", outpath);
02556         grid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin,
02557             pmg->rwork);
02558         Vgrid_writeGZ(grid, "FILE", "ASC", VNULL, outpath, title,
02559             pmg->pvec);
02560         Vgrid_dtor(&grid);
02561         break;
02562     case VDF_FLAT:
02563         sprintf(outpath, "%.s", writestem, "txt");
02564         Vnm_tprint(1, "%s\n", outpath);
02565         Vnm_print(0, "routines: Opening virtual socket...\n");
02566         sock = Vio_ctor("FILE", "ASC", VNULL, outpath, "w");
02567         if (sock == VNULL) {
02568             Vnm_print(2, "routines: Problem opening virtual socket %s\n",
02569                 outpath);
02570             return 0;
02571         }
02572         if (Vio_connect(sock, 0) < 0) {
02573             Vnm_print(2, "routines: Problem connecting virtual socket %s\n",
02574                 outpath);
02575             return 0;
02576         }
02577         Vio_printf(sock, "# Data from %s\n", PACKAGE_STRING);
02578         Vio_printf(sock, "# \n");
02579         Vio_printf(sock, "# %s\n", title);
02580         Vio_printf(sock, "# \n");
02581         natoms = pmg->pbe->alist[pbeparm->molid-1].number;
02582         for (i=0; i<natoms; i++)
02583             Vio_printf(sock, "%12.6e\n", pmg->rwork[i]);
02584         break;
02585     default:
02586         Vnm_tprint(2, "Bogus data format (%d)!\n",
02587             pbeparm->writefmt[i]);
02588         break;
02589 }
02590 }
02591 }
02592
02593     return 1;
02594 }
02595
02596 VPUBLIC double returnEnergy(Vcom *com,
02597     Nosh *nosh,
02598     double totEnergy[NOSH_MAXCALC],
02599     int iprint
02600 ){
02601
02602     int iarg,
02603         calcid;
02604     double ltenergy,
02605         scalar;
02606
02607     calcid = nosh->elec2calc[nosh->printcalc[iprint][0]];
02608     if (nosh->calc[calcid]->pbeparm->calcenergy != PCE_NO) {
02609         ltenergy = Vunit_kb * (1e-3) * Vunit_Na *
02610             nosh->calc[calcid]->pbeparm->temp * totEnergy[calcid];
02611     } else {
02612         Vnm_tprint( 2, " No energy available in Calculation %d\n", calcid+1);
02613         return 0.0;
02614     }
02615     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++){
02616         calcid = nosh->elec2calc[nosh->printcalc[iprint][iarg]];
02617         /* Add or subtract */
02618         if (nosh->printop[iprint][iarg-1] == 0) scalar = 1.0;
02619         else if (nosh->printop[iprint][iarg-1] == 1) scalar = -1.0;
02620         /* Accumulate */
02621         ltenergy += (scalar * Vunit_kb * (1e-3) * Vunit_Na *
02622             nosh->calc[calcid]->pbeparm->temp * totEnergy[calcid]);
02623     }
02624
02625     return ltenergy;
02626 }
02627
02628 VPUBLIC int printEnergy(Vcom *com,
02629     Nosh *nosh,
02630     double totEnergy[NOSH_MAXCALC],

```

```

02631             int iprint
02632         ) {
02633
02634     int iarg,
02635         calcid;
02636     double ltenergy,
02637         gtenergy,
02638         scalar;
02639
02640     Vnm_tprint( 2, "Warning: The 'energy' print keyword is deprecated.\n" \
02641         "           Use eilecEnergy for electrostatics energy calcs.\n\n");
02642
02643     if (Vstring_strcasecmp(nosh->elecname[nosh->
02644 printcalc[iprint][0]], "") == 0){
02645         Vnm_tprint( 1, "print energy %d ", nosh->printcalc[iprint][0]+1);
02646     } else {
02647         Vnm_tprint( 1, "print energy %d (%s) ", nosh->printcalc[iprint][0]+1,
02648             nosh->elecname[nosh->printcalc[iprint][0]]);
02649     }
02650     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
02651         if (nosh->printop[iprint][iarg-1] == 0)
02652             Vnm_tprint(1, "+ ");
02653         else if (nosh->printop[iprint][iarg-1] == 1)
02654             Vnm_tprint(1, "- ");
02655         else {
02656             Vnm_tprint( 2, "Undefined PRINT operation!\n");
02657             return 0;
02658         }
02659         if (Vstring_strcasecmp(nosh->elecname[nosh->
02660 printcalc[iprint][iarg]],
02661             "") == 0) {
02662             Vnm_tprint( 1, "%d ", nosh->printcalc[iprint][iarg]+1);
02663         } else {
02664             Vnm_tprint( 1, "%d (%s) ", nosh->printcalc[iprint][iarg]+1,
02665                 nosh->elecname[nosh->printcalc[iprint][iarg]]);
02666         }
02667     }
02668     Vnm_tprint(1, "end\n");
02669     calcid = nosh->elec2calc[nosh->printcalc[iprint][0]];
02670     if (nosh->calc[calcid]->pbeparm->calcenergy != PCE_NO) {
02671         ltenergy = Vunit_kb * (1e-3) * Vunit_Na *
02672             nosh->calc[calcid]->pbeparm->temp * totEnergy[calcid];
02673     } else {
02674         Vnm_tprint( 2, " Didn't calculate energy in Calculation \
02675 #d\n", calcid+1);
02676         return 0;
02677     }
02678     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
02679         calcid = nosh->elec2calc[nosh->printcalc[iprint][iarg]];
02680         /* Add or subtract? */
02681         if (nosh->printop[iprint][iarg-1] == 0) scalar = 1.0;
02682         else if (nosh->printop[iprint][iarg-1] == 1) scalar = -1.0;
02683         /* Accumulate */
02684         ltenergy += (scalar * Vunit_kb * (1e-3) * Vunit_Na *
02685             nosh->calc[calcid]->pbeparm->temp * totEnergy[calcid]);
02686     }
02687     Vnm_tprint( 1, " Local net energy (PE %d) = %1.12E kJ/mol\n",
02688         Vcom_rank(com), ltenergy);
02689     Vnm_tprint( 0, "printEnergy: Performing global reduction (sum)\n");
02690     Vcom_reduce(com, &ltenergy, &gtenergy, 1, 2, 0);
02691     Vnm_tprint( 1, " Global net ELEC energy = %1.12E kJ/mol\n", gtenergy);
02692     return 1;
02693 }
02694
02695
02696 VPUBLIC int printElecEnergy(Vcom *com,
02697     Nosh *nosh,
02698     double totEnergy[NOSH_MAXCALC],
02699     int iprint
02700 ) {
02701
02702     int iarg,
02703         calcid;
02704     double ltenergy,
02705         gtenergy,
02706         scalar;
02707
02708     if (Vstring_strcasecmp(nosh->elecname[nosh->
02709 printcalc[iprint][0]], "") == 0){

```



```

02709     Vnm_tprint( 1, "\nprint energy %d ", nosh->printcalc[iprint][0]+1);
02710 } else {
02711     Vnm_tprint( 1, "\nprint energy %d (%s) ", nosh->printcalc[iprint][0]+1,
02712               nosh->elecname[nosh->printcalc[iprint][0]]);
02713 }
02714 for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
02715     if (nosh->printtop[iprint][iarg-1] == 0)
02716         Vnm_tprint(1, "+ ");
02717     else if (nosh->printtop[iprint][iarg-1] == 1)
02718         Vnm_tprint(1, "- ");
02719     else {
02720         Vnm_tprint( 2, "Undefined PRINT operation!\n");
02721         return 0;
02722     }
02723     if (Vstring_strcasecmp(nosh->elecname[nosh->
printcalc[iprint][iarg]],
02724                           "") == 0) {
02725         Vnm_tprint( 1, "%d ", nosh->printcalc[iprint][iarg]+1);
02726     } else {
02727         Vnm_tprint( 1, "%d (%s) ", nosh->printcalc[iprint][iarg]+1,
02728                   nosh->elecname[nosh->printcalc[iprint][iarg]]);
02729     }
02730 }
02731 Vnm_tprint(1, "end\n");
02732 calcid = nosh->elec2calc[nosh->printcalc[iprint][0]];
02733 if (nosh->calc[calcid]->pbeparm->calcenergy != PCE_NO) {
02734     ltenergy = Vunit_kb * (1e-3) * Vunit_Na *
02735               nosh->calc[calcid]->pbeparm->temp * totEnergy[calcid];
02736 } else {
02737     Vnm_tprint( 2, " Didn't calculate energy in Calculation \
02738 #d\n", calcid+1);
02739     return 0;
02740 }
02741 for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
02742     calcid = nosh->elec2calc[nosh->printcalc[iprint][iarg]];
02743     /* Add or subtract? */
02744     if (nosh->printtop[iprint][iarg-1] == 0) scalar = 1.0;
02745     else if (nosh->printtop[iprint][iarg-1] == 1) scalar = -1.0;
02746     /* Accumulate */
02747     ltenergy += (scalar * Vunit_kb * (1e-3) * Vunit_Na *
02748                 nosh->calc[calcid]->pbeparm->temp * totEnergy[calcid]);
02749 }
02750
02751 Vnm_tprint( 1, " Local net energy (PE %d) = %1.12E kJ/mol\n",
02752             Vcom_rank(com), ltenergy);
02753 Vnm_tprint( 0, "printEnergy: Performing global reduction (sum)\n");
02754 Vcom_reduce(com, &ltenergy, &gtenergy, 1, 2, 0);
02755 Vnm_tprint( 1, " Global net ELEC energy = %1.12E kJ/mol\n", gtenenergy);
02756
02757 return 1;
02758 }
02759 }
02760
02761 VPUBLIC int printApolEnergy(NOsh *nosh,
02762                             int iprint
02763                             ) {
02764
02765     int iarg,
02766         calcid;
02767     double gtenergy,
02768         scalar;
02769     APOLparm *apolparm = VNULL;
02770
02771     if (Vstring_strcasecmp(nosh->apolname[nosh->
printcalc[iprint][0]], "") == 0){
02772         Vnm_tprint( 1, "\nprint APOL energy %d ", nosh->printcalc[iprint][0]+1);
02773     } else {
02774         Vnm_tprint( 1, "\nprint APOL energy %d (%s) ", nosh->printcalc[iprint][0]+1,
02775                   nosh->apolname[nosh->printcalc[iprint][0]]);
02776     }
02777     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
02778         if (nosh->printtop[iprint][iarg-1] == 0)
02779             Vnm_tprint(1, "+ ");
02780         else if (nosh->printtop[iprint][iarg-1] == 1)
02781             Vnm_tprint(1, "- ");
02782         else {
02783             Vnm_tprint( 2, "Undefined PRINT operation!\n");
02784             return 0;
02785         }
02786         if (Vstring_strcasecmp(nosh->apolname[nosh->
printcalc[iprint][iarg]],

```

```

02787         "" == 0) {
02788             Vnm_tprint( 1, "%d ", nosh->printcalc[iprint][iarg]+1);
02789         } else {
02790             Vnm_tprint( 1, "%d (%s) ", nosh->printcalc[iprint][iarg]+1,
02791                 nosh->apolname[nosh->printcalc[iprint][iarg]]);
02792         }
02793     }
02794     Vnm_tprint(1, "end\n");
02795
02796     calcid = nosh->apol2calc[nosh->printcalc[iprint][0]];
02797     apolparm = nosh->calc[calcid]->apolparm;
02798
02799     if (apolparm->calcenergy == ACE_TOTAL) {
02800         gtenergy = ((apolparm->gamma*apolparm->sasa) + (apolparm->press*apolparm->
02801 sav) + (apolparm->wcaEnergy));
02802     } else {
02803         Vnm_tprint( 2, " Didn't calculate energy in Calculation #d\n", calcid+1);
02804         return 0;
02805     }
02806     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
02807         calcid = nosh->apol2calc[nosh->printcalc[iprint][iarg]];
02808         apolparm = nosh->calc[calcid]->apolparm;
02809
02810         /* Add or subtract? */
02811         if (nosh->printop[iprint][iarg-1] == 0) scalar = 1.0;
02812         else if (nosh->printop[iprint][iarg-1] == 1) scalar = -1.0;
02813         /* Accumulate */
02814         gtenergy += (scalar * ((apolparm->gamma*apolparm->sasa) +
02815             (apolparm->press*apolparm->sav) +
02816             (apolparm->wcaEnergy)));
02817     }
02818
02819     Vnm_tprint( 1, " Global net APOL energy = %1.12E kJ/mol\n", gtenergy);
02820     return 1;
02821 }
02822
02823 VPUBLIC int printForce(Vcom *com,
02824     Nosh *nosh,
02825     int nforce[NOSH_MAXCALC],
02826     AtomForce *atomForce[NOSH_MAXCALC],
02827     int iprint
02828 ) {
02829
02830     int iarg,
02831     ifr,
02832     ivc,
02833     calcid,
02834     refnforce,
02835     refcalcforce;
02836     double temp,
02837     scalar,
02838     totforce[3];
02839     AtomForce *lforce,
02840     *gforce,
02841     *aforce;
02842
02843     Vnm_tprint( 2, "Warning: The 'force' print keyword is deprecated.\n" \
02844         " Use elecForce for electrostatics force calcs.\n\n");
02845
02846     if (Vstring_strcasecmp(nosh->elecname[nosh->
02847 printcalc[iprint][0]], "") == 0){
02848         Vnm_tprint( 1, "print force %d ", nosh->printcalc[iprint][0]+1);
02849     } else {
02850         Vnm_tprint( 1, "print force %d (%s) ", nosh->printcalc[iprint][0]+1,
02851             nosh->elecname[nosh->printcalc[iprint][0]]);
02852     }
02853     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
02854         if (nosh->printop[iprint][iarg-1] == 0)
02855             Vnm_tprint(1, "+ ");
02856         else if (nosh->printop[iprint][iarg-1] == 1)
02857             Vnm_tprint(1, "- ");
02858         else {
02859             Vnm_tprint( 2, "Undefined PRINT operation!\n");
02860             return 0;
02861         }
02862         if (Vstring_strcasecmp(nosh->elecname[nosh->
02863 printcalc[iprint][iarg]],
02864             "" == 0) {
02865             Vnm_tprint( 1, "%d ", nosh->printcalc[iprint][iarg]+1);
02866         } else {

```

```

02865         Vnm_tprint( 1, "%d (%s) ", nosh->printcalc[iprint][iarg]+1,
02866                     nosh->elecname[nosh->printcalc[iprint][iarg]]);
02867     }
02868 }
02869 Vnm_tprint(1, "end\n");
02870
02871 /* First, go through and make sure we did the same type of force
02872  * evaluation in each of the requested calculations */
02873 calcid = nosh->elec2calc[nosh->printcalc[iprint][0]];
02874 refnforce = nforce[calcid];
02875 refcalcforce = nosh->calc[calcid]->pbeparm->calcforce;
02876 if (refcalcforce == PCF_NO) {
02877     Vnm_tprint( 2, " Didn't calculate force in calculation \
02878 #d\n", calcid+1);
02879     return 0;
02880 }
02881 for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
02882     calcid = nosh->elec2calc[nosh->printcalc[iprint][iarg]-1];
02883     if (nosh->calc[calcid]->pbeparm->calcforce != refcalcforce) {
02884         Vnm_tprint(2, " Inconsistent calcforce declarations in \
02885 calculations %d and %d\n", nosh->elec2calc[nosh->printcalc[iprint][0]]+1,
02886                     calcid+1);
02887         return 0;
02888     }
02889     if (nforce[calcid] != refnforce) {
02890         Vnm_tprint(2, " Inconsistent number of forces evaluated in \
02891 calculations %d and %d\n", nosh->elec2calc[nosh->printcalc[iprint][0]]+1,
02892                     calcid+1);
02893         return 0;
02894     }
02895 }
02896
02897 /* Now, allocate space to accumulate the forces */
02898 lforce = (AtomForce *)Vmem_malloc(VNULL, refnforce, sizeof(
AtomForce));
02899 gforce = (AtomForce *)Vmem_malloc(VNULL, refnforce, sizeof(
AtomForce));
02900
02901 /* Now, accumulate the forces */
02902 calcid = nosh->elec2calc[nosh->printcalc[iprint][0]];
02903 aforce = atomForce[calcid];
02904 temp = nosh->calc[calcid]->pbeparm->temp;
02905
02906 /* Load up the first calculation */
02907 if (refcalcforce == PCF_TOTAL) {
02908     /* Set to total force */
02909     for (ivc=0; ivc<3; ivc++) {
02910         lforce[0].qfForce[ivc] =
02911             Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].qfForce[ivc];
02912         lforce[0].ibForce[ivc] =
02913             Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].ibForce[ivc];
02914         lforce[0].dbForce[ivc] =
02915             Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].dbForce[ivc];
02916     }
02917 } else if (refcalcforce == PCF_COMPS) {
02918     for (ifr=0; ifr<refnforce; ifr++) {
02919         for (ivc=0; ivc<3; ivc++) {
02920             lforce[ifr].qfForce[ivc] =
02921                 Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].qfForce[ivc];
02922             lforce[ifr].ibForce[ivc] =
02923                 Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].ibForce[ivc];
02924             lforce[ifr].dbForce[ivc] =
02925                 Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].dbForce[ivc];
02926         }
02927     }
02928 }
02929
02930 /* Load up the rest of the calculations */
02931 for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
02932     calcid = nosh->elec2calc[nosh->printcalc[iprint][iarg]];
02933     temp = nosh->calc[calcid]->pbeparm->temp;
02934     aforce = atomForce[calcid];
02935     /* Get operation */
02936     if (nosh->printop[iprint][iarg-1] == 0) scalar = +1.0;
02937     else if (nosh->printop[iprint][iarg-1] == 1) scalar = -1.0;
02938     else scalar = 0.0;
02939     /* Accumulate */
02940     if (refcalcforce == PCF_TOTAL) {
02941         /* Set to total force */
02942         for (ivc=0; ivc<3; ivc++) {
02943             lforce[0].qfForce[ivc] +=

```

```

02944         (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].qfForce[ivc]);
02945         lforce[0].ibForce[ivc] +=
02946         (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].ibForce[ivc]);
02947         lforce[0].dbForce[ivc] +=
02948         (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].dbForce[ivc]);
02949     }
02950     } else if (refcalcforce == PCF_COMPS) {
02951         for (ifr=0; ifr<refnforce; ifr++) {
02952             for (ivc=0; ivc<3; ivc++) {
02953                 lforce[ifr].qfForce[ivc] +=
02954                 (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].qfForce[ivc]);
02955                 lforce[ifr].ibForce[ivc] +=
02956                 (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].ibForce[ivc]);
02957                 lforce[ifr].dbForce[ivc] +=
02958                 (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].dbForce[ivc]);
02959             }
02960         }
02961     }
02962 }
02963
02964 Vnm_tprint( 0, "printEnergy: Performing global reduction (sum)\n");
02965 for (ifr=0; ifr<refnforce; ifr++) {
02966     Vcom_reduce(com, lforce[ifr].qfForce, gforce[ifr].qfForce, 3, 2, 0);
02967     Vcom_reduce(com, lforce[ifr].ibForce, gforce[ifr].ibForce, 3, 2, 0);
02968     Vcom_reduce(com, lforce[ifr].dbForce, gforce[ifr].dbForce, 3, 2, 0);
02969 }
02970
02971 #if 0
02972     if (refcalcforce == PCF_TOTAL) {
02973         Vnm_tprint( 1, " Local net fixed charge force = \
02974 (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", lforce[0].qfForce[0],
02975         lforce[0].qfForce[1], lforce[0].qfForce[2]);
02976         Vnm_tprint( 1, " Local net ionic boundary force = \
02977 (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", lforce[0].ibForce[0],
02978         lforce[0].ibForce[1], lforce[0].ibForce[2]);
02979         Vnm_tprint( 1, " Local net dielectric boundary force = \
02980 (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", lforce[0].dbForce[0],
02981         lforce[0].dbForce[1], lforce[0].dbForce[2]);
02982     } else if (refcalcforce == PCF_COMPS) {
02983         for (ifr=0; ifr<refnforce; ifr++) {
02984             Vnm_tprint( 1, " Local fixed charge force \
02985 (atom %d) = (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", ifr, lforce[ifr].qfForce[0],
02986             lforce[ifr].qfForce[1], lforce[ifr].qfForce[2]);
02987             Vnm_tprint( 1, " Local ionic boundary force \
02988 (atom %d) = (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", ifr, lforce[ifr].ibForce[0],
02989             lforce[ifr].ibForce[1], lforce[ifr].ibForce[2]);
02990             Vnm_tprint( 1, " Local dielectric boundary force \
02991 (atom %d) = (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", ifr, lforce[ifr].dbForce[0],
02992             lforce[ifr].dbForce[1], lforce[ifr].dbForce[2]);
02993         }
02994     }
02995 #endif
02996
02997     if (refcalcforce == PCF_TOTAL) {
02998         Vnm_tprint( 1, " Printing net forces (kJ/mol/A).\n");
02999         Vnm_tprint( 1, " Legend:\n");
03000         Vnm_tprint( 1, " tot -- Total force\n");
03001         Vnm_tprint( 1, " qf -- Fixed charge force\n");
03002         Vnm_tprint( 1, " db -- Dielectric boundary force\n");
03003         Vnm_tprint( 1, " ib -- Ionic boundary force\n");
03004
03005         for (ivc=0; ivc<3; ivc++) {
03006             totforce[ivc] =
03007             gforce[0].qfForce[ivc] + gforce[0].ibForce[ivc] \
03008             + gforce[0].dbForce[ivc];
03009         }
03010
03011         Vnm_tprint( 1, " tot %1.12E %1.12E %1.12E\n", totforce[0],
03012         totforce[1], totforce[2]);
03013         Vnm_tprint( 1, " qf %1.12E %1.12E %1.12E\n", gforce[0].qfForce[0],
03014         gforce[0].qfForce[1], gforce[0].qfForce[2]);
03015         Vnm_tprint( 1, " ib %1.12E %1.12E %1.12E\n", gforce[0].ibForce[0],
03016         gforce[0].ibForce[1], gforce[0].ibForce[2]);
03017         Vnm_tprint( 1, " db %1.12E %1.12E %1.12E\n", gforce[0].dbForce[0],
03018         gforce[0].dbForce[1], gforce[0].dbForce[2]);
03019
03020     } else if (refcalcforce == PCF_COMPS) {
03021
03022         Vnm_tprint( 1, " Printing per-atom forces (kJ/mol/A).\n");
03023         Vnm_tprint( 1, " Legend:\n");
03024         Vnm_tprint( 1, " tot n -- Total force for atom n\n");

```

```

03025     Vnm_tprint( 1, "    qf  n -- Fixed charge force for atom n\n");
03026     Vnm_tprint( 1, "    db  n -- Dielectric boundary force for atom n\n");
03027     Vnm_tprint( 1, "    ib  n -- Ionic boundary force for atom n\n");
03028     Vnm_tprint( 1, "    tot all -- Total force for system\n");
03029
03030     totforce[0] = 0.0;
03031     totforce[1] = 0.0;
03032     totforce[2] = 0.0;
03033
03034     for (ifr=0; ifr<refnforce; ifr++) {
03035         Vnm_tprint( 1, "    qf  %d %1.12E %1.12E %1.12E\n", ifr,
03036             gforce[ifr].qfForce[0], gforce[ifr].qfForce[1],
03037             gforce[ifr].qfForce[2]);
03038         Vnm_tprint( 1, "    ib  %d %1.12E %1.12E %1.12E\n", ifr,
03039             gforce[ifr].ibForce[0], gforce[ifr].ibForce[1],
03040             gforce[ifr].ibForce[2]);
03041         Vnm_tprint( 1, "    db  %d %1.12E %1.12E %1.12E\n", ifr,
03042             gforce[ifr].dbForce[0], gforce[ifr].dbForce[1],
03043             gforce[ifr].dbForce[2]);
03044         Vnm_tprint( 1, "    tot %d %1.12E %1.12E %1.12E\n", ifr,
03045             (gforce[ifr].dbForce[0] \
03046              + gforce[ifr].ibForce[0] +
03047              gforce[ifr].qfForce[0]),
03048             (gforce[ifr].dbForce[1] \
03049              + gforce[ifr].ibForce[1] +
03050              gforce[ifr].qfForce[1]),
03051             (gforce[ifr].dbForce[2] \
03052              + gforce[ifr].ibForce[2] +
03053              gforce[ifr].qfForce[2]));
03054         for (ivc=0; ivc<3; ivc++) {
03055             totforce[ivc] += (gforce[ifr].dbForce[ivc] \
03056                             + gforce[ifr].ibForce[ivc] \
03057                             + gforce[ifr].qfForce[ivc]);
03058         }
03059     }
03060     Vnm_tprint( 1, "    tot all %1.12E %1.12E %1.12E\n", totforce[0],
03061         totforce[1], totforce[2]);
03062 }
03063
03064 Vmem_free(VNULL, refnforce, sizeof(AtomForce), (void **)(&lforce));
03065 Vmem_free(VNULL, refnforce, sizeof(AtomForce), (void **)(&gforce));
03066
03067 return 1;
03068
03069 }
03070
03071 VPUBLIC int printElecForce(Vcom *com,
03072     Nosh *nosh,
03073     int nforce[NOSH_MAXCALC],
03074     AtomForce *atomForce[NOSH_MAXCALC],
03075     int iprint
03076 ) {
03077
03078     int iarg,
03079     ifr,
03080     ivc,
03081     calcid,
03082     refnforce,
03083     refcalcforce;
03084     double temp,
03085     scalar,
03086     totforce[3];
03087     AtomForce *lforce, *gforce, *aforce;
03088
03089     if (Vstring_strcasecmp(nosh->elecname[nosh->
03090         printcalc[iprint][0]], "") == 0){
03091         Vnm_tprint( 1, "print force %d ", nosh->printcalc[iprint][0]+1);
03092     } else {
03093         Vnm_tprint( 1, "print force %d (%s) ", nosh->printcalc[iprint][0]+1,
03094             nosh->elecname[nosh->printcalc[iprint][0]]);
03095     }
03096     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
03097         if (nosh->printop[iprint][iarg-1] == 0)
03098             Vnm_tprint(1, "+ ");
03099         else if (nosh->printop[iprint][iarg-1] == 1)
03100             Vnm_tprint(1, "- ");
03101         else {
03102             Vnm_tprint( 2, "Undefined PRINT operation!\n");
03103             return 0;
03104         }
03105     }
03106     if (Vstring_strcasecmp(nosh->elecname[nosh->

```

```

    printcalc[iprint][iarg]],
03105         "" == 0) {
03106         Vnm_tprint( 1, "%d ", nosh->printcalc[iprint][iarg]+1);
03107     } else {
03108         Vnm_tprint( 1, "%d (%s) ", nosh->printcalc[iprint][iarg]+1,
03109             nosh->elecname[nosh->printcalc[iprint][iarg]]);
03110     }
03111 }
03112 Vnm_tprint(1, "end\n");
03113
03114 /* First, go through and make sure we did the same type of force
03115    * evaluation in each of the requested calculations */
03116 calcid = nosh->elec2calc[nosh->printcalc[iprint][0]];
03117 refnforce = nforce[calcid];
03118 refcalcforce = nosh->calc[calcid]->pbeparm->calcforce;
03119 if (refcalcforce == PCF_NO) {
03120     Vnm_tprint( 2, " Didn't calculate force in calculation \
03121 #d\n", calcid+1);
03122     return 0;
03123 }
03124 for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
03125     calcid = nosh->elec2calc[nosh->printcalc[iprint][iarg]-1];
03126     if (nosh->calc[calcid]->pbeparm->calcforce != refcalcforce) {
03127         Vnm_tprint(2, " Inconsistent calcforce declarations in \
03128 calculations %d and %d\n", nosh->elec2calc[nosh->printcalc[iprint][0]]+1,
03129             calcid+1);
03130         return 0;
03131     }
03132     if (nforce[calcid] != refnforce) {
03133         Vnm_tprint(2, " Inconsistent number of forces evaluated in \
03134 calculations %d and %d\n", nosh->elec2calc[nosh->printcalc[iprint][0]]+1,
03135             calcid+1);
03136         return 0;
03137     }
03138 }
03139
03140 /* Now, allocate space to accumulate the forces */
03141 lforce = (AtomForce *)Vmem_malloc(VNULL, refnforce, sizeof(
AtomForce));
03142 gforce = (AtomForce *)Vmem_malloc(VNULL, refnforce, sizeof(
AtomForce));
03143
03144 /* Now, accumulate the forces */
03145 calcid = nosh->elec2calc[nosh->printcalc[iprint][0]];
03146 aforce = atomForce[calcid];
03147 temp = nosh->calc[calcid]->pbeparm->temp;
03148
03149 /* Load up the first calculation */
03150 if (refcalcforce == PCF_TOTAL) {
03151     /* Set to total force */
03152     for (ivc=0; ivc<3; ivc++) {
03153         lforce[0].qfForce[ivc] =
03154             Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].qfForce[ivc];
03155         lforce[0].ibForce[ivc] =
03156             Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].ibForce[ivc];
03157         lforce[0].dbForce[ivc] =
03158             Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].dbForce[ivc];
03159     }
03160 } else if (refcalcforce == PCF_COMPS) {
03161     for (ifr=0; ifr<refnforce; ifr++) {
03162         for (ivc=0; ivc<3; ivc++) {
03163             lforce[ifr].qfForce[ivc] =
03164                 Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].qfForce[ivc];
03165             lforce[ifr].ibForce[ivc] =
03166                 Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].ibForce[ivc];
03167             lforce[ifr].dbForce[ivc] =
03168                 Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].dbForce[ivc];
03169         }
03170     }
03171 }
03172
03173 /* Load up the rest of the calculations */
03174 for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
03175     calcid = nosh->elec2calc[nosh->printcalc[iprint][iarg]];
03176     temp = nosh->calc[calcid]->pbeparm->temp;
03177     aforce = atomForce[calcid];
03178     /* Get operation */
03179     if (nosh->printop[iprint][iarg-1] == 0) scalar = +1.0;
03180     else if (nosh->printop[iprint][iarg-1] == 1) scalar = -1.0;
03181     else scalar = 0.0;
03182     /* Accumulate */

```

```

03183         if (refcalcforce == PCF_TOTAL) {
03184             /* Set to total force */
03185             for (ivc=0; ivc<3; ivc++) {
03186                 lforce[0].qfForce[ivc] +=
03187                     (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].qfForce[ivc]);
03188                 lforce[0].ibForce[ivc] +=
03189                     (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].ibForce[ivc]);
03190                 lforce[0].dbForce[ivc] +=
03191                     (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].dbForce[ivc]);
03192             }
03193         } else if (refcalcforce == PCF_COMPS) {
03194             for (ifr=0; ifr<refnforce; ifr++) {
03195                 for (ivc=0; ivc<3; ivc++) {
03196                     lforce[ifr].qfForce[ivc] +=
03197                         (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].qfForce[ivc]);
03198                     lforce[ifr].ibForce[ivc] +=
03199                         (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].ibForce[ivc]);
03200                     lforce[ifr].dbForce[ivc] +=
03201                         (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].dbForce[ivc]);
03202                 }
03203             }
03204         }
03205     }
03206
03207     Vnm_tprint( 0, "printEnergy: Performing global reduction (sum)\n");
03208     for (ifr=0; ifr<refnforce; ifr++) {
03209         Vcom_reduce(com, lforce[ifr].qfForce, gforce[ifr].qfForce, 3, 2, 0);
03210         Vcom_reduce(com, lforce[ifr].ibForce, gforce[ifr].ibForce, 3, 2, 0);
03211         Vcom_reduce(com, lforce[ifr].dbForce, gforce[ifr].dbForce, 3, 2, 0);
03212     }
03213
03214     #if 0
03215     if (refcalcforce == PCF_TOTAL) {
03216         Vnm_tprint( 1, " Local net fixed charge force = \
03217 (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", lforce[0].qfForce[0],
03218 lforce[0].qfForce[1], lforce[0].qfForce[2]);
03219         Vnm_tprint( 1, " Local net ionic boundary force = \
03220 (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", lforce[0].ibForce[0],
03221 lforce[0].ibForce[1], lforce[0].ibForce[2]);
03222         Vnm_tprint( 1, " Local net dielectric boundary force = \
03223 (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", lforce[0].dbForce[0],
03224 lforce[0].dbForce[1], lforce[0].dbForce[2]);
03225     } else if (refcalcforce == PCF_COMPS) {
03226         for (ifr=0; ifr<refnforce; ifr++) {
03227             Vnm_tprint( 1, " Local fixed charge force \
03228 (atom %d) = (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", ifr, lforce[ifr].qfForce[0],
03229 lforce[ifr].qfForce[1], lforce[ifr].qfForce[2]);
03230             Vnm_tprint( 1, " Local ionic boundary force \
03231 (atom %d) = (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", ifr, lforce[ifr].ibForce[0],
03232 lforce[ifr].ibForce[1], lforce[ifr].ibForce[2]);
03233             Vnm_tprint( 1, " Local dielectric boundary force \
03234 (atom %d) = (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", ifr, lforce[ifr].dbForce[0],
03235 lforce[ifr].dbForce[1], lforce[ifr].dbForce[2]);
03236         }
03237     }
03238     #endif
03239
03240     if (refcalcforce == PCF_TOTAL) {
03241         Vnm_tprint( 1, " Printing net forces (kJ/mol/A).\n");
03242         Vnm_tprint( 1, " Legend:\n");
03243         Vnm_tprint( 1, " tot -- Total force\n");
03244         Vnm_tprint( 1, " qf -- Fixed charge force\n");
03245         Vnm_tprint( 1, " db -- Dielectric boundary force\n");
03246         Vnm_tprint( 1, " ib -- Ionic boundary force\n");
03247
03248         for (ivc=0; ivc<3; ivc++) {
03249             totforce[ivc] =
03250                 gforce[0].qfForce[ivc] + gforce[0].ibForce[ivc] \
03251                 + gforce[0].dbForce[ivc];
03252         }
03253
03254         Vnm_tprint( 1, " tot %1.12E %1.12E %1.12E\n", totforce[0],
03255 totforce[1], totforce[2]);
03256         Vnm_tprint( 1, " qf %1.12E %1.12E %1.12E\n", gforce[0].qfForce[0],
03257 gforce[0].qfForce[1], gforce[0].qfForce[2]);
03258         Vnm_tprint( 1, " ib %1.12E %1.12E %1.12E\n", gforce[0].ibForce[0],
03259 gforce[0].ibForce[1], gforce[0].ibForce[2]);
03260         Vnm_tprint( 1, " db %1.12E %1.12E %1.12E\n", gforce[0].dbForce[0],
03261 gforce[0].dbForce[1], gforce[0].dbForce[2]);
03262     } else if (refcalcforce == PCF_COMPS) {

```

```

03264
03265     Vnm_tprint( 1, "   Printing per-atom forces (kJ/mol/A).\n");
03266     Vnm_tprint( 1, "   Legend:\n");
03267     Vnm_tprint( 1, "       tot n -- Total force for atom n\n");
03268     Vnm_tprint( 1, "       qf n -- Fixed charge force for atom n\n");
03269     Vnm_tprint( 1, "       db n -- Dielectric boundary force for atom n\n");
03270     Vnm_tprint( 1, "       ib n -- Ionic boundary force for atom n\n");
03271     Vnm_tprint( 1, "       tot all -- Total force for system\n");
03272
03273     totforce[0] = 0.0;
03274     totforce[1] = 0.0;
03275     totforce[2] = 0.0;
03276
03277     for (ifr=0; ifr<refnforce; ifr++) {
03278         Vnm_tprint( 1, "   qf %d %1.12E %1.12E %1.12E\n", ifr,
03279             gforce[ifr].qfForce[0], gforce[ifr].qfForce[1],
03280             gforce[ifr].qfForce[2]);
03281         Vnm_tprint( 1, "   ib %d %1.12E %1.12E %1.12E\n", ifr,
03282             gforce[ifr].ibForce[0], gforce[ifr].ibForce[1],
03283             gforce[ifr].ibForce[2]);
03284         Vnm_tprint( 1, "   db %d %1.12E %1.12E %1.12E\n", ifr,
03285             gforce[ifr].dbForce[0], gforce[ifr].dbForce[1],
03286             gforce[ifr].dbForce[2]);
03287         Vnm_tprint( 1, "   tot %d %1.12E %1.12E %1.12E\n", ifr,
03288             (gforce[ifr].dbForce[0] \
03289              + gforce[ifr].ibForce[0] +
03290              gforce[ifr].qfForce[0]),
03291             (gforce[ifr].dbForce[1] \
03292              + gforce[ifr].ibForce[1] +
03293              gforce[ifr].qfForce[1]),
03294             (gforce[ifr].dbForce[2] \
03295              + gforce[ifr].ibForce[2] +
03296              gforce[ifr].qfForce[2]));
03297         for (ivc=0; ivc<3; ivc++) {
03298             totforce[ivc] += (gforce[ifr].dbForce[ivc] \
03299                             + gforce[ifr].ibForce[ivc] \
03300                             + gforce[ifr].qfForce[ivc]);
03301         }
03302         Vnm_tprint( 1, "   tot all %1.12E %1.12E %1.12E\n", totforce[0],
03303             totforce[1], totforce[2]);
03304     }
03305
03306     Vmem_free(VNULL, refnforce, sizeof(AtomForce), (void **)(&lforce));
03307     Vmem_free(VNULL, refnforce, sizeof(AtomForce), (void **)(&gforce));
03308
03309     return 1;
03310 }
03311
03312 }
03313
03314 VPUBLIC int printApolForce(Vcom *com,
03315     NOSH *nosh,
03316     int nforce[NOSH_MAXCALC],
03317     AtomForce *atomForce[NOSH_MAXCALC],
03318     int iprint
03319 ) {
03320
03321     int iarg,
03322         ifr,
03323         ivc,
03324         calcid,
03325         refnforce,
03326         refcalcforce;
03327     double temp,
03328         scalar,
03329         totforce[3];
03330     AtomForce *lforce,
03331         *gforce,
03332         *aforce;
03333
03334     if (Vstring_strcasecmp(nosh->apolname, nosh->
03335         printcalc[iprint][0]), "") == 0){
03336         Vnm_tprint( 1, "\nprint APOL force %d ", nosh->printcalc[iprint][0]+1);
03337     } else {
03338         Vnm_tprint( 1, "\nprint APOL force %d (%s) ", nosh->printcalc[iprint][0]+1,
03339             nosh->apolname[nosh->printcalc[iprint][0]]);
03340     }
03341     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
03342         if (nosh->printtop[iprint][iarg-1] == 0)
03343             Vnm_tprint(1, "+ ");
03344         else if (nosh->printtop[iprint][iarg-1] == 1)

```



```

03344     Vnm_tprint(1, "- ");
03345     else {
03346         Vnm_tprint( 2, "Undefined PRINT operation!\n");
03347         return 0;
03348     }
03349     if (Vstring_strcasecmp(nosh->apolname[nosh->
printcalc[iprint][iarg]],
03350         "") == 0) {
03351         Vnm_tprint( 1, "%d ", nosh->printcalc[iprint][iarg]+1);
03352     } else {
03353         Vnm_tprint( 1, "%d (%s) ", nosh->printcalc[iprint][iarg]+1,
03354             nosh->apolname[nosh->printcalc[iprint][iarg]]);
03355     }
03356 }
03357 Vnm_tprint(1, "end\n");
03358
03359 /* First, go through and make sure we did the same type of force
03360    * evaluation in each of the requested calculations */
03361 calcid = nosh->apol2calc[nosh->printcalc[iprint][0]];
03362 refnforce = nforce[calcid];
03363 refcalcf = nosh->calc[calcid]->apolparm->calcf;
03364 if (refcalcf == ACF_NO) {
03365     Vnm_tprint( 2, " Didn't calculate force in calculation \
03366 #d\n", calcid+1);
03367     return 0;
03368 }
03369 for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
03370     calcid = nosh->apol2calc[nosh->printcalc[iprint][iarg]-1];
03371     if (nosh->calc[calcid]->apolparm->calcf != refcalcf) {
03372         Vnm_tprint(2, " Inconsistent calcf declarations in \
03373 calculations %d and %d\n", nosh->apol2calc[nosh->printcalc[iprint][0]]+1,
03374             calcid+1);
03375         return 0;
03376     }
03377     if (nforce[calcid] != refnforce) {
03378         Vnm_tprint(2, " Inconsistent number of forces evaluated in \
03379 calculations %d and %d\n", nosh->apol2calc[nosh->printcalc[iprint][0]]+1,
03380             calcid+1);
03381         return 0;
03382     }
03383 }
03384
03385 /* Now, allocate space to accumulate the forces */
03386 lforce = (AtomForce *)Vmem_malloc(VNULL, refnforce, sizeof(
AtomForce));
03387 gforce = (AtomForce *)Vmem_malloc(VNULL, refnforce, sizeof(
AtomForce));
03388
03389 /* Now, accumulate the forces */
03390 calcid = nosh->apol2calc[nosh->printcalc[iprint][0]];
03391 aforce = atomForce[calcid];
03392 temp = nosh->calc[calcid]->apolparm->temp;
03393
03394 /* Load up the first calculation */
03395 if (refcalcf == ACF_TOTAL) {
03396     /* Set to total force */
03397     for (ivc=0; ivc<3; ivc++) {
03398         lforce[0].sasaForce[ivc] = aforce[0].sasaForce[ivc];
03399         lforce[0].savForce[ivc] = aforce[0].savForce[ivc];
03400         lforce[0].wcaForce[ivc] = aforce[0].wcaForce[ivc];
03401     }
03402 } else if (refcalcf == ACF_COMPS) {
03403     for (ifr=0; ifr<refnforce; ifr++) {
03404         for (ivc=0; ivc<3; ivc++) {
03405             lforce[ifr].sasaForce[ivc] = aforce[ifr].sasaForce[ivc];
03406             lforce[ifr].savForce[ivc] = aforce[ifr].savForce[ivc];
03407             lforce[ifr].wcaForce[ivc] = aforce[ifr].wcaForce[ivc];
03408         }
03409     }
03410 }
03411
03412 /* Load up the rest of the calculations */
03413 for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
03414     calcid = nosh->apol2calc[nosh->printcalc[iprint][iarg]];
03415     temp = nosh->calc[calcid]->apolparm->temp;
03416     aforce = atomForce[calcid];
03417     /* Get operation */
03418     if (nosh->printop[iprint][iarg-1] == 0) scalar = +1.0;
03419     else if (nosh->printop[iprint][iarg-1] == 1) scalar = -1.0;
03420     else scalar = 0.0;
03421     /* Accumulate */

```

```

03422     if (refcalcforce == ACF_TOTAL) {
03423         /* Set to total force */
03424         for (ivc=0; ivc<3; ivc++) {
03425             lforce[0].sasaForce[ivc] += aforce[0].sasaForce[ivc];
03426             lforce[0].savForce[ivc] += aforce[0].savForce[ivc];
03427             lforce[0].wcaForce[ivc] += aforce[0].wcaForce[ivc];
03428         }
03429     } else if (refcalcforce == ACF_COMPS) {
03430         for (ifr=0; ifr<refnforce; ifr++) {
03431             for (ivc=0; ivc<3; ivc++) {
03432                 lforce[ifr].sasaForce[ivc] += aforce[ifr].sasaForce[ivc];
03433                 lforce[ifr].savForce[ivc] += aforce[ifr].savForce[ivc];
03434                 lforce[ifr].wcaForce[ivc] += aforce[ifr].wcaForce[ivc];
03435             }
03436         }
03437     }
03438 }
03439
03440 Vnm_tprint( 0, "printForce: Performing global reduction (sum)\n");
03441 for (ifr=0; ifr<refnforce; ifr++) {
03442     Vcom_reduce(com, lforce[ifr].sasaForce, gforce[ifr].sasaForce, 3, 2, 0);
03443     Vcom_reduce(com, lforce[ifr].savForce, gforce[ifr].savForce, 3, 2, 0);
03444     Vcom_reduce(com, lforce[ifr].wcaForce, gforce[ifr].wcaForce, 3, 2, 0);
03445 }
03446
03447 if (refcalcforce == ACF_TOTAL) {
03448     Vnm_tprint( 1, " Printing net forces (kJ/mol/A)\n");
03449     Vnm_tprint( 1, " Legend:\n");
03450     Vnm_tprint( 1, " tot -- Total force\n");
03451     Vnm_tprint( 1, " sasa -- SASA force\n");
03452     Vnm_tprint( 1, " sav -- SAV force\n");
03453     Vnm_tprint( 1, " wca -- WCA force\n");
03454
03455     for (ivc=0; ivc<3; ivc++) {
03456         totforce[ivc] =
03457             gforce[0].sasaForce[ivc] + gforce[0].savForce[ivc] \
03458             + gforce[0].wcaForce[ivc];
03459     }
03460
03461     Vnm_tprint( 1, " tot %1.12E %1.12E %1.12E\n", totforce[0],
03462                 totforce[1], totforce[2]);
03463     Vnm_tprint( 1, " sasa %1.12E %1.12E %1.12E\n", gforce[0].sasaForce[0],
03464                 gforce[0].sasaForce[1], gforce[0].sasaForce[2]);
03465     Vnm_tprint( 1, " sav %1.12E %1.12E %1.12E\n", gforce[0].savForce[0],
03466                 gforce[0].savForce[1], gforce[0].savForce[2]);
03467     Vnm_tprint( 1, " wca %1.12E %1.12E %1.12E\n", gforce[0].wcaForce[0],
03468                 gforce[0].wcaForce[1], gforce[0].wcaForce[2]);
03469
03470 } else if (refcalcforce == ACF_COMPS) {
03471
03472     Vnm_tprint( 1, " Printing per atom forces (kJ/mol/A)\n");
03473     Vnm_tprint( 1, " Legend:\n");
03474     Vnm_tprint( 1, " tot n -- Total force for atom n\n");
03475     Vnm_tprint( 1, " sasa n -- SASA force for atom n\n");
03476     Vnm_tprint( 1, " sav n -- SAV force for atom n\n");
03477     Vnm_tprint( 1, " wca n -- WCA force for atom n\n");
03478     Vnm_tprint( 1, " tot all -- Total force for system\n");
03479
03480     //Vnm_tprint( 1, " gamma, pressure, bconc are: %f %f %f\n",
03481                 // gamma,press,bconc);
03482
03483     totforce[0] = 0.0;
03484     totforce[1] = 0.0;
03485     totforce[2] = 0.0;
03486
03487     for (ifr=0; ifr<refnforce; ifr++) {
03488         Vnm_tprint( 1, " sasa %d %1.12E %1.12E %1.12E\n", ifr,
03489                     gforce[ifr].sasaForce[0], gforce[ifr].sasaForce[1],
03490                     gforce[ifr].sasaForce[2]);
03491         Vnm_tprint( 1, " sav %d %1.12E %1.12E %1.12E\n", ifr,
03492                     gforce[ifr].savForce[0], gforce[ifr].savForce[1],
03493                     gforce[ifr].savForce[2]);
03494         Vnm_tprint( 1, " wca %d %1.12E %1.12E %1.12E\n", ifr,
03495                     gforce[ifr].wcaForce[0], gforce[ifr].wcaForce[1],
03496                     gforce[ifr].wcaForce[2]);
03497         Vnm_tprint( 1, " tot %d %1.12E %1.12E %1.12E\n", ifr,
03498                     (gforce[ifr].wcaForce[0] \
03499                      + gforce[ifr].savForce[0] +
03500                     gforce[ifr].sasaForce[0]),
03501                     (gforce[ifr].wcaForce[1] \
03502                      + gforce[ifr].savForce[1] +

```

```

03503             gforce[ifr].sasaForce[1]),
03504             (gforce[ifr].wcaForce[2] \
03505              + gforce[ifr].savForce[2] +
03506              gforce[ifr].sasaForce[2]));
03507         for (ivc=0; ivc<3; ivc++) {
03508             totforce[ivc] += (gforce[ifr].wcaForce[ivc] \
03509                              + gforce[ifr].savForce[ivc] \
03510                              + gforce[ifr].sasaForce[ivc]);
03511         }
03512     }
03513     Vnm_tprint( 1, " tot all %1.12E %1.12E %1.12E\n", totforce[0],
03514               totforce[1], totforce[2]);
03515 }
03516
03517 Vmem_free(VNULL, refnforce, sizeof(AtomForce), (void **>(&lforce));
03518 Vmem_free(VNULL, refnforce, sizeof(AtomForce), (void **>(&gforce));
03519
03520 return 1;
03521 }
03522
03523 #ifdef HAVE_MC_H
03524
03525
03526 VPUBLIC void killFE(NOsh *nosh,
03527                   Vpbe *pbe[NOSH_MAXCALC],
03528                   Vfetk *fetk[NOSH_MAXCALC],
03529                   Gem *gm[NOSH_MAXMOL]
03530                 ) {
03531
03532     int i;
03533
03534     #ifndef VAPBSQUIET
03535         Vnm_tprint(1, "Destroying finite element structures.\n");
03536     #endif
03537
03538     for(i=0; i<nosh->ncalc; i++){
03539         Vpbe_dtor(&(pbe[i]));
03540         Vfetk_dtor(&(fetk[i]));
03541     }
03542     for (i=0; i<nosh->nmesh; i++) {
03543         Gem_dtor(&(gm[i]));
03544     }
03545 }
03546
03547 VPUBLIC Vrc_Codes initFE(int icalc,
03548                        NOsh *nosh,
03549                        FEMparm *feparm,
03550                        PBEParm *pbeparm,
03551                        Vpbe *pbe[NOSH_MAXCALC],
03552                        Valist *alist[NOSH_MAXMOL],
03553                        Vfetk *fetk[NOSH_MAXCALC]
03554                      ) {
03555
03556     int iatom,
03557         imesh,
03558         i,
03559         j,
03560         theMol,
03561         focusFlag = 0;
03562     Vio *sock = VNULL;
03563     size_t bytesTotal,
03564           highWater;
03565     Vfetk_MeshLoad meshType;
03566     double length[3],
03567           center[3],
03568           sparm,
03569           q,
03570           iparm = 0.0;
03571     Vrc_Codes vrc;
03572     Valist *myalist;
03573     Vatom *atom = VNULL;
03574     Vnm_tstart(27, "Setup timer");
03575
03576     /* Print some warning messages */
03577     if (pbeparm->useDielMap) Vnm_tprint(2, "FEM ignoring dielectric map!\n");
03578     if (pbeparm->useKappaMap) Vnm_tprint(2, "FEM ignoring kappa map!\n");
03579     if (pbeparm->useChargeMap) Vnm_tprint(2, "FEM ignoring charge map!\n");
03580
03581     /* Fix mesh center for "GCENT MOL #" types of declarations. */
03582     Vnm_tprint(0, "Re-centering mesh...\n");
03583     theMol = pbeparm->molid-1;

```

```

03592     myalist = alist[theMol];
03593     for (j=0; j<3; j++) {
03594         if (theMol < nosh->nmol) {
03595             center[j] = (myalist)->center[j];
03596         } else {
03597             Vnm_tprint(2, "ERROR! Bogus molecule number (%d)!\n",
03598                 (theMol+1));
03599             return VRC_FAILURE;
03600         }
03601     }
03602
03603     /* Check for completely-neutral molecule */
03604     q = 0;
03605     for (iatom=0; iatom<Valist_getNumberAtoms(myalist); iatom++) {
03606         atom = Valist_getAtom(myalist, iatom);
03607         q += VSQR(Vatom_getCharge(atom));
03608     }
03609     /* D. Gohara 10/22/09 - disabled
03610     if (q < (1e-6)) {
03611         Vnm_tprint(2, "Molecule #%d is uncharged!\n", pbeparm->molid);
03612         Vnm_tprint(2, "Sum square charge = %g!\n", q);
03613         return VRC_FAILURE;
03614     }
03615     */
03616
03617     /* Set the feparm pkey value based on the presence of an HB solver */
03618     #ifdef USE_HB
03619         feparm->pkey = 1;
03620     #else
03621         feparm->pkey = 0;
03622     #endif
03623
03624     /* Set up PBE object */
03625     Vnm_tprint(0, "Setting up PBE object...\n");
03626     if (pbeparm->srfm == VSM_SPLINE) {
03627         sparm = pbeparm->swin;
03628     }
03629     else {
03630         sparm = pbeparm->srad;
03631     }
03632     if (pbeparm->nion > 0) {
03633         iparm = pbeparm->ionr[0];
03634     }
03635
03636     pbe[icalc] = Vpbe_ctor(myalist, pbeparm->nion,
03637         pbeparm->ionc, pbeparm->ionr, pbeparm->ionq,
03638         pbeparm->temp, pbeparm->pdie,
03639         pbeparm->sdie, sparm, focusFlag, pbeparm->sdens,
03640         pbeparm->zmem, pbeparm->Lmem, pbeparm->mdie,
03641         pbeparm->memv);
03642
03643     /* Print a few derived parameters */
03644     Vnm_tprint(1, " Debye length: %g A\n", Vpbe_getDeblen(pbe[icalc]));
03645
03646     /* Set up FEtk objects */
03647     Vnm_tprint(0, "Setting up FEtk object...\n");
03648     fetk[icalc] = Vfetk_ctor(pbe[icalc], pbeparm->pbetype);
03649     Vfetk_setParameters(fetk[icalc], pbeparm, feparm);
03650
03651     /* Build mesh - this merely loads an MCSF file from an external source if one is specified or uses the
03652     * current molecule and sets center/length values based on that molecule if no external source is
03653     * specified. */
03654     Vnm_tprint(0, "Setting up mesh...\n");
03655     sock = VNULL;
03656     if (feparm->useMesh) {
03657         imesh = feparm->meshID-1;
03658         meshType = VML_EXTERNAL;
03659         for (i=0; i<3; i++) {
03660             center[i] = 0.0;
03661             length[i] = 0.0;
03662         }
03663         Vnm_print(0, "Using mesh %d (%s) in calculation.\n", imesh+1,
03664             nosh->meshpath[imesh]);
03665         switch (nosh->meshfmt[imesh]) {
03666             case VDF_DX:
03667                 Vnm_tprint(2, "DX finite element mesh input not supported yet!\n");
03668                 return VRC_FAILURE;
03669             case VDF_UHBD:
03670                 Vnm_tprint(2, "UHBD finite element mesh input not supported!\n");
03671                 return VRC_FAILURE;
03672             case VDF_AVS:

```

```

03673         Vnm_tprint( 2, "AVS finite element mesh input not supported!\n");
03674         return VRC_FAILURE;
03675     case VDF_MCSF:
03676         Vnm_tprint(1, "Reading MCSF-format input finite element mesh from %s.\n",
03677                     nosh->meshpath[imesh]);
03678         sock = Vio_ctor("FILE", "ASC", VNULL, nosh->meshpath[imesh], "r");
03679         if (sock == VNULL) {
03680             Vnm_print(2, "Problem opening virtual socket %s!\n",
03681                     nosh->meshpath[imesh]);
03682             return VRC_FAILURE;
03683         }
03684         if (Vio_accept(sock, 0) < 0) {
03685             Vnm_print(2, "Problem accepting virtual socket %s!\n",
03686                     nosh->meshpath[imesh]);
03687             return VRC_FAILURE;
03688         }
03689         break;
03690
03691     default:
03692         Vnm_tprint( 2, "Invalid data format (%d)!\n",
03693                     nosh->meshfmt[imesh]);
03694         return VRC_FAILURE;
03695     }
03696 } else { /* if (feparm->useMesh) */
03697     meshType = VML_DIRICUBE;
03698     for (i=0; i<3; i++) {
03699         center[i] = alist[theMol]->center[i];
03700         length[i] = feparm->glen[i];
03701     }
03702 }
03703
03704 /* Load the mesh with a particular center and vertex length using the provided input socket */
03705 vrc = Vfetc_loadMesh(fetk[icalc], center, length, meshType, sock);
03706 if (vrc == VRC_FAILURE) {
03707     Vnm_print(2, "Error constructing finite element mesh!\n");
03708     return VRC_FAILURE;
03709 }
03710 //Vnm_redirect(0); // Redirect output to io.mc
03711 Gem_shapeChk(fetk[icalc]->gm); // Traverse simplices and check shapes using the geometry manager.
03712 //Vnm_redirect(1);
03713
03714 /* Uniformly refine the mesh a bit */
03715 for (j=0; j<2; j++) {
03716     /* AM_* calls below are from the MC package, mc/src/nam/am.c. Note that these calls actually are
03717      * wrappers around Aprx_* functions found in MC as well. */
03718     /* Mark the mesh for needed refinements */
03719     AM_markRefine(fetk[icalc]->am, 0, -1, 0, 0.0);
03720     /* Do actual mesh refinement */
03721     AM_refine(fetk[icalc]->am, 2, 0, feparm->pkey);
03722     //Vnm_redirect(0); // Redirect output to io.mc
03723     Gem_shapeChk(fetk[icalc]->gm); // Traverse simplices and check shapes using the geometry manager.
03724     //Vnm_redirect(1);
03725 }
03726
03727 /* Setup time statistics */
03728 Vnm_tstop(27, "Setup timer");
03729
03730 /* Memory statistics */
03731 bytesTotal = Vmem_bytesTotal();
03732 highWater = Vmem_highWaterTotal();
03733
03734 #ifndef VAPBSQUIET
03735     Vnm_tprint( 1, " Current memory usage: %4.3f MB total, \
03736 %4.3f MB high water\n", (double)(bytesTotal)/(1024.*1024.),
03737                     (double)(highWater)/(1024.*1024.));
03738 #endif
03739
03740     return VRC_SUCCESS;
03741 }
03742
03743 VPUBLIC void printFEPARM(int icalc,
03744                         Nosh *nosh,
03745                         FEMparm *feparm,
03746                         Vfetc *fetk[NOSH_MAXCALC]
03747                     ) {
03748
03749     Vnm_tprint(1, " Domain size: %g Å x %g Å x %g Å\n",
03750                 feparm->glen[0], feparm->glen[1],
03751                 feparm->glen[2]);
03752     switch(feparm->ekey) {
03753     case FET_SIMP:

```

```

03754         Vnm_tprint(1, " Per-simplex error tolerance:  %g\n", feparm->etol);
03755         break;
03756     case FET_GLOB:
03757         Vnm_tprint(1, " Global error tolerance:  %g\n", feparm->etol);
03758         break;
03759     case FET_FRAC:
03760         Vnm_tprint(1, " Fraction of simps to refine:  %g\n", feparm->etol);
03761         break;
03762     default:
03763         Vnm_tprint(2, "Invalid ekey (%d)!\n", feparm->ekey);
03764         VASSERT(0);
03765         break;
03766 }
03767 switch(feparm->akeyPRE) {
03768     case FRT_UNIF:
03769         Vnm_tprint(1, " Uniform pre-solve refinement.\n");
03770         break;
03771     case FRT_GEOM:
03772         Vnm_tprint(1, " Geometry-based pre-solve refinement.\n");
03773         break;
03774     default:
03775         Vnm_tprint(2, "Invalid akeyPRE (%d)!\n", feparm->akeyPRE);
03776         VASSERT(0);
03777         break;
03778 }
03779 switch(feparm->akeySOLVE) {
03780     case FRT_RESI:
03781         Vnm_tprint(1, " Residual-based a posteriori refinement.\n");
03782         break;
03783     case FRT_DUAL:
03784         Vnm_tprint(1, " Dual-based a posteriori refinement.\n");
03785         break;
03786     case FRT_LOCA:
03787         Vnm_tprint(1, " Local-based a posteriori refinement.\n");
03788         break;
03789     default:
03790         Vnm_tprint(2, "Invalid akeySOLVE (%d)!\n", feparm->akeySOLVE);
03791         break;
03792 }
03793 Vnm_tprint(1, " Refinement of initial mesh to ~%d vertices\n",
03794           feparm->targetNum);
03795 Vnm_tprint(1, " Geometry-based refinement lower bound:  %g A\n",
03796           feparm->targetRes);
03797 Vnm_tprint(1, " Maximum number of solve-estimate-refine cycles:  %d\n",
03798           feparm->maxsolve);
03799 Vnm_tprint(1, " Maximum number of vertices in mesh:  %d\n",
03800           feparm->maxvert);
03801
03802 /* FOLLOWING IS SOLVER-RELATED; BAIL IF NOT SOLVING */
03803 if (nosh->bogus) return;
03804 #ifdef USE_HB
03805     Vnm_tprint(1, " HB linear solver:  AM_hPcg\n");
03806 #else
03807     Vnm_tprint(1, " Non-HB linear solver:  ");
03808     switch (fetc[icalc]->lkey) {
03809         case VLT_SLU:
03810             Vnm_print(1, "SLU direct\n");
03811             break;
03812         case VLT_MG:
03813             Vnm_print(1, "multigrid\n");
03814             break;
03815         case VLT_CG:
03816             Vnm_print(1, "conjugate gradient\n");
03817             break;
03818         case VLT_BCG:
03819             Vnm_print(1, "BiCGStab\n");
03820             break;
03821         default:
03822             Vnm_print(1, "???\n");
03823             break;
03824     }
03825 #endif
03826
03827 Vnm_tprint(1, " Linear solver tol.:  %g\n", fetc[icalc]->ltol);
03828 Vnm_tprint(1, " Linear solver max. iters.:  %d\n", fetc[icalc]->lmax);
03829 Vnm_tprint(1, " Linear solver preconditioner:  ");
03830 switch (fetc[icalc]->lprec) {
03831     case VPT_IDEN:
03832         Vnm_print(1, "identity\n");
03833         break;
03834     case VPT_DIAG:

```

```

03835         Vnm_print(1, "diagonal\n");
03836         break;
03837     case VPT_MG:
03838         Vnm_print(1, "multigrid\n");
03839         break;
03840     default:
03841         Vnm_print(1, "???\n");
03842         break;
03843 }
03844 Vnm_tprint(1, " Nonlinear solver: ");
03845 switch (fetk[icalc]->nkey) {
03846     case VNT_NEW:
03847         Vnm_print(1, "newton\n");
03848         break;
03849     case VNT_INC:
03850         Vnm_print(1, "incremental\n");
03851         break;
03852     case VNT_ARC:
03853         Vnm_print(1, "pseudo-arclength\n");
03854         break;
03855     default:
03856         Vnm_print(1, "?? ");
03857         break;
03858 }
03859 Vnm_tprint(1, " Nonlinear solver tol.: %g\n", fetk[icalc]->ntol);
03860 Vnm_tprint(1, " Nonlinear solver max. iters.: %d\n", fetk[icalc]->nmax);
03861 Vnm_tprint(1, " Initial guess: ");
03862 switch (fetk[icalc]->gues) {
03863     case VGT_ZERO:
03864         Vnm_tprint(1, "zero\n");
03865         break;
03866     case VGT_DIRI:
03867         Vnm_tprint(1, "boundary function\n");
03868         break;
03869     case VGT_PREV:
03870         Vnm_tprint(1, "interpolated previous solution\n");
03871         break;
03872     default:
03873         Vnm_tprint(1, "???\n");
03874         break;
03875 }
03876 }
03877 }
03878
03879 VPUBLIC int partFE(int icalc, NOsh *nosh, FEMparm *feparm,
03880                  Vfetk *fetk[NOSH_MAXCALC]) {
03881
03882     Vfetk_setAtomColors(fetk[icalc]);
03883     return 1;
03884 }
03885
03886 VPUBLIC int preRefineFE(int icalc, /* Calculation index */
03887                      FEMparm *feparm, /* FE-specific parameters */
03888                      Vfetk *fetk[NOSH_MAXCALC] /* Array of FE solver objects */
03889                      ) {
03890
03891     int nverts,
03892         marked;
03893
03894     /* Based on the refinement type, alert the user to what we're trying to refine with. */
03895     switch(feparm->akeyPRE) {
03896     case FRT_UNIF:
03897         Vnm_tprint(1, " Commencing uniform refinement to %d verts.\n",
03898                   feparm->targetNum);
03899         break;
03900     case FRT_GEOM:
03901         Vnm_tprint(1, " Commencing geometry-based refinement to %d \
03902 verts or %g A resolution.\n", feparm->targetNum, feparm->targetRes);
03903         break;
03904     case FRT_DUAL:
03905         Vnm_tprint(2, "What? You can't do a posteriori error estimation \
03906 before you solve!\n");
03907         VASSERT(0);
03908         break;
03909     case FRT_RESI:
03910     case FRT_LOCA:
03911     default:
03912         VASSERT(0);
03913         break;
03914     }
03915 }
03916
03922     Vnm_tprint(1, " Initial mesh has %d vertices\n",

```

```

03923         Gem_numVV(fetk[icalc]->gm));
03924
03925     /* As long as we have simplices marked that need to be refined, run MC's
03926     * AM_markRefine against our data until we hit the error or size tolerance
03927     * for the refinement. */
03928     while (1) {
03929         nverts = Gem_numVV(fetk[icalc]->gm);
03930         if (nverts > feparm->targetNum) {
03931             Vnm_tprint(1, " Hit vertex number limit.\n");
03932             break;
03933         }
03934         marked = AM_markRefine(fetk[icalc]->am, feparm->akeyPRE, -1,
03935                               feparm->ekey, feparm->etol);
03936         if (marked == 0) {
03937             Vnm_tprint(1, " Marked 0 simp; hit error/size tolerance.\n");
03938             break;
03939         }
03940         Vnm_tprint(1, " Have %d verts, marked %d. Refining...\n", nverts,
03941                   marked);
03942         AM_refine(fetk[icalc]->am, 0, 0, feparm->pkey);
03943     }
03944     nverts = Gem_numVV(fetk[icalc]->gm);
03945     Vnm_tprint(1, " Done refining; have %d verts.\n", nverts);
03946
03947     return 1;
03948 }
03949
03950
03951
03956 VPUBLIC int solveFE(int icalc,
03957                      PBEparm *pbeparm,
03958                      FEMparm *feparm,
03959                      Vfetk *fetk[NOSH_MAXCALC]
03960                      ) {
03961
03962     int lkeyHB = 3,
03963         meth = 2,
03964         prob = 0,
03965         prec = 0;
03966     if ((pbeparm->pbetype==PBE_NPBE) ||
03967         (pbeparm->pbetype == PBE_NRPBE) ||
03968         (pbeparm->pbetype == PBE_SMPBE)) {
03969
03970
03971         /* Call MC's nonlinear solver - mc/src/nam/nsolv.c */
03972         AM_nSolve(
03973             fetk[icalc]->am,
03974             fetk[icalc]->nkey,
03975             fetk[icalc]->nmax,
03976             fetk[icalc]->ntol,
03977             meth,
03978             fetk[icalc]->lmax,
03979             fetk[icalc]->ltol,
03980             prec,
03981             fetk[icalc]->gues,
03982             fetk[icalc]->pjac
03983         );
03984     } else if ((pbeparm->pbetype==PBE_LPBE) ||
03985               (pbeparm->pbetype==PBE_LRPBE)) {
03986         /* Note: USEHB is a compile time defined macro. The program flow
03987         is to always take the route using AM_hlSolve when the solver
03988         is linear. D. Gohara 6/29/06
03989         */
03990         #ifdef USE_HB
03991             Vnm_print(2, "SORRY! DON'T USE HB!!!\n");
03992             VASSERT(0);
03993
03994             /* Call MC's hierarchical linear solver - mc/src/nam/lsolv.c */
03995             AM_hlSolve(fetk[icalc]->am, meth, lkeyHB, fetk[icalc]->lmax,
03996                       fetk[icalc]->ltol, fetk[icalc]->gues, fetk[icalc]->pjac);
03997         #else
03998
03999             /* Call MC's linear solver - mc/src/nam/lsolv.c */
04000             AM_lSolve(
04001                 fetk[icalc]->am,
04002                 prob,
04003                 meth,
04004                 fetk[icalc]->lmax,
04005                 fetk[icalc]->ltol,
04006                 prec,
04007                 fetk[icalc]->gues,
04008                 fetk[icalc]->pjac

```



```

04009         );
04010 #endif
04011     }
04012
04013     return 1;
04014 }
04015
04019 VPUBLIC int energyFE(Nosh *nosh,
04020                     int icalc,
04021                     Vfetk *fetk[NOSH_MAXCALC],
04022                     int *nenergy,
04023                     double *totEnergy,
04024                     double *qfEnergy,
04025                     double *qmEnergy,
04026                     double *dielEnergy
04027                 ) {
04028
04029     FEMparm *feparm = nosh->calc[icalc]->feparm;
04030     PBEParm *pbeparm = nosh->calc[icalc]->pbeparm;
04031     *nenergy = 1;
04032     *totEnergy = 0;
04033
04034     if (nosh->bogus == 0) {
04041         if ((pbeparm->pbetype==PBE_NPBE) ||
04042             (pbeparm->pbetype==PBE_NRPBE) ||
04043             (pbeparm->pbetype == PBE_SMPBE)) {
04044             *totEnergy = Vfetk_energy(fetk[icalc], -1, 1); /* Last parameter indicates NPBE */
04045         } else if ((pbeparm->pbetype==PBE_LPBE) ||
04046                    (pbeparm->pbetype==PBE_LRPBE)) {
04047             *totEnergy = Vfetk_energy(fetk[icalc], -1, 0); /* Last parameter indicates LPBE */
04048         } else {
04049             VASSERT(0);
04050         }
04051     }
04052
04053 #ifndef VAPBSQUIET
04054     Vnm_tprint(1, "          Total electrostatic energy = %1.12E kJ/mol\n",
04055                Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*( *totEnergy));
04056     fflush(stdout);
04057 #endif
04058 }
04059
04060     if (pbeparm->calcenergy == PCE_COMPS) {
04061         Vnm_tprint(2, "Error! Verbose energy evaluation not available for FEM yet!\n");
04062         Vnm_tprint(2, "E-mail nathan.baker@pnl.gov if you want this.\n");
04063         *qfEnergy = 0;
04064         *qmEnergy = 0;
04065         *dielEnergy = 0;
04066     } else {
04067         *nenergy = 0;
04068     }
04069     return 1;
04070 }
04071
04079 VPUBLIC int postRefineFE(int icalc,
04080                         FEMparm *feparm,
04081                         Vfetk *fetk[NOSH_MAXCALC]
04082                     ) {
04083
04084     int nverts,
04085         marked;
04086     nverts = Gem_numVV(fetk[icalc]->gm);
04087     if (nverts > feparm->maxvert) {
04088         Vnm_tprint(1, "          Current number of vertices (%d) exceeds max (%d)!\n",
04089                    nverts, feparm->maxvert);
04090         return 0;
04091     }
04092     Vnm_tprint(1, "          Mesh currently has %d vertices\n", nverts);
04093
04094     switch(feparm->akeySOLVE) {
04095     case FRT_UNIF:
04096         Vnm_tprint(1, "          Commencing uniform refinement.\n");
04097         break;
04098     case FRT_GEOM:
04099         Vnm_tprint(1, "          Commencing geometry-based refinement.\n");
04100         break;
04101     case FRT_RESI:
04102         Vnm_tprint(1, "          Commencing residual-based refinement.\n");
04103         break;
04104     case FRT_DUAL:
04105         Vnm_tprint(1, "          Commencing dual problem-based refinement.\n");
04106         break;
04107     }

```

```

04108         case FRT_LOCA:
04109             Vnm_tprint(1, "      Commencing local-based refinement.\n");
04110             break;
04111         default:
04112             Vnm_tprint(2, "      Error -- unknown refinement type (%d)!\n",
04113                 feparm->akeySOLVE);
04114             return 0;
04115             break;
04116     }
04117
04118     /* Run MC's refinement algorithm */
04119     marked = AM_markRefine(fetk[icalc]->am, feparm->akeySOLVE, -1,
04120         feparm->ekey, feparm->etol);
04121
04122     if (marked == 0) {
04123         Vnm_tprint(1, "      Marked 0 simpes; hit error/size tolerance.\n");
04124         return 0;
04125     }
04126     Vnm_tprint(1, "      Have %d verts, marked %d. Refining...\n", nverts,
04127         marked);
04128     AM_refine(fetk[icalc]->am, 0, 0, feparm->pkey);
04129     nverts = Gem_numVV(fetk[icalc]->gm);
04130     Vnm_tprint(1, "      Done refining; have %d verts.\n", nverts);
04131     //Vnm_redirect(0); // Redirect output to io.mc
04132     Gem_shapeChk(fetk[icalc]->gm); // Traverse simplices and check shapes using the geometry manager.
04133     //Vnm_redirect(1);
04134
04135     return 1;
04136 }
04137
04140 VPUBLIC int writedataFE(int rank,
04141     NOSH *nosh,
04142     PBEPARM *pbeparm,
04143     VFETK *fetk
04144 ) {
04145
04146     char writestem[VMAX_ARGLEN];
04147     char outpath[VMAX_ARGLEN];
04148     int i,
04149         writeit;
04150     AM *am;
04151     Bvec *vec;
04152     if (nosh->bogus) return 1;
04153
04154     am = fetk->am;
04155     vec = am->w0;
04156
04157     for (i=0; i<pbeparm->numwrite; i++) {
04158
04159         writeit = 1;
04160
04161         switch (pbeparm->writetype[i]) {
04162
04163             case VDT_CHARGE:
04164
04165                 Vnm_tprint(2, "      Sorry; can't write charge distribution for FEM!\n");
04166                 writeit = 0;
04167                 break;
04168
04169             case VDT_POT:
04170
04171                 Vnm_tprint(1, "      Writing potential to ");
04172                 Vfetk_fillArray(fetk, vec, VDT_POT);
04173                 break;
04174
04175             case VDT_SMOL:
04176
04177                 Vnm_tprint(1, "      Writing molecular accessibility to ");
04178                 Vfetk_fillArray(fetk, vec, VDT_SMOL);
04179                 break;
04180
04181             case VDT_SSPL:
04182
04183                 Vnm_tprint(1, "      Writing spline-based accessibility to ");
04184                 Vfetk_fillArray(fetk, vec, VDT_SSPL);
04185                 break;
04186
04187             case VDT_VDW:
04188
04189                 Vnm_tprint(1, "      Writing van der Waals accessibility to ");
04190                 Vfetk_fillArray(fetk, vec, VDT_VDW);
04191                 break;
04192

```

```

04193
04194     case VDT_IVDW:
04195         Vnm_tprint(1, "    Writing ion accessibility to ");
04196         Vfetk_fillArray(fetk, vec, VDT_IVDW);
04197         break;
04198
04199     case VDT_LAP:
04200
04201         Vnm_tprint(2, "    Sorry; can't write charge distribution for FEM!\n");
04202         writeit = 0;
04203         break;
04204
04205     case VDT_EDENS:
04206
04207         Vnm_tprint(2, "    Sorry; can't write energy density for FEM!\n");
04208         writeit = 0;
04209         break;
04210
04211     case VDT_NDENS:
04212
04213         Vnm_tprint(1, "    Writing number density to ");
04214         Vfetk_fillArray(fetk, vec, VDT_NDENS);
04215         break;
04216
04217     case VDT_QDENS:
04218
04219         Vnm_tprint(1, "    Writing charge density to ");
04220         Vfetk_fillArray(fetk, vec, VDT_QDENS);
04221         break;
04222
04223     case VDT_DIELX:
04224
04225         Vnm_tprint(2, "    Sorry; can't write x-shifted dielectric map for FEM!\n");
04226         writeit = 0;
04227         break;
04228
04229     case VDT_DIELY:
04230
04231         Vnm_tprint(2, "    Sorry; can't write y-shifted dielectric map for FEM!\n");
04232         writeit = 0;
04233         break;
04234
04235     case VDT_DIELZ:
04236
04237         Vnm_tprint(2, "    Sorry; can't write z-shifted dielectric map for FEM!\n");
04238         writeit = 0;
04239         break;
04240
04241     case VDT_KAPPA:
04242
04243         Vnm_tprint(1, "    Sorry; can't write kappa map for FEM!\n");
04244         writeit = 0;
04245         break;
04246
04247     case VDT_ATOMPOT:
04248
04249         Vnm_tprint(1, "    Sorry; can't write atom potentials for FEM!\n");
04250         writeit = 0;
04251         break;
04252
04253     default:
04254
04255         Vnm_tprint(2, "Invalid data type for writing!\n");
04256         writeit = 0;
04257         return 0;
04258 }
04259
04260 if (!writeit) return 0;
04261
04262 #ifdef HAVE_MPI_H
04263     sprintf(writestem, "%s-PE%d", pbeparm->writestem[i], rank);
04264 #else
04265     if (nosh->ispara) {
04266         sprintf(writestem, "%s-PE%d", pbeparm->writestem[i], nosh->
04267             proc_rank);
04268     } else {
04269         sprintf(writestem, "%s", pbeparm->writestem[i]);
04270     }
04271 #endif
04272

```

```

04273
04274     switch (pbeparm->writefmt[i]) {
04275
04276         case VDF_DX:
04277             sprintf(outpath, "%.5s", writestem, "dx");
04278             Vnm_tprint(1, "%s\n", outpath);
04279             Vfetc_write(fetc, "FILE", "ASC", VNULL, outpath, vec,
VDF_DX);
04280             break;
04281
04282         case VDF_AVS:
04283             sprintf(outpath, "%.5s", writestem, "ucd");
04284             Vnm_tprint(1, "%s\n", outpath);
04285             Vfetc_write(fetc, "FILE", "ASC", VNULL, outpath, vec,
VDF_AVS);
04286             break;
04287
04288         case VDF_UHBD:
04289             Vnm_tprint(2, "UHBD format not supported for FEM!\n");
04290             break;
04291
04292         case VDF_MCSF:
04293             Vnm_tprint(2, "MCSF format not supported yet!\n");
04294             break;
04295
04296         default:
04297             Vnm_tprint(2, "Bogus data format (%d)!\n",
pbeparm->writefmt[i]);
04298             break;
04299     }
04300 }
04301 }
04302 }
04303
04304     return 1;
04305 }
04306 #endif /* ifdef HAVE_MCX_H */
04307
04308 VPUBLIC int initAPOL(Nosh *nosh,
04309                    Vmem *mem,
04310                    Vparam *param,
04311                    APOLparam *apolparam,
04312                    int *nforce,
04313                    AtomForce **atomForce,
04314                    Valist *alist
04315                ) {
04316     int i,
04317         natoms,
04318         len,
04319         inhash[3],
04320         rc = 0;
04321     time_t ts;
04322     Vclist *clist = VNULL;
04323     Vacc *acc = VNULL;
04324     Vatom *atom = VNULL;
04325     Vparam_AtomData *atomData = VNULL;
04326     double sasa,
04327         sav,
04328         nhash[3],
04329         sradsPad,
04330         x,
04331         y,
04332         z,
04333         atomRadius,
04334         srads,
04335         *atomsasa,
04336         *atomwcaEnergy,
04337         energy = 0.0,
04338         dist,
04339         charge,
04340         xmin,
04341         xmax,
04342         ymin,
04343         ymax,
04344         zmin,
04345         zmax,
04346         disp[3],
04347         center[3],
04348         soluteXlen,
04349         soluteYlen,
04350         soluteZlen;
04351     atomsasa = (double *)Vmem_malloc(VNULL, Valist_getNumberAtoms(alist), sizeof(

```

```

double));
04355     atomwcaEnergy = (double *)Vmem_malloc(VNULL, Valist_getNumberAtoms(alist), sizeof(
double));
04356
04357     /* Determine solute length and charge*/
04358     atom = Valist_getAtom(alist, 0);
04359     xmin = Vatom_getPosition(atom)[0];
04360     xmax = Vatom_getPosition(atom)[0];
04361     ymin = Vatom_getPosition(atom)[1];
04362     ymax = Vatom_getPosition(atom)[1];
04363     zmin = Vatom_getPosition(atom)[2];
04364     zmax = Vatom_getPosition(atom)[2];
04365     charge = 0;
04366     natoms = Valist_getNumberAtoms(alist);
04367
04368     for (i=0; i < natoms; i++) {
04369         atom = Valist_getAtom(alist, i);
04370         atomRadius = Vatom_getRadius(atom);
04371         x = Vatom_getPosition(atom)[0];
04372         y = Vatom_getPosition(atom)[1];
04373         z = Vatom_getPosition(atom)[2];
04374         if ((x+atomRadius) > xmax) xmax = x + atomRadius;
04375         if ((x-atomRadius) < xmin) xmin = x - atomRadius;
04376         if ((y+atomRadius) > ymax) ymax = y + atomRadius;
04377         if ((y-atomRadius) < ymin) ymin = y - atomRadius;
04378         if ((z+atomRadius) > zmax) zmax = z + atomRadius;
04379         if ((z-atomRadius) < zmin) zmin = z - atomRadius;
04380         disp[0] = (x - center[0]);
04381         disp[1] = (y - center[1]);
04382         disp[2] = (z - center[2]);
04383         dist = (disp[0]*disp[0]) + (disp[1]*disp[1]) + (disp[2]*disp[2]);
04384         dist = VSQRT(dist) + atomRadius;
04385         charge += Vatom_getCharge(Valist_getAtom(alist, i));
04386     }
04387     soluteXlen = xmax - xmin;
04388     soluteYlen = ymax - ymin;
04389     soluteZlen = zmax - zmin;
04390
04391     /* Set up the hash table for the cell list */
04392     Vnm_print(0, "APOL: Setting up hash table and accessibility object...\n");
04393     nhash[0] = soluteXlen/0.5;
04394     nhash[1] = soluteYlen/0.5;
04395     nhash[2] = soluteZlen/0.5;
04396     for (i=0; i<3; i++) inhash[i] = (int) (nhash[i]);
04397
04398     for (i=0; i<3; i++){
04399         if (inhash[i] < 3) inhash[i] = 3;
04400         if (inhash[i] > MAX_HASH_DIM) inhash[i] = MAX_HASH_DIM;
04401     }
04402
04403     /* Pad the radius by 2x the maximum displacement value */
04404     srad = apolparm->srad;
04405     sradPad = srad + (2*apolparm->dpos);
04406     clist = Vclist_ctor(alist, sradPad, inhash, CLIST_AUTO_DOMAIN,
04407                        VNULL, VNULL);
04408     acc = Vacc_ctor(alist, clist, apolparm->sdens);
04409
04410     /* Get WAT (water) LJ parameters from Vparam object */
04411     if (param == VNULL && (apolparm->bconc != 0.0)) {
04412         Vnm_tprint(2, "initAPOL: Got NULL Vparam object!\n");
04413         Vnm_tprint(2, "initAPOL: You are performing an apolar calculation with the van der Waals integral
term,\n");
04414         Vnm_tprint(2, "initAPOL: this term requires van der Waals parameters which are not available from
the \n");
04415         Vnm_tprint(2, "initAPOL: PQR file. Therefore, you need to supply a parameter file with the parm
keyword,\n");
04416         Vnm_tprint(2, "initAPOL: for example,\n");
04417         Vnm_tprint(2, "initAPOL: read parm flat amber94.dat end\n");
04418         Vnm_tprint(2, "initAPOL: where the relevant parameter files can be found in
apbs/tools/conversion/param/vparam.\n");
04419         return VRC_FAILURE;
04420     }
04421
04422     if (apolparm->bconc != 0.0){
04423         atomData = Vparam_getAtomData(param, "WAT", "OW");
04424         if (atomData == VNULL) atomData = Vparam_getAtomData(param, "WAT", "O");
04425         if (atomData == VNULL) {
04426             Vnm_tprint(2, "initAPOL: Couldn't find parameters for WAT OW or WAT O!\n");
04427             Vnm_tprint(2, "initAPOL: These parameters must be present in your file\n");
04428             Vnm_tprint(2, "initAPOL: for apolar calculations.\n");
04429             return VRC_FAILURE;

```

```

04430     }
04431     apolparm->watepsilon = atomData->epsilon;
04432     apolparm->watsigma = atomData->radius;
04433     apolparm->setwat = 1;
04434 }
04435
04436 /* Calculate Energy and Forces */
04437 if (apolparm->calcforce) {
04438     rc = forceAPOL(acc, mem, apolparm, nforce, atomForce, alist, clist);
04439     if (rc == VRC_FAILURE) {
04440         Vnm_print(2, "Error in apolar force calculation!\n");
04441         return VRC_FAILURE;
04442     }
04443 }
04444
04445 /* Get the SAV and SAS */
04446 sasa = 0.0;
04447 sav = 0.0;
04448
04449 if (apolparm->calcenergy) {
04450     len = Valist_getNumberAtoms(alist);
04451
04452     if (VABS(apolparm->gamma) > VSMALL) {
04453         /* Total Solvent Accessible Surface Area (SASA) */
04454         apolparm->sasa = Vacc_totalSASA(acc, srاد);
04455         /* SASA for each atom */
04456         for (i = 0; i < len; i++) {
04457             atom = Valist_getAtom(alist, i);
04458             atomsasa[i] = Vacc_atomSASA(acc, srاد, atom);
04459         }
04460     } else {
04461         /* Total Solvent Accessible Surface Area (SASA) set to zero */
04462         apolparm->sasa = 0.0;
04463         /* SASA for each atom set to zero */
04464         for (i = 0; i < len; i++) {
04465             atomsasa[i] = 0.0;
04466         }
04467     }
04468
04469     /* Inflated van der Waals accessibility */
04470     if (VABS(apolparm->press) > VSMALL) {
04471         apolparm->sav = Vacc_totalSAV(acc, clist, apolparm, srاد);
04472     } else {
04473         apolparm->sav = 0.0;
04474     }
04475
04476     /* wcaEnergy integral code */
04477     if (VABS(apolparm->bconc) > VSMALL) {
04478         /* wcaEnergy for each atom */
04479         for (i = 0; i < len; i++) {
04480             rc = Vacc_wcaEnergyAtom(acc, apolparm, alist, clist, i, &energy);
04481             if (rc == 0) {
04482                 Vnm_print(2, "Error in apolar energy calculation!\n");
04483                 return 0;
04484             }
04485             atomwcaEnergy[i] = energy;
04486         }
04487         /* Total WCA Energy */
04488         rc = Vacc_wcaEnergy(acc, apolparm, alist, clist);
04489         if (rc == 0) {
04490             Vnm_print(2, "Error in apolar energy calculation!\n");
04491             return 0;
04492         }
04493     } else {
04494         apolparm->wcaEnergy = 0.0;
04495     }
04496     energyAPOL(apolparm, apolparm->sasa, apolparm->sav, atomsasa, atomwcaEnergy,
04497 Valist_getNumberAtoms(alist));
04498
04499     Vmem_free(VNULL, Valist_getNumberAtoms(alist), sizeof(double), (void **)&(atomsasa));
04500     Vmem_free(VNULL, Valist_getNumberAtoms(alist), sizeof(double), (void **)&(atomwcaEnergy));
04501     Vclist_dtor(&clist);
04502     Vacc_dtor(&acc);
04503
04504     return VRC_SUCCESS;
04505 }
04506
04507 VPUBLIC int energyAPOL(APOLparm *apolparm,

```

```

04508             double sasa,
04509             double sav,
04510             double atomsasa[],
04511             double atomwcaEnergy[],
04512             int numatoms
04513         ){
04514
04515         double energy = 0.0;
04516         int i = 0;
04517
04518         #ifndef VAPBSQUIET
04519             Vnm_print(1, "\nSolvent Accessible Surface Area (SASA) for each atom:\n");
04520             for (i = 0; i < numatoms; i++) {
04521                 Vnm_print(1, "  SASA for atom %i: %1.12E\n", i, atomsasa[i]);
04522             }
04523
04524             Vnm_print(1, "\nTotal solvent accessible surface area: %g A^2\n", sasa);
04525         #endif
04526
04527         switch(apolparm->calcenergy){
04528             case ACE_NO:
04529                 break;
04530             case ACE_COMPS:
04531                 Vnm_print(1, "energyAPOL: Cannot calculate component energy, skipping.\n");
04532                 break;
04533             case ACE_TOTAL:
04534                 energy = (apolparm->gamma*sasa) + (apolparm->press*sav)
04535                     + (apolparm->wcaEnergy);
04536         #ifndef VAPBSQUIET
04537             Vnm_print(1, "\nSurface tension*area energies (gamma * SASA) for each atom:\n");
04538             for (i = 0; i < numatoms; i++) {
04539                 Vnm_print(1, "  Surface tension*area energy for atom %i: %1.12E\n", i, apolparm->
04540                     gamma*atomsasa[i]);
04541             }
04542             Vnm_print(1, "\nTotal surface tension energy: %g kJ/mol\n", apolparm->
04543                 gamma*sasa);
04544             Vnm_print(1, "\nTotal solvent accessible volume: %g A^3\n", sav);
04545             Vnm_print(1, "\nTotal pressure*volume energy: %g kJ/mol\n", apolparm->
04546                 press*sav);
04547             Vnm_print(1, "\nWCA dispersion Energies for each atom:\n");
04548             for (i = 0; i < numatoms; i++) {
04549                 Vnm_print(1, "  WCA energy for atom %i: %1.12E\n", i, atomwcaEnergy[i]);
04550             }
04551             Vnm_print(1, "\nTotal WCA energy: %g kJ/mol\n", (apolparm->wcaEnergy));
04552             Vnm_print(1, "\nTotal non-polar energy = %1.12E kJ/mol\n", energy);
04553         #endif
04554             break;
04555             default:
04556                 Vnm_print(2, "energyAPOL: Error in energyAPOL. Unknown option.\n");
04557                 break;
04558         }
04559         return VRC_SUCCESS;
04560     }
04561
04562     VPUBLIC int forceAPOL(Vacc *acc,
04563                         Vmem *mem,
04564                         APOLparm *apolparm,
04565                         int *nforce,
04566                         AtomForce **atomForce,
04567                         Valist *alist,
04568                         Vclist *clist
04569                     ) {
04570         time_t ts, ts_main, ts_sub;
04571
04572         int i,
04573             j,
04574             natom;
04575
04576         double srad, /* Probe radius */
04577             xF,
04578             yF,
04579             zF, /* Individual forces */
04580             press,
04581             gamma,
04582             offset,
04583             bconc,
04584             dSASA[3],
04585             dSAV[3],
04586             force[3],

```

```

04586         *apos;
04587
04588     Vatom *atom = VNULL;
04589     ts_main = clock();
04590
04591     srad = apolparm->srad;
04592     press = apolparm->press;
04593     gamma = apolparm->gamma;
04594     offset = apolparm->dpos;
04595     bconc = apolparm->bconc;
04596
04597     natom = Valist_getNumberAtoms(alist);
04598
04599     /* Check to see if we need to build the surface */
04600     Vnm_print(0, "forceAPOL: Trying atom surf...\n");
04601     ts = clock();
04602     if (acc->surf == VNULL) {
04603         acc->surf = (VaccSurf**)Vmem_malloc(acc->mem, natom, sizeof(
VaccSurf *));
04604         for (i=0; i<natom; i++) {
04605             atom = Valist_getAtom(acc->alist, i);
04606             //apos = Vatom_getPosition(atom); // apos never referenced? - Peter
04607             /* NOTE: RIGHT NOW WE DO THIS FOR THE ENTIRE MOLECULE WHICH IS
04608              * INCREDIBLY INEFFICIENT, PARTICULARLY DURING FOCUSING!!! */
04609             acc->surf[i] = Vacc_atomSurf(acc, atom, acc->
refSphere, srad);
04610         }
04611     }
04612     Vnm_print(0, "forceAPOL: atom surf: Time elapsed: %f\n", ((double)clock() - ts) / CLOCKS_PER_SEC);
04613
04614     if (apolparm->calcforce == ACF_TOTAL) {
04615         Vnm_print(0, "forceAPOL: calcforce == ACF_TOTAL\n");
04616         ts = clock();
04617
04618         *nforce = 1;
04619         if (*atomForce != VNULL) {
04620             Vmem_free(mem, *nforce, sizeof(AtomForce), (void **)atomForce);
04621         }
04622
04623         *atomForce = (AtomForce *)Vmem_malloc(mem, *nforce,
sizeof(AtomForce));
04624
04625         /* Clear out force arrays */
04626         for (j=0; j<3; j++) {
04627             (*atomForce)[0].sasaForce[j] = 0.0;
04628             (*atomForce)[0].savForce[j] = 0.0;
04629             (*atomForce)[0].wcaForce[j] = 0.0;
04630         }
04631
04632         // problem block
04633         for (i=0; i<natom; i++) {
04634             atom = Valist_getAtom(alist, i);
04635
04636             for (j=0; j<3; j++) {
04637                 dSASA[j] = 0.0;
04638                 dSAV[j] = 0.0;
04639                 force[j] = 0.0;
04640             }
04641
04642             if (VABS(gamma) > VSMALL) {
04643                 Vacc_atomdSASA(acc, offset, srad, atom, dSASA);
04644             }
04645             if (VABS(press) > VSMALL) {
04646                 Vacc_atomdSAV(acc, srad, atom, dSAV);
04647             }
04648             if (VABS(bconc) > VSMALL) {
04649                 Vacc_wcaForceAtom(acc, apolparm, clist, atom, force);
04650             }
04651
04652             for (j=0; j<3; j++) {
04653                 (*atomForce)[0].sasaForce[j] += dSASA[j];
04654                 (*atomForce)[0].savForce[j] += dSAV[j];
04655                 (*atomForce)[0].wcaForce[j] += force[j];
04656             }
04657         }
04658         // end block
04659
04660         Vnm_tprint( 1, " Printing net forces (kJ/mol/A)\n");
04661         Vnm_tprint( 1, " Legend:\n");
04662         Vnm_tprint( 1, "      sasa -- SASA force\n");
04663         Vnm_tprint( 1, "      sav  -- SAV force\n");

```



```

04665     Vnm_tprint( 1, "    wca  -- WCA force\n\n");
04666
04667     Vnm_tprint( 1, "    sasa %4.3e %4.3e %4.3e\n",
04668               (*atomForce)[0].sasaForce[0],
04669               (*atomForce)[0].sasaForce[1],
04670               (*atomForce)[0].sasaForce[2]);
04671     Vnm_tprint( 1, "    sav  %4.3e %4.3e %4.3e\n",
04672               (*atomForce)[0].savForce[0],
04673               (*atomForce)[0].savForce[1],
04674               (*atomForce)[0].savForce[2]);
04675     Vnm_tprint( 1, "    wca  %4.3e %4.3e %4.3e\n",
04676               (*atomForce)[0].wcaForce[0],
04677               (*atomForce)[0].wcaForce[1],
04678               (*atomForce)[0].wcaForce[2]);
04679
04680     Vnm_print(0, "forceAPOL: calcforce == ACF_TOTAL: %f\n", ((double)clock() - ts) / CLOCKS_PER_SEC);
04681 } else if (apolparm->calcforce == ACF_COMPS ) {
04682     *nforce = Valist_getNumberAtoms(alist);
04683     if(*atomForce == VNULL){
04684         *atomForce = (AtomForce *)Vmem_malloc(mem, *nforce,
04685                                               sizeof(AtomForce));
04686     }else{
04687         Vmem_free(mem,*nforce,sizeof(AtomForce), (void **)atomForce);
04688         *atomForce = (AtomForce *)Vmem_malloc(mem, *nforce,
04689                                               sizeof(AtomForce));
04690     }
04691
04692 #ifndef VAPBSQUIET
04693     Vnm_tprint( 1, "    Printing per atom forces (kJ/mol/A)\n");
04694     Vnm_tprint( 1, "    Legend:\n");
04695     Vnm_tprint( 1, "        tot  n -- Total force for atom n\n");
04696     Vnm_tprint( 1, "        sasa n -- SASA force for atom n\n");
04697     Vnm_tprint( 1, "        sav  n -- SAV force for atom n\n");
04698     Vnm_tprint( 1, "        wca  n -- WCA force for atom n\n");
04699
04700     Vnm_tprint( 1, "        gamma    %f\n" \
04701               "        pressure %f\n" \
04702               "        bconc    %f\n",
04703               gamma,press,bconc);
04704 #endif
04705
04706     for (i=0; i<natom; i++) {
04707         atom = Valist_getAtom(alist, i);
04708
04709         for(j=0; j<3; j++){
04710             dSASA[j] = 0.0;
04711             dSAV[j] = 0.0;
04712             force[j] = 0.0;
04713         }
04714
04715         /* Clear out force arrays */
04716         for (j=0; j<3; j++) {
04717             (*atomForce)[i].sasaForce[j] = 0.0;
04718             (*atomForce)[i].savForce[j] = 0.0;
04719             (*atomForce)[i].wcaForce[j] = 0.0;
04720         }
04721
04722         if (VABS(gamma) > VSMALL) Vacc_atomdSASA(acc, offset, srاد, atom, dSASA);
04723         if (VABS(press) > VSMALL) Vacc_atomdSAV(acc, srاد, atom, dSAV);
04724         if (VABS(bconc) > VSMALL) Vacc_wcaForceAtom(acc,apolparm,clist,atom,force);
04725
04726         xF = -((gamma*dSASA[0]) + (press*dSAV[0]) + (bconc*force[0]));
04727         yF = -((gamma*dSASA[1]) + (press*dSAV[1]) + (bconc*force[1]));
04728         zF = -((gamma*dSASA[2]) + (press*dSAV[2]) + (bconc*force[2]));
04729
04730         for(j=0; j<3; j++){
04731             (*atomForce)[i].sasaForce[j] += dSASA[j];
04732             (*atomForce)[i].savForce[j] += dSAV[j];
04733             (*atomForce)[i].wcaForce[j] += force[j];
04734         }
04735
04736 #ifndef VAPBSQUIET
04737         Vnm_print( 1, "    tot  %i %4.3e %4.3e %4.3e\n",
04738                   i,
04739                   xF,
04740                   yF,
04741                   zF);
04742         Vnm_print( 1, "    sasa %i %4.3e %4.3e %4.3e\n",
04743                   i,
04744                   (*atomForce)[i].sasaForce[0],
04745                   (*atomForce)[i].sasaForce[1],

```

```

04746         (*atomForce)[i].sasaForce[2]);
04747     Vnm_print( 1, " sav %i %4.3e %4.3e %4.3e\n",
04748         i,
04749         (*atomForce)[i].savForce[0],
04750         (*atomForce)[i].savForce[1],
04751         (*atomForce)[i].savForce[2]);
04752     Vnm_print( 1, " wca %i %4.3e %4.3e %4.3e\n",
04753         i,
04754         (*atomForce)[i].wcaForce[0],
04755         (*atomForce)[i].wcaForce[1],
04756         (*atomForce)[i].wcaForce[2]);
04757 #endif
04758     }
04759 } else *nforce = 0;
04760 }
04761 #ifndef VAPBSQUIET
04762     Vnm_print(1, "\n");
04763 #endif
04764 Vnm_print(0, "forceAPOL: Time elapsed: %f\n", ((double)clock() - ts_main) / CLOCKS_PER_SEC);
04765 return VRC_SUCCESS;
04766 }
04767 #ifdef ENABLE_BEM
04772 VPUBLIC int initBEM(int icalc,
04773     NOsh *nosh,
04774     BEMparm *bemparm,
04775     PBEParm *pbeparm,
04776     Vpbe *pbe[NOSH_MAXCALC]
04777 ) {
04782     Vnm_tstart(APBS_TIMER_SETUP, "Setup timer");
04783
04784     /* Setup time statistics */
04785     Vnm_tstop(APBS_TIMER_SETUP, "Setup timer");
04786
04787     return 1;
04788 }
04791 VPUBLIC void killBEM(NOsh *nosh, Vpbe *pbe[NOSH_MAXCALC]
04792 ) {
04793     int i;
04794
04795 #ifndef VAPBSQUIET
04796     Vnm_tprint(1, "Destroying boundary element structures.\n");
04797 #endif
04800 }
04801
04802 void apbs2tabipb_(char**, int*, double*,double* , double*, double*, double*, double*, double*, int*, int*,
04803 double*);
04806 VPUBLIC int solveBEM(NOsh *nosh, PBEParm *pbeparm, BEMparm *bemparm,
04807     BEMparm_CalcType type
04808 ) {
04809     int nx,
04810         ny,
04811         nz,
04812         i;
04813
04814     if (nosh != VNULL) {
04815         if (nosh->bogus) return 1;
04816     }
04817
04818     Vnm_tstart(APBS_TIMER_SOLVER, "Solver timer");
04819
04820 //apbs2tabipb(apbs_pqr_filename, nion, ionc, ionq, ionr, pdie, sdie, sdens, temp)
04821 apbs2tabipb_( (char*)&(nosh->molpath),
04822     &(pbeparm->nion),
04823     (double*)&(pbeparm->ionc),
04824     (double*)&(pbeparm->ionq),
04825     (double*)&(pbeparm->ionr),
04826     (double*)&(pbeparm->pdie),
04827     (double*)&(pbeparm->sdie),
04828     (double*)&(pbeparm->sdens),
04829     (double*)&(pbeparm->temp),
04830     (int*)&(nosh->nion),
04831     (int*)&(nosh->ionc),
04832     (int*)&(nosh->ionq),
04833     (int*)&(nosh->ionr),
04834     (int*)&(nosh->pdie),
04835     (int*)&(nosh->sdie),
04836     (int*)&(nosh->sdens),
04837     (int*)&(nosh->temp),
04838     (int*)&(nosh->nion),
04839     (int*)&(nosh->ionc),
04840     (int*)&(nosh->ionq),
04841     (int*)&(nosh->ionr),
04842     (int*)&(nosh->pdie),
04843     (int*)&(nosh->sdie),
04844     (int*)&(nosh->sdens),
04845     (int*)&(nosh->temp)

```

```

04828         (double*)&(pbeparm->sdie),
04829         (double*)&(pbeparm->sdens),
04830         (double*)&(pbeparm->temp),
04831         &(bemparm->tree_order),
04832         &(bemparm->tree_n0),
04833         (double*)&(bemparm->mac)
04834     );
04835
04836     Vnm_tstop(APBS_TIMER_SOLVER, "Solver timer");
04837
04838     return 1;
04839 }
04840
04841 VPUBLIC int setPartBEM(NOsh *nosh,
04842                      BEMparm *BEMparm
04843                      ) {
04844
04845     int j;
04846     double partMin[3],
04847            partMax[3];
04848
04849     if (nosh->bogus) return 1;
04850
04851     return 1;
04852 }
04853
04854 VPUBLIC int energyBEM(NOsh *nosh,
04855                     int icalc,
04856                     int *nenergy,
04857                     double *totEnergy,
04858                     double *qfEnergy,
04859                     double *qmEnergy,
04860                     double *dielEnergy
04861                     ) {
04862
04863     Valist *alist;
04864     Vatom *atom;
04865     int i,
04866         extEnergy;
04867     double tenergy;
04868     BEMparm *bemparm;
04869     PBEparm *pbeparm;
04870
04871     bemparm = nosh->calc[icalc]->bemparm;
04872     pbeparm = nosh->calc[icalc]->pbeparm;
04873
04874     Vnm_tstart(APBS_TIMER_ENERGY, "Energy timer");
04875     Vnm_tstop(APBS_TIMER_ENERGY, "Energy timer");
04876
04877     return 1;
04878 }
04879
04880 VPUBLIC int forceBEM(
04881     NOsh *nosh,
04882     PBEparm *pbeparm,
04883     BEMparm *bemparm,
04884     int *nforce,
04885     AtomForce **atomForce,
04886     Valist *alist[NOSH_MAXMOL]
04887     ) {
04888
04889     int j,
04890         k;
04891     double qfForce[3],
04892            dbForce[3],
04893            ibForce[3];
04894
04895     Vnm_tstart(APBS_TIMER_FORCE, "Force timer");
04896
04897     #ifndef VAPBSQUIET
04898     Vnm_tprint( 1, " Calculating forces...\n");
04899     #endif
04900
04901     Vnm_tstop(APBS_TIMER_FORCE, "Force timer");
04902
04903     return 1;
04904 }
04905
04906 VPUBLIC void printBEMPARM(BEMparm *bemparm) {

```

```
04909
04910 }
04911
04912
04913 VPUBLIC int writedataBEM(int rank,
04914                         Nosh *nosh,
04915                         PBEparm *pbeparm
04916                     ) {
04917
04918     return 1;
04919 }
04920
04921
04922 VPUBLIC int writematBEM(int rank, Nosh *nosh, PBEparm *pbeparm) {
04923
04924     if (nosh->bogus) return 1;
04925     return 1;
04926 }
04927
04928
04929 #endif
04930
04931
04932 VPUBLIC int initGEOFLOW(int icalc,
04933                       Nosh *nosh,
04934                       GEOFLOWparm *beparm,
04935                       PBEparm *pbeparm,
04936                       Vpbe *pbe[NOSH_MAXCALC]
04937                   ) {
04938
04939     Vnm_tstart(APBS_TIMER_SETUP, "Setup timer");
04940
04941     /* Setup time statistics */
04942     Vnm_tstop(APBS_TIMER_SETUP, "Setup timer");
04943
04944     return 1;
04945 }
04946
04947
04948 }
04949
04950 VPUBLIC void killGEOFLOW(Nosh *nosh, Vpbe *pbe[NOSH_MAXCALC]
04951                       ) {
04952
04953     int i;
04954
04955     #ifndef VAPBSQUIET
04956     Vnm_tprint(1, "Destroying geometric flow structures.\n");
04957     #endif
04958
04959
04960 }
04961
04962 VPUBLIC int solveGEOFLOW(Valist* molecules[NOSH_MAXMOL],
04963                       Nosh *nosh, PBEparm *pbeparm, APOLparm *apolparm, GEOFLOWparm *parm,
04964                       GEOFLOWparm_CalcType type) {
04965     int natm, m, a, i;
04966     double xyzr[MAXATOMS][XYZRWIDTH];
04967     double pqr[MAXATOMS];
04968     double *pos;
04969     Vatom *atom;
04970     GeoflowInput gfin;
04971     GeoflowOutput gf;
04972
04973     if (nosh != VNULL) {
04974         if (nosh->bogus) return 1;
04975     }
04976
04977     Vnm_tstart(APBS_TIMER_SOLVER, "Solver timer");
04978
04979     natm = 0;
04980     for (m=0; m < nosh->nmol; ++m){
04981         natm += Valist_getNumberAtoms(molecules[m]);
04982     }
04983
04984     /* double *xyzr, *pqr;
04985     xyzr = (double*) malloc(natm*4 * sizeof(double));
04986     pqr = (double*) malloc(natm * sizeof(double)); */
04987
04988     atom = VNULL;
04989     for(m=0; m < nosh->nmol; ++m){
04990         for(a=0; a < Valist_getNumberAtoms(molecules[m]); ++a){
04991             atom = Valist_getAtom(molecules[m], a);
04992             i = m*(nosh->nmol) + a;
```

```

04991     pos = Vatom_getPosition(atom);
04992     xyzr[i][0] = pos[0];
04993     xyzr[i][1] = pos[1];
04994     xyzr[i][2] = pos[2];
04995     xyzr[i][3] = Vatom_getRadius(atom);
04996     pqr[i] = Vatom_getCharge(atom);
04997 }
04998
04999 gfin.dcel = apolparm->grid;
05000 gfin.ffmodel = 1;
05001 gfin.extvalue = 1.90;
05002 gfin.pqr = pqr;
05003 gfin.maxstep = 20;
05004 gfin.crevalue = 0.01;
05005 gfin.iadi = 0;
05006 gfin.tottf = 3.5;
05007 gfin.ljepsilon = NULL;
05008 gfin.alpha = 0.50;
05009 gfin.igfin = 1;
05010 gfin.epsilonp = pbeparm->sdie;
05011 gfin.epsilonp = pbeparm->pdie;
05012 gfin.idacsl = 0;
05013 gfin.tol = 1.0e-5;
05014 gfin.iterf = 0;
05015 gfin.tpb = 0.0;
05016 gfin.itert = 0;
05017 gfin.pres = apolparm->press;
05018 gfin.gama = apolparm->gamma;
05019 gfin.tauval = 1.4;
05020 gfin.prob = 0.0;
05021 gfin.vdwdispersion = parm->vdw;
05022 gfin.sigmas = 1.5828;
05023 gfin.density = apolparm->bconc;
05024 gfin.epsilonw = 0.1554;
05025 gf = geoflowSolvation(xyzr, natm, gfin);
05026
05027 Vnm_tprint( 1," Global net energy = %1.12E\n", gf.totalSolvation);
05028 Vnm_tprint( 1," Global net ELEC energy = %1.12E\n", gf.elecSolvation);
05029 Vnm_tprint( 1," Global net APOL energy = %1.12E\n", gf.nonpolarSolvation);
05030
05031 Vnm_tstop(APBS_TIMER_SOLVER, "Solver timer");
05032
05033 return 1;
05034
05035 }
05036
05037 VPUBLIC int setPartGEOFLOW(NOsh *nosh,
05038                          GEOFLOWparm *GEOFLOWparm
05039                          ) {
05040
05041     int j;
05042     double partMin[3],
05043            partMax[3];
05044
05045     if (nosh->bogus) return 1;
05046
05047     return 1;
05048
05049 }
05050
05051 VPUBLIC int energyGEOFLOW(NOsh *nosh,
05052                          int icalc,
05053                          int *nenergy,
05054                          double *totEnergy,
05055                          double *qfEnergy,
05056                          double *qmEnergy,
05057                          double *dielEnergy
05058                          ) {
05059
05060     Valist *alist;
05061     Vatom *atom;
05062     int i,
05063         extEnergy;
05064     double tenergy;
05065     GEOFLOWparm *parm;
05066     PBEParm *pbeparm;
05067
05068     parm = nosh->calc[icalc]->geoflowparm;
05069     pbeparm = nosh->calc[icalc]->pbeparm;
05070
05071     Vnm_tstart(APBS_TIMER_ENERGY, "Energy timer");

```

```

05072     Vnm_tstop(APBS_TIMER_ENERGY, "Energy timer");
05073
05074     return 1;
05075 }
05076
05077 VPUBLIC int forceGEOFLOW(
05078     Nosh *nosh,
05079     PBeparm *pbeparm,
05080     GEOFLOWparm *parm,
05081     int *nforce,
05082     AtomForce **atomForce,
05083     Valist *alist[NOSH_MAXMOL]
05084 ) {
05085
05086     int j,
05087         k;
05088     double qfForce[3],
05089         dbForce[3],
05090         ibForce[3];
05091
05092     Vnm_tstart(APBS_TIMER_FORCE, "Force timer");
05093
05094     #ifndef VAPBSQUIET
05095         Vnm_tprint( 1, " Calculating forces...\n");
05096     #endif
05097
05098     Vnm_tstop(APBS_TIMER_FORCE, "Force timer");
05099
05100     return 1;
05101 }
05102
05103 VPUBLIC void printGEOFLOWPARM(GEOFLOWparm *parm) {
05104
05105 }
05106
05107
05108 VPUBLIC int writedataGEOFLOW(int rank,
05109     Nosh *nosh,
05110     PBeparm *pbeparm
05111 ) {
05112
05113     return 1;
05114 }
05115
05116
05117 VPUBLIC int writematGEOFLOW(int rank, Nosh *nosh, PBeparm *pbeparm) {
05118
05119
05120     if (nosh->bogus) return 1;
05121     return 1;
05122 }
05123

```

10.111 src/routines.h File Reference

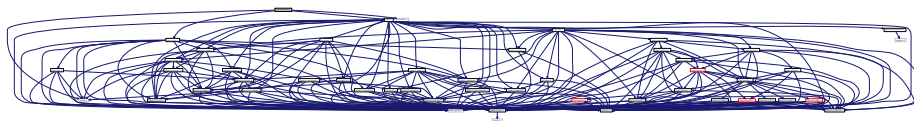
Header file for front end auxiliary routines.

```

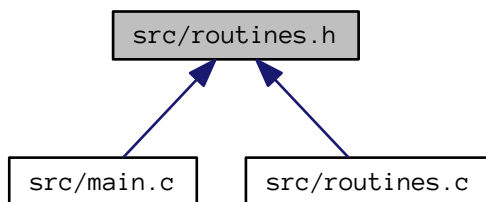
#include "apbs.h"
#include "mc/mc.h"
#include "fem/vfetk.h"
#include "mcx/mcx.h"

```

Include dependency graph for routines.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [AtomForce](#)
Structure to hold atomic forces.

Macros

- #define [APBSRC](#) 13
Return code for APBS during failure.

Typedefs

- typedef struct [AtomForce](#) [AtomForce](#)
Define [AtomForce](#) type.

Functions

- VEXTERNC [Vparam](#) * [loadParameter](#) ([NOsh](#) *nosh)
Loads and returns parameter object.
- VEXTERNC int [loadMolecules](#) ([NOsh](#) *nosh, [Vparam](#) *param, [Valist](#) *alist[[NOSH_MAXMOL](#)])
Load the molecules given in NOsh into atom lists.
- VEXTERNC void [killMolecules](#) ([NOsh](#) *nosh, [Valist](#) *alist[[NOSH_MAXMOL](#)])
Destroy the loaded molecules.
- VEXTERNC int [loadDielMaps](#) ([NOsh](#) *nosh, [Vgrid](#) *dielXMap[[NOSH_MAXMOL](#)], [Vgrid](#) *dielYMap[[NOSH_MAXMOL](#)], [Vgrid](#) *dielZMap[[NOSH_MAXMOL](#)])
Load the dielectric maps given in NOsh into grid objects.
- VEXTERNC void [killDielMaps](#) ([NOsh](#) *nosh, [Vgrid](#) *dielXMap[[NOSH_MAXMOL](#)], [Vgrid](#) *dielYMap[[NOSH_MAXMOL](#)], [Vgrid](#) *dielZMap[[NOSH_MAXMOL](#)])
Destroy the loaded dielectric.
- VEXTERNC int [loadKappaMaps](#) ([NOsh](#) *nosh, [Vgrid](#) *kappa[[NOSH_MAXMOL](#)])
Load the kappa maps given in NOsh into grid objects.

- VEXTERNC void `killKappaMaps` (`NOsh *nosh`, `Vgrid *kappa[NOSH_MAXMOL]`)
Destroy the loaded kappa maps.
- VEXTERNC int `loadPotMaps` (`NOsh *nosh`, `Vgrid *pot[NOSH_MAXMOL]`)
Load the potential maps given in NOsh into grid objects.
- VEXTERNC void `killPotMaps` (`NOsh *nosh`, `Vgrid *pot[NOSH_MAXMOL]`)
Destroy the loaded potential maps.
- VEXTERNC int `loadChargeMaps` (`NOsh *nosh`, `Vgrid *charge[NOSH_MAXMOL]`)
Load the charge maps given in NOsh into grid objects.
- VEXTERNC void `killChargeMaps` (`NOsh *nosh`, `Vgrid *charge[NOSH_MAXMOL]`)
Destroy the loaded charge maps.
- VEXTERNC void `printPBEPARM` (`PBEparm *pbeparm`)
Print out generic PBE params loaded from input.
- VEXTERNC void `printMGPARM` (`MGparm *mgparm`, `double realCenter[3]`)
Print out MG-specific params loaded from input.
- VEXTERNC int `initMG` (`int icalc`, `NOsh *nosh`, `MGparm *mgparm`, `PBEparm *pbeparm`, `double realCenter[3]`, `Vpbe *pbe[NOSH_MAXCALC]`, `Valist *alist[NOSH_MAXMOL]`, `Vgrid *dielXMap[NOSH_MAXMOL]`, `Vgrid *dielYMap[NOSH_MAXMOL]`, `Vgrid *dielZMap[NOSH_MAXMOL]`, `Vgrid *kappaMap[NOSH_MAXMOL]`, `Vgrid *chargeMap[NOSH_MAXMOL]`, `Vpmgp *pmgp[NOSH_MAXCALC]`, `Vpmg *pmg[NOSH_MAXCALC]`, `Vgrid *potMap[NOSH_MAXMOL]`)
Initialize an MG calculation.
- VEXTERNC void `killMG` (`NOsh *nosh`, `Vpbe *pbe[NOSH_MAXCALC]`, `Vpmgp *pmgp[NOSH_MAXCALC]`, `Vpmg *pmg[NOSH_MAXCALC]`)
Kill structures initialized during an MG calculation.
- VEXTERNC int `solveMG` (`NOsh *nosh`, `Vpmg *pmg`, `MGparm_CalcType type`)
Solve the PBE with MG.
- VEXTERNC int `setPartMG` (`NOsh *nosh`, `MGparm *mgparm`, `Vpmg *pmg`)
Set MG partitions for calculating observables and performing I/O.
- VEXTERNC int `energyMG` (`NOsh *nosh`, `int icalc`, `Vpmg *pmg`, `int *nenergy`, `double *totEnergy`, `double *qfEnergy`, `double *qmEnergy`, `double *dielEnergy`)
Calculate electrostatic energies from MG solution.
- VEXTERNC void `killEnergy` ()
Kill arrays allocated for energies.
- VEXTERNC int `forceMG` (`Vmem *mem`, `NOsh *nosh`, `PBEparm *pbeparm`, `MGparm *mgparm`, `Vpmg *pmg`, `int *nforce`, `AtomForce **atomForce`, `Valist *alist[NOSH_MAXMOL]`)
Calculate forces from MG solution.
- VEXTERNC void `killForce` (`Vmem *mem`, `NOsh *nosh`, `int nforce[NOSH_MAXCALC]`, `AtomForce *atomForce[NOSH_MAXCALC]`)
Free memory from MG force calculation.
- VEXTERNC void `storeAtomEnergy` (`Vpmg *pmg`, `int icalc`, `double **atomEnergy`, `int *nenergy`)
Store energy in arrays for future use.
- VEXTERNC int `writedataFlat` (`NOsh *nosh`, `Vcom *com`, `const char *fname`, `double totEnergy[NOSH_MAXCALC]`, `double qfEnergy[NOSH_MAXCALC]`, `double qmEnergy[NOSH_MAXCALC]`, `double dielEnergy[NOSH_MAXCALC]`, `int nenergy[NOSH_MAXCALC]`, `double *atomEnergy[NOSH_MAXCALC]`, `int nforce[NOSH_MAXCALC]`, `AtomForce *atomForce[NOSH_MAXCALC]`)
Write out information to a flat file.
- VEXTERNC int `writedataXML` (`NOsh *nosh`, `Vcom *com`, `const char *fname`, `double totEnergy[NOSH_MAXCALC]`, `double qfEnergy[NOSH_MAXCALC]`, `double qmEnergy[NOSH_MAXCALC]`, `double dielEnergy[NOSH_MAXCALC]`, `int nenergy[NOSH_MAXCALC]`, `double *atomEnergy[NOSH_MAXCALC]`, `int nforce[NOSH_MAXCALC]`, `AtomForce *atomForce[NOSH_MAXCALC]`)
Write out information to an XML file.

- Write out information to an XML file.*

 - VEXTERNC int [writedataMG](#) (int rank, [NOsh](#) *nosh, [PBEparm](#) *pbeparm, [Vpmg](#) *pmg)
- Write out observables from MG calculation to file.*

 - VEXTERNC int [writematMG](#) (int rank, [NOsh](#) *nosh, [PBEparm](#) *pbeparm, [Vpmg](#) *pmg)
- Write out operator matrix from MG calculation to file.*

 - VEXTERNC double [returnEnergy](#) (Vcom *com, [NOsh](#) *nosh, double totEnergy[NOSH_MAXCALC], int iprint)
- Access net local energy.*

 - VEXTERNC int [printEnergy](#) (Vcom *com, [NOsh](#) *nosh, double totEnergy[NOSH_MAXCALC], int iprint)
- Combine and pretty-print energy data (deprecated...see printElecEnergy)*

 - VEXTERNC int [printElecEnergy](#) (Vcom *com, [NOsh](#) *nosh, double totEnergy[NOSH_MAXCALC], int iprint)
- Combine and pretty-print energy data.*

 - VEXTERNC int [printApolEnergy](#) ([NOsh](#) *nosh, int iprint)
- Combine and pretty-print energy data.*

 - VEXTERNC int [printForce](#) (Vcom *com, [NOsh](#) *nosh, int nforce[NOSH_MAXCALC], [AtomForce](#) *atomForce[NOSH_MAXCALC], int i)
- Combine and pretty-print force data (deprecated...see printElecForce)*

 - VEXTERNC int [printElecForce](#) (Vcom *com, [NOsh](#) *nosh, int nforce[NOSH_MAXCALC], [AtomForce](#) *atomForce[NOSH_MAXCALC], int i)
- Combine and pretty-print force data.*

 - VEXTERNC int [printApolForce](#) (Vcom *com, [NOsh](#) *nosh, int nforce[NOSH_MAXCALC], [AtomForce](#) *atomForce[NOSH_MAXCALC], int i)
- Combine and pretty-print force data.*

 - VEXTERNC void [startVio](#) ()
- Wrapper to start MALOC Vio layer.*

 - VEXTERNC int [energyAPOL](#) ([APOLparm](#) *apolparm, double sasa, double sav, double atomsasa[], double atomwcaEnergy[], int numatoms)
- Calculate non-polar energies.*

 - VEXTERNC int [forceAPOL](#) (Vacc *acc, Vmem *mem, [APOLparm](#) *apolparm, int *nforce, [AtomForce](#) **atomForce, [Valist](#) *alist, [Vclist](#) *clist)
- Calculate non-polar forces.*

 - VEXTERNC int [initAPOL](#) ([NOsh](#) *nosh, Vmem *mem, [Vparam](#) *param, [APOLparm](#) *apolparm, int *nforce, [AtomForce](#) **atomForce, [Valist](#) *alist)
- Upperlevel routine to the non-polar energy and force routines.*

 - VEXTERNC void [printFEPARM](#) (int icalc, [NOsh](#) *nosh, [FEMparm](#) *feparm, [Vfetk](#) *fetk[NOSH_MAXCALC])
- Print out FE-specific params loaded from input.*

 - VEXTERNC int [energyFE](#) ([NOsh](#) *nosh, int icalc, [Vfetk](#) *fetk[NOSH_MAXCALC], int *nenergy, double *totEnergy, double *qfEnergy, double *qmEnergy, double *dielEnergy)
- Calculate electrostatic energies from FE solution.*

 - VPUBLIC Vrc_Codes [initFE](#) (int icalc, [NOsh](#) *nosh, [FEMparm](#) *feparm, [PBEparm](#) *pbeparm, [Vpbe](#) *pbe[NOSH_MAXCALC], [Valist](#) *alist[NOSH_MAXMOL], [Vfetk](#) *fetk[NOSH_MAXCALC])
- Initialize FE solver objects.*

 - VEXTERNC void [killFE](#) ([NOsh](#) *nosh, [Vpbe](#) *pbe[NOSH_MAXCALC], [Vfetk](#) *fetk[NOSH_MAXCALC], Gem *gem[NOSH_MAXMOL])
- Kill structures initialized during an FE calculation.*

 - VEXTERNC int [preRefineFE](#) (int i, [FEMparm](#) *feparm, [Vfetk](#) *fetk[NOSH_MAXCALC])
- Pre-refine mesh before solve.*

 - VEXTERNC int [partFE](#) (int i, [NOsh](#) *nosh, [FEMparm](#) *feparm, [Vfetk](#) *fetk[NOSH_MAXCALC])
- Partition mesh (if applicable)*

- VEXTERNC int [solveFE](#) (int i, [PBEparm](#) *pbeparm, [FEMparm](#) *feparm, [Vfetk](#) *fetk[NOSH_MAXCALC])
Solve-estimate-refine.
- VEXTERNC int [postRefineFE](#) (int icalc, [FEMparm](#) *feparm, [Vfetk](#) *fetk[NOSH_MAXCALC])
Estimate error, mark mesh, and refine mesh after solve.
- VEXTERNC int [writedataFE](#) (int rank, [NOsh](#) *nosh, [PBEparm](#) *pbeparm, [Vfetk](#) *fetk)
Write FEM data to files.
- VEXTERNC Vrc_Codes [loadMeshes](#) ([NOsh](#) *nosh, Gem *gm[NOSH_MAXMOL])
Load the meshes given in NOsh into geometry objects.
- VEXTERNC void [killMeshes](#) ([NOsh](#) *nosh, Gem *alist[NOSH_MAXMOL])
Destroy the loaded meshes.
- VEXTERNC int [initGEOFLOW](#) (int icalc, [NOsh](#) *nosh, [GEOFLOWparm](#) *bemparm, [PBEparm](#) *pbeparm, [Vpbe](#) *pbe[NOSH_MAXCALC])
Initialize an GEOFLOW calculation.
- VEXTERNC void [killGEOFLOW](#) ([NOsh](#) *nosh, [Vpbe](#) *pbe[NOSH_MAXCALC])
Kill structures initialized during an GEOFLOW calculation.
- VEXTERNC int [solveGEOFLOW](#) ([Valist](#) *molecules[NOSH_MAXMOL], [NOsh](#) *nosh, [PBEparm](#) *pbeparm, [APOLparm](#) *apolparm, [GEOFLOWparm](#) *parm, [GEOFLOWparm_CalcType](#) type)
Solve the PBE with GEOFLOW.
- VEXTERNC int [setPartGEOFLOW](#) ([NOsh](#) *nosh, [GEOFLOWparm](#) *parm)
Set GEOFLOW partitions for calculating observables and performing I/O.
- VEXTERNC int [energyGEOFLOW](#) ([NOsh](#) *nosh, int icalc, int *nenergy, double *totEnergy, double *qfEnergy, double *qmEnergy, double *dielEnergy)
Calculate electrostatic energies from GEOFLOW solution.
- VEXTERNC int [forceGEOFLOW](#) ([NOsh](#) *nosh, [PBEparm](#) *pbeparm, [GEOFLOWparm](#) *parm, int *nforce, [AtomForce](#) **atomForce, [Valist](#) *alist[NOSH_MAXMOL])
Calculate forces from GEOFLOW solution.
- VEXTERNC void [printGEOFLOWPARAM](#) ([GEOFLOWparm](#) *parm)
Print out GEOFLOW-specific params loaded from input.
- VEXTERNC int [writedataGEOFLOW](#) (int rank, [NOsh](#) *nosh, [PBEparm](#) *pbeparm)
Write out observables from GEOFLOW calculation to file.
- VEXTERNC int [writematGEOFLOW](#) (int rank, [NOsh](#) *nosh, [PBEparm](#) *pbeparm)
Write out operator matrix from GEOFLOW calculation to file.

10.111.1 Detailed Description

Header file for front end auxiliary routines.

Author

Nathan Baker

Version

\$Id\$

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [routines.h](#).

10.112 routines.h

```

00001
00061 #ifndef _APBSROUTINES_H_
00062 #define _APBSROUTINES_H_
00063
00064 #include "apbs.h"
00065
00066 #ifdef HAVE_MC_H
00067 #   include "mc/mc.h"
00068 #   include "fem/vfetc.h"
00069 #endif
00070 #ifdef HAVE_MCX_H
00071 #   include "mcx/mcx.h"
00072 #endif
00073

```

```

00077 #define APBSRC 13
00078
00083 struct AtomForce {
00084     double ibForce[3];
00085     double qfForce[3];
00086     double dbForce[3];
00087     double sasaForce[3];
00088     double savForce[3];
00089     double wcaForce[3];
00090 };
00091
00095 typedef struct AtomForce AtomForce;
00096
00102 VEXTERNC Vparam* loadParameter(
00103     Nosh *nosh
00105 );
00106
00112 VEXTERNC int loadMolecules(
00113     Nosh *nosh,
00114     Vparam *param,
00116     Valist *alist[NOSH_MAXMOL]
00118 );
00119
00126 VEXTERNC void killMolecules(Nosh *nosh, Valist *alist[
    NOSH_MAXMOL]);
00127
00137 VEXTERNC int loadDielMaps(Nosh *nosh,
00138     Vgrid *dielXMap[NOSH_MAXMOL],
00139     Vgrid *dielYMap[NOSH_MAXMOL],
00140     Vgrid *dielZMap[NOSH_MAXMOL]
00141 );
00142
00151 VEXTERNC void killDielMaps(Nosh *nosh, Vgrid *dielXMap[
    NOSH_MAXMOL],
00152     Vgrid *dielYMap[NOSH_MAXMOL], Vgrid *dielZMap[NOSH_MAXMOL]);
00153
00161 VEXTERNC int loadKappaMaps(Nosh *nosh, Vgrid *kappa[
    NOSH_MAXMOL]);
00162
00169 VEXTERNC void killKappaMaps(Nosh *nosh, Vgrid *kappa[
    NOSH_MAXMOL]);
00170
00178 VEXTERNC int loadPotMaps(Nosh *nosh, Vgrid *pot[NOSH_MAXMOL]);
00179
00186 VEXTERNC void killPotMaps(Nosh *nosh, Vgrid *pot[NOSH_MAXMOL]);
00187
00195 VEXTERNC int loadChargeMaps(Nosh *nosh, Vgrid *charge[
    NOSH_MAXMOL]);
00196
00203 VEXTERNC void killChargeMaps(Nosh *nosh, Vgrid *charge[
    NOSH_MAXMOL]);
00204
00210 VEXTERNC void printPBE Parm(PBEparm *pbeparm);
00211
00218 VEXTERNC void printMG Parm(MGparm *mgparm, double realCenter[3]);
00219
00225 VEXTERNC int initMG(
00226     int icalc,
00227     Nosh *nosh,
00228     MGparm *mgparm,
00229     PBEparm *pbeparm,
00230     double realCenter[3],
00231     Vpbe *pbe[NOSH_MAXCALC],
00232     Valist *alist[NOSH_MAXMOL],
00233     Vgrid *dielXMap[NOSH_MAXMOL],
00234     Vgrid *dielYMap[NOSH_MAXMOL],
00235     Vgrid *dielZMap[NOSH_MAXMOL],
00236     Vgrid *kappaMap[NOSH_MAXMOL],
00237     Vgrid *chargeMap[NOSH_MAXMOL],
00238     Vpmgp *pmgp[NOSH_MAXCALC],
00239     Vpmg *pmg[NOSH_MAXCALC],
00240     Vgrid *potMap[NOSH_MAXMOL]
00241 );
00242
00248 VEXTERNC void killMG(
00249     Nosh *nosh,
00250     Vpbe *pbe[NOSH_MAXCALC],
00251     Vpmgp *pmgp[NOSH_MAXCALC],
00252     Vpmg *pmg[NOSH_MAXCALC]
00253 );
00254

```

```

00263 VEXTERNC int solveMG(Nosh *nosh, Vpmg *pmg, MGparm_CalcType type);
00264
00273 VEXTERNC int setPartMG(Nosh *nosh, MGparm *mgparm, Vpmg *pmg);
00274
00288 VEXTERNC int energyMG(Nosh *nosh, int icalc, Vpmg *pmg,
00289     int *nenergy, double *totEnergy, double *qfEnergy, double *qmEnergy,
00290     double *dielEnergy);
00291
00296 VEXTERNC void killEnergy();
00297
00311 VEXTERNC int forceMG(Vmem *mem, Nosh *nosh, PBEparm *pbeparm,
00312     MGparm *mgparm,
00313     Vpmg *pmg, int *nforce, AtomForce **atomForce, Valist *alist[
00314     NOSH_MAXMOL]);
00313
00322 VEXTERNC void killForce(Vmem *mem, Nosh *nosh, int nforce[
00323     NOSH_MAXCALC],
00324     AtomForce *atomForce[NOSH_MAXCALC]);
00324
00329 VEXTERNC void storeAtomEnergy(
00330     Vpmg *pmg,
00331     int icalc,
00332     double **atomEnergy,
00333     int *nenergy
00334     );
00335
00352 VEXTERNC int writedataFlat(Nosh *nosh, Vcom *com, const char *fname,
00353     double totEnergy[NOSH_MAXCALC], double qfEnergy[NOSH_MAXCALC],
00354     double qmEnergy[NOSH_MAXCALC], double dielEnergy[NOSH_MAXCALC],
00355     int nenergy[NOSH_MAXCALC], double *atomEnergy[NOSH_MAXCALC],
00356     int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC]);
00357
00374 VEXTERNC int writedataXML(Nosh *nosh, Vcom *com, const char *fname,
00375     double totEnergy[NOSH_MAXCALC], double qfEnergy[NOSH_MAXCALC],
00376     double qmEnergy[NOSH_MAXCALC], double dielEnergy[NOSH_MAXCALC],
00377     int nenergy[NOSH_MAXCALC], double *atomEnergy[NOSH_MAXCALC],
00378     int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC]);
00379
00389 VEXTERNC int writedataMG(int rank, Nosh *nosh, PBEparm *pbeparm,
00390     Vpmg *pmg);
00390
00400 VEXTERNC int writematMG(int rank, Nosh *nosh, PBEparm *pbeparm,
00401     Vpmg *pmg);
00401
00411 VEXTERNC double returnEnergy(Vcom *com, Nosh *nosh, double totEnergy[
00412     NOSH_MAXCALC], int iprint);
00412
00418 VEXTERNC int printEnergy(
00419     Vcom *com,
00420     Nosh *nosh,
00421     double totEnergy[NOSH_MAXCALC],
00422     int iprint
00423     );
00424
00431 VEXTERNC int printElecEnergy(
00432     Vcom *com,
00433     Nosh *nosh,
00434     double totEnergy[NOSH_MAXCALC],
00435     int iprint
00436     );
00437
00444 VEXTERNC int printApolEnergy(
00445     Nosh *nosh,
00446     int iprint
00447     );
00448
00454 VEXTERNC int printForce(
00455     Vcom *com,
00456     Nosh *nosh,
00457     int nforce[NOSH_MAXCALC],
00458     AtomForce *atomForce[NOSH_MAXCALC],
00459     int i
00460     );
00461
00467 VEXTERNC int printElecForce(
00468     Vcom *com,
00469     Nosh *nosh,
00470     int nforce[NOSH_MAXCALC],
00471     AtomForce *atomForce[NOSH_MAXCALC],
00472     int i
00473     );

```

```

00474
00480 VEXTERNC int printApolForce(
00481     Vcom *com,
00482     Nosh *nosh,
00483     int nforce[NOSH_MAXCALC],
00484     AtomForce *atomForce[NOSH_MAXCALC],
00485     int i
00486 );
00487
00492 VEXTERNC void startVio();
00493
00499 VEXTERNC int energyAPOL(
00500     APOLparm *apolparm,
00501     double sasa,
00502     double sav,
00503     double atomsasa[],
00504     double atomwcaEnergy[],
00505     int numatoms
00506 );
00507
00513 VEXTERNC int forceAPOL(
00514     Vacc *acc,
00515     Vmem *mem,
00516     APOLparm *apolparm,
00517     int *nforce,
00518     AtomForce **atomForce,
00519     Valist *alist,
00520     Vclist *clist
00521 );
00522
00525
00531 VEXTERNC int initAPOL(
00532     Nosh *nosh,
00533     Vmem *mem,
00534     Vparam *param,
00535     APOLparm *apolparm,
00536     int *nforce,
00537     AtomForce **atomForce,
00538     Valist *alist
00539 );
00540
00541
00542 #ifdef HAVE_MC_H
00543 #include "fem/vfetk.h"
00544
00553 VEXTERNC void printFEPARM(int icalc, Nosh *nosh, FEMparm *feparm,
00554     Vfetk *fetk[NOSH_MAXCALC]);
00555
00570 VEXTERNC int energyFE(Nosh* nosh, int icalc, Vfetk *fetk[
00571     NOSH_MAXCALC],
00572     int *nenergy, double *totEnergy, double *qfEnergy, double *qmEnergy,
00573     double *dielEnergy);
00574
00581 VEXTERNC Vrc_Codes initFE(
00582     int icalc,
00583     Nosh *nosh,
00584     FEMparm *feparm,
00585     PBEParm *pbeparm,
00586     Vpbe *pbe[NOSH_MAXCALC],
00587     Valist *alist[NOSH_MAXMOL],
00588     Vfetk *fetk[NOSH_MAXCALC]
00589 );
00590
00596 VEXTERNC void killFE(
00597     Nosh *nosh,
00598     Vpbe *pbe[NOSH_MAXCALC],
00599     Vfetk *fetk[NOSH_MAXCALC],
00600     Gem *gem[NOSH_MAXMOL]
00601 );
00602
00612 VEXTERNC int preRefineFE(int i,
00613     FEMparm *feparm,
00614     Vfetk *fetk[NOSH_MAXCALC]
00615 );
00616
00626 VEXTERNC int partFE(int i, Nosh *nosh, FEMparm *feparm,
00627     Vfetk *fetk[NOSH_MAXCALC]);
00628
00638 VEXTERNC int solveFE(int i,
00639     PBEParm *pbeparm,
00640     FEMparm *feparm,
00641     Vfetk *fetk[NOSH_MAXCALC]

```

```
00642 );
00643
00655 VEXTERNC int postRefineFE(int icalc,
00656                          FEMparm *feparm,
00657                          Vfetc *fetc[NOSH_MAXCALC]
00658 );
00659
00669 VEXTERNC int writedataFE(int rank, Nosh *nosh, PBEparm *pbeparm,
00670                          Vfetc *fetc);
00671
00676 VEXTERNC Vrc_Codes loadMeshes(
00677                          Nosh *nosh,
00678                          Gem *gm[NOSH_MAXMOL]
00679 );
00680
00686 VEXTERNC void killMeshes(
00687                          Nosh *nosh,
00688                          Gem *alist[NOSH_MAXMOL]
00689 );
00690 #endif
00691 #endif
00692 #endif
00693
00694
00695
00696 VEXTERNC void printMGPARAM(MGparm *mgparm, double realCenter[3]);
00697
00698 #ifdef ENABLE_BEM
00699
00704 VEXTERNC int initBEM(
00705                          int icalc,
00706                          Nosh *nosh,
00707                          BEMparm *beparm,
00708                          PBEparm *pbeparm,
00709                          Vpbe *pbe[NOSH_MAXCALC]
00710 );
00711
00717 VEXTERNC void killBEM(
00718                          Nosh *nosh,
00719                          Vpbe *pbe[NOSH_MAXCALC]
00720 );
00721
00730 VEXTERNC int solveBEM(Nosh *nosh, PBEparm *pbeparm, BEMparm *beparm,
00731                      BEMparm_CalcType type);
00732
00740 VEXTERNC int setPartBEM(Nosh *nosh, BEMparm *beparm);
00741
00755 VEXTERNC int energyBEM(Nosh* nosh, int icalc,
00756                      int *nenergy, double *totEnergy, double *qfEnergy, double *qmEnergy,
00757                      double *dielEnergy);
00758
00772 VEXTERNC int forceBEM(Nosh *nosh, PBEparm *pbeparm, BEMparm *beparm,
00773                      int *nforce, AtomForce **atomForce, Valist *alist[NOSH_MAXMOL]);
00774
00781 VEXTERNC void printBEMPARAM(BEMparm *beparm);
00782
00792 VEXTERNC int writedataBEM(int rank, Nosh *nosh, PBEparm *pbeparm);
00793
00803 VEXTERNC int writematBEM(int rank, Nosh *nosh, PBEparm *pbeparm);
00804 #endif
00805
00811 VEXTERNC int initGEOFLOW (
00812                          int icalc,
00813                          Nosh *nosh,
00814                          GEOFLOWparm *beparm,
00815                          PBEparm *pbeparm,
00816                          Vpbe *pbe[NOSH_MAXCALC]
00817 );
00818
00824 VEXTERNC void killGEOFLOW (
00825                          Nosh *nosh,
00826                          Vpbe *pbe[NOSH_MAXCALC]
00827 );
00828
00837 VEXTERNC int solveGEOFLOW(Valist* molecules[NOSH_MAXMOL],
00838                      Nosh *nosh, PBEparm *pbeparm, APOLparm *apolparm, GEOFLOWparm *parm,
00839                      GEOFLOWparm_CalcType type);
00840
00846 VEXTERNC int setPartGEOFLOW(Nosh *nosh, GEOFLOWparm *parm);
00847
00860 VEXTERNC int energyGEOFLOW(Nosh* nosh, int icalc,
```

```
00861  int *nenergy, double *totEnergy, double *qfEnergy, double *qmEnergy,
00862  double *dielEnergy);
00863
00875 VEXTERNC int forceGEOFLOW(Nosh *nosh, PBEparm *pbeparm,
    GEOFLOWparm *parm,
00876  int *nforce, AtomForce **atomForce, Valist *alist[NOSH_MAXMOL]);
00877
00882 VEXTERNC void printGEOFLOWPARM(GEOFLOWparm *parm);
00883
00892 VEXTERNC int writedataGEOFLOW(int rank, Nosh *nosh, PBEparm *pbeparm);
00893
00902 VEXTERNC int writematGEOFLOW(int rank, Nosh *nosh, PBEparm *pbeparm);
00903
```


Index

ACD_ERROR
 APOLparm class, [61](#)
ACD_NO
 APOLparm class, [61](#)
ACD_YES
 APOLparm class, [61](#)
ACE_COMPS
 APOLparm class, [61](#)
ACE_NO
 APOLparm class, [61](#)
ACE_TOTAL
 APOLparm class, [61](#)
ACF_COMPS
 APOLparm class, [61](#)
ACF_NO
 APOLparm class, [61](#)
ACF_TOTAL
 APOLparm class, [61](#)
APOLparm class
 ACD_ERROR, [61](#)
 ACD_NO, [61](#)
 ACD_YES, [61](#)
 ACE_COMPS, [61](#)
 ACE_NO, [61](#)
 ACE_TOTAL, [61](#)
 ACF_COMPS, [61](#)
 ACF_NO, [61](#)
 ACF_TOTAL, [61](#)
BCFL_FOCUS
 Vhal class, [163](#)
BCFL_MAP
 Vhal class, [163](#)
BCFL_MDH
 Vhal class, [163](#)
BCFL_MEM
 Vhal class, [163](#)
BCFL_SDH
 Vhal class, [163](#)
BCFL_UNUSED
 Vhal class, [163](#)
BCFL_ZERO
 Vhal class, [163](#)
BCT_MANUAL
 BEMparm class, [65](#)
BCT_NONE
 BEMparm class, [65](#)
BEMparm class
 BCT_MANUAL, [65](#)
 BCT_NONE, [65](#)
bcolcomp
 Vpmg class, [218](#)
bcolcomp2
 Vpmg class, [218](#)
bcolcomp3
 Vpmg class, [219](#)
bcolcomp4
 Vpmg class, [219](#)
CLIST_AUTO_DOMAIN
 Vclist class, [145](#)
CLIST_MANUAL_DOMAIN
 Vclist class, [145](#)
Comdata, [347](#)
dependencies, [21](#)
FCT_MANUAL
 FEMparm class, [69](#)
FCT_NONE
 FEMparm class, [69](#)
FEMparm class
 FCT_MANUAL, [69](#)
 FCT_NONE, [69](#)
 FET_FRAC, [70](#)
 FET_GLOB, [70](#)
 FET_SIMP, [70](#)
 FRT_DUAL, [70](#)
 FRT_GEOM, [70](#)
 FRT_LOCA, [70](#)
 FRT_RESI, [70](#)
 FRT_UNIF, [70](#)
FET_FRAC
 FEMparm class, [70](#)
FET_GLOB
 FEMparm class, [70](#)
FET_SIMP
 FEMparm class, [70](#)
FRT_DUAL
 FEMparm class, [70](#)
FRT_GEOM
 FEMparm class, [70](#)

- FRT_LOCA
 - FEMparm class, [70](#)
- FRT_RESI
 - FEMparm class, [70](#)
- FRT_UNIF
 - FEMparm class, [70](#)
- GEOFLOWparm class
 - GFCT_AUTO, [74](#)
 - GFCT_MANUAL, [74](#)
 - GFCT_NONE, [74](#)
- GFCT_AUTO
 - GEOFLOWparm class, [74](#)
- GFCT_MANUAL
 - GEOFLOWparm class, [74](#)
- GFCT_NONE
 - GEOFLOWparm class, [74](#)
- High-level front-end routines, [316](#)
 - main, [333](#)
- IPKEY_LPBE
 - Vhal class, [165](#)
- IPKEY_NPBE
 - Vhal class, [165](#)
- IPKEY_SMPBE
 - Vhal class, [165](#)
- MCM_FOCUS
 - MGparm class, [79](#)
- MCM_MOLECULE
 - MGparm class, [79](#)
- MCM_POINT
 - MGparm class, [79](#)
- MCT_AUTO
 - MGparm class, [78](#)
- MCT_DUMMY
 - MGparm class, [79](#)
- MCT_MANUAL
 - MGparm class, [78](#)
- MCT_NONE
 - MGparm class, [79](#)
- MCT_PARALLEL
 - MGparm class, [78](#)
- MGparm class
 - MCM_FOCUS, [79](#)
 - MCM_MOLECULE, [79](#)
 - MCM_POINT, [79](#)
 - MCT_AUTO, [78](#)
 - MCT_DUMMY, [79](#)
 - MCT_MANUAL, [78](#)
 - MCT_NONE, [79](#)
 - MCT_PARALLEL, [78](#)
- main
 - High-level front-end routines, [333](#)
- Mat< T >, [349](#)
- Matrix wrapper class, [167](#)
- NCT_APOL
 - NOsh class, [89](#)
- NCT_BEM
 - NOsh class, [89](#)
- NCT_FEM
 - NOsh class, [89](#)
- NCT_GEOFLOW
 - NOsh class, [89](#)
- NCT_MG
 - NOsh class, [89](#)
- NMF_PDB
 - NOsh class, [90](#)
- NMF_PQR
 - NOsh class, [90](#)
- NMF_XML
 - NOsh class, [90](#)
- NOsh class
 - NCT_APOL, [89](#)
 - NCT_BEM, [89](#)
 - NCT_FEM, [89](#)
 - NCT_GEOFLOW, [89](#)
 - NCT_MG, [89](#)
 - NMF_PDB, [90](#)
 - NMF_PQR, [90](#)
 - NMF_XML, [90](#)
 - NPF_FLAT, [90](#)
 - NPF_XML, [90](#)
 - NPT_APOLENERGY, [90](#)
 - NPT_APOLFORCE, [90](#)
 - NPT_ELECENERGY, [90](#)
 - NPT_ELECFORCE, [90](#)
 - NPT_ENERGY, [90](#)
 - NPT_FORCE, [90](#)
- NPF_FLAT
 - NOsh class, [90](#)
- NPF_XML
 - NOsh class, [90](#)
- NPT_APOLENERGY
 - NOsh class, [90](#)
- NPT_APOLFORCE
 - NOsh class, [90](#)
- NPT_ELECENERGY
 - NOsh class, [90](#)
- NPT_ELECFORCE
 - NOsh class, [90](#)
- NPT_ENERGY
 - NOsh class, [90](#)
- NPT_FORCE
 - NOsh class, [90](#)
- OUTPUT_FLAT
 - Vhal class, [165](#)

- OUTPUT_NULL
 - Vhal class, [165](#)
- PBE_LPBE
 - Vhal class, [165](#)
- PBE_LRPBE
 - Vhal class, [165](#)
- PBE_NPBE
 - Vhal class, [165](#)
- PBE_SMPBE
 - Vhal class, [165](#)
- PBEparm class
 - PCE_COMPS, [104](#)
 - PCE_NO, [104](#)
 - PCE_TOTAL, [104](#)
 - PCF_COMPS, [104](#)
 - PCF_NO, [104](#)
 - PCF_TOTAL, [104](#)
- PCE_COMPS
 - PBEparm class, [104](#)
- PCE_NO
 - PBEparm class, [104](#)
- PCE_TOTAL
 - PBEparm class, [104](#)
- PCF_COMPS
 - PBEparm class, [104](#)
- PCF_NO
 - PBEparm class, [104](#)
- PCF_TOTAL
 - PBEparm class, [104](#)
- pcolcomp
 - Vpmg class, [220](#)
- Stencil< T >, [390](#)
- VCM_BSPL2
 - Vhal class, [163](#)
- VCM_BSPL4
 - Vhal class, [163](#)
- VCM_CHARGE
 - Vhal class, [163](#)
- VCM_INDUCED
 - Vhal class, [163](#)
- VCM_NLINDUCED
 - Vhal class, [163](#)
- VCM_PERMANENT
 - Vhal class, [163](#)
- VCM_TRIL
 - Vhal class, [163](#)
- VDF_AVS
 - Vhal class, [164](#)
- VDF_DX
 - Vhal class, [164](#)
- VDF_FLAT
 - Vhal class, [164](#)
- VDF_GZ
 - Vhal class, [164](#)
- VDF_MCSF
 - Vhal class, [164](#)
- VDF_UHBD
 - Vhal class, [164](#)
- VDT_ATOMPOT
 - Vhal class, [164](#)
- VDT_CHARGE
 - Vhal class, [164](#)
- VDT_DIELX
 - Vhal class, [164](#)
- VDT_DIELY
 - Vhal class, [164](#)
- VDT_DIELZ
 - Vhal class, [164](#)
- VDT_EDENS
 - Vhal class, [164](#)
- VDT_IVDW
 - Vhal class, [164](#)
- VDT_KAPPA
 - Vhal class, [164](#)
- VDT_LAP
 - Vhal class, [164](#)
- VDT_NDENS
 - Vhal class, [164](#)
- VDT_POT
 - Vhal class, [164](#)
- VDT_QDENS
 - Vhal class, [164](#)
- VDT_SMOL
 - Vhal class, [164](#)
- VDT_SSPL
 - Vhal class, [164](#)
- VDT_VDW
 - Vhal class, [164](#)
- VG_T_DIRI
 - Vfetk class, [33](#)
- VG_T_PREV
 - Vfetk class, [33](#)
- VG_T_ZERO
 - Vfetk class, [33](#)
- VLT_BCG
 - Vfetk class, [33](#)
- VLT_CG
 - Vfetk class, [33](#)
- VLT_MG
 - Vfetk class, [33](#)
- VLT_SLU
 - Vfetk class, [33](#)
- VML_DIRICUBE
 - Vfetk class, [33](#)
- VML_EXTERNAL
 - Vfetk class, [33](#)

- VML_NEUMCUBE
 - Vfetk class, [33](#)
- VNT_ARC
 - Vfetk class, [34](#)
- VNT_INC
 - Vfetk class, [34](#)
- VNT_NEW
 - Vfetk class, [34](#)
- VPT_DIAG
 - Vfetk class, [34](#)
- VPT_IDEN
 - Vfetk class, [34](#)
- VPT_MG
 - Vfetk class, [34](#)
- VRC_FAILURE
 - Vhal class, [165](#)
- VRC_SUCCESS
 - Vhal class, [165](#)
- VSM_MOL
 - Vhal class, [166](#)
- VSM_MOLSMOOTH
 - Vhal class, [166](#)
- VSM_SPLINE
 - Vhal class, [166](#)
- VSM_SPLINE3
 - Vhal class, [166](#)
- VSM_SPLINE4
 - Vhal class, [166](#)
- Vacc class, [109](#)
- Valist class, [125](#)
- Vatom class, [132](#)
- Vcap class, [141](#)
- Vclist class, [144](#)
 - CLIST_AUTO_DOMAIN, [145](#)
 - CLIST_MANUAL_DOMAIN, [145](#)
- Vcsm class, [22](#)
- Vfetk class, [29](#)
 - VGT_DIRI, [33](#)
 - VGT_PREV, [33](#)
 - VGT_ZERO, [33](#)
 - VLT_BCG, [33](#)
 - VLT_CG, [33](#)
 - VLT_MG, [33](#)
 - VLT_SLU, [33](#)
 - VML_DIRICUBE, [33](#)
 - VML_EXTERNAL, [33](#)
 - VML_NEUMCUBE, [33](#)
 - VNT_ARC, [34](#)
 - VNT_INC, [34](#)
 - VNT_NEW, [34](#)
 - VPT_DIAG, [34](#)
 - VPT_IDEN, [34](#)
 - VPT_MG, [34](#)
- Vgreen class, [150](#)
- Vgrid class, [196](#)
- Vhal class, [158](#)
 - BCFL_FOCUS, [163](#)
 - BCFL_MAP, [163](#)
 - BCFL_MDH, [163](#)
 - BCFL_MEM, [163](#)
 - BCFL_SDH, [163](#)
 - BCFL_UNUSED, [163](#)
 - BCFL_ZERO, [163](#)
 - IPKEY_LPBE, [165](#)
 - IPKEY_NPBE, [165](#)
 - IPKEY_SMPBE, [165](#)
 - OUTPUT_FLAT, [165](#)
 - OUTPUT_NULL, [165](#)
 - PBE_LPBE, [165](#)
 - PBE_LRPBE, [165](#)
 - PBE_NPBE, [165](#)
 - PBE_SMPBE, [165](#)
 - VCM_BSPL2, [163](#)
 - VCM_BSPL4, [163](#)
 - VCM_CHARGE, [163](#)
 - VCM_INDUCED, [163](#)
 - VCM_NLINDUCED, [163](#)
 - VCM_PERMANENT, [163](#)
 - VCM_TRIL, [163](#)
 - VDF_AVS, [164](#)
 - VDF_DX, [164](#)
 - VDF_FLAT, [164](#)
 - VDF_GZ, [164](#)
 - VDF_MCSF, [164](#)
 - VDF_UHBD, [164](#)
 - VDI_ATOMPOT, [164](#)
 - VDI_CHARGE, [164](#)
 - VDI_DIELX, [164](#)
 - VDI_DIELY, [164](#)
 - VDI_DIELZ, [164](#)
 - VDI_EDENS, [164](#)
 - VDI_IVDW, [164](#)
 - VDI_KAPPA, [164](#)
 - VDI_LAP, [164](#)
 - VDI_NDENS, [164](#)
 - VDI_POT, [164](#)
 - VDI_QDENS, [164](#)
 - VDI_SMOL, [164](#)
 - VDI_SSPL, [164](#)
 - VDI_VDW, [164](#)
 - VRC_FAILURE, [165](#)
 - VRC_SUCCESS, [165](#)
 - VSM_MOL, [166](#)
 - VSM_MOLSMOOTH, [166](#)
 - VSM_SPLINE, [166](#)
 - VSM_SPLINE3, [166](#)
 - VSM_SPLINE4, [166](#)
- vmem

- Vparam, [441](#)
- Vmgrid class, [206](#)
- Vopot class, [211](#)
- Vpackmg
 - Vpmg class, [220](#)
- Vparam, [440](#)
 - vmem, [441](#)
- Vparam class, [168](#)
- Vpbe class, [178](#)
- Vpee class, [55](#)
- Vpmg class, [215](#)
 - bcolcomp, [218](#)
 - bcolcomp2, [218](#)
 - bcolcomp3, [219](#)
 - bcolcomp4, [219](#)
 - pcolcomp, [220](#)
 - Vpackmg, [220](#)
- Vpmgp class, [237](#)
- Vstring class, [192](#)
- Vunit class, [195](#)