

APBS

1.3.1

Generated by Doxygen 1.8.1.2

Thu Jul 19 2012 11:37:42

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 Todo List | 5 |
| 3 Deprecated List | 33 |
| 4 Bug List | 35 |
| 5 Module Index | 39 |
| 5.1 Modules | 39 |
| 6 Data Structure Index | 41 |
| 6.1 Data Structures | 41 |
| 7 File Index | 43 |
| 7.1 File List | 43 |
| 8 Module Documentation | 47 |
| 8.1 Vcsm class | 47 |
| 8.1.1 Detailed Description | 48 |
| 8.1.2 Function Documentation | 48 |
| 8.1.2.1 Gem_setExternalUpdateFunction | 48 |
| 8.1.2.2 Vcsm_ctor | 49 |
| 8.1.2.3 Vcsm_ctor2 | 50 |
| 8.1.2.4 Vcsm_dtor | 51 |
| 8.1.2.5 Vcsm_dtor2 | 51 |
| 8.1.2.6 Vcsm_getAtom | 52 |

| | | |
|----------|--|----|
| 8.1.2.7 | Vcsm_getAtomIndex | 53 |
| 8.1.2.8 | Vcsm_getNumberAtoms | 53 |
| 8.1.2.9 | Vcsm_getNumberSimplices | 54 |
| 8.1.2.10 | Vcsm_getSimplex | 54 |
| 8.1.2.11 | Vcsm_getSimplexIndex | 55 |
| 8.1.2.12 | Vcsm_getValist | 55 |
| 8.1.2.13 | Vcsm_init | 56 |
| 8.1.2.14 | Vcsm_memChk | 56 |
| 8.1.2.15 | Vcsm_update | 57 |
| 8.2 | Vfetk class | 59 |
| 8.2.1 | Detailed Description | 62 |
| 8.2.2 | Enumeration Type Documentation | 63 |
| 8.2.2.1 | eVfetk_GuessType | 63 |
| 8.2.2.2 | eVfetk_LsolvType | 63 |
| 8.2.2.3 | eVfetk_MeshLoad | 63 |
| 8.2.2.4 | eVfetk_NsolvType | 63 |
| 8.2.2.5 | eVfetk_PrecType | 64 |
| 8.2.3 | Function Documentation | 64 |
| 8.2.3.1 | Bmat_printHB | 64 |
| 8.2.3.2 | Vfetk_ctor | 65 |
| 8.2.3.3 | Vfetk_ctor2 | 66 |
| 8.2.3.4 | Vfetk_dqmEnergy | 67 |
| 8.2.3.5 | Vfetk_dtor | 68 |
| 8.2.3.6 | Vfetk_dtor2 | 68 |
| 8.2.3.7 | Vfetk_dumpLocalVar | 69 |
| 8.2.3.8 | Vfetk_energy | 69 |
| 8.2.3.9 | Vfetk_externalUpdateFunction | 70 |
| 8.2.3.10 | Vfetk_fillArray | 71 |
| 8.2.3.11 | Vfetk_genCube | 72 |
| 8.2.3.12 | Vfetk_getAM | 73 |
| 8.2.3.13 | Vfetk_getAtomColor | 73 |
| 8.2.3.14 | Vfetk_getGem | 74 |
| 8.2.3.15 | Vfetk_getSolution | 74 |
| 8.2.3.16 | Vfetk_getVcsm | 75 |
| 8.2.3.17 | Vfetk_getVpbe | 76 |
| 8.2.3.18 | Vfetk_loadGem | 76 |
| 8.2.3.19 | Vfetk_loadMesh | 76 |

| | | |
|----------|--------------------------------------|-----|
| 8.2.3.20 | Vfetk_memChk | 77 |
| 8.2.3.21 | Vfetk_PDE_bisectEdge | 78 |
| 8.2.3.22 | Vfetk_PDE_ctor | 78 |
| 8.2.3.23 | Vfetk_PDE_ctor2 | 80 |
| 8.2.3.24 | Vfetk_PDE_delta | 81 |
| 8.2.3.25 | Vfetk_PDE_DFu_wv | 83 |
| 8.2.3.26 | Vfetk_PDE_dtor | 83 |
| 8.2.3.27 | Vfetk_PDE_dtor2 | 84 |
| 8.2.3.28 | Vfetk_PDE_Fu | 85 |
| 8.2.3.29 | Vfetk_PDE_Fu_v | 86 |
| 8.2.3.30 | Vfetk_PDE_initAssemble | 86 |
| 8.2.3.31 | Vfetk_PDE_initElement | 87 |
| 8.2.3.32 | Vfetk_PDE_initFace | 88 |
| 8.2.3.33 | Vfetk_PDE_initPoint | 88 |
| 8.2.3.34 | Vfetk_PDE_Ju | 89 |
| 8.2.3.35 | Vfetk_PDE_mapBoundary | 90 |
| 8.2.3.36 | Vfetk_PDE_markSimplex | 91 |
| 8.2.3.37 | Vfetk_PDE_oneChart | 91 |
| 8.2.3.38 | Vfetk_PDE_simplexBasisForm | 92 |
| 8.2.3.39 | Vfetk_PDE_simplexBasisInit | 93 |
| 8.2.3.40 | Vfetk_PDE_u_D | 94 |
| 8.2.3.41 | Vfetk_PDE_u_T | 95 |
| 8.2.3.42 | Vfetk_qfEnergy | 95 |
| 8.2.3.43 | Vfetk_readMesh | 97 |
| 8.2.3.44 | Vfetk_setAtomColors | 97 |
| 8.2.3.45 | Vfetk_setParameters | 98 |
| 8.2.3.46 | Vfetk_write | 98 |
| 8.3 | Vpee class | 100 |
| 8.3.1 | Detailed Description | 100 |
| 8.3.2 | Function Documentation | 101 |
| 8.3.2.1 | Vpee_ctor | 101 |
| 8.3.2.2 | Vpee_ctor2 | 101 |
| 8.3.2.3 | Vpee_dtor | 102 |
| 8.3.2.4 | Vpee_dtor2 | 102 |
| 8.3.2.5 | Vpee_markRefine | 102 |
| 8.3.2.6 | Vpee_numSS | 103 |
| 8.4 | APOLparm class | 105 |

| | | |
|---------|--|-----|
| 8.4.1 | Detailed Description | 106 |
| 8.4.2 | Enumeration Type Documentation | 106 |
| 8.4.2.1 | eAPOLparm_calcEnergy | 106 |
| 8.4.2.2 | eAPOLparm_calcForce | 106 |
| 8.4.2.3 | eAPOLparm_doCalc | 107 |
| 8.4.3 | Function Documentation | 107 |
| 8.4.3.1 | APOLparm_check | 107 |
| 8.4.3.2 | APOLparm_copy | 107 |
| 8.4.3.3 | APOLparm_ctor | 108 |
| 8.4.3.4 | APOLparm_ctor2 | 109 |
| 8.4.3.5 | APOLparm_dtor | 109 |
| 8.4.3.6 | APOLparm_dtor2 | 110 |
| 8.5 | FEMparm class | 111 |
| 8.5.1 | Detailed Description | 112 |
| 8.5.2 | Typedef Documentation | 112 |
| 8.5.2.1 | FEMparm_EtolType | 112 |
| 8.5.3 | Enumeration Type Documentation | 112 |
| 8.5.3.1 | eFEMparm_CalcType | 112 |
| 8.5.3.2 | eFEMparm_EstType | 112 |
| 8.5.3.3 | eFEMparm_EtolType | 113 |
| 8.5.4 | Function Documentation | 113 |
| 8.5.4.1 | FEMparm_check | 113 |
| 8.5.4.2 | FEMparm_copy | 114 |
| 8.5.4.3 | FEMparm_ctor | 114 |
| 8.5.4.4 | FEMparm_ctor2 | 115 |
| 8.5.4.5 | FEMparm_dtor | 116 |
| 8.5.4.6 | FEMparm_dtor2 | 117 |
| 8.6 | MGparm class | 118 |
| 8.6.1 | Detailed Description | 119 |
| 8.6.2 | Enumeration Type Documentation | 119 |
| 8.6.2.1 | eMGparm_CalcType | 119 |
| 8.6.2.2 | eMGparm_CentMeth | 120 |
| 8.6.3 | Function Documentation | 120 |
| 8.6.3.1 | APOLparm_parseToken | 120 |
| 8.6.3.2 | FEMparm_parseToken | 121 |
| 8.6.3.3 | MGparm_check | 121 |
| 8.6.3.4 | MGparm_copy | 122 |

| | | |
|----------|--|-----|
| 8.6.3.5 | MGparm_ctor | 122 |
| 8.6.3.6 | MGparm_ctor2 | 123 |
| 8.6.3.7 | MGparm_dtor | 124 |
| 8.6.3.8 | MGparm_dtor2 | 125 |
| 8.6.3.9 | MGparm_getCenterX | 125 |
| 8.6.3.10 | MGparm_getCenterY | 125 |
| 8.6.3.11 | MGparm_getCenterZ | 126 |
| 8.6.3.12 | MGparm_getHx | 126 |
| 8.6.3.13 | MGparm_getHy | 126 |
| 8.6.3.14 | MGparm_getHz | 127 |
| 8.6.3.15 | MGparm_getNx | 127 |
| 8.6.3.16 | MGparm_getNy | 127 |
| 8.6.3.17 | MGparm_getNz | 128 |
| 8.6.3.18 | MGparm_parseToken | 128 |
| 8.6.3.19 | MGparm_setCenterX | 129 |
| 8.6.3.20 | MGparm_setCenterY | 129 |
| 8.6.3.21 | MGparm_setCenterZ | 130 |
| 8.7 | NOsh class | 131 |
| 8.7.1 | Detailed Description | 133 |
| 8.7.2 | Enumeration Type Documentation | 133 |
| 8.7.2.1 | eNOsh_CalcType | 133 |
| 8.7.2.2 | eNOsh_MolFormat | 134 |
| 8.7.2.3 | eNOsh_ParmFormat | 134 |
| 8.7.2.4 | eNOsh_PrintType | 134 |
| 8.7.3 | Function Documentation | 134 |
| 8.7.3.1 | NOsh_apol2calc | 134 |
| 8.7.3.2 | NOsh_calc_copy | 135 |
| 8.7.3.3 | NOsh_calc_ctor | 135 |
| 8.7.3.4 | NOsh_calc_dtor | 136 |
| 8.7.3.5 | NOsh_ctor | 137 |
| 8.7.3.6 | NOsh_ctor2 | 138 |
| 8.7.3.7 | NOsh_dtor | 139 |
| 8.7.3.8 | NOsh_dtor2 | 139 |
| 8.7.3.9 | NOsh_elec2calc | 140 |
| 8.7.3.10 | NOsh_elecname | 140 |
| 8.7.3.11 | NOsh_getCalc | 141 |
| 8.7.3.12 | NOsh_getChargefmt | 141 |

| | | |
|----------|--------------------------------|-----|
| 8.7.3.13 | NOsh_getChargepath | 142 |
| 8.7.3.14 | NOsh_getDielfmt | 142 |
| 8.7.3.15 | NOsh_getDielXpath | 142 |
| 8.7.3.16 | NOsh_getDielYpath | 143 |
| 8.7.3.17 | NOsh_getDielZpath | 143 |
| 8.7.3.18 | NOsh_getKappafmt | 144 |
| 8.7.3.19 | NOsh_getKappapath | 144 |
| 8.7.3.20 | NOsh_getMolpath | 144 |
| 8.7.3.21 | NOsh_getPotfmt | 145 |
| 8.7.3.22 | NOsh_getPotpath | 145 |
| 8.7.3.23 | NOsh_parseInput | 145 |
| 8.7.3.24 | NOsh_parseInputFile | 146 |
| 8.7.3.25 | NOsh_printCalc | 147 |
| 8.7.3.26 | NOsh_printNarg | 147 |
| 8.7.3.27 | NOsh_printOp | 148 |
| 8.7.3.28 | NOsh_printWhat | 149 |
| 8.7.3.29 | NOsh_setupApolCalc | 149 |
| 8.7.3.30 | NOsh_setupElecCalc | 150 |
| 8.8 | PBEparm class | 151 |
| 8.8.1 | Detailed Description | 152 |
| 8.8.2 | Enumeration Type Documentation | 152 |
| 8.8.2.1 | ePBEparm_calcEnergy | 152 |
| 8.8.2.2 | ePBEparm_calcForce | 152 |
| 8.8.3 | Function Documentation | 153 |
| 8.8.3.1 | PBEparm_check | 153 |
| 8.8.3.2 | PBEparm_copy | 153 |
| 8.8.3.3 | PBEparm_ctor | 153 |
| 8.8.3.4 | PBEparm_ctor2 | 154 |
| 8.8.3.5 | PBEparm_dtor | 155 |
| 8.8.3.6 | PBEparm_dtor2 | 155 |
| 8.8.3.7 | PBEparm_getIonCharge | 156 |
| 8.8.3.8 | PBEparm_getIonConc | 156 |
| 8.8.3.9 | PBEparm_getIonRadius | 157 |
| 8.8.3.10 | PBEparm_parseToken | 157 |
| 8.9 | Vacc class | 159 |
| 8.9.1 | Detailed Description | 161 |
| 8.9.2 | Function Documentation | 161 |

| | | |
|----------|--|-----|
| 8.9.2.1 | Vacc_atomdSASA | 161 |
| 8.9.2.2 | Vacc_atomdSAV | 161 |
| 8.9.2.3 | Vacc_atomSASA | 162 |
| 8.9.2.4 | Vacc_atomSASPoints | 163 |
| 8.9.2.5 | Vacc_atomSurf | 163 |
| 8.9.2.6 | Vacc_ctor | 165 |
| 8.9.2.7 | Vacc_ctor2 | 166 |
| 8.9.2.8 | Vacc_dtor | 167 |
| 8.9.2.9 | Vacc_dtor2 | 168 |
| 8.9.2.10 | Vacc_fastMolAcc | 168 |
| 8.9.2.11 | Vacc_ivdwAcc | 169 |
| 8.9.2.12 | Vacc_memChk | 170 |
| 8.9.2.13 | Vacc_molAcc | 171 |
| 8.9.2.14 | Vacc_SASA | 172 |
| 8.9.2.15 | Vacc_splineAcc | 173 |
| 8.9.2.16 | Vacc_splineAccAtom | 174 |
| 8.9.2.17 | Vacc_splineAccGrad | 175 |
| 8.9.2.18 | Vacc_splineAccGradAtomNorm | 176 |
| 8.9.2.19 | Vacc_splineAccGradAtomNorm3 | 177 |
| 8.9.2.20 | Vacc_splineAccGradAtomNorm4 | 178 |
| 8.9.2.21 | Vacc_splineAccGradAtomUnnorm | 179 |
| 8.9.2.22 | Vacc_totalAtomdSASA | 179 |
| 8.9.2.23 | Vacc_totalAtomdSAV | 180 |
| 8.9.2.24 | Vacc_totalSASA | 181 |
| 8.9.2.25 | Vacc_totalSAV | 182 |
| 8.9.2.26 | Vacc_vdwAcc | 183 |
| 8.9.2.27 | Vacc_wcaEnergy | 184 |
| 8.9.2.28 | Vacc_wcaEnergyAtom | 185 |
| 8.9.2.29 | Vacc_wcaForceAtom | 186 |
| 8.9.2.30 | VaccSurf_ctor | 186 |
| 8.9.2.31 | VaccSurf_ctor2 | 187 |
| 8.9.2.32 | VaccSurf_dtor | 188 |
| 8.9.2.33 | VaccSurf_dtor2 | 188 |
| 8.9.2.34 | VaccSurf_refSphere | 189 |
| 8.10 | Valist class | 191 |
| 8.10.1 | Detailed Description | 192 |
| 8.10.2 | Function Documentation | 192 |

| | | |
|-----------|--------------------------------|-----|
| 8.10.2.1 | Valist_ctor | 192 |
| 8.10.2.2 | Valist_ctor2 | 192 |
| 8.10.2.3 | Valist_dtor | 193 |
| 8.10.2.4 | Valist_dtor2 | 193 |
| 8.10.2.5 | Valist_getAtom | 194 |
| 8.10.2.6 | Valist_getAtomList | 195 |
| 8.10.2.7 | Valist_getCenterX | 196 |
| 8.10.2.8 | Valist_getCenterY | 196 |
| 8.10.2.9 | Valist_getCenterZ | 196 |
| 8.10.2.10 | Valist_getNumberAtoms | 197 |
| 8.10.2.11 | Valist_getStatistics | 198 |
| 8.10.2.12 | Valist_memChk | 199 |
| 8.10.2.13 | Valist_readPDB | 199 |
| 8.10.2.14 | Valist_readPQR | 200 |
| 8.10.2.15 | Valist_readXML | 201 |
| 8.11 | Vatom class | 203 |
| 8.11.1 | Detailed Description | 204 |
| 8.11.2 | Macro Definition Documentation | 204 |
| 8.11.2.1 | VMAX_RECLEN | 204 |
| 8.11.3 | Function Documentation | 205 |
| 8.11.3.1 | Vatom_copyFrom | 205 |
| 8.11.3.2 | Vatom_copyTo | 205 |
| 8.11.3.3 | Vatom_ctor | 206 |
| 8.11.3.4 | Vatom_ctor2 | 206 |
| 8.11.3.5 | Vatom_dtor | 207 |
| 8.11.3.6 | Vatom_dtor2 | 207 |
| 8.11.3.7 | Vatom_getAtomID | 208 |
| 8.11.3.8 | Vatom_getAtomName | 208 |
| 8.11.3.9 | Vatom_getCharge | 209 |
| 8.11.3.10 | Vatom_getEpsilon | 210 |
| 8.11.3.11 | Vatom_getPartID | 210 |
| 8.11.3.12 | Vatom_getPosition | 210 |
| 8.11.3.13 | Vatom_getRadius | 211 |
| 8.11.3.14 | Vatom_getResName | 212 |
| 8.11.3.15 | Vatom_memChk | 213 |
| 8.11.3.16 | Vatom_setAtomID | 213 |
| 8.11.3.17 | Vatom_setAtomName | 214 |

| | |
|--|-----|
| 8.11.3.18 <code>Vatom_setCharge</code> | 214 |
| 8.11.3.19 <code>Vatom_setEpsilon</code> | 215 |
| 8.11.3.20 <code>Vatom_setPartID</code> | 215 |
| 8.11.3.21 <code>Vatom_setPosition</code> | 216 |
| 8.11.3.22 <code>Vatom_setRadius</code> | 217 |
| 8.11.3.23 <code>Vatom_setResName</code> | 218 |
| 8.12 <code>Vcap</code> class | 219 |
| 8.12.1 Detailed Description | 219 |
| 8.12.2 Function Documentation | 219 |
| 8.12.2.1 <code>Vcap_cosh</code> | 219 |
| 8.12.2.2 <code>Vcap_exp</code> | 220 |
| 8.12.2.3 <code>Vcap_sinh</code> | 221 |
| 8.13 <code>Vclist</code> class | 222 |
| 8.13.1 Detailed Description | 223 |
| 8.13.2 Enumeration Type Documentation | 223 |
| 8.13.2.1 <code>eVclist_DomainMode</code> | 223 |
| 8.13.3 Function Documentation | 223 |
| 8.13.3.1 <code>Vclist_ctor</code> | 223 |
| 8.13.3.2 <code>Vclist_ctor2</code> | 224 |
| 8.13.3.3 <code>Vclist_dtor</code> | 225 |
| 8.13.3.4 <code>Vclist_dtor2</code> | 226 |
| 8.13.3.5 <code>Vclist_getCell</code> | 227 |
| 8.13.3.6 <code>Vclist_maxRadius</code> | 227 |
| 8.13.3.7 <code>Vclist_memChk</code> | 228 |
| 8.13.3.8 <code>VclistCell_ctor</code> | 228 |
| 8.13.3.9 <code>VclistCell_ctor2</code> | 229 |
| 8.13.3.10 <code>VclistCell_dtor</code> | 229 |
| 8.13.3.11 <code>VclistCell_dtor2</code> | 230 |
| 8.14 <code>Vgreen</code> class | 231 |
| 8.14.1 Detailed Description | 232 |
| 8.14.2 Function Documentation | 233 |
| 8.14.2.1 <code>Vgreen_coulomb</code> | 233 |
| 8.14.2.2 <code>Vgreen_coulomb_direct</code> | 234 |
| 8.14.2.3 <code>Vgreen_coulombD</code> | 235 |
| 8.14.2.4 <code>Vgreen_coulombD_direct</code> | 236 |
| 8.14.2.5 <code>Vgreen_ctor</code> | 237 |
| 8.14.2.6 <code>Vgreen_ctor2</code> | 238 |

| | | |
|-----------|--------------------------------|-----|
| 8.14.2.7 | Vgreen_dtor | 239 |
| 8.14.2.8 | Vgreen_dtor2 | 239 |
| 8.14.2.9 | Vgreen_getValist | 240 |
| 8.14.2.10 | Vgreen_helmholtz | 240 |
| 8.14.2.11 | Vgreen_helmholtzD | 241 |
| 8.14.2.12 | Vgreen_memChk | 242 |
| 8.15 | Vhal class | 243 |
| 8.15.1 | Detailed Description | 246 |
| 8.15.2 | Macro Definition Documentation | 246 |
| 8.15.2.1 | MAX_SPHERE PTS | 246 |
| 8.15.2.2 | VABORT_MSG0 | 246 |
| 8.15.2.3 | VABORT_MSG1 | 246 |
| 8.15.2.4 | VABORT_MSG2 | 247 |
| 8.15.2.5 | VAPBS_BACK | 247 |
| 8.15.2.6 | VAPBS_DOWN | 247 |
| 8.15.2.7 | VAPBS_FRONT | 248 |
| 8.15.2.8 | VAPBS_LEFT | 248 |
| 8.15.2.9 | VAPBS_RIGHT | 248 |
| 8.15.2.10 | VAPBS_UP | 248 |
| 8.15.2.11 | VASSERT_MSG0 | 248 |
| 8.15.2.12 | VASSERT_MSG1 | 249 |
| 8.15.2.13 | VASSERT_MSG2 | 249 |
| 8.15.2.14 | VEMBED | 250 |
| 8.15.2.15 | VFLOOR | 250 |
| 8.15.2.16 | VWARN_MSG0 | 250 |
| 8.15.2.17 | VWARN_MSG1 | 250 |
| 8.15.2.18 | VWARN_MSG2 | 251 |
| 8.15.3 | Enumeration Type Documentation | 251 |
| 8.15.3.1 | eVbcfl | 251 |
| 8.15.3.2 | eVchrg_Meth | 252 |
| 8.15.3.3 | eVchrg_Src | 252 |
| 8.15.3.4 | eVdata_Format | 253 |
| 8.15.3.5 | eVdata_Type | 253 |
| 8.15.3.6 | eVhal_IPKEYType | 254 |
| 8.15.3.7 | eVhal_PBEType | 254 |
| 8.15.3.8 | eVoutput_Format | 254 |
| 8.15.3.9 | eVrc_Codes | 254 |

| | |
|--|-----|
| 8.15.3.10 eVsol_Meth | 255 |
| 8.15.3.11 eVsurf_Meth | 255 |
| 8.16 Matrix wrapper class | 256 |
| 8.16.1 Detailed Description | 256 |
| 8.17 Vparam class | 257 |
| 8.17.1 Detailed Description | 258 |
| 8.17.2 Function Documentation | 259 |
| 8.17.2.1 readFlatFileLine | 259 |
| 8.17.2.2 readXMLFileAtom | 259 |
| 8.17.2.3 Vparam_AtomData_copyFrom | 260 |
| 8.17.2.4 Vparam_AtomData_copyTo | 261 |
| 8.17.2.5 Vparam_AtomData_ctor | 261 |
| 8.17.2.6 Vparam_AtomData_ctor2 | 262 |
| 8.17.2.7 Vparam_AtomData_dtor | 262 |
| 8.17.2.8 Vparam_AtomData_dtor2 | 263 |
| 8.17.2.9 Vparam_ctor | 263 |
| 8.17.2.10 Vparam_ctor2 | 264 |
| 8.17.2.11 Vparam_dtor | 264 |
| 8.17.2.12 Vparam_dtor2 | 265 |
| 8.17.2.13 Vparam_getAtomData | 266 |
| 8.17.2.14 Vparam_getResData | 267 |
| 8.17.2.15 Vparam_memChk | 268 |
| 8.17.2.16 Vparam_readFlatFile | 268 |
| 8.17.2.17 Vparam_readXMLFile | 269 |
| 8.17.2.18 Vparam_ResData_copyTo | 270 |
| 8.17.2.19 Vparam_ResData_ctor | 271 |
| 8.17.2.20 Vparam_ResData_ctor2 | 272 |
| 8.17.2.21 Vparam_ResData_dtor | 272 |
| 8.17.2.22 Vparam_ResData_dtor2 | 273 |
| 8.18 Vpbe class | 274 |
| 8.18.1 Detailed Description | 275 |
| 8.18.2 Function Documentation | 276 |
| 8.18.2.1 Vpbe_ctor | 276 |
| 8.18.2.2 Vpbe_ctor2 | 277 |
| 8.18.2.3 Vpbe_dtor | 278 |
| 8.18.2.4 Vpbe_dtor2 | 278 |
| 8.18.2.5 Vpbe_getBulkIonicStrength | 279 |

| | |
|--|-----|
| 8.18.2.6 <code>Vpbe_getCoulombEnergy1</code> | 280 |
| 8.18.2.7 <code>Vpbe_getDeblen</code> | 281 |
| 8.18.2.8 <code>Vpbe_getGamma</code> | 282 |
| 8.18.2.9 <code>Vpbe_getIons</code> | 282 |
| 8.18.2.10 <code>Vpbe_getLmem</code> | 283 |
| 8.18.2.11 <code>Vpbe_getMaxIonRadius</code> | 283 |
| 8.18.2.12 <code>Vpbe_getmembraneDiel</code> | 284 |
| 8.18.2.13 <code>Vpbe_getmemv</code> | 285 |
| 8.18.2.14 <code>Vpbe_getSoluteCenter</code> | 285 |
| 8.18.2.15 <code>Vpbe_getSoluteCharge</code> | 285 |
| 8.18.2.16 <code>Vpbe_getSoluteDiel</code> | 286 |
| 8.18.2.17 <code>Vpbe_getSoluteRadius</code> | 286 |
| 8.18.2.18 <code>Vpbe_getSoluteXlen</code> | 287 |
| 8.18.2.19 <code>Vpbe_getSoluteYlen</code> | 287 |
| 8.18.2.20 <code>Vpbe_getSoluteZlen</code> | 288 |
| 8.18.2.21 <code>Vpbe_getSolventDiel</code> | 288 |
| 8.18.2.22 <code>Vpbe_getSolventRadius</code> | 289 |
| 8.18.2.23 <code>Vpbe_getTemperature</code> | 289 |
| 8.18.2.24 <code>Vpbe_getVacc</code> | 290 |
| 8.18.2.25 <code>Vpbe_getValist</code> | 291 |
| 8.18.2.26 <code>Vpbe_getXkappa</code> | 292 |
| 8.18.2.27 <code>Vpbe_getZkappa2</code> | 292 |
| 8.18.2.28 <code>Vpbe_getZmagic</code> | 293 |
| 8.18.2.29 <code>Vpbe_getzmem</code> | 294 |
| 8.18.2.30 <code>Vpbe_memChk</code> | 295 |
| 8.19 <code>Vstring</code> class | 296 |
| 8.19.1 Detailed Description | 296 |
| 8.19.2 Function Documentation | 296 |
| 8.19.2.1 <code>Vstring_isdigit</code> | 296 |
| 8.19.2.2 <code>Vstring_strcasecmp</code> | 296 |
| 8.19.2.3 <code>Vstring_wrappedtext</code> | 298 |
| 8.20 <code>Vunit</code> class | 299 |
| 8.20.1 Detailed Description | 299 |
| 8.21 <code>Vgrid</code> class | 300 |
| 8.21.1 Detailed Description | 301 |
| 8.21.2 Function Documentation | 302 |
| 8.21.2.1 <code>Vgrid_ctor</code> | 302 |

| | | |
|-----------|----------------------------------|-----|
| 8.21.2.2 | Vgrid_ctor2 | 302 |
| 8.21.2.3 | Vgrid_curvature | 303 |
| 8.21.2.4 | Vgrid_dtor | 304 |
| 8.21.2.5 | Vgrid_dtor2 | 304 |
| 8.21.2.6 | Vgrid_gradient | 305 |
| 8.21.2.7 | Vgrid_integrate | 305 |
| 8.21.2.8 | Vgrid_memChk | 306 |
| 8.21.2.9 | Vgrid_normH1 | 306 |
| 8.21.2.10 | Vgrid_normL1 | 307 |
| 8.21.2.11 | Vgrid_normL2 | 307 |
| 8.21.2.12 | Vgrid_normLinf | 308 |
| 8.21.2.13 | Vgrid_readDX | 308 |
| 8.21.2.14 | Vgrid_readGZ | 309 |
| 8.21.2.15 | Vgrid_seminormH1 | 309 |
| 8.21.2.16 | Vgrid_value | 310 |
| 8.21.2.17 | Vgrid_writeDX | 311 |
| 8.21.2.18 | Vgrid_writeUHBD | 311 |
| 8.22 | Vmgrid class | 313 |
| 8.22.1 | Detailed Description | 314 |
| 8.22.2 | Function Documentation | 314 |
| 8.22.2.1 | Vmgrid_addGrid | 314 |
| 8.22.2.2 | Vmgrid_ctor | 314 |
| 8.22.2.3 | Vmgrid_ctor2 | 314 |
| 8.22.2.4 | Vmgrid_curvature | 315 |
| 8.22.2.5 | Vmgrid_dtor | 315 |
| 8.22.2.6 | Vmgrid_dtor2 | 315 |
| 8.22.2.7 | Vmgrid_getGridByNum | 316 |
| 8.22.2.8 | Vmgrid_getGridByPoint | 316 |
| 8.22.2.9 | Vmgrid_gradient | 316 |
| 8.22.2.10 | Vmgrid_value | 317 |
| 8.23 | Multi-grid class | 318 |
| 8.23.1 | Detailed Description | 318 |
| 8.24 | Vopot class | 319 |
| 8.24.1 | Detailed Description | 319 |
| 8.24.2 | Function Documentation | 320 |
| 8.24.2.1 | Vopot_ctor | 320 |
| 8.24.2.2 | Vopot_ctor2 | 320 |

| | | |
|-----------|--|-----|
| 8.24.2.3 | Vopot_curvature | 320 |
| 8.24.2.4 | Vopot_dtor | 321 |
| 8.24.2.5 | Vopot_dtor2 | 321 |
| 8.24.2.6 | Vopot_gradient | 321 |
| 8.24.2.7 | Vopot_pot | 322 |
| 8.25 | Vpmg class | 323 |
| 8.25.1 | Detailed Description | 326 |
| 8.25.2 | Function Documentation | 326 |
| 8.25.2.1 | bcolcomp | 326 |
| 8.25.2.2 | bcolcomp2 | 327 |
| 8.25.2.3 | bcolcomp3 | 330 |
| 8.25.2.4 | bcolcomp4 | 332 |
| 8.25.2.5 | pcolcomp | 334 |
| 8.25.2.6 | Vpackmg | 335 |
| 8.25.2.7 | Vpmg_ctor | 336 |
| 8.25.2.8 | Vpmg_ctor2 | 337 |
| 8.25.2.9 | Vpmg_dbDirectPolForce | 339 |
| 8.25.2.10 | Vpmg_dbForce | 339 |
| 8.25.2.11 | Vpmg_dbMutualPolForce | 340 |
| 8.25.2.12 | Vpmg_dbNLDirectPolForce | 341 |
| 8.25.2.13 | Vpmg_dbPermanentMultipoleForce | 341 |
| 8.25.2.14 | Vpmg_dielEnergy | 341 |
| 8.25.2.15 | Vpmg_dielGradNorm | 342 |
| 8.25.2.16 | Vpmg_dtor | 343 |
| 8.25.2.17 | Vpmg_dtor2 | 344 |
| 8.25.2.18 | Vpmg_energy | 344 |
| 8.25.2.19 | Vpmg_fieldSpline4 | 345 |
| 8.25.2.20 | Vpmg_fillArray | 346 |
| 8.25.2.21 | Vpmg_fillco | 346 |
| 8.25.2.22 | Vpmg_force | 348 |
| 8.25.2.23 | Vpmg_ibDirectPolForce | 350 |
| 8.25.2.24 | Vpmg_ibForce | 350 |
| 8.25.2.25 | Vpmg_ibMutualPolForce | 351 |
| 8.25.2.26 | Vpmg_ibNLDirectPolForce | 352 |
| 8.25.2.27 | Vpmg_ibPermanentMultipoleForce | 352 |
| 8.25.2.28 | Vpmg_memChk | 352 |
| 8.25.2.29 | Vpmg_printColComp | 353 |

| | |
|--|-----|
| 8.25.2.30 Vpmg_qfAtomEnergy | 354 |
| 8.25.2.31 Vpmg_qfDirectPolForce | 354 |
| 8.25.2.32 Vpmg_qfEnergy | 355 |
| 8.25.2.33 Vpmg_qfForce | 356 |
| 8.25.2.34 Vpmg_qfMutualPolForce | 358 |
| 8.25.2.35 Vpmg_qfNLDirectPolForce | 358 |
| 8.25.2.36 Vpmg_qfPermanentMultipoleEnergy | 358 |
| 8.25.2.37 Vpmg_qfPermanentMultipoleForce | 359 |
| 8.25.2.38 Vpmg_qmEnergy | 359 |
| 8.25.2.39 Vpmg_setPart | 360 |
| 8.25.2.40 Vpmg_solve | 361 |
| 8.25.2.41 Vpmg_solveLaplace | 363 |
| 8.25.2.42 Vpmg_unsetPart | 363 |
| 8.26 Vpmgp class | 365 |
| 8.26.1 Detailed Description | 365 |
| 8.26.2 Function Documentation | 366 |
| 8.26.2.1 Vpmgp_ctor | 366 |
| 8.26.2.2 Vpmgp_ctor2 | 366 |
| 8.26.2.3 Vpmgp_dtor | 366 |
| 8.26.2.4 Vpmgp_dtor2 | 367 |
| 8.26.2.5 Vpmgp_makeCoarse | 367 |
| 8.26.2.6 Vpmgp_size | 367 |
| 8.27 C translation of Holst group PMG code | 369 |
| 8.27.1 Detailed Description | 373 |
| 8.27.2 Macro Definition Documentation | 373 |
| 8.27.2.1 HARM02 | 373 |
| 8.27.2.2 MAXIONS | 374 |
| 8.27.3 Function Documentation | 375 |
| 8.27.3.1 Vaxrand | 375 |
| 8.27.3.2 Vazeros | 375 |
| 8.27.3.3 VbuildA | 376 |
| 8.27.3.4 Vbuildband | 381 |
| 8.27.3.5 Vbuildband1_27 | 384 |
| 8.27.3.6 Vbuildband1_7 | 388 |
| 8.27.3.7 VbuildG | 390 |
| 8.27.3.8 VbuildG_1 | 394 |
| 8.27.3.9 VbuildG_27 | 401 |

| | |
|--------------------------------------|-----|
| 8.27.3.10 VbuildG_7 | 409 |
| 8.27.3.11 Vbuildgaler0 | 417 |
| 8.27.3.12 Vbuildops | 420 |
| 8.27.3.13 VbuildP | 426 |
| 8.27.3.14 Vbuildstr | 429 |
| 8.27.3.15 Vc_vec | 430 |
| 8.27.3.16 Vc_vecpmg | 432 |
| 8.27.3.17 Vc_vecsmpbe | 434 |
| 8.27.3.18 Vcghs | 435 |
| 8.27.3.19 Vdc_vec | 439 |
| 8.27.3.20 Vdpbsl | 441 |
| 8.27.3.21 Vextrac | 443 |
| 8.27.3.22 VfboundPMG | 444 |
| 8.27.3.23 VfboundPMG00 | 446 |
| 8.27.3.24 Vfmvfas | 447 |
| 8.27.3.25 Vfnewton | 455 |
| 8.27.3.26 Vgetjac | 463 |
| 8.27.3.27 Vgsrb | 465 |
| 8.27.3.28 VinterpPMG | 470 |
| 8.27.3.29 Vipower | 471 |
| 8.27.3.30 Vmatvec | 478 |
| 8.27.3.31 Vmgdriv | 481 |
| 8.27.3.32 Vmgdriv2 | 487 |
| 8.27.3.33 Vmgsz | 493 |
| 8.27.3.34 Vmresid | 497 |
| 8.27.3.35 Vmvcs | 499 |
| 8.27.3.36 Vmvfas | 507 |
| 8.27.3.37 Vmypdefinitlpbe | 513 |
| 8.27.3.38 Vmypdefinitnpbe | 514 |
| 8.27.3.39 Vmypdefinitsmple | 515 |
| 8.27.3.40 Vnewdriv | 516 |
| 8.27.3.41 Vnewdriv2 | 523 |
| 8.27.3.42 Vnewton | 530 |
| 8.27.3.43 Vnmatvec | 536 |
| 8.27.3.44 Vnmresid | 538 |
| 8.27.3.45 Vnsmooth | 539 |
| 8.27.3.46 Vpower | 542 |

| | |
|---|------------|
| 8.27.3.47 Vprtmtd | 547 |
| 8.27.3.48 Vrestrc | 548 |
| 8.27.3.49 VsMOOTH | 550 |
| 8.27.3.50 Vxaxpy | 555 |
| 8.27.3.51 Vxcopy | 556 |
| 8.27.3.52 Vxcopy_large | 559 |
| 8.27.3.53 Vxcopy_small | 560 |
| 8.27.3.54 Vxdot | 560 |
| 8.27.3.55 Vxnrm1 | 561 |
| 8.27.3.56 Vxnrm2 | 562 |
| 8.27.3.57 Vxscal | 563 |
| 9 Data Structure Documentation | 565 |
| 9.1 sAPOLparm Struct Reference | 565 |
| 9.1.1 Detailed Description | 566 |
| 9.1.2 Field Documentation | 566 |
| 9.1.2.1 bconc | 566 |
| 9.1.2.2 calcenergy | 566 |
| 9.1.2.3 calcforce | 566 |
| 9.1.2.4 dpos | 566 |
| 9.1.2.5 gamma | 566 |
| 9.1.2.6 grid | 567 |
| 9.1.2.7 molid | 567 |
| 9.1.2.8 parsed | 567 |
| 9.1.2.9 press | 567 |
| 9.1.2.10 sasa | 567 |
| 9.1.2.11 sav | 567 |
| 9.1.2.12 sdens | 567 |
| 9.1.2.13 setbconc | 567 |
| 9.1.2.14 setcalcenergy | 568 |
| 9.1.2.15 setcalcforce | 568 |
| 9.1.2.16 setdpos | 568 |
| 9.1.2.17 setgamma | 568 |
| 9.1.2.18 setgrid | 568 |
| 9.1.2.19 setmolid | 569 |
| 9.1.2.20 setpress | 569 |
| 9.1.2.21 setsdens | 569 |

| | | |
|----------|-------------------------------------|-----|
| 9.1.2.22 | setsrad | 569 |
| 9.1.2.23 | setsrfm | 569 |
| 9.1.2.24 | setswin | 570 |
| 9.1.2.25 | settemp | 570 |
| 9.1.2.26 | setwat | 570 |
| 9.1.2.27 | srad | 570 |
| 9.1.2.28 | srfm | 570 |
| 9.1.2.29 | swin | 570 |
| 9.1.2.30 | temp | 570 |
| 9.1.2.31 | totForce | 571 |
| 9.1.2.32 | watepsilon | 571 |
| 9.1.2.33 | watsigma | 571 |
| 9.1.2.34 | wcaEnergy | 571 |
| 9.2 | sFEMparm Struct Reference | 571 |
| 9.2.1 | Detailed Description | 572 |
| 9.2.2 | Field Documentation | 572 |
| 9.2.2.1 | akeyPRE | 572 |
| 9.2.2.2 | akeySOLVE | 572 |
| 9.2.2.3 | ekey | 572 |
| 9.2.2.4 | etol | 572 |
| 9.2.2.5 | glen | 572 |
| 9.2.2.6 | maxsolve | 573 |
| 9.2.2.7 | maxvert | 573 |
| 9.2.2.8 | meshID | 573 |
| 9.2.2.9 | parsed | 573 |
| 9.2.2.10 | pkey | 573 |
| 9.2.2.11 | setakeyPRE | 573 |
| 9.2.2.12 | setakeySOLVE | 573 |
| 9.2.2.13 | setekey | 573 |
| 9.2.2.14 | setetol | 573 |
| 9.2.2.15 | setglen | 574 |
| 9.2.2.16 | setmaxsolve | 574 |
| 9.2.2.17 | setmaxvert | 574 |
| 9.2.2.18 | settargetNum | 574 |
| 9.2.2.19 | settargetRes | 574 |
| 9.2.2.20 | settype | 574 |
| 9.2.2.21 | targetNum | 574 |

| | | |
|----------|------------------------------------|-----|
| 9.2.2.22 | targetRes | 574 |
| 9.2.2.23 | type | 575 |
| 9.2.2.24 | useMesh | 575 |
| 9.3 | sMGparm Struct Reference | 575 |
| 9.3.1 | Detailed Description | 576 |
| 9.3.2 | Field Documentation | 576 |
| 9.3.2.1 | async | 576 |
| 9.3.2.2 | ccenter | 576 |
| 9.3.2.3 | ccentmol | 577 |
| 9.3.2.4 | ccmeth | 577 |
| 9.3.2.5 | center | 577 |
| 9.3.2.6 | centmol | 577 |
| 9.3.2.7 | crlen | 577 |
| 9.3.2.8 | chgm | 577 |
| 9.3.2.9 | chgs | 577 |
| 9.3.2.10 | cmeth | 577 |
| 9.3.2.11 | dime | 578 |
| 9.3.2.12 | etol | 578 |
| 9.3.2.13 | fcenter | 578 |
| 9.3.2.14 | fcentmol | 578 |
| 9.3.2.15 | fcmeth | 578 |
| 9.3.2.16 | fglen | 578 |
| 9.3.2.17 | glen | 578 |
| 9.3.2.18 | grid | 578 |
| 9.3.2.19 | method | 578 |
| 9.3.2.20 | nlev | 579 |
| 9.3.2.21 | nonlintype | 579 |
| 9.3.2.22 | ofrac | 579 |
| 9.3.2.23 | parsed | 579 |
| 9.3.2.24 | partDisjCenter | 579 |
| 9.3.2.25 | partDisjLength | 579 |
| 9.3.2.26 | partDisjOwnSide | 579 |
| 9.3.2.27 | pdime | 579 |
| 9.3.2.28 | proc_rank | 580 |
| 9.3.2.29 | proc_size | 580 |
| 9.3.2.30 | setasync | 580 |
| 9.3.2.31 | setcgcen | 580 |

| | | |
|----------|------------------------|-----|
| 9.3.2.32 | setcglen | 580 |
| 9.3.2.33 | setchgm | 580 |
| 9.3.2.34 | setdime | 581 |
| 9.3.2.35 | setetol | 581 |
| 9.3.2.36 | setfgcent | 581 |
| 9.3.2.37 | setfglen | 581 |
| 9.3.2.38 | setgcent | 581 |
| 9.3.2.39 | setglen | 582 |
| 9.3.2.40 | setgrid | 582 |
| 9.3.2.41 | setmethod | 582 |
| 9.3.2.42 | setnlev | 582 |
| 9.3.2.43 | setnonlintype | 582 |
| 9.3.2.44 | setofrac | 583 |
| 9.3.2.45 | setpdime | 583 |
| 9.3.2.46 | setrank | 583 |
| 9.3.2.47 | setsize | 583 |
| 9.3.2.48 | setUseAqua | 583 |
| 9.3.2.49 | type | 584 |
| 9.3.2.50 | useAqua | 584 |
| 9.4 | sNOsh Struct Reference | 584 |
| 9.4.1 | Detailed Description | 585 |
| 9.4.2 | Field Documentation | 586 |
| 9.4.2.1 | alist | 586 |
| 9.4.2.2 | apol | 586 |
| 9.4.2.3 | apol2calc | 586 |
| 9.4.2.4 | apolname | 586 |
| 9.4.2.5 | bogus | 586 |
| 9.4.2.6 | calc | 586 |
| 9.4.2.7 | chargefmt | 586 |
| 9.4.2.8 | chargepath | 586 |
| 9.4.2.9 | dielfmt | 587 |
| 9.4.2.10 | dielXpath | 587 |
| 9.4.2.11 | dielYpath | 587 |
| 9.4.2.12 | dielZpath | 587 |
| 9.4.2.13 | elec | 587 |
| 9.4.2.14 | elec2calc | 587 |
| 9.4.2.15 | elecname | 587 |

| | | |
|----------|-----------------------------|-----|
| 9.4.2.16 | gotparm | 587 |
| 9.4.2.17 | ispara | 588 |
| 9.4.2.18 | kappafmt | 588 |
| 9.4.2.19 | kappapath | 588 |
| 9.4.2.20 | meshfmt | 588 |
| 9.4.2.21 | meshpath | 588 |
| 9.4.2.22 | molfmt | 588 |
| 9.4.2.23 | molpath | 588 |
| 9.4.2.24 | napol | 588 |
| 9.4.2.25 | ncalc | 588 |
| 9.4.2.26 | ncharge | 589 |
| 9.4.2.27 | ndiel | 589 |
| 9.4.2.28 | nelec | 589 |
| 9.4.2.29 | nkappa | 589 |
| 9.4.2.30 | nmesh | 589 |
| 9.4.2.31 | nmol | 589 |
| 9.4.2.32 | npot | 589 |
| 9.4.2.33 | nprint | 589 |
| 9.4.2.34 | parmfmt | 589 |
| 9.4.2.35 | parmpath | 590 |
| 9.4.2.36 | parsed | 590 |
| 9.4.2.37 | potfmt | 590 |
| 9.4.2.38 | potpath | 590 |
| 9.4.2.39 | printcalc | 590 |
| 9.4.2.40 | printnarg | 590 |
| 9.4.2.41 | printop | 590 |
| 9.4.2.42 | printwhat | 590 |
| 9.4.2.43 | proc_rank | 591 |
| 9.4.2.44 | proc_size | 591 |
| 9.5 | sNOsh_calc Struct Reference | 591 |
| 9.5.1 | Detailed Description | 591 |
| 9.5.2 | Field Documentation | 592 |
| 9.5.2.1 | apolparm | 592 |
| 9.5.2.2 | calctype | 592 |
| 9.5.2.3 | femparm | 592 |
| 9.5.2.4 | mgparm | 592 |
| 9.5.2.5 | pbeparm | 592 |

| | | |
|----------|-------------------------------------|-----|
| 9.6 | sPBEmprm Struct Reference | 592 |
| 9.6.1 | Detailed Description | 594 |
| 9.6.2 | Field Documentation | 594 |
| 9.6.2.1 | bcfl | 594 |
| 9.6.2.2 | calcenergy | 594 |
| 9.6.2.3 | calcforce | 594 |
| 9.6.2.4 | chargeMapID | 594 |
| 9.6.2.5 | dielMapID | 595 |
| 9.6.2.6 | ionc | 595 |
| 9.6.2.7 | ionq | 595 |
| 9.6.2.8 | ionr | 595 |
| 9.6.2.9 | kappaMapID | 595 |
| 9.6.2.10 | Lmem | 595 |
| 9.6.2.11 | mdie | 595 |
| 9.6.2.12 | memv | 595 |
| 9.6.2.13 | molid | 595 |
| 9.6.2.14 | nion | 596 |
| 9.6.2.15 | numwrite | 596 |
| 9.6.2.16 | parsed | 596 |
| 9.6.2.17 | pbetype | 596 |
| 9.6.2.18 | pdie | 596 |
| 9.6.2.19 | potMapID | 596 |
| 9.6.2.20 | sdens | 596 |
| 9.6.2.21 | sdie | 596 |
| 9.6.2.22 | setbcfl | 596 |
| 9.6.2.23 | setcalcenergy | 597 |
| 9.6.2.24 | setcalcforce | 597 |
| 9.6.2.25 | setion | 597 |
| 9.6.2.26 | setLmem | 597 |
| 9.6.2.27 | setmdie | 597 |
| 9.6.2.28 | setmemv | 597 |
| 9.6.2.29 | setmolid | 598 |
| 9.6.2.30 | setnion | 598 |
| 9.6.2.31 | setpbetype | 598 |
| 9.6.2.32 | setpdie | 598 |
| 9.6.2.33 | setsdens | 598 |
| 9.6.2.34 | setsdie | 599 |

| | | |
|----------|----------------------------|-----|
| 9.6.2.35 | setsmsize | 599 |
| 9.6.2.36 | setsmvolume | 599 |
| 9.6.2.37 | setsrad | 599 |
| 9.6.2.38 | setsrfm | 599 |
| 9.6.2.39 | setswin | 600 |
| 9.6.2.40 | settemp | 600 |
| 9.6.2.41 | setwritemat | 600 |
| 9.6.2.42 | setzmem | 600 |
| 9.6.2.43 | smsize | 600 |
| 9.6.2.44 | smvolume | 600 |
| 9.6.2.45 | srad | 600 |
| 9.6.2.46 | srfm | 601 |
| 9.6.2.47 | swin | 601 |
| 9.6.2.48 | temp | 601 |
| 9.6.2.49 | useChargeMap | 601 |
| 9.6.2.50 | useDielMap | 601 |
| 9.6.2.51 | useKappaMap | 601 |
| 9.6.2.52 | usePotMap | 601 |
| 9.6.2.53 | writefmt | 601 |
| 9.6.2.54 | writemat | 601 |
| 9.6.2.55 | writematflag | 602 |
| 9.6.2.56 | writematstem | 602 |
| 9.6.2.57 | writestem | 602 |
| 9.6.2.58 | writetype | 602 |
| 9.6.2.59 | zmem | 602 |
| 9.7 | sVacc Struct Reference | 602 |
| 9.7.1 | Detailed Description | 603 |
| 9.7.2 | Field Documentation | 604 |
| 9.7.2.1 | acc | 604 |
| 9.7.2.2 | alist | 604 |
| 9.7.2.3 | atomFlags | 604 |
| 9.7.2.4 | clist | 604 |
| 9.7.2.5 | mem | 604 |
| 9.7.2.6 | refSphere | 604 |
| 9.7.2.7 | surf | 604 |
| 9.7.2.8 | surf_density | 604 |
| 9.8 | sVaccSurf Struct Reference | 605 |

| | | |
|----------|--------------------------|-----|
| 9.8.1 | Detailed Description | 605 |
| 9.8.2 | Field Documentation | 605 |
| 9.8.2.1 | area | 605 |
| 9.8.2.2 | bpts | 605 |
| 9.8.2.3 | mem | 605 |
| 9.8.2.4 | npts | 605 |
| 9.8.2.5 | probe_radius | 606 |
| 9.8.2.6 | xpts | 606 |
| 9.8.2.7 | ypts | 606 |
| 9.8.2.8 | zpts | 606 |
| 9.9 | sValist Struct Reference | 606 |
| 9.9.1 | Detailed Description | 607 |
| 9.9.2 | Field Documentation | 607 |
| 9.9.2.1 | atoms | 607 |
| 9.9.2.2 | center | 607 |
| 9.9.2.3 | charge | 607 |
| 9.9.2.4 | maxcrd | 607 |
| 9.9.2.5 | maxrad | 607 |
| 9.9.2.6 | mincrd | 608 |
| 9.9.2.7 | number | 608 |
| 9.9.2.8 | vmem | 608 |
| 9.10 | sVatom Struct Reference | 608 |
| 9.10.1 | Detailed Description | 608 |
| 9.10.2 | Field Documentation | 609 |
| 9.10.2.1 | atomName | 609 |
| 9.10.2.2 | charge | 609 |
| 9.10.2.3 | epsilon | 609 |
| 9.10.2.4 | id | 609 |
| 9.10.2.5 | partID | 609 |
| 9.10.2.6 | position | 609 |
| 9.10.2.7 | radius | 609 |
| 9.10.2.8 | resName | 609 |
| 9.11 | sVclist Struct Reference | 610 |
| 9.11.1 | Detailed Description | 610 |
| 9.11.2 | Field Documentation | 611 |
| 9.11.2.1 | alist | 611 |
| 9.11.2.2 | cells | 611 |

| | | |
|-----------|------------------------------|-----|
| 9.11.2.3 | lower_corner | 611 |
| 9.11.2.4 | max_radius | 611 |
| 9.11.2.5 | mode | 611 |
| 9.11.2.6 | n | 611 |
| 9.11.2.7 | npts | 611 |
| 9.11.2.8 | spacs | 611 |
| 9.11.2.9 | upper_corner | 612 |
| 9.11.2.10 | vmem | 612 |
| 9.12 | sVclistCell Struct Reference | 612 |
| 9.12.1 | Detailed Description | 612 |
| 9.12.2 | Field Documentation | 613 |
| 9.12.2.1 | atoms | 613 |
| 9.12.2.2 | natoms | 613 |
| 9.13 | sVcsm Struct Reference | 613 |
| 9.13.1 | Detailed Description | 614 |
| 9.13.2 | Field Documentation | 614 |
| 9.13.2.1 | alist | 614 |
| 9.13.2.2 | gm | 614 |
| 9.13.2.3 | initFlag | 614 |
| 9.13.2.4 | msimp | 614 |
| 9.13.2.5 | natom | 615 |
| 9.13.2.6 | nqsm | 615 |
| 9.13.2.7 | nsimp | 615 |
| 9.13.2.8 | nsqm | 615 |
| 9.13.2.9 | qsm | 615 |
| 9.13.2.10 | sqm | 615 |
| 9.13.2.11 | vmem | 615 |
| 9.14 | sVfetk Struct Reference | 615 |
| 9.14.1 | Detailed Description | 617 |
| 9.14.2 | Field Documentation | 617 |
| 9.14.2.1 | am | 617 |
| 9.14.2.2 | aprx | 617 |
| 9.14.2.3 | csm | 617 |
| 9.14.2.4 | feparm | 617 |
| 9.14.2.5 | gm | 617 |
| 9.14.2.6 | gues | 617 |
| 9.14.2.7 | level | 618 |

| | |
|---|-----|
| 9.14.2.8 lkey | 618 |
| 9.14.2.9 lmax | 618 |
| 9.14.2.10 lprec | 618 |
| 9.14.2.11 ltol | 618 |
| 9.14.2.12 nkey | 618 |
| 9.14.2.13 nmax | 618 |
| 9.14.2.14 ntol | 618 |
| 9.14.2.15 pbe | 618 |
| 9.14.2.16 pbeparm | 619 |
| 9.14.2.17 pde | 619 |
| 9.14.2.18 pjac | 619 |
| 9.14.2.19 type | 619 |
| 9.14.2.20 vmem | 619 |
| 9.15 sVfetk_LocalVar Struct Reference | 619 |
| 9.15.1 Detailed Description | 621 |
| 9.15.2 Field Documentation | 621 |
| 9.15.2.1 A | 621 |
| 9.15.2.2 B | 621 |
| 9.15.2.3 d2W | 621 |
| 9.15.2.4 DB | 622 |
| 9.15.2.5 delta | 622 |
| 9.15.2.6 DFu_wv | 622 |
| 9.15.2.7 dU | 622 |
| 9.15.2.8 dW | 622 |
| 9.15.2.9 F | 622 |
| 9.15.2.10 fetk | 622 |
| 9.15.2.11 fType | 622 |
| 9.15.2.12 Fu_v | 622 |
| 9.15.2.13 green | 623 |
| 9.15.2.14 initGreen | 623 |
| 9.15.2.15 ionConc | 623 |
| 9.15.2.16 ionQ | 623 |
| 9.15.2.17 ionRadii | 623 |
| 9.15.2.18 ionstr | 623 |
| 9.15.2.19 jumpDiel | 623 |
| 9.15.2.20 nion | 623 |
| 9.15.2.21 nvec | 623 |

| | |
|---|-----|
| 9.15.2.22 nverts | 624 |
| 9.15.2.23 simp | 624 |
| 9.15.2.24 sType | 624 |
| 9.15.2.25 U | 624 |
| 9.15.2.26 u_D | 624 |
| 9.15.2.27 u_T | 624 |
| 9.15.2.28 verts | 624 |
| 9.15.2.29 vx | 624 |
| 9.15.2.30 W | 624 |
| 9.15.2.31 xq | 625 |
| 9.15.2.32 zkappa2 | 625 |
| 9.15.2.33 zks2 | 625 |
| 9.16 sVgreen Struct Reference | 625 |
| 9.16.1 Detailed Description | 626 |
| 9.16.2 Field Documentation | 626 |
| 9.16.2.1 alist | 626 |
| 9.16.2.2 np | 626 |
| 9.16.2.3 qp | 626 |
| 9.16.2.4 vmem | 626 |
| 9.16.2.5 xp | 626 |
| 9.16.2.6 yp | 627 |
| 9.16.2.7 zp | 627 |
| 9.17 sVgrid Struct Reference | 627 |
| 9.17.1 Detailed Description | 627 |
| 9.17.2 Field Documentation | 628 |
| 9.17.2.1 ctordata | 628 |
| 9.17.2.2 data | 628 |
| 9.17.2.3 hx | 628 |
| 9.17.2.4 hy | 628 |
| 9.17.2.5 hzed | 628 |
| 9.17.2.6 mem | 628 |
| 9.17.2.7 nx | 628 |
| 9.17.2.8 ny | 628 |
| 9.17.2.9 nz | 629 |
| 9.17.2.10 readdata | 629 |
| 9.17.2.11 xmax | 629 |
| 9.17.2.12 xmin | 629 |

| | | |
|-----------|------------------------------|-----|
| 9.17.2.13 | ymax | 629 |
| 9.17.2.14 | ymin | 629 |
| 9.17.2.15 | zmax | 629 |
| 9.17.2.16 | zmin | 629 |
| 9.18 | sVmgrid Struct Reference | 630 |
| 9.18.1 | Detailed Description | 630 |
| 9.18.2 | Field Documentation | 630 |
| 9.18.2.1 | grids | 630 |
| 9.18.2.2 | ngrids | 630 |
| 9.19 | sVmultigrid Struct Reference | 631 |
| 9.19.1 | Detailed Description | 632 |
| 9.19.2 | Field Documentation | 632 |
| 9.19.2.1 | coarse_count | 632 |
| 9.19.2.2 | coarse_solver | 632 |
| 9.19.2.3 | coarsen_method | 632 |
| 9.19.2.4 | discrete_method | 632 |
| 9.19.2.5 | fine_count | 633 |
| 9.19.2.6 | integer_workspace_limit | 633 |
| 9.19.2.7 | integer_workspace_size | 633 |
| 9.19.2.8 | ipcon | 633 |
| 9.19.2.9 | irite | 633 |
| 9.19.2.10 | level_count | 633 |
| 9.19.2.11 | nonlin | 634 |
| 9.19.2.12 | prolong_method | 634 |
| 9.19.2.13 | real_workspace_limit | 634 |
| 9.19.2.14 | real_workspace_size | 634 |
| 9.19.2.15 | smooth_method | 634 |
| 9.19.2.16 | stop_criterion | 634 |
| 9.19.2.17 | verbosity | 635 |
| 9.19.2.18 | x_size | 635 |
| 9.19.2.19 | y_size | 635 |
| 9.19.2.20 | z_size | 635 |
| 9.20 | sVopot Struct Reference | 635 |
| 9.20.1 | Detailed Description | 636 |
| 9.20.2 | Field Documentation | 637 |
| 9.20.2.1 | bcfl | 637 |
| 9.20.2.2 | mgrid | 637 |

| | | |
|-----------|-----------------------------------|-----|
| 9.20.2.3 | pbe | 637 |
| 9.21 | sVparam_AtomData Struct Reference | 637 |
| 9.21.1 | Detailed Description | 637 |
| 9.21.2 | Field Documentation | 638 |
| 9.21.2.1 | atomName | 638 |
| 9.21.2.2 | charge | 638 |
| 9.21.2.3 | epsilon | 638 |
| 9.21.2.4 | radius | 638 |
| 9.21.2.5 | resName | 638 |
| 9.22 | sVpbe Struct Reference | 639 |
| 9.22.1 | Detailed Description | 640 |
| 9.22.2 | Field Documentation | 640 |
| 9.22.2.1 | acc | 640 |
| 9.22.2.2 | alist | 640 |
| 9.22.2.3 | bulkIonicStrength | 641 |
| 9.22.2.4 | clist | 641 |
| 9.22.2.5 | deblen | 641 |
| 9.22.2.6 | ionConc | 641 |
| 9.22.2.7 | ionQ | 641 |
| 9.22.2.8 | ionRadii | 641 |
| 9.22.2.9 | ipkey | 641 |
| 9.22.2.10 | L | 641 |
| 9.22.2.11 | maxlonRadius | 641 |
| 9.22.2.12 | membraneDiel | 642 |
| 9.22.2.13 | numlon | 642 |
| 9.22.2.14 | param2Flag | 642 |
| 9.22.2.15 | paramFlag | 642 |
| 9.22.2.16 | smsize | 642 |
| 9.22.2.17 | smvolume | 642 |
| 9.22.2.18 | soluteCenter | 642 |
| 9.22.2.19 | soluteCharge | 642 |
| 9.22.2.20 | soluteDiel | 642 |
| 9.22.2.21 | soluteRadius | 643 |
| 9.22.2.22 | soluteXlen | 643 |
| 9.22.2.23 | soluteYlen | 643 |
| 9.22.2.24 | soluteZlen | 643 |
| 9.22.2.25 | solventDiel | 643 |

| | |
|---------------------------------------|-----|
| 9.22.2.26 solventRadius | 643 |
| 9.22.2.27 T | 643 |
| 9.22.2.28 V | 643 |
| 9.22.2.29 vmem | 643 |
| 9.22.2.30 xkappa | 644 |
| 9.22.2.31 z_mem | 644 |
| 9.22.2.32 zkappa2 | 644 |
| 9.22.2.33 zmagic | 644 |
| 9.23 sVpee Struct Reference | 644 |
| 9.23.1 Detailed Description | 644 |
| 9.23.2 Field Documentation | 645 |
| 9.23.2.1 gm | 645 |
| 9.23.2.2 killFlag | 645 |
| 9.23.2.3 killParam | 645 |
| 9.23.2.4 localPartCenter | 645 |
| 9.23.2.5 localPartID | 645 |
| 9.23.2.6 localPartRadius | 645 |
| 9.23.2.7 mem | 645 |
| 9.24 sVpmg Struct Reference | 646 |
| 9.24.1 Detailed Description | 648 |
| 9.24.2 Field Documentation | 648 |
| 9.24.2.1 a1cf | 648 |
| 9.24.2.2 a2cf | 648 |
| 9.24.2.3 a3cf | 648 |
| 9.24.2.4 ccf | 648 |
| 9.24.2.5 charge | 648 |
| 9.24.2.6 chargeMap | 648 |
| 9.24.2.7 chargeMeth | 649 |
| 9.24.2.8 chargeSrc | 649 |
| 9.24.2.9 dielXMap | 649 |
| 9.24.2.10 dielYMap | 649 |
| 9.24.2.11 dielZMap | 649 |
| 9.24.2.12 epsx | 649 |
| 9.24.2.13 epsy | 649 |
| 9.24.2.14 epsz | 649 |
| 9.24.2.15 extDiEnergy | 649 |
| 9.24.2.16 extNpEnergy | 650 |

| | |
|--|-----|
| 9.24.2.17 extQfEnergy | 650 |
| 9.24.2.18 extQmEnergy | 650 |
| 9.24.2.19 fcf | 650 |
| 9.24.2.20 filled | 650 |
| 9.24.2.21 gxfc | 650 |
| 9.24.2.22 gycf | 650 |
| 9.24.2.23 gzcf | 650 |
| 9.24.2.24 iparm | 650 |
| 9.24.2.25 iwork | 651 |
| 9.24.2.26 kappa | 651 |
| 9.24.2.27 kappaMap | 651 |
| 9.24.2.28 pbe | 651 |
| 9.24.2.29 pmgp | 651 |
| 9.24.2.30 pot | 651 |
| 9.24.2.31 potMap | 651 |
| 9.24.2.32 pvec | 651 |
| 9.24.2.33 rparm | 651 |
| 9.24.2.34 rwork | 652 |
| 9.24.2.35 splineWin | 652 |
| 9.24.2.36 surfMeth | 652 |
| 9.24.2.37 tcf | 652 |
| 9.24.2.38 u | 652 |
| 9.24.2.39 useChargeMap | 652 |
| 9.24.2.40 useDielXMap | 652 |
| 9.24.2.41 useDielYMap | 652 |
| 9.24.2.42 useDielZMap | 652 |
| 9.24.2.43 useKappaMap | 653 |
| 9.24.2.44 usePotMap | 653 |
| 9.24.2.45 vmem | 653 |
| 9.24.2.46 xf | 653 |
| 9.24.2.47 yf | 653 |
| 9.24.2.48 zf | 653 |
| 9.25 sVpmgp Struct Reference | 653 |
| 9.25.1 Detailed Description | 655 |
| 9.25.2 Field Documentation | 655 |
| 9.25.2.1 bcfl | 655 |
| 9.25.2.2 errtol | 655 |

| | |
|----------------------------|-----|
| 9.25.2.3 hx | 655 |
| 9.25.2.4 hy | 655 |
| 9.25.2.5 hzed | 655 |
| 9.25.2.6 iinfo | 655 |
| 9.25.2.7 ipcon | 656 |
| 9.25.2.8 iperf | 656 |
| 9.25.2.9 ipkey | 656 |
| 9.25.2.10 irite | 656 |
| 9.25.2.11 istop | 657 |
| 9.25.2.12 itmax | 657 |
| 9.25.2.13 key | 657 |
| 9.25.2.14 meth | 657 |
| 9.25.2.15 mgcoar | 658 |
| 9.25.2.16 mgdisc | 658 |
| 9.25.2.17 mgkey | 658 |
| 9.25.2.18 mgprol | 658 |
| 9.25.2.19 mgsmoo | 658 |
| 9.25.2.20 mgsolv | 659 |
| 9.25.2.21 n_ipc | 659 |
| 9.25.2.22 n_iz | 659 |
| 9.25.2.23 n_rpc | 659 |
| 9.25.2.24 narr | 659 |
| 9.25.2.25 narrc | 659 |
| 9.25.2.26 nc | 659 |
| 9.25.2.27 nf | 660 |
| 9.25.2.28 niwk | 660 |
| 9.25.2.29 nlev | 660 |
| 9.25.2.30 nonlin | 660 |
| 9.25.2.31 nrwk | 660 |
| 9.25.2.32 nu1 | 660 |
| 9.25.2.33 nu2 | 660 |
| 9.25.2.34 nx | 660 |
| 9.25.2.35 nxc | 661 |
| 9.25.2.36 ny | 661 |
| 9.25.2.37 nyc | 661 |
| 9.25.2.38 nz | 661 |
| 9.25.2.39 nzc | 661 |

| | |
|---|------------|
| 9.25.2.40 omegal | 661 |
| 9.25.2.41 omegan | 661 |
| 9.25.2.42 xcent | 661 |
| 9.25.2.43 xlen | 661 |
| 9.25.2.44 xmax | 662 |
| 9.25.2.45 xmin | 662 |
| 9.25.2.46 ycent | 662 |
| 9.25.2.47 ylen | 662 |
| 9.25.2.48 ymax | 662 |
| 9.25.2.49 ymin | 662 |
| 9.25.2.50 zcent | 662 |
| 9.25.2.51 zlen | 662 |
| 9.25.2.52 zmax | 662 |
| 9.25.2.53 zmin | 663 |
| 9.26 Vparam Struct Reference | 663 |
| 9.26.1 Detailed Description | 663 |
| 9.26.2 Field Documentation | 664 |
| 9.26.2.1 nResData | 664 |
| 9.26.2.2 resData | 664 |
| 9.26.2.3 vmem | 664 |
| 9.27 Vparam_ResData Struct Reference | 664 |
| 9.27.1 Detailed Description | 665 |
| 9.27.2 Field Documentation | 665 |
| 9.27.2.1 atomData | 665 |
| 9.27.2.2 name | 665 |
| 9.27.2.3 nAtomData | 665 |
| 9.27.2.4 vmem | 665 |
| 10 File Documentation | 667 |
| 10.1 doc/license/LICENSE.h File Reference | 667 |
| 10.1.1 Detailed Description | 667 |
| 10.2 LICENSE.h | 668 |
| 10.3 src/aaa_inc/apbs/apbs.h File Reference | 668 |
| 10.3.1 Detailed Description | 669 |
| 10.4 apbs.h | 670 |
| 10.5 src/aaa_lib/apbs_link.c File Reference | 671 |
| 10.5.1 Detailed Description | 671 |

| | |
|---|-----|
| 10.6 apbs_link.c | 673 |
| 10.7 src/fem/apbs/vcsm.h File Reference | 673 |
| 10.7.1 Detailed Description | 675 |
| 10.8 vcsm.h | 676 |
| 10.9 src/fem/apbs/vfetk.h File Reference | 678 |
| 10.9.1 Detailed Description | 682 |
| 10.10 vfetk.h | 683 |
| 10.11 src/fem/apbs/vpee.h File Reference | 688 |
| 10.11.1 Detailed Description | 690 |
| 10.12 vpee.h | 691 |
| 10.13 src/fem/dummy.c File Reference | 692 |
| 10.13.1 Detailed Description | 692 |
| 10.14 dummy.c | 693 |
| 10.15 src/fem/vcsm.c File Reference | 694 |
| 10.15.1 Detailed Description | 695 |
| 10.16 vcsm.c | 696 |
| 10.17 src/fem/vfetk.c File Reference | 702 |
| 10.17.1 Detailed Description | 706 |
| 10.17.2 Variable Documentation | 707 |
| 10.17.2.1 diriCubeString | 707 |
| 10.17.2.2 lgr_2DP1 | 707 |
| 10.17.2.3 lgr_2DP1x | 708 |
| 10.17.2.4 lgr_2DP1y | 708 |
| 10.17.2.5 lgr_2DP1z | 708 |
| 10.17.2.6 lgr_3DP1 | 708 |
| 10.17.2.7 lgr_3DP1x | 709 |
| 10.17.2.8 lgr_3DP1y | 709 |
| 10.17.2.9 lgr_3DP1z | 709 |
| 10.17.2.10 neumCubeString | 709 |
| 10.18 vfetk.c | 710 |
| 10.19 src/fem/vpee.c File Reference | 741 |
| 10.19.1 Detailed Description | 742 |
| 10.20 vpee.c | 743 |
| 10.21 src/generic/apbs/femparm.h File Reference | 750 |
| 10.21.1 Detailed Description | 752 |
| 10.22 femparm.h | 754 |
| 10.23 src/generic/apbs/mgparm.h File Reference | 755 |

| | |
|--|-----|
| 10.23.1 Detailed Description | 758 |
| 10.24mgparm.h | 759 |
| 10.25src/generic/apbs/nosh.h File Reference | 760 |
| 10.25.1 Detailed Description | 764 |
| 10.26nosh.h | 765 |
| 10.27src/generic/apbs/pbeparm.h File Reference | 768 |
| 10.27.1 Detailed Description | 769 |
| 10.28pbeparm.h | 770 |
| 10.29src/generic/apbs/vacc.h File Reference | 772 |
| 10.29.1 Detailed Description | 775 |
| 10.30vacc.h | 777 |
| 10.31src/generic/apbs/valist.h File Reference | 780 |
| 10.31.1 Detailed Description | 782 |
| 10.32valist.h | 783 |
| 10.33src/generic/apbs/vatom.h File Reference | 785 |
| 10.33.1 Detailed Description | 787 |
| 10.34vatom.h | 788 |
| 10.35src/generic/apbs/vcap.h File Reference | 790 |
| 10.35.1 Detailed Description | 791 |
| 10.36vcap.h | 792 |
| 10.37src/generic/apbs/vclist.h File Reference | 792 |
| 10.37.1 Detailed Description | 794 |
| 10.38vclist.h | 795 |
| 10.39src/generic/apbs/vgreen.h File Reference | 797 |
| 10.39.1 Detailed Description | 799 |
| 10.40vgreen.h | 799 |
| 10.41src/generic/apbs/vhal.h File Reference | 800 |
| 10.41.1 Detailed Description | 804 |
| 10.41.2 Macro Definition Documentation | 805 |
| 10.41.2.1 PRINT_DBL | 805 |
| 10.41.2.2 PRINT_FUNC | 805 |
| 10.41.2.3 PRINT_INT | 805 |
| 10.41.2.4 VCOPY | 805 |
| 10.41.2.5 VFILL | 805 |
| 10.41.2.6 VMESSAGE0 | 806 |
| 10.41.2.7 VMESSAGE1 | 806 |
| 10.41.2.8 VMESSAGE2 | 806 |

| | |
|--|-----|
| 10.42vhal.h | 806 |
| 10.43src/generic/apbs/vmatrix.h File Reference | 812 |
| 10.43.1 Detailed Description | 813 |
| 10.43.2 Macro Definition Documentation | 814 |
| 10.43.2.1 MAT2 | 814 |
| 10.43.2.2 MAT3 | 814 |
| 10.44vmatrix.h | 814 |
| 10.45src/generic/apbs/vparam.h File Reference | 815 |
| 10.45.1 Detailed Description | 817 |
| 10.46vparam.h | 818 |
| 10.47src/generic/apbs/vpbe.h File Reference | 820 |
| 10.47.1 Detailed Description | 822 |
| 10.48vpbe.h | 823 |
| 10.49src/generic/apbs/vstring.h File Reference | 826 |
| 10.49.1 Detailed Description | 827 |
| 10.50vstring.h | 828 |
| 10.51src/generic/apbs/vunit.h File Reference | 828 |
| 10.51.1 Detailed Description | 830 |
| 10.52vunit.h | 831 |
| 10.53src/generic/apolparm.c File Reference | 831 |
| 10.53.1 Detailed Description | 833 |
| 10.54apolparm.c | 834 |
| 10.55src/generic/femparm.c File Reference | 841 |
| 10.55.1 Detailed Description | 843 |
| 10.56femparm.c | 844 |
| 10.57src/generic/mgparm.c File Reference | 849 |
| 10.57.1 Detailed Description | 851 |
| 10.58mgparm.c | 852 |
| 10.59src/generic/nosh.c File Reference | 863 |
| 10.59.1 Detailed Description | 866 |
| 10.60nosh.c | 867 |
| 10.61src/generic/pbeparm.c File Reference | 896 |
| 10.61.1 Detailed Description | 898 |
| 10.62pbeparm.c | 899 |
| 10.63src/generic/vacc.c File Reference | 915 |
| 10.63.1 Detailed Description | 917 |
| 10.63.2 Function Documentation | 918 |

| | |
|---|------|
| 10.63.2.1 ivdwAccExclus | 918 |
| 10.63.2.2 splineAcc | 919 |
| 10.63.2.3 Vacc_allocate | 920 |
| 10.63.2.4 Vacc_storeParms | 921 |
| 10.64vacc.c | 922 |
| 10.65src/generic/valist.c File Reference | 946 |
| 10.65.1 Detailed Description | 947 |
| 10.66valist.c | 948 |
| 10.67src/generic/vatom.c File Reference | 960 |
| 10.67.1 Detailed Description | 961 |
| 10.68vatom.c | 962 |
| 10.69src/generic/vcap.c File Reference | 965 |
| 10.69.1 Detailed Description | 966 |
| 10.70vcap.c | 967 |
| 10.71src/generic/vclist.c File Reference | 968 |
| 10.71.1 Detailed Description | 970 |
| 10.72vclist.c | 971 |
| 10.73src/generic/vgreen.c File Reference | 977 |
| 10.73.1 Detailed Description | 978 |
| 10.74vgreen.c | 979 |
| 10.75src/generic/vparam.c File Reference | 986 |
| 10.75.1 Detailed Description | 988 |
| 10.76vparam.c | 989 |
| 10.77src/generic/vpbe.c File Reference | 998 |
| 10.77.1 Detailed Description | 1001 |
| 10.78vpbe.c | 1002 |
| 10.79src/generic/vstring.c File Reference | 1008 |
| 10.79.1 Detailed Description | 1009 |
| 10.80vstring.c | 1010 |
| 10.81src/mg/apbs/vgrid.h File Reference | 1012 |
| 10.81.1 Detailed Description | 1015 |
| 10.81.2 Function Documentation | 1016 |
| 10.81.2.1 Vgrid_writeGZ | 1016 |
| 10.82vgrid.h | 1016 |
| 10.83src/mg/apbs/vmgrid.h File Reference | 1017 |
| 10.83.1 Detailed Description | 1019 |
| 10.84vmgrid.h | 1020 |

| | |
|--|------|
| 10.85src/mg/apbs/vmultigrid.h File Reference | 1021 |
| 10.85.1 Detailed Description | 1023 |
| 10.85.2 Enumeration Type Documentation | 1024 |
| 10.85.2.1 eVmultigrid_boundary_method | 1024 |
| 10.85.2.2 eVmultigrid_coarse_solver | 1025 |
| 10.85.2.3 eVmultigrid_coarsen_method | 1025 |
| 10.85.2.4 eVmultigrid_discrete_method | 1025 |
| 10.85.2.5 eVmultigrid_iterate_method | 1025 |
| 10.85.2.6 eVmultigrid_linear | 1026 |
| 10.85.2.7 eVmultigrid_method | 1026 |
| 10.85.2.8 eVmultigrid_operator_analysis | 1026 |
| 10.85.2.9 eVmultigrid_prolong_method | 1027 |
| 10.85.2.10eVmultigrid_smooth_method | 1027 |
| 10.85.2.11eVmultigrid_stop_criterion | 1027 |
| 10.85.2.12eVmultigrid_verbosity | 1027 |
| 10.86vmultigrid.h | 1028 |
| 10.87src/mg/apbs/vopot.h File Reference | 1031 |
| 10.87.1 Detailed Description | 1032 |
| 10.88vopot.h | 1033 |
| 10.89src/mg/apbs/vpmg.h File Reference | 1034 |
| 10.89.1 Detailed Description | 1040 |
| 10.89.2 Function Documentation | 1041 |
| 10.89.2.1 bcCalc | 1041 |
| 10.89.2.2 bcf1 | 1042 |
| 10.89.2.3 bspline2 | 1042 |
| 10.89.2.4 bspline4 | 1043 |
| 10.89.2.5 d2bspline4 | 1043 |
| 10.89.2.6 d3bspline4 | 1044 |
| 10.89.2.7 dbspline2 | 1044 |
| 10.89.2.8 dbspline4 | 1045 |
| 10.89.2.9 fillcoCharge | 1046 |
| 10.89.2.10fillcoChargeMap | 1047 |
| 10.89.2.11fillcoChargeSpline1 | 1048 |
| 10.89.2.12fillcoChargeSpline2 | 1048 |
| 10.89.2.13fillcoCoef | 1049 |
| 10.89.2.14fillcoCoefMap | 1050 |
| 10.89.2.15fillcoCoefMol | 1051 |

| | |
|---|------|
| 10.89.2.16fillcoCoefMolDiel | 1052 |
| 10.89.2.17fillcoCoefMolDielNoSmooth | 1053 |
| 10.89.2.18fillcoCoefMolDielSmooth | 1054 |
| 10.89.2.19fillcoCoefMollon | 1055 |
| 10.89.2.20fillcoCoefSpline | 1056 |
| 10.89.2.21fillcoCoefSpline3 | 1058 |
| 10.89.2.22fillcoCoefSpline4 | 1059 |
| 10.89.2.23fillcoInducedDipole | 1060 |
| 10.89.2.24fillcoNLInducedDipole | 1060 |
| 10.89.2.25fillcoPermanentMultipole | 1060 |
| 10.89.2.26markSphere | 1062 |
| 10.89.2.27multipolebc | 1062 |
| 10.89.2.28qfForceSpline1 | 1063 |
| 10.89.2.29qfForceSpline2 | 1063 |
| 10.89.2.30qfForceSpline4 | 1064 |
| 10.89.2.31VFCHI4 | 1065 |
| 10.89.2.32Vpmg_polarizEnergy | 1066 |
| 10.89.2.33Vpmg_qfEnergyPoint | 1067 |
| 10.89.2.34Vpmg_qfEnergyVolume | 1068 |
| 10.89.2.35Vpmg_qmEnergySMPBE | 1069 |
| 10.89.2.36Vpmg_splineSelect | 1070 |
| 10.89.2.37zlapSolve | 1071 |
| 10.90vpmg.h | 1072 |
| 10.91src/mg/apbs/vpmgp.h File Reference | 1080 |
| 10.91.1 Detailed Description | 1082 |
| 10.92vpmgp.h | 1083 |
| 10.93src/mg/vgrid.c File Reference | 1085 |
| 10.93.1 Detailed Description | 1087 |
| 10.93.2 Function Documentation | 1088 |
| 10.93.2.1 Vgrid_writeGZ | 1088 |
| 10.94vgrid.c | 1088 |
| 10.95src/mg/vopot.c File Reference | 1107 |
| 10.95.1 Detailed Description | 1107 |
| 10.96vopot.c | 1109 |
| 10.97src/mg/vpmg.c File Reference | 1113 |
| 10.97.1 Detailed Description | 1116 |
| 10.97.2 Function Documentation | 1118 |

| | |
|---|------|
| 10.97.2.1 bcCalc | 1118 |
| 10.97.2.2 bcfl1 | 1119 |
| 10.97.2.3 bspline2 | 1119 |
| 10.97.2.4 bspline4 | 1120 |
| 10.97.2.5 d2bspline4 | 1120 |
| 10.97.2.6 d3bspline4 | 1121 |
| 10.97.2.7 dbspline2 | 1121 |
| 10.97.2.8 dbspline4 | 1122 |
| 10.97.2.9 fillcoCharge | 1123 |
| 10.97.2.10fillcoChargeMap | 1124 |
| 10.97.2.11fillcoChargeSpline1 | 1125 |
| 10.97.2.12fillcoChargeSpline2 | 1125 |
| 10.97.2.13fillcoCoef | 1126 |
| 10.97.2.14fillcoCoefMap | 1127 |
| 10.97.2.15fillcoCoefMol | 1128 |
| 10.97.2.16fillcoCoefMolDiel | 1129 |
| 10.97.2.17fillcoCoefMolDielNoSmooth | 1130 |
| 10.97.2.18fillcoCoefMolDielSmooth | 1131 |
| 10.97.2.19fillcoCoefMollon | 1132 |
| 10.97.2.20fillcoCoefSpline | 1133 |
| 10.97.2.21fillcoCoefSpline3 | 1135 |
| 10.97.2.22fillcoCoefSpline4 | 1136 |
| 10.97.2.23fillcoPermanentMultipole | 1137 |
| 10.97.2.24markSphere | 1138 |
| 10.97.2.25multipolebc | 1138 |
| 10.97.2.26qfForceSpline1 | 1139 |
| 10.97.2.27qfForceSpline2 | 1140 |
| 10.97.2.28qfForceSpline4 | 1141 |
| 10.97.2.29VFCHI4 | 1142 |
| 10.97.2.30Vpmg_polarizEnergy | 1142 |
| 10.97.2.31Vpmg_qfEnergyPoint | 1143 |
| 10.97.2.32Vpmg_qfEnergyVolume | 1144 |
| 10.97.2.33Vpmg_qmEnergySMPBE | 1145 |
| 10.97.2.34Vpmg_splineSelect | 1146 |
| 10.97.2.35zlapSolve | 1147 |
| 10.98vpmg.c | 1148 |
| 10.99src/mg/vpmgp.c File Reference | 1289 |

| | |
|--|------|
| 10.99.1 Detailed Description | 1290 |
| 10.100pmgp.c | 1292 |

Chapter 1

APBS Programmers Guide

APBS was written by Nathan A. Baker.

Additional contributing authors listed in the code documentation.

1.1 Table of Contents

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

1.2 License

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

Pacific Northwest National Laboratory

Additional contributing authors are listed in the code documentation.

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

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

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

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

Portions Copyright (c) 1995, Michael Holst.

All rights reserved.

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

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

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

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

1.3 Programming Style

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

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

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

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

would be declared as

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

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

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

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

would be declared as

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

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

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

in C++ would be

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

in Clean OO C.

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

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

1.4 Application programming interface documentation

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

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

Chapter 2

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.

Class `sVpmgp`

Value ipcon does not currently allow for preconditioning in PMG

Global `Vacc_fastMolAcc` (`Vacc *thee, double center[VAPBS_DIM], double radius`)

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

Global `Vacc_molAcc` (`Vacc *thee, double center[VAPBS_DIM], double radius`)

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

Global `Vfetk_dumpLocalVar` ()

This function is not thread-safe

Global `Vfetk_externalUpdateFunction` (`SS **simps, int num`)

This function is not thread-safe.

Global `Vfetk_fillArray` (`Vfetk *thee, Bvec *vec, Vdata_Type type`)

Several values of type are not implemented

Global `Vfetk_PDE_ctor` (`Vfetk *fetk`)

Not thread-safe

Global `Vfetk_PDE_ctor2` (`PDE *thee, Vfetk *fetk`)

Not thread-safe

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

This function is not thread-safe

Global `Vfetk_PDE_DFu_wv` (`PDE *thee, int key, double W[], double dW[][VAPBS_DIM], double V[], double d-V[][VAPBS_DIM]`)

This function is not thread-safe

Global `Vfetk_PDE_Fu` (`PDE *thee, int key, double F[]`)

This function is not thread-safe

This function is not implemented (sets error to zero)

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

This function is not thread-safe

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

This function is not thread-safe

Global `Vfetk_PDE_initFace` (`PDE *thee, int faceType, int chart, double tnvec[])`

This function is not thread-safe

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

This function is not thread-safe

This function uses pre-defined 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

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

Not implemented yet

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

Not implemented yet

Global `Vgrid_writeUHBD` (`Vgrid *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, char *title, double *pvec`)

This routine does not respect partition information

Global `Vpackmg` (`int *iparm, double *rparm, int *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 *mgsmo, 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 soluteDiels, double solventDiels, double solventRadius, int focusFlag, double sdens, double z_mem, double L, double membraneDiels, double V`)

The focusing flag is currently not used!!

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

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

Global `Vpmg_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:

| | |
|---|-----|
| Vcsm class | 47 |
| Vfetk class | 59 |
| Vpee class | 100 |
| APOLparm class | 105 |
| FEMparm class | 111 |
| MGparm class | 118 |
| NOSh class | 131 |
| PBEparm class | 151 |
| Vacc class | 159 |
| Valist class | 191 |
| Vatom class | 203 |
| Vcap class | 219 |
| Vclist class | 222 |
| Vgreen class | 231 |
| Vhal class | 243 |
| Matrix wrapper class | 256 |
| Vparam class | 257 |
| Vpbe class | 274 |
| Vstring class | 296 |
| Vunit class | 299 |
| Vgrid class | 300 |
| Vmgrid class | 313 |
| Multi-grid class | 318 |
| Vopot class | 319 |
| Vpmg class | 323 |
| Vpmgp class | 365 |
| C translation of Holst group PMG code | 369 |

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

| | | |
|---------------------------------|--|-----|
| sAPOLparm | Parameter structure for APOL-specific variables from input files | 565 |
| sFEMparm | Parameter structure for FEM-specific variables from input files | 571 |
| sMGparm | Parameter structure for MG-specific variables from input files | 575 |
| sNOsh | Class for parsing fixed format input files | 584 |
| sNOsh_calc | Calculation class for use when parsing fixed format input files | 591 |
| sPBEparm | Parameter structure for PBE variables from input files | 592 |
| sVacc | Oracle for solvent- and ion-accessibility around a biomolecule | 602 |
| sVaccSurf | Surface object list of per-atom surface points | 605 |
| sValist | Container class for list of atom objects | 606 |
| sVatom | Contains public data members for Vatom class/module | 608 |
| sVclist | Atom cell list | 610 |
| sVclistCell | Atom cell list cell | 612 |
| sVcsm | Charge-simplex map class | 613 |
| sVfetk | Contains public data members for Vfetk class/module | 615 |
| sVfetk_LocalVar | Vfetk LocalVar subclass | 619 |
| sVgreen | Contains public data members for Vgreen class/module | 625 |
| sVgrid | Electrostatic potential oracle for Cartesian mesh data | 627 |

| | | |
|----------------------------------|--|-----|
| sVmgrid | Multiresoltion oracle for Cartesian mesh data | 630 |
| sVmultigrid | Declaration of Vmultigrid class | 631 |
| sVopot | Electrostatic potential oracle for Cartesian mesh data | 635 |
| sVparam_AtomData | AtomData sub-class; stores atom data | 637 |
| sVpbe | Contains public data members for Vpbe class/module | 639 |
| sVpee | Contains public data members for Vpee class/module | 644 |
| sVpmg | Contains public data members for Vpmg class/module | 646 |
| sVpmgp | Contains public data members for Vpmgp class/module | 653 |
| Vparam | Reads and assigns charge/radii parameters | 663 |
| Vparam_ResData | ResData sub-class; stores residue data | 664 |

Chapter 6

File Index

6.1 File List

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

| | | |
|--|---|-----|
| doc/license/ LICENSE.h | APBS license | 668 |
| doc/programmer/ mainpage.h | | ?? |
| src/aaa_inc/apbs/ apbs.h | Top-level header for APBS | 670 |
| src/aaa_lib/ apbs_link.c | Autoconf linkage assistance for packages built on top of APBS | 673 |
| src/fem/ dummy.c | Give libtool something to do | 693 |
| src/fem/ vcsm.c | Class Vcsm methods | 696 |
| src/fem/ vfetk.c | Class Vfetk methods | 710 |
| src/fem/ vpee.c | Class Vpee methods | 743 |
| src/fem/apbs/ vcsm.h | Contains declarations for the Vcsm class | 676 |
| src/fem/apbs/ vfetk.h | Contains declarations for class Vfetk | 683 |
| src/fem/apbs/ vpee.h | Contains declarations for class Vpee | 691 |
| src/generic/ apolparm.c | Class APOLparm methods | 834 |
| src/generic/ femparm.c | Class FEMparm methods | 844 |
| src/generic/ mgparm.c | Class MGparm methods | 852 |
| src/generic/ nosh.c | Class NOsh methods | 867 |
| src/generic/ pbeparm.c | Class PBEparm methods | 899 |
| src/generic/ vacc.c | Class Vacc methods | 922 |

| | |
|--|------|
| src/generic/ valist.c | |
| Class Valist methods | 948 |
| src/generic/ vatom.c | |
| Class Vatom methods | 962 |
| src/generic/ vcap.c | |
| Class Vcap methods | 967 |
| src/generic/ vclist.c | |
| Class Vclist methods | 971 |
| src/generic/ vgreen.c | |
| Class Vgreen methods | 979 |
| src/generic/ vparam.c | |
| Class Vparam methods | 989 |
| src/generic/ vpbe.c | |
| Class Vpbe methods | 1002 |
| src/generic/ vstring.c | |
| Class Vstring methods | 1010 |
| src/generic/apbs/ apolparm.h | |
| Contains declarations for class APOLparm | 754 |
| src/generic/apbs/ mgparm.h | |
| Contains declarations for class MGparm | 759 |
| src/generic/apbs/ nosh.h | |
| Contains declarations for class NOsh | 765 |
| src/generic/apbs/ pbeparm.h | |
| Contains declarations for class PBEparm | 770 |
| src/generic/apbs/ vacc.h | |
| Contains declarations for class Vacc | 777 |
| src/generic/apbs/ valist.h | |
| Contains declarations for class Valist | 783 |
| src/generic/apbs/ vatom.h | |
| Contains declarations for class Vatom | 788 |
| src/generic/apbs/ vcap.h | |
| Contains declarations for class Vcap | 792 |
| src/generic/apbs/ vclist.h | |
| Contains declarations for class Vclist | 795 |
| src/generic/apbs/ vgreen.h | |
| Contains declarations for class Vgreen | 799 |
| src/generic/apbs/ vhal.h | |
| Contains generic macro definitions for APBS | 806 |
| src/generic/apbs/ vmatrix.h | |
| Contains inclusions for matrix data wrappers | 814 |
| src/generic/apbs/ vparam.h | |
| Contains declarations for class Vparam | 818 |
| src/generic/apbs/ vpbe.h | |
| Contains declarations for class Vpbe | 823 |
| src/generic/apbs/ vstring.h | |
| Contains declarations for class Vstring | 828 |
| src/generic/apbs/ vunit.h | |
| Contains a collection of useful constants and conversion factors | 831 |
| src/mg/ vgrid.c | |
| Class Vgrid methods | 1088 |
| src/mg/ vmultigrid.c | |
| | ?? |
| src/mg/ vopot.c | |
| Class Vopot methods | 1109 |

| | |
|--|------|
| src/mg/ vpmg.c | 1148 |
| Class Vpmg methods | |
| src/mg/ vpmgp.c | 1292 |
| Class Vpmgp methods | |
| src/mg/apbs/ vgrid.h | 1016 |
| Potential oracle for Cartesian mesh data | |
| src/mg/apbs/ vmgrid.h | 1020 |
| Multiresolution oracle for Cartesian mesh data | |
| src/mg/apbs/ vmultigrid.h | 1028 |
| src/mg/apbs/ vopot.h | |
| Potential oracle for Cartesian mesh data | 1033 |
| src/mg/apbs/ vpmg.h | |
| Contains declarations for class Vpmg | 1072 |
| src/mg/apbs/ vpmgp.h | |
| Contains declarations for class Vpmgp | 1083 |
| src/pmgc/ buildAd.c | ?? |
| src/pmgc/ buildBd.c | ?? |
| src/pmgc/ buildGd.c | ?? |
| src/pmgc/ buildPd.c | ?? |
| src/pmgc/ cgd.c | ?? |
| src/pmgc/ gsd.c | ?? |
| src/pmgc/ matvecd.c | ?? |
| src/pmgc/ mgcsd.c | ?? |
| src/pmgc/ mgdrvrd.c | ?? |
| src/pmgc/ mgfasd.c | ?? |
| src/pmgc/ mgsubd.c | ?? |
| src/pmgc/ mikpckd.c | ?? |
| src/pmgc/ mlinpckd.c | ?? |
| src/pmgc/ mypdec.c | ?? |
| src/pmgc/ newdrvrd.c | ?? |
| src/pmgc/ newtond.c | ?? |
| src/pmgc/ powerd.c | ?? |
| src/pmgc/ smoothd.c | ?? |
| src/pmgc/apbs/ buildAd.h | ?? |
| src/pmgc/apbs/ buildBd.h | ?? |
| src/pmgc/apbs/ buildGd.h | ?? |
| src/pmgc/apbs/ buildPd.h | ?? |
| src/pmgc/apbs/ cgd.h | ?? |
| src/pmgc/apbs/ gsd.h | ?? |
| src/pmgc/apbs/ matvecd.h | ?? |
| src/pmgc/apbs/ mgcsd.h | ?? |
| src/pmgc/apbs/ mgdrvrd.h | ?? |
| src/pmgc/apbs/ mgfasd.h | ?? |
| src/pmgc/apbs/ mgsubd.h | ?? |
| src/pmgc/apbs/ mikpckd.h | ?? |
| src/pmgc/apbs/ mlinpckd.h | ?? |
| src/pmgc/apbs/ mypdec.h | ?? |
| src/pmgc/apbs/ newdrvrd.h | ?? |
| src/pmgc/apbs/ newtond.h | ?? |
| src/pmgc/apbs/ powerd.h | ?? |
| src/pmgc/apbs/ smoothd.h | ?? |

Chapter 7

Module Documentation

7.1 Vcsm class

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

Files

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

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)

- VEXTERNC int `Vcsm_getNumberSimplices` (`Vcsm *thee`, int `iatom`)
Get number of simplices associated with an atom.
- VEXTERNC SS * `Vcsm_getSimplex` (`Vcsm *thee`, int `isimp`, int `iatom`)
Get particular simplex associated with an atom.
- VEXTERNC int `Vcsm_getSimplexIndex` (`Vcsm *thee`, int `isimp`, int `iatom`)
Get index particular simplex associated with an atom.
- VEXTERNC unsigned long int `Vcsm_memChk` (`Vcsm *thee`)
Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC `Vcsm * Vcsm_ctor` (`Valist *alist`, `Gem *gm`)
Construct Vcsm object.
- VEXTERNC int `Vcsm_ctor2` (`Vcsm *thee`, `Valist *alist`, `Gem *gm`)
FORTRAN stub to construct Vcsm object.
- VEXTERNC void `Vcsm_dtor` (`Vcsm **thee`)
Destroy Vcsm object.
- VEXTERNC void `Vcsm_dtor2` (`Vcsm *thee`)
FORTRAN stub to destroy Vcsm object.
- VEXTERNC void `Vcsm_init` (`Vcsm *thee`)
Initialize charge-simplex map with mesh and atom data.
- VEXTERNC int `Vcsm_update` (`Vcsm *thee`, `SS **simps`, int `num`)
Update the charge-simplex and simplex-charge maps after refinement.

7.1.1 Detailed Description

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

7.1.2 Function Documentation

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

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

Author

Nathan Baker

Parameters

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

Here is the caller graph for this function:



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

Construct Vcsm object.

Author

Nathan Baker

Note

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

Returns

Pointer to newly allocated Vcsm object

Parameters

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

FORTRAN stub to construct Vcsm object.

Author

Nathan Baker

Note

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

Returns

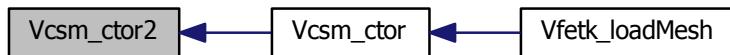
1 if successful, 0 otherwise

Parameters

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

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

Here is the caller graph for this function:



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

Destroy Vcsm object.

Author

Nathan Baker

Parameters

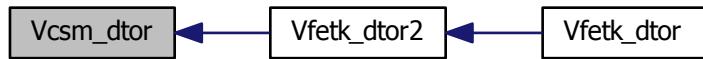
| | |
|-------------|--|
| <i>thee</i> | Pointer to memory location for Vcsm object |
|-------------|--|

Definition at line 296 of file [vcsm.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

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

FORTRAN stub to destroy Vcsm object.

Author

Nathan Baker

Parameters

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

Definition at line 304 of file [vcsm.c](#).

Here is the caller graph for this function:



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

Get particular atom associated with a simplex.

Author

Nathan Baker

Returns

Array of atoms associated with a simplex

Parameters

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

Get ID of particular atom in a simplex.

Author

Nathan Baker

Returns

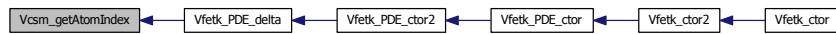
Index of atom in Valist object

Parameters

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

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

Here is the caller graph for this function:



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

Get number of atoms associated with a simplex.

Author

Nathan Baker

Returns

Number of atoms associated with a simplex

Parameters

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

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

Here is the caller graph for this function:



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

Get number of simplices associated with an atom.

Author

Nathan Baker

Returns

Number of simplices associated with an atom

Parameters

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

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

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

Get particular simplex associated with an atom.

Author

Nathan Baker

Returns

Pointer to simplex object

Parameters

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

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

Here is the caller graph for this function:



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

Get index particular simplex associated with an atom.

Author

Nathan Baker

Returns

Gem index of specified simplex

Parameters

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

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

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

Get atom list.

Author

Nathan Baker

Returns

Pointer to Valist atom list

Parameters

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

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

7.1.2.13 VEXTERNC void Vcsm_init (*Vcsm* * *thee*)

Initialize charge-simplex map with mesh and atom data.

Author

Nathan Baker

Note

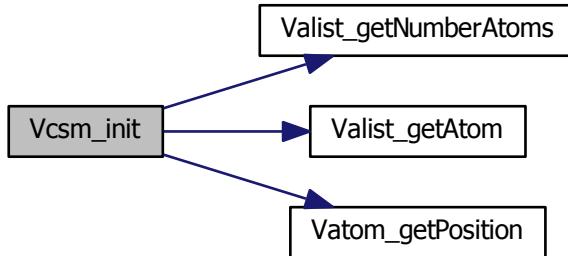
The initial mesh must be sufficiently coarse for the assignment procedures to be efficient

Parameters

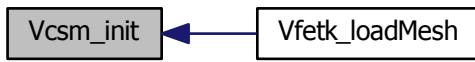
| | |
|-------------|------------------------|
| <i>thee</i> | The <i>Vcsm</i> object |
|-------------|------------------------|

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.2.14 VEXTERNC unsigned long int Vcsm_memChk (*Vcsm* * *thee*)

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

Author

Nathan Baker

Returns

The memory used by this structure and its contents in bytes

Parameters

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

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

Here is the caller graph for this function:



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

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

Author

Nathan Baker

Returns

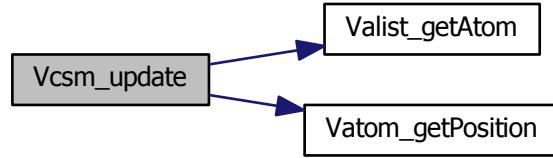
1 if successful, 0 otherwise

Parameters

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.2 Vfetk class

FEtk master class (interface between FEtk and APBS)

Files

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

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` { `VGT_ZERO` = 0, `VGT_DIRI` = 1, `VGT_PREV` = 2 }
Initial guess type.
- enum `eVfetk_PrecType` { `VPT_IDEN` = 0, `VPT_DIAG` = 1, `VPT_MG` = 2 }
Preconditioner type.

Functions

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

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

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

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

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

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

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

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

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

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.

7.2.1 Detailed Description

FEtk master class (interface between FEtk and APBS)

7.2.2 Enumeration Type Documentation

7.2.2.1 enum eVfetk_GuessType

Initial guess type.

Note

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

Enumerator:

VGT_ZERO Zero initial guess

VGT_DIRI Dirichlet boundary condition initial guess

VGT_PREV Previous level initial guess

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

7.2.2.2 enum eVfetk_LsolvType

Linear solver type.

Note

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

Enumerator:

VLT_SLU SuperLU direct solve

VLT_MG Multigrid

VLT(CG) Conjugate gradient

VLT_BCG BiCGStab

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

7.2.2.3 enum eVfetk_MeshLoad

Mesh loading operation.

Enumerator:

VML_DIRICUBE Dirichlet cube

VML_NEUMCUBE Neumann cube

VML_EXTERNAL External mesh (from socket)

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

7.2.2.4 enum eVfetk_NsolvType

Non-linear solver type.

Note

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

Enumerator:

VNT_NEW Newton solver

VNT_INC Incremental

VNT_ARC Psuedo-arclength

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

7.2.2.5 enum eVfetk_PrecType

Preconditioner type.

Note

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

Enumerator:

VPT_IDEN Identity matrix

VPT_DIAG Diagonal scaling

VPT_MG Multigrid

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

7.2.3 Function Documentation**7.2.3.1 VEXTERNC void Bmat_printHB (Bmat * *thee*, char * *fname*)**

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

Author

Stephen Bond

Note

This is a friend function of Bmat

Bug Hardwired to only handle the single block symmetric case.

Parameters

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

Definition at line 1031 of file [vfetk.c](#).

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

Constructor for Vfetk object.

Author

Nathan Baker

Returns

Pointer to newly allocated Vfetk object

Note

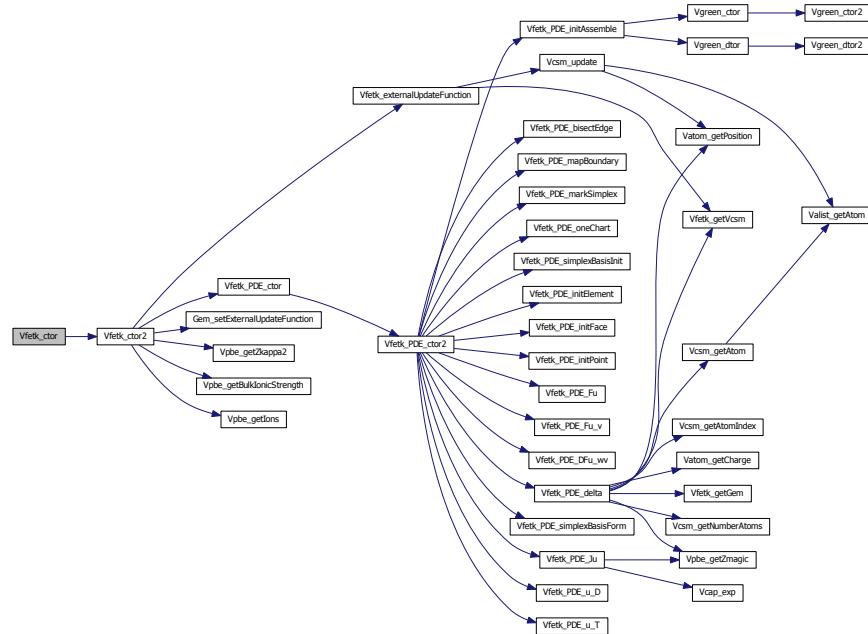
This sets up the Gem, AM, and Aprx FEtk objects but does not create a mesh. The easiest way to create a mesh is to then call Vfetk_genCube

Parameters

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

Definition at line 537 of file [vfetk.c](#).

Here is the call graph for this function:



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

FORTRAN stub constructor for Vfetk object.

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Note

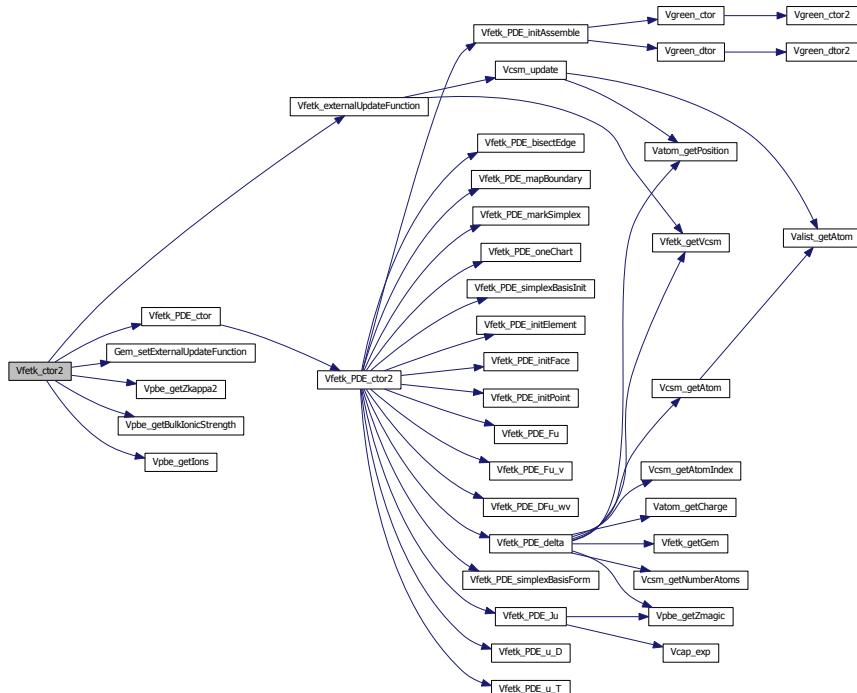
This sets up the Gem, AM, and Aprx FETk objects but does not create a mesh. The easiest way to create a mesh is to then call Vfetk_genCube

Parameters

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

Definition at line 550 of file [vfetk.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

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

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

for the NPBE and

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

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

Author

Nathan Baker

Parameters

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

Returns

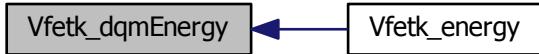
The "mobile charge" and "polarization" contributions to the electrostatic energy in units of $k_b T$.

Parameters

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

Definition at line 847 of file [vfetk.c](#).

Here is the caller graph for this function:



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

Object destructor.

Author

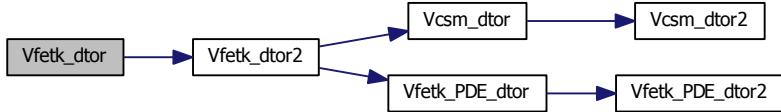
Nathan Baker

Parameters

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

Definition at line 630 of file [vfetk.c](#).

Here is the call graph for this function:



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

FORTRAN stub object destructor.

Author

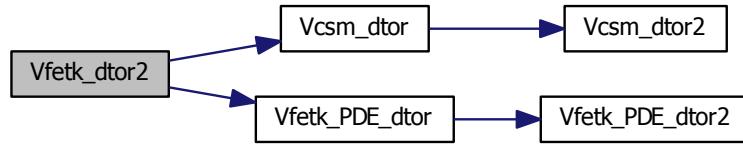
Nathan Baker

Parameters

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

Definition at line 638 of file [vfetk.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.7 VEXTERNC void Vfetk_dumpLocalVar()

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

Author

Nathan Baker

Bug This function is not thread-safe

Definition at line 2260 of file [vfetk.c](#).

7.2.3.8 VPUBLIC double Vfetk_energy(Vfetk * thee, int color, int nonlin)

Return the total electrostatic energy.

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

Author

Nathan Baker

Returns

Total electrostatic energy in units of $k_B T$.

< Total energy

<

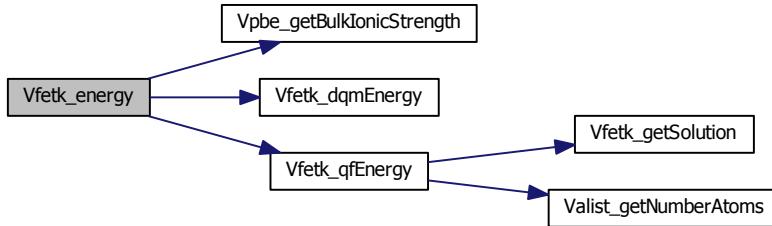
<

Parameters

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

Definition at line 698 of file [vfetk.c](#).

Here is the call graph for this function:



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

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

Author

Nathan Baker

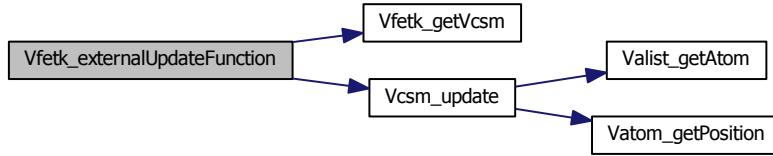
Bug This function is not thread-safe.

Parameters

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

Definition at line 2083 of file [vfetk.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

Fill an array with the specified data.

Author

Nathan Baker

Note

This function is thread-safe

Bug Several values of type are not implemented

Returns

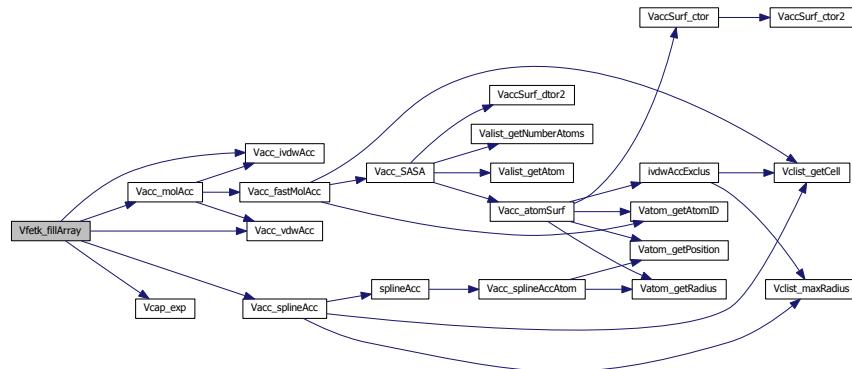
1 if successful, 0 otherwise

Parameters

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

Definition at line 2304 of file [vfetk.c](#).

Here is the call graph for this function:



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

Construct a rectangular mesh (in the current Vfetk object)

Author

Nathan Baker

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 890 of file [vfetk.c](#).

Here is the caller graph for this function:



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

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

Author

Nathan Baker

Returns

Pointer to the AM (algebra manager) object

Parameters

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

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

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

Get the partition information for a particular atom.

Author

Nathan Baker

Note

Friend function of Vatom

Returns

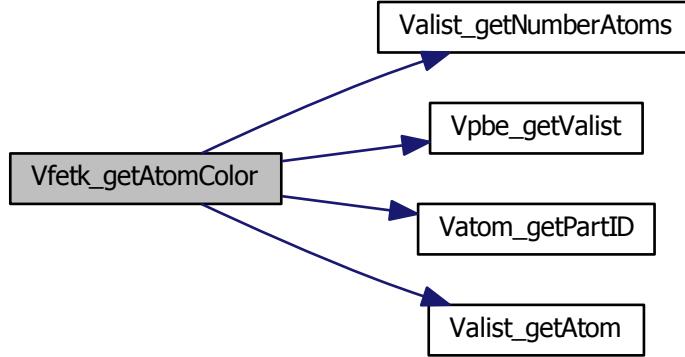
Partition ID

Parameters

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

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

Here is the call graph for this function:



7.2.3.14 VEXTERNC Gem* Vfetk_getGem (Vfetk * thee)

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

Author

Nathan Baker

Returns

Pointer to the Gem (grid manager) object

Parameters

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

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

Here is the caller graph for this function:



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

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

Author

Nathan Baker and Michael Holst

Note

The user is responsible for destroying the newly created array

Returns

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

Parameters

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

Definition at line 646 of file [vfetk.c](#).

Here is the caller graph for this function:



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

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

Author

Nathan Baker

Returns

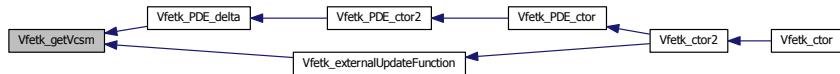
Pointer to the Vcsm (charge-simplex map) object

Parameters

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

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

Here is the caller graph for this function:



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

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

Author

Nathan Baker

Returns

Pointer to the Vpbe (PBE manager) object

Parameters

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

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

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

Load a Gem geometry manager object into Vfetk.

Author

Nathan Baker

Parameters

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

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

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

Author

Nathan Baker

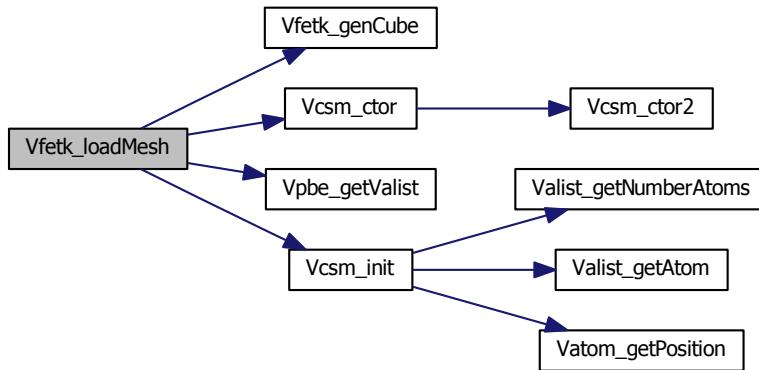
If we have an external mesh, load that external mesh from the provided socket. If we specify a non-external mesh type, we generate a new mesh cube based on templates. We then create and store a new Vcsm object in our Vfetk structure, which will carry the mesh data.

Parameters

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

Definition at line 985 of file [vfetk.c](#).

Here is the call graph for this function:

**7.2.3.20 VEXTERNC unsigned long int Vfetk_memChk (Vfetk * *thee*)**

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

Author

Nathan Baker

Returns

The memory used by this structure and its contents in bytes

Parameters

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

Definition at line 872 of file [vfetk.c](#).

Here is the call graph for this function:



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

Define the way manifold edges are bisected.

Author

Nathan Baker and Mike Holst

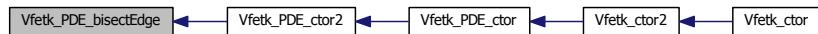
Note

This function is thread-safe.

Parameters

| | |
|-----------------|---|
| <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 |

Here is the caller graph for this function:



7.2.3.22 VEXTERNC PDE* Vfetk_PDE_ctor (Vfetk * ftk)

Constructs the FEtk PDE object.

Author

Nathan Baker

Returns

Newly-allocated PDE object

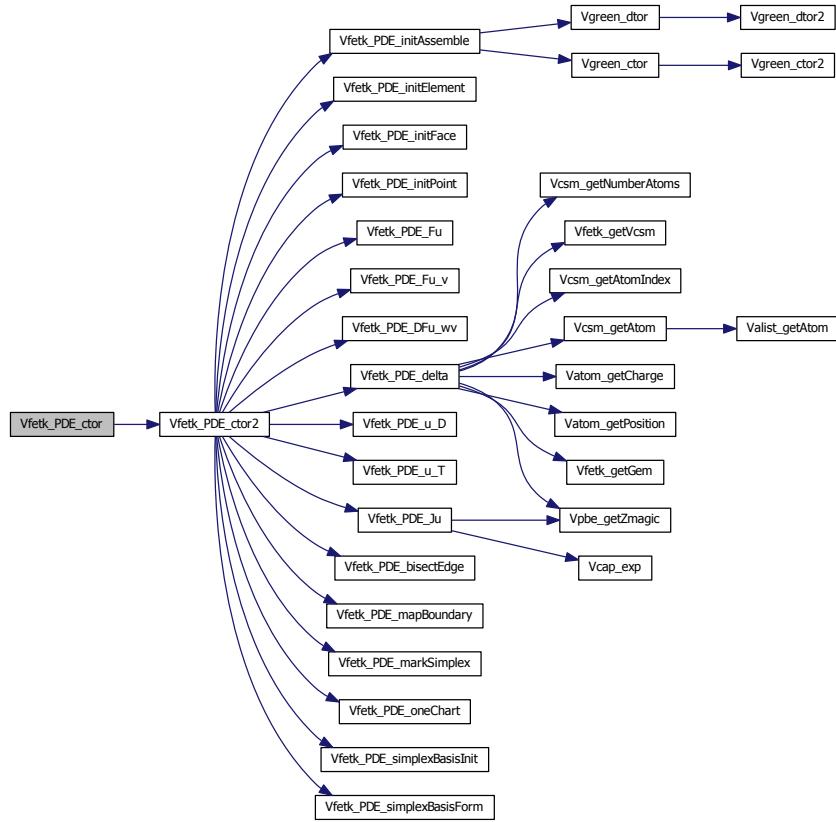
Bug Not thread-safe

Parameters

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

Definition at line 1181 of file [vfetk.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

Initializes the FEtk PDE object.

Author

Nathan Baker (with code by Mike Holst)

Returns

1 if successful, 0 otherwise

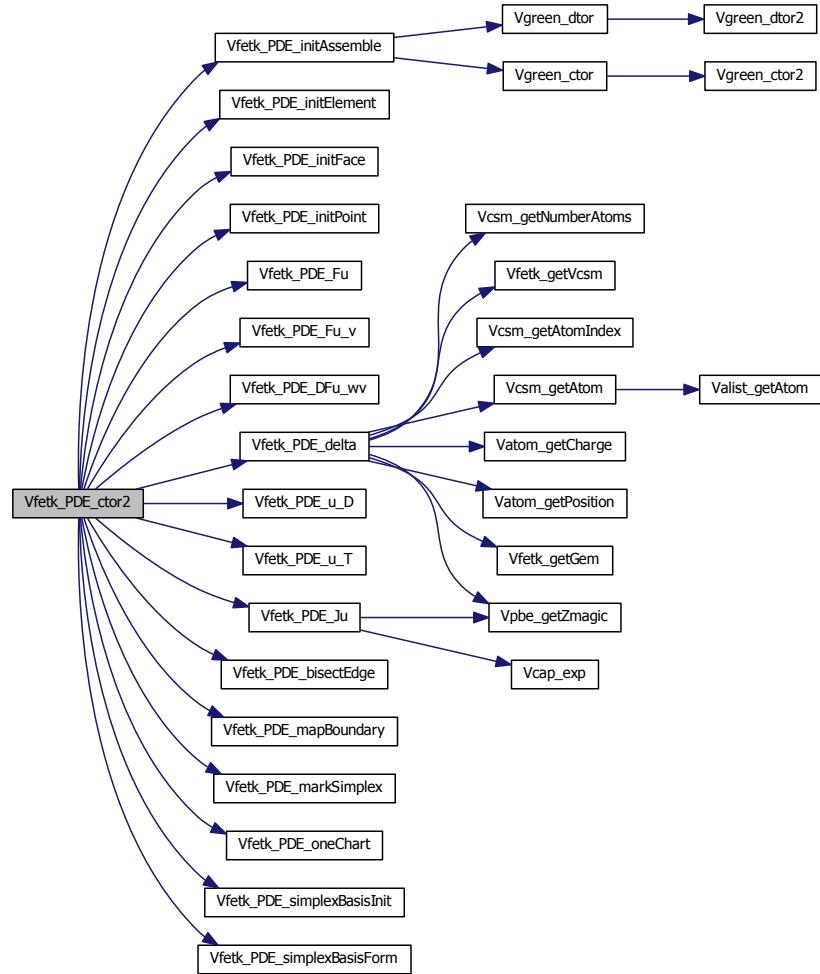
Bug Not thread-safe

Parameters

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

Definition at line 1192 of file [vfetk.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

Author

Nathan Baker

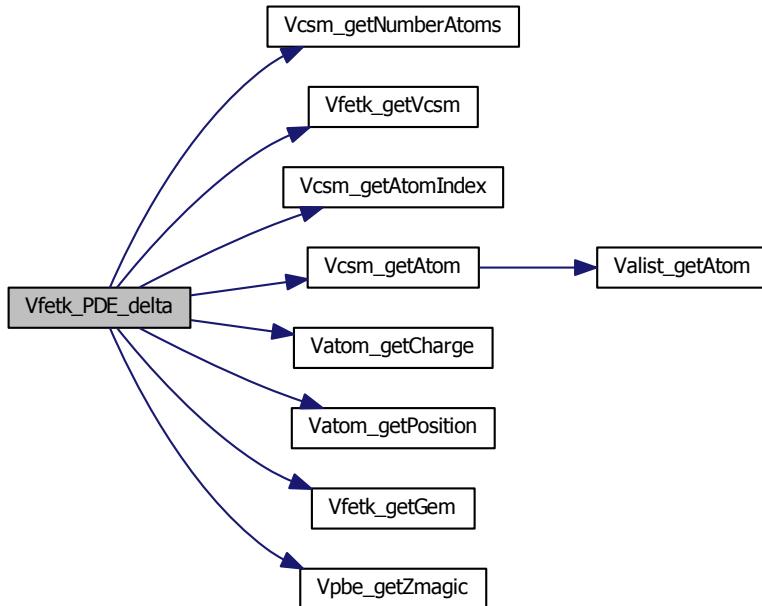
Bug This function is not thread-safe

Parameters

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

Definition at line 1785 of file [vfetk.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

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

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

Author

Nathan Baker and Mike Holst

Returns

Integrand value

Bug This function is not thread-safe

Parameters

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

Here is the caller graph for this function:



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

Destroys FEtk PDE object.

Author

Nathan Baker

Note

Thread-safe

Parameters

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

Definition at line 1236 of file [vfetk.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.2.3.27 VEXTERNC void Vfetk_PDE_dtor2(PDE * *thee*)

FORTRAN stub: destroys FEtk PDE object.

Author

Nathan Baker

Note

Thread-safe

Parameters

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

Definition at line 1251 of file [vfetk.c](#).

Here is the caller graph for this function:



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

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

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

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

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

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

Author

Nathan Baker

Bug This function is not thread-safe

This function is not implemented (sets error to zero)

Parameters

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

Definition at line 1700 of file [vfetk.c](#).

Here is the caller graph for this function:



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

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

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

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

Author

Nathan Baker and Mike Holst

Returns

Integrand value

Bug This function is not thread-safe

Parameters

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

Definition at line 1708 of file [vfetk.c](#).

Here is the caller graph for this function:



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

Do once-per-assembly initialization.

Author

Nathan Baker and Mike Holst

Note

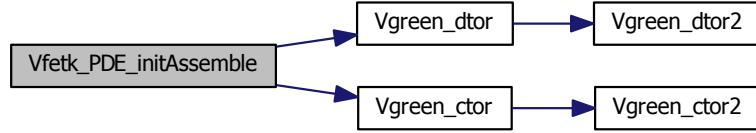
Thread-safe

Parameters

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

Do once-per-element initialization.

Author

Nathan Baker and Mike Holst

Bug This function is not thread-safe

Parameters

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

Here is the caller graph for this function:



7.2.3.32 VEXTERNC void Vfetk_PDE_initFace (PDE * *thee*, int *faceType*, int *chart*, double *tvec*[])

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>tvec</i> | Coordinates of outward normal vector for face |

Definition at line 1564 of file [vfetk.c](#).

Here is the caller graph for this function:



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

Do once-per-point initialization.

Author

Nathan Baker

Bug This function is not thread-safe

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

Parameters

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

Here is the caller graph for this function:



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

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

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

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

Author

Nathan Baker

Returns

Energy value (in kT)

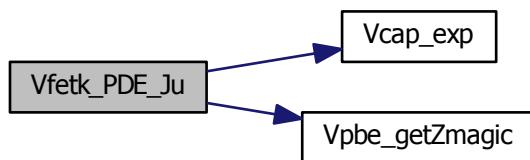
Bug This function is not thread-safe.

Parameters

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

Definition at line 2008 of file [vfetk.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

Map a boundary point to some pre-defined shape.

Author

Nathan Baker and Mike Holst

Note

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

Parameters

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

Here is the caller graph for this function:



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

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

Author

Nathan Baker

Returns

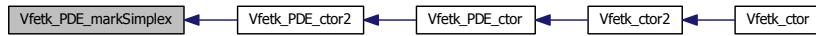
1 if mark simplex for refinement, 0 otherwise

Bug This function is not thread-safe

Parameters

| | |
|--------------------|-------------------------------|
| <i>dim</i> | Intrinsic manifold dimension |
| <i>dimll</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 |

Here is the caller graph for this function:



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

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

Author

Nathan Baker

Note

Thread-safe; a no-op

Parameters

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

Here is the caller graph for this function:



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

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

Author

Mike Holst

Parameters

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

Definition at line 2208 of file [vfetk.c](#).

Here is the caller graph for this function:



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

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

Author

Mike Holst

Note

```

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

Parameters

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

Definition at line 2145 of file [vfetk.c](#).

Here is the caller graph for this function:



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

Evaluate the Dirichlet boundary condition at the given point.

Author

Nathan Baker

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

This function is not thread-safe.

Parameters

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

Definition at line 1872 of file [vfetk.c](#).

Here is the caller graph for this function:



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

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

Author

Nathan Baker

Note

This function only returns zero.

Bug This function is not thread-safe.

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 [vfetk.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 1891 of file [vfetk.c](#).

Here is the caller graph for this function:



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

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

Using the solution at the finest mesh level, get the electrostatic energy due to the interaction of the fixed charges with the potential: \form#12 and return the result in units of \form#1. Clearly, no self-interaction terms are removed. A factor a 1/2 has to be included to convert this to a real energy.

Author

Nathan Baker

Parameters

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

Returns

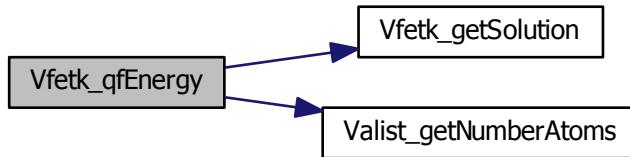
The fixed charge electrostatic energy in units of $k_B T$.

Parameters

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

Definition at line 737 of file [vfetk.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

Read in mesh and initialize associated internal structures.

Author

Nathan Baker

Note

See Also[Vfetk_genCube](#)**Parameters**

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

7.2.3.44 VEXTERNC void Vfetk_setAtomColors (Vfetk * *thee*)

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

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

Warning

This function should only be used immediately after mesh partitioning

Author

Nathan Baker

Note

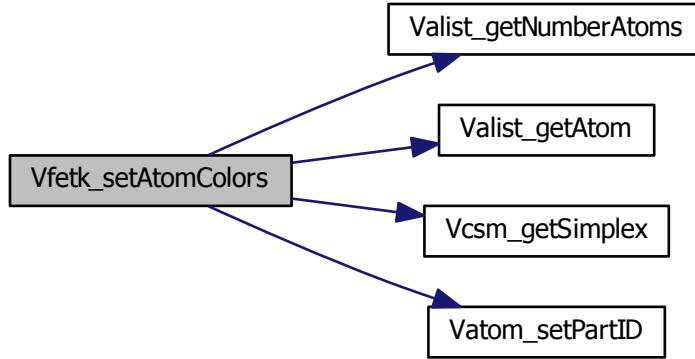
This is a friend function of Vcsm

Parameters

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

Definition at line 854 of file [vfetk.c](#).

Here is the call graph for this function:



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

Set the parameter objects.

Author

Nathan Baker

Parameters

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

Definition at line 620 of file [vfetk.c](#).

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

Write out data.

Author

Nathan Baker

Parameters

| | |
|---------------|-------------------------|
| <i>thee</i> | Vfetk object |
| <i>vec</i> | FEtk Bvec vector to use |
| <i>format</i> | Format for data |

| | |
|--------------|--|
| <i>iofmt</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>iofmt</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 2469 of file [vfetk.c](#).

7.3 Vpee class

This class provides some functionality for error estimation in parallel.

Files

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

Data Structures

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

TypeDefs

- typedef struct [sVpee](#) [Vpee](#)
Declaration of the Vpee class as the Vpee structure.

Functions

- VEXTERNC [Vpee](#) * [Vpee_ctor](#) (Gem *gm, int localPartID, int killFlag, double killParam)
Construct the Vpee object.
- VEXTERNC int [Vpee_ctor2](#) ([Vpee](#) *thee, Gem *gm, int localPartID, int killFlag, double killParam)
FORTRAN stub to construct the Vpee object.
- VEXTERNC void [Vpee_dtor](#) ([Vpee](#) **thee)
Object destructor.
- VEXTERNC void [Vpee_dtor2](#) ([Vpee](#) *thee)
FORTRAN stub object destructor.
- VEXTERNC int [Vpee_markRefine](#) ([Vpee](#) *thee, AM *am, int level, int akey, int rcol, double etol, int bkey)
Mark simplices for refinement based on attenuated error estimates.
- VEXTERNC int [Vpee_numSS](#) ([Vpee](#) *thee)
Returns the number of simplices in the local partition.

7.3.1 Detailed Description

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

7.3.2 Function Documentation

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

Construct the Vpee object.

Author

Nathan Baker

Returns

Newly constructed Vpee object

Parameters

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

See Also

[killFlag](#) for usage

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

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

FORTRAN stub to construct the Vpee object.

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Parameters

| | |
|--------------------|---|
| <i>thee</i> | The Vpee object |
| <i>gm</i> | FEtk geometry manager object |
| <i>localPartID</i> | ID of the local partition (focus of refinement) |

| | |
|------------------|---|
| <i>killFlag</i> | A flag to indicate how error estimates are to be attenuated outside the local partition: <ul style="list-style-type: none"> • 0: no attenuation • 1: all error outside the local partition set to zero • 2: all error is set to zero outside a sphere of radius (<i>killParam</i>*<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 [121](#) of file [vpee.c](#).

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

Object destructor.

Author

Nathan Baker

Parameters

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

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

7.3.2.4 VEXTERNC void Vpee_dtor2 (Vpee * *thee*)

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

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

Definition at line [247](#) of file [vpee.c](#).

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

Mark simplices for refinement based on attenuated error estimates.

A wrapper/reimplementation of AM_markRefine that allows for more flexible attenuation of error-based markings outside the local partition. The error in each simplex is modified by the method (see *killFlag*) specified in the Vpee constructor.

This allows the user to confine refinement to an arbitrary area around the local partition.

Author

Nathan Baker and Mike Holst

Note

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

Returns

The number of simplices marked for refinement.

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

Parameters

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

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

7.3.2.6 VEXTERNC int Vpee_numSS (Vpee * *thee*)

Returns the number of simplices in the local partition.

Author

Nathan Baker

Returns

Number of simplices in the local partition

Parameters

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

Definition at line [486](#) of file [vpee.c](#).

7.4 APOLparm class

Parameter structure for APOL-specific variables from input files.

Collaboration diagram for APOLparm class:



Files

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

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

7.4.1 Detailed Description

Parameter structure for APOL-specific variables from input files.

7.4.2 Enumeration Type Documentation

7.4.2.1 enum eAPOLparm_calcEnergy

Define energy calculation enumeration.

Enumerator:

- ACE_NO** Do not perform energy calculation
- ACE_TOTAL** Calculate total energy only
- ACE_COMPS** Calculate per-atom energy components

Definition at line 76 of file `apolparm.h`.

7.4.2.2 enum eAPOLparm_calcForce

Define force calculation enumeration.

Enumerator:

- ACF_NO** Do not perform force calculation
- ACF_TOTAL** Calculate total force only
- ACF_COMPS** Calculate per-atom force components

Definition at line 92 of file `apolparm.h`.

7.4.2.3 enum eAPOLparm_doCalc

Define force calculation enumeration.

Enumerator:

ACD_NO Do not perform calculation

ACD_YES Perform calculations

ACD_ERROR Error setting up calculation

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

7.4.3 Function Documentation

7.4.3.1 VEXTERNC Vrc_Codes APOLparm_check (APOLparm * *thee*)

Consistency check for parameter values stored in object.

Author

David Gohara, Yong Huang

Parameters

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

Returns

Success enumeration

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

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

Copy target object into thee.

Author

Nathan Baker

Parameters

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

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

Here is the caller graph for this function:



7.4.3.3 VEXTERNC APOLparm* APOLparm_ctor()

Construct APOLparm.

Author

David Gohara

Returns

Newly allocated and initialized Vpmgp object

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.3.4 VEXTERNC Vrc_Codes APOLparm_ctor2 (APOLparm * *thee*)

FORTRAN stub to construct APOLparm.

Author

David Gohara, Yong Huang

Parameters

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

Returns

Success enumeration

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

Here is the caller graph for this function:

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

Object destructor.

Author

David Gohara

Parameters

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.3.6 VEXTERNC void APOLparm_dtor2 (APOLparm * *thee*)

FORTRAN stub for object destructor.

Author

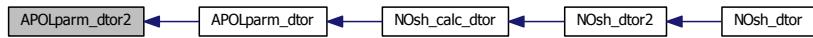
David Gohara

Parameters

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

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

Here is the caller graph for this function:



7.5 FEMparm class

Parameter structure for FEM-specific variables from input files.

Collaboration diagram for FEMparm class:



Files

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

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

7.5.1 Detailed Description

Parameter structure for FEM-specific variables from input files.

7.5.2 Typedef Documentation

7.5.2.1 `typedef enum eFEMparm_EtolType FEMparm_EtolType`

Declare FEparm_EtolType type.

Author

Nathan Baker

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

7.5.3 Enumeration Type Documentation

7.5.3.1 `enum eFEMparm_CalcType`

Calculation type.

Enumerator:

`FCT_MANUAL` fe-manual

`FCT_NONE` unspecified

Definition at line 114 of file `femparm.h`.

7.5.3.2 `enum eFEMparm_EstType`

Adaptive refinement error estimator method.

Note

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

Author

Nathan Baker

Enumerator:

FRT_UNIF Uniform refinement

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

FRT_RESI Nonlinear residual estimate-based refinement

FRT_DUAL Dual-solution weight nonlinear residual estimate-based refinement

FRT_LOCA Local problem error estimate-based refinement

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

7.5.3.3 enum eFEMparm_EtolType

Adaptive refinement error estimate tolerance key.

Author

Nathan Baker

Enumerator:

FET_SIMP per-simplex error tolerance

FET_GLOB global error tolerance

FET_FRAC fraction of simplices we want to have refined

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

7.5.4 Function Documentation

7.5.4.1 VEXTERNC int FEMparm_check (**FEMparm** * *thee*)

Consistency check for parameter values stored in object.

Author

Nathan Baker

Parameters

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

Returns

1 if OK, 0 otherwise

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

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

Copy target object into thee.

Author

Nathan Baker

Parameters

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

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

Here is the caller graph for this function:

**7.5.4.3 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 67 of file [femparm.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

FORTRAN stub to construct FEMparm.

Author

Nathan Baker

Parameters

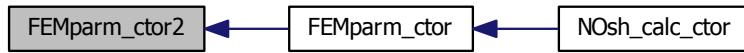
| | |
|-------------|-------------------------------------|
| <i>thee</i> | Pointer to allocated FEMparm object |
| <i>type</i> | FEM calculation type |

Returns

1 if successful, 0 otherwise

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

Here is the caller graph for this function:



7.5.4.5 VEXTERNC void FEMPARM_DTOR (FEMPARM ** *thee*)

Object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 135 of file [femparm.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.5.4.6 VEXTERNC void FEMparm_dtor2 (FEMparm * *thee*)

FORTRAN stub for object destructor.

Author

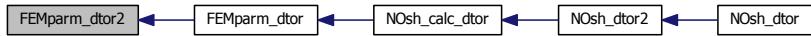
Nathan Baker

Parameters

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

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

Here is the caller graph for this function:



7.6 MGparm class

Parameter which holds useful parameters for generic multigrid calculations.

Files

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

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

7.6.1 Detailed Description

Parameter which holds useful parameters for generic multigrid calculations.

7.6.2 Enumeration Type Documentation

7.6.2.1 enum eMGparm_CalcType

Calculation type.

Enumerator:

MCT_MANUAL mg-manual
MCT_AUTO mg-auto

MCT_PARALLEL mg-para
MCT_DUMMY mg-dummy
MCT_NONE unspecified

Definition at line 75 of file [mgparm.h](#).

7.6.2.2 enum eMGparm_CentMeth

Centering method.

Enumerator:

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

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

7.6.3 Function Documentation

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

Parse an MG keyword from an input file.

Author

David Gohara

Parameters

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

Returns

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

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

Here is the call graph for this function:



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

Parse an MG keyword from an input file.

Author

Nathan Baker

Parameters

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

Returns

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

Definition at line 433 of file [femparm.c](#).

Here is the call graph for this function:



7.6.3.3 VEXTERNC Vrc_Codes MGparm_check (**MGparm * thee**)

Consistency check for parameter values stored in object.

Author

Nathan Baker

Parameters

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

Returns

Success enumeration

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

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

Copy MGparm object into thee.

Author

Nathan Baker and Todd Dolinsky

Parameters

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

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

Here is the caller graph for this function:



7.6.3.5 VEXTERNC MGparm* MGparm_ctor (MGparm_CalcType *type*)

Construct MGparm object.

Author

Nathan Baker

Parameters

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

Returns

Newly allocated and initialized MGparm object

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

FORTRAN stub to construct MGparm object.

Author

Nathan Baker and Todd Dolinsky

Parameters

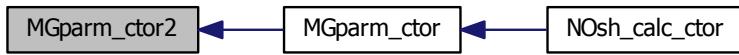
| | |
|-------------|-------------------------|
| <i>thee</i> | Space for MGparm object |
| <i>type</i> | Type of MG calculation |

Returns

Success enumeration

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

Here is the caller graph for this function:



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

Object destructor.

Author

Nathan Baker

Parameters

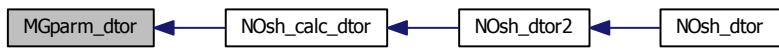
| | |
|-------------|---|
| <i>thee</i> | Pointer to memory location of MGparm object |
|-------------|---|

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.6.3.8 VEXTERNC void MGparm_dtor2(MGparm * *thee*)

FORTRAN stub for object destructor.

Author

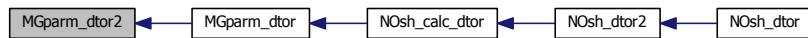
Nathan Baker

Parameters

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

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

Here is the caller graph for this function:

**7.6.3.9 VEXTERNC double MGparm_getCenterX(MGparm * *thee*)**

Get center x-coordinate.

Author

Nathan Baker

Parameters

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

Returns

x-coordinate

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

7.6.3.10 VEXTERNC double MGparm_getCenterY(MGparm * *thee*)

Get center y-coordinate.

Author

Nathan Baker

Parameters

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

Returns

y-coordinate

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

7.6.3.11 VEXTERNC double MGparm_getCenterZ (MGparm * *thee*)

Get center z-coordinate.

Author

Nathan Baker

Parameters

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

Returns

z-coordinate

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

7.6.3.12 VEXTERNC double MGparm_getHx (MGparm * *thee*)

Get grid spacing in x direction (Å)

Author

Nathan Baker

Parameters

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

Returns

Grid spacing in the x direction

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

7.6.3.13 VEXTERNC double MGparm_getHy (MGparm * *thee*)

Get grid spacing in y direction (Å)

Author

Nathan Baker

Parameters

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

Returns

Grid spacing in the y direction

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

7.6.3.14 VEXTERNC double MGparm_getHz (MGparm * *thee*)

Get grid spacing in z direction (Å)

Author

Nathan Baker

Parameters

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

Returns

Grid spacing in the z direction

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

7.6.3.15 VEXTERNC int MGparm_getNx (MGparm * *thee*)

Get number of grid points in x direction.

Author

Nathan Baker

Parameters

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

Returns

Number of grid points in the x direction

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

7.6.3.16 VEXTERNC int MGparm_getNy (MGparm * *thee*)

Get number of grid points in y direction.

Author

Nathan Baker

Parameters

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

Returns

Number of grid points in the y direction

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

7.6.3.17 VEXTERNC int MGparm_getNz (MGparm * *thee*)

Get number of grid points in z direction.

Author

Nathan Baker

Parameters

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

Returns

Number of grid points in the z direction

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

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

Parse an MG keyword from an input file.

Author

Nathan Baker and Todd Dolinsky

Parameters

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

Returns

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

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

Here is the call graph for this function:



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

Set center x-coordinate.

Author

Nathan Baker

Parameters

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

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

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

Set center y-coordinate.

Author

Nathan Baker

Parameters

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

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

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

Set center z-coordinate.

Author

Nathan Baker

Parameters

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

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

7.7 NOsh class

Class for parsing for fixed format input files.

Files

- file `nosh.h`
Contains declarations for class NOsh.
- file `nosh.c`
Class NOsh methods.

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 }

NOsh calculation types.

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

Parameter file format types.

- enum `eNOsh_PrintType` {
`NPT_ENERGY` = 0, `NPT_FORCE` = 1, `NPT_ELECENERGY`, `NPT_ELECFORCE`,
`NPT_APOLENERGY`, `NPT_APOLFORCE` }

NOsh print types.

Functions

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

- VEXTERNC int [NOsh_printNarg](#) (NOsh *thee, int iprint)

Return number of arguments to PRINT statement .
- VEXTERNC int [NOsh_printOp](#) (NOsh *thee, int iprint, int iarg)

Return integer ID for specified operation .
- VEXTERNC int [NOsh_printCalc](#) (NOsh *thee, int iprint, int iarg)

Return calculation ID for specified PRINT statement .
- VEXTERNC NOsh * [NOsh_ctor](#) (int rank, int size)

Construct NOsh.
- VEXTERNC NOsh_calc * [NOsh_calc_ctor](#) (NOsh_CalcType calcType)

Construct NOsh_calc.
- VEXTERNC int [NOsh_calc_copy](#) (NOsh_calc *thee, NOsh_calc *source)

Copy NOsh_calc object into thee.
- VEXTERNC void [NOsh_calc_dtor](#) (NOsh_calc **thee)

Object destructor.
- VEXTERNC int [NOsh_ctor2](#) (NOsh *thee, int rank, int size)

FORTRAN stub to construct NOsh.
- VEXTERNC void [NOsh_dtor](#) (NOsh **thee)

Object destructor.
- VEXTERNC void [NOsh_dtor2](#) (NOsh *thee)

FORTRAN stub for object destructor.
- VEXTERNC int [NOsh_parseInput](#) (NOsh *thee, Vio *sock)

Parse an input file from a socket.
- VEXTERNC int [NOsh_parseInputFile](#) (NOsh *thee, char *filename)

Parse an input file only from a file.
- VEXTERNC int [NOsh_setupElecCalc](#) (NOsh *thee, Valist *alist[NOSH_MAXMOL])

Setup the series of electrostatics calculations.
- VEXTERNC int [NOsh_setupApolCalc](#) (NOsh *thee, Valist *alist[NOSH_MAXMOL])

Setup the series of non-polar calculations.

7.7.1 Detailed Description

Class for parsing for fixed format input files.

7.7.2 Enumeration Type Documentation

7.7.2.1 enum eNOsh_CalcType

NOsh calculation types.

Enumerator:

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

Definition at line 112 of file [nosh.h](#).

7.7.2.2 enum eNOsh_MolFormat

Molecule file format types.

Enumerator:

NMF_PQR PQR format

NMF_PDB PDB format

NMF_XML XML format

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

7.7.2.3 enum eNOsh_ParmFormat

Parameter file format types.

Enumerator:

NPF_FLAT Flat-file format

NPF_XML XML format

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

7.7.2.4 enum eNOsh_PrintType

NOsh print types.

Enumerator:

NPT_ENERGY Energy (deprecated)

NPT_FORCE Force (deprecated)

NPT_ELECENERGY Elec Energy

NPT_ELECFORCE Elec Force

NPT_APOLENERGY Apol Energy

NPT_APOLFORCE Apol Force

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

7.7.3 Function Documentation

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

Return the name of an apol statement.

Author

David Gohara

Parameters

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

Returns

The name (if present) of an APOL statement

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

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

Copy NOsh_calc object into thee.

Author

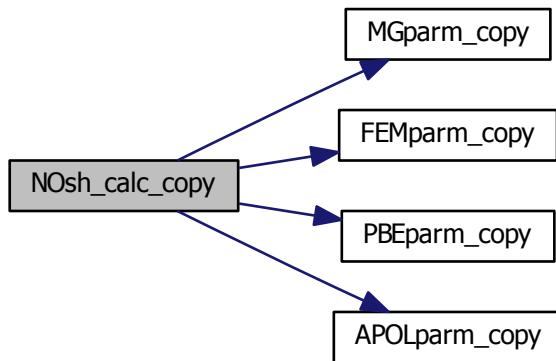
Nathan Baker

Parameters

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

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

Here is the call graph for this function:

**7.7.3.3 VEXTERNC NOsh_calc* NOsh_calc_ctor (NOsh_CalcType calcType)**

Construct NOsh_calc.

Author

Nathan Baker

Parameters

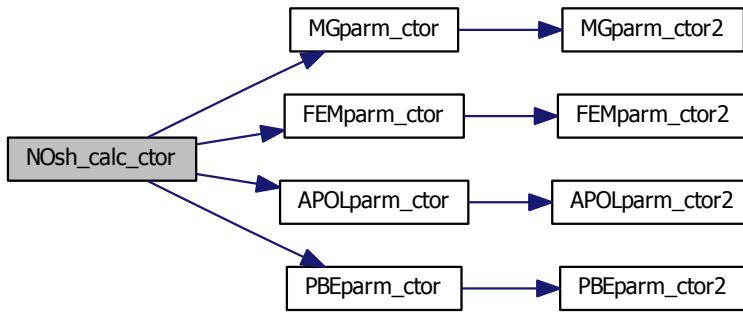
| | |
|-----------------|------------------|
| <i>calcType</i> | Calculation type |
|-----------------|------------------|

Returns

Newly allocated and initialized NOsh object

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

Here is the call graph for this function:



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

Object destructor.

Author

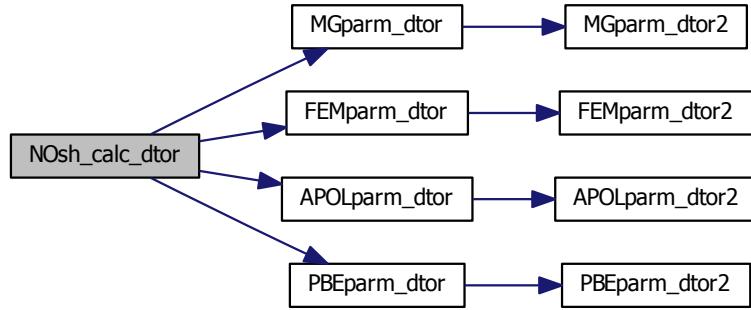
Nathan Baker

Parameters

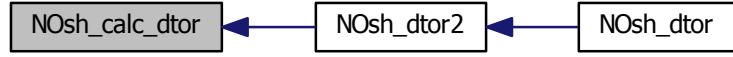
| | |
|-------------|--|
| <i>thee</i> | Pointer to memory location of NOsh_calc object |
|-------------|--|

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

Construct NOsh.

Author

Nathan Baker

Parameters

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

Returns

Newly allocated and initialized NOsh object

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

Here is the call graph for this function:



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

FORTRAN stub to construct NOsh.

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

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

Here is the caller graph for this function:



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

Object destructor.

Author

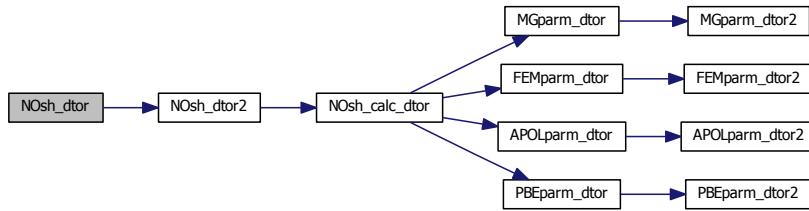
Nathan Baker

Parameters

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

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

Here is the call graph for this function:



7.7.3.8 VEXTERNC void NOsh_dtor2 (NOsh * *thee*)

FORTRAN stub for object destructor.

Author

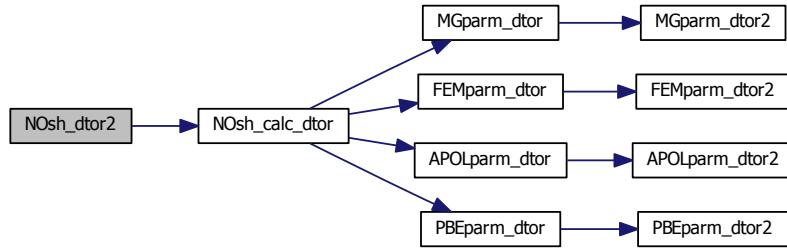
Nathan Baker

Parameters

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

Return the name of an elec statement.

Author

Todd Dolinsky

Parameters

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

Returns

The name (if present) of an ELEC statement

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

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

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

See Also

[elec2calc\(\)](#)

Author

Nathan Baker

Parameters

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

Returns

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

See Also

[elec2calc\(\)](#)

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

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

Returns specified calculation object.

Author

Nathan Baker

Parameters

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

Returns

Pointer to specified calculation object

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

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

Returns format of specified charge map.

Author

Nathan Baker

Parameters

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

Returns

Format of charge map

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

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

Returns path to specified charge distribution map.

Author

Nathan Baker

Parameters

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

Returns

Path string

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

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

Returns format of specified dielectric map.

Author

Nathan Baker

Parameters

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

Returns

Format of dielectric map

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

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

Returns path to specified x-shifted dielectric map.

Author

Nathan Baker

Parameters

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

Returns

Path string

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

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

Returns path to specified y-shifted dielectric map.

Author

Nathan Baker

Parameters

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

Returns

Path string

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

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

Returns path to specified z-shifted dielectric map.

Author

Nathan Baker

Parameters

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

Returns

Path string

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

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

Returns format of specified kappa map.

Author

Nathan Baker

Parameters

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

Returns

Format of kappa map

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

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

Returns path to specified kappa map.

Author

Nathan Baker

Parameters

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

Returns

Path string

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

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

Returns path to specified molecule.

Author

Nathan Baker

Parameters

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

Returns

Path string

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

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

Returns format of specified potential map.

Author

Nathan Baker

Parameters

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

Returns

Format of potential map

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

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

Returns path to specified potential map.

Author

David Gohara

Parameters

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

Returns

Path string

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

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

Parse an input file from a socket.

Note

Should be called before NOsh_setupCalc

Author

Nathan Baker and Todd Dolinsky

Parameters

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

Returns

1 if successful, 0 otherwise

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

Parse an input file only from a file.

Note

Included for SWIG wrapper compatibility
Should be called before NOsh_setupCalc

Author

Nathan Baker and Todd Dolinsky

Parameters

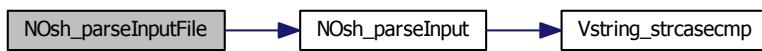
| | |
|-----------------|----------------------------|
| <i>thee</i> | Pointer to NOsh object |
| <i>filename</i> | Name/path of readable file |

Returns

1 if successful, 0 otherwise

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

Here is the call graph for this function:

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

Return calculation ID for specified PRINT statement (.

See Also

[printcalc\(\)](#)

Author

Nathan Baker

Parameters

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

Returns

Calculation ID for specified PRINT statement (

See Also

[printcalc\(\)](#)

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

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

Return number of arguments to PRINT statement (.

See Also

`printnarg()`

Author

Nathan Baker

Parameters

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

Returns

Number of arguments to PRINT statement (

See Also

`printnarg()`

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

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

Return integer ID for specified operation (.

See Also

`printop()`

Author

Nathan Baker

Parameters

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

Returns

Integer ID for specified operation (

See Also

`printop()`

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

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

Return an integer ID of the observable to print (.

See Also

[printwhat\(\)](#)

Author

Nathan Baker

Parameters

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

Returns

An integer ID of the observable to print (

See Also

[printwhat\(\)](#)

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

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

Setup the series of non-polar calculations.

Note

Should be called after NOsh_parseInput*

Author

Nathan Baker and Todd Dolinsky

Parameters

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

Returns

1 if successful, 0 otherwise

Parameters

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

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

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

Setup the series of electrostatics calculations.

Note

Should be called after NOsh_parseInput*

Author

Nathan Baker and Todd Dolinsky

Parameters

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

Returns

1 if successful, 0 otherwise

Parameters

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

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

7.8 PBparm class

Parameter structure for PBE variables independent of solver.

Files

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

Data Structures

- struct [sPBparm](#)
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 ePBparm_calcEnergy PBparm_calcEnergy`
Define ePBparm_calcEnergy enumeration as PBparm_calcEnergy.
- `typedef enum ePBparm_calcForce PBparm_calcForce`
Define ePBparm_calcForce enumeration as PBparm_calcForce.
- `typedef struct sPBparm PBparm`
Declaration of the PBparm class as the PBparm structure.

Enumerations

- `enum ePBparm_calcEnergy { PCE_NO = 0, PCE_TOTAL = 1, PCE_COMPS = 2 }`
Define energy calculation enumeration.
- `enum ePBparm_calcForce { PCF_NO = 0, PCF_TOTAL = 1, PCF_COMPS = 2 }`
Define force calculation enumeration.

Functions

- VEXTERNC double [PBparm_getIonCharge](#) ([PBparm](#) *thee, int iion)
Get charge (e) of specified ion species.
- VEXTERNC double [PBparm_getIonConc](#) ([PBparm](#) *thee, int iion)
Get concentration (M) of specified ion species.
- VEXTERNC double [PBparm_getIonRadius](#) ([PBparm](#) *thee, int iion)
Get radius (A) of specified ion species.
- VEXTERNC [PBparm](#) * [PBparm_ctor](#) ()

- VEXTERNC int **PBEparm_ctor2** (PBEparm *thee)

FORTRAN stub to construct PBEparm object.
- VEXTERNC void **PBEparm_dtor** (PBEparm **thee)

Object destructor.
- VEXTERNC void **PBEparm_dtor2** (PBEparm *thee)

FORTRAN stub for object destructor.
- VEXTERNC int **PBEparm_check** (PBEparm *thee)

Consistency check for parameter values stored in object.
- VEXTERNC void **PBEparm_copy** (PBEparm *thee, PBEparm *parm)

Copy PBEparm object into thee.
- VEXTERNC int **PBEparm_parseToken** (PBEparm *thee, char tok[VMAX_BUFSIZE], Vio *sock)

Parse a keyword from an input file.

7.8.1 Detailed Description

Parameter structure for PBE variables independent of solver.

7.8.2 Enumeration Type Documentation

7.8.2.1 enum ePBEparm_calcEnergy

Define energy calculation enumeration.

Enumerator:

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

Definition at line 80 of file [pbeparm.h](#).

7.8.2.2 enum ePBEparm_calcForce

Define force calculation enumeration.

Enumerator:

- PCF_NO** Do not perform force calculation
- PCF_TOTAL** Calculate total force only
- PCF_COMPS** Calculate per-atom force components

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

7.8.3 Function Documentation

7.8.3.1 VEXTERNC int PBparm_check (PBparm * *thee*)

Consistency check for parameter values stored in object.

Author

Nathan Baker

Returns

1 if OK, 0 otherwise

Parameters

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

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

7.8.3.2 VEXTERNC void PBparm_copy (PBparm * *thee*, PBparm * *parm*)

Copy PBparm object into thee.

Author

Nathan Baker

Parameters

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

Definition at line 285 of file [pbparm.c](#).

Here is the caller graph for this function:



7.8.3.3 VEXTERNC PBparm* PBparm_ctor()

Construct PBparm object.

Author

Nathan Baker

Returns

Newly allocated and initialized PBEparm object

Definition at line 106 of file [pbeparm.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.8.3.4 VEXTERNC int PBEparm_ctor2(PBEparm * *thee*)

FORTRAN stub to construct PBEparm object.

Author

Nathan Baker

Returns

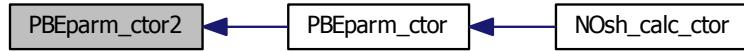
1 if successful, 0 otherwise

Parameters

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

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

Here is the caller graph for this function:



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

Object destructor.

Author

Nathan Baker

Parameters

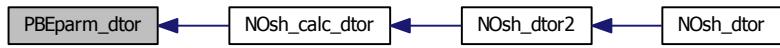
| | |
|-------------|--------------------------------------|
| <i>thee</i> | Pointer to memory location of object |
|-------------|--------------------------------------|

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.8.3.6 VEXTERNC void PBparm_dtor2 (PBparm * *thee*)

FORTRAN stub for object destructor.

Author

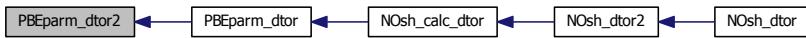
Nathan Baker

Parameters

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

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

Here is the caller graph for this function:



7.8.3.7 VEXTERNC double PBEparm_getIonCharge (PBEparm * *thee*, int *iion*)

Get charge (e) of specified ion species.

Author

Nathan Baker

Returns

Charge of ion species (e)

Parameters

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

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

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

Get concentration (M) of specified ion species.

Author

Nathan Baker

Returns

Concentration of ion species (M)

Parameters

| | |
|-------------|----------------------|
| <i>thee</i> | PBEmprm object |
| <i>iion</i> | Ion species ID/index |

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

7.8.3.9 VEXTERNC double PBEmprm_getlonRadius (PBEmprm * *thee*, int *iion*)

Get radius (A) of specified ion species.

Author

Nathan Baker

Returns

Radius of ion species (A)

Parameters

| | |
|-------------|----------------------|
| <i>thee</i> | PBEmprm object |
| <i>iion</i> | Ion species ID/index |

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

7.8.3.10 VEXTERNC int PBEmprm_parseToken (PBEmprm * *thee*, char *tok*[VMAX_BUFSIZE], Vio * *sock*)

Parse a keyword from an input file.

Author

Nathan Baker

Returns

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

Parameters

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

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

Here is the call graph for this function:



7.9 Vacc class

Solvent- and ion-accessibility oracle.

Files

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

Data Structures

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

Typedefs

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

Functions

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

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

- VEXTERNC void `Vacc_totalAtomdSASA` (`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.

7.9.1 Detailed Description

Solvent- and ion-accessibility oracle.

7.9.2 Function Documentation

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

Get the derivative of solvent accessible area.

Author

Jason Wagoner, David Gohara, Nathan Baker

Parameters

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

Definition at line 1338 of file `vacc.c`.

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

Get the derivative of solvent accessible volume.

Author

Jason Wagoner, Nathan Baker

Parameters

| | |
|---------------------|---------------------|
| <code>thee</code> | Acessibility object |
| <code>radius</code> | Probe radius (Å) |

| | |
|-------------|-------------------------------|
| <i>atom</i> | Atom of interest |
| <i>dSA</i> | Array holding answers of calc |

Definition at line 1218 of file [vacc.c](#).

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

Return the atomic solvent accessible surface area (SASA)

Note

Alias for Vacc_SASA

Author

Nathan Baker

Returns

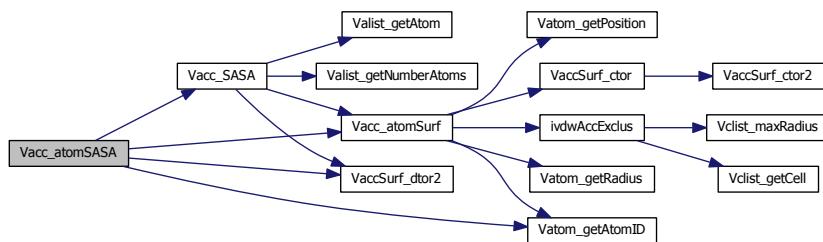
Atomic solvent accessible area (A^2)

Parameters

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

Definition at line 786 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

Author

Nathan Baker

Returns

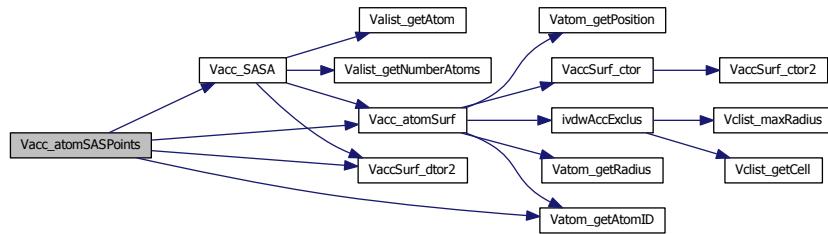
Pointer to VaccSurf object for this atom

Parameters

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

Definition at line 1000 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

Author

Nathan Baker

Returns

Atom sphere surface object

Parameters

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

See Also

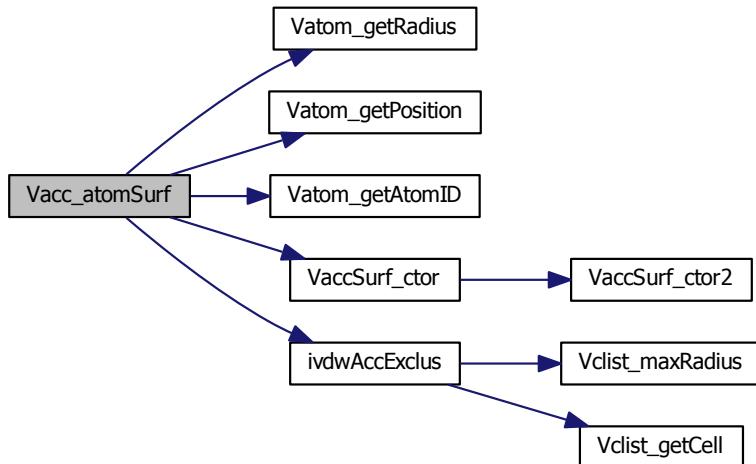
[VaccSurf_refSphere](#)

Parameters

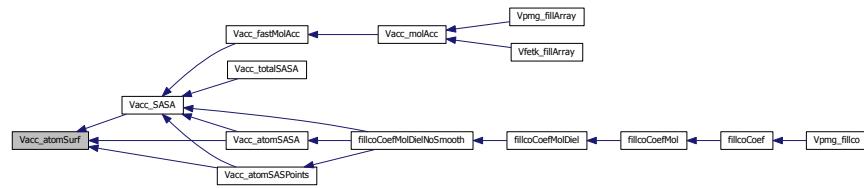
| | |
|---------------------|---------------------|
| <i>probe_radius</i> | Probe radius (in Å) |
|---------------------|---------------------|

Definition at line 879 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

Construct the accessibility object.

Author

Nathan Baker

Returns

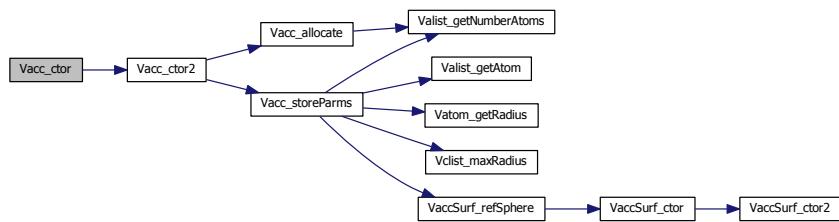
Newly allocated Vacc object

Parameters

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

FORTRAN stub to construct the accessibility object.

Author

Nathan Baker

Returns

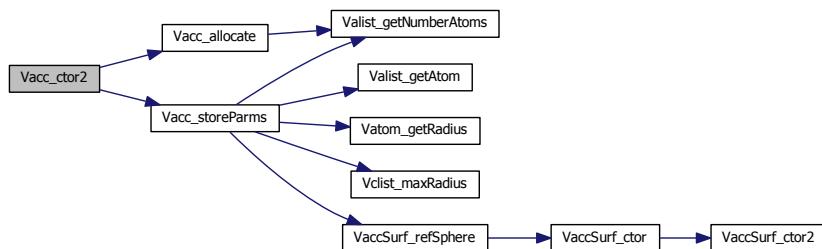
1 if successful, 0 otherwise

Parameters

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

Definition at line 219 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

Destroy object.

Author

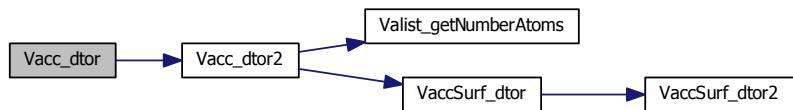
Nathan Baker

Parameters

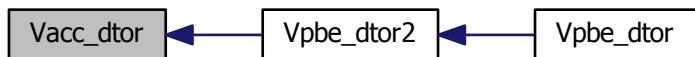
| | |
|-------------|--------------------------------------|
| <i>thee</i> | Pointer to memory location of object |
|-------------|--------------------------------------|

Definition at line 251 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.9.2.9 VEXTERNC void Vacc_dtor2 (Vacc * *thee*)

FORTRAN stub to destroy object.

Author

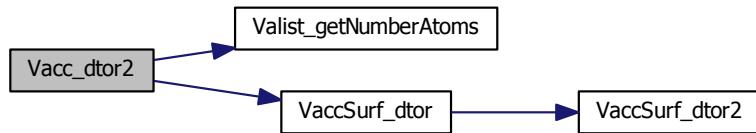
Nathan Baker

Parameters

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

Definition at line 261 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

Report molecular accessibility quickly.

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

Note

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

Author

Nathan Baker

Returns

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

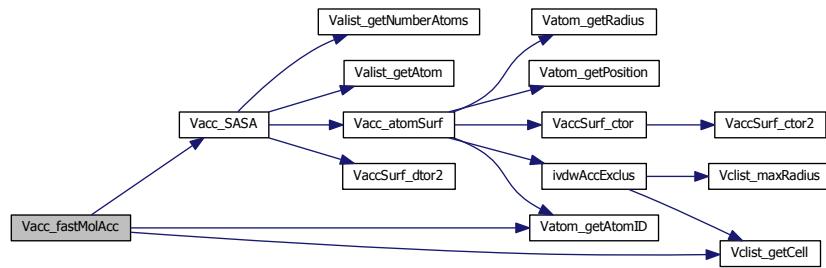
Bug This routine has a slight bug which can generate very small internal regions of high dielectric (thanks to John Mongan and Jess Swanson for finding this)

Parameters

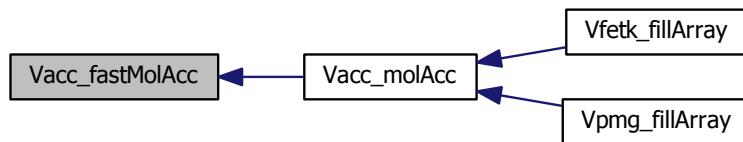
| | |
|---------------|--------------------------|
| <i>thee</i> | Accessibility object |
| <i>center</i> | Probe center coordinates |
| <i>radius</i> | Probe radius (in Å) |

Definition at line 643 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

Report inflated van der Waals accessibility.

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

Author

Nathan Baker

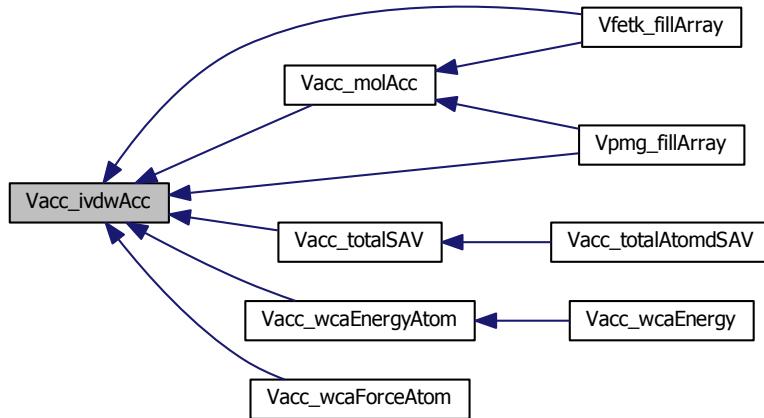
Returns

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

Parameters

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

Here is the caller graph for this function:



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

Get number of bytes in this object and its members.

Author

Nathan Baker

Returns

Number of bytes allocated for object

Parameters

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

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

Here is the caller graph for this function:



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

Report molecular accessibility.

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

Author

Nathan Baker

Returns

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

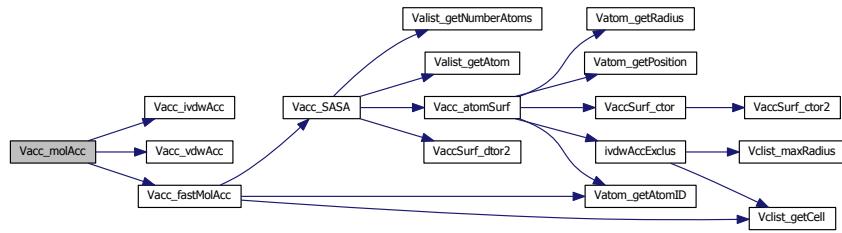
Bug This routine has a slight bug which can generate very small internal regions of high dielectric (thanks to John Mongan and Jess Swanson for finding this)

Parameters

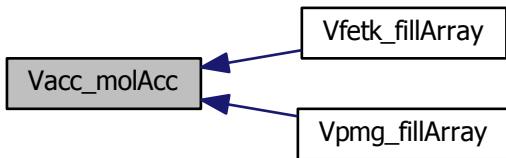
| | |
|---------------|--------------------------|
| <i>thee</i> | Accessibility object |
| <i>center</i> | Probe center coordinates |
| <i>radius</i> | Probe radius (in Å) |

Definition at line 614 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

Note

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

Author

Nathan Baker (original FORTRAN routine by Brock Luty)

Returns

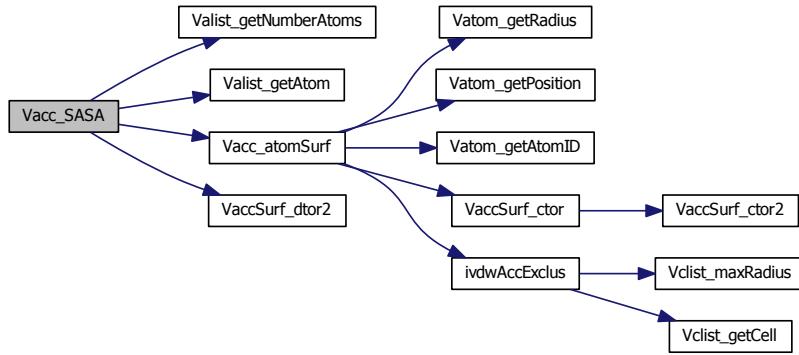
Total solvent accessible area (A^2)

Parameters

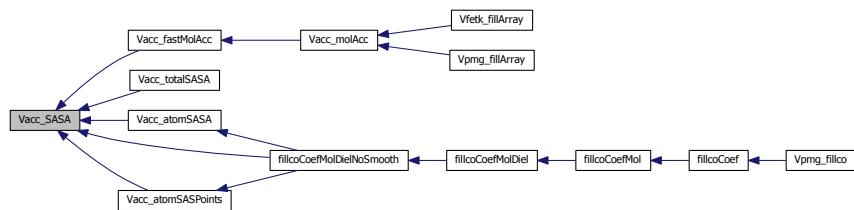
| | |
|---------------|--|
| <i>thee</i> | Accessibility object |
| <i>radius</i> | Probe molecule radius (\AA) |

Definition at line 719 of file `vacc.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



7.9.2.15 VEXTERNC double Vacc_splineAcc (`Vacc * thee, double center[VAPBS_DIM], double win, double inftrad`)

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 evaluation; basically a cubic spline.

Author

Nathan Baker

Returns

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

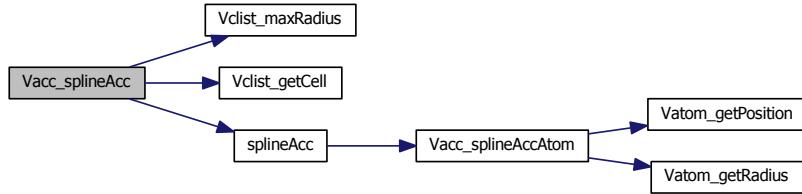
Parameters

| | |
|----------------------|--------------------------------------|
| <code>thee</code> | Accessibility object |
| <code>center</code> | Probe center coordinates |
| <code>win</code> | Spline window (Å) |
| <code>inftrad</code> | Inflation radius (Å) for ion access. |

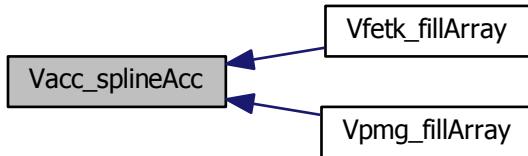
Generated on Thu Jul 19 2012 11:37:32 for APBS by doxygen

Definition at line 534 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

Report spline-based accessibility for a given atom.

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

Author

Nathan Baker

Returns

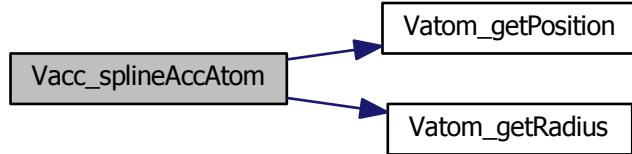
Characteristic function value between 1.0 (accessible) and 0.0 (inaccessible)

Parameters

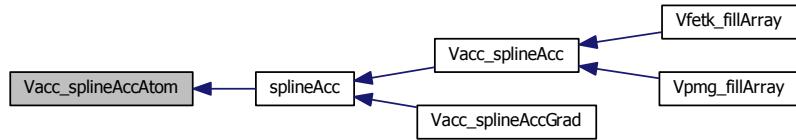
| | |
|---------------|--------------------------------------|
| <i>thee</i> | Accessibility object |
| <i>center</i> | Probe center coordinates |
| <i>win</i> | Spline window (Å) |
| <i>infrad</i> | Inflation radius (Å) for ion access. |
| <i>atom</i> | Atom |

Definition at line 444 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.9.2.17 VEXTERNC void Vacc_splineAccGrad (Vacc * *thee*, double *center*[VAPBS_DIM], double *win*, double *infrad*, double * *grad*)

Report gradient of spline-based accessibility.

Author

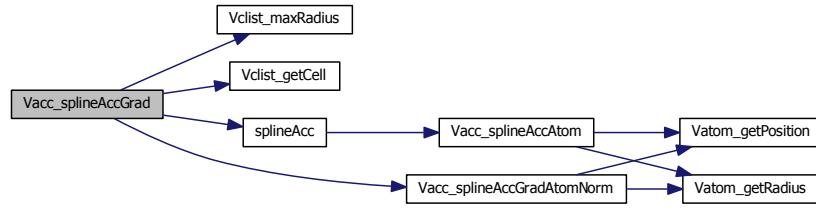
Nathan Baker

Parameters

| | |
|---------------|---|
| <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 567 of file [vacc.c](#).

Here is the call graph for this function:



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

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

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

Author

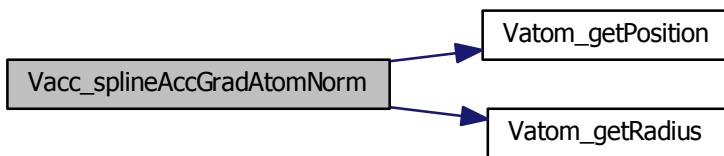
Nathan Baker

Parameters

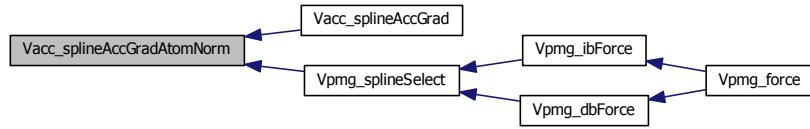
| | |
|---------------|---|
| <i>thee</i> | Accessibility object |
| <i>center</i> | Probe center coordinates |
| <i>win</i> | Spline window (Å) |
| <i>infrad</i> | Inflation radius (Å) for ion access. |
| <i>atom</i> | Atom |
| <i>force</i> | VAPBS_DIM-vector set to gradient of accessibility |

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

Author

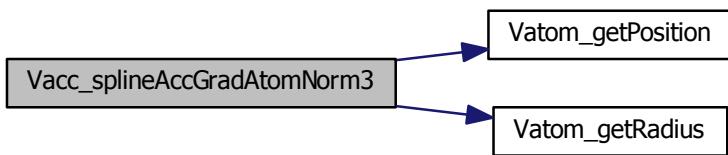
Michael Schnieders

Parameters

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

Definition at line 1117 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

Author

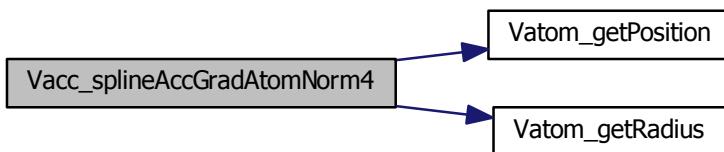
Michael Schnieders

Parameters

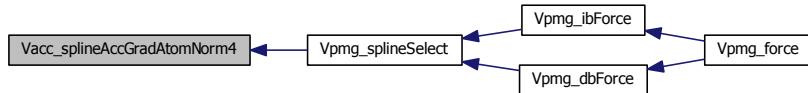
| | |
|---------------|---|
| <i>thee</i> | Accessibility object |
| <i>center</i> | Probe center coordinates |
| <i>win</i> | Spline window (Å) |
| <i>infrad</i> | Inflation radius (Å) for ion access. |
| <i>atom</i> | Atom |
| <i>force</i> | VAPBS_DIM-vector set to gradient of accessibility |

Definition at line 1024 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

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

Author

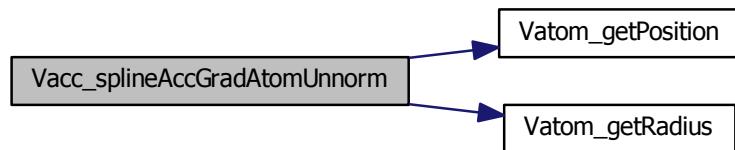
Nathan Baker

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 383 of file [vacc.c](#).

Here is the call graph for this function:



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

Testing purposes only.

Author

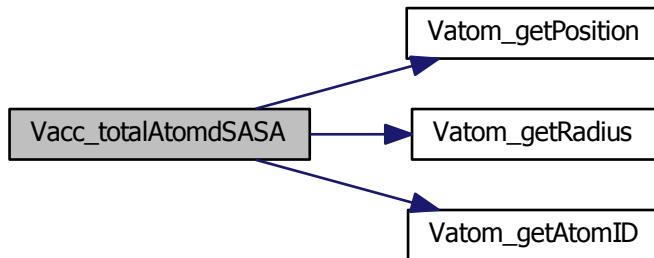
David Gohara, Nathan Baker

Parameters

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

Definition at line 1407 of file [vacc.c](#).

Here is the call graph for this function:



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

Total solvent accessible volume.

Author

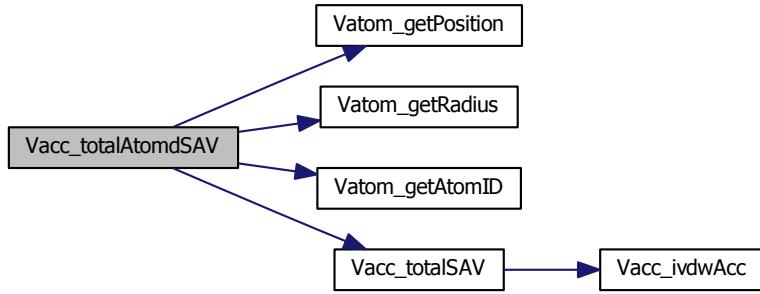
David Gohara, Nathan Baker

Parameters

| | |
|---------------|-------------------------------|
| <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 1466 of file [vacc.c](#).

Here is the call graph for this function:



7.9.2.24 VEXTERNC double Vacc_totalSASA (`Vacc * thee, double radius`)

Return the total solvent accessible surface area (SASA)

Note

Alias for `Vacc_SASA`

Author

Nathan Baker

Returns

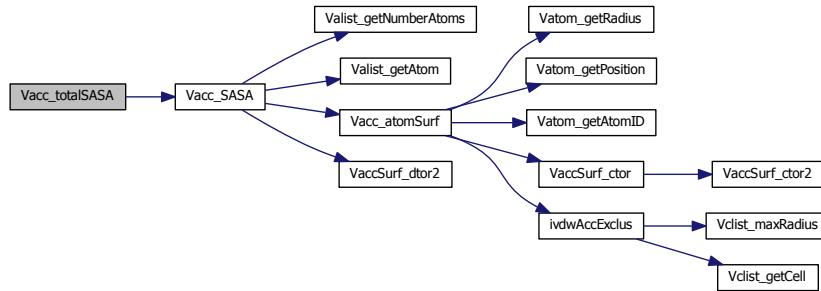
Total solvent accessible area (A^2)

Parameters

| | |
|---------------------|--|
| <code>thee</code> | Accessibility object |
| <code>radius</code> | Probe molecule radius (\AA) |

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

Here is the call graph for this function:



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

Return the total solvent accessible volume (SAV)

Note

Alias for Vacc_SAV

Author

David Gohara

Returns

Total solvent accessible volume (\AA^3)

Parameters

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

Definition at line 1521 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

Report van der Waals accessibility.

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

Author

Nathan Baker

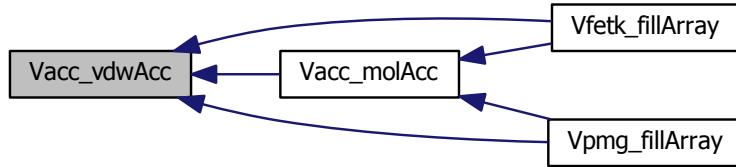
Returns

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

Parameters

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

Here is the caller graph for this function:



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

Return the WCA integral energy.

Author

David Gohara

Returns

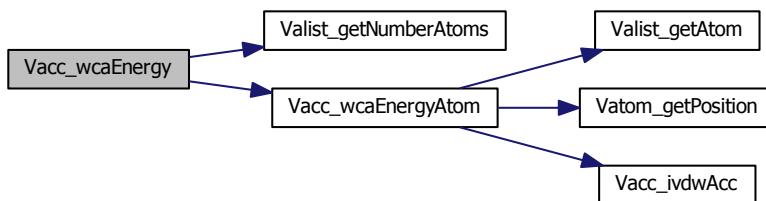
Success flag

Parameters

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

Definition at line 1739 of file [vacc.c](#).

Here is the call graph for this function:



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

Calculate the WCA energy for an atom.

Author

Dave Gohara and Nathan Baker

Returns

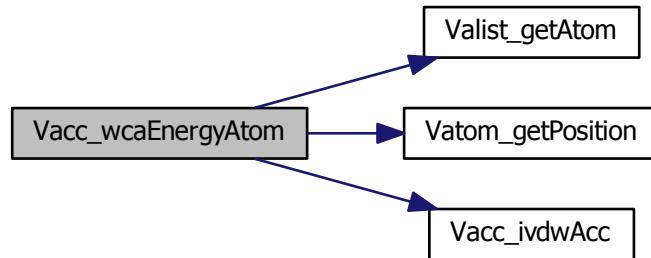
Success flag

Parameters

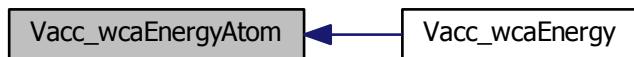
| | |
|-----------------|---------------------------------------|
| <i>thee</i> | Accessibility object |
| <i>apolparm</i> | Apolar calculation parameters |
| <i>alist</i> | Atom list |
| <i>clist</i> | Cell list associated with Vacc object |
| <i>iatom</i> | Index for atom of interest |
| <i>value</i>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

Definition at line 1598 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

Return the WCA integral force.

Author

David Gohara

Returns

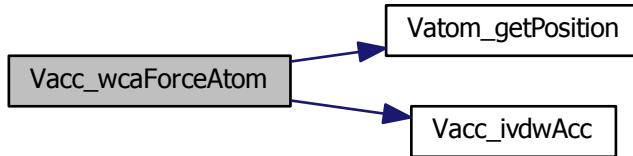
WCA energy (kJ/mol/A)

Parameters

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

Definition at line 1774 of file [vacc.c](#).

Here is the call graph for this function:



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

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

Author

Nathan Baker

Returns

Newly allocated and constructed surface object

Parameters

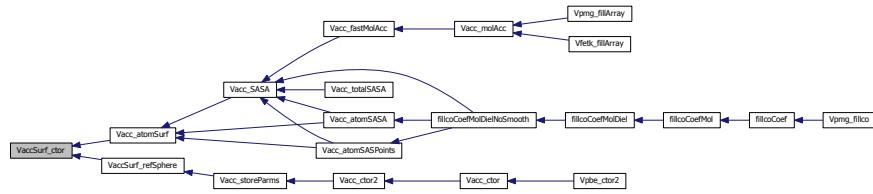
| | |
|---------------------|--------------------------------------|
| <i>mem</i> | Memory manager (can be VNULL) |
| <i>probe_radius</i> | Probe radius (in A) for this surface |
| <i>nsphere</i> | Number of points in sphere |

Definition at line 809 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

Author

Nathan Baker

Returns

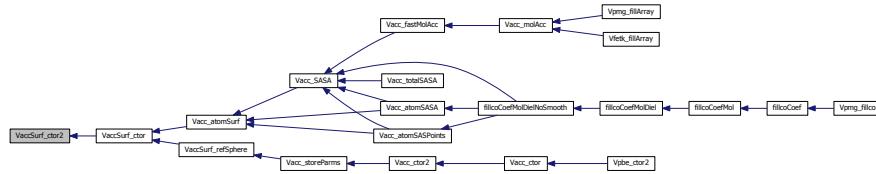
1 if successful, 0 otherwise

Parameters

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

Definition at line 824 of file [vacc.c](#).

Here is the caller graph for this function:



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

Destroy the surface object and free its memory.

Author

Nathan Baker

Parameters

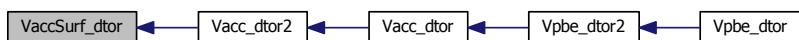
| | |
|-------------|------------------------|
| <i>thee</i> | Object to be destroyed |
|-------------|------------------------|

Definition at line 850 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.9.2.33 VEXTERNC void VaccSurf_dtor2 (VaccSurf * *thee*)

Destroy the surface object.

Author

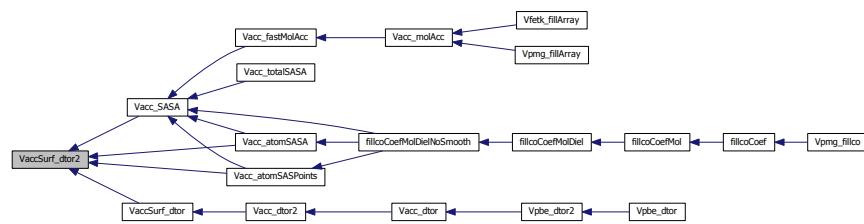
Nathan Baker

Parameters

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

Definition at line 864 of file [vacc.c](#).

Here is the caller graph for this function:



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

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

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

Note

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

Author

Nathan Baker (original FORTRAN code by Mike Gilson)

Returns

Reference sphere surface object

Parameters

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

Definition at line 944 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.10 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)
FORTRAN stub to construct the atom list object.
- VEXTERNC void [Valist_dtor](#) ([Valist](#) **thee)
Destroys atom list object.
- VEXTERNC void [Valist_dtor2](#) ([Valist](#) *thee)
FORTRAN stub to destroy atom list object.
- VEXTERNC [Vrc_Codes](#) [Valist_readPQR](#) ([Valist](#) *thee, [Vparam](#) *param, [Vio](#) *sock)
Fill atom list with information from a PQR file.
- VEXTERNC [Vrc_Codes](#) [Valist_readPDB](#) ([Valist](#) *thee, [Vparam](#) *param, [Vio](#) *sock)

Fill atom list with information from a PDB file.

- VEXTERNC Vrc_Codes [Valist_readXML](#) (*Valist *thee, Vparam *param, Vio *sock*)

Fill atom list with information from an XML file.

- VEXTERNC Vrc_Codes [Valist_getStatistics](#) (*Valist *thee*)

Load up Valist with various statistics.

7.10.1 Detailed Description

Container class for list of atom objects.

7.10.2 Function Documentation

7.10.2.1 VEXTERNC Valist* Valist_ctor()

Construct the atom list object.

Author

Nathan Baker

Returns

Pointer to newly allocated (empty) atom list

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

Here is the call graph for this function:



7.10.2.2 VEXTERNC Vrc_Codes Valist_ctor2(Valist * thee)

FORTRAN stub to construct the atom list object.

Author

Nathan Baker, Yong Huang

Returns

Success enumeration

Parameters

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

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

Here is the caller graph for this function:



7.10.2.3 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 168 of file [valist.c](#).

Here is the call graph for this function:



7.10.2.4 VEXTERNC void Valist_dtor2 (Valist * *thee*)

FORTRAN stub to destroy atom list object.

Author

Nathan Baker

Parameters

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

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

Here is the caller graph for this function:

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

Get pointer to particular atom in list.

Author

Nathan Baker

Returns

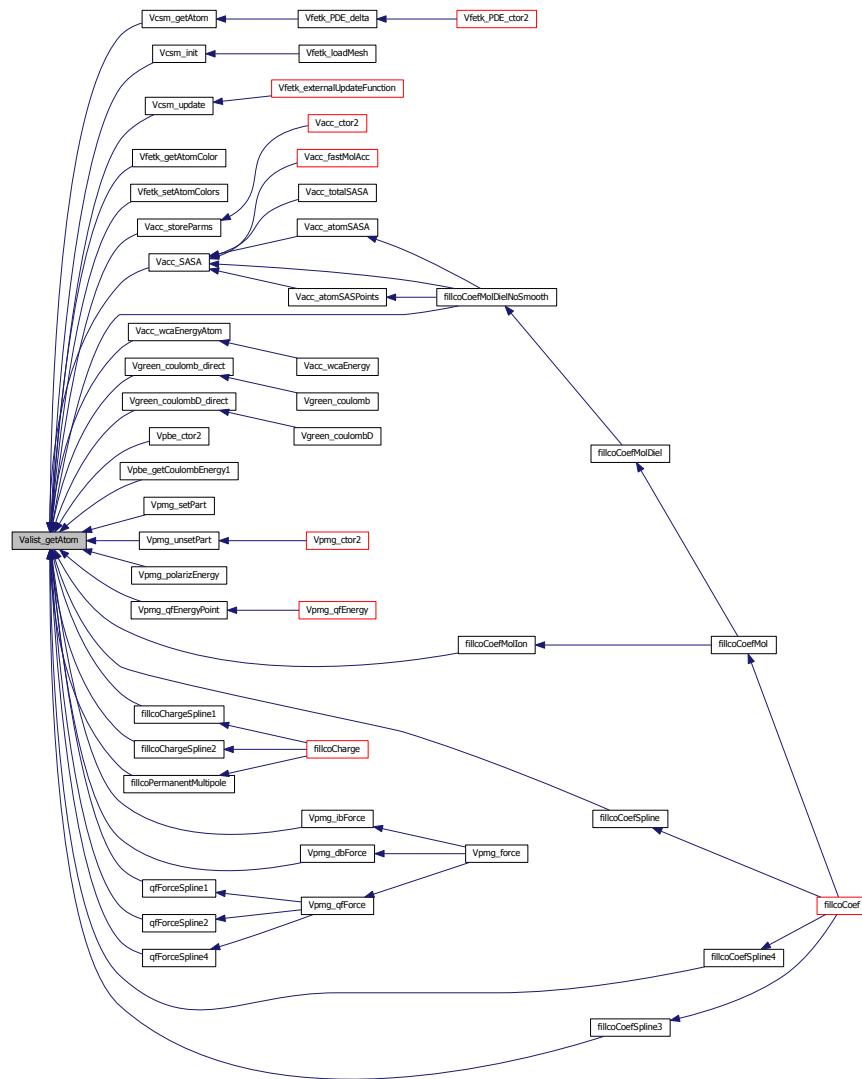
Pointer to atom object *i*

Parameters

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

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

Here is the caller graph for this function:



7.10.2.6 VEXTERNC Vatom* Valist_getAtomList (Valist *thee)

Get actual array of atom objects from the list.

Author

Nathan Baker

Returns

Array of atom objects

Parameters

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

Definition at line 96 of file [valist.c](#).

7.10.2.7 VEXTERNC double Valist_getCenterX (Valist * *thee*)

Get x-coordinate of molecule center.

Author

Nathan Baker

Returns

X-coordinate of molecule center

Parameters

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

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

7.10.2.8 VEXTERNC double Valist_getCenterY (Valist * *thee*)

Get y-coordinate of molecule center.

Author

Nathan Baker

Returns

Y-coordinate of molecule center

Parameters

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

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

7.10.2.9 VEXTERNC double Valist_getCenterZ (Valist * *thee*)

Get z-coordinate of molecule center.

Author

Nathan Baker

Returns

Z-coordinate of molecule center

Parameters

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

Definition at line [86](#) of file [valist.c](#).

7.10.2.10 VEXTERNC int Valist_getNumberAtoms (Valist * *thee*)

Get number of atoms in the list.

Author

Nathan Baker

Returns

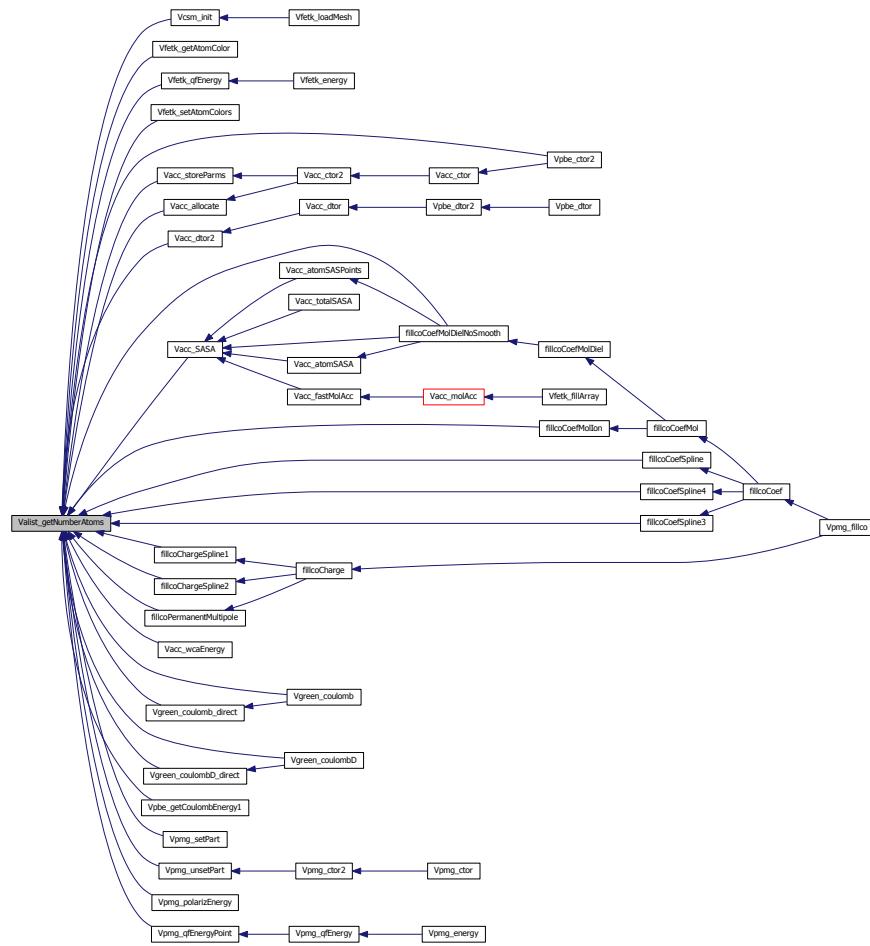
Number of atoms in list

Parameters

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

Definition at line [106](#) of file [valist.c](#).

Here is the caller graph for this function:



7.10.2.11 VEXTERNC Vrc_Codes Valist_getStatistics (Valist * thee)

Load up Valist with various statistics.

Author

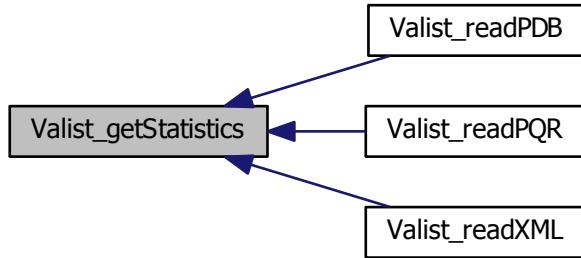
Nathan Baker, Yong Huang

Returns

Success enumeration

Definition at line 859 of file [valist.c](#).

Here is the caller graph for this function:

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

Get total memory allocated for this object and its members.

Author

Nathan Baker

Returns

Total memory in bytes

Parameters

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

Definition at line 130 of file [valist.c](#).

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

Fill atom list with information from a PDB file.

Author

Nathan Baker, Todd Dolinsky, Yong Huang

Returns

Success enumeration

Note

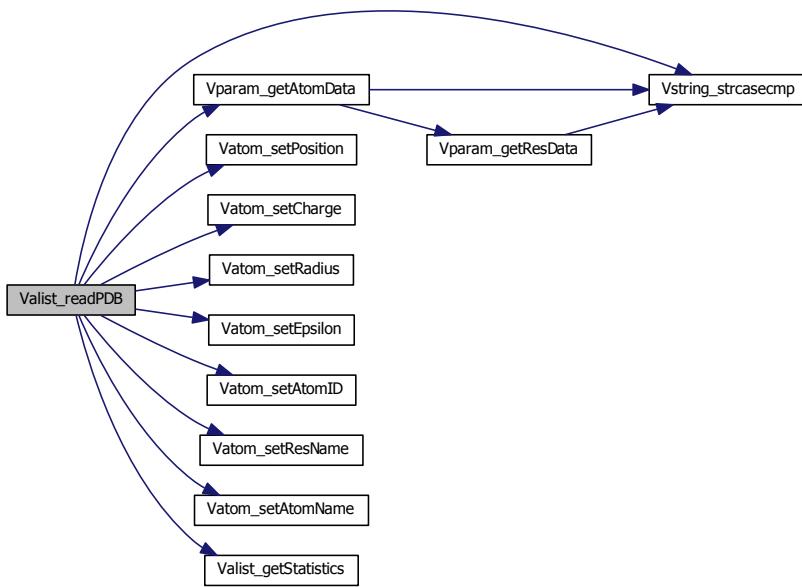
We don't actually respect PDB format; instead recognize whitespace- or tab-delimited fields which allows us to deal with structures with coordinates > 999 or < -999.

Parameters

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

Definition at line 516 of file [valist.c](#).

Here is the call graph for this function:



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

Fill atom list with information from a PQR file.

Author

Nathan Baker, Yong Huang

Returns

Success enumeration

Note

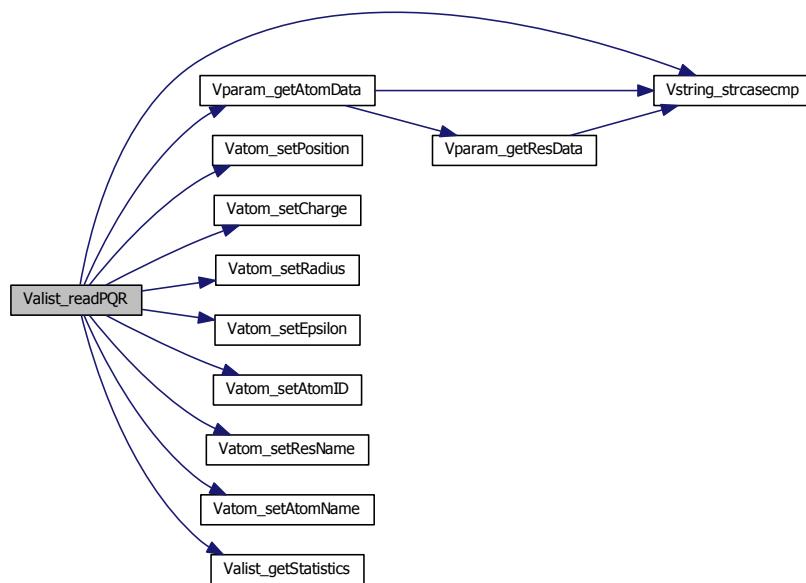
- A PQR file has PDB structure with charge and radius in the last two columns instead of weight and occupancy
- We don't actually respect PDB format; instead recognize whitespace- or tab-delimited fields which allows us to deal with structures with coordinates > 999 or < -999.

Parameters

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

Definition at line 607 of file [valist.c](#).

Here is the call graph for this function:



7.10.2.15 VEXTERN C Vrc_Codes Valist_readXML (Valist *thee, Vparam *param, Vio *sock)

Fill atom list with information from an XML file.

Author

Todd Dolinsky, Yong Huang

Returns

Success enumeration

Note

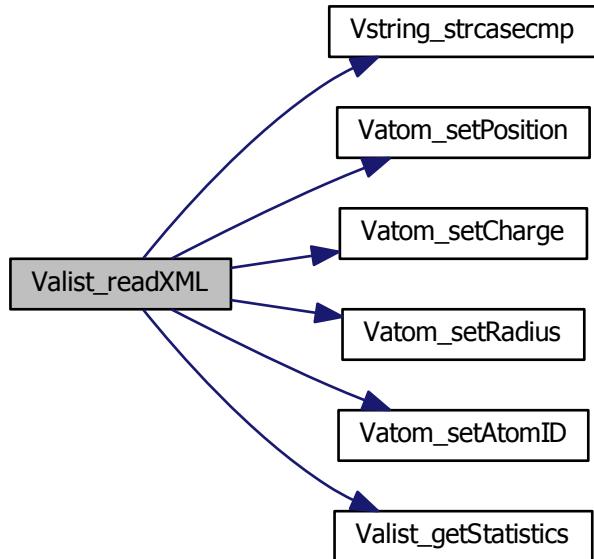
- The XML file must adhere to some guidelines, notably the presence of an <atom> tag with all other useful information (x, y, z, charge, and radius) as nested elements.

Parameters

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

Definition at line 715 of file [valist.c](#).

Here is the call graph for this function:



7.11 Vatom class

Atom class for interfacing APBS with PDB files.

Files

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

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.

7.11.1 Detailed Description

Atom class for interfacing APBS with PDB files.

7.11.2 Macro Definition Documentation

7.11.2.1 #define VMAX_RECLEN 64

Residue name length.

Author

Nathan Baker, David Gohara, Mike Schneiders

Definition at line 74 of file [vatom.h](#).

7.11.3 Function Documentation

7.11.3.1 VEXTERNC void Vatom_copyFrom (Vatom * *thee*, Vatom * *src*)

Copy information to another atom.

Author

Nathan Baker

Parameters

| | |
|-------------|----------------------------------|
| <i>thee</i> | Destination for atom information |
| <i>src</i> | Source for atom information |

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

Here is the call graph for this function:



7.11.3.2 VEXTERNC void Vatom_copyTo (Vatom * *thee*, Vatom * *dest*)

Copy information to another atom.

Author

Nathan Baker

Parameters

| | |
|-------------|----------------------------------|
| <i>thee</i> | Source for atom information |
| <i>dest</i> | Destination for atom information |

Definition at line 166 of file [vatom.c](#).

Here is the caller graph for this function:



7.11.3.3 VEXTERNC Vatom* Vatom_ctor()

Constructor for the Vatom class.

Author

Nathan Baker

Returns

Pointer to newly allocated Vatom object

Definition at line 131 of file [vatom.c](#).

Here is the call graph for this function:



7.11.3.4 VEXTERNC int Vatom_ctor2(Vatom * *thee*)

FORTRAN stub constructor for the Vatom class.

Author

Nathan Baker

Parameters

| | |
|-------------|--|
| <i>thee</i> | Pointer to Vatom allocated memory location |
|-------------|--|

Returns

1 if successful, 0 otherwise

Definition at line 142 of file [vatom.c](#).

Here is the caller graph for this function:

**7.11.3.5 VEXTERNC void Vatom_dtor (Vatom ** *thee*)**

Object destructor.

Author

Nathan Baker

Parameters

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

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

Here is the call graph for this function:

**7.11.3.6 VEXTERNC void Vatom_dtor2 (Vatom * *thee*)**

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

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

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

Here is the caller graph for this function:



7.11.3.7 VEXTERNC double Vatom_getAtomID (Vatom * *thee*)

Get atom ID.

Author

Nathan Baker

Parameters

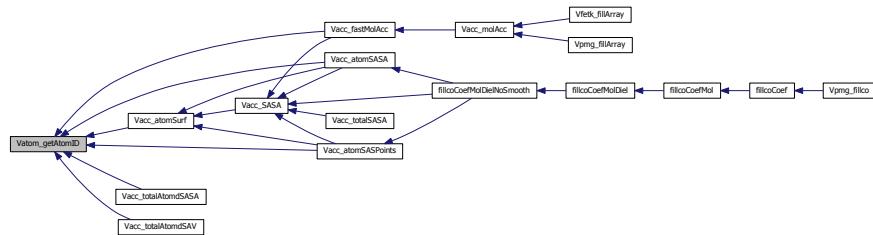
| | |
|-------------|--------------|
| <i>thee</i> | Vatom object |
|-------------|--------------|

Returns

Unique non-negative number

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

Here is the caller graph for this function:



7.11.3.8 VEXTERNC void Vatom_getAtomName (Vatom * *thee*, char *atomName*[VMAX_RECLEN])

Retrieve atom name.

Author

Jason Wagoner

Parameters

| | |
|-----------------|--------------|
| <i>thee</i> | Vatom object |
| <i>atomName</i> | Atom name |

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

7.11.3.9 VEXTERNC double Vatom_getCharge (Vatom * *thee*)

Get atomic charge.

Author

Nathan Baker

Parameters

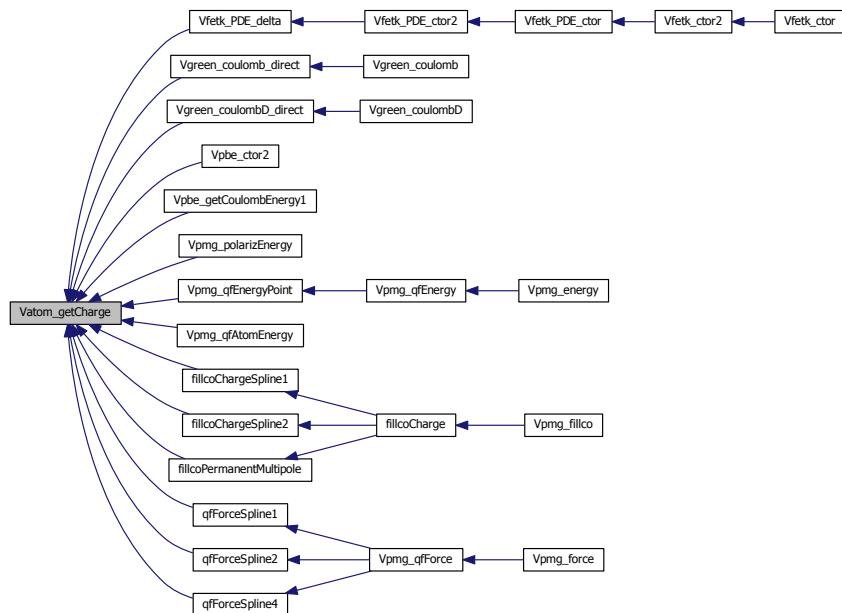
| | |
|-------------|--------------|
| <i>thee</i> | Vatom object |
|-------------|--------------|

Returns

Atom partial charge (in e)

Definition at line 120 of file [vatom.c](#).

Here is the caller graph for this function:



7.11.3.10 VEXTERNC double Vatom_getEpsilon (Vatom * *thee*)

Get atomic epsilon.

Author

David Gohara

Parameters

| | |
|-------------|--------------|
| <i>thee</i> | Vatom object |
|-------------|--------------|

Returns

Atomic epsilon (in Å)

7.11.3.11 VEXTERNC double Vatom_getPartID (Vatom * *thee*)

Get partition ID.

Author

Nathan Baker

Parameters

| | |
|-------------|--------------|
| <i>thee</i> | Vatom object |
|-------------|--------------|

Returns

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

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

Here is the caller graph for this function:

**7.11.3.12 VEXTERNC double* Vatom_getPosition (Vatom * *thee*)**

Get atomic position.

Author

Nathan Baker

Parameters

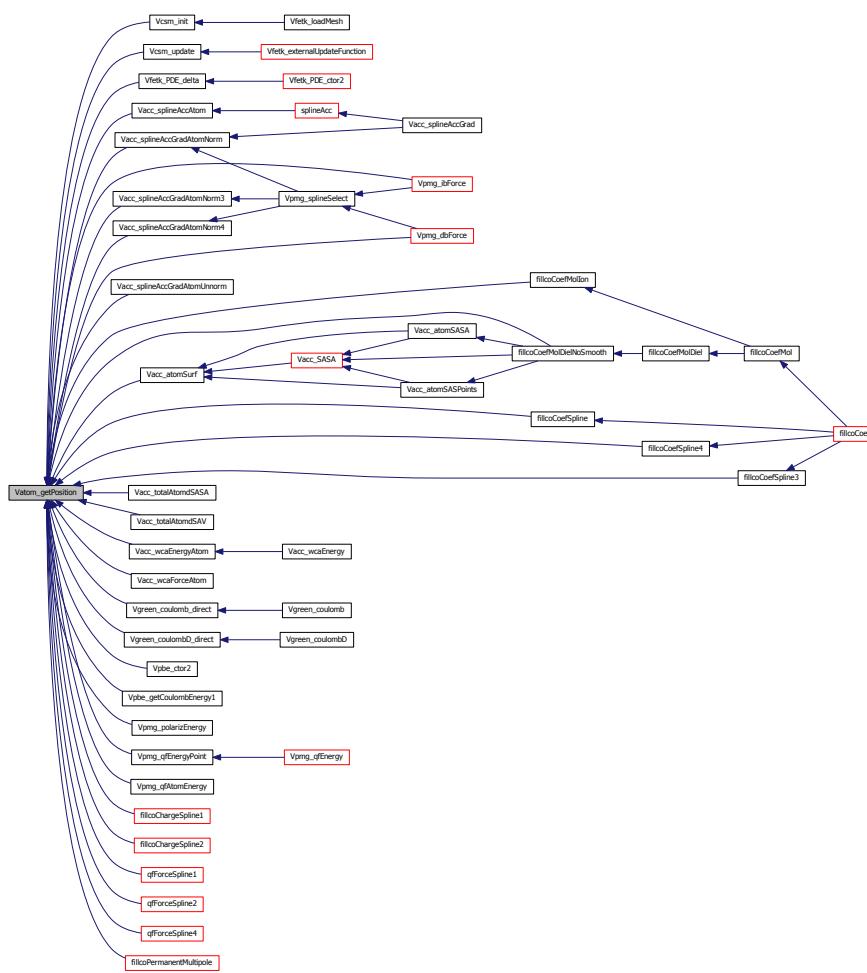
| | |
|-------------|--------------|
| <i>thee</i> | Vatom object |
|-------------|--------------|

Returns

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

Definition at line 64 of file [vatom.c](#).

Here is the caller graph for this function:

**7.11.3.13 VEXTERNC double Vatom_getRadius (Vatom * *thee*)**

Get atomic position.

Author

Nathan Baker

Parameters

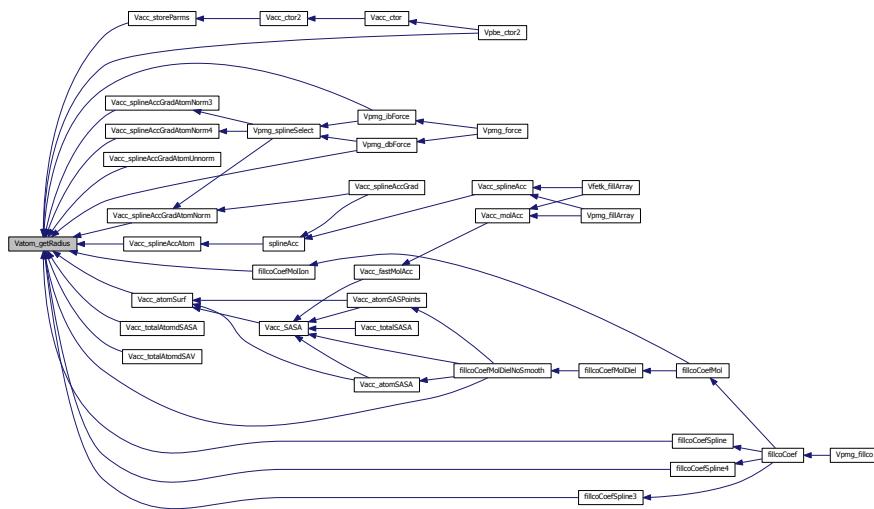
thee Vatom object

Returns

Atomic radius (in Å)

Definition at line 106 of file [vatom.c](#).

Here is the caller graph for this function:



7.11.3.14 VEXTERNC void Vatom_getResName (Vatom * *thee*, char *resName*[VMAX_RECLEN])

Retrieve residue name.

Author

Jason Wagoner

Parameters

| | |
|----------------|--------------|
| <i>thee</i> | Vatom object |
| <i>resName</i> | Residue Name |

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

7.11.3.15 VEXTERNC unsigned long int Vatom_memChk (**Vatom * *thee*)**

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

Author

Nathan Baker

Parameters

| | |
|-------------|-------------|
| <i>thee</i> | Vpmg object |
|-------------|-------------|

Returns

The memory used by this structure and its contents in bytes

Definition at line 127 of file [vatom.c](#).

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

Set atom ID.

Author

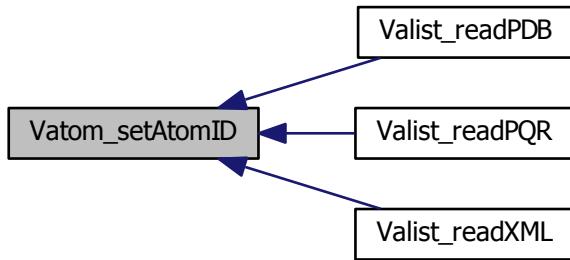
Nathan Baker

Parameters

| | |
|-------------|----------------------------|
| <i>thee</i> | Vatom object |
| <i>id</i> | Unique non-negative number |

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

Here is the caller graph for this function:



7.11.3.17 VEXTERNC void Vatom_setAtomName (*Vatom* * *thee*, char *atomName*[*VMAX_RECLEN*])

Set atom name.

Author

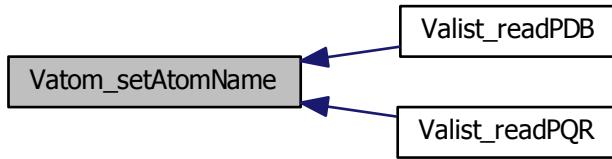
Jason Wagoner

Parameters

| | |
|-----------------|--------------|
| <i>thee</i> | Vatom object |
| <i>atomName</i> | Atom name |

Definition at line 196 of file [vatom.c](#).

Here is the caller graph for this function:



7.11.3.18 VEXTERNC void Vatom_setCharge (*Vatom* * *thee*, double *charge*)

Set atomic charge.

Author

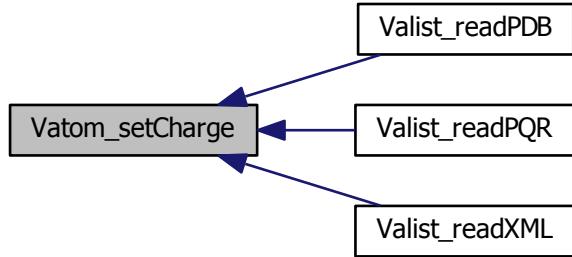
Nathan Baker

Parameters

| | |
|---------------|----------------------------|
| <i>thee</i> | Vatom object |
| <i>charge</i> | Atom partial charge (in e) |

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

Here is the caller graph for this function:



7.11.3.19 VEXTERNC void Vatom_setEpsilon (Vatom * *thee*, double *epsilon*)

Set atomic epsilon.

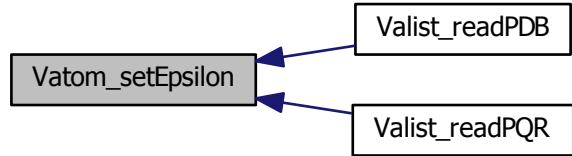
Author

David Gohara

Parameters

| | |
|----------------|-----------------------|
| <i>thee</i> | Vatom object |
| <i>epsilon</i> | Atomic epsilon (in Å) |

Here is the caller graph for this function:



7.11.3.20 VEXTERNC void Vatom_setPartID (Vatom * *thee*, int *partID*)

Set partition ID.

Author

Nathan Baker

Parameters

| | |
|---------------|---|
| <i>thee</i> | Vatom object |
| <i>partID</i> | Partition ID; a negative value means this atom is not assigned to any partition |

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

Here is the caller graph for this function:



7.11.3.21 VEXTERNC void Vatom_setPosition (Vatom * *thee*, double *position*[3])

Set the atomic position.

Author

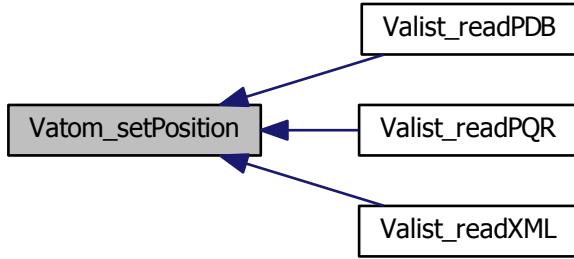
Nathan Baker

Parameters

| | |
|-----------------|-----------------------------|
| <i>thee</i> | Vatom object to be modified |
| <i>position</i> | Coordinates (in Å) |

Definition at line 157 of file [vatom.c](#).

Here is the caller graph for this function:



7.11.3.22 VEXTERNC void Vatom_setRadius (*Vatom* * *thee*, double *radius*)

Set atomic radius.

Author

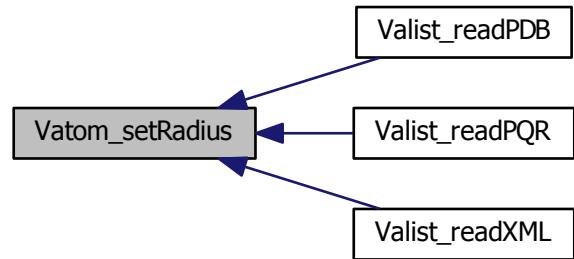
Nathan Baker

Parameters

| | |
|---------------|----------------------|
| <i>thee</i> | Vatom object |
| <i>radius</i> | Atomic radius (in Å) |

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

Here is the caller graph for this function:



7.11.3.23 VEXTERNC void Vatom_setResName (Vatom * *thee*, char *resName*[VMAX_RECLEN])

Set residue name.

Author

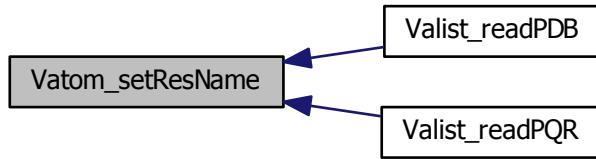
Jason Wagoner

Parameters

| | |
|----------------|--------------|
| <i>thee</i> | Vatom object |
| <i>resName</i> | Residue Name |

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

Here is the caller graph for this function:



7.12 Vcap class

Collection of routines which cap certain exponential and hyperbolic functions.

Files

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

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.

7.12.1 Detailed Description

Collection of routines which cap certain exponential and hyperbolic functions.

Note

These routines are based on FORTRAN code by Mike Holst

7.12.2 Function Documentation

7.12.2.1 VEXTERNC double Vcap_cosh (double x, int * ichop)

Provide a capped cosh() function.

If the argument x of Vcap_cosh() exceeds EXPMAX or EXPMIN, then we return cosh(EXPMAX) or cosh(EXPMIN) rather than cosh(x).

Note

Original FORTRAN routine from PMG library by Mike Holst Original notes: to control overflow in the hyperbolic and exp functions, note that the following are the argument limits of the various functions on various machines after which overflow occurs: Convex C240, Sun 3/60, Sun SPARC, IBM RS/6000: sinh, cosh, exp: maximal argument (abs value) = 88.0d0 dsinh, dcosh, dexp: maximal argument (abs value) = 709.0d0

Author

Nathan Baker (based on FORTRAN code by Mike Holst)

Returns

$\cosh(x)$ or capped equivalent

Parameters

| | |
|--------------|--|
| <i>x</i> | Argument to $\cosh()$ |
| <i>ichop</i> | Set to 1 if function capped, 0 otherwise |

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

7.12.2.2 VEXTERNC double Vcap_exp (double *x*, int * *ichop*)

Provide a capped $\exp()$ function.

If the argument *x* of $\text{Vcap_exp}()$ exceeds EXPMAX or EXPMIN, then we return $\exp(\text{EXPMAX})$ or $\exp(\text{EXPMIN})$ 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

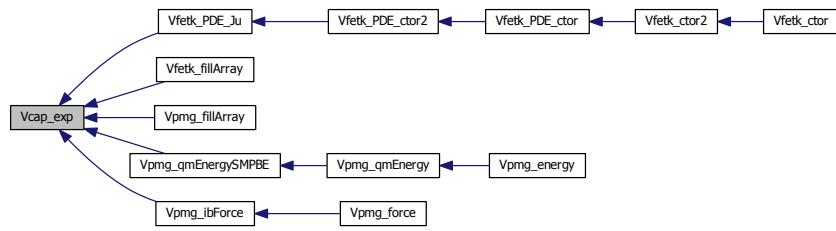
$\exp(x)$ or capped equivalent

Parameters

| | |
|--------------|--|
| <i>x</i> | Argument to $\exp()$ |
| <i>ichop</i> | Set to 1 if function capped, 0 otherwise |

Definition at line 60 of file [vcap.c](#).

Here is the caller graph for this function:



7.12.2.3 VEXTERNC double Vcap_sinh (double x, int * ichop)

Provide a capped sinh() function.

If the argument x of `Vcap_sinh()` exceeds EXPMAX or EXPMIN, then we return $\sinh(\text{EXPMAX})$ or $\sinh(\text{EXPMIN})$ rather than $\sinh(x)$.

Note

Original FORTRAN routine from PMG library by Mike Holst Original notes: to control overflow in the hyperbolic and exp functions, note that the following are the argument limits of the various functions on various machines after which overflow occurs: Convex C240, Sun 3/60, Sun SPARC, IBM RS/6000: sinh, cosh, exp: maximal argument (abs value) = 88.0d0 dsinh, dcosh, dexp: maximal argument (abs value) = 709.0d0

Author

Nathan Baker (based on FORTRAN code by Mike Holst)

Returns

$\sinh(x)$ or capped equivalent

Parameters

| | |
|---------|--|
| x | Argument to $\sinh()$ |
| $ichop$ | Set to 1 if function capped, 0 otherwise |

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

7.13 Vclist class

Atom cell list.

Files

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

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 A) the cell list was constructed with.
- VEXTERNC [Vclist](#) * [Vclist_ctor](#) ([Valist](#) *alist, double max_radius, int npts[VAPBS_DIM], [Vclist_DomainMode](#) mode, double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM])
Construct the cell list object.
- VEXTERNC [Vrc_Codes](#) [Vclist_ctor2](#) ([Vclist](#) *thee, [Valist](#) *alist, double max_radius, int npts[VAPBS_DIM], [Vclist_DomainMode](#) mode, double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM])
FORTRAN stub to construct the cell list object.
- VEXTERNC void [Vclist_dtor](#) ([Vclist](#) **thee)
Destroy object.

- VEXTERNC void `Vclist_dtor2 (Vclist *thee)`
FORTRAN stub to destroy object.
- VEXTERNC `VclistCell * Vclist_getCell (Vclist *thee, double position[VAPBS_DIM])`
Return cell corresponding to specified position or return VNULL.
- VEXTERNC `VclistCell * VclistCell_ctor (int natoms)`
Allocate and construct a cell list cell object.
- VEXTERNC `Vrc_Codes VclistCell_ctor2 (VclistCell *thee, int natoms)`
Construct a cell list object.
- VEXTERNC void `VclistCell_dtor (VclistCell **thee)`
Destroy object.
- VEXTERNC void `VclistCell_dtor2 (VclistCell *thee)`
FORTRAN stub to destroy object.

7.13.1 Detailed Description

Atom cell list.

7.13.2 Enumeration Type Documentation

7.13.2.1 enum eVclist_DomainMode

Atom cell list domain setup mode.

Author

Nathan Baker

Enumerator:

`CLIST_AUTO_DOMAIN` Setup the cell list domain automatically to encompass the entire molecule

`CLIST_MANUAL_DOMAIN` Specify the cell list domain manually through the constructor

Definition at line 79 of file `vclist.h`.

7.13.3 Function Documentation

7.13.3.1 VEXTERNC `Vclist* Vclist_ctor (Valist * alist, double max_radius, int npts[VAPBS_DIM], Vclist_DomainMode mode, double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM])`

Construct the cell list object.

Author

Nathan Baker

Returns

Newly allocated Vclist object

Parameters

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

Definition at line 80 of file [vclist.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.3.2 VEXTERNC Vrc_Codes Vclist_ctor2 (*Vclist * thee, Valist * alist, double max_radius, int npts[VAPBS_DIM], Vclist_DomainMode mode, double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM]*)

FORTRAN stub to construct the cell list object.

Author

Nathan Baker, Yong Huang

Returns

Success enumeration

Parameters

| | |
|---------------------|--|
| <i>thee</i> | Memory for Vclist object |
| <i>alist</i> | Molecule for cell list queries |
| <i>max_radius</i> | Max probe radius (\AA) to be queried |
| <i>npts</i> | Number of in hash table points in each direction |
| <i>mode</i> | Mode to construct table |
| <i>lower_corner</i> | Hash table lower corner for manual construction (see mode variable); ignored otherwise |
| <i>upper_corner</i> | Hash table upper corner for manual construction (see mode variable); ignored otherwise |

Definition at line 348 of file [vclist.c](#).

Here is the caller graph for this function:



7.13.3.3 VEXTERNC void Vclist_dtor (Vclist ** *thee*)

Destroy object.

Author

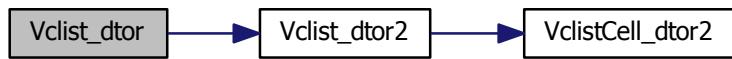
Nathan Baker

Parameters

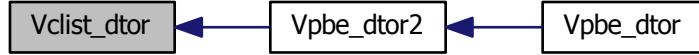
| | |
|-------------|--------------------------------------|
| <i>thee</i> | Pointer to memory location of object |
|-------------|--------------------------------------|

Definition at line 402 of file [vclist.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.3.4 VEXTERNC void Vclist_dtor2 (*Vclist* * *thee*)

FORTRAN stub to destroy object.

Author

Nathan Baker

Parameters

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

Definition at line 413 of file [vclist.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.3.5 VEXTERNC VclistCell* Vclist_getCell (Vclist * thee, double position[VAPBS_DIM])

Return cell corresponding to specified position or return VNULL.

Author

Nathan Baker

Returns

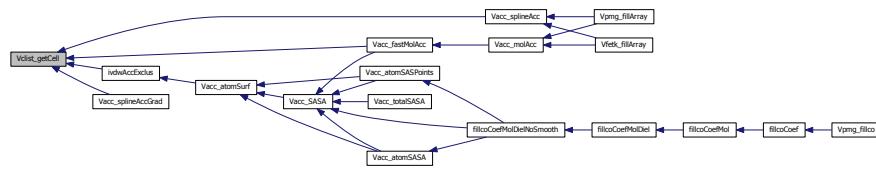
Pointer to VclistCell object or VNULL if no cell available (away from molecule).

Parameters

| | |
|-----------------|-----------------------------|
| <i>thee</i> | Pointer to Vclist cell list |
| <i>position</i> | Position to evaluate |

Definition at line 428 of file [vclist.c](#).

Here is the caller graph for this function:



7.13.3.6 VEXTERNC double Vclist_maxRadius (Vclist * thee)

Get the max probe radius value (in A) the cell list was constructed with.

Author

Nathan Baker

Returns

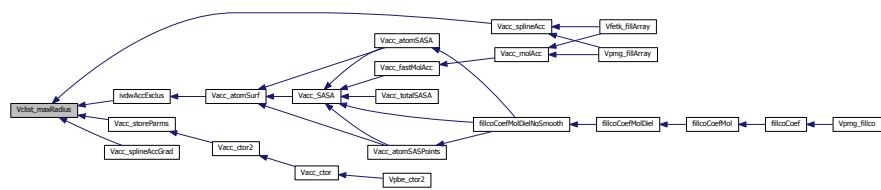
Max probe radius (in A)

Parameters

| | |
|-------------|------------------|
| <i>thee</i> | Cell list object |
|-------------|------------------|

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

Here is the caller graph for this function:



7.13.3.7 VEXTERNC unsigned long int Vclist_memChk (Vclist * thee)

Get number of bytes in this object and its members.

Author

Nathan Baker

Returns

Number of bytes allocated for object

Parameters

thee | Object for memory check

Definition at line 68 of file [vclist.c](#).

7.13.3.8 VEXTERNC VclistCell* VclistCell_ctor (int *natoms*)

Allocate and construct a cell list cell object.

Author

Nathan Baker

Returns

Pointer to newly-allocated and constructed object.

Parameters

natoms Number of atoms associated with this cell

Definition at line 457 of file `vclist.c`.

Here is the call graph for this function:



7.13.3.9 VEXTERNC Vrc_Codes VclistCell_ctor2(VclistCell * *thee*, int *natoms*)

Construct a cell list object.

Author

Nathan Baker, Yong Huang

Returns

Success enumeration

Parameters

| | |
|---------------|---|
| <i>thee</i> | Memory location for object |
| <i>natoms</i> | Number of atoms associated with this cell |

Definition at line 469 of file [vclist.c](#).

Here is the caller graph for this function:



7.13.3.10 VEXTERNC void VclistCell_dtor(VclistCell ** *thee*)

Destroy object.

Author

Nathan Baker

Parameters

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

Definition at line 491 of file [vclist.c](#).

Here is the call graph for this function:



7.13.3.11 VEXTERNC void VclistCell_dtor2(VclistCell * *thee*)

FORTRAN stub to destroy object.

Author

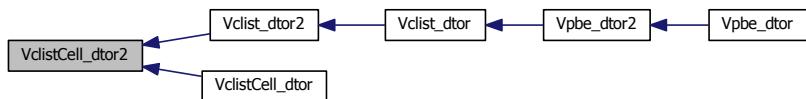
Nathan Baker

Parameters

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

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

Here is the caller graph for this function:



7.14 Vgreen class

Provides capabilities for pointwise evaluation of free space Green's function for point charges in a uniform dielectric.

Files

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

Data Structures

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

TypeDefs

- typedef struct [sVgreen](#) [Vgreen](#)
Declaration of the Vgreen class as the Vgreen structure.

Functions

- VEXTERNC [Valist](#) * [Vgreen_getValist](#) ([Vgreen](#) *thee)
Get the atom list associated with this Green's function object.
- VEXTERNC unsigned long int [Vgreen_memChk](#) ([Vgreen](#) *thee)
Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC [Vgreen](#) * [Vgreen_ctor](#) ([Valist](#) *alist)
Construct the Green's function oracle.
- VEXTERNC int [Vgreen_ctor2](#) ([Vgreen](#) *thee, [Valist](#) *alist)
FORTRAN stub to construct the Green's function oracle.
- VEXTERNC void [Vgreen_dtor](#) ([Vgreen](#) **thee)
Destruct the Green's function oracle.
- VEXTERNC void [Vgreen_dtor2](#) ([Vgreen](#) *thee)
FORTRAN stub to destruct the Green's function oracle.
- VEXTERNC int [Vgreen_helmholtz](#) ([Vgreen](#) *thee, int npos, double *x, double *y, double *z, double *val, double kappa)
Get the Green's function for Helmholtz's equation integrated over the atomic point charges.
- VEXTERNC int [Vgreen_helmholtzD](#) ([Vgreen](#) *thee, int npos, double **x, double *y, double *z, double *gradx, double *grady, double *gradz, double kappa)
Get the gradient of Green's function for Helmholtz's equation integrated over the atomic point charges.
- VEXTERNC int [Vgreen_coulomb_direct](#) ([Vgreen](#) *thee, int npos, double *x, double *y, double *z, double *val)
Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.
- VEXTERNC int [Vgreen_coulomb](#) ([Vgreen](#) *thee, int npos, double *x, double *y, double *z, double *val)
Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation or H. E. Johnston, R. Krasny FMM library (if available)

- VEXTERNC int `Vgreen_coulombD_direct (Vgreen *thee, int npos, double *x, double *y, double *z, double *pot, double *gradx, double *grady, double *gradz)`

Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.

- VEXTERNC int `Vgreen_coulombD (Vgreen *thee, int npos, double *x, double *y, double *z, double *pot, double *gradx, double *grady, double *gradz)`

Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using either direct summation or H. E. Johnston/R. Krasny FMM library (if available)

7.14.1 Detailed Description

Provides capabilities for pointwise evaluation of free space Green's function for point charges in a uniform dielectric.

Note

Right now, these are very slow methods without any fast multipole acceleration.

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 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.
*
*
```

7.14.2 Function Documentation

7.14.2.1 VEXTERNC int Vgreen_coulomb (Vgreen * *thee*, int *npos*, double * *x*, double * *y*, double * *z*, double * *val*)

Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation or H. E. Johnston, R. Krasny FMM library (if available)

Returns the potential \form#23 defined by

$$\phi(r) = \sum_i \frac{q_i}{r_i}$$

where q_i is the atomic charge (in e) and r_i is the distance to the observation point r . The potential is scaled to units of V.

Author

Nathan Baker

Parameters

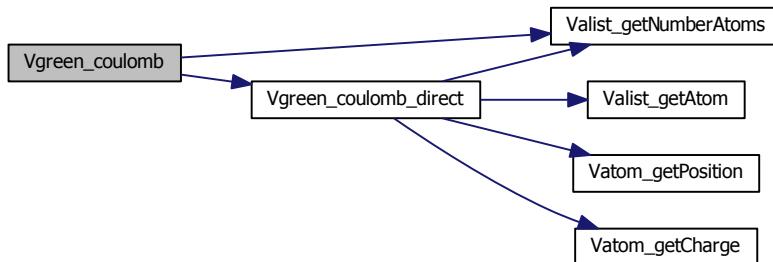
| | |
|-------------|-------------------------------------|
| <i>thee</i> | Vgreen object |
| <i>npos</i> | The number of positions to evaluate |
| <i>x</i> | The npos x-coordinates |
| <i>y</i> | The npos y-coordinates |
| <i>z</i> | The npos z-coordinates |
| <i>val</i> | The npos values |

Returns

1 if successful, 0 otherwise

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

Here is the call graph for this function:



7.14.2.2 VEXTERNC int Vgreen_coulomb_direct (*Vgreen * thee*, *int npos*, *double * x*, *double * y*, *double * z*, *double * val*)

Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.

Returns the potential \form#23 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

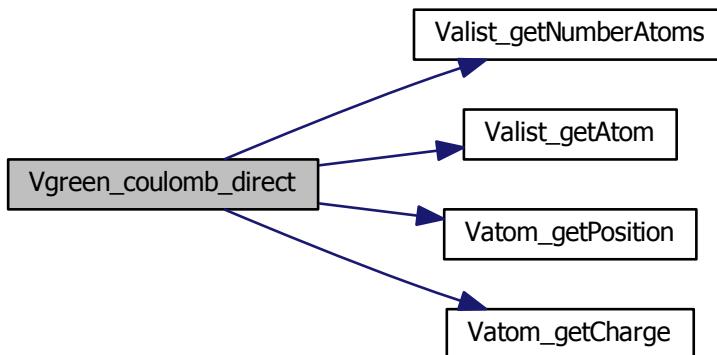
| | |
|-------------|-------------------------------------|
| <i>thee</i> | Vgreen object |
| <i>npos</i> | The number of positions to evaluate |
| <i>x</i> | The npos x-coordinates |
| <i>y</i> | The npos y-coordinates |
| <i>z</i> | The npos z-coordinates |
| <i>val</i> | The npos values |

Returns

1 if successful, 0 otherwise

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.2.3 VEXTERNC int Vgreen_coulombD (Vgreen * *thee*, int *npos*, double * *x*, double * *y*, double * *z*, double * *pot*, double * *gradx*, double * *grady*, double * *gradz*)

Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using either direct summation or H. E. Johnston/R. Krasny FMM library (if available)

Returns the field \form#28 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

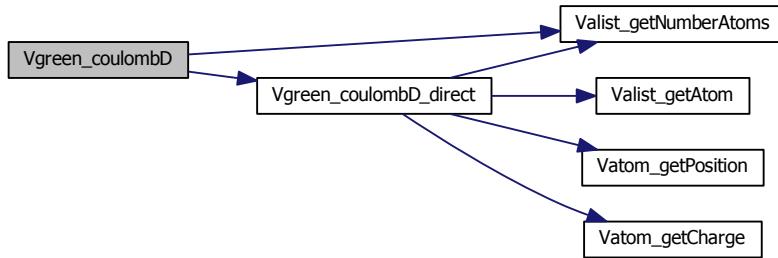
| | |
|--------------|-------------------------------------|
| <i>thee</i> | Vgreen object |
| <i>npos</i> | The number of positions to evaluate |
| <i>x</i> | The npos x-coordinates |
| <i>y</i> | The npos y-coordinates |
| <i>z</i> | The npos z-coordinates |
| <i>pot</i> | The npos potential values |
| <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 363 of file [vgreen.c](#).

Here is the call graph for this function:



7.14.2.4 VEXTERNC int Vgreen_coulombD_direct (**Vgreen * *thee*, int *npos*, double * *x*, double * *y*, double * *z*, double * *pot*, double * *gradx*, double * *grady*, double * *gradz*)**

Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.

Returns the field \form#28 defined by

$$\nabla\phi(r) = \sum_i \frac{q_i}{r_i}$$

where q_i is the atomic charge (in e) and r_i is the distance to the observation point r . The field is scaled to units of V/Å.

Author

Nathan Baker

Parameters

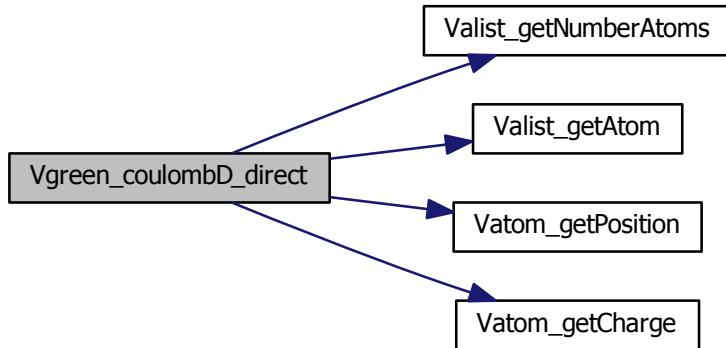
| | |
|--------------|-------------------------------------|
| <i>thee</i> | Vgreen object |
| <i>npos</i> | The number of positions to evaluate |
| <i>x</i> | The npos x-coordinates |
| <i>y</i> | The npos y-coordinates |
| <i>z</i> | The npos z-coordinates |
| <i>pot</i> | The npos potential values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

Returns

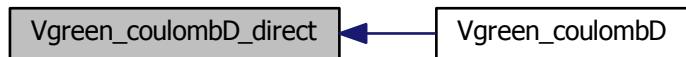
1 if successful, 0 otherwise

Definition at line 311 of file [vgreen.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.2.5 VEXTERNC `Vgreen*` `Vgreen_ctor (Valist * alist)`

Construct the Green's function oracle.

Author

Nathan Baker

Parameters

| | |
|--------------------|---|
| <code>alist</code> | Atom (charge) list associated with object |
|--------------------|---|

Returns

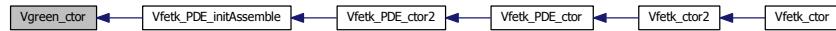
Pointer to newly allocated Green's function oracle

Definition at line 157 of file [vgreen.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.2.6 VEXTERNC int Vgreen_ctor2 (**Vgreen** * *thee*, **Valist** * *alist*)

FORTRAN stub to construct the Green's function oracle.

Author

Nathan Baker

Parameters

| | |
|--------------|---|
| <i>thee</i> | Pointer to memory allocated for object |
| <i>alist</i> | Atom (charge) list associated with object |

Returns

1 if successful, 0 otherwise

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

Here is the caller graph for this function:



7.14.2.7 VEXTERNC void Vgreen_dtor (Vgreen ** *thee*)

Destruct the Green's function oracle.

Author

Nathan Baker

Parameters

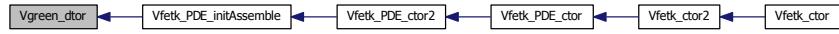
| | |
|-------------|---------------------------------------|
| <i>thee</i> | Pointer to memory location for object |
|-------------|---------------------------------------|

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.2.8 VEXTERNC void Vgreen_dtor2 (Vgreen * *thee*)

FORTRAN stub to destruct the Green's function oracle.

Author

Nathan Baker

Parameters

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

Definition at line 201 of file [vgreen.c](#).

Here is the caller graph for this function:



7.14.2.9 VEXTERNC Valist* Vgreen_getValist (**Vgreen** * *thee*)

Get the atom list associated with this Green's function object.

Author

Nathan Baker

Parameters

| | |
|-------------|---------------|
| <i>thee</i> | Vgreen object |
|-------------|---------------|

Returns

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

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

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

Get the Green's function for Helmholtz's equation integrated over the atomic point charges.

Returns the potential \form#23 defined by

$$\phi(r) = \sum_i \frac{q_i e^{-\kappa r_i}}{r_i}$$

where \form#25 is the inverse screening length (in Å)

q_i is the atomic charge (in e), and *r_i* is the distance from atom *i* to the observation point *r*. The potential is scaled to units of V.

Author

Nathan Baker

Bug Not implemented yet

Note

Not implemented yet

Parameters

| | |
|--------------|-----------------------------------|
| <i>thee</i> | Vgreen object |
| <i>npos</i> | Number of positions to evaluate |
| <i>x</i> | The 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

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

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

Get the gradient of Green's function for Helmholtz's equation integrated over the atomic point charges.

Returns the field \form#28 defined by

$$\nabla\phi(r) = \nabla \sum_i \frac{q_i e^{-\kappa r_i}}{r_i}$$

where \form#25 is the inverse screening length (in Å).

q_i is the atomic charge (in e), and r_i is the distance from atom i to the observation point r . The potential is scaled to units of V/Å.

Author

Nathan Baker

Bug Not implemented yet

Note

Not implemented yet

Parameters

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

Returns

int 1 if sucessful, 0 otherwise

Definition at line 217 of file [vgreen.c](#).

7.14.2.12 VEXTERNC unsigned long int Vgreen_memChk(Vgreen * *thee*)

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

Author

Nathan Baker

Parameters

| | |
|-------------|---------------|
| <i>thee</i> | Vgreen object |
|-------------|---------------|

Returns

The memory used by this structure and its contents in bytes

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

7.15 Vhal class

A "class" which consists solely of macro definitions which are used by several other classes.

Files

- file [vhal.h](#)

Contains generic macro definitions for APBS.

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`

- **#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.
- **#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, arg)**
- **#define VWARN_MSG2(cnd, msg, arg0, arg1)**
- **#define VABORT_MSG0(msg)**
- **#define VABORT_MSG1(msg, arg)**
- **#define VABORT_MSG2(msg, arg0, arg1)**

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_TRI = 0, VCM_BSPL2 = 1, VCM_BSPL4 = 2 }**
Types of charge discretization methods.
- **enum eVchrg_Src { VCM_CHARGE = 0, VCM_PERMANENT = 1, VCM_INDUCED = 2, VCM_NLINDUCED = 3 }**
Charge source.
- **enum eVdata_Type { VDT_CHARGE, VDT_POT, VDT_ATOMPOT, VDT_SMOL, VDT_SSPL, VDT_VDW, VDT_IVDW, VDT_LAP, VDT_EDENS, VDT_NDENS, VDT_QDENS, VDT_DIELX, VDT_DIELY, VDT_DIELZ, VDT_KAPPA }**
Types of (scalar) data that can be written out of APBS.
- **enum eVdata_Format { VDF_DX = 0, VDF_UHBD = 1, VDF_AVIS = 2, VDF_MCSF = 3, VDF_GZ = 4, VDF_FLAT = 5 }**
Format of data for APBS I/O.

7.15.1 Detailed Description

A "class" which consists solely of macro definitions which are used by several other classes.

7.15.2 Macro Definition Documentation

7.15.2.1 #define MAX_SPHERE PTS 50000

Maximum number of points on a sphere.

Note

Used by VaccSurf

Definition at line 412 of file [vhal.h](#).

7.15.2.2 #define VABORT_MSG0(msg)

Value:

```
do {
    Vnm_print(2, "%s:%d [%s()]: ABORTING:\n"
              " %s\n", \
              __FILE__, __LINE__, __FUNCTION__, msg); \
    abort(); \
} while(0)
```

Prints a message and aborts

Author

Tucker Beck

Note

The do{} while(0) simply enforces that a semicolon at the end

Definition at line 725 of file [vhal.h](#).

7.15.2.3 #define VABORT_MSG1(msg, arg)

Value:

```
do {
    char buff[1000];
    snprintf(buff, 1000, msg, arg );
    Vnm_print(2, "%s:%d [%s()]: ABORTING:\n"
              " %s\n", \
              __FILE__, __LINE__, __FUNCTION__, buff); \
    abort(); \
} while(0)
```

Prints a message with an argument and aborts

Author

Tucker Beck

Note

The do{...} while(0) simply enforces that a semicolon at the end\

Definition at line [738](#) of file [vhal.h](#).

7.15.2.4 #define VABORT_MSG2(msg, arg0, arg1)**Value:**

```
do {                                \
    char buff[1000];                  \
    sprintf( buff, 1000, msg, arg0, arg1); \
    Vnm_print(2, "%s:%d [%s()]: ABORTING:\n" \
              " %s\n\n",                \
              __FILE__, __LINE__, __FUNCTION__, buff); \
    abort();                           \
} while(0)
```

Prints a message with an argument and aborts

Author

Tucker Beck

Note

The do{...} while(0) simply enforces that a semicolon at the end\

Definition at line [753](#) of file [vhal.h](#).

7.15.2.5 #define VAPBS_BACK 4

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line [436](#) of file [vhal.h](#).

7.15.2.6 #define VAPBS_DOWN 5

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line [442](#) of file [vhal.h](#).

7.15.2.7 #define VAPBS_FRONT 1

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line [418](#) of file [vhal.h](#).

7.15.2.8 #define VAPBS_LEFT 3

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line [430](#) of file [vhal.h](#).

7.15.2.9 #define VAPBS_RIGHT 0

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line [406](#) of file [vhal.h](#).

7.15.2.10 #define VAPBS_UP 2

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line [424](#) of file [vhal.h](#).

7.15.2.11 #define VASSERT_MSG0(cnd, msg)

Value:

```
do {
    if( (cnd) == 0 ) {
        Vnm_print(2, "%s:%d [%s()]: ERROR:\n"
                  "          Assertion Failed (%s): %s\n\n",
                  __FILE__, __LINE__, __FUNCTION__, #cnd, msg);
        abort();
    }
} while(0)
```

Utility assertion; if it fails prints a message and aborts

Author

Tucker Beck

Note

The do{...} while(0) simply enforces that a semicolon at the end

Definition at line 627 of file [vhal.h](#).

7.15.2.12 #define VASSERT_MSG1(cnd, msg, arg)**Value:**

```
do {
    if( (cnd) == 0 ) {
        char buff[1000];
        sprintf( buff, 1000, msg, arg );
        Vnm_print(2, "%s:%d [%s()]: ERROR:\n"
                  "           Assertion Failed (%s): %s\n\n",
                  __FILE__, __LINE__, __FUNCTION__, #cnd, buff);
        abort();
    }
} while(0)
```

Utility assertion; if it fails prints a message with an argument and aborts

Author

Tucker Beck

Note

The do{...} while(0) simply enforces that a semicolon at the end

Definition at line 642 of file [vhal.h](#).

7.15.2.13 #define VASSERT_MSG2(cnd, msg, arg0, arg1)**Value:**

```
do {
    if( (cnd) == 0 ) {
        char buff[1000];
        sprintf( buff, 1000, msg, arg0, arg1 );
        Vnm_print(2, "%s:%d [%s()]: ERROR:\n"
                  "           Assertion Failed (%s): %s\n\n",
                  __FILE__, __LINE__, __FUNCTION__, #cnd, buff);
        abort();
    }
} while(0)
```

Utility assertion; if it fails prints a message with an argument and aborts

Author

Tucker Beck

Note

The do{...} while(0) simply enforces that a semicolon at the end

Definition at line 662 of file [vhal.h](#).

7.15.2.14 #define VMBED(*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 583 of file [vhal.h](#).

7.15.2.15 #define VFLOOR(*value*) floor(*value*)

Wrapped floor to fix floating point issues in the Intel compiler.

Author

Todd Dolinsky

Definition at line 574 of file [vhal.h](#).

7.15.2.16 #define VWARN_MSG0(*cnd*, *msg*)

Value:

```
do {
    if( (cnd) == 0 ) {
        Vnm_print(2, "%s:%d [%s()]: WARNING:\n"
                  "    Condition Failed (%s): %s\n\n",
                  __FILE__, __LINE__, __FUNCTION__, #cnd, msg);
    }
} while(0)
```

Conditional warning; if true prints a message

Author

Tucker Beck

Note

The do{} while(0) simply enforces that a semicolon at the end

Definition at line 679 of file [vhal.h](#).

7.15.2.17 #define VWARN_MSG1(*cnd*, *msg*, *arg*)

Value:

```

do {
    if( (cnd) == 0 ) {
        char buff[1000];
        sprintf( buff, 1000, msg, arg );
        Vnm_print(2, "%s:%d [%s()]: WARNING:\n"
                   "    Condition Failed (%s): %s\n\n",
                   __FILE__, __LINE__, __FUNCTION__, #cnd, buff);
    }
} while(0)

```

Conditional warning; if true prints a message and an argument

Author

Tucker Beck

Note

The do{} while(0) simply enforces that a semicolon at the end

Definition at line 693 of file [vhal.h](#).

7.15.2.18 #define VWARN_MSG2(cnd, msg, arg0, arg1)

Value:

```

do {
    if( (cnd) == 0 ) {
        char buff[1000];
        sprintf( buff, 1000, msg, arg0, arg1 );
        Vnm_print(2, "%s:%d [%s()]: WARNING:\n"
                   "    Condition Failed (%s): %s\n\n",
                   __FILE__, __LINE__, __FUNCTION__, #cnd, buff);
    }
} while(0)

```

Conditional warning; if true prints a message and an argument

Author

Tucker Beck

Note

The do{} while(0) simply enforces that a semicolon at the end

Definition at line 709 of file [vhal.h](#).

7.15.3 Enumeration Type Documentation

7.15.3.1 enum eVbcfl

Types of boundary conditions.

Author

Nathan Baker

Enumerator:

- BCFL_ZERO*** Zero Dirichlet boundary conditions
- BCFL_SDH*** Single-sphere Debye-Huckel Dirichlet boundary condition
- BCFL_MDH*** Multiple-sphere Debye-Huckel Dirichlet boundary condition
- BCFL_UNUSED*** Unused boundary condition method (placeholder)
- BCFL_FOCUS*** Focusing Dirichlet boundary condition
- BCFL_MEM*** Focusing membrane boundary condition
- BCFL_MAP*** Skip first level of focusing use an external map

Definition at line 206 of file [vhal.h](#).

7.15.3.2 enum eVchrg_Meth

Types of charge discretization methods.

Author

Nathan Baker

Enumerator:

- VCM_TRIL*** Trilinear interpolation of charge to 8 nearest grid points. The traditional method; not particularly good to use with PBE forces.
- VCM_BSPL2*** Cubic B-spline across nearest- and next-nearest-neighbors. Mainly for use in grid-sensitive applications (such as force calculations).
- VCM_BSPL4*** 5th order B-spline for AMOEBA permanent multipoles.

Definition at line 229 of file [vhal.h](#).

7.15.3.3 enum eVchrg_Src

Charge source.

Author

Michael Schnieders

Enumerator:

- VCM_CHARGE*** Partial Charge source distribution
- VCM_PERMANENT*** Permanent Multipole source distribution
- VCM_INDUCED*** Induced Dipole source distribution
- VCM_NLINDUCED*** NL Induced Dipole source distribution

Definition at line 250 of file [vhal.h](#).

7.15.3.4 enum eVdata_Format

Format of data for APBS I/O.

Author

Nathan Baker

Enumerator:

- VDF_DX** OpenDX (Data Explorer) format
- VDF_UHBD** UHBD format
- VDF_AVIS** AVS UCD format
- VDF_MCSF** FEtk MC Simplex Format (MCSF)
- VDF_GZ** Binary file (GZip)
- VDF_FLAT** Write flat file

Definition at line 308 of file [vhal.h](#).

7.15.3.5 enum eVdata_Type

Types of (scalar) data that can be written out of APBS.

Author

Nathan Baker

Enumerator:

- VDT_CHARGE** Charge distribution (e)
- VDT_POT** Potential (kT/e)
- VDT_ATOMPOT** Atom potential (kT/e)
- VDT_SMOL** Solvent accessibility defined by molecular/Connolly surface definition (1 = accessible, 0 = inaccessible)
- VDT_SSPL** Spline-based solvent accessibility (1 = accessible, 0 = inaccessible)
- VDT_VDW** van der Waals-based accessibility (1 = accessible, 0 = inaccessible)
- VDT_IVDW** Ion accessibility/inflated van der Waals (1 = accessible, 0 = inaccessible)
- VDT_LAP** Laplacian of potential (kT/e/A²)
- VDT_EDEN** Energy density $\epsilon(\nabla u)^2$, where u is potential (kT/e/A)²
- VDT_NDENS** Ion number density $\sum c_i \exp(-q_i u)^2$, where u is potential (output in M)
- VDT_QDENS** Ion charge density $\sum q_i c_i \exp(-q_i u)^2$, where u is potential (output in $e_c M$)
- VDT_DIELX** Dielectric x-shifted map as calculated with the currently specified scheme (dimensionless)
- VDT_DIELY** Dielectric y-shifted map as calculated with the currently specified scheme (dimensionless)
- VDT_DIELZ** Dielectric z-shifted map as calculated with the currently specified scheme (dimensionless)
- VDT_KAPPA** Kappa map as calculated with the currently specified scheme ($^{-3}$)

Definition at line 268 of file [vhal.h](#).

7.15.3.6 enum eVhal_IPKEYType

Type of ipkey to use for MG methods.

Enumerator:

IPKEY_SMPBE SMPBE ipkey
IPKEY_LPBE LPBE ipkey
IPKEY_NPBE NPBE ipkey

Definition at line 156 of file [vhal.h](#).

7.15.3.7 enum eVhal_PBEType

Version of PBE to solve.

Enumerator:

PBE_LPBE Traditional Poisson-Boltzmann equation, linearized
PBE_NPBE Traditional Poisson-Boltzmann equation, full
PBE_LRPBE Regularized Poisson-Boltzmann equation, linearized
PBE_SMPBE < Regularized Poisson-Boltzmann equation, full SM PBE

Definition at line 138 of file [vhal.h](#).

7.15.3.8 enum eVoutput_Format

Output file format.

Enumerator:

OUTPUT_NULL No output
OUTPUT_FLAT Output in flat-file format

Definition at line 190 of file [vhal.h](#).

7.15.3.9 enum eVrc_Codes

Return code enumerations.

Author

David Gohara

Note

Note that the enumerated values are opposite the standard for FAILURE and SUCCESS

Enumerator:

VRC_FAILURE A non-fatal error
VRC_SUCCESS A fatal error

Definition at line 65 of file [vhal.h](#).

7.15.3.10 enum eVsol_Meth

Solution Method enumerations.

Author

David Gohara

Note

Note that the enumerated values are opposite the standard for FAILURE and SUCCESS

Definition at line 80 of file [vhal.h](#).

7.15.3.11 enum eVsurf_Meth

Types of molecular surface definitions.

Author

Nathan Baker

Enumerator:

VSM_MOL Ion accessibility is defined using inflated van der Waals radii, the dielectric coefficient () is defined using the molecular (Conolly) surface definition without smoothing

VSM_MOLSMOOTH As VSM_MOL but with a simple harmonic average smoothing

VSM_SPLINE Spline-based surface definitions. This is primarily for use with force calculations, since it requires substantial reparameterization of radii. This is based on the work of Im et al, Comp. Phys. Comm. 111 , (1998) and uses a cubic spline to define a smoothly varying characteristic function for the surface-based parameters. Ion accessibility is defined using inflated van der Waals radii with the spline function and the dielectric coefficient is defined using the standard van der Waals radii with the spline function.

VSM_SPLINE3 A 5th order polynomial spline is used to create a smoothly varying characteristic function (continuity through 2nd derivatives) for surface based paramters.

VSM_SPLINE4 A 7th order polynomial spline is used to create a smoothly varying characteristic function (continuity through 3rd derivatives) for surface based paramters.

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

7.16 Matrix wrapper class

A header for including data wrapping matrices.

Files

- file [vmatrix.h](#)

Contains inclusions for matrix data wrappers.

7.16.1 Detailed Description

A header for including data wrapping matrices.

7.17 Vparam class

Reads and assigns charge/radii parameters.

Files

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

Data Structures

- struct [sVparam_AtomData](#)
AtomData sub-class; stores atom data.
- struct [Vparam_ResData](#)
ResData sub-class; stores residue data.
- struct [Vparam](#)
Reads and assigns charge/radii parameters.

Typedefs

- typedef struct [sVparam_AtomData](#) [Vparam_AtomData](#)
Declaration of the [Vparam_AtomData](#) class as the [sVparam_AtomData](#) structure.
- typedef struct [Vparam_ResData](#) [Vparam_ResData](#)
Declaration of the [Vparam_ResData](#) class as the [Vparam_ResData](#) structure.
- typedef struct [Vparam](#) [Vparam](#)
Declaration of the [Vparam](#) class as the [Vparam](#) structure.

Functions

- VEXTERNC unsigned long int [Vparam_memChk](#) ([Vparam](#) *thee)
Get number of bytes in this object and its members.
- VEXTERNC [Vparam_AtomData](#) * [Vparam_AtomData_ctor](#) ()
Construct the object.
- VEXTERNC int [Vparam_AtomData_ctor2](#) ([Vparam_AtomData](#) *thee)
FORTRAN stub to construct the object.
- VEXTERNC void [Vparam_AtomData_dtor](#) ([Vparam_AtomData](#) **thee)
Destroy object.
- VEXTERNC void [Vparam_AtomData_dtor2](#) ([Vparam_AtomData](#) *thee)
FORTRAN stub to destroy object.
- VEXTERNC void [Vparam_AtomData_copyTo](#) ([Vparam_AtomData](#) *thee, [Vparam_AtomData](#) *dest)
Copy current atom object to destination.
- VEXTERNC void [Vparam_ResData_copyTo](#) ([Vparam_ResData](#) *thee, [Vparam_ResData](#) *dest)
Copy current residue object to destination.
- VEXTERNC void [Vparam_AtomData_copyFrom](#) ([Vparam_AtomData](#) *thee, [Vparam_AtomData](#) *src)

- Copy current atom object from another.
- VEXTERNC `Vparam_ResData * Vparam_ResData_ctor (Vmem *mem)`

Construct the object.
- VEXTERNC int `Vparam_ResData_ctor2 (Vparam_ResData *thee, Vmem *mem)`

FORTRAN stub to construct the object.
- VEXTERNC void `Vparam_ResData_dtor (Vparam_ResData **thee)`

Destroy object.
- VEXTERNC void `Vparam_ResData_dtor2 (Vparam_ResData *thee)`

FORTRAN stub to destroy object.
- VEXTERNC `Vparam * Vparam_ctor ()`

Construct the object.
- VEXTERNC int `Vparam_ctor2 (Vparam *thee)`

FORTRAN stub to construct the object.
- VEXTERNC void `Vparam_dtor (Vparam **thee)`

Destroy object.
- VEXTERNC void `Vparam_dtor2 (Vparam *thee)`

FORTRAN stub to destroy object.
- VEXTERNC `Vparam_ResData * Vparam_getResData (Vparam *thee, char resName[VMAX_ARGLEN])`

Get residue data.
- VEXTERNC `Vparam_AtomData * Vparam_getAtomData (Vparam *thee, char resName[VMAX_ARGLEN], char atomName[VMAX_ARGLEN])`

Get atom data.
- VEXTERNC int `Vparam_readFlatFile (Vparam *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname)`

Read a flat-file format parameter database.
- VEXTERNC int `Vparam_readXMLFile (Vparam *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname)`

Read an XML format parameter database.
- VPRIATE int `readFlatFileLine (Vio *sock, Vparam_AtomData *atom)`

Read a single line of the flat file database.
- VPRIATE int `readXMLFileAtom (Vio *sock, Vparam_AtomData *atom)`

Read atom information from an XML file.

Variables

- VPRIATE char * `MCwhiteChars = " =,\t\n\r"`

Whitespace characters for socket reads.
- VPRIATE char * `MCcommChars = "#%"`

Comment characters for socket reads.
- VPRIATE char * `MCxmlwhiteChars = " =,\t\n\r<>"`

Whitespace characters for XML socket reads.

7.17.1 Detailed Description

Reads and assigns charge/radii parameters.

7.17.2 Function Documentation

7.17.2.1 VPRIVATE int readFlatFileLine (*Vio * sock*, *Vparam_AtomData * atom*)

Read a single line of the flat file database.

Author

Nathan Baker

Parameters

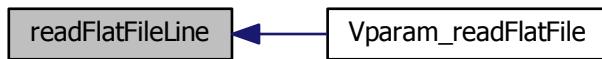
| | |
|-------------|--------------------------|
| <i>sock</i> | Socket ready for reading |
| <i>atom</i> | Atom to hold parsed data |

Returns

1 if successful, 0 otherwise

Definition at line 696 of file [vparam.c](#).

Here is the caller graph for this function:



7.17.2.2 VPRIVATE int readXMLFileAtom (*Vio * sock*, *Vparam_AtomData * atom*)

Read atom information from an XML file.

Author

Todd Dolinsky

Parameters

| | |
|-------------|--------------------------|
| <i>sock</i> | Socket ready for reading |
| <i>atom</i> | Atom to hold parsed data |

Returns

1 if successful, 0 otherwise

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.17.2.3 VEXTERNC void Vparam_AtomData_copyFrom (Vparam_AtomData * *thee*, Vparam_AtomData * *src*)

Copy current atom object from another.

Author

Nathan Baker

Parameters

| | |
|-------------|-------------------------------|
| <i>thee</i> | Pointer to destination object |
| <i>src</i> | Pointer to source object |

Definition at line 612 of file [vparam.c](#).

Here is the call graph for this function:



7.17.2.4 VEXTERNC void Vparam_AtomData_copyTo (Vparam_AtomData * *thee*, Vparam_AtomData * *dest*)

Copy current atom object to destination.

Author

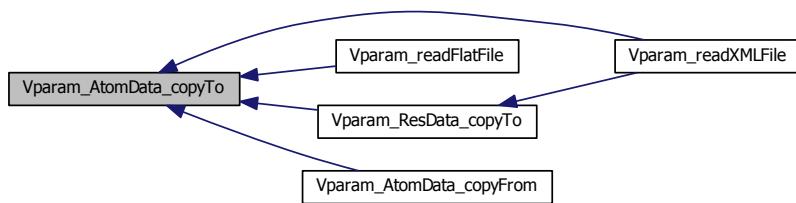
Nathan Baker

Parameters

| | |
|-------------|-------------------------------|
| <i>thee</i> | Pointer to source object |
| <i>dest</i> | Pointer to destination object |

Definition at line 576 of file [vparam.c](#).

Here is the caller graph for this function:



7.17.2.5 VEXTERNC Vparam_AtomData* Vparam_AtomData_ctor ()

Construct the object.

Author

Nathan Baker

Returns

Newly allocated object

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

Here is the call graph for this function:



7.17.2.6 VEXTERNC int Vparam_AtomData_ctor2 (Vparam_AtomData * *thee*)

FORTRAN stub to construct the object.

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 126 of file [vparam.c](#).

Here is the caller graph for this function:



7.17.2.7 VEXTERNC void Vparam_AtomData_dtor (Vparam_AtomData ** *thee*)

Destroy object.

Author

Nathan Baker

Parameters

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

Definition at line 128 of file [vparam.c](#).

Here is the call graph for this function:



7.17.2.8 VEXTERNC void Vparam_AtomData_dtor2 (Vparam_AtomData * *thee*)

FORTRAN stub to destroy object.

Author

Nathan Baker

Parameters

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

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

Here is the caller graph for this function:



7.17.2.9 VEXTERNC Vparam* Vparam_ctor ()

Construct the object.

Author

Nathan Baker

Returns

Newly allocated [Vparam](#) object

Definition at line 186 of file [vparam.c](#).

Here is the call graph for this function:



7.17.2.10 VEXTERNC int Vparam_ctor2 (Vparam * *thee*)

FORTRAN stub to construct the object.

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

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

Here is the caller graph for this function:



7.17.2.11 VEXTERNC void Vparam_dtor (Vparam ** *thee*)

Destroy object.

Author

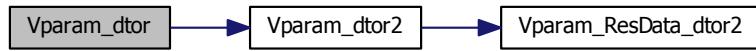
Nathan Baker

Parameters

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

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

Here is the call graph for this function:

**7.17.2.12 VEXTERNC void Vparam_dtor2(Vparam * *thee*)**

FORTRAN stub to destroy object.

Author

Nathan Baker

Parameters

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.17.2.13 VEXTERNC Vparam_AtomData* Vparam_getAtomData (Vparam * *thee*, char *resName*[VMAX_ARGLEN], char *atomName*[VMAX_ARGLEN])

Get atom data.

Author

Nathan Baker

Parameters

| | |
|-----------------|---------------|
| <i>thee</i> | Vparam object |
| <i>resName</i> | Residue name |
| <i>atomName</i> | Atom name |

Returns

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

Note

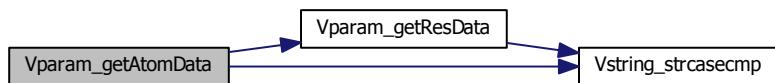
Some method to initialize the database must be called before this method (e.g.,

See Also

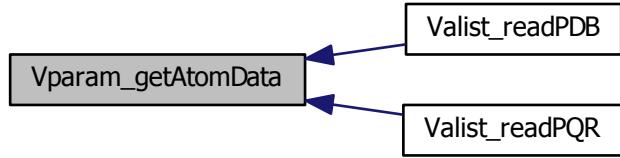
[Vparam_readFlatFile\(\)](#)

Definition at line 272 of file [vparam.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.17.2.14 VEXTERNC Vparam_ResData* Vparam_getResData (Vparam * thee, char resName[VMAX_ARGLEN])

Get residue data.

Author

Nathan Baker

Parameters

| | |
|----------------|---------------|
| <i>thee</i> | Vparam object |
| <i>resName</i> | Residue name |

Returns

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

Note

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

See Also

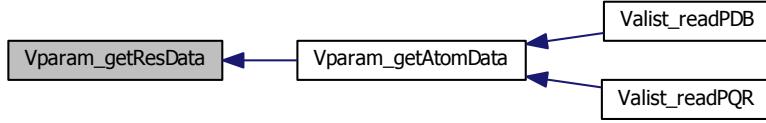
[Vparam_readFlatFile](#))

Definition at line 246 of file [vparam.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.17.2.15 VEXTERNC unsigned long int Vparam_memChk (Vparam * *thee*)

Get number of bytes in this object and its members.

Author

Nathan Baker

Parameters

| | |
|-------------|---------------|
| <i>thee</i> | Vparam object |
|-------------|---------------|

Returns

Number of bytes allocated for object

Definition at line 107 of file [vparam.c](#).

7.17.2.16 VEXTERNC int Vparam_readFlatFile (Vparam * *thee*, const char * *iodev*, const char * *iofmt*, const char * *thost*, const char * *fname*)

Read a flat-file format parameter database.

Author

Nathan Baker

Parameters

| | |
|--------------|--|
| <i>thee</i> | Vparam object |
| <i>iodev</i> | Input device type (FILE/BUFF/UNIX/INET) |
| <i>iofmt</i> | Input device format (ASCII/XDR) |
| <i>thost</i> | Input hostname (for sockets) |
| <i>fname</i> | Input FILE/BUFF/UNIX/INET name (see note below for format) |

Returns

1 if successful, 0 otherwise

Note

The database file should have the following format:

```
RESIDUE ATOM CHARGE RADIUS EPSILON
```

where RESIDUE is the residue name string, ATOM is the atom name string, CHARGE is the charge in e, RADIUS is the van der Waals radius (σ_i) in Å, and EPSILON is the van der Waals well-depth (ε_i) in kJ/mol. See the [Vparam](#) structure documentation for the precise definitions of σ_i and ε_i .

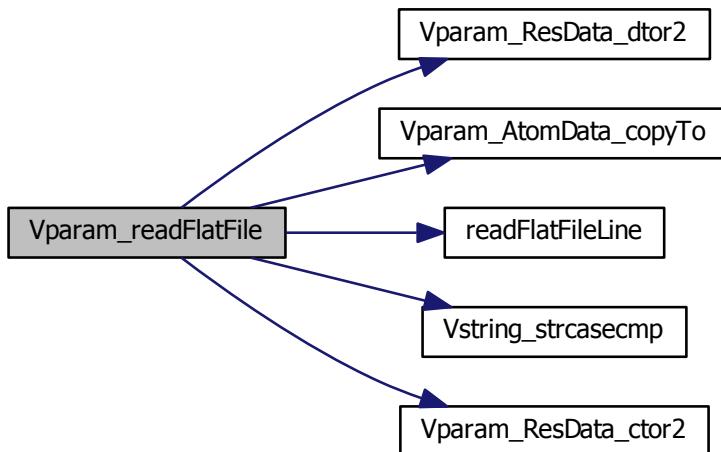
ASCII-format flat files are provided with the APBS source code:

tools/conversion/vparam-amber-parmp94.dat AMBER parmp94 parameters

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

Definition at line 450 of file [vparam.c](#).

Here is the call graph for this function:



7.17.2.17 VEXTERNC int Vparam_readXMLFile (Vparam * thee, const char * iodev, const char * ifofmt, const char * thost, const char * fname)

Read an XML format parameter database.

Author

Todd Dolinsky

Parameters

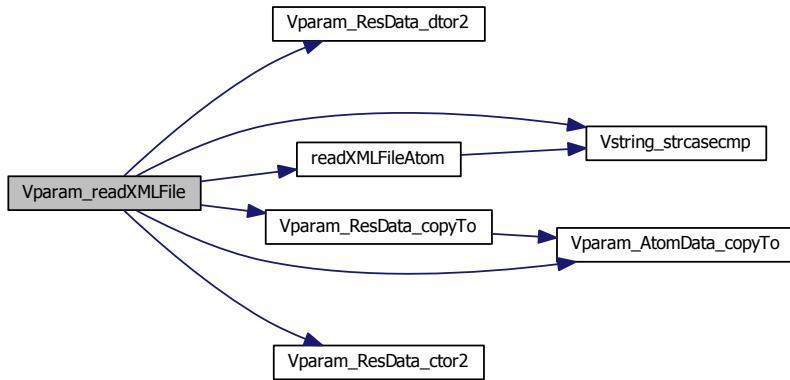
| | |
|--------------|---|
| <i>thee</i> | Vparam object |
| <i>iodev</i> | Input device type (FILE/BUFF/UNIX/INET) |
| <i>iofmt</i> | Input device format (ASCII/XDR) |
| <i>thost</i> | Input hostname (for sockets) |
| <i>fname</i> | Input FILE/BUFF/UNIX/INET name |

Returns

1 if successful, 0 otherwise

Definition at line 311 of file [vparam.c](#).

Here is the call graph for this function:



7.17.2.18 VEXTERNC void Vparam_ResData_copyTo (Vparam_ResData * *thee*, Vparam_ResData * *dest*)

Copy current residue object to destination.

Author

Todd Dolinsky

Parameters

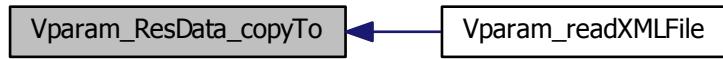
| | |
|-------------|-------------------------------|
| <i>thee</i> | Pointer to source object |
| <i>dest</i> | Pointer to destination object |

Definition at line 590 of file [vparam.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.17.2.19 VEXTERNC Vparam_ResData* Vparam_ResData_ctor (Vmem * mem)

Construct the object.

Author

Nathan Baker

Parameters

| | |
|-----|--|
| mem | Memory object of Vparam master class |
|-----|--|

Returns

Newly allocated object

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

Here is the call graph for this function:



7.17.2.20 VEXTERNC int Vparam_ResData_ctor2(Vparam_ResData *thee, Vmem *mem)

FORTRAN stub to construct the object.

Author

Nathan Baker

Parameters

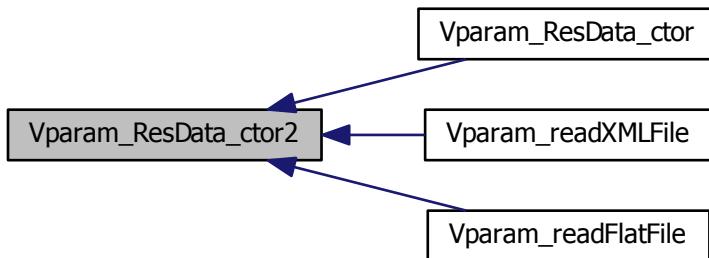
| | |
|-------------|--|
| <i>thee</i> | Allocated memory |
| <i>mem</i> | Memory object of Vparam master class |

Returns

1 if successful, 0 otherwise

Definition at line 152 of file [vparam.c](#).

Here is the caller graph for this function:



7.17.2.21 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 165 of file [vparam.c](#).

Here is the call graph for this function:



7.17.2.22 VEXTERNC void Vparam_ResData_dtor2 (Vparam_ResData * *thee*)

FORTRAN stub to destroy object.

Author

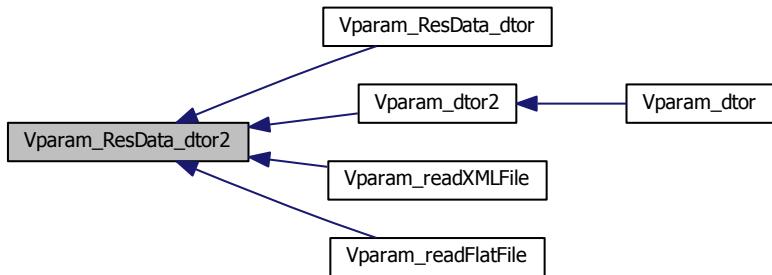
Nathan Baker

Parameters

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

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

Here is the caller graph for this function:



7.18 Vpbe class

The Poisson-Boltzmann master class.

Files

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

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)

- **VEXTERNC double `Vpbe_getSoluteCharge` (`Vpbe *thee`)**
Get total solute charge.
- **VEXTERNC double `Vpbe_getSolventDiel` (`Vpbe *thee`)**
Get solvent dielectric constant.
- **VEXTERNC double `Vpbe_getSolventRadius` (`Vpbe *thee`)**
Get solvent molecule radius.
- **VEXTERNC double `Vpbe_getXkappa` (`Vpbe *thee`)**
Get Debye-Huckel parameter.
- **VEXTERNC double `Vpbe_getDeblen` (`Vpbe *thee`)**
Get Debye-Huckel screening length.
- **VEXTERNC double `Vpbe_getZkappa2` (`Vpbe *thee`)**
Get modified squared Debye-Huckel parameter.
- **VEXTERNC double `Vpbe_getZmagic` (`Vpbe *thee`)**
Get charge scaling factor.
- **VEXTERNC double `Vpbe_getzmem` (`Vpbe *thee`)**
Get z position of the membrane bottom.
- **VEXTERNC double `Vpbe_getLmem` (`Vpbe *thee`)**
*Get length of the membrane (A)
aauthor Michael Grabe.*
- **VEXTERNC double `Vpbe_getmembraneDiel` (`Vpbe *thee`)**
Get membrane dielectric constant.
- **VEXTERNC double `Vpbe_getmemv` (`Vpbe *thee`)**
Get membrane potential (kT)
- **VEXTERNC `Vpbe *` `Vpbe_ctor` (`Valist *alist`, int `ionNum`, double `*ionConc`, double `*ionRadii`, double `*ionQ`, double `T`, double `soluteDiel`, double `solventDiel`, double `solventRadius`, int `focusFlag`, double `sdens`, double `z_mem`, double `L`, double `membraneDiel`, double `V`)**
Construct Vpbe object.
- **VEXTERNC int `Vpbe_ctor2` (`Vpbe *thee`, `Valist *alist`, int `ionNum`, double `*ionConc`, double `*ionRadii`, double `*ionQ`, double `T`, double `soluteDiel`, double `solventDiel`, double `solventRadius`, int `focusFlag`, double `sdens`, double `z_mem`, double `L`, double `membraneDiel`, double `V`)**
FORTRAN stub to construct Vpbe objct.
- **VEXTERNC int `Vpbe_getIons` (`Vpbe *thee`, int `*nion`, double `ionConc[MAXION]`, double `ionRadii[MAXION]`, double `ionQ[MAXION]`)**
Get information about the counterion species present.
- **VEXTERNC void `Vpbe_dtor` (`Vpbe **thee`)**
Object destructor.
- **VEXTERNC void `Vpbe_dtor2` (`Vpbe *thee`)**
FORTRAN stub object destructor.
- **VEXTERNC double `Vpbe_getCoulombEnergy1` (`Vpbe *thee`)**
Calculate coulombic energy of set of charges.
- **VEXTERNC unsigned long int `Vpbe_memChk` (`Vpbe *thee`)**
Return the memory used by this structure (and its contents) in bytes.

7.18.1 Detailed Description

The Poisson-Boltzmann master class. Contains objects and parameters used in every PBE calculation, regardless of method.

7.18.2 Function Documentation

7.18.2.1 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 m^{-2} , we multiply by 10^{-16} . Thus, in m^{-2} , where k_B and e_c are in gaussian rather than mks units, the proper value for kappa is:

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

and the factor of 10^{-16} results from converting cm^2 to angstroms 2 , noting that the 1000 in the denominator has converted m^3 to cm^3 , since the ionic strength I_s is assumed to have been provided in moles per liter, which is moles per 1000 cm^3 .

Returns

Pointer to newly allocated Vpbe object

Parameters

| | |
|----------------------|---|
| <i>alist</i> | Atom list |
| <i>ionNum</i> | Number of counterion species |
| <i>ionConc</i> | Array containing counterion concentrations (M) |
| <i>ionRadii</i> | Array containing counterion radii (A) |
| <i>ionQ</i> | Array containing counterion charges (e) |
| <i>T</i> | Temperature for Boltzmann distribution (K) |
| <i>soluteDiel</i> | Solute internal dielectric constant |
| <i>solventDiel</i> | Solvent dielectric constant |
| <i>solventRadius</i> | Solvent probe radius for surfaces that use it (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 247 of file [vpbe.c](#).

7.18.2.2 VEXTERNC int Vpbe_ctor2 (*Vpbe * thee, Valist * alist, int ionNum, double * ionConc, double * ionRadii, double * ionQ, double T, double soluteDiel, double solventDiel, double solventRadius, int focusFlag, double sdens, double z_mem, double L, double membraneDiel, double V*)

FORTRAN stub to construct Vpbe 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:

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

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

$$\kappa^2 = \frac{8\pi N_A e_c^2 I_s}{1000 \epsilon \rho s_w k_B T} \times 10^{-16}$$

and the factor of 10^{-16} results from converting cm^2 to angstroms^2 , noting that the 1000 in the denominator has converted m^3 to cm^3 , since the ionic strength I_s is assumed to have been provided in moles per liter, which is moles per 1000 cm^3 .

Bug The focusing flag is currently not used!!

Returns

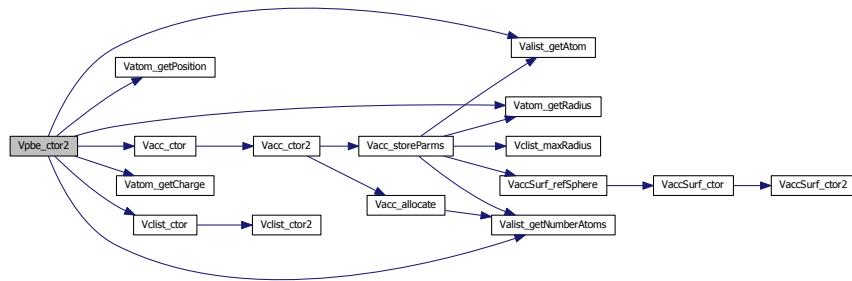
1 if successful, 0 otherwise

Parameters

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

Definition at line 265 of file [vpbe.c](#).

Here is the call graph for this function:



7.18.2.3 VEXTERNC void Vpbe_dtor (*Vpbe* ** *thee*)

Object destructor.

Author

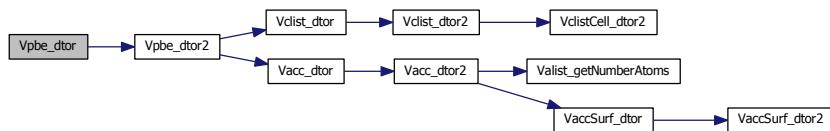
Nathan Baker

Parameters

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

Definition at line 468 of file [vpbe.c](#).

Here is the call graph for this function:



7.18.2.4 VEXTERNC void Vpbe_dtor2 (*Vpbe* * *thee*)

FORTRAN stub object destructor.

Author

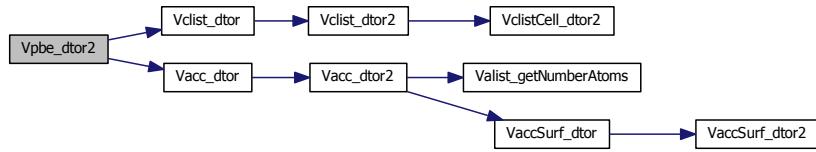
Nathan Baker

Parameters

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

Definition at line 476 of file [vpbe.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.18.2.5 VEXTERNC double Vpbe_getBulkIonicStrength (Vpbe * *thee*)

Get bulk ionic strength.

Author

Nathan Baker

Parameters

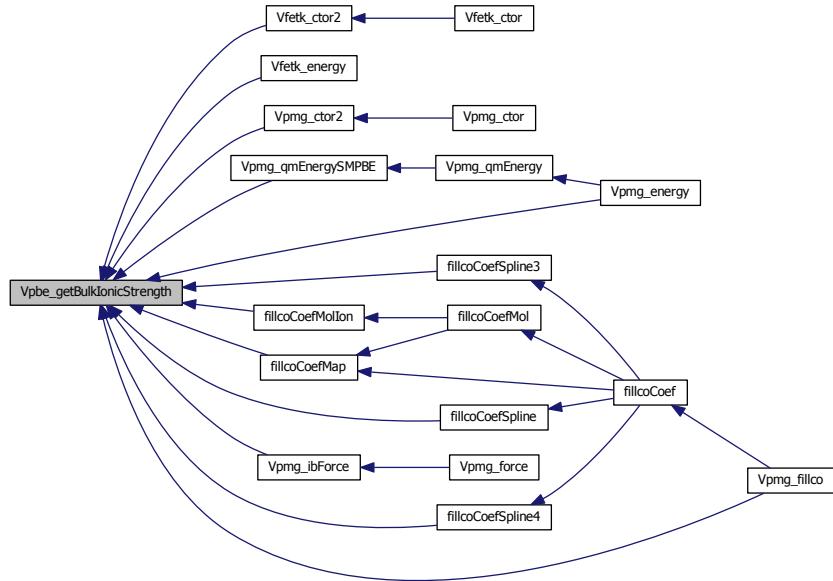
| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

Bulk ionic strength (M)

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

Here is the caller graph for this function:



7.18.2.6 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

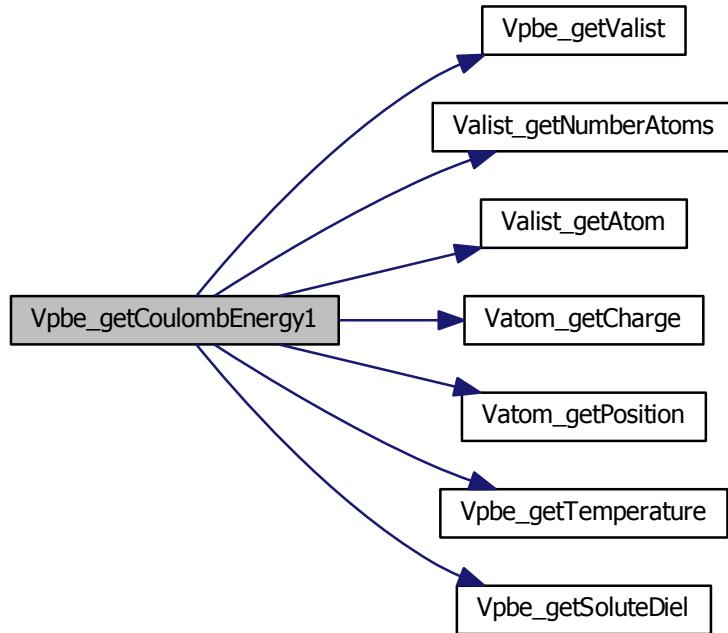
| | |
|-------------------|-------------|
| <code>thee</code> | Vpbe object |
|-------------------|-------------|

Returns

Coulombic energy in units of $k_B T$.

Definition at line 482 of file [vpbe.c](#).

Here is the call graph for this function:

**7.18.2.7 VEXTERNC double Vpbe_getDeblen (Vpbe * *thee*)**

Get Debye-Hückel screening length.

Author

Nathan Baker

Parameters

| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

Debye-Hückel screening length (Å)

Definition at line 142 of file [vpbe.c](#).

7.18.2.8 VEXTERNC double Vpbe_getGamma (*Vpbe * thee*)

Get apolar coefficient.

Author

Nathan Baker

Parameters

| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

Apolar coefficient (kJ/mol/A²)

7.18.2.9 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

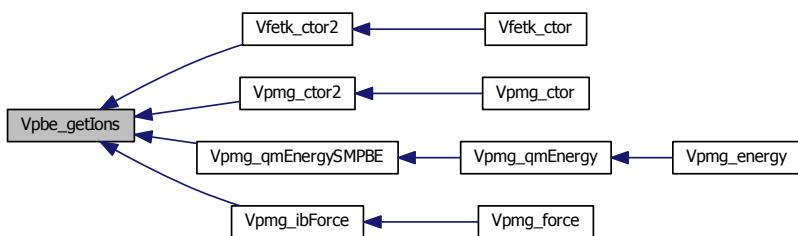
| | |
|-----------------|---|
| <i>thee</i> | Pointer to Vpbe object |
| <i>nion</i> | Set to the number of counterion species |
| <i>ionConc</i> | Array to store counterion species' concentrations (M) |
| <i>ionRadii</i> | Array to store counterion species' radii (A) |
| <i>ionQ</i> | Array to store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

Returns

Number of ions

Definition at line 536 of file [vpbe.c](#).

Here is the caller graph for this function:



7.18.2.10 VEXTERNC double Vpbe_getLmem (Vpbe * *thee*)

Get length of the membrane (A)

Author Michael Grabe.

Parameters

| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

Length of the membrane (A)

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

Here is the caller graph for this function:

**7.18.2.11 VEXTERNC double Vpbe_getMaxIonRadius (Vpbe * *thee*)**

Get maximum radius of ion species.

Author

Nathan Baker

Parameters

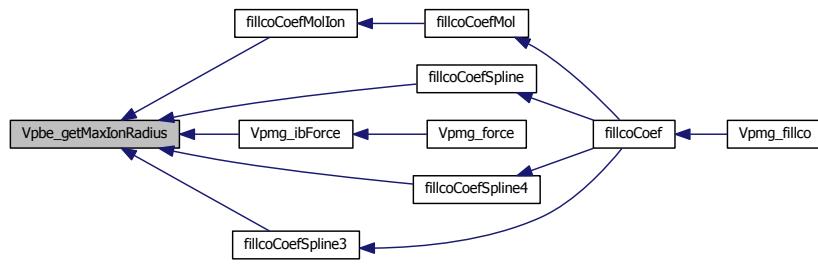
| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

Maximum radius (A)

Definition at line 128 of file [vpbe.c](#).

Here is the caller graph for this function:



7.18.2.12 VEXTERNC double Vpbe_getmembraneDiel (**Vpbe** * *thee*)

Get membrane dielectric constant.

Author

Michael Grabe

Parameters

| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

Membrane dielectric constant

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

Here is the caller graph for this function:



7.18.2.13 VEXTERNC double Vpbe_getmemv (Vpbe * *thee*)

Get membrane potential (kT)

Author

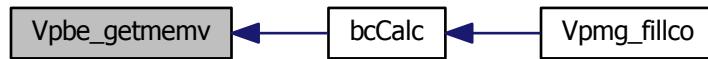
Michael Grabe

Parameters

| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

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

Here is the caller graph for this function:

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

Get coordinates of solute center.

Author

Nathan Baker

Parameters

| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

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

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

7.18.2.15 VEXTERNC double Vpbe_getSoluteCharge (Vpbe * *thee*)

Get total solute charge.

Author

Nathan Baker

Parameters

| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

Total solute charge (e)

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

7.18.2.16 VEXTERNC double Vpbe_getSoluteDiel (Vpbe * *thee*)

Get solute dielectric constant.

Author

Nathan Baker

Parameters

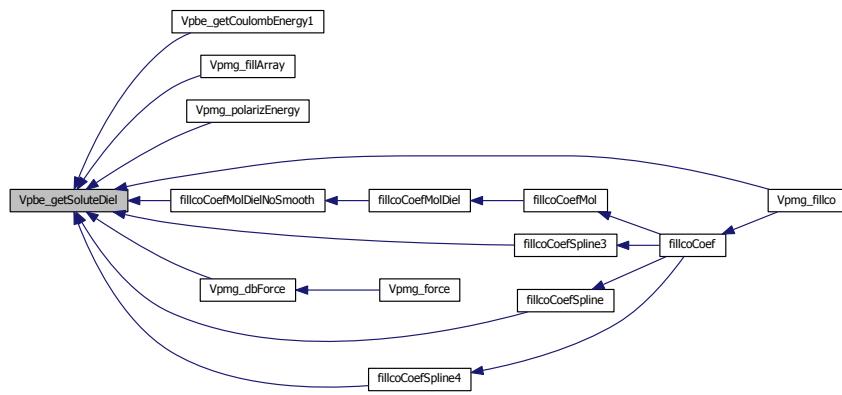
| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

Solute dielectric constant

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

Here is the caller graph for this function:

**7.18.2.17 VEXTERNC double Vpbe_getSoluteRadius (Vpbe * *thee*)**

Get sphere radius which bounds biomolecule.

Author

Nathan Baker

Parameters

| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

Sphere radius which bounds biomolecule (A)

Definition at line 163 of file [vpbe.c](#).

7.18.2.18 VEXTERNC double Vpbe_getSoluteXlen (Vpbe * *thee*)

Get length of solute in x dimension.

Author

Nathan Baker

Parameters

| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

Length of solute in x dimension (A)

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

7.18.2.19 VEXTERNC double Vpbe_getSoluteYlen (Vpbe * *thee*)

Get length of solute in y dimension.

Author

Nathan Baker

Parameters

| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

Length of solute in y dimension (A)

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

7.18.2.20 VEXTERNC double Vpbe_getSoluteZlen (*Vpbe * thee*)

Get length of solute in z dimension.

Author

Nathan Baker

Parameters

| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

Length of solute in z dimension (A)

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

7.18.2.21 VEXTERNC double Vpbe_getSolventDiel (*Vpbe * thee*)

Get solvent dielectric constant.

Author

Nathan Baker

Parameters

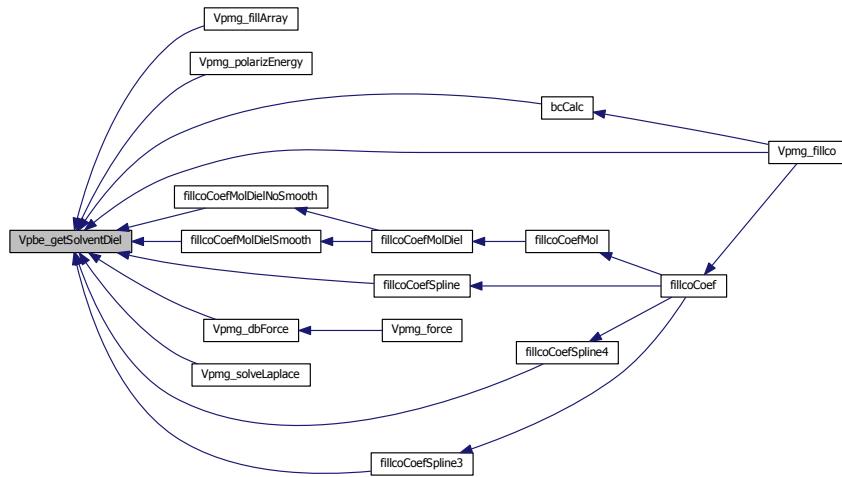
| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

Solvent dielectric constant

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

Here is the caller graph for this function:



7.18.2.22 VEXTERNC double Vpbe_getSolventRadius (`Vpbe * thee`)

Get solvent molecule radius.

Author

Nathan Baker

Parameters

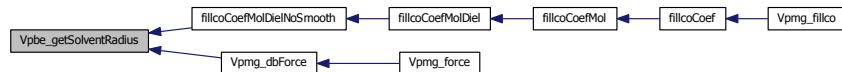
| | |
|-------------------|--------------------------|
| <code>thee</code> | <code>Vpbe</code> object |
|-------------------|--------------------------|

Returns

Solvent molecule radius (A)

Definition at line 121 of file [vpbe.c](#).

Here is the caller graph for this function:



7.18.2.23 VEXTERNC double Vpbe_getTemperature (`Vpbe * thee`)

Get temperature.

Author

Nathan Baker

Parameters

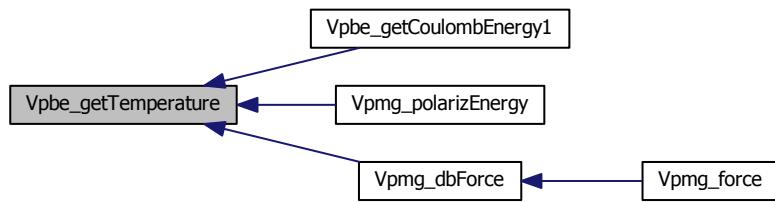
| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

Temperature (K)

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

Here is the caller graph for this function:



7.18.2.24 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 [77](#) of file [vpbe.c](#).

Here is the caller graph for this function:

**7.18.2.25 VEXTERNC Valist* Vpbe_getValist (Vpbe * *thee*)**

Get atom list.

Author

Nathan Baker

Parameters

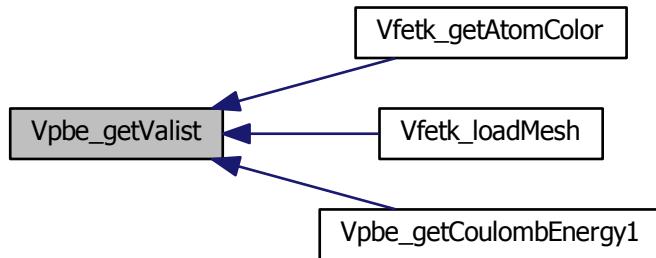
| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

Pointer to internal Valist object

Definition at line [70](#) of file [vpbe.c](#).

Here is the caller graph for this function:



7.18.2.26 VEXTERNC double Vpbe_getXkappa (Vpbe * *thee*)

Get Debye-Hückel parameter.

Author

Nathan Baker

Parameters

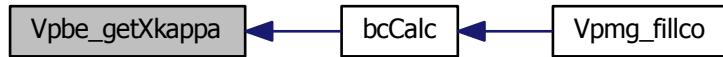
| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

Bulk Debye-Hückel parameter (\AA)

Definition at line 135 of file [vpbe.c](#).

Here is the caller graph for this function:

**7.18.2.27 VEXTERNC double Vpbe_getZkappa2 (Vpbe * *thee*)**

Get modified squared Debye-Hückel parameter.

Author

Nathan Baker

Parameters

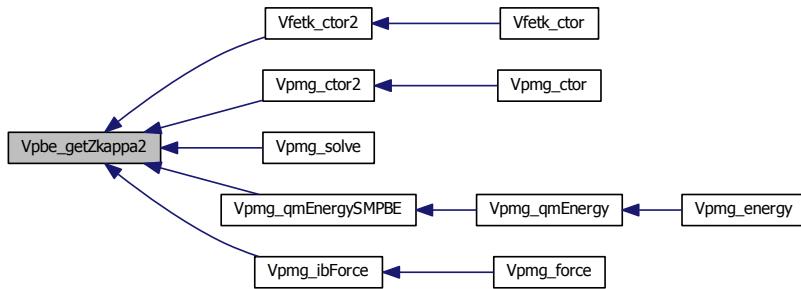
| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

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

Definition at line 149 of file [vpbe.c](#).

Here is the caller graph for this function:



7.18.2.28 VEXTERNC double Vpbe_getZmagic (Vpbe * *thee*)

Get charge scaling factor.

Author

Nathan Baker and Mike Holst

Parameters

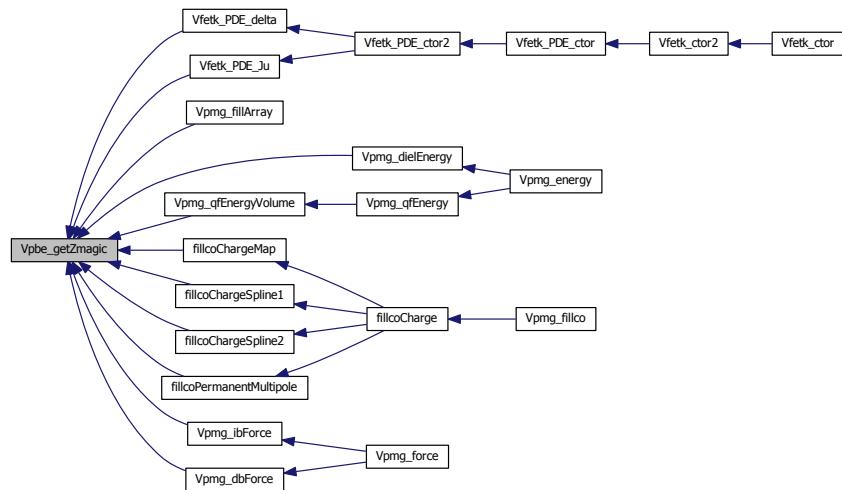
| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

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

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

Here is the caller graph for this function:



7.18.2.29 VEXTERNC double Vpbe_getzmem (`Vpbe * thee`)

Get z position of the membrane bottom.

Author

Michael Grabe

Parameters

| | |
|-------------------|--------------------------|
| <code>thee</code> | <code>Vpbe</code> object |
|-------------------|--------------------------|

Returns

`z` value of membrane (A)

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

Here is the caller graph for this function:



7.18.2.30 VEXTERNC unsigned long int Vpbe_memChk (Vpbe * *thee*)

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

Author

Nathan Baker

Parameters

| | |
|-------------|-------------|
| <i>thee</i> | Vpbe object |
|-------------|-------------|

Returns

The memory used by this structure and its contents in bytes

Definition at line [524](#) of file [vpbe.c](#).

Here is the call graph for this function:



7.19 Vstring class

Provides a collection of useful non-ANSI string functions.

Files

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

Functions

- VEXTERNC int [Vstring_strcasecmp](#) (const char *s1, const char *s2)
Case-insensitive string comparison (BSD standard)
- VEXTERNC int [Vstring_isdigit](#) (const char *tok)
A modified sscanf that examines the complete string.
- VEXTERNC char * [Vstring_wrappedtext](#) (const char *str, int right_margin, int left_padding)

7.19.1 Detailed Description

Provides a collection of useful non-ANSI string functions.

7.19.2 Function Documentation

7.19.2.1 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](#).

7.19.2.2 VEXTERNC int Vstring_strcasecmp (const char * s1, const char * s2)

Case-insensitive string comparison (BSD standard)

Author

Copyright (c) 1988-1993 The Regents of the University of California. Copyright (c) 1995-1996 Sun Microsystems, Inc.

Note

Copyright (c) 1988-1993 The Regents of the University of California. Copyright (c) 1995-1996 Sun Microsystems, Inc.

Parameters

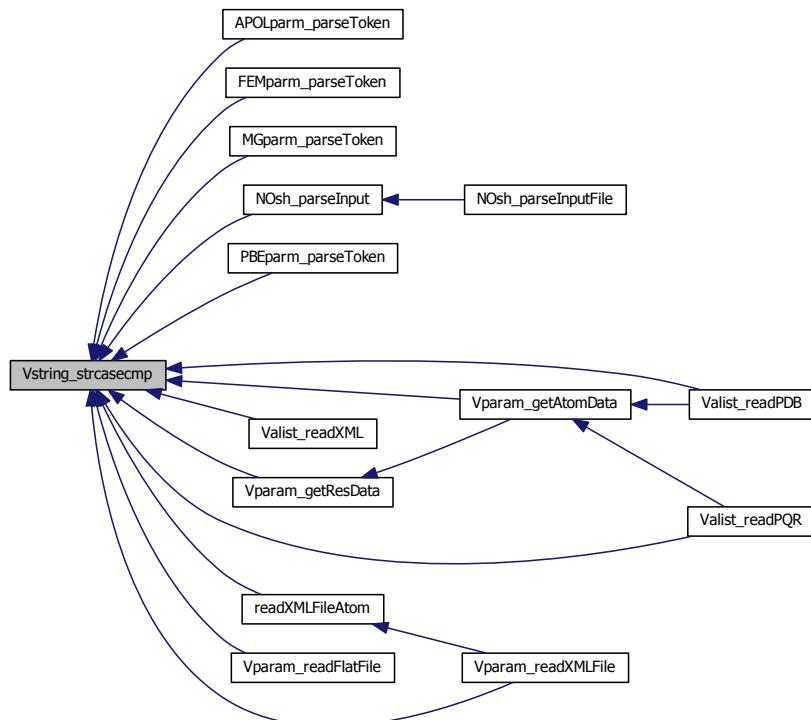
| | |
|-----------|------------------------------|
| <i>s1</i> | First string for comparison |
| <i>s2</i> | Second string for comparison |

Returns

An integer less than, equal to, or greater than zero if *s1* is found, respectively, to be less than, to match, or be greater than *s2*. (Source: Linux man pages)

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

Here is the caller graph for this function:



7.19.2.3 `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 funciton allocates a new string, so be sure to free it!

Note

: The +2 is for the newline character and the null-terminating character;

Parameters

| | |
|---------------------------|--|
| <code>str</code> | The input string to wrap and indent |
| <code>right_margin</code> | The number of characters to the right margin |
| <code>left_padding</code> | The number of characters in the left indent |

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

7.20 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`
 $e_c^2 / \text{in ESU units} \Rightarrow \text{kcal/mol}$
- `#define Vunit_esu_kb 1.9871913e-03`
 $k_b \text{ in ESU units} \Rightarrow \text{kcal/mol}$

7.20.1 Detailed Description

Collection of constants and conversion factors.

7.21 Vgrid class

Oracle for Cartesian mesh data.

Files

- file [vgrid.h](#)
Potential oracle for Cartesian mesh data.
- file [vgrid.c](#)
Class Vgrid methods.

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)
FORTRAN stub object destructor.
- VEXTERNC int [Vgrid_curvature](#) ([Vgrid](#) *thee, double pt[3], int cflag, double *curv)
Get second derivative values at a point.
- VEXTERNC int [Vgrid_gradient](#) ([Vgrid](#) *thee, double pt[3], double grad[3])
Get first derivative values at a point.

- VEXTERNC int [Vgrid_readGZ](#) ([Vgrid](#) *thee, const char *fname)
Read in OpenDX data in GZIP format.
- VEXTERNC void [Vgrid_writeUHBD](#) ([Vgrid](#) *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, char *title, double *pvec)
Write out the data in UHBD grid format.
- VEXTERNC void [Vgrid_writeDX](#) ([Vgrid](#) *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, char *title, double *pvec)
Write out the data in OpenDX grid format.
- VEXTERNC int [Vgrid_readDX](#) ([Vgrid](#) *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname)
Read in data in OpenDX grid format.
- VEXTERNC double [Vgrid_integrate](#) ([Vgrid](#) *thee)
Get the integral of the data.
- VEXTERNC double [Vgrid_normL1](#) ([Vgrid](#) *thee)
Get the L_1 norm of the data. This returns the integral:

$$\|u\|_{L_1} = \int_{\Omega} |u(x)| dx$$

- VEXTERNC double [Vgrid_normL2](#) ([Vgrid](#) *thee)

Get the L_2 norm of the data. This returns the integral:

$$\|u\|_{L_2} = \left(\int_{\Omega} |u(x)|^2 dx \right)^{1/2}$$

- VEXTERNC double [Vgrid_normLinf](#) ([Vgrid](#) *thee)

Get the L_∞ norm of the data. This returns the integral:

$$\|u\|_{L_\infty} = \sup_{x \in \Omega} |u(x)|$$

- VEXTERNC double [Vgrid_seminormH1](#) ([Vgrid](#) *thee)

Get the H_1 semi-norm of the data. This returns the integral:

$$\|u\|_{H_1} = \left(\int_{\Omega} |\nabla u(x)|^2 dx \right)^{1/2}$$

- VEXTERNC double [Vgrid_normH1](#) ([Vgrid](#) *thee)

Get the H_1 norm (or energy norm) of the data. This returns the integral:

$$\|u\|_{H_1} = \left(\int_{\Omega} |\nabla u(x)|^2 dx + \int_{\Omega} |u(x)|^2 dx \right)^{1/2}$$

7.21.1 Detailed Description

Oracle for Cartesian mesh data.

7.21.2 Function Documentation

7.21.2.1 VEXTERNC `Vgrid* Vgrid_ctor(int nx, int ny, int nz, double hx, double hy, double hzed, double xmin, double ymin, double zmin, double * data)`

Construct Vgrid object with values obtained from Vpmg_readDX (for example)

Author

Nathan Baker

Parameters

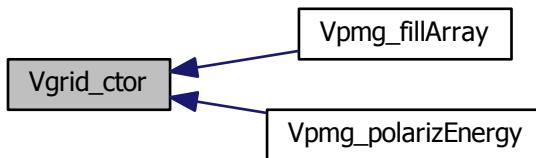
| | |
|-------------------|---|
| <code>nx</code> | Number grid points in x direction |
| <code>ny</code> | Number grid points in y direction |
| <code>nz</code> | Number grid points in z direction |
| <code>hx</code> | Grid spacing in x direction |
| <code>hy</code> | Grid spacing in y direction |
| <code>hzed</code> | Grid spacing in z direction |
| <code>xmin</code> | x coordinate of lower grid corner |
| <code>ymin</code> | y coordinate of lower grid corner |
| <code>zmin</code> | z coordinate of lower grid corner |
| <code>data</code> | 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 78 of file [vgrid.c](#).

Here is the caller graph for this function:



7.21.2.2 VEXTERNC `int Vgrid_ctor2(Vgrid * thee, int nx, int ny, int nz, double hx, double hy, double hzed, double xmin, double ymin, double zmin, double * data)`

Initialize Vgrid object with values obtained from Vpmg_readDX (for example)

Author

Nathan Baker

Parameters

| | |
|-------------|--|
| <i>thee</i> | Pointer to newly allocated Vgrid object |
| <i>nx</i> | Number grid points in x direction |
| <i>ny</i> | Number grid points in y direction |
| <i>nz</i> | Number grid points in z direction |
| <i>hx</i> | Grid spacing in x direction |
| <i>hy</i> | Grid spacing in y direction |
| <i>hzed</i> | Grid spacing in z direction |
| <i>xmin</i> | x coordinate of lower grid corner |
| <i>ymin</i> | y coordinate of lower grid corner |
| <i>zmin</i> | z coordinate of lower grid corner |
| <i>data</i> | <i>nx*ny*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 104 of file [vgrid.c](#).

7.21.2.3 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

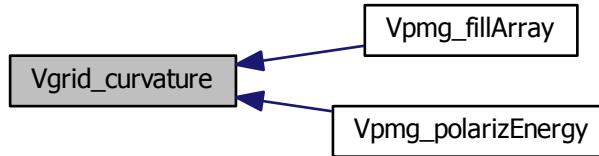
| | |
|--------------|--|
| <i>thee</i> | Pointer to Vgrid object |
| <i>pt</i> | Location to evaluate second derivative |
| <i>cflag</i> | <ul style="list-style-type: none"> • 0: Reduced Maximal Curvature • 1: Mean Curvature (Laplace) • 2: Gauss Curvature • 3: True Maximal Curvature |
| <i>curv</i> | Specified curvature value |

Returns

1 if successful, 0 if off grid

Definition at line 289 of file [vgrid.c](#).

Here is the caller graph for this function:



7.21.2.4 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 144 of file [vgrid.c](#).

Here is the caller graph for this function:



7.21.2.5 VEXTERNC void Vgrid_dtor2 (Vgrid * *thee*)

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 157 of file [vgrid.c](#).

7.21.2.6 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

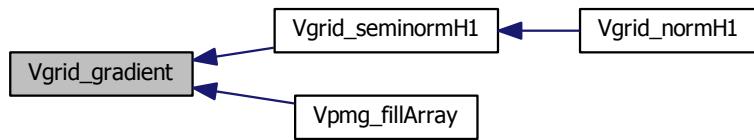
| | |
|-------------|-------------------------------|
| <i>thee</i> | Pointer to Vgrid object |
| <i>pt</i> | Location to evaluate gradient |
| <i>grad</i> | Gradient |

Returns

1 if successful, 0 if off grid

Definition at line 369 of file [vgrid.c](#).

Here is the caller graph for this function:



7.21.2.7 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 [1349](#) of file [vgrid.c](#).

7.21.2.8 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 [63](#) of file [vgrid.c](#).

7.21.2.9 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

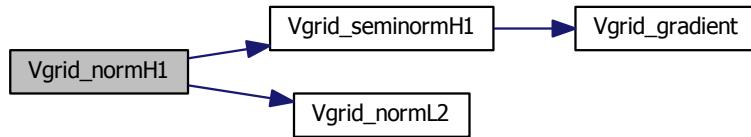
| | |
|-------------|--------------|
| <i>thee</i> | Vgrid object |
|-------------|--------------|

Returns

Integral of data

Definition at line 1486 of file [vgrid.c](#).

Here is the call graph for this function:

**7.21.2.10 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

| | |
|-------------------|--------------|
| <code>thee</code> | Vgrid object |
|-------------------|--------------|

Returns

L_1 norm of data

Definition at line 1386 of file [vgrid.c](#).

7.21.2.11 VEXTERNC double Vgrid_normL2 (Vgrid * thee)

Get the L_2 norm of the data. This returns the integral:

$$\|u\|_{L_2} = \left(\int_{\Omega} |u(x)|^2 dx \right)^{1/2}$$

Author

Nathan Baker

Parameters

| | |
|-------------|--------------|
| <i>thee</i> | Vgrid object |
|-------------|--------------|

Returns

L_2 norm of data

Definition at line 1415 of file [vgrid.c](#).

Here is the caller graph for this function:

**7.21.2.12 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 1501 of file [vgrid.c](#).

7.21.2.13 VEXTERNC int Vgrid_readDX (Vgrid * *thee*, const char * *iodev*, const char * *iofmt*, const char * *thost*, const char * *fname*)

Read in data in OpenDX grid format.

Note

All dimension information is given in order: z, y, x

Author

Nathan Baker

Parameters

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

Returns

1 if sucessful, 0 otherwise

Load grid from an input file using sockets.

Author

Nathan Baker

Definition at line [576](#) of file [vgrid.c](#).

7.21.2.14 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 [452](#) of file [vgrid.c](#).

7.21.2.15 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 1444 of file [vgrid.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.21.2.16 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

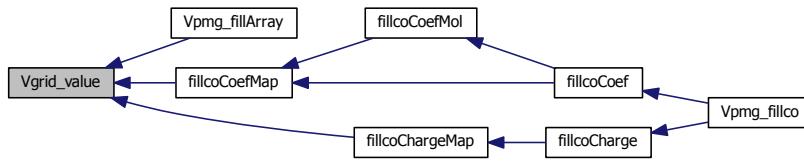
| | |
|--------------|--------------------------------------|
| <i>thee</i> | Vgrid obejct |
| <i>x</i> | Point at which to evaluate potential |
| <i>value</i> | Value of data at point x |

Returns

1 if successful, 0 if off grid

Definition at line 171 of file [vgrid.c](#).

Here is the caller graph for this function:



7.21.2.17 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>iodev</i> | Output device type (FILE/BUFF/UNIX/INET) |
| <i>iofmt</i> | Output device format (ASCII/XDR) |
| <i>thost</i> | Output hostname (for sockets) |
| <i>fname</i> | Output FILE/BUFF/UNIX/INET name |
| <i>title</i> | Title to be inserted in grid file |
| <i>pvec</i> | Partition weight (if 1: point in current partition, if 0 point not in current partition if > 0 && < 1 point on/near boundary) |

Definition at line 1002 of file [vgrid.c](#).

7.21.2.18 VEXTERNC void Vgrid_writeUHBD (*Vgrid * thee, const char * iodev, const char * io fmt, const char * thost, const char * fname, char * title, double * pvec*)

Write out the data in UHBD grid format.

Note

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

Author

Nathan Baker

Parameters

| | |
|--------------|---|
| <i>thee</i> | Grid object |
| <i>iodev</i> | Output device type (FILE/BUFF/UNIX/INET) |
| <i>iofmt</i> | Output device format (ASCII/XDR) |
| <i>thost</i> | Output hostname (for sockets) |
| <i>fname</i> | Output FILE/BUFF/UNIX/INET name |
| <i>title</i> | Title to be inserted in grid file |
| <i>pvec</i> | Partition weight (if 1: point in current partition, if 0 point not in current partition if > 0 && < 1 point on/near boundary) |

Bug This routine does not respect partition information

Definition at line 1252 of file [vgrid.c](#).

7.22 Vmgrid class

Oracle for Cartesian mesh data.

Files

- 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 Vgmrid structure.

Functions

- VEXTERNC [Vmgrid](#) * [Vmgrid_ctor](#) ()

Construct Vmgrid object.

- VEXTERNC int [Vmgrid_ctor2](#) ([Vmgrid](#) *thee)

Initialize Vmgrid object.

- VEXTERNC int [Vmgrid_value](#) ([Vmgrid](#) *thee, double x[3], double *value)

Get potential value (from mesh or approximation) at a point.

- VEXTERNC void [Vmgrid_dtor](#) ([Vmgrid](#) **thee)

Object destructor.

- VEXTERNC void [Vmgrid_dtor2](#) ([Vmgrid](#) *thee)

FORTRAN stub object destructor.

- VEXTERNC int [Vmgrid_addGrid](#) ([Vmgrid](#) *thee, [Vgrid](#) *grid)

Add a grid to the hierarchy.

- VEXTERNC int [Vmgrid_curvature](#) ([Vmgrid](#) *thee, double pt[3], int cflag, double *curv)

Get second derivative values at a point.

- VEXTERNC int [Vmgrid_gradient](#) ([Vmgrid](#) *thee, double pt[3], double grad[3])

Get first derivative values at a point.

- VEXTERNC [Vgrid](#) * [Vmgrid_getGridByNum](#) ([Vmgrid](#) *thee, int num)

Get specific grid in hierarchy.

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

Get grid in hierarchy which contains specified point or VNULL.

7.22.1 Detailed Description

Oracle for Cartesian mesh data.

7.22.2 Function Documentation

7.22.2.1 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

7.22.2.2 VEXTERNC *Vmgrid** Vmgrid_ctor ()

Construct *Vmgrid* object.

Author

Nathan Baker

Returns

Newly allocated and initialized *Vmgrid* object

7.22.2.3 VEXTERNC int Vmgrid_ctor2 (*Vmgrid * thee*)

Initialize *Vmgrid* object.

Author

Nathan Baker

Parameters

| | |
|-------------|--------------------------------------|
| <i>thee</i> | Newly allocated <i>Vmgrid</i> object |
|-------------|--------------------------------------|

Returns

Newly allocated and initialized Vmgrid object

7.22.2.4 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

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

Object destructor.

Author

Nathan Baker

Parameters

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

7.22.2.6 VEXTERNC void Vmgrid_dtor2 (*Vmgrid * thee*)

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

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

7.22.2.7 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

7.22.2.8 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

7.22.2.9 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

7.22.2.10 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

7.23 Multi-grid class

Multi-grid class used to encapsulate values and methods for multi-grid computations.

Files

- file [vmultigrid.h](#)

Data Structures

- struct [sVmultigrid](#)

Declaration of Vmultigrid class.

Typedefs

- typedef struct [sVmultigrid](#) [Vmultigrid](#)

Declaration of the Vmultigrid class as the Vmultigrid structure.

7.23.1 Detailed Description

Multi-grid class used to encapsulate values and methods for multi-grid computations.

7.24 Vopot class

Potential oracle for Cartesian mesh data.

Files

- file [vopot.h](#)

Potential oracle for Cartesian mesh data.

- file [vopot.c](#)

Class Vopot methods.

Data Structures

- struct [sVopot](#)

Electrostatic potential oracle for Cartesian mesh data.

Typedefs

- typedef struct [sVopot](#) [Vopot](#)

Declaration of the Vopot class as the Vopot structure.

Functions

- VEXTERNC [Vopot](#) * [Vopot_ctor](#) ([Vmgrid](#) *mgrid, [Vpbe](#) *pbe, [Vbcfl](#) bcfl)

Construct Vopot object with values obtained from Vpmg_readDX (for example)

- VEXTERNC int [Vopot_ctor2](#) ([Vopot](#) *thee, [Vmgrid](#) *mgrid, [Vpbe](#) *pbe, [Vbcfl](#) bcfl)

Initialize Vopot object with values obtained from Vpmg_readDX (for example)

- VEXTERNC int [Vopot_pot](#) ([Vopot](#) *thee, double x[3], double *pot)

Get potential value (from mesh or approximation) at a point.

- VEXTERNC void [Vopot_dtor](#) ([Vopot](#) **thee)

Object destructor.

- VEXTERNC void [Vopot_dtor2](#) ([Vopot](#) *thee)

FORTRAN stub object destructor.

- VEXTERNC int [Vopot_curvature](#) ([Vopot](#) *thee, double pt[3], int cflag, double *curv)

Get second derivative values at a point.

- VEXTERNC int [Vopot_gradient](#) ([Vopot](#) *thee, double pt[3], double grad[3])

Get first derivative values at a point.

7.24.1 Detailed Description

Potential oracle for Cartesian mesh data.

7.24.2 Function Documentation

7.24.2.1 VEXTERNC **Vopot*** Vopot_ctor (**Vmgrid * mgrid**, **Vpbe * pbe**, **Vbcfl bcfl**)

Construct Vopot object with values obtained from Vpmg_readDX (for example)

Author

Nathan Baker

Parameters

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

Returns

Newly allocated and initialized Vopot object

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

7.24.2.2 VEXTERNC int Vopot_ctor2 (**Vopot * thee**, **Vmgrid * mgrid**, **Vpbe * pbe**, **Vbcfl bcfl**)

Initialize Vopot object with values obtained from Vpmg_readDX (for example)

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 82 of file [vopot.c](#).

7.24.2.3 VEXTERNC int Vopot_curvature (**Vopot * thee**, **double pt[3]**, **int cflag**, **double * curv**)

Get second derivative values at a point.

Author

Nathan Baker

Parameters

| | |
|--------------|--|
| <i>thee</i> | Pointer to Vopot object |
| <i>pt</i> | Location to evaluate second derivative |
| <i>cflag</i> | <ul style="list-style-type: none"> • 0: Reduced Maximal Curvature • 1: Mean Curvature (Laplace) • 2: Gauss Curvature • 3: True Maximal Curvature |
| <i>curv</i> | Set to specified curvature value |

Returns

1 if successful, 0 otherwise

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

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

Object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 96 of file [vopot.c](#).

7.24.2.5 VEXTERNC void Vopot_dtor2 (Vopot * *thee*)

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

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

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

7.24.2.6 VEXTERNC int Vopot_gradient (Vopot * *thee*, double *pt*[3], double *grad*[3])

Get first derivative values at a point.

Author

Nathan Baker

Parameters

| | |
|-------------|-------------------------------|
| <i>thee</i> | Pointer to Vopot object |
| <i>pt</i> | Location to evaluate gradient |
| <i>grad</i> | Gradient |

Returns

1 if successful, 0 otherwise

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

7.24.2.7 VEXTERNC int Vopot_pot (Vopot * *thee*, double *x*[3], double * *pot*)

Get potential value (from mesh or approximation) at a point.

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

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

7.25 Vpmg class

A wrapper for Mike Holst's PMG multigrid code.

Files

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

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 \(Vpmg *thee, Vpmgp *parms, Vpbe *pbe, int focusFlag, Vpmg *pmgOLD, MGparm *mgparm, PBEparm_calcEnergy energyFlag\)](#)
FORTRAN stub constructor for the Vpmg class (uses previously-allocated memory)
- [VEXTERNC void Vpmg_dtor \(Vpmg **thee\)](#)
Object destructor.
- [VEXTERNC void Vpmg_dtor2 \(Vpmg *thee\)](#)
FORTRAN stub object destructor.
- [VEXTERNC int Vpmg_fillco \(Vpmg *thee, Vsurf_Meth surfMeth, double splineWin, Vchrg_Meth chargeMeth, int useDielXMap, Vgrid *dielXMap, int useDielYMap, Vgrid *dielYMap, int useDielZMap, Vgrid *dielZMap, int useKappaMap, Vgrid *kappaMap, int usePotMap, Vgrid *potMap, int useChargeMap, Vgrid *chargeMap\)](#)
Fill the coefficient arrays prior to solving the equation.
- [VEXTERNC int Vpmg_solve \(Vpmg *thee\)](#)
Solve the PBE using PMG.
- [VEXTERNC int Vpmg_solveLaplace \(Vpmg *thee\)](#)
Solve Poisson's equation with a homogeneous Laplacian operator using the solvent dielectric constant. This solution is performed by a sine wave decomposition.
- [VEXTERNC double Vpmg_energy \(Vpmg *thee, int extFlag\)](#)
Get the total electrostatic energy.
- [VEXTERNC double Vpmg_qtEnergy \(Vpmg *thee, int extFlag\)](#)

- VEXTERNC double `Vpmg_qfAtomEnergy` (`Vpmg` *thee, `Vatom` *atom)

Get the "fixed charge" contribution to the electrostatic energy.
- VEXTERNC double `Vpmg_qmEnergy` (`Vpmg` *thee, int extFlag)

Get the per-atom "fixed charge" contribution to the electrostatic energy.
- VEXTERNC double `Vpmg_dielEnergy` (`Vpmg` *thee, int extFlag)

Get the "mobile charge" contribution to the electrostatic energy.
- VEXTERNC double `Vpmg_dielGradNorm` (`Vpmg` *thee)

Get the "polarization" contribution to the electrostatic energy.
- VEXTERNC double `Vpmg_qfForce` (`Vpmg` *thee, double *force, int atomID, `Vsurf_Meth` srfm, `Vchrg_Meth` chgm)

Calculate the total force on the specified atom in units of k_B T/AA.
- VEXTERNC int `Vpmg_dbForce` (`Vpmg` *thee, double *dbForce, int atomID, `Vsurf_Meth` srfm)

Calculate the "charge-field" force on the specified atom in units of k_B T/AA.
- VEXTERNC int `Vpmg_ibForce` (`Vpmg` *thee, double *force, int atomID, `Vsurf_Meth` srfm)

Calculate the dielectric boundary forces 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` *nllInduced, int atomID, double force[3], double torque[3])

q-Phi direct polarization force between permanent multipoles and non-local induced dipoles based on 5th Order B-Splines. Keep in mind that the "non-local" induced dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.
- VEXTERNC void `Vpmg_ibDirectPolForce` (`Vpmg` *thee, `Vgrid` *perm, `Vgrid` *induced, int atomID, double force[3])

Ionic boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.

- VEXTERNC void `Vpmg_ibNLDirectPolForce` (`Vpmg` *thee, `Vgrid` *perm, `Vgrid` *nllInduced, int atomID, double force[3])

*Ionic boundary direct polarization force between permanent multipoles and non-local induced dipoles based on 5th order
Keep in mind that the "non-local" induced dipoles are just a mathematical quantity that result from differentiation of the
AMOEBA polarization energy.*

- VEXTERNC void `Vpmg_dbDirectPolForce` (`Vpmg` *thee, `Vgrid` *perm, `Vgrid` *induced, int atomID, double force[3])

*Dielectric boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the
sum of the permanent intramolecular field and the permanent reaction field.*

- VEXTERNC void `Vpmg_dbNLDirectPolForce` (`Vpmg` *thee, `Vgrid` *perm, `Vgrid` *nllInduced, int atomID, double force[3])

*Dielectric bounday direct polarization force between permanent multipoles and non-local induced dipoles. Keep in mind
that the "non-local" induced dipoles are just a mathematical quantity that result from differentiation of the AMOEBA
polarization energy.*

- VEXTERNC void `Vpmg_qfMutualPolForce` (`Vpmg` *thee, `Vgrid` *induced, `Vgrid` *nllInduced, int atomID, double force[3])

*Mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent con-
vergence of the solute induced dipoles and reaction field.*

- VEXTERNC void `Vpmg_ibMutualPolForce` (`Vpmg` *thee, `Vgrid` *induced, `Vgrid` *nllInduced, int atomID, double force[3])

*Ionic boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to
self-consistent convergence of the solute induced dipoles and reaction field.*

- VEXTERNC void `Vpmg_dbMutualPolForce` (`Vpmg` *thee, `Vgrid` *induced, `Vgrid` *nllInduced, int atomID, double force[3])

*Dielectric boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to
self-consistent convergence of the solute induced dipoles and reaction field.*

- VEXTERNC void `Vpmg_printColComp` (`Vpmg` *thee, char path[72], char title[72], char mxtype[3], int flag)

Print out a column-compressed sparse matrix in Harwell-Boeing format.

- VPRIATE 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.

- VPRIATE 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.

- VPRIATE 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.

- VPRIATE 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.

- VPRIATE 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, int *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.

7.25.1 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.

7.25.2 Function Documentation

7.25.2.1 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

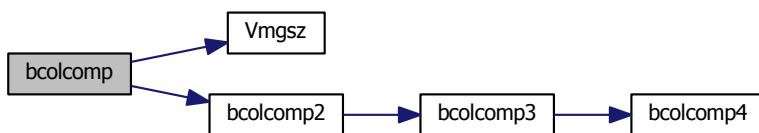
Tucker Beck [C Translation] Nathan Baker [Original] (mostly ripped off from Harwell-Boeing format documentation) Michael Schnieders)

Parameters

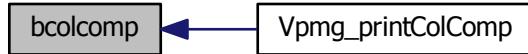
| | |
|---------------|--|
| <i>iparm</i> | |
| <i>rparm</i> | |
| <i>iwork</i> | |
| <i>rwork</i> | |
| <i>values</i> | |
| <i>rowind</i> | |
| <i>colptr</i> | |
| <i>flag</i> | Operation selection parameter 0 = Use Poisson operator only 1 = Use linearization of full operation around current solution. |

Definition at line 10729 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.25.2.2 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

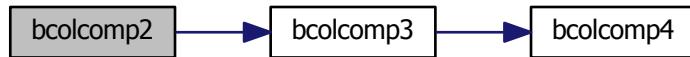
Tucker Beck [C Translation] Nathan Baker [Original] (mostly ripped off from Harwell-Boeing format documentation) Michael Schnieders

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 10784 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.25.2.3 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

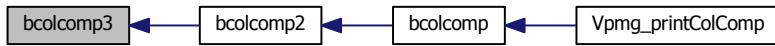
| | |
|---------------|--|
| <i>nx</i> | |
| <i>ny</i> | |
| <i>nz</i> | |
| <i>ipc</i> | |
| <i>rpc</i> | |
| <i>ac</i> | |
| <i>cc</i> | |
| <i>values</i> | |
| <i>rowind</i> | |
| <i>colptr</i> | |
| <i>flag</i> | |

Definition at line 10821 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.25.2.4 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

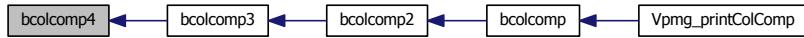
Tucker Beck [C Translation] Nathan Baker [Original] (mostly ripped off from Harwell-Boeing format documentation)
Michael Schnieders)

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 10849 of file [vpmg.c](#).

Here is the caller graph for this function:



7.25.2.5 VPRIVATE void pcolcomp (int * *nrow*, int * *ncol*, int * *nnzero*, double * *values*, int * *rowind*, int * *colptr*, char * *path*, char * *title*, char * *mxtpe*)

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>mxtpe</i> | |

Definition at line 11015 of file [vpmg.c](#).

Here is the caller graph for this function:



7.25.2.6 VEXTERNC void Vpackmg (int * *iparm*, double * *rparm*, int * *nrwk*, int * *nwk*, 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.

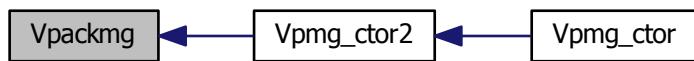
Author

Nathan Baker

Bug Can this path variable be replaced with a Vio socket?

Definition at line 569 of file [mgsubd.c](#).

Here is the caller graph for this function:



7.25.2.7 VEXTERNC Vpmg* Vpmg_ctor (Vpmgp * *parms*, Vpbe * *pbe*, int *focusFlag*, Vpmg * *pmgOLD*, MGparm * *mpparm*, PBEparm_calcEnergy *energyFlag*)

Constructor for the Vpmg class (allocates new memory)

Author

Nathan Baker

Returns

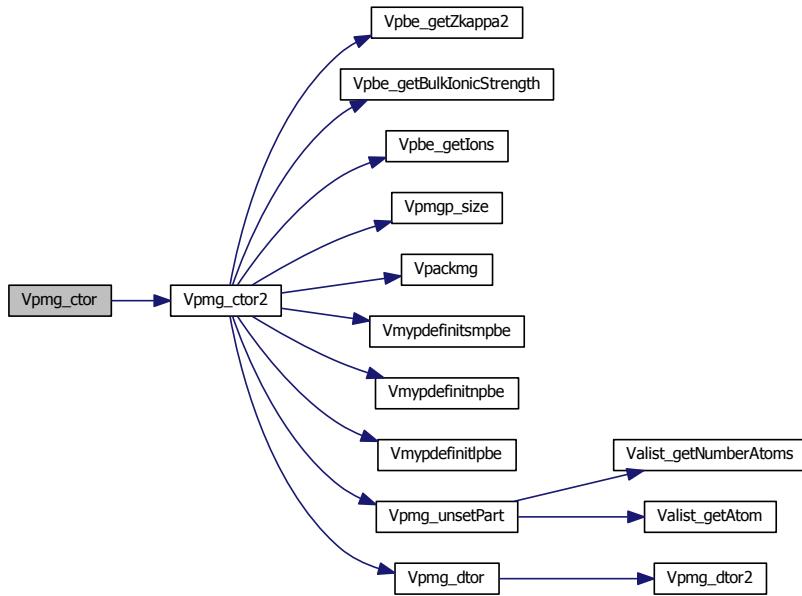
Pointer to newly allocated Vpmg object

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>mpparm</i> | MGparm parameter object for boundary conditions |
| <i>energyFlag</i> | What types of energies to calculate |

Definition at line 142 of file [vpmg.c](#).

Here is the call graph for this function:



7.25.2.8 VEXTERNC int Vpmg_ctor2 (Vpmg * *thee*, Vpmgp * *parms*, Vpbe * *pbe*, int *focusFlag*, Vpmg * *pmgOLD*, MGparm * *mpparm*, 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

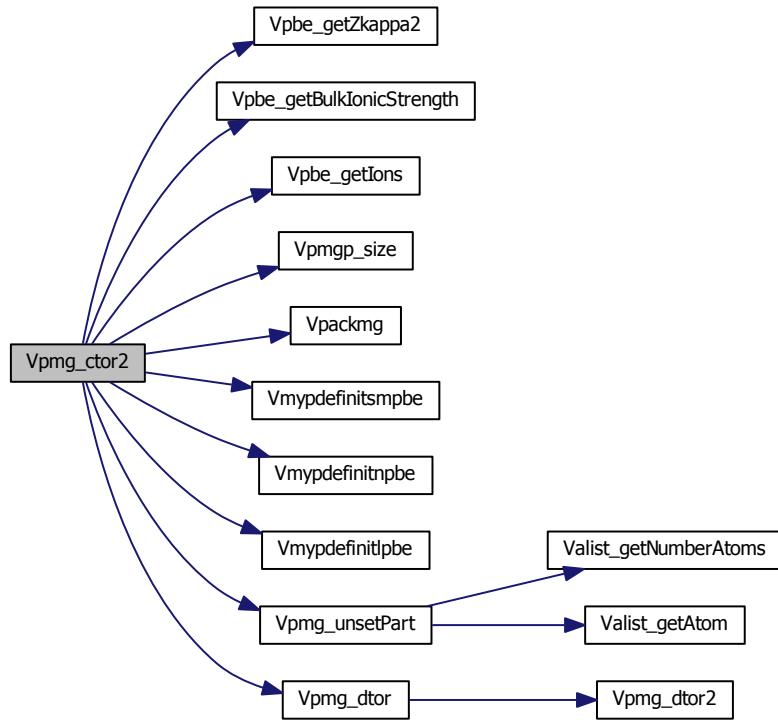
Parameters

| | |
|--------------|----------------------------|
| <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>mparm</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 154 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.25.2.9 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 |

7.25.2.10 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

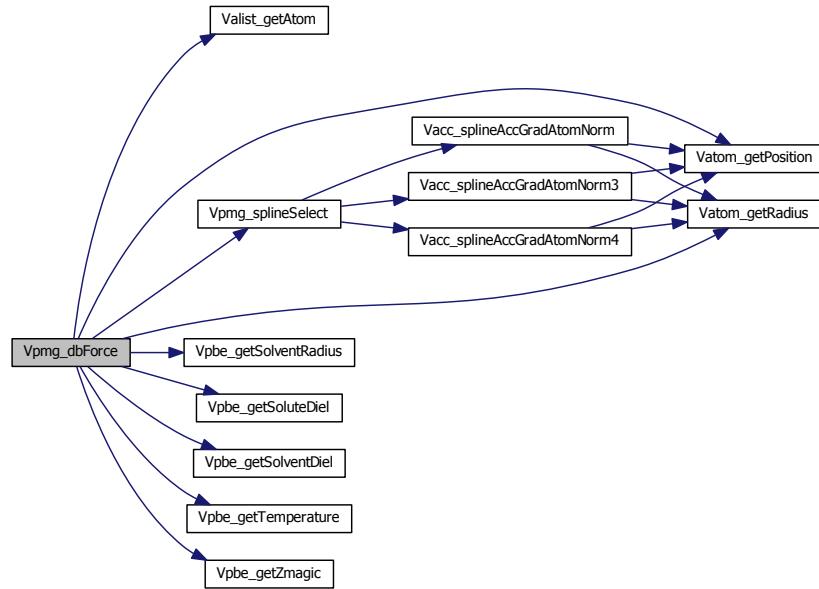
1 if successful, 0 otherwise

Parameters

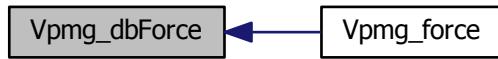
| | |
|----------------|---|
| <i>thee</i> | Vpmg object |
| <i>dbForce</i> | 3*sizeof(double) space to hold the dielectric boundary force in units of k_B T/AA |
| <i>atomID</i> | Valist ID of desired atom |
| <i>srfm</i> | Surface discretization method |

Definition at line 5998 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.25.2.11 VEXTERNC void Vpmg_dbMutualPolForce (*Vpmg * thee, Vgrid * induced, Vgrid * nInduced, int atomID, double force[3]*)

Dielectric boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

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 |

7.25.2.12 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 |

7.25.2.13 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 |

7.25.2.14 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

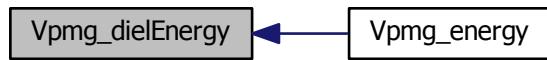
| | |
|----------------------|--|
| <code>thee</code> | Vpmg object |
| <code>extFlag</code> | If this was a focused calculation, include (1 – for serial calculations) or ignore (0 – for parallel calculations) energy contributions from outside the focusing domain |

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.25.2.15 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 A^2.

Author

Nathan Baker restrictions on the subdomain over which it is calculated. Such limits can be set via `Vpmg_setPart` and are generally useful for parallel runs.

Returns

The integral in units of A^2.

Parameters

| | |
|-------------|-------------|
| <i>thee</i> | Vpmg object |
|-------------|-------------|

Definition at line 1345 of file [vpmg.c](#).

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

Object destructor.

Author

Nathan Baker

Parameters

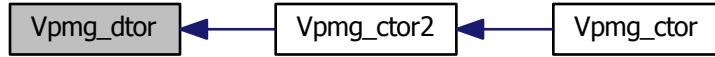
| | |
|-------------|--|
| <i>thee</i> | Pointer to memory location of object to be destroyed |
|-------------|--|

Definition at line 559 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.25.2.17 VEXTERNC void Vpmg_dtor2 (Vpmg * *thee*)

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

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

Definition at line 569 of file [vpmg.c](#).

Here is the caller graph for this function:



7.25.2.18 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

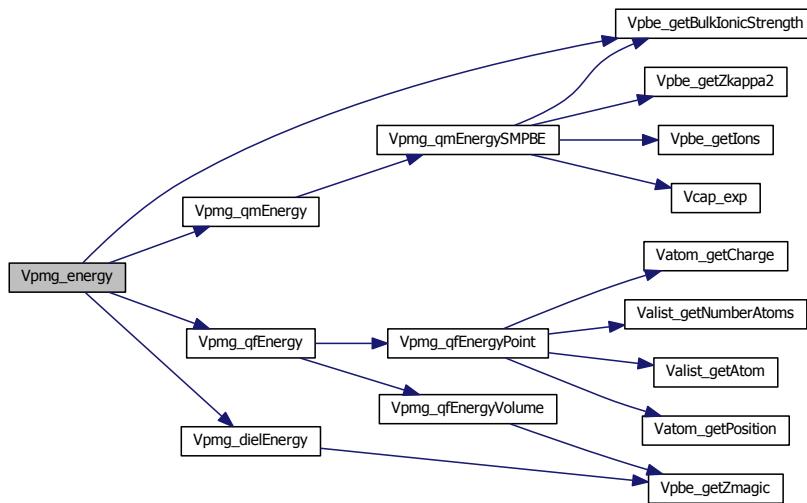
The electrostatic energy in units of $k_B T$.

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 1251 of file [vpmg.c](#).

Here is the call graph for this function:



7.25.2.19 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 |

7.25.2.20 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

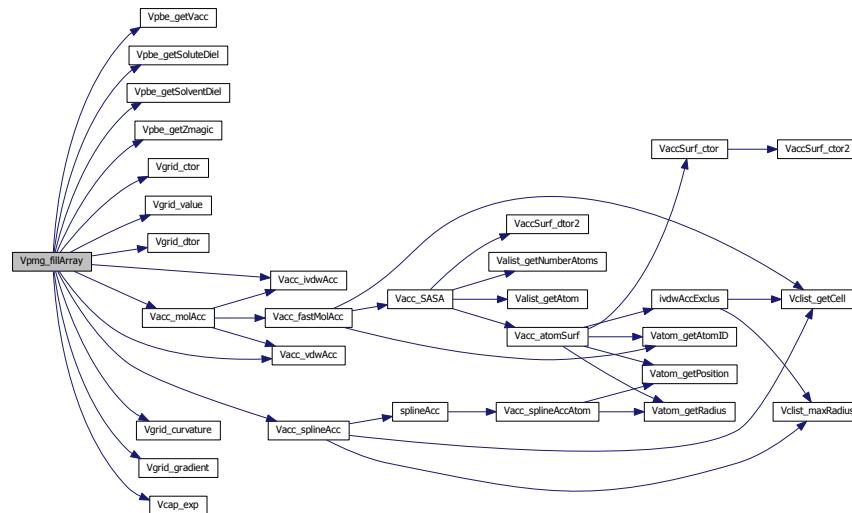
1 if successful, 0 otherwise

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 888 of file [vpmg.c](#).

Here is the call graph for this function:



7.25.2.21 VEXTERNC int Vpmg_fillco (Vpmg * *thee*, Vsurf_Meth *surfIMeth*, 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

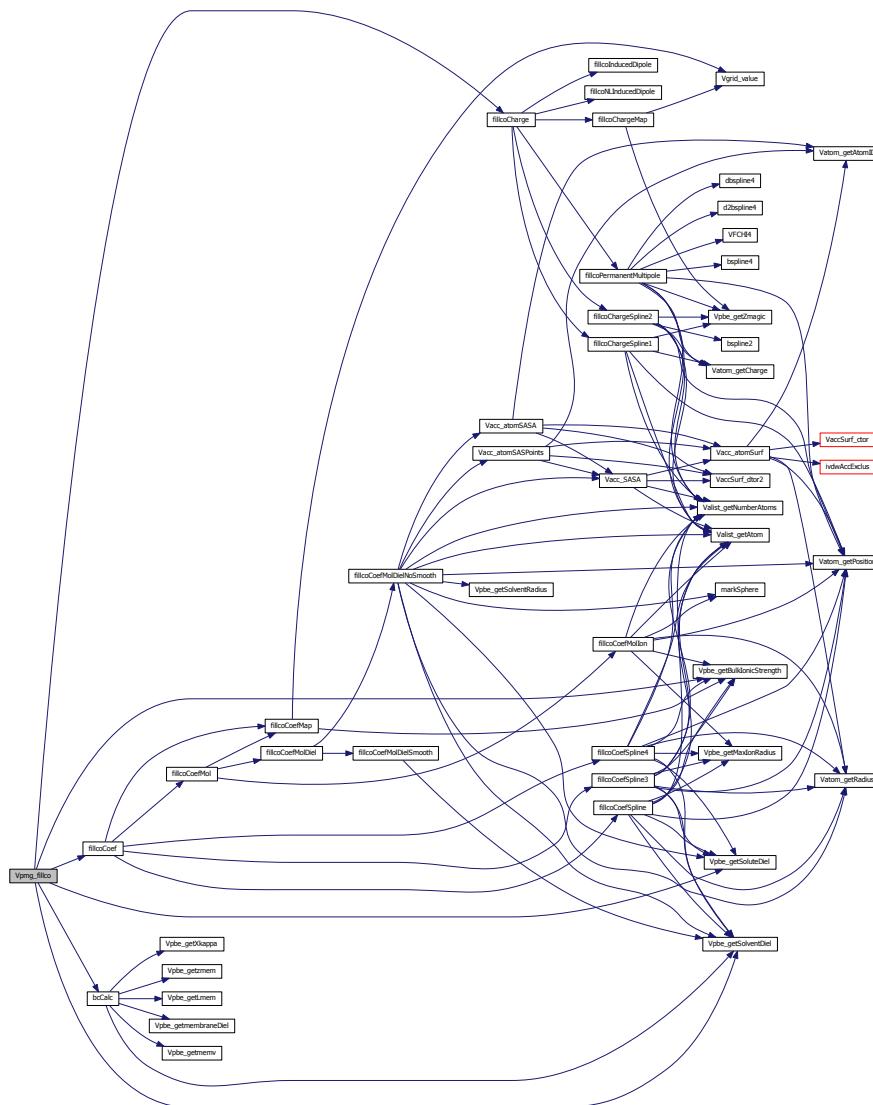
Bug useDielMap could only be passed once, not three times, to this function - why not just once? that's what the call in routines.c ends up doing - just passing useDielMap three times. - P. Ellis 11/3/11

Parameters

| | |
|---------------------|--|
| <i>thee</i> | Vpmg object |
| <i>surfMeth</i> | Surface discretization method |
| <i>splineWin</i> | Spline window (in A) for surfMeth = VSM SPLINE |
| <i>chargeMeth</i> | Charge discretization method |
| <i>useDielXMap</i> | Boolean to use dielectric map argument |
| <i>dielXMap</i> | External dielectric map |
| <i>useDielYMap</i> | Boolean to use dielectric map argument |
| <i>dielYMap</i> | External dielectric map |
| <i>useDielZMap</i> | Boolean to use dielectric map argument |
| <i>dielZMap</i> | External dielectric map |
| <i>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 5643 of file [vpmg.c](#).

Here is the call graph for this function:



7.25.2.22 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

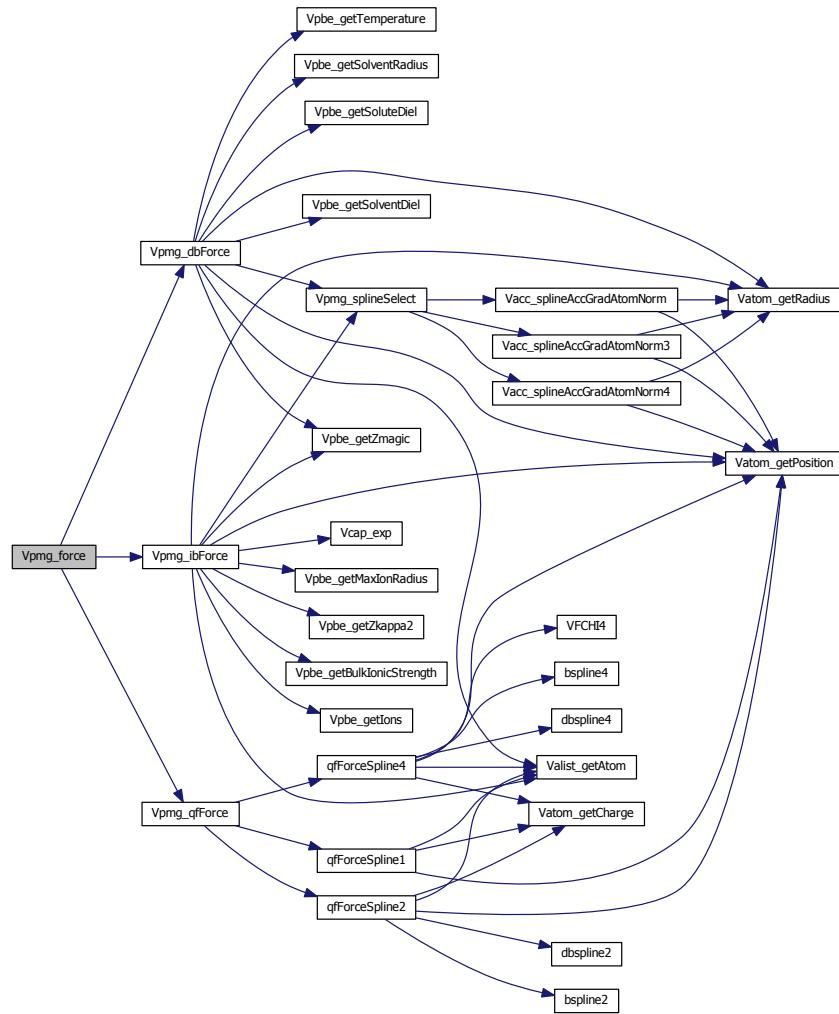
1 if successful, 0 otherwise

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 5810 of file [vpmg.c](#).

Here is the call graph for this function:



7.25.2.23 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 |

7.25.2.24 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

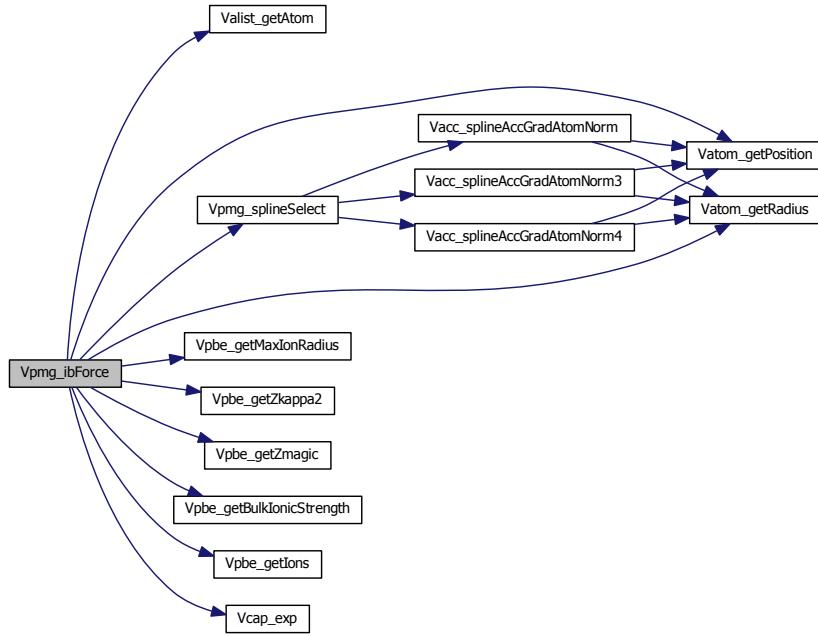
1 if successful, 0 otherwise

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 5833 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.25.2.25 VEXTERNC void Vpmg_ibMutualPolForce (Vpmg * *thee*, Vgrid * *induced*, Vgrid * *nInduced*, 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 |

7.25.2.26 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 |

7.25.2.27 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 |

7.25.2.28 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 80 of file [vpmg.c](#).

7.25.2.29 VEXTERNC void Vpmg_printColComp (Vpmg * *thee*, char *path*[72], char *title*[72], char *mxtyp*e[3], int *flag*)

Print out a column-compressed sparse matrix in Harwell-Boeing format.

Author

Nathan Baker

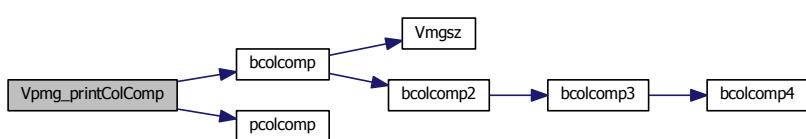
Bug Can this path variable be replaced with a Vio socket?

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>mxtyp</i> e | 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 88 of file [vpmg.c](#).

Here is the call graph for this function:



7.25.2.30 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: $\text{q}^2 / (4\pi \epsilon_0 r)$ 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 of $1/2$ 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

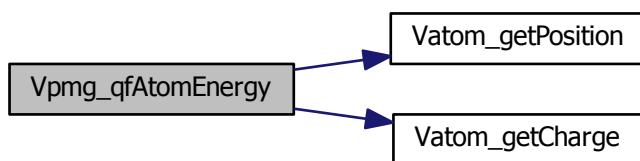
The fixed charge electrostatic energy in units of $k_B T$.

Parameters

| | |
|-------------|----------------------------------|
| <i>thee</i> | The <code>Vpmg</code> object |
| <i>atom</i> | The atom for energy calculations |

Definition at line 1794 of file [vpmg.c](#).

Here is the call graph for this function:



7.25.2.31 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 |

7.25.2.32 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: \form#12 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

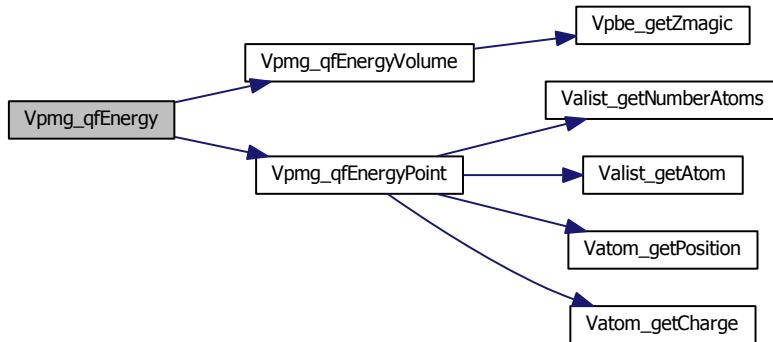
The fixed charge electrostatic energy in units of k_B T.

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 1690 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.25.2.33 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

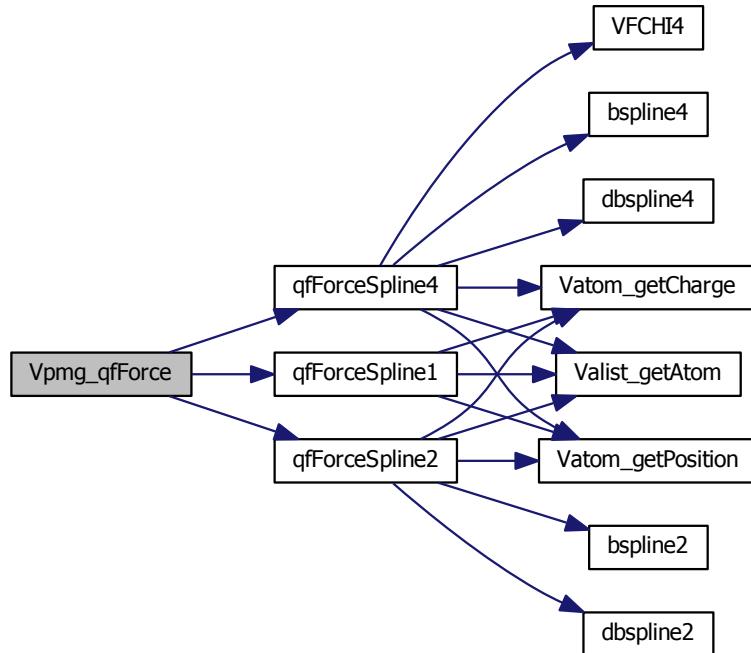
1 if sucessful, 0 otherwise

Parameters

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

Definition at line 6255 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.25.2.34 VEXTERNC void Vpmg_qfMutualPolForce (*Vpmg * thee, Vgrid * induced, Vgrid * nllInduced, int atomID, double force[3]*)

Mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

Author

Michael Schnieders

Parameters

| | |
|-------------------|------------------------------------|
| <i>thee</i> | Vpmg object |
| <i>induced</i> | Induced dipole potential |
| <i>nllInduced</i> | Non-local induced dipole potential |
| <i>atomID</i> | Atom index |
| <i>force</i> | (returned) force |

7.25.2.35 VEXTERNC void Vpmg_qfNLDDirectPolForce (*Vpmg * thee, Vgrid * perm, Vgrid * nllInduced, int atomID, double force[3], double torque[3]*)

q-Phi direct polarization force between permanent multipoles and non-local induced dipoles based on 5th Order B-Splines. Keep in mind that the "non-local" induced dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

Author

Michael Schnieders

Parameters

| | |
|-------------------|------------------------------------|
| <i>thee</i> | Vpmg object |
| <i>perm</i> | Permanent multipole potential |
| <i>nllInduced</i> | Non-local induced dipole potential |
| <i>atomID</i> | Atom index |
| <i>force</i> | (returned) force |
| <i>torque</i> | (returned) torque |

7.25.2.36 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 |

7.25.2.37 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 |

7.25.2.38 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

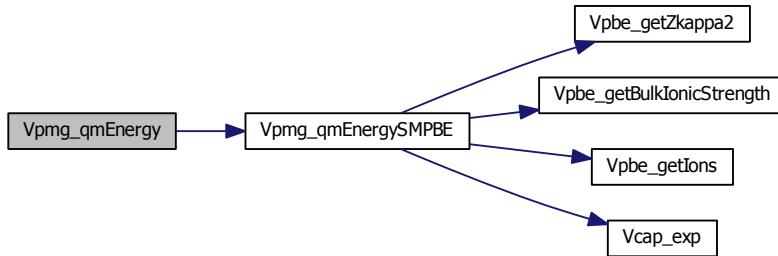
The mobile charge electrostatic energy in units of k_B T.

Parameters

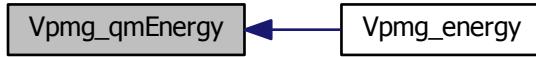
| | |
|----------------|--|
| <i>thee</i> | Vpmg object |
| <i>extFlag</i> | If this was a focused calculation, include (1 – for serial calculations) or ignore (0 – for parallel calculations) energy contributions from outside the focusing domain |

Definition at line 1389 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.25.2.39 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

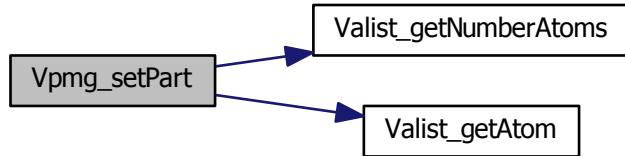
Nathan Baker

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 623 of file [vpmg.c](#).

Here is the call graph for this function:



7.25.2.40 VEXTERNC int Vpmg_solve (Vpmg * *thee*)

Solve the PBE using PMG.

Author

Nathan Baker

Returns

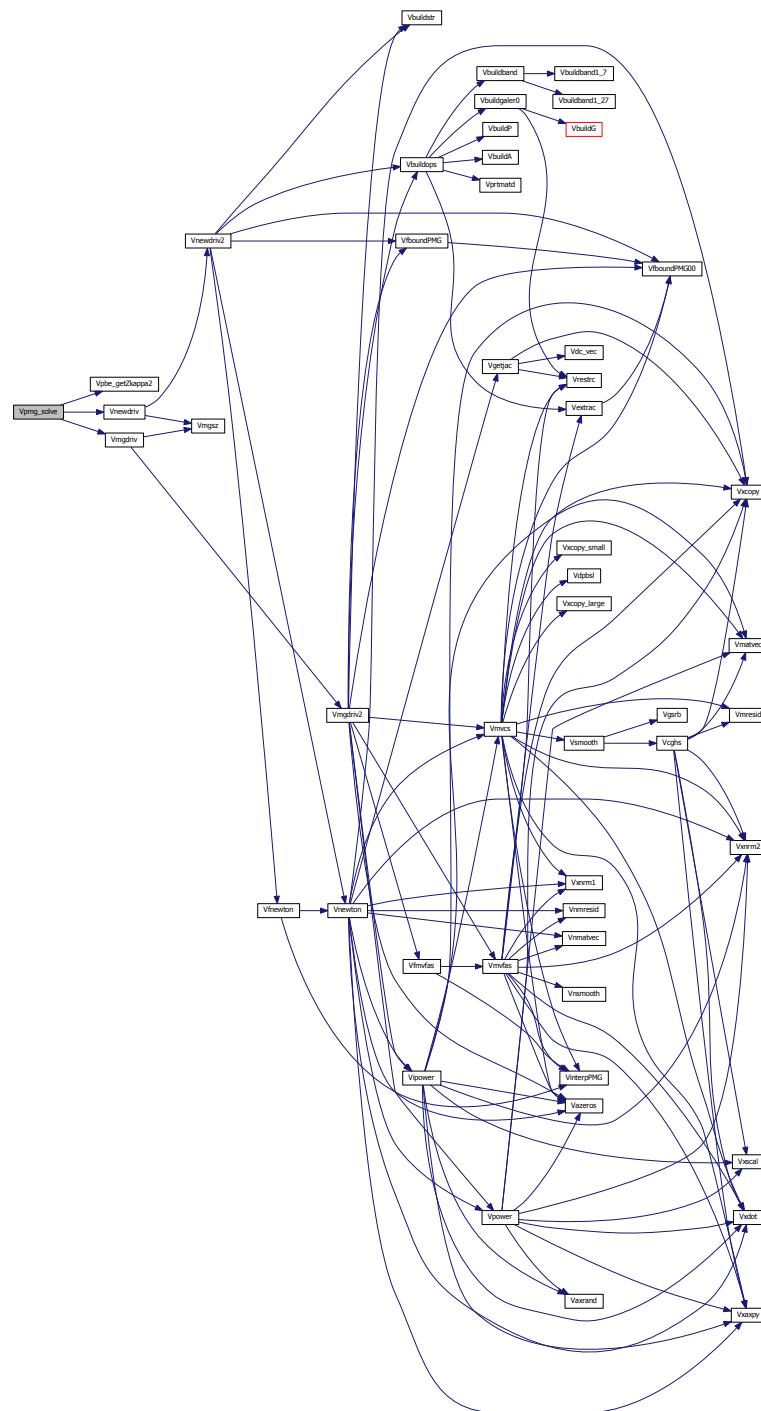
1 if successful, 0 otherwise

Parameters

| | |
|-------------|-------------|
| <i>thee</i> | Vpmg object |
|-------------|-------------|

Definition at line 399 of file [vpmg.c](#).

Here is the call graph for this function:



7.25.2.41 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

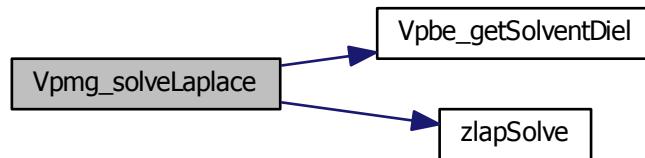
This function is really only for testing purposes as the PMG multigrid solver can solve the homogeneous system much more quickly. Perhaps we should implement an FFT version at some point...

Parameters

| | |
|-------------|-------------|
| <i>thee</i> | Vpmg object |
|-------------|-------------|

Definition at line 7030 of file [vpmg.c](#).

Here is the call graph for this function:



7.25.2.42 VEXTERNC void Vpmg_unsetPart (*Vpmg * thee*)

Remove partition restrictions.

Author

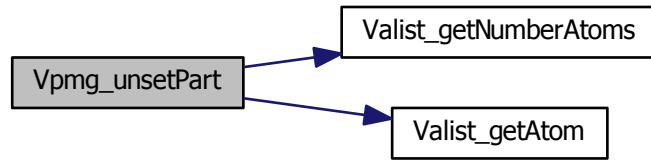
Nathan Baker

Parameters

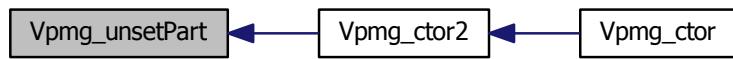
| | |
|-------------|-------------|
| <i>thee</i> | Vpmg object |
|-------------|-------------|

Definition at line 868 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.26 Vpmgp class

Parameter structure for Mike Holst's PMGP code.

Files

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

Data Structures

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

Typedefs

- typedef struct [sVpmgp](#) [Vpmgp](#)
Declaration of the Vpmgp class as the [sVpmgp](#) structure.

Functions

- VEXTERNC [Vpmgp * Vpmgp_ctor](#) ([MGparm](#) *mgparm)
Construct PMG parameter object and initialize to default values.
- VEXTERNC int [Vpmgp_ctor2](#) ([Vpmgp](#) *thee, [MGparm](#) *mgparm)
FORTRAN stub to construct PMG parameter object and initialize to default values.
- VEXTERNC void [Vpmgp_dtor](#) ([Vpmgp](#) **thee)
Object destructor.
- VEXTERNC void [Vpmgp_dtor2](#) ([Vpmgp](#) *thee)
FORTRAN stub for object destructor.
- VEXTERNC void [Vpmgp_size](#) ([Vpmgp](#) *thee)
Determine array sizes and parameters for multigrid solver.
- VEXTERNC void [Vpmgp_makeCoarse](#) (int numLevel, int nxOld, int nyOld, int nzOld, int *nxNew, int *nyNew, int *nzNew)
Coarsen the grid by the desired number of levels and determine the resulting numbers of grid points.

7.26.1 Detailed Description

Parameter structure for Mike Holst's PMGP code.

Note

Variables and many default values taken directly from PMG

7.26.2 Function Documentation

7.26.2.1 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 78 of file [vpmgp.c](#).

7.26.2.2 VEXTERNC int Vpmgp_ctor2 (Vpmgp * *thee*, MGparm * *mgparm*)

FORTRAN stub to construct PMG parameter object and initialize to default values.

Author

Nathan Baker

Parameters

| | |
|---------------|---|
| <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 95 of file [vpmgp.c](#).

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

Object destructor.

Author

Nathan Baker

Parameters

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

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

7.26.2.4 VEXTERNC void Vpmgp_dtor2 (*Vpmgp * thee*)

FORTRAN stub for object destructor.

Author

Nathan Baker

Parameters

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

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

7.26.2.5 VEXTERNC void Vpmgp_makeCoarse (int *numLevel*, int *nxOld*, int *nyOld*, int *nzOld*, int * *nxNew*, int * *nyNew*, int * *nzNew*)

Coarsen the grid by the desired number of levels and determine the resulting numbers of grid points.

Author

Mike Holst and Nathan Baker

Parameters

| | |
|-----------------|---|
| <i>numLevel</i> | Number of levels to coarsen |
| <i>nxOld</i> | Number of old grid points in this direction |
| <i>nyOld</i> | Number of old grid points in this direction |
| <i>nzOld</i> | Number of old grid points in this direction |
| <i>nxNew</i> | Number of new grid points in this direction |
| <i>nyNew</i> | Number of new grid points in this direction |
| <i>nzNew</i> | Number of new grid points in this direction |

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

7.26.2.6 VEXTERNC void Vpmgp_size (*Vpmgp * thee*)

Determine array sizes and parameters for multigrid solver.

Author

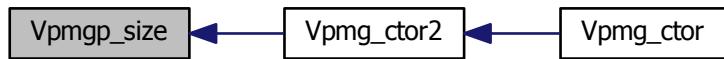
Mike Holst and Nathan Baker

Parameters

| | |
|-------------|--------------------|
| <i>thee</i> | Object to be sized |
|-------------|--------------------|

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

Here is the caller graph for this function:



7.27 C translation of Holst group PMG code

C translation of Holst group PMG code.

Macros

- `#define HARM02(a, b) (2.0 * (a) * (b) / ((a) + (b)))`

Multigrid subroutines.

- `#define MAXIONS 50`

Specifies the PDE definition for PMG to solve.

Functions

- 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.

- 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.

- 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.

- 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.

- VEXTERNC void `VbuildG` (int *nx, int *ny, int *nz, int *nx, int *ny, int *nz, int *nyc, int *nc, int *numdia, double *pcFF, double *acFF, double *ac)

Build Galerkin matrix structures.

- 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.

- 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.

- VEXTERNC void `VbuildG_27` (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 *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.

- 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.

- 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).

- 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)

Gauss-Seidel solver.

- VEXTERNC void [Vmavvec](#) (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 *nx, int *ny, int *nz, 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 *nx, int *ny, int *nz, double *xin, double *xout, double *pc)

Apply the prolongation operator.

- VEXTERNC void [Vextrac](#) (int *nx, int *ny, int *nz, int *nxc, int *ny, int *nz, double *xin, double *xout)

Simple injection of a fine grid function into coarse grid.

- VEXTERNC void [Vmvc5](#) (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 *ierrror, 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 *tr)

MG helper functions.

- VEXTERNC void [Vmadriv](#) (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 [Vmadriv2](#) (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.

- VEXTERNC void [Vfmvfaz](#) (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 [Vmfvfas](#) (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.

- VEXTERNC 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 *gxfc, 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.

- 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 *nx, int *ny, int *nz, 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.

- VEXTERNC 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)

saxy operation for a grid function with boundary values.

- VEXTERNC double [Vxnrm1](#) (int *nx, int *ny, int *nz, double *x)

Norm operation for a grid function with boundary values.

- VEXTERNC double [Vxnrm2](#) (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 [Vprtmad](#) (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *ac)

LINPACK interface.
- VEXTERNC void [Vdpbsl](#) (double *abd, int *lda, int *n, int *m, double *b)
- VEXTERNC void [Vmypdefinitpbe](#) (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 [Vmypdefinitpbe](#) (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 [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.
- 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 [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 *tr)

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 *tr)

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.
- 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 *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 *tr)

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 *tr)

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.

7.27.1 Detailed Description

C translation of Holst group PMG code.

7.27.2 Macro Definition Documentation

7.27.2.1 #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 64 of file [mgsubd.h](#).

7.27.2.2 #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 63 of file [mypdec.h](#).

7.27.3 Function Documentation

7.27.3.1 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

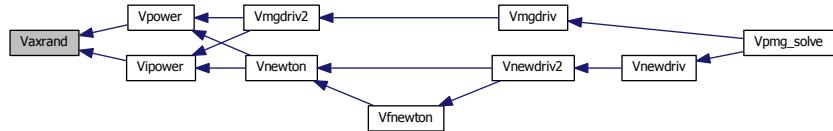
Replaces axrand from mikpckd.f

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 292 of file [mikpckd.c](#).

Here is the caller graph for this function:



7.27.3.2 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

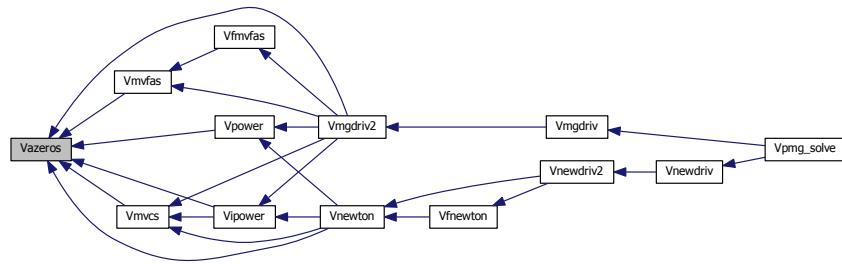
Replaces azeros from mikpckd.f

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 |
| <i>x</i> | The matrix to zero out |

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

Here is the caller graph for this function:



7.27.3.3 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.

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]
 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
```

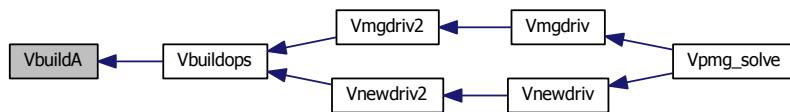
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; 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>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 55 of file [buildAd.c](#).

Here is the caller graph for this function:



7.27.3.4 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.

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

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.
*
*

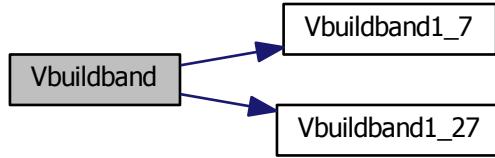
```

Parameters

| | |
|-------------|--|
| <i>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 54 of file [buildBd.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.5 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

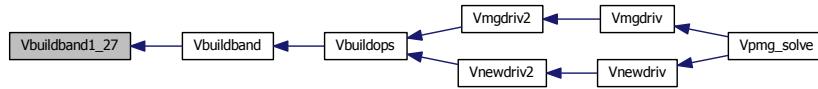
Parameters

| | |
|------------|--|
| <i>nx</i> | |
| <i>ny</i> | |
| <i>nz</i> | |
| <i>ipc</i> | |
| <i>rpc</i> | |
| <i>oC</i> | |
| <i>oE</i> | |
| <i>oN</i> | |

| |
|--------|
| uC |
| oNE |
| oNW |
| uE |
| uW |
| uN |
| uS |
| uNE |
| uNW |
| uSE |
| uSW |
| $ipcB$ |
| $rpcB$ |
| acB |
| n |
| m |
| lda |

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

Here is the caller graph for this function:



7.27.3.6 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

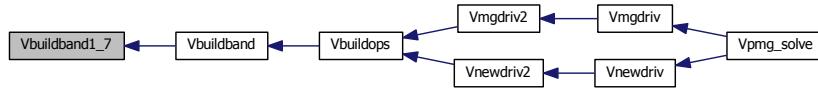
Parameters

| |
|-------|
| nx |
| ny |
| nz |
| ipc |
| rpc |
| oC |

| |
|-------------|
| <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 118 of file [buildBd.c](#).

Here is the caller graph for this function:



7.27.3.7 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.

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.
*
*
```

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

Attention

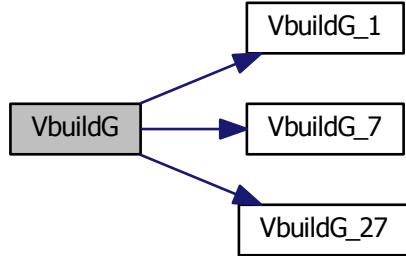
```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National 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

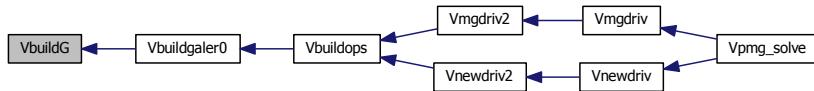
| |
|---------------|
| <i>nxf</i> |
| <i>nyf</i> |
| <i>nzf</i> |
| <i>nxc</i> |
| <i>nyc</i> |
| <i>nzc</i> |
| <i>numdia</i> |
| <i>pcFF</i> |
| <i>acFF</i> |
| <i>ac</i> |

Definition at line 52 of file [buildGd.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.8 VEXTERNC 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.

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) ] ]) ]);

```

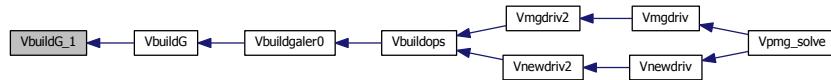
Parameters

| |
|-------------|
| <i>nxf</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 142 of file [buildGd.c](#).

Here is the caller graph for this function:



7.27.3.9 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( oNW(x+1,y-1,z) ): end: oSW := 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

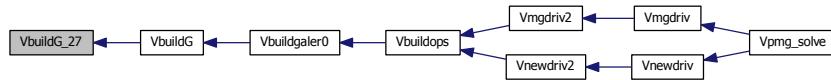
| |
|------------|
| <i>nxf</i> |
| <i>nyf</i> |
| <i>nzf</i> |
| <i>nx</i> |
| <i>ny</i> |
| <i>nz</i> |
| <i>oPC</i> |
| <i>oPN</i> |
| <i>oPS</i> |

| |
|-------------|
| <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>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> |

| |
|------|
| XuSW |
|------|

Definition at line 1216 of file [buildGd.c](#).

Here is the caller graph for this function:



7.27.3.10 VEXTERNC 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 * *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:
```

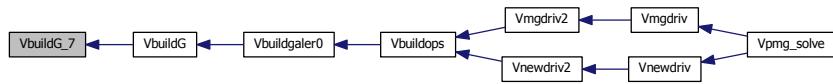
Parameters

| | |
|-------------|--|
| <i>nxf</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>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> | |

| |
|------|
| XuE |
| XuW |
| XuN |
| XuS |
| XuNE |
| XuNW |
| XuSE |
| XuSW |

Definition at line 420 of file [buildGd.c](#).

Here is the caller graph for this function:



7.27.3.11 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.

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 mgsubd.f

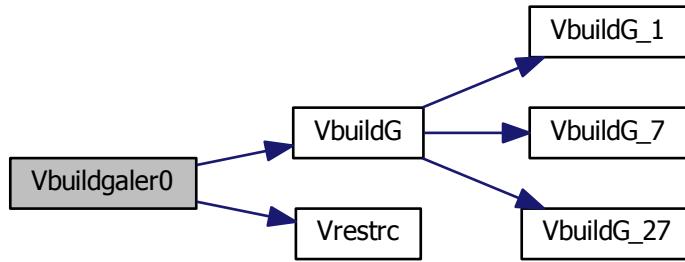
Parameters

| |
|---------------|
| <i>nxf</i> |
| <i>nyf</i> |
| <i>nzf</i> |
| <i>nxc</i> |
| <i>nyc</i> |
| <i>nzc</i> |
| <i>ipkey</i> |
| <i>numdia</i> |
| <i>pcFF</i> |

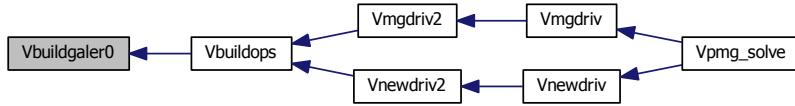
| |
|--------------|
| <i>ipcFF</i> |
| <i>rpcFF</i> |
| <i>acFF</i> |
| <i>ccFF</i> |
| <i>fcFF</i> |
| <i>ipc</i> |
| <i>rpc</i> |
| <i>ac</i> |
| <i>cc</i> |
| <i>fc</i> |

Definition at line 350 of file [mgsubd.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.12 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 * *gzcf*, double * *a1cf*, double * *a2cf*, double * *a3cf*, double * *ccf*, double * *fef*, 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.

Note

The fine level must be build before any coarse levels.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Replaces buildops from mgsubd.f

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

Attention

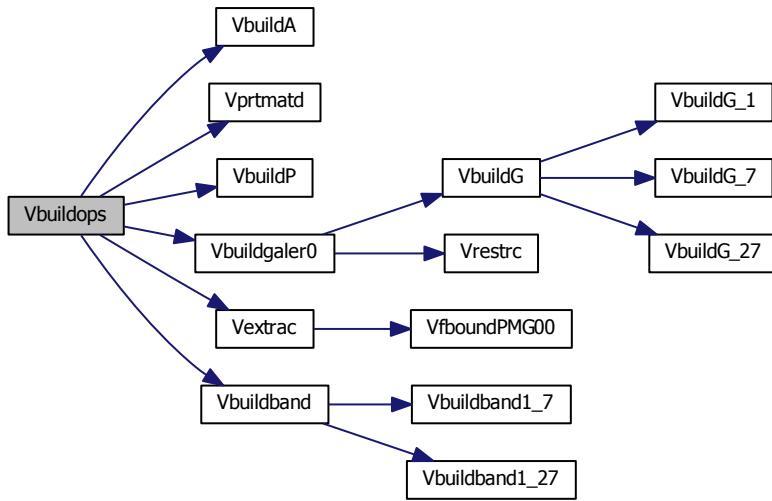
```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the Pacific 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

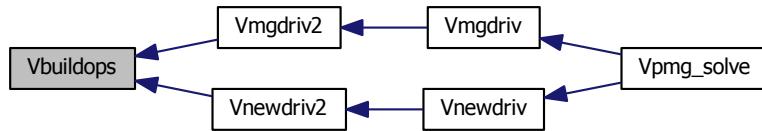
| | |
|---------------|--|
| <i>nx</i> | |
| <i>ny</i> | |
| <i>nz</i> | |
| <i>nlev</i> | |
| <i>ipkey</i> | |
| <i>iinfo</i> | |
| <i>ido</i> | |
| <i>iz</i> | |
| <i>mgprol</i> | |
| <i>mgcoar</i> | |
| <i>mgsolv</i> | |
| <i>mgdisc</i> | |
| <i>ipc</i> | |
| <i>rpc</i> | |
| <i>pc</i> | |
| <i>ac</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](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.13 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.

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.
*
*

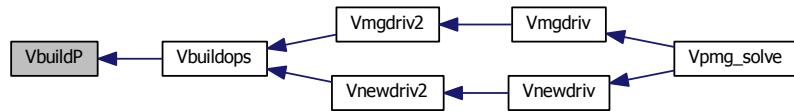
```

Parameters

| | |
|---------------|--|
| <i>nxf</i> | |
| <i>nyf</i> | |
| <i>nzf</i> | |
| <i>nxc</i> | |
| <i>nyc</i> | |
| <i>ncz</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 53 of file [buildPd.c](#).

Here is the caller graph for this function:



7.27.3.14 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

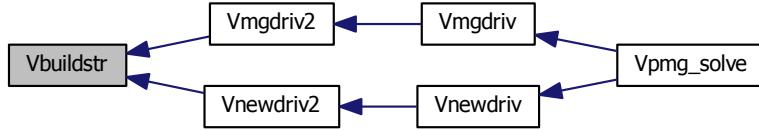
Replaces buildstr from mgsubd.f

Parameters

| | |
|-------------|--|
| <i>nx</i> | |
| <i>ny</i> | |
| <i>nz</i> | |
| <i>nlev</i> | |
| <i>iz</i> | |

Definition at line 264 of file [mgsubd.c](#).

Here is the caller graph for this function:



7.27.3.15 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

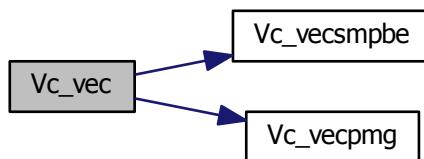
Replaces c_vec from mypde.f

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 121 of file [mypdec.c](#).

Here is the call graph for this function:



7.27.3.16 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 135 of file [mypdec.c](#).

Here is the caller graph for this function:



7.27.3.17 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

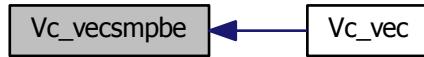
Replaces c_vecpmg from mypde.f

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 217 of file [mypdec.c](#).

Here is the caller graph for this function:



7.27.3.18 VPUBLIC void Veghs (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.
*
*

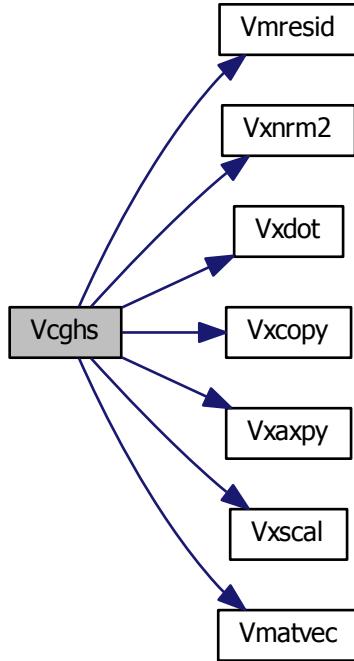
```

Parameters

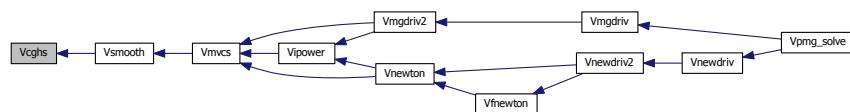
| | |
|-----------------|--|
| <i>nx</i> | |
| <i>ny</i> | |
| <i>nz</i> | |
| <i>ipc</i> | |
| <i>rpc</i> | |
| <i>ac</i> | |
| <i>cc</i> | |
| <i>fc</i> | |
| <i>x</i> | |
| <i>p</i> | |
| <i>ap</i> | |
| <i>r</i> | |
| <i>itmax</i> | |
| <i>iters</i> | |
| <i>errtol</i> | |
| <i>omega</i> | |
| <i>i resid</i> | |
| <i>iadjoint</i> | |

Definition at line 52 of file [cgd.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.19 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 346 of file [mypdec.c](#).

Here is the caller graph for this function:

**7.27.3.20 VPUBLIC void Vdpbsl (double * abd, int * lda, int * n, int * m, double * b)**

LINPACK interface.

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 $A*X = B$ 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 mgsubd.f

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.
*
*

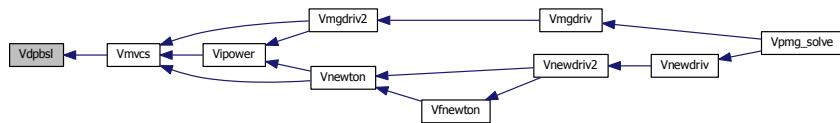
```

Parameters

| | |
|------------|---|
| <i>abd</i> | The output from dpbco or dpbfa |
| <i>lدا</i> | The leading dimension of the array <i>abd</i> |
| <i>n</i> | The order of the matrix <i>a</i> |
| <i>m</i> | The number of diagonals above the main diagonal |
| <i>b</i> | The right hand side vector |

Definition at line 53 of file [mlinpckd.c](#).

Here is the caller graph for this function:



7.27.3.21 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

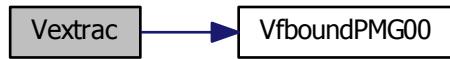
Parameters

| | |
|-------------|--|
| <i>nxf</i> | |
| <i>nyf</i> | |
| <i>nzf</i> | |
| <i>nxc</i> | |
| <i>ny</i> | |
| <i>nzc</i> | |
| <i>xin</i> | |
| <i>xout</i> | |

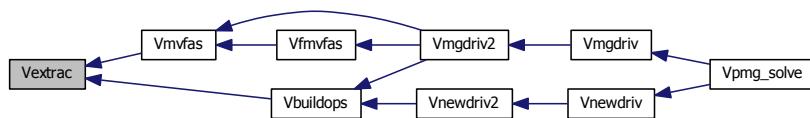
Generated on Thu Jul 19 2012 11:37:32 for APBS by Doxygen

Definition at line 1117 of file [matvecd.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.22 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

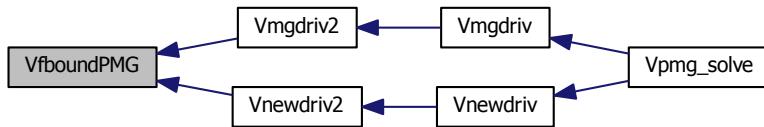
| | |
|---------------|--|
| <i>ibound</i> | |
| <i>nx</i> | |
| <i>ny</i> | |
| <i>nz</i> | |
| <i>x</i> | |
| <i>gxc</i> | |
| <i>gyc</i> | |
| <i>gzc</i> | |

Definition at line 206 of file [mikpckd.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.23 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

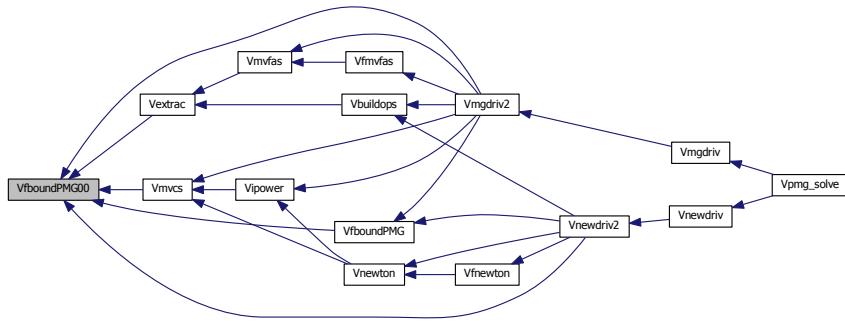
Replaces fboundPMG00 from mikpckd.f

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 257 of file [mikpckd.c](#).

Here is the caller graph for this function:



7.27.3.24 VPUBLIC 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 * *mgsmoo*, 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]

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

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.
*

```

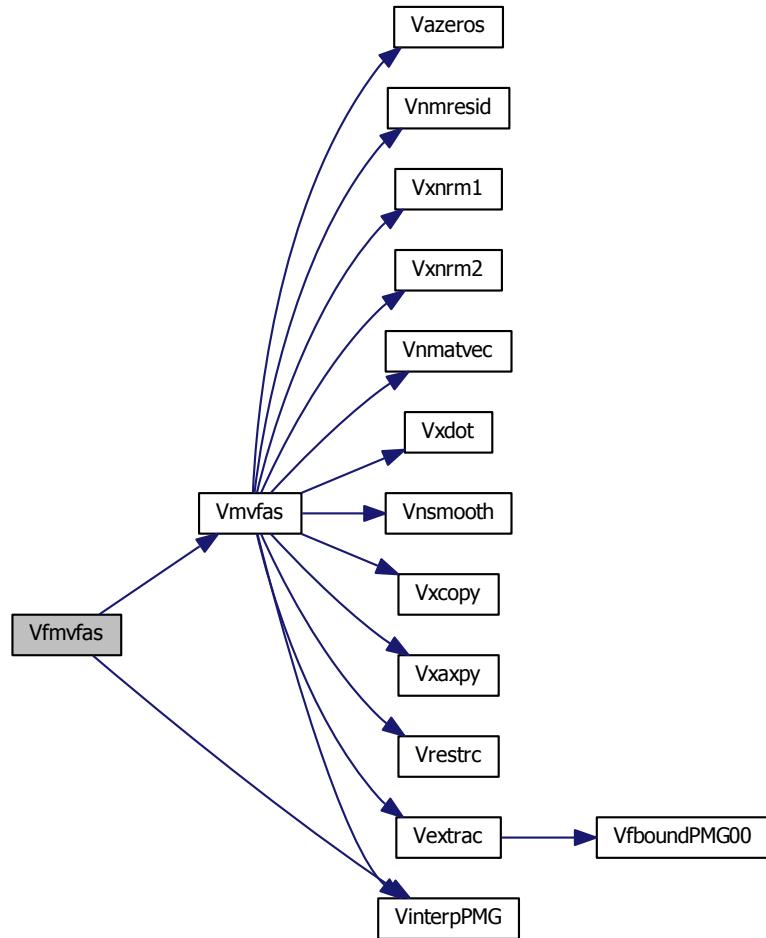
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS 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>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>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>fc</i> | |
| <i>tru</i> | |

Definition at line 52 of file [mgfasd.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.25 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 * iter, int * ierror, int * nlev, int * ilev, int * nlev_real, int * mgsolv, int * iok, int * iinfo, double * epsilon, 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]

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

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.
*
*

```

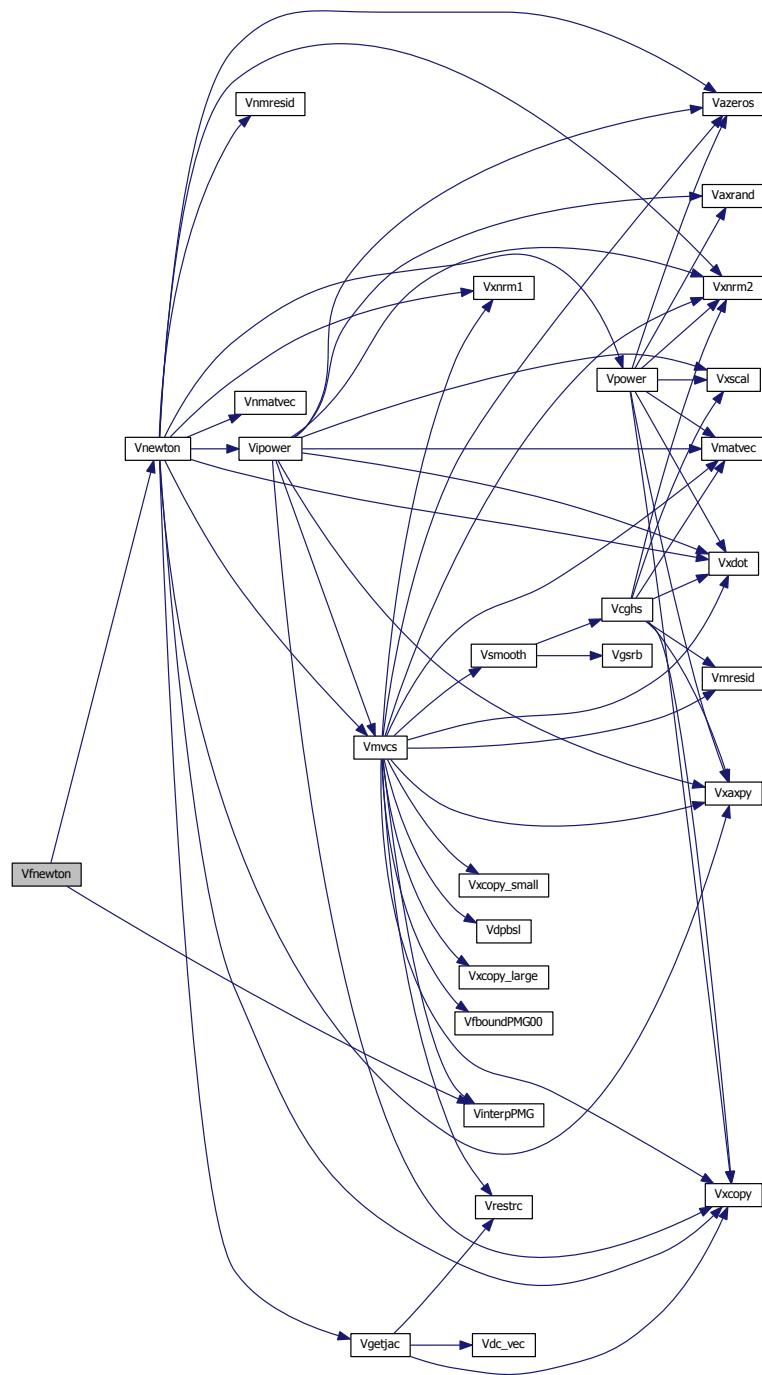
Parameters

| | |
|-----------|--|
| <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](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.26 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

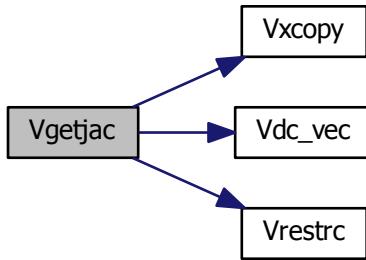
Replaces getjac from newtond.f

Parameters

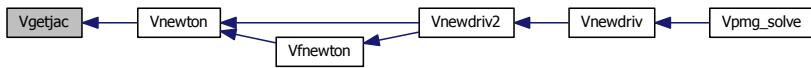
| |
|------------------|
| <i>nx</i> |
| <i>ny</i> |
| <i>nz</i> |
| <i>nlev_real</i> |
| <i>iz</i> |
| <i>lev</i> |
| <i>ipkey</i> |
| <i>x</i> |
| <i>r</i> |
| <i>cprime</i> |
| <i>rhs</i> |
| <i>cc</i> |
| <i>pc</i> |

Definition at line 547 of file [newtond.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.27 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 * *i resid*, int * *iadjoint*)

Guass-Seidel solver.

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 gsr from gsd.f

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.
*
*

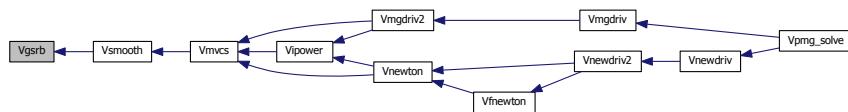
```

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>w1</i> |
| <i>w2</i> |
| <i>r</i> |
| <i>itmax</i> |
| <i>iters</i> |
| <i>errtol</i> |
| <i>omega</i> |
| <i>i resid</i> |
| <i>iadjoint</i> |

Definition at line 52 of file [gsd.c](#).

Here is the caller graph for this function:



7.27.3.28 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.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

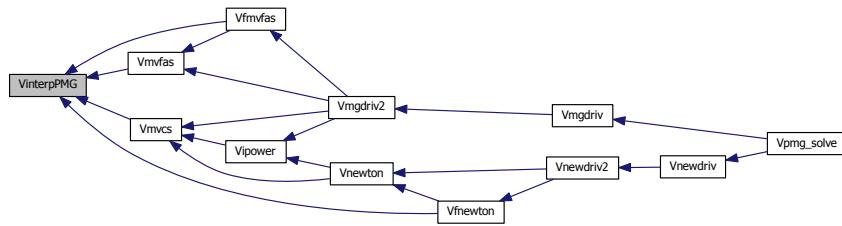
Replaces interpPMG from matvecd.f

Parameters

| | |
|-------------|--|
| <i>nxc</i> | |
| <i>nyc</i> | |
| <i>nzc</i> | |
| <i>nxf</i> | |
| <i>nyf</i> | |
| <i>nzf</i> | |
| <i>xin</i> | |
| <i>xout</i> | |
| <i>pc</i> | |

Definition at line 950 of file [matvecd.c](#).

Here is the caller graph for this function:



7.27.3.29 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 * epsilon, double * errtol, double * omega, int * nu1, int * nu2, int * mgsmo, 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

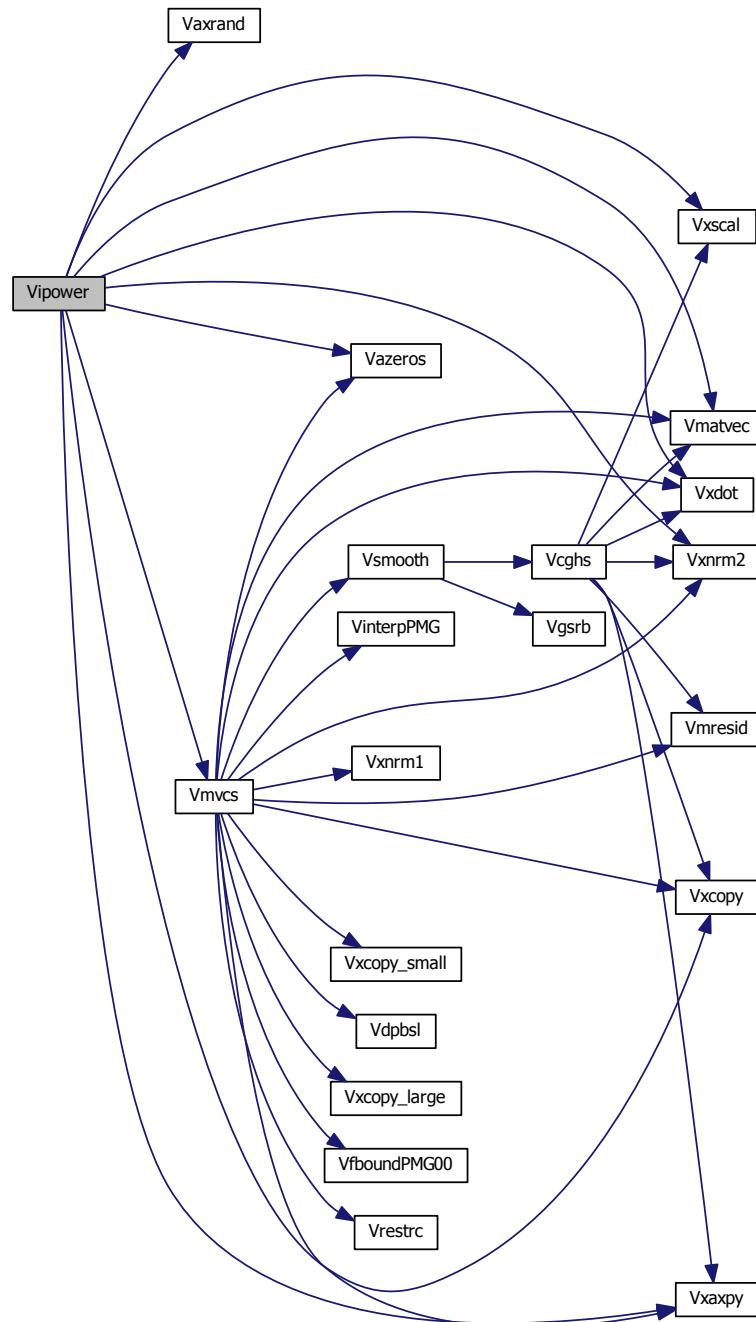
Replaces ipower from powerd.f

Parameters

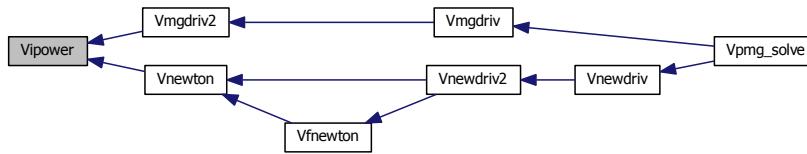
| | |
|---------------------|--|
| <i>nx</i> | |
| <i>ny</i> | |
| <i>nz</i> | |
| <i>u</i> | |
| <i>iz</i> | |
| <i>w0</i> | |
| <i>w1</i> | |
| <i>w2</i> | |
| <i>w3</i> | |
| <i>w4</i> | |
| <i>eigmin</i> | |
| <i>eigmin_model</i> | |
| <i>tol</i> | |
| <i>itmax</i> | |
| <i>iters</i> | |
| <i>nlev</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 161 of file [powerd.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.30 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.

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

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

\$Id:

Attention

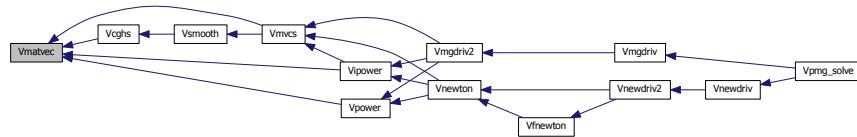
```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the Pacific 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>x</i> |
| <i>y</i> |

Definition at line 52 of file [matvecd.c](#).

Here is the caller graph for this function:



7.27.3.31 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.

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 mgdrv from mgdrv.f

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

```

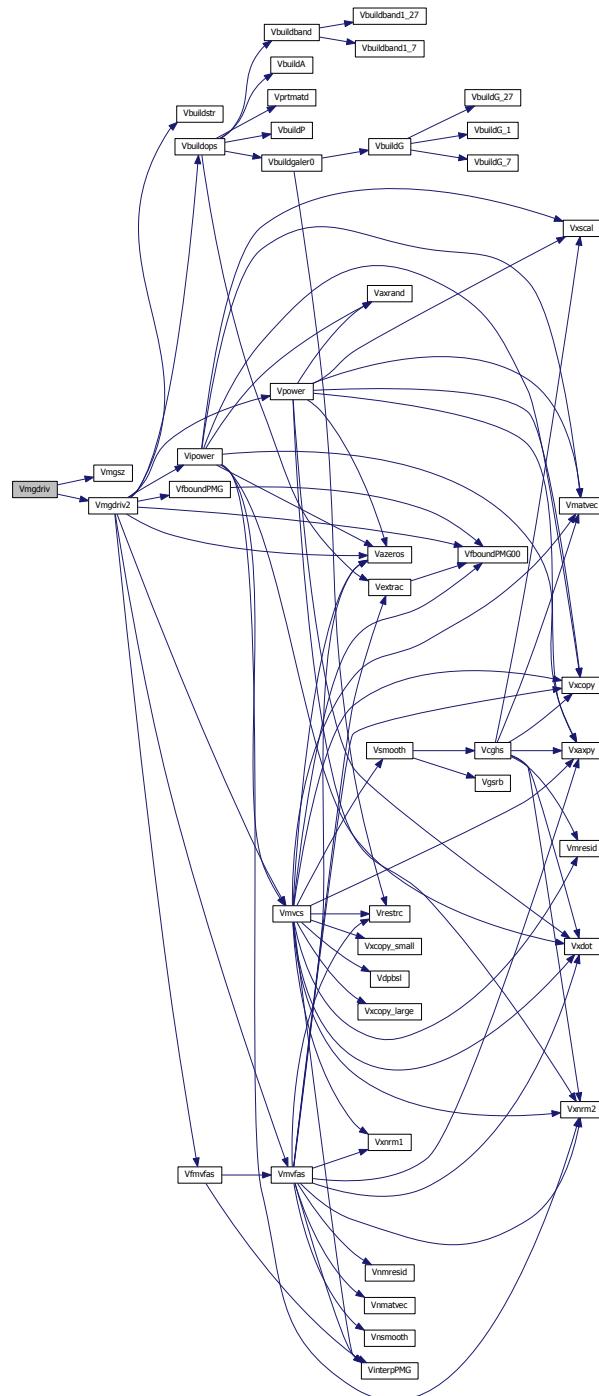
* contributors may be used to endorse or promote products derived from this
 * software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED 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>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 53 of file [mgdrvdc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.32 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 mgdrv2 from mgdrv.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 omega. (discontinuities must be along fine grid lines).
boundary function g(x,y,z) is smooth on boundary of omega.
```

```
the function c(u) is a possibly nonlinear function of the
unknown u, and varies (possibly discontinuously) with the
spatial position also.
```

user inputs:

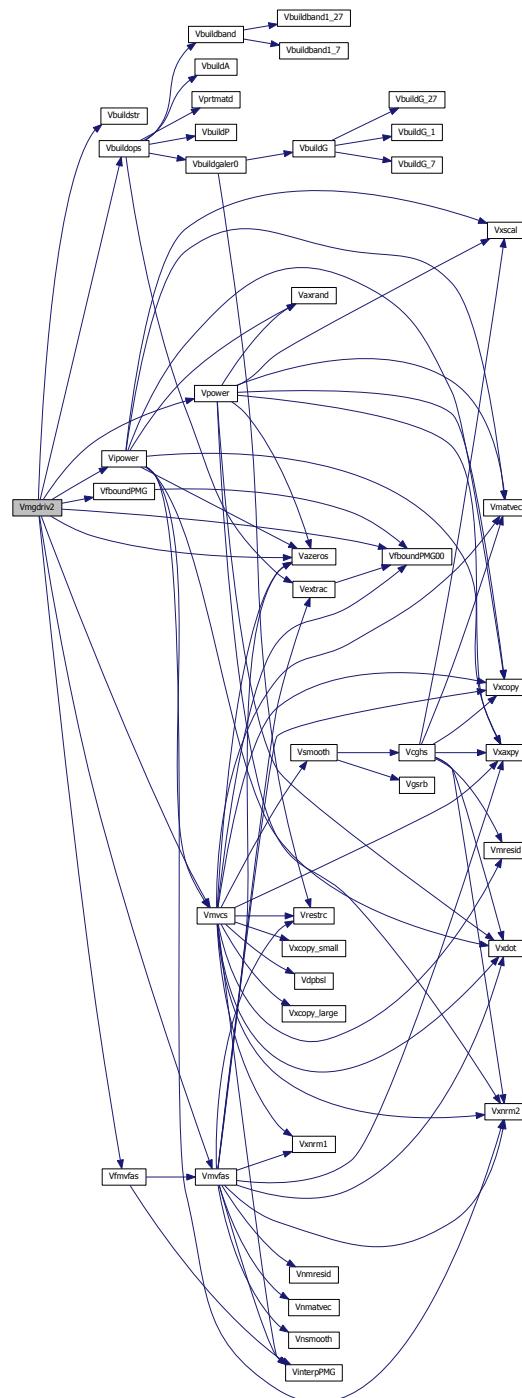
```
the user must provide the coefficients of the differential
operator, some initial parameter settings in an integer and a
real parameter array, and various work arrays.
```

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>fco</i> | |
| <i>tco</i> | |

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.33 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,mgsolv,nx,ny,nz,nlev)
iintot   = function_of(mgcoar,mgdisc,mgsolv,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

mgsolv   = 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. opers 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]

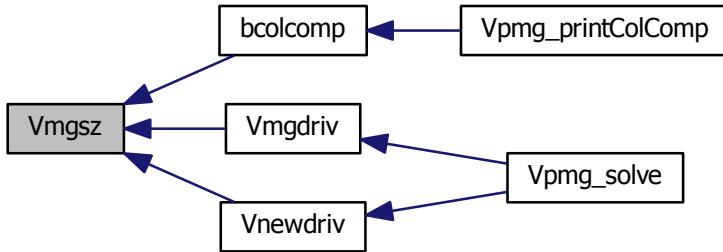
Replaces mgsz from mgdrv.f

Parameters

| |
|---------------|
| <i>mgcoar</i> |
| <i>mgdisc</i> |
| <i>mgsolv</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 540 of file [mgdrv.c](#).

Here is the caller graph for this function:



7.27.3.34 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

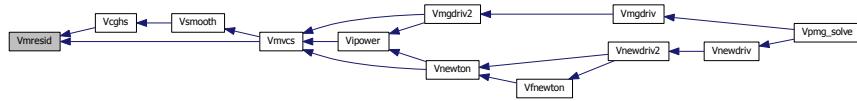
Replaces mresid from matvecd.f

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 437 of file [matvecd.c](#).

Here is the caller graph for this function:



7.27.3.35 VEXTERNC void Vmcs (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, 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.
*
*

```

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

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.
*
*

```

New grid size

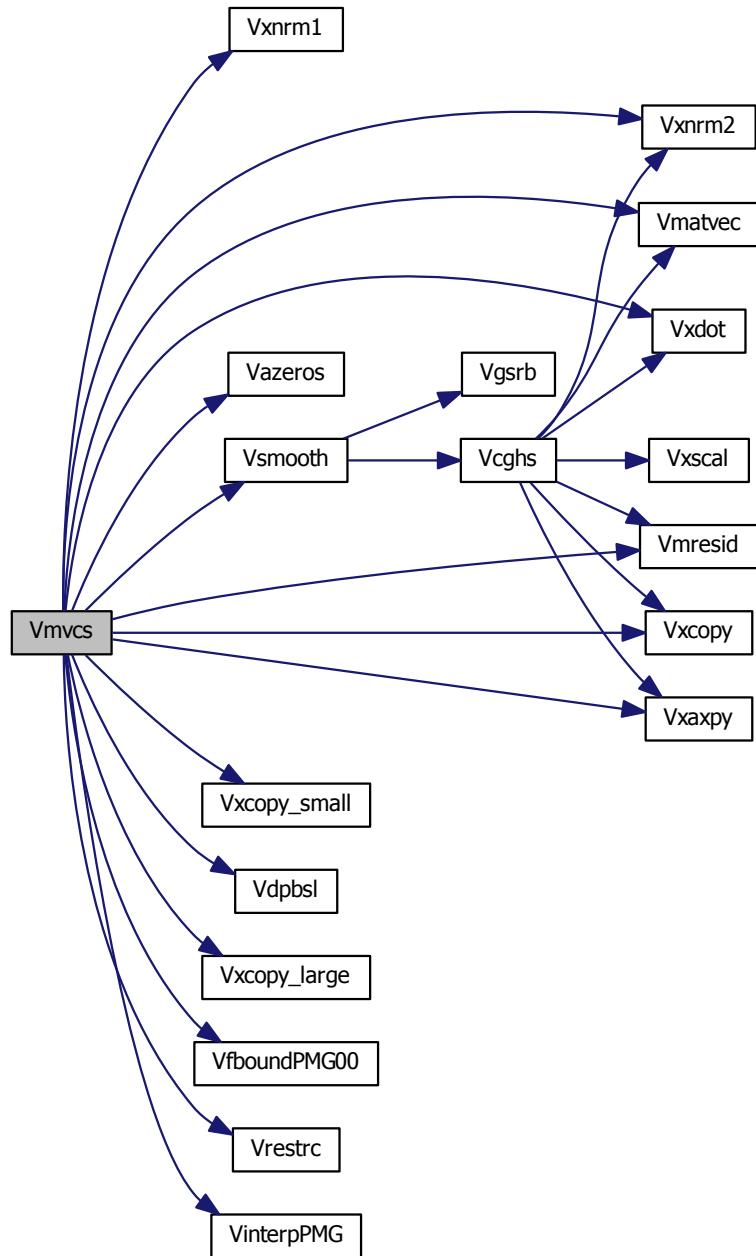
Parameters

| | |
|------------------|--|
| <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>mgsolv</i> | |
| <i>iock</i> | |
| <i>iinfo</i> | |
| <i>epsilon</i> | |
| <i>errtol</i> | |
| <i>omega</i> | |
| <i>nu1</i> | |
| <i>nu2</i> | |
| <i>mgsmoo</i> | |
| <i>ipc</i> | |
| <i>rpc</i> | |

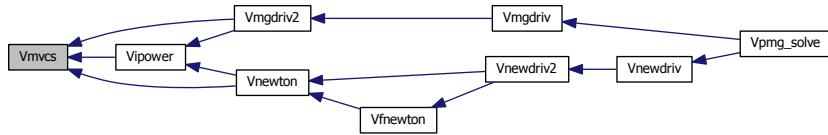
| |
|------------|
| <i>pc</i> |
| <i>ac</i> |
| <i>cc</i> |
| <i>fc</i> |
| <i>tru</i> |

Definition at line 52 of file [mgcsd.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.36 VEXTERNC void Vmvmfas (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 * *iock*, int * *jinfo*, 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: $u_{1,h} = \text{smooth}(L_h, f_h, u_h)$ (2) restrict defect: $d_H = r(L_h(u_{1,h}) - f_h)$ restrict solution: $u_H = r(u_{1,h})$ (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: $u_{2,h} = u_{1,h} - p(c_H - u_H)$ (5) post-smooth: $u_h = \text{smooth}(L_h, f_h, u_{2,h})$

(of course, c_H is determined with another two-grid algorithm)

implementation notes: (0) " $u_{1,h}$ " and " u_H " must be kept on each level until " c_H " is computed, and then all three are used to compute " $u_{2,h}$ ". (1) " u_h " (and then " $u_{1,h}$ ") 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.

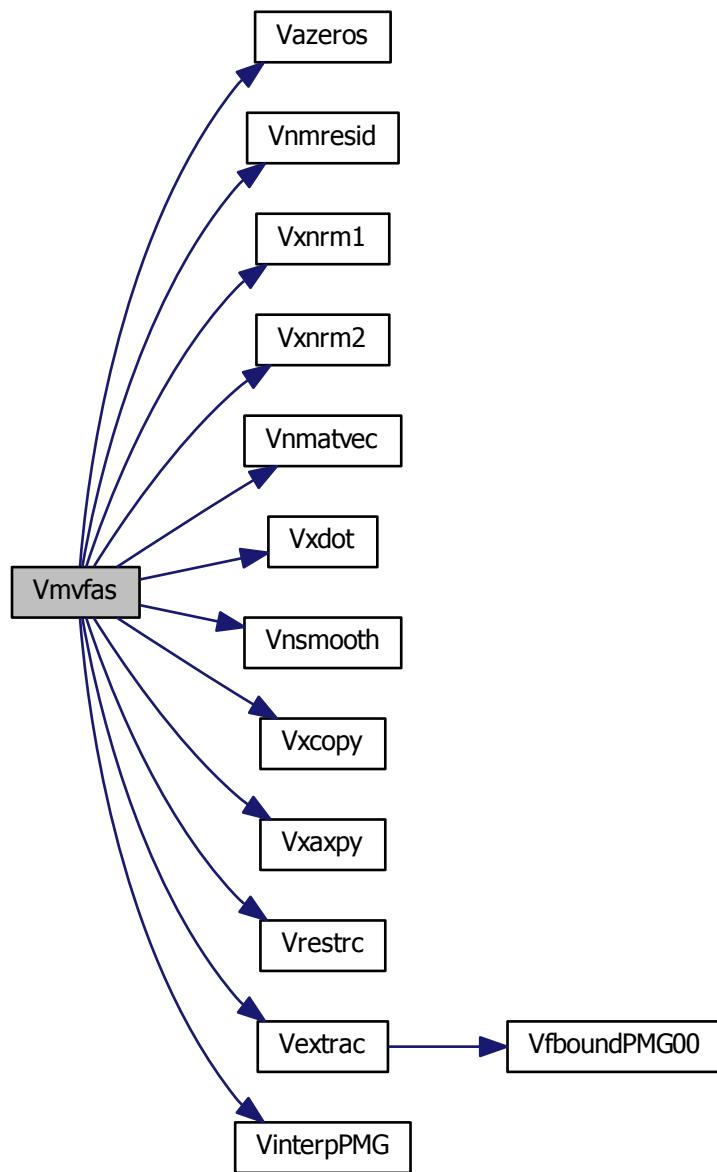
Parameters

| |
|-----------|
| <i>nx</i> |
| <i>ny</i> |
| <i>nz</i> |
| <i>x</i> |
| <i>iz</i> |
| <i>w0</i> |
| <i>w1</i> |

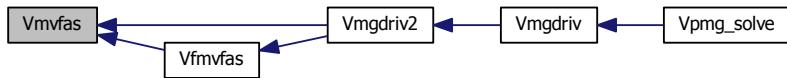
| | |
|-----------|--|
| w2 | |
| w3 | |
| w4 | |
| istop | |
| itmax | |
| iters | |
| ierror | |
| nlev | |
| ilev | |
| nlev_real | |
| mgsolv | |
| iock | |
| iinfo | |
| epsiln | |
| errtol | |
| omega | |
| nu1 | |
| nu2 | |
| mgsmoo | |
| ipc | |
| rpc | |
| pc | |
| ac | |
| cc | |
| fc | |
| tru | |

Definition at line 152 of file [mgfasd.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.37 VPUBLIC void Vmvpdefinitlpe (int * *tunion*, 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 mypdefinitlpe from mypde.f

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.
*
*

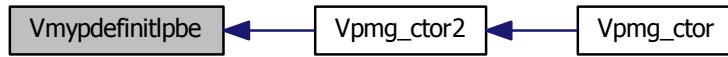
```

Parameters

| | |
|----------------|---|
| <i>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 [mypydec.c](#).

Here is the caller graph for this function:



7.27.3.38 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

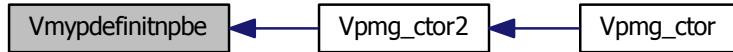
Replaces mypdefinitnpbe from mypde.f

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 72 of file [mypydec.c](#).

Here is the caller graph for this function:



7.27.3.39 VEXTERNC void Vmypdefinitmpbe (int * *tion*, 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

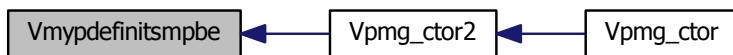
Replaces mypdefinitmpbe from mypde.f

Parameters

| | |
|-----------------|---|
| <i>tion</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 92 of file [mypydec.c](#).

Here is the caller graph for this function:



7.27.3.40 VEXTERNC void Vnewdrv (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.
*
*

```

Driver for a screaming inexact-newton-multilevel solver.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces newdrv from newdrv.f

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.
*
*

```

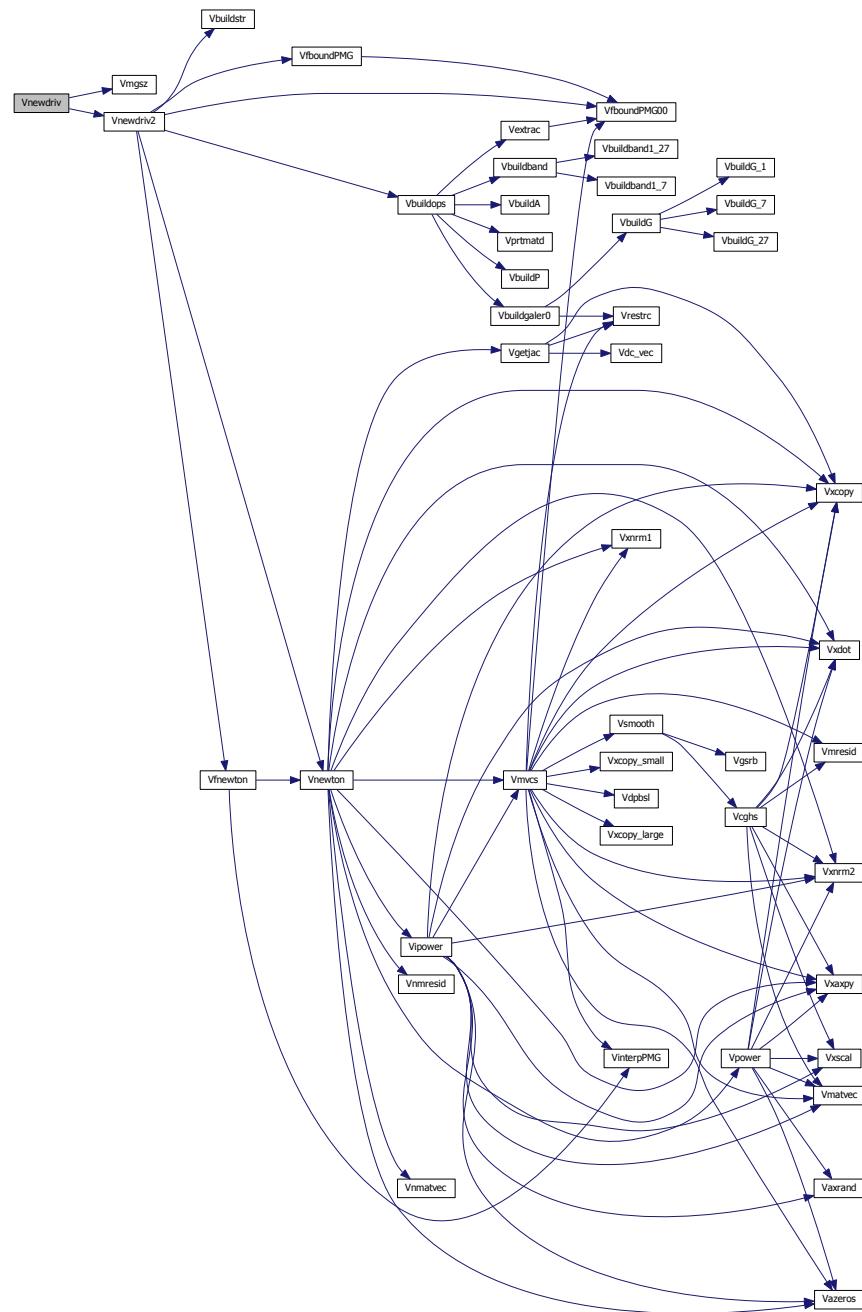
Parameters

| | |
|--------------|--|
| <i>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 [newdrvdc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.41 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:

```
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 omega. (discontinuities must be along fine grid lines).
boundary function g(x,y,z) is smooth on boundary of omega.
```

```
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

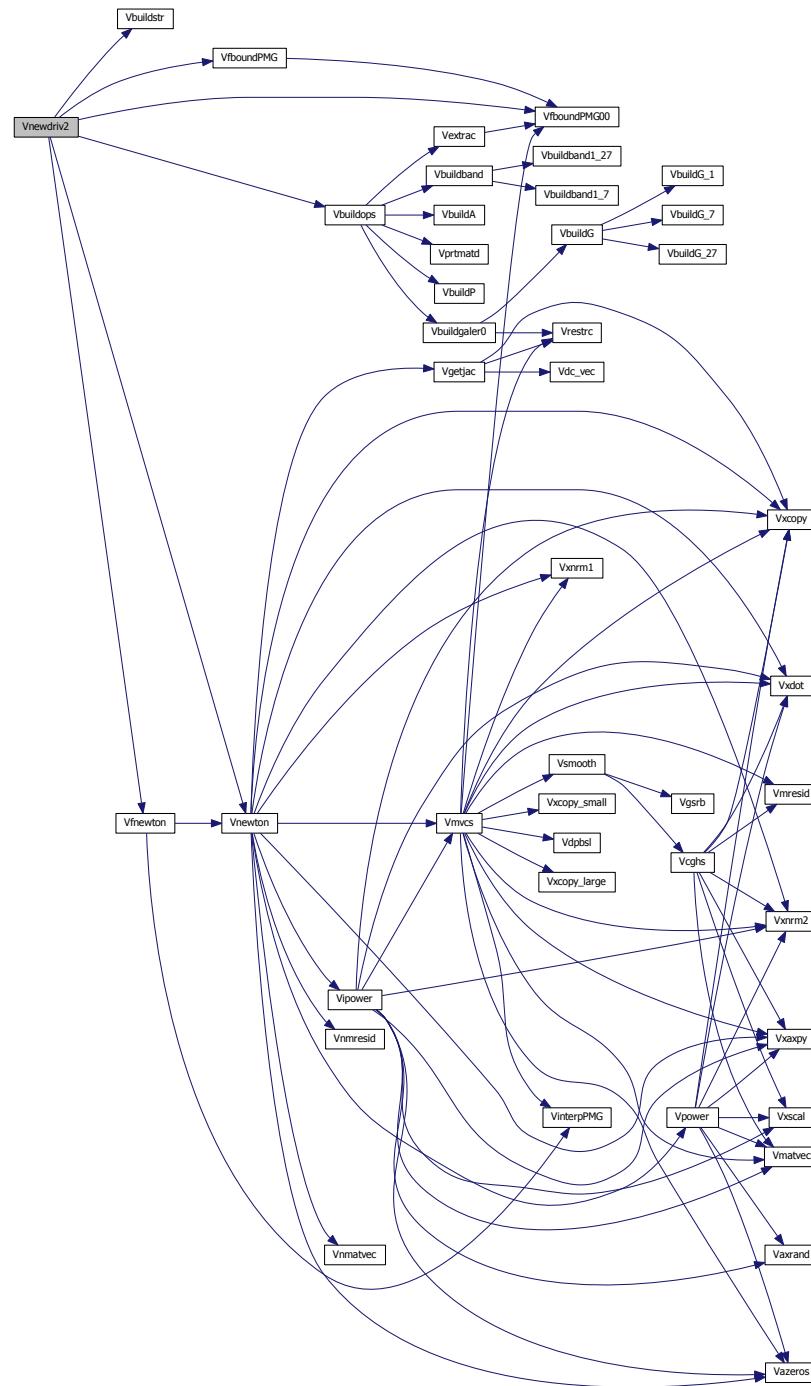
Replaces newdrv2 from newdrv.f

Parameters

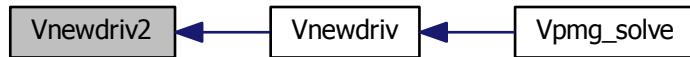
| | |
|--------------|--|
| <i>iparm</i> | |
| <i>rparm</i> | |
| <i>nx</i> | |
| <i>ny</i> | |
| <i>nz</i> | |
| <i>u</i> | |
| <i>iz</i> | |
| <i>w1</i> | |
| <i>w2</i> | |
| <i>ipc</i> | |
| <i>rpc</i> | |
| <i>pc</i> | |
| <i>ac</i> | |
| <i>cc</i> | |
| <i>fc</i> | |
| <i>xf</i> | |
| <i>yf</i> | |
| <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 180 of file [newdrvdc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.42 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 * iter, int * ierror, int * nlev, int * ilev, int * nlev_real, int * mgsolv, int * iok, int * iinfo, double * epsilon, 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

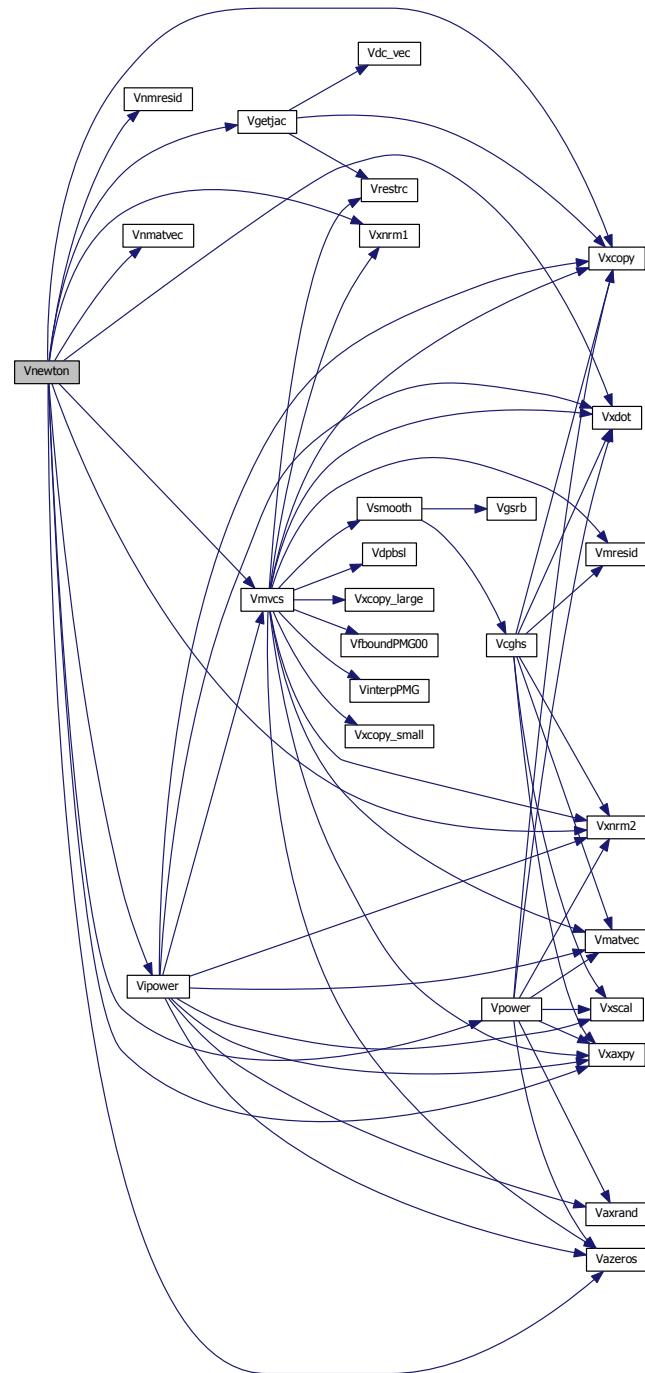
Parameters

| |
|------------------|
| <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>iter</i> |
| <i>ierror</i> |
| <i>nlev</i> |
| <i>ilev</i> |
| <i>nlev_real</i> |
| <i>mgsolv</i> |
| <i>iok</i> |
| <i>iinfo</i> |
| <i>epsilon</i> |
| <i>errtol</i> |
| <i>omega</i> |

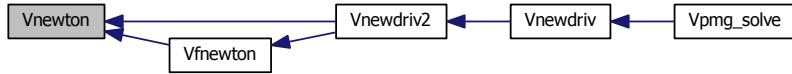
| |
|---------------|
| <i>nu1</i> |
| <i>nu2</i> |
| <i>mgsmoo</i> |
| <i>cprime</i> |
| <i>rhs</i> |
| <i>xtmp</i> |
| <i>ipc</i> |
| <i>rpc</i> |
| <i>pc</i> |
| <i>ac</i> |
| <i>cc</i> |
| <i>fc</i> |
| <i>tru</i> |

Definition at line 163 of file [newtond.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.43 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

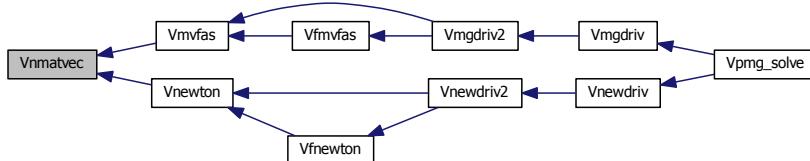
Replaces nmatvec from matvecd.f

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 236 of file [matvecd.c](#).

Here is the caller graph for this function:



7.27.3.44 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

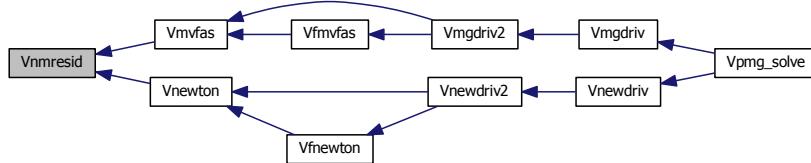
Replaces nmresid from matvecd.f

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 [619](#) of file [matvecd.c](#).

Here is the caller graph for this function:



7.27.3.45 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 * itmax, 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

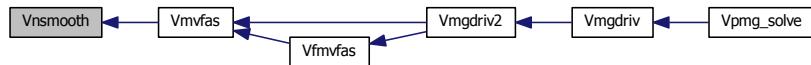
Replaces nsmooth from nsmoothd.f

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>i resid</i> |
| <i>iadjoint</i> |
| <i>meth</i> |

Definition at line 96 of file [smoothd.c](#).

Here is the caller graph for this function:



7.27.3.46 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.

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*dcos(pi/(nx-1)) c* - 2*dcos(pi/(ny-1)) c* - 2*dcos(pi/(nz-1)) c* c* lambda_max = 6 - 2*dcos((nx-2)*pi/(nx-1)) c* - 2*dcos((ny-2)*pi/(ny-1)) c* - 2*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

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.
*
*

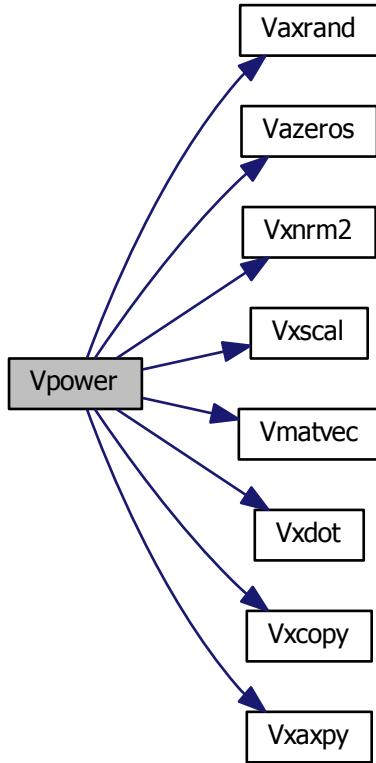
```

Parameters

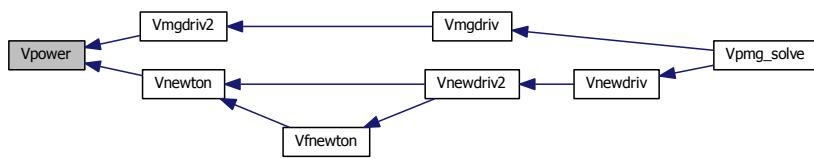
| | |
|---------------------|--|
| <i>nx</i> | |
| <i>ny</i> | |
| <i>nz</i> | |
| <i>iz</i> | |
| <i>ilev</i> | |
| <i>ipc</i> | |
| <i>rpc</i> | |
| <i>ac</i> | |
| <i>cc</i> | |
| <i>w1</i> | |
| <i>w2</i> | |
| <i>w3</i> | |
| <i>w4</i> | |
| <i>eigmax</i> | |
| <i>eigmax_model</i> | |
| <i>tol</i> | |
| <i>itmax</i> | |
| <i>iters</i> | |
| <i>iinfo</i> | |

Definition at line 52 of file [powerd.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.47 VEXTERNC void Vprtmad (int * nx, int * ny, int * nz, int * ipc, double * rpc, double * ac)

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

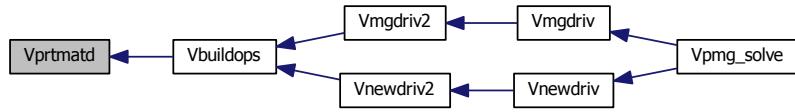
Replaces prtmtd from mikpckd.f

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 334 of file [mikpckd.c](#).

Here is the caller graph for this function:



7.27.3.48 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.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

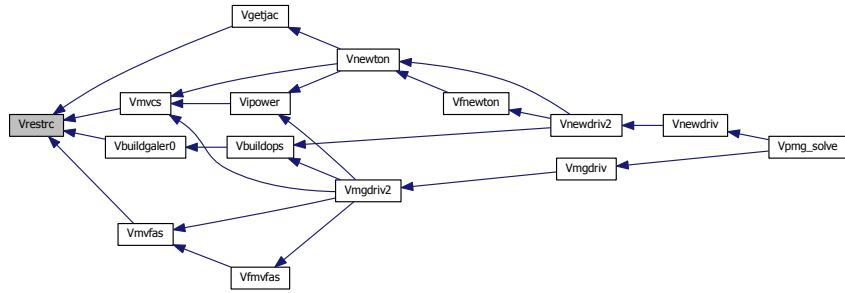
Replaces restrc from matvecd.f

Parameters

| | |
|-------------|--|
| <i>nxf</i> | |
| <i>nyf</i> | |
| <i>nzf</i> | |
| <i>nxc</i> | |
| <i>nyc</i> | |
| <i>nzc</i> | |
| <i>xin</i> | |
| <i>xout</i> | |
| <i>pc</i> | |

Definition at line 813 of file [matvecd.c](#).

Here is the caller graph for this function:



7.27.3.49 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 * *iressid*, int * *iadjoin*, int * *meth*)

Multigrid smoothing functions.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnln.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

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,
```

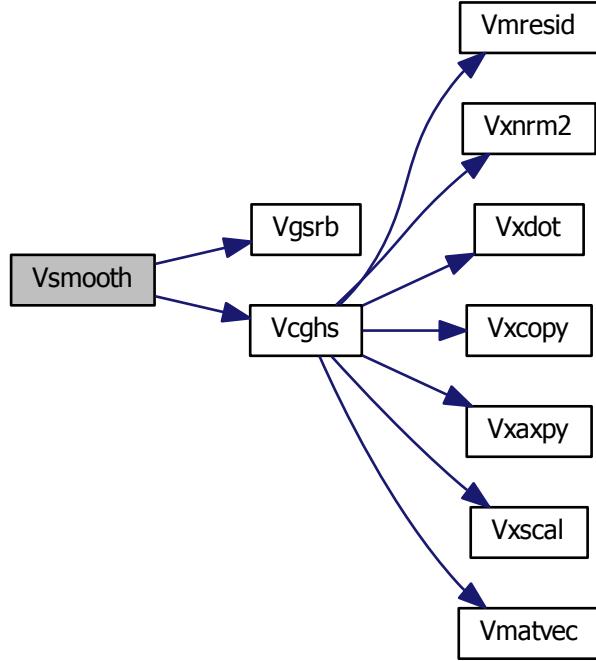
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 *

Parameters

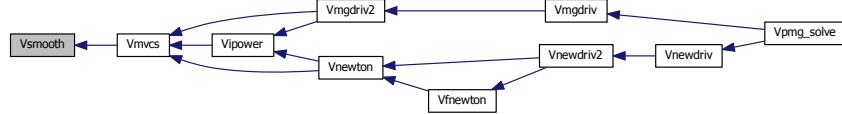
| | |
|-----------------|--|
| <i>nx</i> | |
| <i>ny</i> | |
| <i>nz</i> | |
| <i>ipc</i> | |
| <i>rpc</i> | |
| <i>ac</i> | |
| <i>cc</i> | |
| <i>fc</i> | |
| <i>x</i> | |
| <i>w1</i> | |
| <i>w2</i> | |
| <i>r</i> | |
| <i>itmax</i> | |
| <i>iters</i> | |
| <i>errtol</i> | |
| <i>omega</i> | |
| <i>i resid</i> | |
| <i>iadjoint</i> | |
| <i>meth</i> | |

Definition at line 54 of file [smoothd.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.3.50 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

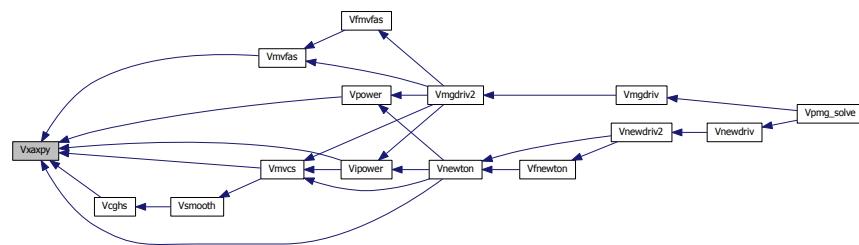
Replaces xaxpy from mikpckd.f

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](#).

Here is the caller graph for this function:



7.27.3.51 VPUBLIC 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.
*
*

```

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

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.
*
*

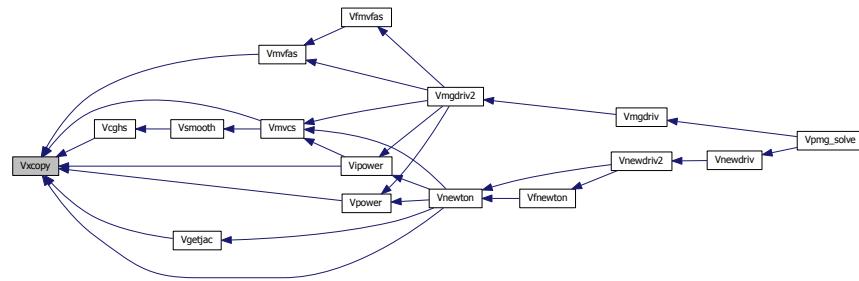
```

Parameters

| | |
|-----------|---|
| <i>nx</i> | 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](#).

Here is the caller graph for this function:



7.27.3.52 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

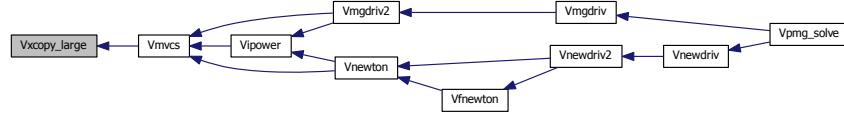
This function is exactly equivalent to calling `xcopy_small` with the matrix arguments reversed.

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](#).

Here is the caller graph for this function:



7.27.3.53 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

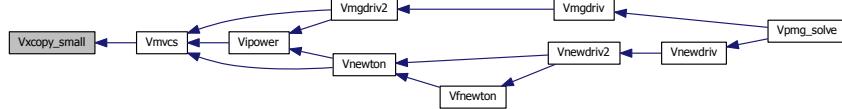
Replaces `xcopy_small` from `mikpckd.f`

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](#).

Here is the caller graph for this function:



7.27.3.54 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

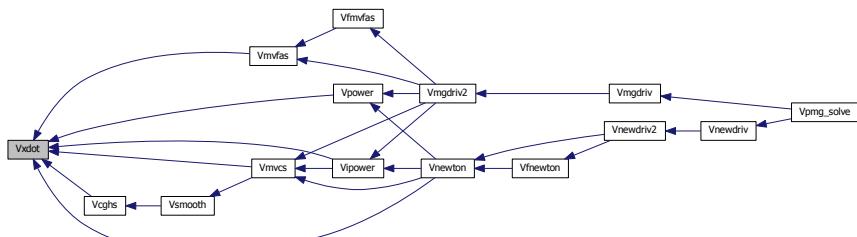
Replaces xdot from mikpckd.f

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](#).

Here is the caller graph for this function:



7.27.3.55 VEXTERNC double Vxnrm1 (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 xnrm1 from mikpckd.f

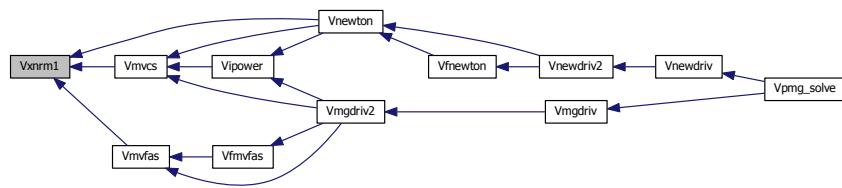
< Accumulates the calculated normal value

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](#).

Here is the caller graph for this function:



7.27.3.56 VEXTERNC double Vxnrm2 (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 xnrm2 from mikpckd.f

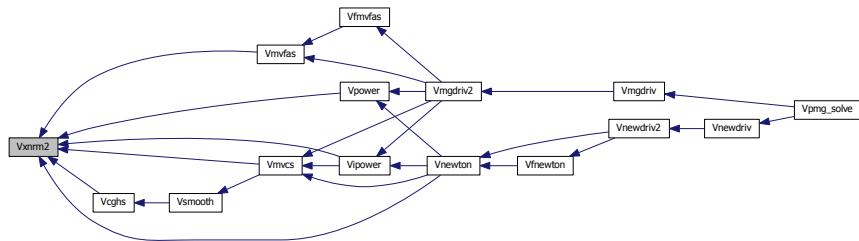
< Accumulates the calculated normal value

Parameters

| | |
|-----------------|--|
| <code>nx</code> | The size of the x dimension of the 3d matrix |
| <code>ny</code> | The size of the y dimension of the 3d matrix |
| <code>nz</code> | The size of the z dimension of the 3d matrix |
| <code>x</code> | The matrix to normalize |

Definition at line 147 of file `mikpckd.c`.

Here is the caller graph for this function:



7.27.3.57 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

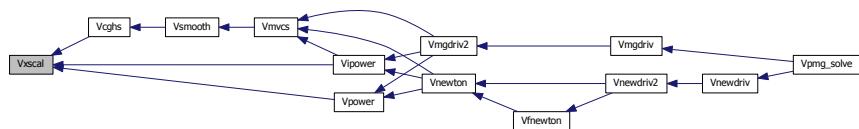
Replaces xscal from mikpckd.f

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 320 of file [mikpckd.c](#).

Here is the caller graph for this function:



Chapter 8

Data Structure Documentation

8.1 sAPOLparm Struct Reference

Parameter structure for APOL-specific variables from input files.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/generic/apbs/apolparm.h>
```

Data Fields

- int `parsed`
- double `grid` [3]
- int `setgrid`
- int `molid`
- int `setmolid`
- double `bconc`
- int `setbconc`
- double `sdens`
- int `setsdens`
- double `dpos`
- int `setdpos`
- double `press`
- int `setpress`
- `Vsurf_Meth srfm`
- int `setsrfm`
- double `srad`
- int `setsrad`
- double `swin`
- int `setswin`
- double `temp`
- int `settemp`
- double `gamma`
- int `setgamma`
- `APOLparm_calcEnergy calcenergy`
- int `setcalcenergy`
- `APOLparm_calcForce calcforce`

- int `setcalcforce`
- double `watsigma`
- double `watepsilon`
- double `sasa`
- double `sav`
- double `wcaEnergy`
- double `totForce` [3]
- int `setwat`

8.1.1 Detailed Description

Parameter structure for APOL-specific variables from input files.

Author

David Gohara

Definition at line 126 of file [apolparm.h](#).

8.1.2 Field Documentation

8.1.2.1 double bconc

Vacc sphere density

Definition at line 136 of file [apolparm.h](#).

8.1.2.2 APOLparm_calcEnergy calcenergy

Energy calculation flag

Definition at line 164 of file [apolparm.h](#).

8.1.2.3 APOLparm_calcForce calcforce

Atomic forces calculation

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

8.1.2.4 double dpos

Atom position offset

Definition at line 142 of file [apolparm.h](#).

8.1.2.5 double gamma

Surface tension for apolar energies/forces (in kJ/mol/A²)

Definition at line 160 of file [apolparm.h](#).

8.1.2.6 double grid[3]

Grid spacing

Definition at line 130 of file [apolparm.h](#).

8.1.2.7 int molid

Molecule ID to perform calculation on

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

8.1.2.8 int parsed

Flag: Has this structure been filled with anything other than the default values? (0 = no, 1 = yes)

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

8.1.2.9 double press

Solvent pressure

Definition at line 145 of file [apolparm.h](#).

8.1.2.10 double sasa

Solvent accessible surface area for this calculation

Definition at line 172 of file [apolparm.h](#).

8.1.2.11 double sav

Solvent accessible volume for this calculation

Definition at line 173 of file [apolparm.h](#).

8.1.2.12 double sdens

Vacc sphere density

Definition at line 139 of file [apolparm.h](#).

8.1.2.13 int setbconc

Flag,

See Also

[bconc](#)

Definition at line 137 of file [apolparm.h](#).

8.1.2.14 int setcalcenergy

Flag,

See Also

[calcenergy](#)

Definition at line 165 of file [apolparm.h](#).

8.1.2.15 int setcalcforce

Flag,

See Also

[calcforce](#)

Definition at line 168 of file [apolparm.h](#).

8.1.2.16 int setdpos

Flag,

See Also

[dpos](#)

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

8.1.2.17 int setgamma

Flag,

See Also

[gamma](#)

Definition at line 162 of file [apolparm.h](#).

8.1.2.18 int setgrid

Flag,

See Also

[grid](#)

Definition at line 131 of file [apolparm.h](#).

8.1.2.19 int setmolid

Flag,

See Also

[molid](#)

Definition at line 134 of file [apolparm.h](#).

8.1.2.20 int setpress

Flag,

See Also

[press](#)

Definition at line 146 of file [apolparm.h](#).

8.1.2.21 int setsdens

Flag,

See Also

[sdens](#)

Definition at line 140 of file [apolparm.h](#).

8.1.2.22 int setsrad

Flag,

See Also

[srad](#)

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

8.1.2.23 int setsrfm

Flag,

See Also

[srfm](#)

Definition at line 149 of file [apolparm.h](#).

8.1.2.24 int setswin

Flag,

See Also

[swin](#)

Definition at line 155 of file [apolparm.h](#).

8.1.2.25 int settemp

Flag,

See Also

[temp](#)

Definition at line 158 of file [apolparm.h](#).

8.1.2.26 int setwat

Boolean for determining if a water parameter is supplied. Yes = 1, No = 0

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

8.1.2.27 double srad

Solvent radius

Definition at line 151 of file [apolparm.h](#).

8.1.2.28 Vsurf_Meth srfm

Surface calculation method

Definition at line 148 of file [apolparm.h](#).

8.1.2.29 double swin

Cubic spline window

Definition at line 154 of file [apolparm.h](#).

8.1.2.30 double temp

Temperature (in K)

Definition at line 157 of file [apolparm.h](#).

8.1.2.31 double totForce[3]

Total forces on x, y, z

Definition at line 175 of file [apolparm.h](#).

8.1.2.32 double watepsilon

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

Definition at line 171 of file [apolparm.h](#).

8.1.2.33 double watsigma

Water oxygen Lennard-Jones radius (A)

Definition at line 170 of file [apolparm.h](#).

8.1.2.34 double wcaEnergy

wcaEnergy

Definition at line 174 of file [apolparm.h](#).

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

- src/generic/apbs/apolparm.h

8.2 sFEMparm Struct Reference

Parameter structure for FEM-specific variables from input files.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/trunk/src/generic/apbs/femparm.h>
```

Data Fields

- int [parsed](#)
- [FEMparm_CalcType](#) type
- int [settype](#)
- double [glen](#) [3]
- int [setglen](#)
- double [etol](#)
- int [setetol](#)
- [FEMparm_EtolType](#) ekey
- int [setekey](#)
- [FEMparm_EstType](#) akeyPRE
- int [setakeyPRE](#)
- [FEMparm_EstType](#) akeySOLVE
- int [setakeySOLVE](#)
- int [targetNum](#)

- int `settargetNum`
- double `targetRes`
- int `settargetRes`
- int `maxsolve`
- int `setmaxsolve`
- int `maxvert`
- int `setmaxvert`
- int `pkey`
- int `useMesh`
- int `meshID`

8.2.1 Detailed Description

Parameter structure for FEM-specific variables from input files.

Author

Nathan Baker

Definition at line 130 of file [femparm.h](#).

8.2.2 Field Documentation

8.2.2.1 FEMparm_EstType akeyPRE

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

Definition at line 145 of file [femparm.h](#).

8.2.2.2 FEMparm_EstType akeySOLVE

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

Definition at line 149 of file [femparm.h](#).

8.2.2.3 FEMparm_EtolType ekey

Adaptive refinement interpretation of error tolerance

Definition at line 142 of file [femparm.h](#).

8.2.2.4 double etol

Error tolerance for refinement; interpretation depends on the adaptive refinement method chosen

Definition at line 139 of file [femparm.h](#).

8.2.2.5 double glen[3]

Domain side lengths (in Å)

Definition at line 137 of file [femparm.h](#).

8.2.2.6 int maxsolve

Maximum number of solve-estimate-refine cycles

Definition at line 162 of file [femparm.h](#).

8.2.2.7 int maxvert

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

Definition at line 164 of file [femparm.h](#).

8.2.2.8 int meshID

External finite element mesh ID (if used)

Definition at line 171 of file [femparm.h](#).

8.2.2.9 int parsed

Flag: Has this structure been filled with anything other than * the default values? (0 = no, 1 = yes)

Definition at line 132 of file [femparm.h](#).

8.2.2.10 int pkey

Boolean sets the pkey type for going into AM_Refine pkey = 0 for non-HB based methods pkey = 1 for HB based methods

Definition at line 167 of file [femparm.h](#).

8.2.2.11 int setakeyPRE

Boolean

Definition at line 148 of file [femparm.h](#).

8.2.2.12 int setakeySOLVE

Boolean

Definition at line 151 of file [femparm.h](#).

8.2.2.13 int setekey

Boolean

Definition at line 144 of file [femparm.h](#).

8.2.2.14 int setetol

Boolean

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

8.2.2.15 int setglen

Boolean

Definition at line 138 of file [femparm.h](#).

8.2.2.16 int setmaxsolve

Boolean

Definition at line 163 of file [femparm.h](#).

8.2.2.17 int setmaxvert

Boolean

Definition at line 166 of file [femparm.h](#).

8.2.2.18 int settargetNum

Boolean

Definition at line 156 of file [femparm.h](#).

8.2.2.19 int settargetRes

Boolean

Definition at line 161 of file [femparm.h](#).

8.2.2.20 int settype

Boolean

Definition at line 136 of file [femparm.h](#).

8.2.2.21 int targetNum

Initial mesh will continue to be marked and refined with the method specified by akeyPRE until the mesh contains this many vertices or until targetRes is reached.

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

8.2.2.22 double targetRes

Initial mesh will continue to be marked and refined with the method specified by akeyPRE until the mesh contains no markable simplices with longest edges above this size or until targetNum is reached.

Definition at line 157 of file [femparm.h](#).

8.2.2.23 FEMparm_CalcType type

Calculation type

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

8.2.2.24 int useMesh

Indicates whether we use external finite element mesh

Definition at line 170 of file [femparm.h](#).

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

- [src/generic/apbs/femparm.h](#)

8.3 sMGparm Struct Reference

Parameter structure for MG-specific variables from input files.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/generic/apbs/mgparm.h>
```

Data Fields

- [MGparm_CalcType type](#)
- int [parsed](#)
- int [dime](#) [3]
- int [setdime](#)
- [Vchrg_Meth chgm](#)
- int [setchgm](#)
- [Vchrg_Src chgs](#)
- int [nlev](#)
- int [setnlev](#)
- double [etol](#)
- int [setetol](#)
- double [grid](#) [3]
- int [setgrid](#)
- double [glen](#) [3]
- int [setglen](#)
- [MGparm_CentMeth cmeth](#)
- double [center](#) [3]
- int [centmol](#)
- int [setgent](#)
- double [cglen](#) [3]
- int [setcglen](#)
- double [fglen](#) [3]
- int [setfglen](#)
- [MGparm_CentMeth ccmeth](#)
- double [ccenter](#) [3]
- int [ccentmol](#)

- int `setcgcent`
- MGparm_CentMeth `fcmeth`
- double `fcenter` [3]
- int `fcentmol`
- int `setfgcent`
- double `partDisjCenter` [3]
- double `partDisjLength` [3]
- int `partDisjOwnSide` [6]
- int `pdime` [3]
- int `setpdime`
- int `proc_rank`
- int `setrank`
- int `proc_size`
- int `setszie`
- double `ofrac`
- int `setofrac`
- int `async`
- int `setasync`
- int `nonlintype`
- int `setnonlintype`
- int `method`
- int `setmethod`
- int `useAqua`
- int `setUseAqua`

8.3.1 Detailed Description

Parameter structure for MG-specific variables from input files.

Author

Nathan Baker and Todd Dolinsky

Note

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

Definition at line 112 of file [mgparm.h](#).

8.3.2 Field Documentation

8.3.2.1 int `async`

Processor ID for asynchronous calculation

Definition at line 184 of file [mgparm.h](#).

8.3.2.2 double `ccenter[3]`

Coarse grid center.

Definition at line 155 of file [mgparm.h](#).

8.3.2.3 int ccentmol

Particular molecule on which we want to center the grid. This should be the appropriate index in an array of molecules, not the positive definite integer specified by the user.

Definition at line 156 of file [mgparm.h](#).

8.3.2.4 MGparm_CentMeth ccmeth

Coarse grid centering method

Definition at line 154 of file [mgparm.h](#).

8.3.2.5 double center[3]

Grid center. If ispart = 0, then this is only meaningful if cmeth = 0. However, if ispart = 1 and cmeth = MCM_PNT, then this is the center of the non-disjoint (overlapping) partition. If ispart = 1 and cmeth = MCM_MOL, then this is the vector that must be added to the center of the molecule to give the center of the non-disjoint partition.

Definition at line 136 of file [mgparm.h](#).

8.3.2.6 int centmol

Particular molecule on which we want to center the grid. This should be the appropriate index in an array of molecules, not the positive definite integer specified by the user.

Definition at line 144 of file [mgparm.h](#).

8.3.2.7 double cglen[3]

Coarse grid side lengths

Definition at line 150 of file [mgparm.h](#).

8.3.2.8 Vchrg_Meth chgm

Charge discretization method

Definition at line 120 of file [mgparm.h](#).

8.3.2.9 Vchrg_Src chgs

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

Definition at line 122 of file [mgparm.h](#).

8.3.2.10 MGparm_CentMeth cmeth

Centering method

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

8.3.2.11 int dime[3]

Grid dimensions

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

8.3.2.12 double etol

User-defined error tolerance

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

8.3.2.13 double fcenter[3]

Fine grid center.

Definition at line 161 of file [mgparm.h](#).

8.3.2.14 int fcenmol

Particular molecule on which we want to center the grid. This should be the appropriate index in an array of molecules, not the positive definite integer specified by the user.

Definition at line 162 of file [mgparm.h](#).

8.3.2.15 MGparm_CentMeth fcmeth

Fine grid centering method

Definition at line 160 of file [mgparm.h](#).

8.3.2.16 double fglen[3]

Fine grid side lengths

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

8.3.2.17 double glen[3]

Grid side lengths.

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

8.3.2.18 double grid[3]

Grid spacings

Definition at line 131 of file [mgparm.h](#).

8.3.2.19 int method

Solver Method

Definition at line 190 of file [mgparm.h](#).

8.3.2.20 int nlev

Levels in multigrid hierarchy

Deprecated Just ignored now

Definition at line 126 of file [mgparm.h](#).

8.3.2.21 int nonlintype

Linearity Type Method to be used

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

8.3.2.22 double ofrac

Overlap fraction between procs

Definition at line 182 of file [mgparm.h](#).

8.3.2.23 int parsed

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

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

8.3.2.24 double partDisjCenter[3]

This gives the center of the disjoint partitions

Definition at line 169 of file [mgparm.h](#).

8.3.2.25 double partDisjLength[3]

This gives the lengths of the disjoint partitions

Definition at line 171 of file [mgparm.h](#).

8.3.2.26 int partDisjOwnSide[6]

Tells whether the boundary points are ours (1) or not (0)

Definition at line 173 of file [mgparm.h](#).

8.3.2.27 int pdime[3]

Grid of processors to be used in calculation

Definition at line 176 of file [mgparm.h](#).

8.3.2.28 int proc_rank

Rank of this processor

Definition at line 178 of file [mgparm.h](#).

8.3.2.29 int proc_size

Total number of processors

Definition at line 180 of file [mgparm.h](#).

8.3.2.30 int setasync

Flag,

See Also

[asynch](#)

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

8.3.2.31 int setcgcent

Flag,

See Also

[ccmeth](#)

Definition at line 159 of file [mgparm.h](#).

8.3.2.32 int setcglen

Flag,

See Also

[crlen](#)

Definition at line 151 of file [mgparm.h](#).

8.3.2.33 int setchgm

Flag,

See Also

[chgm](#)

Definition at line 121 of file [mgparm.h](#).

8.3.2.34 int setdime

Flag,

See Also

[dime](#)

Definition at line 119 of file [mgparm.h](#).

8.3.2.35 int setetol

Flag,

See Also

[etol](#)

Definition at line 130 of file [mgparm.h](#).

8.3.2.36 int setfgcent

Flag,

See Also

[fcmeth](#)

Definition at line 165 of file [mgparm.h](#).

8.3.2.37 int setfglen

Flag,

See Also

[fglen](#)

Definition at line 153 of file [mgparm.h](#).

8.3.2.38 int setgcent

Flag,

See Also

[cmeth](#)

Definition at line 147 of file [mgparm.h](#).

8.3.2.39 int setglen

Flag,

See Also

[glen](#)

Definition at line [134](#) of file [mgparm.h](#).

8.3.2.40 int setgrid

Flag,

See Also

[grid](#)

Definition at line [132](#) of file [mgparm.h](#).

8.3.2.41 int setmethod

Flag,

See Also

[method](#)

Definition at line [191](#) of file [mgparm.h](#).

8.3.2.42 int setnlev

Flag,

See Also

[nlev](#)

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

8.3.2.43 int setnonlintype

Flag,

See Also

[nonlintype](#)

Definition at line [188](#) of file [mgparm.h](#).

8.3.2.44 int setofrac

Flag,

See Also

[ofrac](#)

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

8.3.2.45 int setpdime

Flag,

See Also

[pdime](#)

Definition at line 177 of file [mgparm.h](#).

8.3.2.46 int setrank

Flag,

See Also

[proc_rank](#)

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

8.3.2.47 int setsize

Flag,

See Also

[proc_size](#)

Definition at line 181 of file [mgparm.h](#).

8.3.2.48 int setUseAqua

Flag,

See Also

[useAqua](#)

Definition at line 194 of file [mgparm.h](#).

8.3.2.49 MGparm_CalcType type

What type of MG calculation?

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

8.3.2.50 int useAqua

Enable use of lpbe/aqua

Definition at line 193 of file [mgparm.h](#).

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

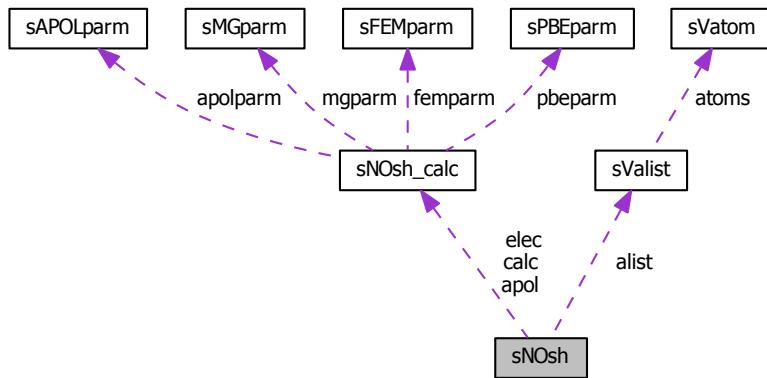
- src/generic/apbs/[mgparm.h](#)

8.4 sNOsh Struct Reference

Class for parsing fixed format input files.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-S/trunk/src/generic/apbs/nosh.h>
```

Collaboration diagram for sNOsh:



Data Fields

- `NOsh_calc * calc [NOSH_MAXCALC]`
- `int ncalc`
- `NOsh_calc * elec [NOSH_MAXCALC]`
- `int nelec`
- `NOsh_calc * apol [NOSH_MAXCALC]`
- `int napol`
- `int ispara`

- int `proc_rank`
- int `proc_size`
- int `bogus`
- int `elec2calc` [NOSH_MAXCALC]
- int `apol2calc` [NOSH_MAXCALC]
- int `nmol`
- char `molpath` [NOSH_MAXMOL][VMAX_ARGLEN]
- NOsh_MolFormat `molfmt` [NOSH_MAXMOL]
- Valist * `alist` [NOSH_MAXMOL]
- int `gotparm`
- char `parmpath` [VMAX_ARGLEN]
- NOsh_ParmFormat `parmfmt`
- int `ndiel`
- char `dielXpath` [NOSH_MAXMOL][VMAX_ARGLEN]
- char `dielYpath` [NOSH_MAXMOL][VMAX_ARGLEN]
- char `dielZpath` [NOSH_MAXMOL][VMAX_ARGLEN]
- Vdata_Format `dielfmt` [NOSH_MAXMOL]
- int `nkappa`
- char `kappapath` [NOSH_MAXMOL][VMAX_ARGLEN]
- Vdata_Format `kappafmt` [NOSH_MAXMOL]
- int `npot`
- char `potpath` [NOSH_MAXMOL][VMAX_ARGLEN]
- Vdata_Format `potfmt` [NOSH_MAXMOL]
- int `ncharge`
- char `chargepath` [NOSH_MAXMOL][VMAX_ARGLEN]
- Vdata_Format `chargefmt` [NOSH_MAXMOL]
- int `nmesh`
- char `meshpath` [NOSH_MAXMOL][VMAX_ARGLEN]
- Vdata_Format `meshfmt` [NOSH_MAXMOL]
- int `nprint`
- NOsh_PrintType `printwhat` [NOSH_MAXPRINT]
- int `printnarg` [NOSH_MAXPRINT]
- int `printcalc` [NOSH_MAXPRINT][NOSH_MAXPOP]
- int `printop` [NOSH_MAXPRINT][NOSH_MAXPOP]
- int `parsed`
- char `elecname` [NOSH_MAXCALC][VMAX_ARGLEN]
- char `apolname` [NOSH_MAXCALC][VMAX_ARGLEN]

8.4.1 Detailed Description

Class for parsing fixed format input files.

Author

Nathan Baker

Definition at line 182 of file `nosh.h`.

8.4.2 Field Documentation

8.4.2.1 Valist* alist[NOSH_MAXMOL]

Molecules for calculation (can be used in setting mesh centers

Definition at line [221](#) of file [nosh.h](#).

8.4.2.2 NOsh_calc* apol[NOSH_MAXCALC]

The array of calculation objects corresponding to APOLAR statements read in the input file. Compare to [sNOsh::calc](#)

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

8.4.2.3 int apol2calc[NOSH_MAXCALC]

(see elec2calc)

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

8.4.2.4 char apolname[NOSH_MAXCALC][VMAX_ARGLEN]

Optional user-specified name for APOLAR statement

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

8.4.2.5 int bogus

A flag which tells routines using NOsh that this particular NOsh is broken – useful for parallel focusing calculations where the user gave us too many processors (1 => ignore this NOsh; 0 => this NOsh is OK)

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

8.4.2.6 NOsh_calc* calc[NOSH_MAXCALC]

The array of calculation objects corresponding to actual calculations performed by the code. Compare to [sNOsh::elec](#)

Definition at line [184](#) of file [nosh.h](#).

8.4.2.7 Vdata_Format chargefmt[NOSH_MAXMOL]

Charge maps fileformats

Definition at line [242](#) of file [nosh.h](#).

8.4.2.8 char chargepath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to charge map files

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

8.4.2.9 Vdata_Format dielfmt[NOSH_MAXMOL]

Dielectric maps file formats

Definition at line 233 of file [nosh.h](#).

8.4.2.10 char dielXpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to x-shifted dielectric map files

Definition at line 227 of file [nosh.h](#).

8.4.2.11 char dielYpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to y-shifted dielectric map files

Definition at line 229 of file [nosh.h](#).

8.4.2.12 char dielZpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to z-shifted dielectric map files

Definition at line 231 of file [nosh.h](#).

8.4.2.13 NOsh_calc* elec[NOSH_MAXCALC]

The array of calculation objects corresponding to ELEC statements read in the input file. Compare to [sNOsh::calc](#)

Definition at line 189 of file [nosh.h](#).

8.4.2.14 int elec2calc[NOSH_MAXCALC]

A mapping between ELEC statements which appear in the input file and calc objects stored above. Since we allow both normal and focused multigrid, there isn't a 1-to-1 correspondence between ELEC statements and actual calcuations. This can really confuse operations which work on specific calculations further down the road (like PRINT). Therefore this array is the initial point of entry for any calculation-specific operation. It points to a specific entry in the calc array.

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

8.4.2.15 char elecname[NOSH_MAXCALC][VMAX_ARGLEN]

Optional user-specified name for ELEC statement

Definition at line 254 of file [nosh.h](#).

8.4.2.16 int gotparm

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

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

8.4.2.17 int ispara

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

Definition at line 201 of file [nosh.h](#).

8.4.2.18 Vdata_Format kappafmt[NOSH_MAXMOL]

Kappa maps file formats

Definition at line 236 of file [nosh.h](#).

8.4.2.19 char kappapath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to kappa map files

Definition at line 235 of file [nosh.h](#).

8.4.2.20 Vdata_Format meshfmt[NOSH_MAXMOL]

Mesh fileformats

Definition at line 245 of file [nosh.h](#).

8.4.2.21 char meshpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to mesh files

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

8.4.2.22 NOsh_MolFormat molfmt[NOSH_MAXMOL]

Mol files formats

Definition at line 220 of file [nosh.h](#).

8.4.2.23 char molpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to mol files

Definition at line 219 of file [nosh.h](#).

8.4.2.24 int napol

The number of apolar statements in the input file and in the apolar array

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

8.4.2.25 int ncalc

The number of calculations in the calc array

Definition at line 187 of file [nosh.h](#).

8.4.2.26 int ncharge

Number of charge maps

Definition at line [240](#) of file [nosh.h](#).

8.4.2.27 int ndiel

Number of dielectric maps

Definition at line [226](#) of file [nosh.h](#).

8.4.2.28 int nelec

The number of elec statements in the input file and in the elec array

Definition at line [192](#) of file [nosh.h](#).

8.4.2.29 int nkappa

Number of kappa maps

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

8.4.2.30 int nmesh

Number of meshes

Definition at line [243](#) of file [nosh.h](#).

8.4.2.31 int nmol

Number of molecules

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

8.4.2.32 int npot

Number of potential maps

Definition at line [237](#) of file [nosh.h](#).

8.4.2.33 int nprint

How many print sections?

Definition at line [246](#) of file [nosh.h](#).

8.4.2.34 NOsh_ParmFormat parmfmt

Parm file format

Definition at line [225](#) of file [nosh.h](#).

8.4.2.35 char parmpath[VMAX_ARGLEN]

Paths to parm file

Definition at line 224 of file [nosh.h](#).

8.4.2.36 int parsed

Have we parsed an input file yet?

Definition at line 253 of file [nosh.h](#).

8.4.2.37 Vdata_Format potfmt[NOSH_MAXMOL]

Potential maps file formats

Definition at line 239 of file [nosh.h](#).

8.4.2.38 char potpath[NOSH_MAXMOL][VMAX_ARGLEN]

Paths to potential map files

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

8.4.2.39 int printcalc[NOSH_MAXPRINT][NOSH_MAXPOP]

ELEC id (see elec2calc)

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

8.4.2.40 int printnarg[NOSH_MAXPRINT]

How many arguments in energy list

Definition at line 249 of file [nosh.h](#).

8.4.2.41 int printop[NOSH_MAXPRINT][NOSH_MAXPOP]

Operation id (0 = add, 1 = subtract)

Definition at line 251 of file [nosh.h](#).

8.4.2.42 NOsh_PrintType printwhat[NOSH_MAXPRINT]

What do we print:

- 0 = energy,
- 1 = force

Definition at line 247 of file [nosh.h](#).

8.4.2.43 int proc_rank

Processor rank in parallel calculation

Definition at line 202 of file [nosh.h](#).

8.4.2.44 int proc_size

Number of processors in parallel calculation

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

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

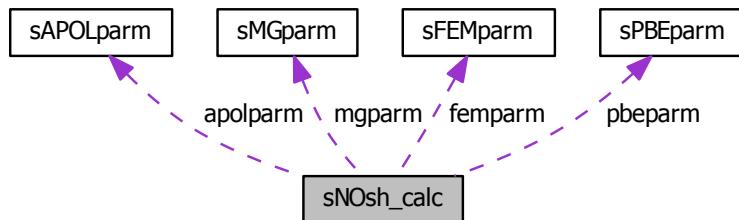
- src/generic/apbs/[nosh.h](#)

8.5 sNOsh_calc Struct Reference

Calculation class for use when parsing fixed format input files.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/generic/apbs/nosh.h>
```

Collaboration diagram for sNOsh_calc:



Data Fields

- `MGparm * mgparm`
- `FEMparm * femparm`
- `PBEParm * pbeparm`
- `APOLparm * apolparm`
- `NOsh_CalcType calctype`

8.5.1 Detailed Description

Calculation class for use when parsing fixed format input files.

Author

Nathan Baker

Definition at line 163 of file [nosh.h](#).

8.5.2 Field Documentation

8.5.2.1 APOLparm* apolparm

Non-polar parameters

Definition at line 167 of file [nosh.h](#).

8.5.2.2 NOsh_CalcType calctype

Calculation type

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

8.5.2.3 FEMparm* femparm

Finite element parameters

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

8.5.2.4 MGparm* mgparm

Multigrid parameters

Definition at line 164 of file [nosh.h](#).

8.5.2.5 PBEParm* pbeparm

Generic PBE parameters

Definition at line 166 of file [nosh.h](#).

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

- [src/generic/apbs/nosh.h](#)

8.6 sPBEParm Struct Reference

Parameter structure for PBE variables from input files.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/trunk/src/generic/apbs/pbeparm.h>
```

Data Fields

- int [molid](#)

- int `setmolid`
- int `useDielMap`
- int `dielMapID`
- int `useKappaMap`
- int `kappaMapID`
- int `usePotMap`
- int `potMapID`
- int `useChargeMap`
- int `chargeMapID`
- `Vhal_PBEType pbetype`
- int `setpbetype`
- `Vbcfl bcfl`
- int `setbcfl`
- int `nion`
- int `setnion`
- double `ionq [MAXION]`
- double `ionc [MAXION]`
- double `ionr [MAXION]`
- int `setion [MAXION]`
- double `pdie`
- int `setpdie`
- double `sdens`
- int `setsdens`
- double `sdie`
- int `setsdie`
- `Vsurf_Meth srfm`
- int `setsrfm`
- double `srad`
- int `setsrad`
- double `swin`
- int `setswin`
- double `temp`
- int `settemp`
- double `smsize`
- int `setsmsize`
- double `smvolume`
- int `setsmvolume`
- `PBEParm_calcEnergy calcenergy`
- int `setcalcenergy`
- `PBEParm_calcForce calcforce`
- int `setcalcforce`
- double `zmem`
- int `setzmem`
- double `Lmem`
- int `setLmem`
- double `mdie`
- int `setmdie`
- double `memv`
- int `setmemv`
- int `numwrite`
- char `writestem [PBEPARM_MAXWRITE][VMAX_ARGLEN]`

- `Vdata_Type writetype [PBEPARM_MAXWRITE]`
- `Vdata_Format writefmt [PBEPARM_MAXWRITE]`
- int `writemat`
- int `setwritemat`
- char `writematstem [VMAX_ARGLEN]`
- int `writematflag`
- int `parsed`

8.6.1 Detailed Description

Parameter structure for PBE variables from input files.

Author

Nathan Baker

Note

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

Definition at line [116](#) of file `pbeparm.h`.

8.6.2 Field Documentation

8.6.2.1 Vbcfl bcfl

Boundary condition method

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

8.6.2.2 PBEParm_calcEnergy calcenergy

Energy calculation flag

Definition at line [164](#) of file `pbeparm.h`.

8.6.2.3 PBEParm_calcForce calcforce

Atomic forces calculation

Definition at line [166](#) of file `pbeparm.h`.

8.6.2.4 int chargeMapID

Charge distribution map ID (if used)

Definition at line [132](#) of file `pbeparm.h`.

8.6.2.5 int dielMapID

Dielectric map ID (if used)

Definition at line 122 of file [pbeparm.h](#).

8.6.2.6 double ionc[MAXION]

Counterion concentrations (in M)

Definition at line 140 of file [pbeparm.h](#).

8.6.2.7 double ionq[MAXION]

Counterion charges (in e)

Definition at line 139 of file [pbeparm.h](#).

8.6.2.8 double ionr[MAXION]

Counterion radii (in A)

Definition at line 141 of file [pbeparm.h](#).

8.6.2.9 int kappaMapID

Kappa map ID (if used)

Definition at line 125 of file [pbeparm.h](#).

8.6.2.10 double Lmem

membrane width

Definition at line 175 of file [pbeparm.h](#).

8.6.2.11 double mdie

membrane dielectric constant

Definition at line 177 of file [pbeparm.h](#).

8.6.2.12 double memv

Membrane potential

Definition at line 179 of file [pbeparm.h](#).

8.6.2.13 int molid

Molecule ID to perform calculation on

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

8.6.2.14 int nion

Number of counterion species

Definition at line 137 of file [pbeparm.h](#).

8.6.2.15 int numwrite

Number of write statements encountered

Definition at line 184 of file [pbeparm.h](#).

8.6.2.16 int parsed

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

Definition at line 200 of file [pbeparm.h](#).

8.6.2.17 Vhal_PBEType pbetype

Which version of the PBE are we solving?

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

8.6.2.18 double pdie

Solute dielectric

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

8.6.2.19 int potMapID

Kappa map ID (if used)

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

8.6.2.20 double sdens

Vacc sphere density

Definition at line 145 of file [pbeparm.h](#).

8.6.2.21 double sdie

Solvent dielectric

Definition at line 147 of file [pbeparm.h](#).

8.6.2.22 int setbcfl

Flag,

See Also

[bcfl](#)

Definition at line 136 of file [pbeparm.h](#).

8.6.2.23 int setcalcenergy

Flag,

See Also

[calcenergy](#)

Definition at line 165 of file [pbeparm.h](#).

8.6.2.24 int setcalcforce

Flag,

See Also

[calcforce](#)

Definition at line 167 of file [pbeparm.h](#).

8.6.2.25 int setion[MAXION]

Flag,

See Also

[ionq](#)

Definition at line 142 of file [pbeparm.h](#).

8.6.2.26 int setLmem

Flag

Definition at line 176 of file [pbeparm.h](#).

8.6.2.27 int setmdie

Flag

Definition at line 178 of file [pbeparm.h](#).

8.6.2.28 int setmemv

Flag

Definition at line 180 of file [pbeparm.h](#).

8.6.2.29 int setmolid

Flag,

See Also

[molid](#)

Definition at line 119 of file [pbeparm.h](#).

8.6.2.30 int setnion

Flag,

See Also

[nion](#)

Definition at line 138 of file [pbeparm.h](#).

8.6.2.31 int setpbetype

Flag,

See Also

[pbetype](#)

Definition at line 134 of file [pbeparm.h](#).

8.6.2.32 int setpdie

Flag,

See Also

[pdie](#)

Definition at line 144 of file [pbeparm.h](#).

8.6.2.33 int setsdens

Flag,

See Also

[sdens](#)

Definition at line 146 of file [pbeparm.h](#).

8.6.2.34 int setsdie

Flag,

See Also

[sdie](#)

Definition at line 148 of file [pbeparm.h](#).

8.6.2.35 int setsmsize

Flag,

See Also

[temp](#)

Definition at line 159 of file [pbeparm.h](#).

8.6.2.36 int setsmvolume

Flag,

See Also

[temp](#)

Definition at line 162 of file [pbeparm.h](#).

8.6.2.37 int setsrad

Flag,

See Also

[srad](#)

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

8.6.2.38 int setsrfm

Flag,

See Also

[srfm](#)

Definition at line 150 of file [pbeparm.h](#).

8.6.2.39 int setswin

Flag,

See Also

[swin](#)

Definition at line 154 of file [pbeparm.h](#).

8.6.2.40 int settemp

Flag,

See Also

[temp](#)

Definition at line 156 of file [pbeparm.h](#).

8.6.2.41 int setwritemat

Flag,

See Also

[writemat](#)

Definition at line 193 of file [pbeparm.h](#).

8.6.2.42 int setzmem

Flag

Definition at line 174 of file [pbeparm.h](#).

8.6.2.43 double smsize

SMPBE size

Definition at line 158 of file [pbeparm.h](#).

8.6.2.44 double smvolume

SMPBE size

Definition at line 161 of file [pbeparm.h](#).

8.6.2.45 double srad

Solvent radius

Definition at line 151 of file [pbeparm.h](#).

8.6.2.46 Vsurf_Meth srfm

Surface calculation method

Definition at line 149 of file [pbeparm.h](#).

8.6.2.47 double swin

Cubic spline window

Definition at line 153 of file [pbeparm.h](#).

8.6.2.48 double temp

Temperature (in K)

Definition at line 155 of file [pbeparm.h](#).

8.6.2.49 int useChargeMap

Indicates whether we use an external charge distribution map

Definition at line 130 of file [pbeparm.h](#).

8.6.2.50 int useDielMap

Indicates whether we use external dielectric maps (note plural)

Definition at line 120 of file [pbeparm.h](#).

8.6.2.51 int useKappaMap

Indicates whether we use an external kappa map

Definition at line 123 of file [pbeparm.h](#).

8.6.2.52 int usePotMap

Indicates whether we use an external kappa map

Definition at line 126 of file [pbeparm.h](#).

8.6.2.53 Vdata_Format writefmt[PBEPARM_MAXWRITE]

File format to write data in

Definition at line 188 of file [pbeparm.h](#).

8.6.2.54 int writemat

Write out the operator matrix?

- 0 => no
- 1 => yes

Definition at line 190 of file [pbeparm.h](#).

8.6.2.55 int writematflag

What matrix should we write:

- 0 => Poisson (differential operator)
- 1 => Poisson-Boltzmann operator linearized around solution (if applicable)

Definition at line 195 of file [pbeparm.h](#).

8.6.2.56 char writematstem[VMAX_ARGLEN]

File stem to write mat

Definition at line 194 of file [pbeparm.h](#).

8.6.2.57 char writestem[PBEPARM_MAXWRITE][VMAX_ARGLEN]

File stem to write data to

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

8.6.2.58 Vdata_Type writetype[PBEPARM_MAXWRITE]

What data to write

Definition at line 187 of file [pbeparm.h](#).

8.6.2.59 double zmem

z value of membrane bottom

Definition at line 173 of file [pbeparm.h](#).

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

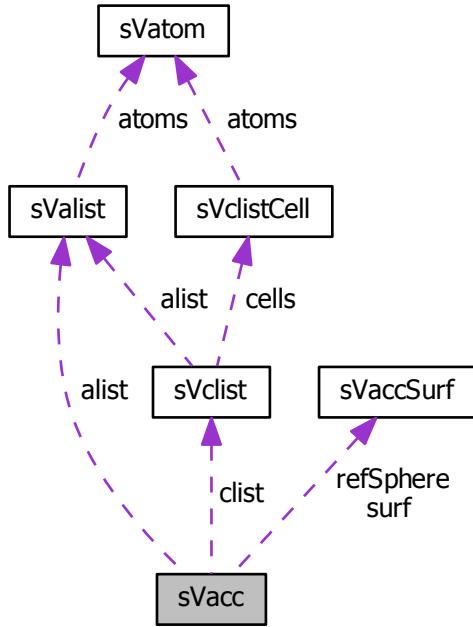
- [src/generic/apbs/pbeparm.h](#)

8.7 sVacc Struct Reference

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

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/generic/apbs/vacc.h>
```

Collaboration diagram for sVacc:



Data Fields

- Vmem * **mem**
- Valist * **alist**
- Vclist * **clist**
- int * **atomFlags**
- VaccSurf * **refSphere**
- VaccSurf ** **surf**
- Vset **acc**
- double **surf_density**

8.7.1 Detailed Description

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

Author

Nathan Baker

Definition at line 105 of file [vacc.h](#).

8.7.2 Field Documentation

8.7.2.1 Vset acc

An integer array (to be treated as bitfields) of Vset type with length equal to the number of vertices in the mesh

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

8.7.2.2 Valist* alist

Valist structure for list of atoms

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

8.7.2.3 int* atomFlags

Array of boolean flags of length Valist_getNumberAtoms(thee->alist) to prevent double-counting atoms during calculations

Definition at line 110 of file [vacc.h](#).

8.7.2.4 Vclist* clist

Vclist structure for atom cell list

Definition at line 109 of file [vacc.h](#).

8.7.2.5 Vmem* mem

Memory management object for this class

Definition at line 107 of file [vacc.h](#).

8.7.2.6 VaccSurf* refSphere

Reference sphere for SASA calculations

Definition at line 113 of file [vacc.h](#).

8.7.2.7 VaccSurf** surf

Array of surface points for each atom; is not initialized until needed (test against VNULL to determine initialization state)

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

8.7.2.8 double surf_density

Minimum solvent accessible surface point density (in pts/A²)

Definition at line 119 of file [vacc.h](#).

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

- src/generic/apbs/[vacc.h](#)

8.8 sVaccSurf Struct Reference

Surface object list of per-atom surface points.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/generic/apbs/vacc.h>
```

Data Fields

- Vmem * [mem](#)
- double * [xpts](#)
- double * [ypts](#)
- double * [zpts](#)
- char * [bpts](#)
- double [area](#)
- int [npts](#)
- double [probe_radius](#)

8.8.1 Detailed Description

Surface object list of per-atom surface points.

Author

Nathan Baker

Definition at line [81](#) of file [vacc.h](#).

8.8.2 Field Documentation

8.8.2.1 double area

Area spanned by these points

Definition at line [88](#) of file [vacc.h](#).

8.8.2.2 char* bpts

Array of booleans indicating whether a point is (1) or is not (0) part of the surface

Definition at line [86](#) of file [vacc.h](#).

8.8.2.3 Vmem* mem

Memory object

Definition at line [82](#) of file [vacc.h](#).

8.8.2.4 int npts

Length of thee->xpts, ypts, zpts arrays

Definition at line [89](#) of file [vacc.h](#).

8.8.2.5 double probe_radius

Probe radius (A) with which this surface was constructed

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

8.8.2.6 double* xpts

Array of point x-locations

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

8.8.2.7 double* ypts

Array of point y-locations

Definition at line 84 of file [vacc.h](#).

8.8.2.8 double* zpts

Array of point z-locations

Definition at line 85 of file [vacc.h](#).

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

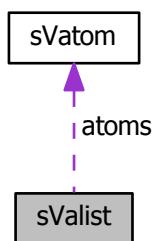
- [src/generic/apbs/vacc.h](#)

8.9 sValist Struct Reference

Container class for list of atom objects.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/generic/apbs/valist.h>
```

Collaboration diagram for sValist:



Data Fields

- int `number`
- double `center` [3]
- double `mincrd` [3]
- double `maxcrd` [3]
- double `maxrad`
- double `charge`
- `Vatom *` `atoms`
- `Vmem *` `vmem`

8.9.1 Detailed Description

Container class for list of atom objects.

Author

Nathan Baker

Definition at line [78](#) of file `valist.h`.

8.9.2 Field Documentation

8.9.2.1 `Vatom* atoms`

Atom list

Definition at line [86](#) of file `valist.h`.

8.9.2.2 `double center[3]`

Molecule center ($x_{\min} - x_{\max}/2$, etc.

Definition at line [81](#) of file `valist.h`.

8.9.2.3 `double charge`

Net charge

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

8.9.2.4 `double maxcrd[3]`

Maximum coordinates

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

8.9.2.5 `double maxrad`

Maximum radius

Definition at line [84](#) of file `valist.h`.

8.9.2.6 double mincrd[3]

Minimum coordinates

Definition at line 82 of file [valist.h](#).

8.9.2.7 int number

Number of atoms in list

Definition at line 80 of file [valist.h](#).

8.9.2.8 Vmem* vmem

Memory management object

Definition at line 87 of file [valist.h](#).

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

- src/generic/apbs/[valist.h](#)

8.10 sVatom Struct Reference

Contains public data members for Vatom class/module.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/generic/apbs/vatom.h>
```

Data Fields

- double [position](#) [3]
- double [radius](#)
- double [charge](#)
- double [partID](#)
- double [epsilon](#)
- int [id](#)
- char [resName](#) [[VMAX_RECLEN](#)]
- char [atomName](#) [[VMAX_RECLEN](#)]

8.10.1 Detailed Description

Contains public data members for Vatom class/module.

Author

Nathan Baker, David Gohara, Mike Schneiders

Definition at line 81 of file [vatom.h](#).

8.10.2 Field Documentation

8.10.2.1 char atomName[VMAX_RECLEN]

Atom name from PDB/PDR file

Definition at line [95](#) of file [vatom.h](#).

8.10.2.2 double charge

Atomic charge

Definition at line [85](#) of file [vatom.h](#).

8.10.2.3 double epsilon

Epsilon value for WCA calculations

Definition at line [88](#) of file [vatom.h](#).

8.10.2.4 int id

Atomic ID; this should be a unique non-negative integer assigned based on the index of the atom in a Valist atom array

Definition at line [90](#) of file [vatom.h](#).

8.10.2.5 double partID

Partition value for assigning atoms to particular processors and/or partitions

Definition at line [86](#) of file [vatom.h](#).

8.10.2.6 double position[3]

Atomic position

Definition at line [83](#) of file [vatom.h](#).

8.10.2.7 double radius

Atomic radius

Definition at line [84](#) of file [vatom.h](#).

8.10.2.8 char resName[VMAX_RECLEN]

Residue name from PDB/PQR file

Definition at line [94](#) of file [vatom.h](#).

The documentation for this struct was generated from the following file:

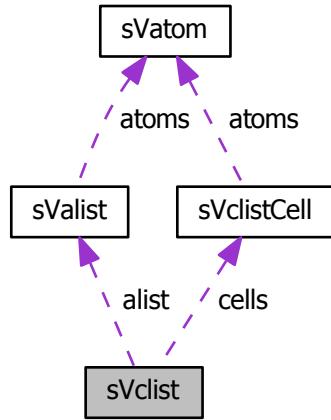
- src/generic/apbs/[vatom.h](#)

8.11 sVclist Struct Reference

Atom cell list.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/generic/apbs/vclist.h>
```

Collaboration diagram for sVclist:



Data Fields

- Vmem * vmem
- Valist * alist
- Vclist_DomainMode mode
- int npts [VAPBS_DIM]
- int n
- double max_radius
- VclistCell * cells
- double lower_corner [VAPBS_DIM]
- double upper_corner [VAPBS_DIM]
- double spacs [VAPBS_DIM]

8.11.1 Detailed Description

Atom cell list.

Author

Nathan Baker

Definition at line 114 of file [vclist.h](#).

8.11.2 Field Documentation

8.11.2.1 Valist* alist

Original Valist structure for list of atoms

Definition at line 117 of file [vclist.h](#).

8.11.2.2 VclistCell* cells

Cell array of length $\text{thee} \rightarrow n$

Definition at line 122 of file [vclist.h](#).

8.11.2.3 double lower_corner[VAPBS_DIM]

Hash table grid corner

Definition at line 123 of file [vclist.h](#).

8.11.2.4 double max_radius

Maximum probe radius

Definition at line 121 of file [vclist.h](#).

8.11.2.5 Vclist_DomainMode mode

How the cell list was constructed

Definition at line 118 of file [vclist.h](#).

8.11.2.6 int n

$n = nx * nz * ny$

Definition at line 120 of file [vclist.h](#).

8.11.2.7 int npts[VAPBS_DIM]

Hash table grid dimensions

Definition at line 119 of file [vclist.h](#).

8.11.2.8 double spacs[VAPBS_DIM]

Hash table grid spacings

Definition at line 125 of file [vclist.h](#).

8.11.2.9 double upper_corner[VAPBS_DIM]

Hash table grid corner

Definition at line 124 of file [vclist.h](#).

8.11.2.10 Vmem* vmem

Memory management object for this class

Definition at line 116 of file [vclist.h](#).

The documentation for this struct was generated from the following file:

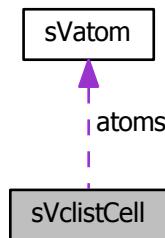
- [src/generic/apbs/vclist.h](#)

8.12 sVclistCell Struct Reference

Atom cell list cell.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/generic/apbs/vclist.h>
```

Collaboration diagram for sVclistCell:



Data Fields

- [Vatom ** atoms](#)
- int [natoms](#)

8.12.1 Detailed Description

Atom cell list cell.

Author

Nathan Baker

Definition at line 98 of file [vclist.h](#).

8.12.2 Field Documentation

8.12.2.1 Vatom** atoms

Array of atom objects associated with this cell

Definition at line 99 of file [vclist.h](#).

8.12.2.2 int natoms

Length of thee->atoms array

Definition at line 100 of file [vclist.h](#).

The documentation for this struct was generated from the following file:

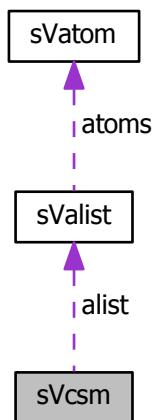
- [src/generic/apbs/vclist.h](#)

8.13 sVcsm Struct Reference

Charge-simplex map class.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/fem/apbs/vcsm.h>
```

Collaboration diagram for sVcsm:



Data Fields

- `Valist * alist`
- `int natom`
- `Gem * gm`
- `int ** sqm`
- `int * nsqm`
- `int nsimp`
- `int msimp`
- `int ** qsm`
- `int * nqsm`
- `int initFlag`
- `Vmem * vmem`

8.13.1 Detailed Description

Charge-simplex map class.

Author

Nathan Baker

Definition at line 89 of file [vcsm.h](#).

8.13.2 Field Documentation

8.13.2.1 `Valist* alist`

Atom (charge) list

Definition at line 91 of file [vcsm.h](#).

8.13.2.2 `Gem* gm`

Grid manager (container class for master vertex and simplex lists as well as prolongation operator for updating after refinement)

Definition at line 94 of file [vcsm.h](#).

8.13.2.3 `int initFlag`

Indicates whether the maps have been initialized yet

Definition at line 112 of file [vcsm.h](#).

8.13.2.4 `int msimp`

The maximum number of entries that can be accomodated by sqm or nsqm – saves on realloc's

Definition at line 107 of file [vcsm.h](#).

8.13.2.5 int natom

Size of thee->alist; redundant, but useful for convenience

Definition at line 92 of file [vcsm.h](#).

8.13.2.6 int* nqsm

The length of the simplex lists in thee->qsm

Definition at line 111 of file [vcsm.h](#).

8.13.2.7 int nsimp

The _currently used) length of sqm, nsqm – may not always be up-to-date with Gem

Definition at line 105 of file [vcsm.h](#).

8.13.2.8 int* nsqm

The length of the charge lists in thee->sqm

Definition at line 104 of file [vcsm.h](#).

8.13.2.9 int qsm**

The inverse of sqm; the list of simplices associated with a given charge

Definition at line 109 of file [vcsm.h](#).

8.13.2.10 int sqm**

The map which gives the list charges associated with each simplex in gm->simplices. The indices of the first dimension are associated with the simplex ID's in Vgm. Each charge list (second dimension) contains entries corresponding to indicies in thee->alist with lengths given in thee->nsqm

Definition at line 97 of file [vcsm.h](#).

8.13.2.11 Vmem* vmem

Memory management object

Definition at line 114 of file [vcsm.h](#).

The documentation for this struct was generated from the following file:

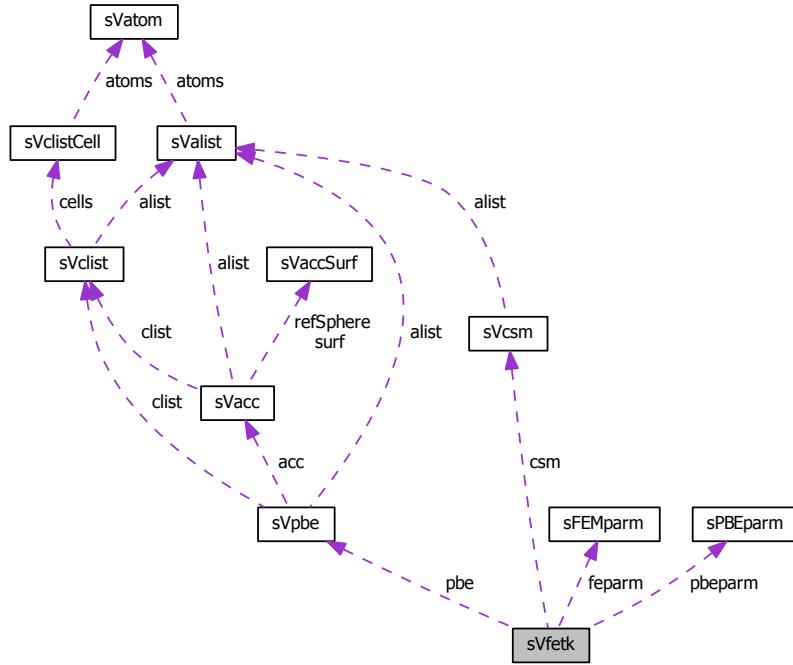
- src/fem/apbs/[vcsm.h](#)

8.14 sVfetk Struct Reference

Contains public data members for Vfetk class/module.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/fem/apbs/vfetk.h>
```

Collaboration diagram for sVfetk:



Data Fields

- `Vmem * vmem`
- `Gem * gm`
- `AM * am`
- `Aprx * aprx`
- `PDE * pde`
- `Vpbe * pbe`
- `Vcsm * csm`
- `Vfetk_LsolvType lkey`
- `int lmax`
- `double ltol`
- `Vfetk_NsolvType nkey`
- `int nmax`
- `double ntol`
- `Vfetk_GuessType gues`
- `Vfetk_PrecType lprec`
- `int pjac`
- `PBEparm * pbeparm`
- `FEMparm * feparm`
- `Vhal_PBEType type`
- `int level`

8.14.1 Detailed Description

Contains public data members for Vfetk class/module.

Author

Nathan Baker Many of the routines and macros are borrowed from the main.c driver (written by Mike Holst) provided with the PMG code.

Definition at line 173 of file [vfetk.h](#).

8.14.2 Field Documentation

8.14.2.1 AM* am

Multilevel algebra manager.

Definition at line 179 of file [vfetk.h](#).

8.14.2.2 Aprx* aprx

Approximation manager.

Definition at line 180 of file [vfetk.h](#).

8.14.2.3 Vcsm* csm

Charge-simplex map

Definition at line 183 of file [vfetk.h](#).

8.14.2.4 FEMparm* feparm

FEM-specific parameters

Definition at line 195 of file [vfetk.h](#).

8.14.2.5 Gem* gm

Grid manager (container class for master vertex and simplex lists as well as prolongation operator for updating after refinement).

Definition at line 176 of file [vfetk.h](#).

8.14.2.6 Vfetk_GuessType gues

Initial guess method

Definition at line 190 of file [vfetk.h](#).

8.14.2.7 int level

Refinement level (starts at 0)

Definition at line 197 of file [vfetk.h](#).

8.14.2.8 Vfetk_LsolvType lkey

Linear solver method

Definition at line 184 of file [vfetk.h](#).

8.14.2.9 int lmax

Maximum number of linear solver iterations

Definition at line 185 of file [vfetk.h](#).

8.14.2.10 Vfetk_PrecType lprec

Linear preconditioner

Definition at line 191 of file [vfetk.h](#).

8.14.2.11 double ltol

Residual tolerance for linear solver

Definition at line 186 of file [vfetk.h](#).

8.14.2.12 Vfetk_NsolvType nkey

Nonlinear solver method

Definition at line 187 of file [vfetk.h](#).

8.14.2.13 int nmax

Maximum number of nonlinear solver iterations

Definition at line 188 of file [vfetk.h](#).

8.14.2.14 double ntol

Residual tolerance for nonlinear solver

Definition at line 189 of file [vfetk.h](#).

8.14.2.15 Vpbe* pbe

Poisson-Boltzmann object

Definition at line 182 of file [vfetk.h](#).

8.14.2.16 PBEparm* pbeparm

Generic PB parameters

Definition at line 194 of file [vfetk.h](#).

8.14.2.17 PDE* pde

FEtk PDE object

Definition at line 181 of file [vfetk.h](#).

8.14.2.18 int pjac

Flag to print the jacobians (usually set this to -1, please)

Definition at line 192 of file [vfetk.h](#).

8.14.2.19 Vhal_PBEType type

Version of PBE to solve

Definition at line 196 of file [vfetk.h](#).

8.14.2.20 Vmem* vmem

Memory management object

Definition at line 175 of file [vfetk.h](#).

The documentation for this struct was generated from the following file:

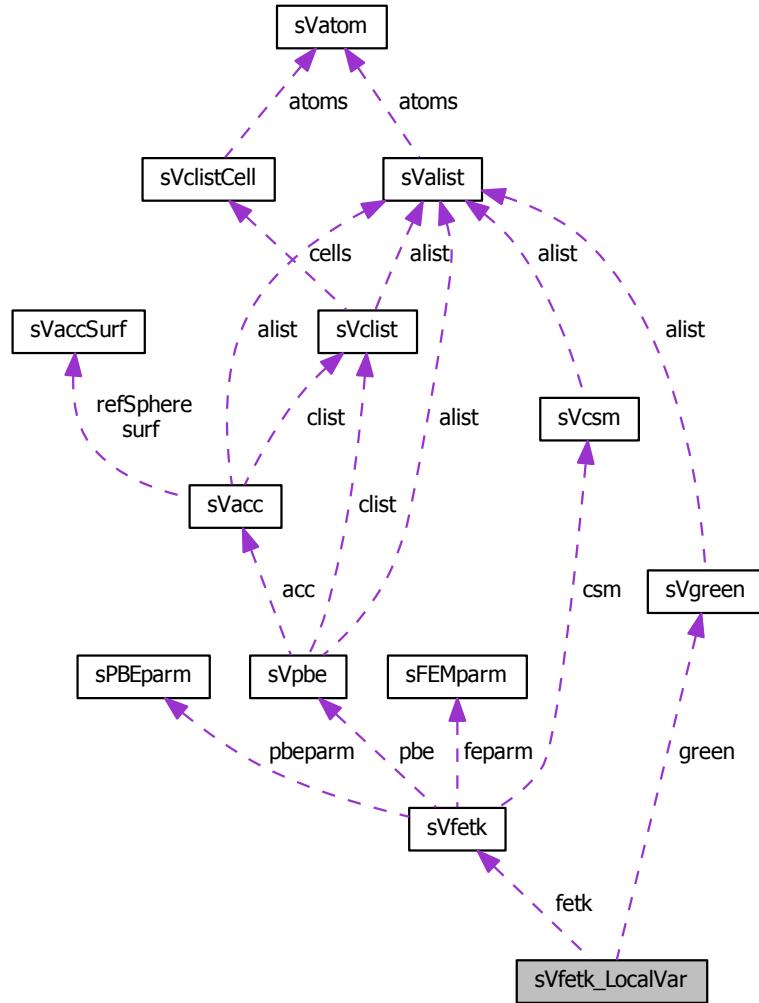
- src/fem/apbs/[vfetk.h](#)

8.15 sVfetk_LocalVar Struct Reference

Vfetk LocalVar subclass.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/fem/apbs/vfetk.h>
```

Collaboration diagram for sVfetk_LocalVar:



Data Fields

- double `nvec` [VAPBS_DIM]
- double `vx` [4][VAPBS_DIM]
- double `xq` [VAPBS_DIM]
- double `U` [MAXV]
- double `dU` [MAXV][VAPBS_DIM]
- double `W`
- double `dW` [VAPBS_DIM]
- double `d2W`
- int `sType`
- int `fType`

- double [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](#)

8.15.1 Detailed Description

Vfetk LocalVar subclass.

Author

Nathan Baker Contains variables used when solving the PDE with FEtk

Definition at line [212](#) of file [vfetk.h](#).

8.15.2 Field Documentation

8.15.2.1 double A

Second-order differential term

Definition at line [225](#) of file [vfetk.h](#).

8.15.2.2 double B

Entire ionic strength term

Definition at line [227](#) of file [vfetk.h](#).

8.15.2.3 double d2W

Coulomb regularization term Laplacian

Definition at line [220](#) of file [vfetk.h](#).

8.15.2.4 double DB

Entire ionic strength term derivative

Definition at line [228](#) of file [vfetk.h](#).

8.15.2.5 double delta

Store delta value

Definition at line [247](#) of file [vfetk.h](#).

8.15.2.6 double DFu_wv

Store DFu_wv value

Definition at line [246](#) of file [vfetk.h](#).

8.15.2.7 double dU[MAXV][VAPBS_DIM]

Solution gradient

Definition at line [217](#) of file [vfetk.h](#).

8.15.2.8 double dW[VAPBS_DIM]

Coulomb regularization term gradient

Definition at line [219](#) of file [vfetk.h](#).

8.15.2.9 double F

RHS characteristic function value

Definition at line [226](#) of file [vfetk.h](#).

8.15.2.10 Vfetk* fetk

Pointer to the VFETK object

Definition at line [230](#) of file [vfetk.h](#).

8.15.2.11 int fType

Face type

Definition at line [222](#) of file [vfetk.h](#).

8.15.2.12 double Fu_v

Store Fu_v value

Definition at line [245](#) of file [vfetk.h](#).

8.15.2.13 Vgreen* green

Pointer to a Green's function object

Definition at line [231](#) of file [vfetk.h](#).

8.15.2.14 int initGreen

Boolean to designate whether Green's function has been initialized

Definition at line [232](#) of file [vfetk.h](#).

8.15.2.15 double ionConc[MAXION]

Counterion species' concentrations

Definition at line [238](#) of file [vfetk.h](#).

8.15.2.16 double ionQ[MAXION]

Counterion species' valencies

Definition at line [239](#) of file [vfetk.h](#).

8.15.2.17 double ionRadii[MAXION]

Counterion species' radii

Definition at line [240](#) of file [vfetk.h](#).

8.15.2.18 double ionstr

Ionic strength parameters (M)

Definition at line [243](#) of file [vfetk.h](#).

8.15.2.19 double jumpDiel

Dielectric value on one side of a simplex face

Definition at line [229](#) of file [vfetk.h](#).

8.15.2.20 int nion

Number of ion species

Definition at line [244](#) of file [vfetk.h](#).

8.15.2.21 double nvec[VAPBS_DIM]

Normal vector for a simplex face

Definition at line [213](#) of file [vfetk.h](#).

8.15.2.22 int nverts

number of vertices in the simplex

Definition at line [237](#) of file [vfetk.h](#).

8.15.2.23 SS* simp

Pointer to the latest simplex object; set in initElement() and [delta\(\)](#)

Definition at line [234](#) of file [vfetk.h](#).

8.15.2.24 int sType

Simplex type

Definition at line [221](#) of file [vfetk.h](#).

8.15.2.25 double U[MAXV]

Solution value

Definition at line [216](#) of file [vfetk.h](#).

8.15.2.26 double u_D

Store Dirichlet value

Definition at line [248](#) of file [vfetk.h](#).

8.15.2.27 double u_T

Store true value

Definition at line [249](#) of file [vfetk.h](#).

8.15.2.28 VV* verts[4]

Pointer to the latest vertices; set in initElement

Definition at line [236](#) of file [vfetk.h](#).

8.15.2.29 double vx[4][VAPBS_DIM]

Vertex coordinates

Definition at line [214](#) of file [vfetk.h](#).

8.15.2.30 double W

Coulomb regularization term scalar value

Definition at line [218](#) of file [vfetk.h](#).

8.15.2.31 double xq[VAPBS_DIM]

Quadrature pt

Definition at line 215 of file [vfetk.h](#).

8.15.2.32 double zkappa2

Ionic strength parameters

Definition at line 241 of file [vfetk.h](#).

8.15.2.33 double zks2

Ionic strength parameters

Definition at line 242 of file [vfetk.h](#).

The documentation for this struct was generated from the following file:

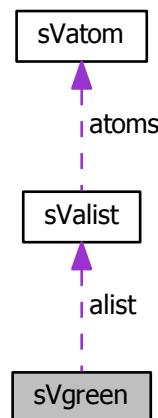
- src/fem/apbs/[vfetk.h](#)

8.16 sVgreen Struct Reference

Contains public data members for Vgreen class/module.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/trunk/src/generic/apbs/vgreen.h>
```

Collaboration diagram for sVgreen:



Data Fields

- `Valist * alist`
- `Vmem * vmem`
- `double * xp`
- `double * yp`
- `double * zp`
- `double * qp`
- `int np`

8.16.1 Detailed Description

Contains public data members for Vgreen class/module.

Author

Nathan Baker

Definition at line 83 of file [vgreen.h](#).

8.16.2 Field Documentation

8.16.2.1 Valist* alist

Atom (charge) list for Green's function

Definition at line 85 of file [vgreen.h](#).

8.16.2.2 int np

Set to size of above arrays

Definition at line 95 of file [vgreen.h](#).

8.16.2.3 double* qp

Array of particle charges for use with treecode routines

Definition at line 93 of file [vgreen.h](#).

8.16.2.4 Vmem* vmem

Memory management object

Definition at line 86 of file [vgreen.h](#).

8.16.2.5 double* xp

Array of particle x-coordinates for use with treecode routines

Definition at line 87 of file [vgreen.h](#).

8.16.2.6 double* yp

Array of particle y-coordinates for use with treecode routines

Definition at line 89 of file [vgreen.h](#).

8.16.2.7 double* zp

Array of particle z-coordinates for use with treecode routines

Definition at line 91 of file [vgreen.h](#).

The documentation for this struct was generated from the following file:

- [src/generic/apbs/vgreen.h](#)

8.17 sVgrid Struct Reference

Electrostatic potential oracle for Cartesian mesh data.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/mg/apbs/vgrid.h>
```

Data Fields

- int [nx](#)
- int [ny](#)
- int [nz](#)
- double [hx](#)
- double [hy](#)
- double [hzed](#)
- double [xmin](#)
- double [ymin](#)
- double [zmin](#)
- double [xmax](#)
- double [ymax](#)
- double [zmax](#)
- double * [data](#)
- int [readdata](#)
- int [ctordata](#)
- Vmem * [mem](#)

8.17.1 Detailed Description

Electrostatic potential oracle for Cartesian mesh data.

Author

Nathan Baker

Definition at line 79 of file [vgrid.h](#).

8.17.2 Field Documentation

8.17.2.1 int ctordata

flag indicating whether data was included at construction

Definition at line [95](#) of file [vgrid.h](#).

8.17.2.2 double* data

$nx*ny*nz$ array of data

Definition at line [93](#) of file [vgrid.h](#).

8.17.2.3 double hx

Grid spacing in x direction

Definition at line [84](#) of file [vgrid.h](#).

8.17.2.4 double hy

Grid spacing in y direction

Definition at line [85](#) of file [vgrid.h](#).

8.17.2.5 double hzed

Grid spacing in z direction

Definition at line [86](#) of file [vgrid.h](#).

8.17.2.6 Vmem* mem

Memory manager object

Definition at line [97](#) of file [vgrid.h](#).

8.17.2.7 int nx

Number grid points in x direction

Definition at line [81](#) of file [vgrid.h](#).

8.17.2.8 int ny

Number grid points in y direction

Definition at line [82](#) of file [vgrid.h](#).

8.17.2.9 int nz

Number grid points in z direction

Definition at line 83 of file [vgrid.h](#).

8.17.2.10 int readdata

flag indicating whether data was read from file

Definition at line 94 of file [vgrid.h](#).

8.17.2.11 double xmax

x coordinate of upper grid corner

Definition at line 90 of file [vgrid.h](#).

8.17.2.12 double xmin

x coordinate of lower grid corner

Definition at line 87 of file [vgrid.h](#).

8.17.2.13 double ymax

y coordinate of upper grid corner

Definition at line 91 of file [vgrid.h](#).

8.17.2.14 double ymin

y coordinate of lower grid corner

Definition at line 88 of file [vgrid.h](#).

8.17.2.15 double zmax

z coordinate of upper grid corner

Definition at line 92 of file [vgrid.h](#).

8.17.2.16 double zmin

z coordinate of lower grid corner

Definition at line 89 of file [vgrid.h](#).

The documentation for this struct was generated from the following file:

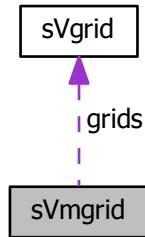
- src/mg/apbs/[vgrid.h](#)

8.18 sVmgrid Struct Reference

Multiresolution oracle for Cartesian mesh data.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/mg/apbs/vmgrid.h>
```

Collaboration diagram for sVmgrid:



Data Fields

- int [ngrids](#)
- [Vgrid * grids](#) [[VMGRIDMAX](#)]

8.18.1 Detailed Description

Multiresolution oracle for Cartesian mesh data.

Author

Nathan Baker

Definition at line [84](#) of file [vmgrid.h](#).

8.18.2 Field Documentation

8.18.2.1 [Vgrid* grids\[VMGRIDMAX\]](#)

Grids in hierarchy. Our convention will be to have the finest grid first, however, this will not be enforced as it may be useful to search multiple grids for parallel datasets, etc.

Definition at line [87](#) of file [vmgrid.h](#).

8.18.2.2 int [ngrids](#)

Number of grids in hierarchy

Definition at line 86 of file [vmgrid.h](#).

The documentation for this struct was generated from the following file:

- [src/mg/apbs/vmgrid.h](#)

8.19 sVmgrid Struct Reference

Declaration of Vmultigrid class.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/mg/apbs/vmultigrid.h>
```

Data Fields

- Vmultigrid_method [method](#)
The computation method to be used.
- Vmultigrid_boundary_method [boundary_method](#)
The method to be used for handling boundary conditions.
 - Vmultigrid_prolong_method [prolong_method](#)
 - Vmultigrid_coarsen_method [coarsen_method](#)
 - Vmultigrid_discrete_method [discrete_method](#)
 - Vmultigrid_smooth_method [smooth_method](#)
 - Vmultigrid_coarse_solver [coarse_solver](#)
 - Vmultigrid_iterate_method [iterate_method](#)
The multigrid iteration method to be used.
 - Vmultigrid_verbosity [verbosity](#)
 - Vmultigrid_stop_criterion [stop_criterion](#)
 - Vmultigrid_linear [linear](#)
The linear or non-linear approximation to use.
 - Vmultigrid_operator_analysis [operator_analysis](#)
The operator analysis method to use.
 - int [real_workspace_limit](#)
 - int [integer_workspace_limit](#)
 - int [real_workspace_size](#)
 - int [integer_workspace_size](#)
 - int [x_size](#)
 - int [y_size](#)
 - int [z_size](#)
 - int [level_count](#)
 - int [fine_count](#)
 - int [coarse_count](#)
 - int [max_iterations](#)
The maximum number of iterations to perform.
 - Vm int [ipcon](#)
 - int [irite](#)
 - int [nonlin](#)

8.19.1 Detailed Description

Declaration of Vmultigrid class.

Author

Tucker Beck

Definition at line 220 of file [vmultigrid.h](#).

8.19.2 Field Documentation

8.19.2.1 int coarse_count

The number of unknowns on the coarsest mesh

Note

Replaces nc from [mgdrv.d.h](#)

Definition at line 313 of file [vmultigrid.h](#).

8.19.2.2 Vmultigrid_coarse_solver coarse_solver

The coarse solver to be used

Note

Replaces mgsolv from [mgdrv.d.c](#)

Definition at line 248 of file [vmultigrid.h](#).

8.19.2.3 Vmultigrid_coarsen_method coarsen_method

The coarsening method to be used

Note

Replaces mgcoar from [mgdrv.d.c](#)

Definition at line 236 of file [vmultigrid.h](#).

8.19.2.4 Vmultigrid_discrete_method discrete_method

The discretization method to be used

Note

Replaces mgdisc from [mgdrv.d.c](#)

Definition at line 240 of file [vmultigrid.h](#).

8.19.2.5 int fine_count

The number of unknowns on the finest mesh

Note

Replaces nf from [mgdrv.h](#)

Definition at line [309](#) of file [vmultigrid.h](#).

8.19.2.6 int integer_workspace_limit

The upper bound on the size of the integer work space

Note

Replaces irwk from [mgdrv.c](#)

Definition at line [277](#) of file [vmultigrid.h](#).

8.19.2.7 int integer_workspace_size

The total size of the integer work space

Note

Replaces iintot from [mgdrv.c](#)

Definition at line [285](#) of file [vmultigrid.h](#).

8.19.2.8 Vm int ipcon

Definition at line [333](#) of file [vmultigrid.h](#).

8.19.2.9 int irite

Definition at line [336](#) of file [vmultigrid.h](#).

8.19.2.10 int level_count

The number of levels of the multi-grid

Note

Replaces nlev from [mgdrv.h](#)

Definition at line [305](#) of file [vmultigrid.h](#).

8.19.2.11 int nonlin

Definition at line [339](#) of file [vmultigrid.h](#).

8.19.2.12 Vmultigrid_prolong_method prolong_method

The prolongation method to be used

Note

Replaces mgprol from [mgdrv.c](#)

Definition at line [232](#) of file [vmultigrid.h](#).

8.19.2.13 int real_workspace_limit

The upper bound on the size of the real work space

Note

Replaces nrwk from [mgdrv.c](#)

Definition at line [273](#) of file [vmultigrid.h](#).

8.19.2.14 int real_workspace_size

The total size of the real work space

Note

Replaces iretot from [mgdrv.c](#)

Definition at line [281](#) of file [vmultigrid.h](#).

8.19.2.15 Vmultigrid_smooth_method smooth_method

The smoothing method to be used

Note

Replaces mgsmoo from [mgdrv.c](#)

Definition at line [244](#) of file [vmultigrid.h](#).

8.19.2.16 Vmultigrid_stop_criterion stop_criterion

The stopping criterion to cease comutation iterations

Note

Replaces istop from [mgdrv.c](#)

Definition at line [259](#) of file [vmultigrid.h](#).

8.19.2.17 Vmultigrid.verbosity verbosity

The verbosity of status messages to display

Note

Replaces iinfo from [mgdrv.c](#)

Definition at line [255](#) of file [vmultigrid.h](#).

8.19.2.18 int x_size

The size of the grid in the x dimension

Note

Replaces nx from [mgdrv.h](#)

Definition at line [293](#) of file [vmultigrid.h](#).

8.19.2.19 int y_size

The size of the grid in the y dimension

Note

Replaces ny from [mgdrv.h](#)

Definition at line [297](#) of file [vmultigrid.h](#).

8.19.2.20 int z_size

The size of the grid in the z dimension

Note

Replaces nz from [mgdrv.h](#)

Definition at line [301](#) of file [vmultigrid.h](#).

The documentation for this struct was generated from the following file:

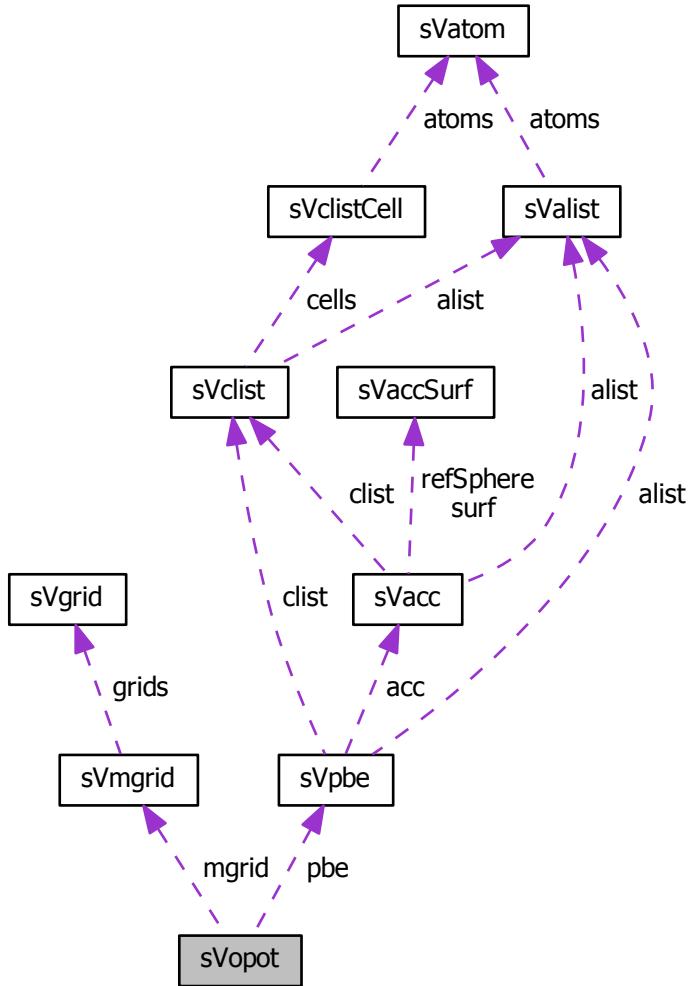
- src/mg/apbs/[vmultigrid.h](#)

8.20 sVopot Struct Reference

Electrostatic potential oracle for Cartesian mesh data.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/mg/apbs/vopot.h>
```

Collaboration diagram for sVopot:



Data Fields

- `Vmgrid * mgrid`
- `Vpbe * pbe`
- `Vbcfl bcfl`

8.20.1 Detailed Description

Electrostatic potential oracle for Cartesian mesh data.

Author

Nathan Baker

Definition at line 82 of file [vopot.h](#).

8.20.2 Field Documentation

8.20.2.1 Vbcfl bcfl

Boundary condition flag for returning potential values at points off the grid.

Definition at line 87 of file [vopot.h](#).

8.20.2.2 Vmgrid* mgrid

Multiple grid object containing potential data (in units kT/e)

Definition at line 84 of file [vopot.h](#).

8.20.2.3 Vpbe* pbe

Pointer to PBE object

Definition at line 86 of file [vopot.h](#).

The documentation for this struct was generated from the following file:

- src/mg/apbs/[vopot.h](#)

8.21 sVparam_AtomData Struct Reference

AtomData sub-class; stores atom data.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS/trunk/src/generic/apbs/vparam.h>
```

Data Fields

- char [atomName](#) [VMAX_ARGLEN]
- char [resName](#) [VMAX_ARGLEN]
- double [charge](#)
- double [radius](#)
- double [epsilon](#)

8.21.1 Detailed Description

AtomData sub-class; stores atom data.

Author

Nathan Baker

Note

The epsilon and radius members of this class refer use the following formula for calculating the van der Waals energy of atom i interacting with atom j :

$$V_{ij}(r_{ij}) = \epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - 2 \left(\frac{\sigma_{ij}}{r_{ij}} \right)^6 \right]$$

where $\epsilon_{ij} = \sqrt{\epsilon_i \epsilon_j}$ is the well-depth (in the desired energy units), r_{ij} is the distance between atoms i and j , and $\sigma_{ij} = \sigma_i + \sigma_j$ is the sum of the van der Waals radii.

Definition at line 87 of file [vpParam.h](#).

8.21.2 Field Documentation

8.21.2.1 char atomName[VMAX_ARGLEN]

Atom name

Definition at line 88 of file [vpParam.h](#).

8.21.2.2 double charge

Atom charge (in e)

Definition at line 90 of file [vpParam.h](#).

8.21.2.3 double epsilon

Atom VdW well depth (ϵ_i above; in kJ/mol)

Definition at line 92 of file [vpParam.h](#).

8.21.2.4 double radius

Atom VdW radius (σ_i above; in Å)

Definition at line 91 of file [vpParam.h](#).

8.21.2.5 char resName[VMAX_ARGLEN]

Residue name

Definition at line 89 of file [vpParam.h](#).

The documentation for this struct was generated from the following file:

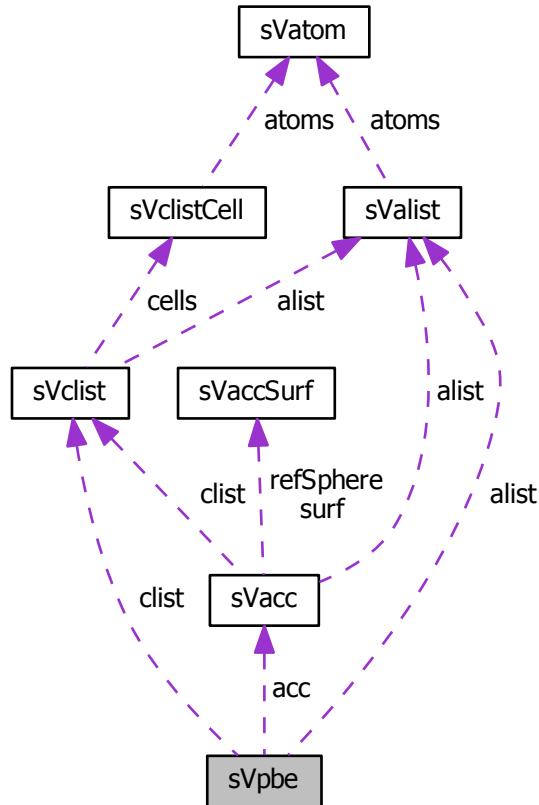
- [src/generic/apbs/vpParam.h](#)

8.22 sVpbe Struct Reference

Contains public data members for Vpbe class/module.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/generic/apbs/vpbe.h>
```

Collaboration diagram for sVpbe:



Data Fields

- Vmem * vmem
- Valist * alist
- Vclist * clist
- Vacc * acc
- double T
- double soluteDiel
- double solventDiel
- double solventRadius
- double bulkIonicStrength

- double maxlonRadius
- int numlon
- 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

8.22.1 Detailed Description

Contains public data members for Vpbe class/module.

Author

Nathan Baker

Definition at line 84 of file [vpbe.h](#).

8.22.2 Field Documentation

8.22.2.1 Vacc* acc

Accessibility object

Definition at line 90 of file [vpbe.h](#).

8.22.2.2 Valist* alist

Atom (charge) list

Definition at line 88 of file [vpbe.h](#).

8.22.2.3 double bulkIonicStrength

Bulk ionic strength (M)

Definition at line 99 of file [vpbe.h](#).

8.22.2.4 Vclist* clist

Atom location cell list

Definition at line 89 of file [vpbe.h](#).

8.22.2.5 double deblen

Debye length (bulk)

Definition at line 109 of file [vpbe.h](#).

8.22.2.6 double ionConc[MAXION]

Concentration (M) of each species

Definition at line 104 of file [vpbe.h](#).

8.22.2.7 double ionQ[MAXION]

Charge (e) of each species

Definition at line 106 of file [vpbe.h](#).

8.22.2.8 double ionRadii[MAXION]

Ionic radius (A) of each species

Definition at line 105 of file [vpbe.h](#).

8.22.2.9 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](#).

8.22.2.10 double L

Length of the membrane (A)

Definition at line 132 of file [vpbe.h](#).

8.22.2.11 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](#).

8.22.2.12 double membraneDielectric

Membrane dielectric constant

Definition at line 133 of file [vpbe.h](#).

8.22.2.13 int numIon

Total number of ion species

Definition at line 103 of file [vpbe.h](#).

8.22.2.14 int param2Flag

Check to see if bcfl=3 parms have been set

Definition at line 135 of file [vpbe.h](#).

8.22.2.15 int paramFlag

Check to see if the parameters have been set

Definition at line 125 of file [vpbe.h](#).

8.22.2.16 double smsize

Size-Modified PBE size

Definition at line 121 of file [vpbe.h](#).

8.22.2.17 double smvolume

Size-Modified PBE relative volume

Definition at line 120 of file [vpbe.h](#).

8.22.2.18 double soluteCenter[3]

Center of solute molecule (A)

Definition at line 113 of file [vpbe.h](#).

8.22.2.19 double soluteCharge

Charge of solute molecule (e)

Definition at line 118 of file [vpbe.h](#).

8.22.2.20 double soluteDielectric

Solute dielectric constant (unitless)

Definition at line 93 of file [vpbe.h](#).

8.22.2.21 double soluteRadius

Radius of solute molecule (A)

Definition at line 114 of file [vpbe.h](#).

8.22.2.22 double soluteXlen

Solute length in x-direction

Definition at line 115 of file [vpbe.h](#).

8.22.2.23 double soluteYlen

Solute length in y-direction

Definition at line 116 of file [vpbe.h](#).

8.22.2.24 double soluteZlen

Solute length in z-direction

Definition at line 117 of file [vpbe.h](#).

8.22.2.25 double solventDiel

Solvent dielectric constant (unitless)

Definition at line 94 of file [vpbe.h](#).

8.22.2.26 double solventRadius

Solvent probe radius (angstroms) for accessibility; determining defining volumes for the dielectric coefficient

Definition at line 96 of file [vpbe.h](#).

8.22.2.27 double T

Temperature (K)

Definition at line 92 of file [vpbe.h](#).

8.22.2.28 double V

Membrane potential

Definition at line 134 of file [vpbe.h](#).

8.22.2.29 Vmem* vmem

Memory management object

Definition at line 86 of file [vpbe.h](#).

8.22.2.30 double xkappa

Debye-Huckel parameter (bulk)

Definition at line 108 of file [vpbe.h](#).

8.22.2.31 double z_mem

Z value of the bottom of the membrane (A)

Definition at line 131 of file [vpbe.h](#).

8.22.2.32 double zkappa2

Square of modified Debye-Huckel parameter (bulk)

Definition at line 110 of file [vpbe.h](#).

8.22.2.33 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/apbs/[vpbe.h](#)

8.23 sVpee Struct Reference

Contains public data members for Vpee class/module.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/fem/apbs/vpee.h>
```

Data Fields

- Gmem * gm
- int localPartID
- double localPartCenter [3]
- double localPartRadius
- int killFlag
- double killParam
- Vmem * mem

8.23.1 Detailed Description

Contains public data members for Vpee class/module.

Author

Nathan Baker

Definition at line 88 of file [vpee.h](#).

8.23.2 Field Documentation

8.23.2.1 Gem* gm

Grid manager

Definition at line 90 of file [vpee.h](#).

8.23.2.2 int killFlag

A flag indicating the method we're using to artificially decrease the error estimate outside the local partition

Definition at line 98 of file [vpee.h](#).

8.23.2.3 double killParam

A parameter for the error estimate attenuation method

Definition at line 101 of file [vpee.h](#).

8.23.2.4 double localPartCenter[3]

The coordinates of the center of the local partition

Definition at line 94 of file [vpee.h](#).

8.23.2.5 int localPartID

The local partition ID: i.e. the partition whose boundary simplices we're keeping track of

Definition at line 91 of file [vpee.h](#).

8.23.2.6 double localPartRadius

The radius of the circle/sphere which circumscribes the local partition

Definition at line 96 of file [vpee.h](#).

8.23.2.7 Vmem* mem

Memory manager

Definition at line 103 of file [vpee.h](#).

The documentation for this struct was generated from the following file:

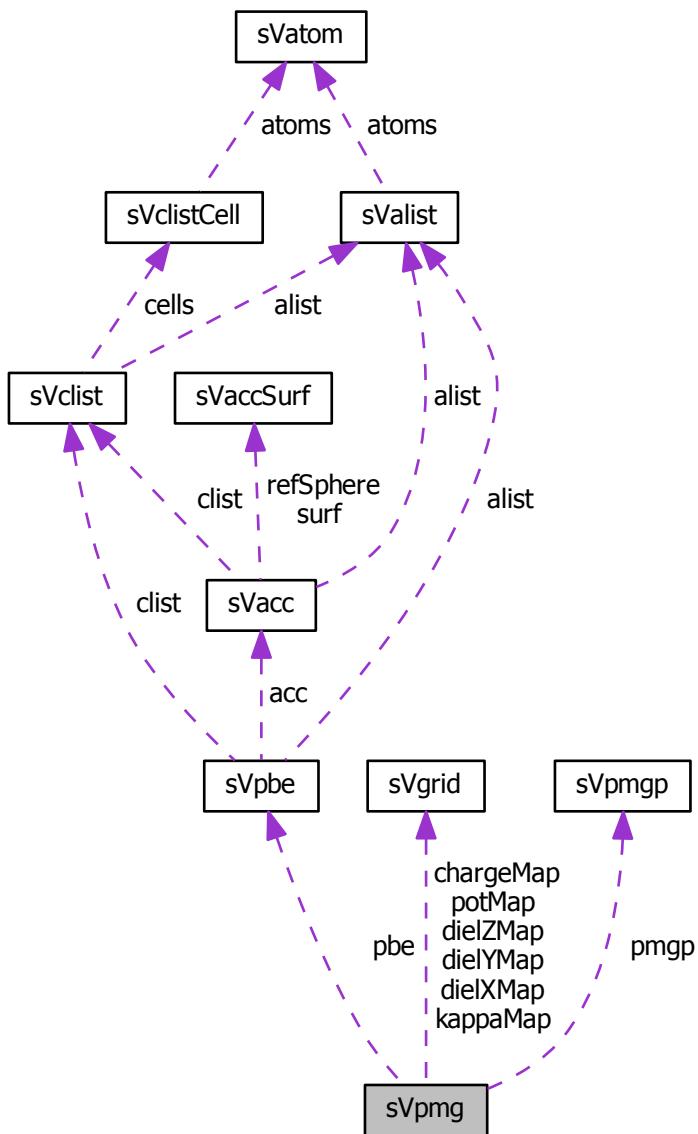
- src/fem/apbs/[vpee.h](#)

8.24 sVpmg Struct Reference

Contains public data members for Vpmg class/module.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/mg/apbs/vpmg.h>
```

Collaboration diagram for sVpmg:



Data Fields

- `Vmem * vmem`
- `Vpmgp * pmgp`
- `Vpbe * pbe`
- `double * epsx`
- `double * epsy`
- `double * epsz`
- `double * kappa`
- `double * pot`
- `double * charge`
- `int * iparm`
- `double * rparm`
- `int * iwork`
- `double * rwork`
- `double * a1cf`
- `double * a2cf`
- `double * a3cf`
- `double * ccf`
- `double * fcf`
- `double * tcf`
- `double * u`
- `double * xf`
- `double * yf`
- `double * zf`
- `double * gxcf`
- `double * gycf`
- `double * gzcf`
- `double * pvec`
- `double extDiEnergy`
- `double extQmEnergy`
- `double extQfEnergy`
- `double extNpEnergy`
- `Vsurf_Meth surfMeth`
- `double splineWin`
- `Vchrg_Meth chargeMeth`
- `Vchrg_Src chargeSrc`
- `int filled`
- `int useDielXMap`
- `Vgrid * dielXMap`
- `int useDielYMap`
- `Vgrid * dielYMap`
- `int useDielZMap`
- `Vgrid * dielZMap`
- `int useKappaMap`
- `Vgrid * kappaMap`
- `int usePotMap`
- `Vgrid * potMap`
- `int useChargeMap`
- `Vgrid * chargeMap`

8.24.1 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 120 of file [vpmg.h](#).

8.24.2 Field Documentation

8.24.2.1 double* a1cf

Operator coefficient values (a11) – this array can be overwritten

Definition at line 142 of file [vpmg.h](#).

8.24.2.2 double* a2cf

Operator coefficient values (a22) – this array can be overwritten

Definition at line 144 of file [vpmg.h](#).

8.24.2.3 double* a3cf

Operator coefficient values (a33) – this array can be overwritten

Definition at line 146 of file [vpmg.h](#).

8.24.2.4 double* ccf

Helmholtz term – this array can be overwritten

Definition at line 148 of file [vpmg.h](#).

8.24.2.5 double* charge

Charge map

Definition at line 136 of file [vpmg.h](#).

8.24.2.6 Vgrid* chargeMap

External charge distribution map

Definition at line 192 of file [vpmg.h](#).

8.24.2.7 Vchrg_Meth chargeMeth

Charge discretization method

Definition at line 169 of file [vpmg.h](#).

8.24.2.8 Vchrg_Src chargeSrc

Charge source

Definition at line 170 of file [vpmg.h](#).

8.24.2.9 Vgrid* dielXMap

External x-shifted dielectric map

Definition at line 176 of file [vpmg.h](#).

8.24.2.10 Vgrid* dielYMap

External y-shifted dielectric map

Definition at line 179 of file [vpmg.h](#).

8.24.2.11 Vgrid* dielZMap

External z-shifted dielectric map

Definition at line 182 of file [vpmg.h](#).

8.24.2.12 double* epsx

X-shifted dielectric map

Definition at line 131 of file [vpmg.h](#).

8.24.2.13 double* epsy

Y-shifted dielectric map

Definition at line 132 of file [vpmg.h](#).

8.24.2.14 double* epsz

Z-shifted dielectric map

Definition at line 133 of file [vpmg.h](#).

8.24.2.15 double extDiEnergy

Stores contributions to the dielectric energy from regions outside the problem domain

Definition at line 159 of file [vpmg.h](#).

8.24.2.16 double extNpEnergy

Stores contributions to the apolar energy from regions outside the problem domain

Definition at line 165 of file [vpmg.h](#).

8.24.2.17 double extQfEnergy

Stores contributions to the fixed charge energy from regions outside the problem domain

Definition at line 163 of file [vpmg.h](#).

8.24.2.18 double extQmEnergy

Stores contributions to the mobile ion energy from regions outside the problem domain

Definition at line 161 of file [vpmg.h](#).

8.24.2.19 double* fcf

Right-hand side – this array can be overwritten

Definition at line 149 of file [vpmg.h](#).

8.24.2.20 int filled

Indicates whether Vpmg_fillco has been called

Definition at line 172 of file [vpmg.h](#).

8.24.2.21 double* gxcf

Boundary conditions for x faces

Definition at line 155 of file [vpmg.h](#).

8.24.2.22 double* gycf

Boundary conditions for y faces

Definition at line 156 of file [vpmg.h](#).

8.24.2.23 double* gzcfc

Boundary conditions for z faces

Definition at line 157 of file [vpmg.h](#).

8.24.2.24 int* iparm

Passing int parameters to FORTRAN

Definition at line 138 of file [vpmg.h](#).

8.24.2.25 int* iwork

Work array

Definition at line 140 of file [vpmg.h](#).

8.24.2.26 double* kappa

Ion accessibility map ($0 \leq \text{kappa}(x) \leq 1$)

Definition at line 134 of file [vpmg.h](#).

8.24.2.27 Vgrid* kappaMap

External kappa map

Definition at line 185 of file [vpmg.h](#).

8.24.2.28 Vpbe* pbe

Information about the PBE system

Definition at line 124 of file [vpmg.h](#).

8.24.2.29 Vpmgp* pmgp

Parameters

Definition at line 123 of file [vpmg.h](#).

8.24.2.30 double* pot

Potential map

Definition at line 135 of file [vpmg.h](#).

8.24.2.31 Vgrid* potMap

External potential map

Definition at line 188 of file [vpmg.h](#).

8.24.2.32 double* pvec

Partition mask array

Definition at line 158 of file [vpmg.h](#).

8.24.2.33 double* rparm

Passing real parameters to FORTRAN

Definition at line 139 of file [vpmg.h](#).

8.24.2.34 double* rwork

Work array

Definition at line 141 of file [vpmg.h](#).

8.24.2.35 double splineWin

Spline window parm for surf defs

Definition at line 168 of file [vpmg.h](#).

8.24.2.36 Vsurf_Meth surfMeth

Surface definition method

Definition at line 167 of file [vpmg.h](#).

8.24.2.37 double* tcf

True solution

Definition at line 150 of file [vpmg.h](#).

8.24.2.38 double* u

Solution

Definition at line 151 of file [vpmg.h](#).

8.24.2.39 int useChargeMap

Indicates whether Vpmg_fillco was called with an external charge distribution map

Definition at line 190 of file [vpmg.h](#).

8.24.2.40 int useDielXMap

Indicates whether Vpmg_fillco was called with an external x-shifted dielectric map

Definition at line 174 of file [vpmg.h](#).

8.24.2.41 int useDielYMap

Indicates whether Vpmg_fillco was called with an external y-shifted dielectric map

Definition at line 177 of file [vpmg.h](#).

8.24.2.42 int useDielZMap

Indicates whether Vpmg_fillco was called with an external z-shifted dielectric map

Definition at line 180 of file [vpmg.h](#).

8.24.2.43 int useKappaMap

Indicates whether Vpmgp_fillco was called with an external kappa map

Definition at line 183 of file [vpmg.h](#).

8.24.2.44 int usePotMap

Indicates whether Vpmgp_fillco was called with an external potential map

Definition at line 186 of file [vpmg.h](#).

8.24.2.45 Vmem* vmem

Memory management object for this class

Definition at line 122 of file [vpmg.h](#).

8.24.2.46 double* xf

Mesh point x coordinates

Definition at line 152 of file [vpmg.h](#).

8.24.2.47 double* yf

Mesh point y coordinates

Definition at line 153 of file [vpmg.h](#).

8.24.2.48 double* zf

Mesh point z coordinates

Definition at line 154 of file [vpmg.h](#).

The documentation for this struct was generated from the following file:

- [src/mg/apbs/vpmg.h](#)

8.25 sVpmgp Struct Reference

Contains public data members for Vpmgp class/module.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/mg/apbs/vpmgp.h>
```

Data Fields

- int [nx](#)
- int [ny](#)
- int [nz](#)

- int `nlev`
- double `hx`
- double `hy`
- double `hzed`
- int `nonlin`
- int `nxc`
- int `nyc`
- int `nzc`
- int `nf`
- int `nc`
- int `narrc`
- int `n_rpc`
- int `n_iz`
- int `n_ipc`
- int `nrwk`
- int `niwk`
- int `narr`
- int `ipkey`
- double `xcent`
- double `ycent`
- double `zcent`
- double `errtol`
- int `itmax`
- int `istop`
- int `iinfo`
- `Vbcfl bcfl`
- int `key`
- int `iperf`
- int `meth`
- int `mgkey`
- int `nu1`
- int `nu2`
- int `mgsmoo`
- int `mgprol`
- int `mgcoar`
- int `mgsolv`
- int `mgdisc`
- double `omegal`
- double `omegan`
- int `irite`
- int `ipcon`
- double `xlen`
- double `ylen`
- double `zlen`
- double `xmin`
- double `ymin`
- double `zmin`
- double `xmax`
- double `ymax`
- double `zmax`

8.25.1 Detailed Description

Contains public data members for Vpmgp class/module.

Author

Nathan Baker

Bug Value ipcon does not currently allow for preconditioning in PMG

Definition at line [78](#) of file [vpmgp.h](#).

8.25.2 Field Documentation

8.25.2.1 Vbcfl bcfl

Boundary condition method [default = BCFL_SDH]

Definition at line [133](#) of file [vpmgp.h](#).

8.25.2.2 double errtol

Desired error tolerance [default = 1e-9]

Definition at line [119](#) of file [vpmgp.h](#).

8.25.2.3 double hx

Grid x spacings [no default]

Definition at line [85](#) of file [vpmgp.h](#).

8.25.2.4 double hy

Grid y spacings [no default]

Definition at line [86](#) of file [vpmgp.h](#).

8.25.2.5 double hzed

Grid z spacings [no default]

Definition at line [87](#) of file [vpmgp.h](#).

8.25.2.6 int iinfo

Runtime status messages [default = 1]

- 0: none
- 1: some

- 2: lots
- 3: more

Definition at line 128 of file [vpmgp.h](#).

8.25.2.7 int ipcon

Preconditioning method [default = 3]

- 0: diagonal
- 1: ICCG
- 2: ICCGDW
- 3: MICCGDW
- 4: none

Definition at line 181 of file [vpmgp.h](#).

8.25.2.8 int iperf

Analysis of the operator [default = 0]

- 0: no
- 1: condition number
- 2: spectral radius
- 3: cond. number & spectral radius

Definition at line 137 of file [vpmgp.h](#).

8.25.2.9 int ipkey

Toggles nonlinearity (set by nonlin)

- -2: Size-Modified PBE
- -1: Linearized PBE
- 0: Nonlinear PBE with capped sinh term [default]
- >1: Polynomial approximation to sinh, note that ipkey must be odd

Definition at line 107 of file [vpmgp.h](#).

8.25.2.10 int irite

FORTRAN output unit [default = 8]

Definition at line 180 of file [vpmgp.h](#).

8.25.2.11 int istop

Stopping criterion [default = 1]

- 0: residual
- 1: relative residual
- 2: diff
- 3: errc
- 4: errd
- 5: aerrd

Definition at line 121 of file [vpmgp.h](#).

8.25.2.12 int itmax

Maximum number of iters [default = 100]

Definition at line 120 of file [vpmgp.h](#).

8.25.2.13 int key

Print solution to file [default = 0]

- 0: no
- 1: yes

Definition at line 134 of file [vpmgp.h](#).

8.25.2.14 int meth

Solution method [default = 2]

- 0: conjugate gradient multigrid
- 1: newton
- 2: multigrid
- 3: conjugate gradient
- 4: sucessive overrelaxation
- 5: red-black gauss-seidel
- 6: weighted jacobi
- 7: richardson
- 8: conjugate gradient multigrid aqua
- 9: newton aqua

Definition at line 142 of file [vpmgp.h](#).

8.25.2.15 int mgcoar

Coarsening method [default = 2]

- 0: standard
- 1: harmonic
- 2: galerkin

Definition at line 168 of file [vpmgp.h](#).

8.25.2.16 int mgdisc

Discretization method [default = 0]

- 0: finite volume
- 1: finite element

Definition at line 175 of file [vpmgp.h](#).

8.25.2.17 int mgkey

Multigrid method [default = 0]

- 0: variable v-cycle
- 1: nested iteration

Definition at line 153 of file [vpmgp.h](#).

8.25.2.18 int mgprol

Prolongation method [default = 0]

- 0: trilinear
- 1: operator-based
- 2: mod. operator-based

Definition at line 164 of file [vpmgp.h](#).

8.25.2.19 int mgsmoo

Smoothing method [default = 1]

- 0: weighted jacobi
- 1: gauss-seidel
- 2: SOR

- 3: richardson
- 4: cgls

Definition at line 158 of file [vpmgp.h](#).

8.25.2.20 int mgsolv

Coarse equation solve method [default = 1]

- 0: cgls
- 1: banded linpack

Definition at line 172 of file [vpmgp.h](#).

8.25.2.21 int n_ipc

Integer info work array required storage

Definition at line 102 of file [vpmgp.h](#).

8.25.2.22 int n_iz

Integer storage parameter (index max)

Definition at line 101 of file [vpmgp.h](#).

8.25.2.23 int n_rpc

Real info work array required storage

Definition at line 100 of file [vpmgp.h](#).

8.25.2.24 int narr

Array work storage

Definition at line 106 of file [vpmgp.h](#).

8.25.2.25 int narrc

Size of vector on coarse level

Definition at line 99 of file [vpmgp.h](#).

8.25.2.26 int nc

Number of coarse grid unknowns

Definition at line 98 of file [vpmgp.h](#).

8.25.2.27 int nf

Number of fine grid unknowns

Definition at line 97 of file [vpmgp.h](#).

8.25.2.28 int niwk

Integer work storage

Definition at line 105 of file [vpmgp.h](#).

8.25.2.29 int nlev

Number of mesh levels [no default]

Definition at line 84 of file [vpmgp.h](#).

8.25.2.30 int nonlin

Problem type [no default]

- 0: linear
- 1: nonlinear
- 2: linear then nonlinear

Definition at line 88 of file [vpmgp.h](#).

8.25.2.31 int nrwk

Real work storage

Definition at line 104 of file [vpmgp.h](#).

8.25.2.32 int nu1

Number of pre-smoothings [default = 2]

Definition at line 156 of file [vpmgp.h](#).

8.25.2.33 int nu2

Number of post-smoothings [default = 2]

Definition at line 157 of file [vpmgp.h](#).

8.25.2.34 int nx

Grid x dimensions [no default]

Definition at line 81 of file [vpmgp.h](#).

8.25.2.35 int nxc

Coarse level grid x dimensions

Definition at line 94 of file [vpmgp.h](#).

8.25.2.36 int ny

Grid y dimensions [no default]

Definition at line 82 of file [vpmgp.h](#).

8.25.2.37 int nyc

Coarse level grid y dimensions

Definition at line 95 of file [vpmgp.h](#).

8.25.2.38 int nz

Grid z dimensions [no default]

Definition at line 83 of file [vpmgp.h](#).

8.25.2.39 int nzc

Coarse level grid z dimensions

Definition at line 96 of file [vpmgp.h](#).

8.25.2.40 double omegal

Linear relax parameter [default = 8e-1]

Definition at line 178 of file [vpmgp.h](#).

8.25.2.41 double omegan

Nonlin relax parameter [default = 9e-1]

Definition at line 179 of file [vpmgp.h](#).

8.25.2.42 double xcent

Grid x center [0]

Definition at line 116 of file [vpmgp.h](#).

8.25.2.43 double xlen

Domain x length

Definition at line 187 of file [vpmgp.h](#).

8.25.2.44 double xmax

Domain upper x corner

Definition at line 193 of file [vpmgp.h](#).

8.25.2.45 double xmin

Domain lower x corner

Definition at line 190 of file [vpmgp.h](#).

8.25.2.46 double ycent

Grid y center [0]

Definition at line 117 of file [vpmgp.h](#).

8.25.2.47 double ylen

Domain y length

Definition at line 188 of file [vpmgp.h](#).

8.25.2.48 double ymax

Domain upper y corner

Definition at line 194 of file [vpmgp.h](#).

8.25.2.49 double ymin

Domain lower y corner

Definition at line 191 of file [vpmgp.h](#).

8.25.2.50 double zcent

Grid z center [0]

Definition at line 118 of file [vpmgp.h](#).

8.25.2.51 double zlen

Domain z length

Definition at line 189 of file [vpmgp.h](#).

8.25.2.52 double zmax

Domain upper z corner

Definition at line 195 of file [vpmgp.h](#).

8.25.2.53 double zmin

Domain lower z corner

Definition at line 192 of file [vpmgp.h](#).

The documentation for this struct was generated from the following file:

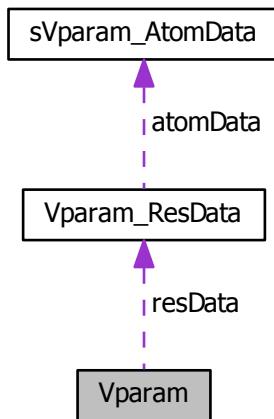
- src/mg/apbs/[vpmgp.h](#)

8.26 Vparam Struct Reference

Reads and assigns charge/radii parameters.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/generic/apbs/vparam.h>
```

Collaboration diagram for Vparam:



Data Fields

- Vmem * [vmem](#)
- int [nResData](#)
- [Vparam_ResData](#) * [resData](#)

8.26.1 Detailed Description

Reads and assigns charge/radii parameters.

Author

Nathan Baker

Definition at line 130 of file [vparam.h](#).

8.26.2 Field Documentation

8.26.2.1 int nResData

Number of [Vparam_ResData](#) objects associated with this object

Definition at line 133 of file [vparam.h](#).

8.26.2.2 Vparam_ResData* resData

Array of nResData [Vparam_ResData](#) objects

Definition at line 135 of file [vparam.h](#).

8.26.2.3 Vmem* vmem

Memory management object for this class

Definition at line 132 of file [vparam.h](#).

The documentation for this struct was generated from the following file:

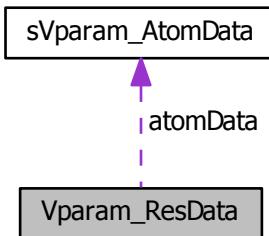
- [src/generic/apbs/vparam.h](#)

8.27 Vparam_ResData Struct Reference

ResData sub-class; stores residue data.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APBS-trunk/src/generic/apbs/vparam.h>
```

Collaboration diagram for Vparam_ResData:



Data Fields

- Vmem * `vmem`
- char `name` [VMAX_ARGLEN]
- int `nAtomData`
- Vparam_AtomData * `atomData`

8.27.1 Detailed Description

ResData sub-class; stores residue data.

Author

Nathan Baker

Definition at line 109 of file [vparam.h](#).

8.27.2 Field Documentation

8.27.2.1 Vparam_AtomData* atomData

Array of Vparam_AtomData natom objects

Definition at line 114 of file [vparam.h](#).

8.27.2.2 char name[VMAX_ARGLEN]

Residue name

Definition at line 111 of file [vparam.h](#).

8.27.2.3 int nAtomData

Number of Vparam_AtomData objects associated with this object

Definition at line 112 of file [vparam.h](#).

8.27.2.4 Vmem* vmem

Pointer to memory manager from Vparam master class

Definition at line 110 of file [vparam.h](#).

The documentation for this struct was generated from the following file:

- src/generic/apbs/[vparam.h](#)

Chapter 9

File Documentation

9.1 doc/license/LICENSE.h File Reference

APBS license.

9.1.1 Detailed Description

APBS license.

Author

Nathan Baker

Version

Id:

[LICENSE.h](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*
```

```
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
```

Definition in file [LICENSE.h](#).

9.2 LICENSE.h

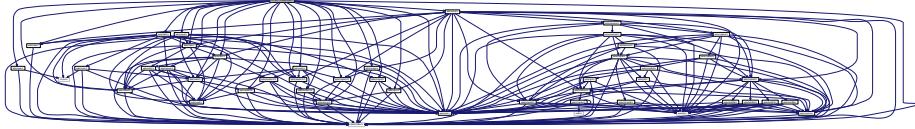
00001

9.3 src/aaa_inc/apbs/apbs.h File Reference

Top-level header for APBS.

```
#include "maloc/malloc.h"
#include "apbs/femparm.h"
#include "apbs/mgparm.h"
#include "apbs/nosh.h"
#include "apbs/pbeparm.h"
#include "apbs/vacc.h"
#include "apbs/valist.h"
#include "apbs/vatom.h"
#include "apbs/vcap.h"
#include "apbs/vhal.h"
#include "apbs/vpbe.h"
#include "apbs/vstring.h"
#include "apbs/vunit.h"
#include "apbs/vparam.h"
#include "apbs/vgreen.h"
#include "apbs/vgrid.h"
#include "apbs/vmgrid.h"
#include "apbs/vopot.h"
#include "apbs/vpmg.h"
#include "apbs/vpmgp.h"
#include "apbs/vfetk.h"
#include "apbs/vpee.h"
```

Include dependency graph for apbs.h:



9.3.1 Detailed Description

Top-level header for APBS.

Version

Id:

[apbs.h](#) 1750 2012-07-18 18:34:27Z tuckerbeck

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 [apbs.h](#).

9.4 apbs.h

```

00001
00073 #ifndef _APBS_H_
00074 #define _APBS_H_
00075

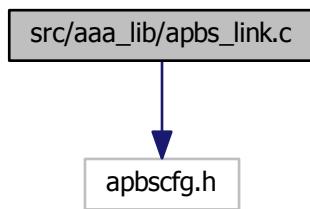
```

```
00076 /* MALOC headers */
00077 #include "malloc/malloc.h"
00078
00079 /* Generic headers */
00080 #include "apbs/femparm.h"
00081 #include "apbs/mgparm.h"
00082 #include "apbs/nosh.h"
00083 #include "apbs/pbeparm.h"
00084 #include "apbs/vacc.h"
00085 #include "apbs/valist.h"
00086 #include "apbs/vatom.h"
00087 #include "apbs/vcap.h"
00088 #include "apbs/vhal.h"
00089 #include "apbs/vpbe.h"
00090 #include "apbs/vstring.h"
00091 #include "apbs/vunit.h"
00092 #include "apbs/vparam.h"
00093
00094 #include "apbs/vgreen.h"
00095
00096 /* MG headers */
00097 #include "apbs/vgrid.h"
00098 #include "apbs/vmggrid.h"
00099 #include "apbs/vopot.h"
00100 #include "apbs/vpmq.h"
00101 #include "apbs/vpmgp.h"
00102
00103 /* FEM headers */
00104 #if defined(HAVE_MC_H)
00105 #include "apbs/vfetk.h"
00106 #include "apbs/vpee.h"
00107 #endif
00108
00109 #endif /* ifndef _APBS_H_ */
```

9.5 src/aaa_lib/apbs_link.c File Reference

Autoconf linkage assistance for packages built on top of APBS.

```
#include "apbscfg.h"  
Include dependency graph for apbs_link.c:
```



9.5.1 Detailed Description

Autoconf linkage assistance for packages built on top of APBS.

Author

Nathan Baker and Michael Holst

Version**Id:**

[apbs_link.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

Definition in file [apbs_link.c](#).

9.6 apbs_link.c

```

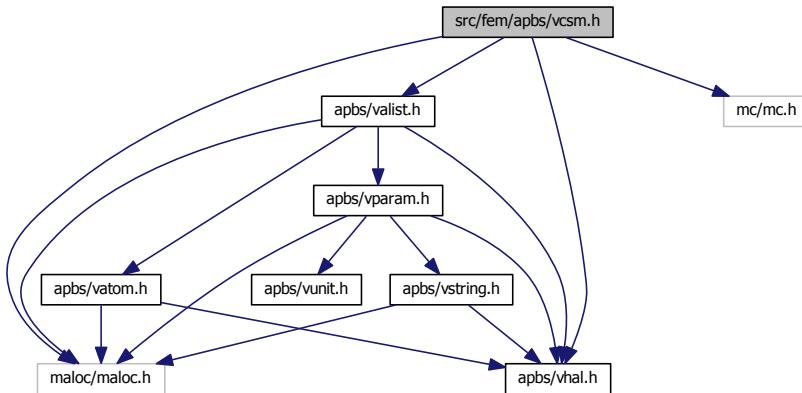
00001 #include "apbscfg.h"
00002
00058 #if defined(HAVE_MC_H)
00059     void apbs_needs_mc(void) { }
00060 #endif
00061 #if !defined(USE_PMG_BLAS)
00062     void apbs_needs_blas(void) { }
00063 #endif
00064
00065 void apbs_link(void)
00066 {
00067 }
00068

```

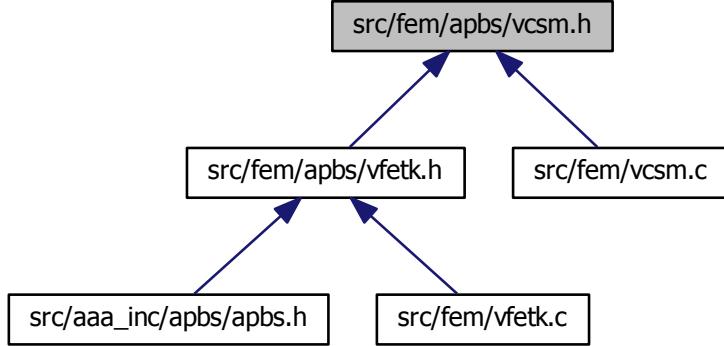
9.7 src/fem/apbs/vcsm.h File Reference

Contains declarations for the Vcsm class.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/valist.h"
#include "mc/mc.h"
Include dependency graph for vcsm.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [sVcsm](#)

Charge-simplex map class.

TypeDefs

- typedef struct [sVcsm](#) [Vcsm](#)

Declaration of the Vcsm class as the Vcsm structure.

Functions

- VEXTERNC void [Gem_setExternalUpdateFunction](#) ([Gem](#) *thee, void(*externalUpdate)([SS](#) **simps, int num))
External function for FEtk Gem class to use during mesh refinement.
- VEXTERNC [Valist](#) * [Vcsm_getValist](#) ([Vcsm](#) *thee)
Get atom list.
- VEXTERNC int [Vcsm_getNumberAtoms](#) ([Vcsm](#) *thee, int isimp)
Get number of atoms associated with a simplex.
- VEXTERNC [Vatom](#) * [Vcsm_getAtom](#) ([Vcsm](#) *thee, int iatom, int isimp)
Get particular atom associated with a simplex.
- VEXTERNC int [Vcsm_getAtomIndex](#) ([Vcsm](#) *thee, int iatom, int isimp)
Get ID of particular atom in a simplex.
- VEXTERNC int [Vcsm_getNumberSimplices](#) ([Vcsm](#) *thee, int iatom)
Get number of simplices associated with an atom.
- VEXTERNC [SS](#) * [Vcsm_getSimplex](#) ([Vcsm](#) *thee, int isimp, int iatom)
Get particular simplex associated with an atom.
- VEXTERNC int [Vcsm_getSimplexIndex](#) ([Vcsm](#) *thee, int isimp, int iatom)

- VEXTERNC unsigned long int [Vcsm_memChk](#) ([Vcsm](#) *thee)

Get index particular simplex associated with an atom.
- VEXTERNC [Vcsm](#) unsigned long int [Vcsm_memChk](#) ([Vcsm](#) *thee)

Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC [Vcsm](#) * [Vcsm_ctor](#) ([Valist](#) *alist, [Gem](#) *gm)

Construct Vcsm object.
- VEXTERNC int [Vcsm_ctor2](#) ([Vcsm](#) *thee, [Valist](#) *alist, [Gem](#) *gm)

FORTRAN stub to construct Vcsm object.
- VEXTERNC void [Vcsm_dtor](#) ([Vcsm](#) **thee)

Destroy Vcsm object.
- VEXTERNC void [Vcsm_dtor2](#) ([Vcsm](#) *thee)

FORTRAN stub to destroy Vcsm object.
- VEXTERNC void [Vcsm_init](#) ([Vcsm](#) *thee)

Initialize charge-simplex map with mesh and atom data.
- VEXTERNC int [Vcsm_update](#) ([Vcsm](#) *thee, [SS](#) **ssimps, int num)

Update the charge-simplex and simplex-charge maps after refinement.

9.7.1 Detailed Description

Contains declarations for the [Vcsm](#) class.

Version

Id:

[vcsm.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*

```

```

* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vcsm.h](#).

9.8 vcsm.h

```

00001
00063 #ifndef _VCSM_H_
00064 #define _VCSM_H_
00065
00066 /* Generic headers */
00067 #include "maloc/maloc.h"
00068 #include "apbs/vhal.h"
00069 #include "apbs/valist.h"
00070
00071 /* Specific headers */
00072 #include "mc/mc.h"
00073
00078 VEXTERNC void Gem_setExternalUpdateFunction(
00079     Gem *thee,
00080     void (*externalUpdate)(SS **simps, int num)
00083 );
00084
00089 struct sVcsm {
00090
00091     Valist *alist;
00092     int natom;
00094     Gem *gm;
00097     int **sqm;
00104     int *nsqm;
00105     int nsimp;
00107     int msimp;
00109     int **qsm;
00111     int *nqsm;
00112     int initFlag;
00114     Vmem *vmem;
00116 };
00117
00122 typedef struct sVcsm Vcsm;
00123
00124 /* //////////////////////////////// Class Vcsm: Inlineable methods (vcsm.c)
00125 // Class Vcsm: Inlineable methods (vcsm.c)
00127
00128 #if !defined(VINLINE_VCSM)
00129
00135     VEXTERNC Valist* Vcsm_getValist(
00136         Vcsm *thee
00137     );

```

```

00138
00144     VEXTERNC int Vcsm_getNumberAtoms(
00145         Vcsm *thee,
00146         int isimp
00147     );
00148
00154     VEXTERNC Vatom* Vcsm_getAtom(
00155         Vcsm *thee,
00156         int iatom,
00157         int isimp
00158     );
00159
00165     VEXTERNC int Vcsm_getAtomIndex(
00166         Vcsm *thee,
00167         int iatom,
00168         int isimp
00169     );
00170
00176     VEXTERNC int Vcsm_getNumberSimplices(
00177         Vcsm *thee,
00178         int iatom
00179     );
00180
00186     VEXTERNC SS* Vcsm_getSimplex(
00187         Vcsm *thee,
00188         int isimp,
00189         int iatom
00190     );
00191
00197     VEXTERNC int Vcsm_getSimplexIndex(
00198         Vcsm *thee,
00199         int isimp,
00200         int iatom
00201     );
00202
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)->nsgm[isimp])
00216 #    define Vcsm_getAtom(thee, iatom, isimp) (Valist_getAtom((thee)->alist,
00217 #        ((thee)->sqm)[isimp][iatom]))
00218 #    define Vcsm_getAtomIndex(thee, iatom, isimp) (((thee)->sqm)[isimp][iatom])
00219 #    define Vcsm_getNumberSimplices(thee, iatom) (((thee)->nqsm)[iatom])
00220 #    define Vcsm_getSimplex(thee, isimp, iatom) (Gem_SS((thee)->gm,
00221 #        ((thee)->qsm)[iatom][isimp]))
00222 #    define Vcsm_getSimplexIndex(thee, isimp, iatom)
00223 #        (((thee)->qsm)[iatom][isimp])
00224 #    define Vcsm_memChk(thee) (Vmem_bytes((thee)->vmem))
00225 #endif /* if !defined(VINLINE_VCSM) */
00226
00227 /* //////////////////////////////// */
00228 // Class Vcsm: Non-Inlineable methods (vcsm.c)
00229
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,

```

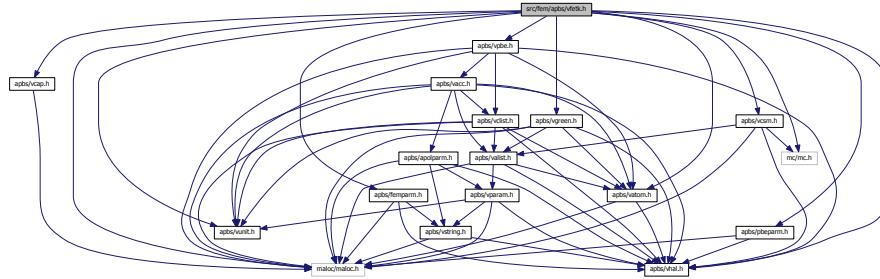
```
00289     SS **simpss,
00294     int num
00295 );
00296
00297 #endif /* ifndef _VCSM_H_ */
```

9.9 src/fem/apbs/vfetk.h File Reference

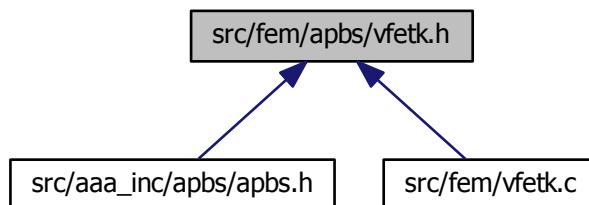
Contains declarations for class Vfetk.

```
#include "maloc/maloc.h"
#include "mc/mc.h"
#include "apbs/vhal.h"
#include "apbs/vatom.h"
#include "apbs/vcsm.h"
#include "apbs/vpbe.h"
#include "apbs/vunit.h"
#include "apbs/vgreen.h"
#include "apbs/vcap.h"
#include "apbs/pbeparm.h"
#include "apbs/femparm.h"
Include dependency graph for vfetk.h
```

Include dependency graph for vfetk.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `sVfetk`
Contains public data members for Vfetk class/module.
- struct `sVfetk_LocalVar`
Vfetk LocalVar subclass.

Typedefs

- typedef enum `eVfetk_LsolvType` `Vfetk_LsolvType`
Declare FEMparm_LsolvType type.
- typedef enum `eVfetk_MeshLoad` `Vfetk_MeshLoad`
Declare FEMparm_GuessType type.
- typedef enum `eVfetk_NsolvType` `Vfetk_NsolvType`
Declare FEMparm_NsolvType type.
- typedef enum `eVfetk_GuessType` `Vfetk_GuessType`
Declare FEMparm_GuessType type.
- typedef enum `eVfetk_PrecType` `Vfetk_PrecType`
Declare FEMparm_GuessType type.
- typedef struct `sVfetk` `Vfetk`
Declaration of the Vfetk class as the Vfetk structure.
- typedef struct `sVfetk_LocalVar` `Vfetk_LocalVar`
Declaration of the Vfetk_LocalVar subclass as the Vfetk_LocalVar structure.

Enumerations

- enum `eVfetk_LsolvType` { `VLT_SLU` = 0, `VLT_MG` = 1, `VLT(CG` = 2, `VLT_BCG` = 3 }
Linear solver type.
- enum `eVfetk_MeshLoad` { `VML_DIRICUBE`, `VML_NEUMCUBE`, `VML_EXTERNAL` }
Mesh loading operation.
- enum `eVfetk_NsolvType` { `VNT_NEW` = 0, `VNT_INC` = 1, `VNT_ARC` = 2 }
Non-linear solver type.
- enum `eVfetk_GuessType` { `VGT_ZERO` = 0, `VGT_DIRI` = 1, `VGT_PREV` = 2 }
Initial guess type.
- enum `eVfetk_PrecType` { `VPT_IDEN` = 0, `VPT_DIAG` = 1, `VPT_MG` = 2 }
Preconditioner type.

Functions

- VEXTERNC `Gem` * `Vfetk_getGem` (`Vfetk` *`thee`)
Get a pointer to the Gem (grid manager) object.
- VEXTERNC `AM` * `Vfetk_getAM` (`Vfetk` *`thee`)
Get a pointer to the AM (algebra manager) object.
- VEXTERNC `Vpbe` * `Vfetk_getVpbe` (`Vfetk` *`thee`)
Get a pointer to the Vpbe (PBE manager) object.
- VEXTERNC `Vcsm` * `Vfetk_getVcsm` (`Vfetk` *`thee`)
Get a pointer to the Vcsm (charge-simplex map) object.

- VEXTERNC int `Vfetk_getAtomColor (Vfetk *thee, int iatom)`
Get the partition information for a particular atom.
- VEXTERNC `Vfetk * Vfetk_ctor (Vpbe *pbe, Vhal_PBEType type)`
Constructor for Vfetk object.
- VEXTERNC int `Vfetk_ctor2 (Vfetk *thee, Vpbe *pbe, Vhal_PBEType type)`
FORTRAN stub constructor for Vfetk object.
- VEXTERNC void `Vfetk_dtor (Vfetk **thee)`
Object destructor.
- VEXTERNC void `Vfetk_dtor2 (Vfetk *thee)`
FORTRAN stub object destructor.
- VEXTERNC double * `Vfetk_getSolution (Vfetk *thee, int *length)`
Create an array containing the solution (electrostatic potential in units of $k_B T / e$) at the finest mesh level.
- VEXTERNC void `Vfetk_setParameters (Vfetk *thee, PBEparm *pbeparm, FEMparm *feparm)`
Set the parameter objects.
- VEXTERNC double `Vfetk_energy (Vfetk *thee, int color, int nonlin)`
Return the total electrostatic energy.
- VEXTERNC double `Vfetk_dqmEnergy (Vfetk *thee, int color)`
Get the "mobile charge" and "polarization" contributions to the electrostatic energy.
- VEXTERNC double `Vfetk_qfEnergy (Vfetk *thee, int color)`
Get the "fixed charge" contribution to the electrostatic energy.
- VEXTERNC unsigned long int `Vfetk_memChk (Vfetk *thee)`
Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC void `Vfetk_setAtomColors (Vfetk *thee)`
Transfer color (partition ID) information from a partitioned mesh to the atoms.
- VEXTERNC void `Bmat_printHB (Bmat *thee, char *fname)`
Writes a Bmat to disk in Harwell-Boeing sparse matrix format.
- VEXTERNC Vrc_Codes `Vfetk_genCube (Vfetk *thee, double center[3], double length[3], Vfetk_MeshLoad meshType)`
Construct a rectangular mesh (in the current Vfetk object)
- VEXTERNC Vrc_Codes `Vfetk_loadMesh (Vfetk *thee, double center[3], double length[3], Vfetk_MeshLoad meshType, Vio *sock)`
Loads a mesh into the Vfetk (and associated) object(s).
- VEXTERNC PDE * `Vfetk_PDE_ctor (Vfetk *fetk)`
Constructs the FEtk PDE object.
- VEXTERNC int `Vfetk_PDE_ctor2 (PDE *thee, Vfetk *fetk)`
Initializes the FEtk PDE object.
- VEXTERNC void `Vfetk_PDE_dtor (PDE **thee)`
Destroys FEtk PDE object.
- VEXTERNC void `Vfetk_PDE_dtor2 (PDE *thee)`
FORTRAN stub: destroys FEtk PDE object.
- VEXTERNC void `Vfetk_PDE_initAssemble (PDE *thee, int ip[], double rp[])`
Do once-per-assembly initialization.
- VEXTERNC void `Vfetk_PDE_initElement (PDE *thee, int elementType, int chart, double tvx[][VAPBS_DIM], void *data)`
Do once-per-element initialization.
- VEXTERNC void `Vfetk_PDE_initFace (PDE *thee, int faceType, int chart, double tvec[])`
Do once-per-face initialization.

- VEXTERNC void [Vfetk_PDE_initPoint](#) (PDE *thee, int pointType, int chart, double txq[], double tU[], double tU[][[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.

9.9.1 Detailed Description

Contains declarations for class [Vfetk](#).

Version

Id:

[vfetk.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.

```

```

*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vfetk.h](#).

9.10 vfetk.h

```

00001
00062 #ifndef _VFETK_H_
00063 #define _VFETK_H_
00064
00065 #include "maloc/maloc.h"
00066 #include "mc/mc.h"
00067 #include "apbs/vhal.h"
00068 #include "apbs/vatom.h"
00069 /* #include "apbs/valist.h" */
00070 #include "apbs/vcsm.h"
00071 #include "apbs/vpbe.h"
00072 #include "apbs/vunit.h"
00073 #include "apbs/vgreen.h"
00074 #include "apbs/vcap.h"
00075 #include "apbs/pbeparm.h"
00076 #include "apbs/femparm.h"
00077
00083 enum eVfetk_LsolvType {
00084     VLT_SLU=0,
00085     VLT_MG=1,
00086     VLT(CG)=2,
00087     VLT_BCG=3
00088 };
00089
00094 typedef enum eVfetk_LsolvType Vfetk_LsolvType;
00095
00096
00101 enum eVfetk_MeshLoad {
00102     VML_DIRICUBE,
00103     VML_NEUMCUBE,
00104     VML_EXTERNAL
00105 };
00106
00111 typedef enum eVfetk_MeshLoad Vfetk_MeshLoad;
00112
00118 enum eVfetk_NsolvType {
00119     VNT_NEW=0,
00120     VNT_INC=1,

```

```

00121     VNT_ARC=2
00122 };
00123
00128 typedef enum eVfetk_NsolvType Vfetk_NsolvType;
00129
00135 enum eVfetk_GuessType {
00136     VGT_ZERO=0,
00137     VGT_DIRI=1,
00138     VGT_PREV=2
00139 };
00140
00145 typedef enum eVfetk_GuessType Vfetk_GuessType;
00146
00152 enum eVfetk_PrecType {
00153     VPT_IDEN=0,
00154     VPT_DIAG=1,
00155     VPT_MG=2
00156 };
00157
00162 typedef enum eVfetk_PrecType Vfetk_PrecType;
00163
00173 struct sVfetk {
00174
00175     Vmem *vmem;
00176     Gem *gm;
00179     AM *am;
00180     Aprx *aprx;
00181     PDE *pde;
00182     Vpbe *pbe;
00183     Vcsm *csm;
00184     Vfetk_LsolvType lkey;
00185     int lmax;
00186     double ltol;
00187     Vfetk_NsolvType nkey;
00188     int nmax;
00189     double ntol;
00190     Vfetk_GuessType gues;
00191     Vfetk_PrecType lprec;
00192     int pjac;
00194     PBEParm *pbeparm;
00195     FEMparm *feparm;
00196     Vhal_PBEType type;
00197     int level;
00199 };
00200
00204 typedef struct sVfetk Vfetk;
00205
00212 struct sVfetk_LocalVar {
00213     double nvec[VAPBS_DIM];
00214     double vx[4][VAPBS_DIM];
00215     double xq[VAPBS_DIM];
00216     double U[MAXV];
00217     double dU[MAXV][VAPBS_DIM];
00218     double W;
00219     double dW[VAPBS_DIM];
00220     double d2W;
00221     int sType;
00222     int fType;
00223     double diel;
00224     double ionacc;
00225     double A;
00226     double F;
00227     double B;
00228     double DB;
00229     double jumpDiel;
00230     Vfetk *fetk;
00231     Vgreen *green;
00232     int initGreen;
00234     SS *simp;
00236     VV *verts[4];
00237     int nverts;
00238     double ionConc[MAXION];
00239     double ionQ[MAXION];
00240     double ionRadii[MAXION];
00241     double zkappa2;
00242     double zks2;
00243     double ionstr;
00244     int nion;
00245     double Fu_v;
00246     double DFu_wv;
00247     double delta;

```

```

00248     double u_D;
00249     double u_T;
00250 };
00251
00256 typedef struct sVfetk_LocalVar Vfetk_LocalVar;
00257
00258 #if !defined(VINLINE_VFETK)
00259
00265     VEXTERNC Gem* Vfetk_getGem(
00266         Vfetk *thee
00267     );
00268
00274     VEXTERNC AM* Vfetk_getAM(
00275         Vfetk *thee
00276     );
00277
00283     VEXTERNC Vpbe* Vfetk_getVpbe(
00284         Vfetk *thee
00285     );
00286
00292     VEXTERNC Vcsm* Vfetk_getVcsm(
00293         Vfetk *thee
00294     );
00295
00302     VEXTERNC int Vfetk_getAtomColor(
00303         Vfetk *thee,
00304         int iatom
00305     );
00306
00307 /*else /* if defined(VINLINE_VFETK) */
00308 #   define Vfetk_getGem(thee) ((thee)->gm)
00309 #   define Vfetk_getAM(thee) ((thee)->am)
00310 #   define Vfetk_getVpbe(thee) ((thee)->pbe)
00311 #   define Vfetk_getVcsm(thee) ((thee)->csm)
00312 #   define Vfetk_getAtomColor(thee, iatom)
00313     (Vatom_getPartID(Valist_getAtom(Vpbe_getValist(thee->pbe), iatom)))
00314 #endif /* if !defined(VINLINE_VFETK) */
00315 /* //////////////////////////////// */
00316 // Class Vfetk: Non-Inlineable methods (vfetk.c)
00318
00328 VEXTERNC Vfetk* Vfetk_ctor(
00329     Vpbe *pbe,
00330     Vhal_PBEType type
00331 );
00332
00342 VEXTERNC int Vfetk_ctor2(
00343     Vfetk *thee,
00344     Vpbe *pbe,
00345     Vhal_PBEType type
00346 );
00347
00353 VEXTERNC void Vfetk_dtor(
00354     Vfetk **thee
00355 );
00356
00362 VEXTERNC void Vfetk_dtor2(
00363     Vfetk *thee
00364 );
00365
00375 VEXTERNC double* Vfetk_getSolution(
00376     Vfetk *thee,
00377     int *length
00378 );
00379
00385 VEXTERNC void Vfetk_setParameters(
00386     Vfetk *thee,
00387     PBEparm *pbeparm,
00388     FEMparm *feparm
00389 );
00390
00409 VEXTERNC double Vfetk_energy(
00410     Vfetk *thee,
00411     int color,
00415     int nonlin
00417 );
00418
00448 VEXTERNC double Vfetk_dqmEnergy(
00449     Vfetk *thee,
00450     int color
00454 );

```

```

00455
00473 VEXTERNC double Vfetk_qfEnergy(
00474     Vfetk *thee,
00475     int color
00477 );
00478
00486 VEXTERNC unsigned long int Vfetk_memChk(
00487     Vfetk *thee
00488 );
00489
00505 VEXTERNC void Vfetk_setAtomColors(
00506     Vfetk *thee
00507 );
00508
00517 VEXTERNC void Bmat_printHB(
00518     Bmat *thee,
00519     char *fname
00520 );
00521
00527 VEXTERNC Vrc_Codes Vfetk_genCube(
00528     Vfetk *thee,
00529     double center[3],
00530     double length[3],
00531     Vfetk_MeshLoad meshType
00532 );
00533
00539 VEXTERNC Vrc_Codes Vfetk_loadMesh(
00540     Vfetk *thee,
00541     double center[3],
00542     double length[3],
00543     Vfetk_MeshLoad meshType,
00544     Vio *sock
00545 );
00546
00553 VEXTERNC PDE* Vfetk_PDE_ctor(
00554     Vfetk *fetk
00555 );
00556
00563 VEXTERNC int Vfetk_PDE_ctor2(
00564     PDE *thee,
00565     Vfetk *fetk
00566 );
00567
00574 VEXTERNC void Vfetk_PDE_dtor(
00575     PDE **thee
00576 );
00577
00584 VEXTERNC void Vfetk_PDE_dtor2(
00585     PDE *thee
00586 );
00587
00593 VEXTERNC void Vfetk_PDE_initAssemble(
00594     PDE *thee,
00595     int ip[],
00596     double rp[]
00597 );
00598
00605 VEXTERNC void Vfetk_PDE_initElement(
00606     PDE *thee,
00607     int elementType,
00608     int chart,
00611     double tvx[][VAPBS_DIM],
00612     void *data
00613 );
00614
00620 VEXTERNC void Vfetk_PDE_initFace(
00621     PDE *thee,
00622     int faceType,
00624     int chart,
00626     double tnvec[]
00627 );
00628
00636 VEXTERNC void Vfetk_PDE_initPoint(
00637     PDE *thee,
00638     int pointType,
00639     int chart,
00641     double txq[],
00642     double tU[],
00643     double tdU[][VAPBS_DIM]
00644 );
00645

```

```
00663 VEXTERNC void Vfetk_PDE_Fu(
00664     PDE *thee,
00665     int key,
00666     double F[]
00667 );
00669
00680 VEXTERNC double Vfetk_PDE_Fu_v(
00681     PDE *thee,
00682     int key,
00683     double V[],
00684     double dV[] [VAPBS_DIM]
00685 );
00687
00699 VEXTERNC double Vfetk_PDE_DFu_wv(
00700     PDE *thee,
00701     int key,
00702     double W[],
00703     double dW[] [VAPBS_DIM],
00704     double V[],
00705     double dV[] [VAPBS_DIM]
00706 );
00708
00715 VEXTERNC void Vfetk_PDE_delta(
00716     PDE *thee,
00717     int type,
00718     int chart,
00719     double txq[],
00720     void *user,
00721     double F[]
00722 );
00723
00731 VEXTERNC void Vfetk_PDE_u_D(
00732     PDE *thee,
00733     int type,
00734     int chart,
00735     double txq[],
00736     double F[]
00737 );
00738
00746 VEXTERNC void Vfetk_PDE_u_T(
00747     PDE *thee,
00748     int type,
00749     int chart,
00750     double txq[],
00751     double F[]
00752 );
00753
00759 VEXTERNC void Vfetk_PDE_bisectEdge(
00760     int dim,
00761     int dimII,
00762     int edgeType,
00763     int chart[],
00764     double vx[] [VAPBS_DIM]
00765 );
00766
00767
00773 VEXTERNC void Vfetk_PDE_mapBoundary(
00774     int dim,
00775     int dimII,
00776     int vertexType,
00777     int chart,
00778     double vx[VAPBS_DIM]
00779 );
00780
00789 VEXTERNC int Vfetk_PDE_markSimplex(
00790     int dim,
00791     int dimII,
00792     int simplexType,
00793     int faceType[VAPBS_NVs],
00794     int vertexType[VAPBS_NVs],
00795     int chart[],
00796     double vx[] [VAPBS_DIM],
00797     void *simplex
00798 );
00799
00805 VEXTERNC void Vfetk_PDE_oneChart(
00806     int dim,
00807     int dimII,
00808     int objType,
00809     int chart[],
00810     double vx[] [VAPBS_DIM],
00811     int dimV
```

```

00812         );
00813
00823 VEXTERNC double Vfetk_PDE_Ju(
00824     PDE *thee,
00825     int key
00826 );
00827
00835 VEXTERNC void Vfetk_externalUpdateFunction(
00836     SS **simps,
00838     int num
00839 );
00840
00841
00904 VEXTERNC int Vfetk_PDE_simplexBasisInit(
00905     int key,
00907     int dim,
00908     int comp,
00910     int *ndof,
00911     int dof[]
00912 );
00913
00921 VEXTERNC void Vfetk_PDE_simplexBasisForm(
00922     int key,
00924     int dim,
00925     int comp ,
00926     int pdkey,
00935     double xq[],
00936     double basis[]
00938 );
00939
00945 VEXTERNC void Vfetk_readMesh(
00946     Vfetk *thee,
00947     int skey,
00948     Vio *sock
00949 );
00950
00956 VEXTERNC void Vfetk_dumpLocalVar();
00957
00965 VEXTERNC int Vfetk_fillArray(
00966     Vfetk *thee,
00967     Bvec *vec,
00968     Vdata_Type type
00969 );
00970
00985 VEXTERNC int Vfetk_write(
00986     Vfetk *thee,
00987     const char *iodev,
00989     const char *iofmt,
00991     const char *thost,
00992     const char *fname,
00993     Bvec *vec,
00994     Vdata_Format format
00995 );
00996
01002 VEXTERNC Vrc_Codes Vfetk_loadGem(
01003     Vfetk *thee,
01004     Gem *gm
01005 );
01006
01007
01008 #endif /* ifndef _VFETK_H_ */

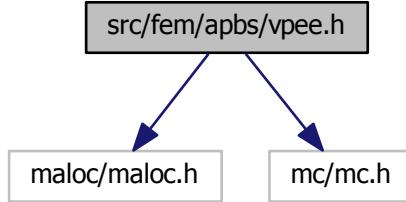
```

9.11 src/fem/apbs/vpee.h File Reference

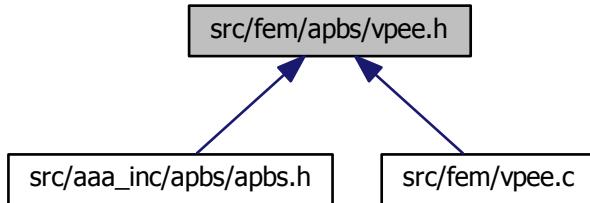
Contains declarations for class Vpee.

```
#include "maloc/maloc.h"
#include "mc/mc.h"
```

Include dependency graph for vpee.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [sVpee](#)

Contains public data members for Vpee class/module.

TypeDefs

- typedef struct [sVpee](#) [Vpee](#)

Declaration of the Vpee class as the Vpee structure.

Functions

- VEXTERNC [Vpee](#) * [Vpee_ctor](#) (Gem *gm, int localPartID, int killFlag, double killParam)
Construct the Vpee object.
- VEXTERNC int [Vpee_ctor2](#) ([Vpee](#) *thee, Gem *gm, int localPartID, int killFlag, double killParam)
FORTRAN stub to construct the Vpee object.

- VEXTERNC void [Vpee_dtor](#) ([Vpee](#) **thee)
Object destructor.
- VEXTERNC void [Vpee_dtor2](#) ([Vpee](#) *thee)
FORTRAN stub object destructor.
- VEXTERNC int [Vpee_markRefine](#) ([Vpee](#) *thee, AM *am, int level, int akey, int rcol, double etol, int bkey)
Mark simplices for refinement based on attenuated error estimates.
- VEXTERNC int [Vpee_numSS](#) ([Vpee](#) *thee)
Returns the number of simplices in the local partition.

9.11.1 Detailed Description

Contains declarations for class Vpee.

Version

Id:

[vpee.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.

```

* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.

*

*

Definition in file [vpee.h](#).

9.12 vpee.h

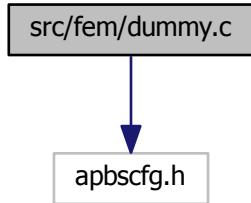
```
00001
00076 #ifndef _VPEE_H
00077 #define _VPEE_H
00078
00079 /* Generic headers */
00080 #include "malloc/malloc.h"
00081 #include "mc/mc.h"
00082
00088 struct sVpee {
00089
00090     Gem *gm;
00091     int localPartID;
00094     double localPartCenter[3];
00096     double localPartRadius;
00098     int killFlag;
00101     double killParam;
00103     Vmem *mem;
00105 };
00106
00111 typedef struct sVpee Vpee;
00112
00113 /* //////////////////////////////// */
00114 // Class Vpee Inlineable methods
00116
00117 #if !defined(VINLINE_VPEE)
00118 #else /* if defined(VINLINE_VPEE) */
00119 #endif /* if !defined(VINLINE_VPEE) */
00120
00121 /* //////////////////////////////// */
00122 // Class Vpee: Non-Inlineable methods (vpee.c)
00124
00131 VEXTERNC Vpee* Vpee_ctor(
00132     Gem *gm,
00133     int localPartID,
00134     int killFlag,
00145     double killParam
00146 );
00147
00154 VEXTERNC int Vpee_ctor2(
00155     Vpee *thee,
00156     Gem *gm,
00157     int localPartID,
00158     int killFlag,
00169     double killParam
00170 );
00171
00176 VEXTERNC void Vpee_dtor(
00177     Vpee **thee
00178 );
00179
00184 VEXTERNC void Vpee_dtor2(
00185     Vpee *thee
00186 );
00187
```

```
00203 VEXTERNC int Vpee_markRefine(
00204     Vpee *thee,
00205     AM *am,
00206     int level,
00207     int akey,
00215     int rcol,
00218     double etol,
00219     int bkey
00223 );
00224
00230 VEXTERNC int Vpee_numSS(
00231     Vpee *thee
00232 );
00233
00234 #endif /* ifndef _VPEE_H_ */
```

9.13 src/fem/dummy.c File Reference

Give libtool something to do.

```
#include "apbscfg.h"
Include dependency graph for dummy.c:
```



9.13.1 Detailed Description

Give libtool something to do.

Author

Nathan Baker

Version

Id:

[dummy.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

Definition in file [dummy.c](#).

9.14 dummy.c

```
00001  
00056 #include "apbscfg.h"  
00057  
00058 int APBSFEM_dummy(int i) {  
00059     int j;  
00060     j = i;  
00061     return j;  
00062 }  
00063  
00064  
00065 }
```

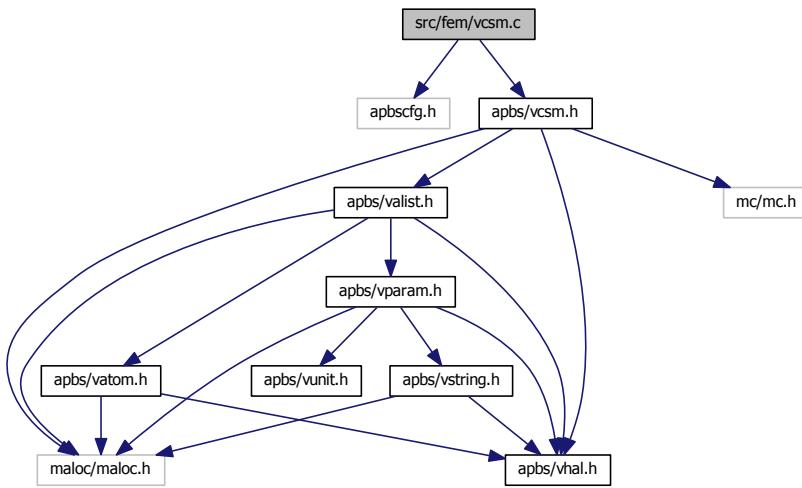
9.15 src/fem/vcsm.c File Reference

Class Vcsm methods.

```
#include "apbscfg.h"
```

```
#include "apbs/vcsm.h"
```

Include dependency graph for vcsm.c:



Functions

- VPUBLIC [Valist * Vcsm_getValist \(Vcsm *thee\)](#)
Get atom list.
- VPUBLIC int [Vcsm_getNumberAtoms \(Vcsm *thee, int isimp\)](#)
Get number of atoms associated with a simplex.
- VPUBLIC [Vatom * Vcsm_getAtom \(Vcsm *thee, int iatom, int isimp\)](#)
Get particular atom associated with a simplex.
- VPUBLIC int [Vcsm_getAtomIndex \(Vcsm *thee, int iatom, int isimp\)](#)
Get ID of particular atom in a simplex.
- VPUBLIC int [Vcsm_getNumberSimplices \(Vcsm *thee, int iatom\)](#)
Get number of simplices associated with an atom.
- VPUBLIC SS * [Vcsm_getSimplex \(Vcsm *thee, int isimp, int iatom\)](#)
Get particular simplex associated with an atom.
- VPUBLIC int [Vcsm_getSimplexIndex \(Vcsm *thee, int isimp, int iatom\)](#)
Get index particular simplex associated with an atom.
- VPUBLIC unsigned long int [Vcsm_memChk \(Vcsm *thee\)](#)
Return the memory used by this structure (and its contents) in bytes.
- VPUBLIC [Vcsm * Vcsm_ctor \(Valist *alist, Gem *gm\)](#)
Construct Vcsm object.
- VPUBLIC int [Vcsm_ctor2 \(Vcsm *thee, Valist *alist, Gem *gm\)](#)
FORTRAN stub to construct Vcsm object.

- VPUBLIC void `Vcsm_init` (`Vcsm` *thee)
Initialize charge-simplex map with mesh and atom data.
- VPUBLIC void `Vcsm_dtor` (`Vcsm` **thee)
Destroy Vcsm object.
- VPUBLIC void `Vcsm_dtor2` (`Vcsm` *thee)
FORTRAN stub to destroy Vcsm object.
- VPUBLIC int `Vcsm_update` (`Vcsm` *thee, `SS` **simps, int num)
Update the charge-simplex and simplex-charge maps after refinement.

9.15.1 Detailed Description

Class `Vcsm` methods.

Author

Nathan Baker

Version

Id:

`vcsm.c` 1750 2012-07-18 18:34:27Z tuckerbeck

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*

```

```

* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vcsm.c](#).

9.16 vcsm.c

```

00001
00058 #include "apbscfg.h"
00059
00060 #if defined(HAVE_MC_H)
00061 #include "apbs/vcsm.h"
00062
00063 /* Inlineable methods */
00064 #if !defined(VINLINE_VCSM)
00065
00066 VPUBLIC Valist* Vcsm_getValist(Vcsm *thee) {
00067
00068     VASSERT(thee != VNULL);
00069     return thee->alist;
00070
00071 }
00072
00073 VPUBLIC int Vcsm_getNumberAtoms(Vcsm *thee, int isimp) {
00074
00075     VASSERT(thee != VNULL);
00076     VASSERT(thee->initFlag);
00077     return thee->nsqm[isimp];
00078
00079 }
00080
00081 VPUBLIC Vatom* Vcsm_getAtom(Vcsm *thee, int iatom, int
00082     isimp) {
00083
00084     VASSERT(thee != VNULL);
00085     VASSERT(thee->initFlag);
00086
00087     VASSERT(iatom < (thee->nsqm)[isimp]);
00088     return Valist_getAtom(thee->alist, (thee->sqm)[isimp][
00089         iatom]);
00090
00091
00092 VPUBLIC int Vcsm_getAtomIndex(Vcsm *thee, int iatom, int
00093     isimp) {
00094
00095     VASSERT(thee != VNULL);
00096     VASSERT(thee->initFlag);
00097
00098     VASSERT(iatom < (thee->nsqm)[isimp]);
00099     return (thee->sqm)[isimp][iatom];
00100
00101 }
00102
00103 VPUBLIC int Vcsm_getNumberSimplices(Vcsm *thee, int
00104     iatom) {
00105
00106     VASSERT(thee != VNULL);
00107     VASSERT(thee->initFlag);

```

```

00108     return (thee->ngsm)[iatom];
00109 }
00110 }
00111 }
00112
00113 VPUBLIC SS* Vcsm_getSimplex(Vcsm *thee, int isimp, int iatom
00114 ) {
00115
00116     VASSERT(thee != VNULL);
00117     VASSERT(thee->initFlag);
00118
00119     return Gem_SS(thee->gm, (thee->qsm)[iatom][isimp]);
00120
00121 }
00122
00123 VPUBLIC int Vcsm_getSimplexIndex(Vcsm *thee, int isimp,
00124     int iatom) {
00125
00126     VASSERT(thee != VNULL);
00127     VASSERT(thee->initFlag);
00128
00129     return (thee->qsm)[iatom][isimp];
00130
00131 }
00132
00133 VPUBLIC unsigned long int Vcsm_memChk(Vcsm *thee) {
00134     if (thee == VNULL) return 0;
00135     return Vmem_bytes(thee->vmem);
00136 }
00137
00138 #endif /* if !defined(VINLINE_VCSM) */
00139
00140 VPUBLIC Vcsm* Vcsm_ctor(Valist *alist, Gem *gm) {
00141
00142     /* Set up the structure */
00143     Vcsm *thee = VNULL;
00144     thee = (Vcsm*)Vmem_malloc(VNULL, 1, sizeof(Vcsm));
00145     VASSERT( thee != VNULL );
00146     VASSERT( Vcsm_ctor2(thee, alist, gm) );
00147
00148     return thee;
00149 }
00150
00151 VPUBLIC int Vcsm_ctor2(Vcsm *thee, Valist *alist, Gem *gm
00152 ) {
00153
00154     VASSERT( thee != VNULL );
00155
00156     /* Memory management object */
00157     thee->vmem = Vmem_ctor("APBS:VCSM");
00158
00159     /* Set up the atom list and grid manager */
00160     if( alist == VNULL) {
00161         Vnm_print(2,"Vcsm_ctor2: got null pointer to Valist object!\n");
00162         return 0;
00163     }
00164     thee->alist = alist;
00165     if( gm == VNULL) {
00166         Vnm_print(2,"Vcsm_ctor2: got a null pointer to the Gem object!\n");
00167         return 0;
00168     }
00169     thee->gm = gm;
00170
00171     thee->initFlag = 0;
00172     return 1;
00173 }
00174
00175 VPUBLIC void Vcsm_init(Vcsm *thee) {
00176
00177     /* Counters */
00178     int iatom,
00179         jatom,
00180         isimp,
00181         jsimp,
00182         gotSimp;
00183
00184     /* Atomic information */
00185     Vatom *atom;
00186     double *position;
00187
00188     /* Simplex/Vertex information */

```

```

00186     SS *simplex;
00187     /* Basis function values */
00188
00189     if (thee == VNULL) {
00190         Vnm_print(2, "Vcsm_init: Error! Got NULL thee!\n");
00191         VASSERT(0);
00192     }
00193     if (thee->gm == VNULL) {
00194         Vnm_print(2, "Vcsm_init: Error! Got NULL thee->gm!\n");
00195         VASSERT(0);
00196     }
00197     thee->nssimp = Gem_numSS(thee->gm);
00198     if (thee->nssimp <= 0) {
00199         Vnm_print(2, "Vcsm_init: Error! Got %d simplices!\n", thee->nssimp
00200     );
00201         VASSERT(0);
00202     }
00203     thee->natom = Valist_getNumberAtoms(thee->alist
00204 );
00205
00206     /* Allocate and initialize space for the first dimensions of the
00207      * simplex-charge map, the simplex array, and the counters */
00208     thee->sqm = (int**)Vmem_malloc(thee->vmem, thee->nssimp, sizeof(
00209     int *));
00210     VASSERT(thee->sqm != VNULL);
00211     thee->nqsm = (int*)Vmem_malloc(thee->vmem, thee->natom, sizeof(
00212     int));
00213     VASSERT(thee->nqsm != VNULL);
00214     for (isimp=0; isimp<thee->nssimp; isimp++) (thee->sqm)[isimp] = 0;
00215
00216     /* Count the number of charges per simplex. */
00217     for (iatom=0; iatom<thee->natom; iatom++) {
00218         atom = Valist_getAtom(thee->alist, iatom);
00219         position = Vatom_getPosition(atom);
00220         gotSimp = 0;
00221         for (isimp=0; isimp<thee->nssimp; isimp++) {
00222             simplex = Gem_SS(thee->gm, isimp);
00223             if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00224                 (thee->sqm)[isimp]++;
00225                 gotSimp = 1;
00226             }
00227         }
00228     }
00229
00230     /* @todo Combine the following two loops? - PCE */
00231     /* Allocate the space for the simplex-charge map */
00232     for (isimp=0; isimp<thee->nssimp; isimp++) {
00233         if ((thee->sqm)[isimp] > 0) {
00234             thee->sqm[isimp] = (int)Vmem_malloc(thee->vmem, (thee->natom
00235 )[isimp],
00236             sizeof(int));
00237             VASSERT(thee->sqm[isimp] != VNULL);
00238         }
00239
00240     /* Finally, set up the map */
00241     for (isimp=0; isimp<thee->nssimp; isimp++) {
00242         jsimp = 0;
00243         simplex = Gem_SS(thee->gm, isimp);
00244         for (iatom=0; iatom<thee->natom; iatom++) {
00245             atom = Valist_getAtom(thee->alist, iatom);
00246             position = Vatom_getPosition(atom);
00247             /* Check to see if the atom's in this simplex */
00248             if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00249                 /* Assign the entries in the next vacant spot */
00250                 (thee->sqm)[isimp][jsimp] = iatom;
00251                 jsimp++;
00252             }
00253         }
00254     }
00255
00256     /* Allocate space for the charge-simplex map */
00257     thee->qsm = (int**)Vmem_malloc(thee->vmem, thee->natom, sizeof(
00258     int *));
00259     VASSERT(thee->qsm != VNULL);
00260     thee->nqsm = (int*)Vmem_malloc(thee->vmem, thee->natom, sizeof(
00261     int));
00262     VASSERT(thee->nqsm != VNULL);
00263     for (iatom=0; iatom<thee->natom; iatom++) (thee->nqsm)[iatom] = 0;

```

```

00260     /* Loop through the list of simplices and count the number of times
00261      * each atom appears */
00262     for (isimp=0; isimp<thee->nsimp; isimp++) {
00263         for (iatom=0; iatom<thee->nqsm[isimp]; iatom++) {
00264             jatom = thee->sqm[isimp][iatom];
00265             thee->nqsm[jatom]++;
00266         }
00267     }
00268     /* Do a TIME-CONSUMING SANITY CHECK to make sure that each atom was
00269      * placed in at simplex */
00270     for (iatom=0; iatom<thee->natom; iatom++) {
00271         if (thee->nqsm[iatom] == 0) {
00272             Vnm_print(2, "Vcsm_init: Atom %d not placed in simplex!\n", iatom);
00273             VASSERT(0);
00274         }
00275     }
00276     /* Allocate the appropriate amount of space for each entry in the
00277      * charge-simplex map and clear the counter for re-use in assignment */
00278     for (iatom=0; iatom<thee->natom; iatom++) {
00279         thee->qsm[iatom] = (int*)Vmem_malloc(thee->vmem, (thee->nqsm
00280         )[iatom],
00281             sizeof(int));
00282         VASSERT(thee->qsm[iatom] != VNULL);
00283         thee->nqsm[iatom] = 0;
00284     }
00285     /* Assign the simplices to atoms */
00286     for (isimp=0; isimp<thee->nsimp; isimp++) {
00287         for (iatom=0; iatom<thee->nqsm[isimp]; iatom++) {
00288             jatom = thee->sqm[isimp][iatom];
00289             thee->qsm[jatom][thee->nqsm[jatom]] = isimp;
00290             thee->nqsm[jatom]++;
00291         }
00292     }
00293     thee->initFlag = 1;
00294 }
00295
00296 VPUBLIC void Vcsm_dtor(Vcsm **thee) {
00297     if ((*thee) != VNULL) {
00298         Vcsm_dtor2(*thee);
00299         Vmem_free(VNULL, 1, sizeof(Vcsm), (void **)thee);
00300         (*thee) = VNULL;
00301     }
00302 }
00303
00304 VPUBLIC void Vcsm_dtor2(Vcsm *thee) {
00305     int i;
00306
00307     if ((thee != VNULL) && thee->initFlag) {
00308         for (i=0; i<thee->msimp; i++) {
00309             if (thee->nqsm[i] > 0) Vmem_free(thee->vmem, thee->nqsm
00310             [i],
00311                 sizeof(int), (void **)&(thee->sqm[i]));
00312             }
00313             for (i=0; i<thee->natom; i++) {
00314                 if (thee->nqsm[i] > 0) Vmem_free(thee->vmem, thee->nqsm
00315                 [i],
00316                     sizeof(int), (void **)&(thee->qsm[i]));
00317                 Vmem_free(thee->vmem, thee->msimp, sizeof(int *),
00318                     (void **)&(thee->sqm));
00319                 Vmem_free(thee->vmem, thee->msimp, sizeof(int),
00320                     (void **)&(thee->nqsm));
00321                 Vmem_free(thee->vmem, thee->natom, sizeof(int *),
00322                     (void **)&(thee->qsm));
00323                 Vmem_free(thee->vmem, thee->natom, sizeof(int),
00324                     (void **)&(thee->nqsm));
00325             }
00326         }
00327         Vmem_dtor(&(thee->vmem));
00328     }
00329
00330 VPUBLIC int Vcsm_update(Vcsm *thee, SS **simpes, int num) {
00331
00332     /* Counters */
00333     int isimp, jsimp, iatom, jatom, atomID, simpID;
00334     int nsimps, gotMem;
00335     /* Object info */
00336     Vatom *atom;
00337     SS *simplex;

```

```

00338     double *position;
00339     /* Lists */
00340     int *qParent, nqParent;
00341     int **sqmNew, *nsqmNew;
00342     int *affAtoms, nAffAtoms;
00343     int *dnqsm, *nqsmNew, **qsmNew;
00344
00345     VASSERT(thee != VNULL);
00346     VASSERT(thee->initFlag);
00347
00348     /* If we don't have enough memory to accommodate the new entries,
00349      * add more by doubling the existing amount */
00350     isimp = thee->nsimp + num - 1;
00351     gotMem = 0;
00352     while (!gotMem) {
00353         if (isimp > thee->msimp) {
00354             isimp = 2 * isimp;
00355             thee->nsqm = (int*)Vmem_realloc(thee->vmem, thee->msimp
00356 , sizeof(int),
00357             (void **)&(thee->nsqm), isimp);
00358             VASSERT(thee->nsqm != VNULL);
00359             thee->sqm = (int**)Vmem_realloc(thee->vmem, thee->msimp
00360 , sizeof(int *),
00361             (void **)&(thee->sqm), isimp);
00362             VASSERT(thee->sqm != VNULL);
00363             thee->msimp = isimp;
00364         } else gotMem = 1;
00365     }
00366     /* Initialize the nsqm entires we just allocated */
00367     for (isimp = thee->nsimp; isimp<thee->nsimp+num-1 ; isimp++) {
00368         thee->nsqm[isimp] = 0;
00369     }
00370     thee->nsimp = thee->nsimp + num - 1;
00371
00372     /* There's a simple case to deal with: if simps[0] didn't have a
00373      * charge in the first place */
00374     isimp = SS_id(simps[0]);
00375     if (thee->nsqm[isimp] == 0) {
00376         for (isimp=1; isimp<num; isimp++) {
00377             thee->nsqm[SS_id(simps[isimp])] = 0;
00378         }
00379         return 1;
00380     }
00381
00382     /* The more complicated case has occured; the parent simplex had one or
00383      * more charges. First, generate the list of affected charges. */
00384     isimp = SS_id(simps[0]);
00385     nqParent = thee->nsqm[isimp];
00386     qParent = thee->sqm[isimp];
00387
00388     sqmNew = (int**)Vmem_malloc(thee->vmem, num, sizeof(int *));
00389     VASSERT(sqmNew != VNULL);
00390     nsqmNew = (int*)Vmem_malloc(thee->vmem, num, sizeof(int));
00391     VASSERT(nsqmNew != VNULL);
00392     for (isimp=0; isimp<num; isimp++) nsqmNew[isimp] = 0;
00393
00394     /* Loop through the affected atoms to determine how many atoms each
00395      * simplex will get. */
00396     for (iatom=0; iatom<nqParent; iatom++) {
00397
00398         atomID = qParent[iatom];
00399         atom = Valist_getAtom(thee->alist, atomID);
00400         position = Vatom_getPosition(atom);
00401         nsimps = 0;
00402
00403         jsimp = 0;
00404
00405         for (isimp=0; isimp<num; isimp++) {
00406             simplex = simps[isimp];
00407             if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00408                 nsqmNew[isimp]++;
00409                 jsimp = 1;
00410             }
00411         }
00412         VASSERT(jsimp != 0);
00413     }
00414
00415     /* Sanity check that we didn't lose any atoms... */
00416     iatom = 0;

```

```

00417     for (isimp=0; isimp<num; isimp++) iatom += nsqmNew[isimp];
00418     if (iatom < nqParent) {
00419         Vnm_print(2, "Vcsm_update: Lost %d (of %d) atoms!\n",
00420                 nqParent - iatom, nqParent);
00421         VASSERT(0);
00422     }
00423
00424     /* Allocate the storage */
00425     for (isimp=0; isimp<num; isimp++) {
00426         if (nsqmNew[isimp] > 0) {
00427             sqmNew[isimp] = (int*)Vmem_malloc(thee->vmem, nsqmNew[isimp],
00428                                             sizeof(int));
00429             VASSERT(sqmNew[isimp] != VNULL);
00430         }
00431     }
00432
00433     /* Assign charges to simplices */
00434     for (isimp=0; isimp<num; isimp++) {
00435
00436         jsimp = 0;
00437         simplex = simps[isimp];
00438
00439         /* Loop over the atoms associated with the parent simplex */
00440         for (iatom=0; iatom<nqParent; iatom++) {
00441
00442             atomID = qParent[iatom];
00443             atom = Valist_getAtom(thee->alist, atomID);
00444             position = Vatom_getPosition(atom);
00445             if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00446                 sqmNew[isimp][jsimp] = atomID;
00447                 jsimp++;
00448             }
00449         }
00450     }
00451
00452     /* Update the QSM map using the old and new SQM lists */
00453     /* The affected atoms are those contained in the parent simplex; i.e.
00454      * thee->sqm[SS_id(simps[0])] */
00455     affAtoms = thee->sqm[SS_id(simps[0])];
00456     nAffAtoms = thee->nsqm[SS_id(simps[0])];
00457
00458     /* Each of these atoms will go somewhere else; i.e., the entries in
00459      * thee->qsm are never destroyed and thee->nqsm never decreases.
00460      * However, it is possible that a subdivision could cause an atom to be
00461      * shared by two child simplices. Here we record the change, if any,
00462      * in the number of simplices associated with each atom. */
00463     dnqsm = (int*)Vmem_malloc(thee->vmem, nAffAtoms, sizeof(int));
00464     VASSERT(dnqsm != VNULL);
00465     nqsmNew = (int*)Vmem_malloc(thee->vmem, nAffAtoms, sizeof(int));
00466     VASSERT(nqsmNew != VNULL);
00467     qsmNew = (int**)Vmem_malloc(thee->vmem, nAffAtoms, sizeof(int*));
00468     VASSERT(qsmNew != VNULL);
00469     for (iatom=0; iatom<nAffAtoms; iatom++) {
00470
00471         dnqsm[iatom] = -1;
00472         atomID = affAtoms[iatom];
00473         for (isimp=0; isimp<num; isimp++) {
00474             for (jatom=0; jatom<nsqmNew[isimp]; jatom++) {
00475                 if (sqmNew[isimp][jatom] == atomID) dnqsm[iatom]++;
00476             }
00477         }
00478         VASSERT(dnqsm[iatom] > -1);
00479     }
00480
00481     /* Setup the new entries in the array */
00482     for (iatom=0; iatom<nAffAtoms; iatom++) {
00483         atomID = affAtoms[iatom];
00484         qsmNew[iatom] = (int*)Vmem_malloc(thee->vmem,
00485                                         (dnqsm[iatom] + thee->nqsm[atomID]),
00486                                         sizeof(int));
00487         nqsmNew[iatom] = 0;
00488         VASSERT(qsmNew[iatom] != VNULL);
00489     }
00490
00491     /* Fill the new entries in the array */
00492     /* First, do the modified entries */
00493     for (isimp=0; isimp<num; isimp++) {
00494         simpID = SS_id(simps[isimp]);
00495         for (iatom=0; iatom<nsqmNew[isimp]; iatom++) {
00496             atomID = sqmNew[isimp][iatom];
00497             for (jatom=0; jatom<nAffAtoms; jatom++) {
00498                 if (atomID == affAtoms[jatom]) break;
00499             }
00500             if (jatom < nAffAtoms) {
00501                 qsmNew[jatom][nqsmNew[jatom]] = simpID;
00502             }
00503         }
00504     }

```

```

00498             nqsmNew[jatom]++;
00499         }
00500     }
00501 }
00502 /* Now do the unmodified entries */
00503 for (iatom=0; iatom<nAffAtoms; iatom++) {
00504     atomID = affAtoms[iatom];
00505     for (isimp=0; isimp<thee->nqsm[atomID]; isimp++) {
00506         for (jsimp=0; jsimp<num; jsimp++) {
00507             simpID = SS_id(simps[jsimp]);
00508             if (thee->qsm[atomID][isimp] == simpID) break;
00509         }
00510         if (jsimp == num) {
00511             qsmNew[iatom][nqsmNew[iatom]] = thee->qsm[atomID][isimp];
00512             nqsmNew[iatom]++;
00513         }
00514     }
00515 }
00516
00517 /* Replace the existing entries in the table. Do the QSM entires
00518 * first, since they require affAtoms = thee->sqm[simps[0]] */
00519 for (iatom=0; iatom<nAffAtoms; iatom++) {
00520     atomID = affAtoms[iatom];
00521     Vmem_free(thee->vmem, thee->nqsm[atomID], sizeof(int),
00522                 (void **)&(thee->qsm[atomID]));
00523     thee->qsm[atomID] = qsmNew[iatom];
00524     thee->nqsm[atomID] = nqsmNew[iatom];
00525 }
00526 for (isimp=0; isimp<num; isimp++) {
00527     simpID = SS_id(simps[isimp]);
00528     if (thee->nsqm[simpID] > 0) Vmem_free(thee->vmem, thee->nsqm
00529 [simpID],
00530             sizeof(int), (void **)&(thee->sqm[simpID]));
00531     thee->sqm[simpID] = sqmNew[isimp];
00532     thee->nsqm[simpID] = nsqmNew[isimp];
00533 }
00534 Vmem_free(thee->vmem, num, sizeof(int *), (void **)&sqmNew);
00535 Vmem_free(thee->vmem, num, sizeof(int), (void **)&nsqmNew);
00536 Vmem_free(thee->vmem, nAffAtoms, sizeof(int *), (void **)&qsmNew);
00537 Vmem_free(thee->vmem, nAffAtoms, sizeof(int), (void **)&nqsmNew);
00538 Vmem_free(thee->vmem, nAffAtoms, sizeof(int), (void **)&dnqsm);
00539
00540
00541     return 1;
00542
00543
00544 }
00545
00546 #endif

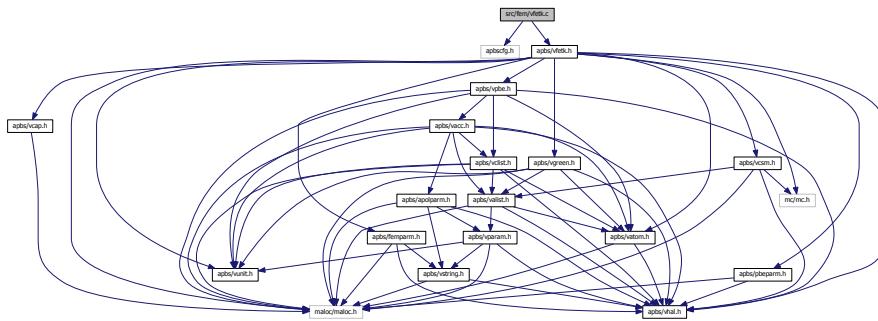
```

9.17 src/fem/vfetk.c File Reference

Class Vfetk methods.

```
#include "apbscfg.h"
#include "apbs/vfetk.h"
```

Include dependency graph for vfetk.c:



Macros

- `#define VRINGMAX 1000`
Maximum number of simplices in a simplex ring.
 - `#define VATOMMAX 1000000`
Maximum number of atoms associated with a vertex.

Functions

- VPRIVATE void **polyEval** (int numP, double p[], double c[][VMAXP], double xv[])
 - VPUBLIC Gem * **Vfetk_getGem** (Vfetk *thee)
 - Get a pointer to the Gem (grid manager) object.*
 - VPUBLIC AM * **Vfetk_getAM** (Vfetk *thee)
 - Get a pointer to the AM (algebra manager) object.*
 - VPUBLIC Vpbe * **Vfetk_getVpbe** (Vfetk *thee)
 - Get a pointer to the Vpbe (PBE manager) object.*
 - VPUBLIC Vcsm * **Vfetk_getVcsm** (Vfetk *thee)
 - Get a pointer to the Vcsm (charge-simplex map) object.*
 - VPUBLIC int **Vfetk_getAtomColor** (Vfetk *thee, int iatom)
 - Get the partition information for a particular atom.*
 - VPUBLIC Vfetk * **Vfetk_ctor** (Vpbe *pbe, Vhal_PBEType type)
 - Constructor for Vfetk object.*
 - VPUBLIC int **Vfetk_ctor2** (Vfetk *thee, Vpbe *pbe, Vhal_PBEType type)
 - FORTRAN stub constructor for Vfetk object.*
 - VPUBLIC void **Vfetk_setParameters** (Vfetk *thee, PBEmprm *pbeparm, FEMprm *feparm)
 - Set the parameter objects.*
 - VPUBLIC void **Vfetk_dtor** (Vfetk **thee)
 - Object destructor.*
 - VPUBLIC void **Vfetk_dtor2** (Vfetk *thee)
 - FORTRAN stub object destructor.*
 - VPUBLIC double * **Vfetk_getSolution** (Vfetk *thee, int *length)
 - Create an array containing the solution (electrostatic potential in units of $k_B T / e$) at the finest mesh level.*
 - VEXTERNC double **Vfetk_energy** (Vfetk *thee, int color, int nonlin)

- VPUBLIC double [Vfetk_qfEnergy](#) (*Vfetk* *thee, int color)

Return the total electrostatic energy.
- VPUBLIC double [Vfetk_dqmEnergy](#) (*Vfetk* *thee, int color)

Get the "fixed charge" contribution 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)

FORTRAN stub: destroys FEtk PDE object.
- VPUBLIC void [Vfetk_PDE_initAssemble](#) (PDE *thee, int ip[], double rp[])

Do once-per-assembly initialization.
- VPUBLIC void [Vfetk_PDE_initElement](#) (PDE *thee, int elementType, int chart, double tvx[][3], void *data)

Do once-per-face initialization.
- 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 tU[][3])

Do once-per-point initialization.
- VPUBLIC void [Vfetk_PDE_Fu](#) (PDE *thee, int key, double F[])

Evaluate strong form of PBE. For interior points, this is:

$$-\nabla \cdot \epsilon \nabla u + b(u) - f$$

where $b(u)$ is the (possibly nonlinear) mobile ion term and f is the source charge distribution term (for PBE) or the induced surface charge distribution (for RPBE). For an interior-boundary (simplex face) point, this is:

$$[\epsilon(x) \nabla u(x) \cdot n(x)]_{x=0^+} - [\epsilon(x) \nabla u(x) \cdot n(x)]_{x=0^-}$$

where $n(x)$ is the normal to the simplex face and the term represents the jump in dielectric displacement across the face. There is no outer-boundary contribution for this problem.

- VPUBLIC double [Vfetk_PDE_Fu_v](#) (PDE *thee, int key, double V[], double dV[] [[VAPBS_DIM](#)])

This is the weak form of the PBE; i.e. the strong form integrated with a test function to give:

$$\int_{\Omega} [\epsilon \nabla u \cdot \nabla v + b(u)v - fv] dx$$

where $b(u)$ denotes the mobile ion term.

- VPUBLIC double **Vfetk_PDE_DFu_wv** (PDE *thee, int key, double W[], double dW[][VAPBS_DIM], double V[], double dV[][3])
- VPUBLIC void **Vfetk_PDE_delta** (PDE *thee, int type, int chart, double txq[], void *user, double F[])

Evaluate a (discretized) delta function source term at the given point.
- VPUBLIC void **Vfetk_PDE_u_D** (PDE *thee, int type, int chart, double txq[], double F[])

Evaluate the Dirichlet boundary condition at the given point.
- VPUBLIC void **Vfetk_PDE_u_T** (PDE *thee, int type, int chart, double txq[], double F[])

Evaluate the "true solution" at the given point for comparison with the numerical solution.
- VPUBLIC void **Vfetk_PDE_bisectEdge** (int dim, int dimll, int edgeType, int chart[], double vx[][3])
- VPUBLIC void **Vfetk_PDE_mapBoundary** (int dim, int dimll, int vertexType, int chart, double vx[3])
- VPUBLIC int **Vfetk_PDE_markSimplex** (int dim, int dimll, int simplexType, int faceType[VAPBS_NVS], int vertexType[VAPBS_NVS], int chart[], double vx[][3], void *simplex)
- VPUBLIC void **Vfetk_PDE_oneChart** (int dim, int dimll, 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 **Igr_3DP1z** [VAPBS_NVS][VMAXP]
- VPRIVATE const int **P_DEG** = 1
- VPRIVATE int **numP**
- VPRIVATE double **c** [VMAXP][VMAXP]
- VPRIVATE double **cx** [VMAXP][VMAXP]
- VPRIVATE double **cy** [VMAXP][VMAXP]
- VPRIVATE double **cz** [VMAXP][VMAXP]

9.17.1 Detailed Description

Class Vfetk methods.

Author

Nathan Baker

Version

Id:

[vfetk.c](#) 1750 2012-07-18 18:34:27Z tuckerbeck

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
```

```

* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*

```

Definition in file [vfetk.c](#).

9.17.2 Variable Documentation

9.17.2.1 VPRIVATE char* diriCubeString

Initial value:

```

"mcsf_begin=1;\n\
\n\
dim=3;\n\
dimii=3;\n\
vertices=8;\n\
simplices=6;\n\
\n\
vert=[\n\
0 0 -0.5 -0.5 -0.5\n\
1 0 0.5 -0.5 -0.5\n\
2 0 -0.5 0.5 -0.5\n\
3 0 0.5 0.5 -0.5\n\
4 0 -0.5 -0.5 0.5\n\
5 0 0.5 -0.5 0.5\n\
6 0 -0.5 0.5 0.5\n\
7 0 0.5 0.5 0.5\n\
];\n\
\n\
simp=[\n\
0 0 0 0 1 0 1 0 5 1 2\n\
1 0 0 0 1 1 0 0 5 2 4\n\
2 0 0 0 1 0 1 1 5 3 2\n\
3 0 0 0 1 0 1 3 5 7 2\n\
4 0 0 1 1 0 0 2 5 7 6\n\
5 0 0 1 1 0 0 2 5 6 4\n\
];\n\
\n\
mcsf_end=1;\n\
"

```

Definition at line 98 of file [vfetk.c](#).

9.17.2.2 VPRIVATE int lgr_2DP1[3][VMAXP]

Initial value:

```

{
{ 2, -2, -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
}
```

Definition at line 391 of file [vfetk.c](#).

9.17.2.3 VPRIVATE int lgr_2DP1x[3][VMAXP]

Initial value:

```
{
{ -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
}
```

Definition at line 400 of file [vfetk.c](#).

9.17.2.4 VPRIVATE int lgr_2DP1y[3][VMAXP]

Initial value:

```
{
{ -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
}
```

Definition at line 407 of file [vfetk.c](#).

9.17.2.5 VPRIVATE int lgr_2DP1z[3][VMAXP]

Initial value:

```
{
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
}
```

Definition at line 414 of file [vfetk.c](#).

9.17.2.6 VPRIVATE int lgr_3DP1[VAPBS_NVS][VMAXP]

Initial value:

```
{
{ 2, -2, -2, -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
}
```

Definition at line 443 of file [vfetk.c](#).

9.17.2.7 VPRIVATE int lgr_3DP1x[VAPBS_NVS][VMAXP]

Initial value:

```
{
{ -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
}
```

Definition at line 451 of file [vfetk.c](#).

9.17.2.8 VPRIVATE int lgr_3DP1y[VAPBS_NVS][VMAXP]

Initial value:

```
{
{ -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
}
```

Definition at line 459 of file [vfetk.c](#).

9.17.2.9 VPRIVATE int lgr_3DP1z[VAPBS_NVS][VMAXP]

Initial value:

```
{
{ -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
{ }
}
```

Definition at line 467 of file [vfetk.c](#).

9.17.2.10 VPRIVATE char* neumCubeString

Initial value:

```
"mcsf_begin=1;\n\
\n\
dim=3;\n\
dimii=3;\n\
vertices=8;\n\
simplices=6;\n\
\n\
vert=[\n\
0 0 -0.5 -0.5 -0.5\n\
1 0 0.5 -0.5 -0.5\n\
2 0 -0.5 0.5 -0.5\n\
3 0 0.5 0.5 -0.5\n\
4 0 -0.5 -0.5 0.5\n\
5 0 0.5 -0.5 0.5\n\
]
```

```

6 0 -0.5 0.5 0.5\n\
7 0 0.5 0.5 0.5\n\
];\n\
\n\
simp=[\n\
0 0 0 0 2 0 2 0 5 1 2\n\
1 0 0 0 2 2 0 0 5 2 4\n\
2 0 0 0 2 0 2 1 5 3 2\n\
3 0 0 0 2 0 2 3 5 7 2\n\
4 0 0 2 2 0 0 2 5 7 6\n\
5 0 0 2 2 0 0 2 5 6 4\n\
1;\n\
\n\
mcsf_end=1;\n\
\n\
"

```

Definition at line 135 of file [vfetk.c](#).

9.18 vfetk.c

```

00001
00058 #include "apbscfg.h"
00059
00060 #ifdef HAVE_MC_H
00061
00062 #include "apbs/vfetk.h"
00063
00064 /* Define the macro DONEUMANN to run with all-Neumann boundary conditions.
00065 * Set this macro at your own risk! */
00066 /* #define DONEUMANN 1 */
00067
00068 /*
00069 * @brief Calculate the contribution to the charge-potential energy from one
00070 * atom
00071 * @ingroup Vfetk
00072 * @author Nathan Baker
00073 * @param thee current Vfetk object
00074 * @param iatom current atom index
00075 * @param color simplex subset (partition) under consideration
00076 * @param sol current solution
00077 * @returns Per-atom energy
00078 */
00079 VPRIvate double Vfetk_qfEnergyAtom(
00080     Vfetk *thee,
00081     int iatom,
00082     int color,
00083     double *sol
00084 );
00085
00086 /*
00087 * @brief Container for local variables
00088 * @ingroup Vfetk
00089 * @bug Not thread-safe
00090 */
00091 VPRIvate Vfetk_LocalVar var;
00092
00093 /*
00094 * @brief MCSF-format cube mesh (all Dirichlet)
00095 * @ingroup Vfetk
00096 * @author Based on mesh by Mike Holst
00097 */
00098 VPRIvate char *diriCubeString =
00099 "mcsf_begin=1;\n\
00100 \n\
00101 dim=3;\n\
00102 dimii=3;\n\
00103 vertices=8;\n\
00104 simplices=6;\n\
00105 \n\
00106 vert=[\n\
00107 0 0 -0.5 -0.5 -0.5\n\
00108 1 0 0.5 -0.5 -0.5\n\
00109 2 0 -0.5 0.5 -0.5\n\
00110 3 0 0.5 0.5 -0.5\n\
00111 4 0 -0.5 -0.5 0.5\n\

```

```

00112 5 0 0.5 -0.5 0.5\n\
00113 6 0 -0.5 0.5 0.5\n\
00114 7 0 0.5 0.5 0.5\n\
00115 ];\n\
00116 \n\
00117 simp=[\n\
00118 0 0 0 0 1 0 1 0 5 1 2\n\
00119 1 0 0 0 1 1 0 0 5 2 4\n\
00120 2 0 0 0 1 0 1 1 5 3 2\n\
00121 3 0 0 0 1 0 1 3 5 7 2\n\
00122 4 0 0 1 1 0 0 2 5 7 6\n\
00123 5 0 0 1 1 0 0 2 5 6 4\n\
00124 ];\n\
00125 \n\
00126 mcsf_end=1;\n\
00127 \n\
00128 ";
00129
00130 /*
00131 * @brief MCSF-format cube mesh (all Neumann)
00132 * @ingroup Vfetk
00133 * @author Based on mesh by Mike Holst
00134 */
00135 VPRIIVATE char *neumCubeString =
00136 "mcsf_begin=1;\n\
00137 \n\
00138 dim=3;\n\
00139 dimi=3;\n\
00140 vertices=8;\n\
00141 simplices=6;\n\
00142 \n\
00143 vert=[\n\
00144 0 0 -0.5 -0.5 -0.5\n\
00145 1 0 0.5 -0.5 -0.5\n\
00146 2 0 -0.5 0.5 -0.5\n\
00147 3 0 0.5 0.5 -0.5\n\
00148 4 0 -0.5 -0.5 0.5\n\
00149 5 0 0.5 -0.5 0.5\n\
00150 6 0 -0.5 0.5 0.5\n\
00151 7 0 0.5 0.5 0.5\n\
00152 ];\n\
00153 \n\
00154 simp=[\n\
00155 0 0 0 0 2 0 2 0 5 1 2\n\
00156 1 0 0 0 2 2 0 0 5 2 4\n\
00157 2 0 0 0 2 0 2 1 5 3 2\n\
00158 3 0 0 0 2 0 2 3 5 7 2\n\
00159 4 0 0 2 2 0 0 2 5 7 6\n\
00160 5 0 0 2 2 0 0 2 5 6 4\n\
00161 ];\n\
00162 \n\
00163 mcsf_end=1;\n\
00164 \n\
00165 ";
00166
00167 /*
00168 * @brief Return the smoothed value of the dielectric coefficient at the
00169 * current point using a fast, chart-based method
00170 * @ingroup Vfetk
00171 * @author Nathan Baker
00172 * @returns Value of dielectric coefficient
00173 * @bug Not thread-safe
00174 */
00175 VPRIIVATE double diel();
00176
00177 /*
00178 * @brief Return the smoothed value of the ion accessibility at the
00179 * current point using a fast, chart-based method
00180 * @ingroup Vfetk
00181 * @author Nathan Baker
00182 * @returns Value of mobile ion coefficient
00183 * @bug Not thread-safe
00184 */
00185 VPRIIVATE double ionacc();
00186
00187 /*
00188 * @brief Smooths a mesh-based coefficient with a simple harmonic function
00189 * @ingroup Vfetk
00190 * @author Nathan Baker
00191 * @param meth Method for smoothing
00192 * \li 0 ==> arithmetic mean (gives bad results)

```

```

00193 * \li 1 ==> geometric mean
00194 * @param nverts Number of vertices
00195 * @param dist distance from point to each vertex
00196 * @param coeff coefficient value at each vertex
00197 * @note Thread-safe
00198 * @return smoothed value of coefficient at point of interest */
00199 VPRIATE double smooth(
00200     int nverts,
00201     double dist[VAPBS_NVS],
00202     double coeff[VAPBS_NVS],
00203     int meth
00204 );
00205
00206
00207 /*
00208 * @brief Return the analytical multi-sphere Debye-Hückel approximation (in
00209 * kT/e) at the specified point
00210 * @ingroup Vfetk
00211 * @author Nathan Baker
00212 * @param pbe Vpbe object
00213 * @param d Dimension of x
00214 * @param x Coordinates of point of interest (in Å)
00215 * @note Thread-safe
00216 * @returns Multi-sphere Debye-Hückel potential in kT/e
00217 */
00218 VPRIATE double debye_U(
00219     Vpbe *pbe,
00220     int d,
00221     double x[]
00222 );
00223
00224 /*
00225 * @brief Return the difference between the analytical multi-sphere
00226 * Debye-Hückel approximation and Coulomb's law (in kT/e) at the specified
00227 * point
00228 * @ingroup Vfetk
00229 * @author Nathan Baker
00230 * @param pbe Vpbe object
00231 * @param d Dimension of x
00232 * @param x Coordinates of point of interest (in Å)
00233 * @note Thread-safe
00234 * @returns Multi-sphere Debye-Hückel potential in kT/e */
00235 VPRIATE double debye_Udiff(
00236     Vpbe *pbe,
00237     int d,
00238     double x[]
00239 );
00240
00241 /*
00242 * @brief Calculate the Coulomb's
00243 * Debye-Hückel approximation and Coulomb's law (in kT/e) at the specified
00244 * point
00245 * @ingroup Vfetk
00246 * @author Nathan Baker
00247 * @param pbe Vpbe object
00248 * @param d Dimension of x
00249 * @param x Coordinates of point of interest (in Å)
00250 * @param eps Dielectric constant
00251 * @param U Set to potential (in kT/e)
00252 * @param dU Set to potential gradient (in kT/e/Å)
00253 * @param d2U Set to Laplacian of potential (in  $\text{kT e}^{-1} \text{AA}^{-2}$ )
00254 * @returns Multi-sphere Debye-Hückel potential in kT/e */
00255 VPRIATE void coulomb(
00256     Vpbe *pbe,
00257     int d,
00258     double x[],
00259     double eps,
00260     double *U,
00261     double dU[],
00262     double *d2U
00263 );
00264
00265 /*
00266 * @brief 2D linear master simplex information generator
00267 * @ingroup Vfetk
00268 * @author Mike Holst
00269 * @param dimIS dunno
00270 * @param ndof dunno
00271 * @param dof dunno
00272 * @param c dunno
00273 * @param cx dunno

```

```

00274 * @note Trust in Mike */
00275 VPRIIVATE void init_2DP1(
00276     int dimIS[],
00277     int *ndof,
00278     int dof[],
00279     double c[][VMAXP],
00280     double cx[][VMAXP],
00281     double cy[][VMAXP],
00282     double cz[][VMAXP]
00283 );
00284
00285 /*
00286 * @brief 3D linear master simplex information generator
00287 * @ingroup Vfetk
00288 * @author Mike Holst
00289 * @param dimIS dunno
00290 * @param ndof dunno
00291 * @param dof dunno
00292 * @param c dunno
00293 * @param cx dunno
00294 * @param cy dunno
00295 * @param cz dunno
00296 * @note Trust in Mike */
00297 VPRIIVATE void init_3DP1(
00298     int dimIS[],
00299     int *ndof,
00300     int dof[],
00301     double c[][VMAXP],
00302     double cx[][VMAXP],
00303     double cy[][VMAXP],
00304     double cz[][VMAXP]
00305 );
00306
00307 /*
00308 * @brief Setup coefficients of polynomials from integer table data
00309 * @ingroup Vfetk
00310 * @author Mike Holst
00311 * @param numP dunno
00312 * @param c dunno
00313 * @param cx dunno
00314 * @param cy dunno
00315 * @param cz dunno
00316 * @param ic dunno
00317 * @param icx dunno
00318 * @param icy dunno
00319 * @param icz dunno
00320 * @note Trust in Mike */
00321 VPRIIVATE void setCoef(
00322     int numP,
00323     double c[][VMAXP],
00324     double cx[][VMAXP],
00325     double cy[][VMAXP],
00326     double cz[][VMAXP],
00327     int ic[][VMAXP],
00328     int icx[][VMAXP],
00329     int icy[][VMAXP],
00330     int icz[][VMAXP]
00331 );
00332
00333 /*
00334 * @brief Evaluate a collection of at most cubic polynomials at a
00335 * specified point in at most R^3.
00336 * @ingroup Vfetk
00337 * @author Mike Holst
00338 * @param numP the number of polynomials to evaluate
00339 * @param p the results of the evaluation
00340 * @param c the coefficients of each polynomial
00341 * @param xv the point (x,y,z) to evaluate the polynomials.
00342 * @note Mike says:
00343 * <pre>
00344 * Note that "VMAXP" must be >= 19 for cubic polynomials.
00345 * The polynomials are build from the coefficients c[][] as
00346 * follows. To build polynomial "k", fix k and set:
00347 *
00348 * c0=c[k][0], c1=c[k][1], ..., cp=c[k][p]
00349 *
00350 * Then evaluate as:
00351 *
00352 * p3(x,y,z) = c0 + c1*x + c2*y + c3*z
00353 *             + c4*x*x + c5*x*y + c6*z*z + c7*x*y + c8*x*z + c9*y*z
00354 *             + c10*x*x*x + c11*y*y*y + c12*z*z*z

```

```

00355 *           + c13*x*x*y + c14*x*x*z + c15*x*y*y
00356 *           + c16*y*y*z + c17*x*z*z + c18*y*z*z
00357 * </pre>
00358 */
00359 VPRIPRIVATE void polyEval(
00360     int numP,
00361     double p[],
00362     double c[][VMAXP],
00363     double xv[]
00364 );
00365
00366 /*
00367 * @brief I have no clue what this variable does, but we need it to initialize
00368 * the simplices
00369 * @ingroup Vfetk
00370 * @author Mike Holst */
00371 VPRIPRIVATE int dim_2DP1 = 3;
00372
00373 /*
00374 * @brief I have no clue what these variable do, but we need it to initialize
00375 * the simplices
00376 * @ingroup Vfetk
00377 * @author Mike Holst
00378 * @note Mike says:
00379 * <pre>
00380 * 2D-P1 Basis:
00381 *
00382 * p1(x,y) = c0 + c1*x + c2*y
00383 *
00384 * Lagrange Point      Lagrange Basis Function Definition
00385 * -----
00386 * (0, 0)          p[0](x,y) = 1 - x - y
00387 * (1, 0)          p[1](x,y) = x
00388 * (0, 1)          p[2](x,y) = y
00389 * </pre>
00390 */
00391 VPRIPRIVATE int lgr_2DP1[3][VMAXP] = {
00392 /*c0 c1 c2 c3
00393 * -----
00394 /* 1   x   y   z
00395 * -----
00396 { 2, -2, -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00397 { 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00398 { 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
00399 };
00400 VPRIPRIVATE int lgr_2DP1x[3][VMAXP] = {
00401 /*c0 -----
00402 /* 1 -----
00403 { -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00404 { 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00405 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
00406 };
00407 VPRIPRIVATE int lgr_2DP1y[3][VMAXP] = {
00408 /*c0 -----
00409 /* 1 -----
00410 { -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00411 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00412 { 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
00413 };
00414 VPRIPRIVATE int lgr_2DP1z[3][VMAXP] = {
00415 /*c0 -----
00416 /* 1 -----
00417 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00418 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00419 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
00420 };
00421
00422
00423 /*
00424 * @brief I have no clue what these variable do, but we need it to initialize
00425 * the simplices
00426 * @ingroup Vfetk
00427 * @author Mike Holst
00428 * @note Mike says:
00429 * <pre>
00430 * 3D-P1 Basis:
00431 *
00432 * p1(x,y,z) = c0 + c1*x + c2*y + c3*z
00433 *
00434 * Lagrange Point      Lagrange Basis Function Definition
00435 * -----

```

```

00436 * (0, 0, 0)          p[0](x,y,z) = 1 - x - y - z
00437 * (1, 0, 0)          p[1](x,y,z) = x
00438 * (0, 1, 0)          p[2](x,y,z) = y
00439 * (0, 0, 1)          p[3](x,y,z) = z
00440 * </pre>
00441 */
00442 VPRIVATE int dim_3DP1 = VAPBS_NVS;
00443 VPRIVATE int lgr_3DP1[VAPBS_NVS][VMAXP] = {
00444 /*c0 c1 c2 c3 */ */
00445 /* 1 x y z */ */
00446 { 2, -2, -2, -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00447 { 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00448 { 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00449 { 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00450 };
00451 VPRIVATE int lgr_3DP1x[VAPBS_NVS][VMAXP] = {
00452 /*c0 */ */
00453 /* 1 */ */
00454 { -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00455 { 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00456 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00457 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00458 };
00459 VPRIVATE int lgr_3DP1y[VAPBS_NVS][VMAXP] = {
00460 /*c0 */ */
00461 /* 1 */ */
00462 { -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00463 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00464 { 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00465 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00466 };
00467 VPRIVATE int lgr_3DP1z[VAPBS_NVS][VMAXP] = {
00468 /*c0 */ */
00469 /* 1 */ */
00470 { -2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00471 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00472 { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00473 { 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
00474 };
00475 /*
00476 * @brief Another Holst variable
00477 * @ingroup Vfetk
00478 * @author Mike Holst
00479 * @note Mike says: 1 = linear, 2 = quadratic */
00480 VPRIVATE const int P_DEG=1;
00481
00482 /*
00483 * @brief Another Holst variable
00484 * @ingroup Vfetk
00485 * @author Mike Holst */
00486 VPRIVATE int numP;
00487 VPRIVATE double c[VMAXP][VMAXP];
00488 VPRIVATE double cx[VMAXP][VMAXP];
00489 VPRIVATE double cy[VMAXP][VMAXP];
00490 VPRIVATE double cz[VMAXP][VMAXP];
00491
00492 #if !defined(VINLINE_VFETK)
00493
00494 VPUBLIC Gem* Vfetk_getGem(Vfetk *thee) {
00495
00496     VASSERT(thee != VNULL);
00497     return thee->gm;
00498 }
00499
00500 }
00501
00502 VPUBLIC AM* Vfetk_getAM(Vfetk *thee) {
00503
00504     VASSERT(thee != VNULL);
00505     return thee->am;
00506 }
00507
00508 VPUBLIC Vpbe* Vfetk_getVpbe(Vfetk *thee) {
00509
00510     VASSERT(thee != VNULL);
00511     return thee->pbe;
00512 }
00513
00514
00515 VPUBLIC Vcsm* Vfetk_getVcsm(Vfetk *thee) {
00516

```

```

00517     VASSERT(thee != VNULL);
00518     return thee->csm;
00519
00520 }
00521
00522 VPUBLIC int Vfetk_getAtomColor(Vfetk *thee,
00523                                     int iatom
00524                                     ) {
00525
00526     int natoms;
00527
00528     VASSERT(thee != VNULL);
00529
00530     natoms = Valist_getNumberAtoms(Vpbe_getValist
00531     (thee->pbe));
00532     VASSERT(iatom < natoms);
00533
00534     return Vatom_getPartID(Valist_getAtom(
00535     Vpbe_getValist(thee->pbe), iatom));
00536
00537 #endif /* if !defined(VINLINE_VFETK) */
00538
00539 VPUBLIC Vfetk* Vfetk_ctor(Vpbe *pbe,
00540                               Vhal_PBEType type
00541                               ) {
00542
00543     /* Set up the structure */
00544     Vfetk *thee = VNULL;
00545     thee = (Vfetk*)Vmem_malloc(VNULL, 1, sizeof(Vfetk) );
00546     VASSERT(thee != VNULL);
00547     VASSERT(Vfetk_ctor2(thee, pbe, type));
00548
00549
00550 VPUBLIC int Vfetk_ctor2(Vfetk *thee,
00551                           Vpbe *pbe,
00552                           Vhal_PBEType type
00553                           ) {
00554
00555     int i;
00556     double center[VAPBS_DIM];
00557
00558     /* Make sure things have been properly initialized & store them */
00559     VASSERT(pbe != VNULL);
00560     thee->pbe = pbe;
00561     VASSERT(pbe->alist != VNULL);
00562     VASSERT(pbe->acc != VNULL);
00563
00564     /* Store PBE type */
00565     thee->type = type;
00566
00567     /* Set up memory management object */
00568     thee->vmem = Vmem_ctor("APBS::VFETK");
00569
00570     /* Set up FEtk objects */
00571     Vnm_print(0, "Vfetk_ctor2: Constructing PDE...\n");
00572     thee->pde = Vfetk_PDE_ctor(thee);
00573     Vnm_print(0, "Vfetk_ctor2: Constructing Gem...\n");
00574     thee->gm = Gem_ctor(thee->vmem, thee->pde);
00575     Vnm_print(0, "Vfetk_ctor2: Constructing Aprx...\n");
00576     thee->aprx = Aprx_ctor(thee->vmem, thee->gm, thee->pde);
00577     Vnm_print(0, "Vfetk_ctor2: Constructing Aprx...\n");
00578     thee->am = AM_ctor(thee->vmem, thee->aprx);
00579
00580     /* Reset refinement level */
00581     thee->level = 0;
00582
00583     /* Set default solver variables */
00584     thee->lkey = VLT_MG;
00585     thee->lmax = 1000000;
00586     thee->ltol = 1e-5;
00587     thee->lprec = VPT_MG;
00588     thee->nkey = VNT_NEW;
00589     thee->nmax = 1000000;
00590     thee->ntol = 1e-5;
00591     thee->gues = VGT_ZERO;
00592     thee->pjac = -1;
00593
00594     /* Store local copy of myself */
00595     var.fetk = thee;

```

```

00596     var.initGreen = 0;
00597
00598 /* Set up the external Gem subdivision hook */
00599     Gem_SetExternalUpdateFunction(thee->gm,
00600         Vfetk_externalUpdateFunction);
00601
00602     /* Set up ion-related variables */
00603     var.zkappa2 = Vpbe_getZkappa2(var.fetk->pbe);
00604     var.ionstr = Vpbe_getBulkIonicStrength(var.
00605         fetk->pbe);
00606     if (var.ionstr > 0.0) var.zks2 = 0.5*var.zkappa2/var.
00607         ionstr;
00608     else var.zks2 = 0.0;
00609     Vpbe_getIons(var.fetk->pbe, &(var.nion), var.ionConc
00610     , var.ionRadii,
00611         var.ionQ);
00612     for (i=0; i<var.nion; i++) {
00613         var.ionConc[i] = var.zks2 * var.ionConc[i] * var.ionQ
00614         [i];
00615     }
00616
00617     /* Set uninitialized objects to NULL */
00618     thee->pbeparm = VNULL;
00619     thee->feparm = VNULL;
00620     thee->csm = VNULL;
00621
00622     return 1;
00623 }
00624
00625 VPUBLIC void Vfetk_setParameters(Vfetk *thee,
00626                                     PBEparm *pbeparm,
00627                                     FEMparm *feparm
00628                                     ) {
00629
00630     VASSERT(thee != VNULL);
00631     thee->feparm = feparm;
00632     thee->pbeparm = pbeparm;
00633 }
00634
00635 VPUBLIC void Vfetk_dtor(Vfetk **thee) {
00636     if ((*thee) != VNULL) {
00637         Vfetk_dtor2(*thee);
00638         //Vmem_free(VNULL, 1, sizeof(Vfetk), (void **)thee);
00639         (*thee) = VNULL;
00640     }
00641 }
00642
00643 VPUBLIC void Vfetk_dtor2(Vfetk *thee) {
00644     Vcsm_dtor(&(thee->csm));
00645     AM_dtor(&(thee->am));
00646     Aprx_dtor(&(thee->aprx));
00647     Vfetk_PDE_dtor(&(thee->pde));
00648     Vmem_dtor(&(thee->vmem));
00649 }
00650
00651 VPUBLIC double* Vfetk_getSolution(Vfetk *thee,
00652                                         int *length
00653                                         ) {
00654
00655     int i;
00656     double *solution,
00657         *theAnswer;
00658     AM *am;
00659
00660     VASSERT(thee != VNULL);
00661
00662     /* Get the AM object */
00663     am = thee->am;
00664     /* Copy the solution into the w0 vector */
00665     Bvec_copy(am->w0, am->u);
00666     /* Add the Dirichlet conditions */
00667     Bvec_axpy(am->w0, am->ud, 1.);
00668     /* Get the data from the Bvec */
00669     solution = Bvec_addr(am->w0);
00670     /* Get the length of the data from the Bvec */
00671     *length = Bvec_numRT(am->w0);
00672     /* Make sure that we got scalar data (only one block) for the solution
00673      * to the FETK */
00674     VASSERT(1 == Bvec_numB(am->w0));
00675     /* Allocate space for the returned vector and copy the solution into it */
00676     theAnswer = VNULL;
00677

```

```

00672     theAnswer = (double*)Vmem_malloc(VNULL, *length, sizeof(double));
00673     VASSERT(theAnswer != VNULL);
00674     for (i=0; i<(*length); i++) theAnswer[i] = solution[i];
00675
00676     return theAnswer;
00677 }
00678
00679
00680 VPUBLIC double Vfetk_energy(Vfetk *thee,
00681                             int color,
00682                             int nonlin
00683                             ) {
00684
00685     double totEnergy = 0.0,
00686             qfEnergy = 0.0,
00687             dqmEnergy = 0.0;
00688     VASSERT(thee != VNULL);
00689
00690     if (nonlin && (Vpbe_getBulkIonicStrength(thee->pbe
00691 ) > 0.)) {
00692         Vnm_print(0, "Vfetk_energy: calculating full PBE energy\n");
00693         Vnm_print(0, "Vfetk_energy: bulk ionic strength = %g M\n",
00694                 Vpbe_getBulkIonicStrength(thee->pbe));
00695         dqmEnergy = Vfetk_dqmEnergy(thee, color);
00696         Vnm_print(0, "Vfetk_energy: dqmEnergy = %g kT\n", dqmEnergy);
00697         qfEnergy = Vfetk_qfEnergy(thee, color);
00698         Vnm_print(0, "Vfetk_energy: qfEnergy = %g kT\n", qfEnergy);
00699
00700         totEnergy = qfEnergy - dqmEnergy;
00701     } else {
00702         Vnm_print(0, "Vfetk_energy: calculating only q-phi energy\n");
00703         dqmEnergy = Vfetk_dqmEnergy(thee, color);
00704         Vnm_print(0, "Vfetk_energy: dqmEnergy = %g kT (NOT USED)\n", dqmEnergy
00705 );
00706         qfEnergy = Vfetk_qfEnergy(thee, color);
00707         Vnm_print(0, "Vfetk_energy: qfEnergy = %g kT\n", qfEnergy);
00708         totEnergy = 0.5*qfEnergy;
00709     }
00710
00711     return totEnergy;
00712 }
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737 VPUBLIC double Vfetk_qfEnergy(Vfetk *thee,
00738                               int color
00739                               ) {
00740
00741     double *sol,
00742             energy = 0.0;
00743     int nsol,
00744             iatom,
00745             natoms;
00746     AM *am;
00747
00748     VASSERT(thee != VNULL);
00749     am = thee->am;
00750
00751     /* Get the finest level solution */
00752     sol= VNULL;
00753     sol = Vfetk_getSolution(thee, &nsol);
00754     VASSERT(sol != VNULL);
00755
00756     /* Make sure the number of entries in the solution array matches the
00757      * number of vertices currently in the mesh */
00758     if (nsol != Gem_numVV(thee->gm)) {
00759         Vnm_print(2, "Vfetk_qfEnergy: Number of unknowns in solution does not
00760 match\n");
00761         Vnm_print(2, "Vfetk_qfEnergy: number of vertices in mesh!!! Bailing
00762 out!\n");
00763         VASSERT(0);
00764     }
00765
00766     /* Now we do the sum over atoms... */
00767     natoms = Valist_getNumberAtoms(thee->pbe->alist
00768 );
00769     for (iatom=0; iatom<natoms; iatom++) {
00770         energy = energy + Vfetk_qfEnergyAtom(thee, iatom, color, sol);
00771     } /* end for iatom */

```

```

00771
00772     /* Destroy the finest level solution */
00773     Vmem_free(VNULL, nsol, sizeof(double), (void **) &sol);
00774
00775     /* Return the energy */
00776     return energy;
00777 }
00778
00779 VPUBLIC double Vfetk_qfEnergyAtom(
00780     Vfetk *thee,
00781     int iatom,
00782     int color,
00783     double *sol) {
00784
00785     Vatom *atom;
00786     double charge,
00787         phi[VAPBS_NVS],
00788         phix[VAPBS_NVS][3],
00789         *position,
00790         uval,
00791         energy = 0.0;
00792     int isimp,
00793     nsimps,
00794     icolor,
00795     invert,
00796     usingColor;
00797     SS *simp;
00798
00799
00800     /* Get atom information */
00801     atom = Valist_getAtom(thee->pbe->alist, iatom);
00802     icolor = Vfetk_getAtomColor(thee, iatom);
00803     charge = Vatom_getCharge(atom);
00804     position = VatomGetPosition(atom);
00805
00806     /* Find out if we're using colors */
00807     usingColor = (color >= 0);
00808
00809     if (usingColor && (icolor<0)) {
00810         Vnm_print(2, "Vfetk_qfEnergy: Atom colors not set!\n");
00811         VASSERT(0);
00812     }
00813
00814     /* Check if this atom belongs to the specified partition */
00815     if ((icolor==color) || (!usingColor)) {
00816         /* Loop over the simplices associated with this atom */
00817         nsimps = Vcsm_getNumberSimplices(thee->csm,
00818                                         iatom);
00819
00820         /* Get the first simplex of the correct color; we can use just one
00821          * simplex for energy evaluations, but not for force
00822          * evaluations */
00823         for (isimp=0; isimp<nsimps; isimp++) {
00824             simp = Vcsm_getSimplex(thee->csm, isimp, iatom);
00825
00826             /* If we've asked for a particular partition AND if the atom
00827              * is our partition, then compute the energy */
00828             if ((SS_chart(simp)==color)|| (color<0)) {
00829                 /* Get the value of each basis function evaluated at this
00830                  * point */
00831                 Gem_pointInSimplexVal(thee->gm, simp, position, phi, phix);
00832                 for (invert=0; invert<SS_dimVV(simp); invert++) {
00833                     uval = sol[VV_id(SS_vertex(simp, invert))];
00834                     energy += (charge*phi[invert]*uval);
00835                 } /* end for invert */
00836                 /* We only use one simplex of the appropriate color for
00837                  * energy calculations, so break here */
00838                 break;
00839             } /* endif (color) */
00840         } /* end for isimp */
00841     }
00842
00843     return energy;
00844 }
00845
00846
00847 VPUBLIC double Vfetk_dqmEnergy(Vfetk *thee,
00848                                     int color) {
00849
00850     return AM_evalJ(thee->am);

```

```

00851
00852 }
00853
00854 VPUBLIC void Vfetk_setAtomColors(Vfetk *thee) {
00855
00856     SS *simp;
00857     Vatom *atom;
00858     int i,
00859         natoms;
00860
00861     VASSERT(thee != VNULL);
00862
00863     natoms = Valist_getNumberAtoms(thee->pbe->alist
00864 );
00865     for (i=0; i<natoms; i++) {
00866         atom = Valist_getAtom(thee->pbe->alist, i);
00867         simp = Vcsm_getSimplex(thee->csm, 0, i);
00868         Vatom_setPartID(atom, SS_chart(simp));
00869     }
00870 }
00871
00872 VPUBLIC unsigned long int Vfetk_memChk(Vfetk *thee) {
00873
00874     int memUse = 0;
00875
00876     if (thee == VNULL) return 0;
00877
00878     memUse = memUse + sizeof(Vfetk);
00879     memUse = memUse + Vcsm_memChk(thee->csm);
00880
00881     return memUse;
00882 }
00883
00890 VPUBLIC Vrc_Codes Vfetk_genCube(Vfetk *thee,
00891                                     double center[3],
00892                                     double length[3],
00893                                     Vfetk_MeshLoad meshType
00894 ) {
00895
00896     VASSERT(thee != VNULL);
00897
00898     AM *am = VNULL; /* @todo - no idea what this is */
00899     Gem *gm = VNULL; /* Geometry manager */
00900
00901     int skey = 0, /* Simplex format */
00902         bufsize = 0, /* Buffer size */
00903         i, /* Loop counter */
00904         j; /* Loop counter */
00905     char *key = "r", /* Read */
00906     *iodev = "BUFF", /* Buffer */
00907     *iofmt = "ASC", /* ASCII */
00908     *iohost = "localhost", /* localhost (dummy) */
00909     *iofile = "0", /*< socket 0 (dummy) */
00910     buf[VMAX_BUFSIZE]; /* Socket buffer */
00911     Vio *sock = VNULL; /* Socket object */
00912     VV *vx = VNULL; /* @todo - no idea what this is */
00913     double x;
00914
00915     am = thee->am;
00916     VASSERT(am != VNULL);
00917     gm = thee->gm;
00918     VASSERT(gm != VNULL);
00919
00920     /* @note This code is based on Gem_makeCube by Mike Holst */
00921     /* Write mesh string to buffer and read back */
00922     switch (meshType) {
00923     case VML_DIRICUBE:
00924         /* Create a new copy of the DIRICUBE mesh (see globals higher in this
00925         file) */
00926         bufsize = strlen(diriCubeString);
00927         VASSERT( bufsize <= VMAX_BUFSIZE );
00928         strncpy(buf, diriCubeString, VMAX_BUFSIZE);
00929         break;
00929     case VML_NEUMCUBE:
00930         /* Create a new copy of the NEUMCUBE mesh (see globals higher in this
00931         file) */
00932         bufsize = strlen(neumCubeString);
00933         Vnm_print(2, "Vfetk_genCube: WARNING! USING EXPERIMENTAL NEUMANN BOUNDARY
CONDITIONS!\n");
00933         VASSERT( bufsize <= VMAX_BUFSIZE );

```

```

00934     strncpy(buf, neumCubeString, VMAX_BUFSIZE);
00935     break;
00936 case VML_EXTERNAL:
00937     Vnm_print(2, "Vfetk_genCube: Got request for external mesh!\n");
00938     Vnm_print(2, "Vfetk_genCube: How did we get here?\n");
00939     return VRC_FAILURE;
00940 default:
00941     Vnm_print(2, "Vfetk_genCube: Unknown mesh type (%d)\n", meshType);
00942     return VRC_FAILURE;
00943 }
00944
00945     VASSERT( VNULL != (sock=Vio_socketOpen(key, iodev, iofmt, iohost, iofile)) );
00946 /* Open socket */
00947     Vio_bufTake(sock, buf, bufsize); /* Initialize internal buffer for socket
00948 */
00949     AM_read(am, skey, sock); /* Take the initial mesh from the socket and load
00950         into internal AM data structure with simplex
00951         format */
00950 Vio_connectFree(sock); /* Purge output buffers */
00951     Vio_bufGive(sock); /* Get pointer to output buffer? No assignment of
00952         return value... */
00952     Vio_dtor(&sock); /* Destroy output buffer */
00953
00954 /* @todo - could the following be done in a single pass? - PCE */
00955 /* Scale (unit) cube - for each vertex, set the new coordinates of that
00956     vertex based on the vertex length */
00957 for (i=0; i<Gem_numVV(gm); i++) {
00958     vx = Gem_VV(gm, i);
00959     for (j=0; j<3; j++) {
00960         x = VV_coord(vx, j);
00961         x *= length[j];
00962         VV_setCoord(vx, j, x);
00963     }
00964 }
00965
00966 /* Add new center - for each vertex, set a new center for the vertex */
00967 for (i=0; i<Gem_numVV(gm); i++) {
00968     vx = Gem_VV(gm, i);
00969     for (j=0; j<3; j++) {
00970         x = VV_coord(vx, j);
00971         x += center[j];
00972         VV_setCoord(vx, j, x);
00973     }
00974 }
00975
00976     return VRC_SUCCESS;
00977 }
00978
00979 VPUBLIC Vrc_Codes Vfetk_loadMesh(Vfetk *thee, /* Vfetk
00980     object to load into */
00981     constructed) /* Center for mesh (if
00982     constructed) */
00983     length[3], /* Mesh lengths (if
00984     constructed) */
00985     Vfetk_MeshLoad meshType, /* Type
00986     of mesh to load */
00987     Vio *sock /* Socket for external mesh data
00988     (NULL otherwise) */
00989 )
00990 {
00991
00992 Vrc_Codes vrc; /* Function return codes - see vhal.h for enum */
00993 int skey = 0; /* Simplex format */
00994
00995 /* Load mesh from socket if external mesh, otherwise generate mesh */
00996 switch (meshType) {
00997 case VML_EXTERNAL:
00998     if (sock == VNULL) {
00999         Vnm_print(2, "Vfetk_loadMesh: Got NULL socket!\n");
01000         return VRC_FAILURE;
01001     }
01002     AM_read(thee->am, skey, sock);
01003     Vio_connectFree(sock);
01004     Vio_bufGive(sock);
01005     Vio_dtor(&sock);
01006     break;
01007 case VML_DIRICUBE:
01008 case VML_NEUMCUBE:
01009     /* Create new mesh and store in thee */
01010     vrc = Vfetk_genCube(thee, center, length, meshType);
01011     if (vrc == VRC_FAILURE) return VRC_FAILURE;
01012     break;

```

```

01013     default:
01014     Vnm_print(2, "Vfetk_loadMesh: unrecognized mesh type (%d)!\n",
01015             meshType);
01016     return VRC_FAILURE;
01017 };
01018
01019 /* Setup charge-simplex map */
01020 Vnm_print(0, "Vfetk_ctor2: Constructing Vcsm...\n");
01021 thee->csm = VNULL;
01022 /* Construct a new Vcsm with the atom list and gem data */
01023 thee->csm = Vcsm_ctor(Vpbe_getValist(thee->pbe
01024 ), thee->gm);
01025 VASSERT(thee->csm != VNULL);
01026 Vcsm_init(thee->csm);
01027 return VRC_SUCCESS;
01028 }
01029
01030
01031 VPUBLIC void Bmat_printHB(Bmat *thee,
01032                               char *fname
01033                               ) {
01034
01035     Mat *Ablock;
01036     MATsym pqsym;
01037     int i, j, jj;
01038     int *IA, *JA;
01039     double *D, *L, *U;
01040     FILE *fp;
01041
01042     char mmtitle[72];
01043     char mmkey[] = {"8charkey"};
01044     int totc = 0, ptrc = 0, indc = 0, valc = 0;
01045     char mxtyp[] = {"RUA"}; /* Real Unsymmetric Assembled */
01046     int nrow = 0, ncol = 0, numZ = 0;
01047     int numZdigits = 0, nrowdigits = 0;
01048     int nptrline = 8, nindline = 8, nvalline = 5;
01049     char ptrfmt[] = {"(8I10)           ", ptrfmtstr[] = {"%10d"};
01050     char indfmt[] = {"(8I10)           ", indfmtstr[] = {"%10d"};
01051     char valfmt[] = {"(5E16.8)        ", valfmtstr[] = {"%16.8E"};
01052
01053     VASSERT( thee->numB == 1 );          /* HARDWIRE FOR NOW */
01054     Ablock = thee->AD[0][0];
01055
01056     VASSERT( Mat_format( Ablock ) == DRC_FORMAT ); /* HARDWIRE FOR NOW */
01057
01058     pqsym = Mat_sym( Ablock );
01059
01060     if ( pqsym == IS_SYM ) {
01061         mxtyp[1] = 'S';
01062     } else if ( pqsym == ISNOT_SYM ) {
01063         mxtyp[1] = 'U';
01064     } else {
01065         VASSERT( 0 ); /* NOT VALID */
01066     }
01067
01068     nrow = Bmat_numRT( thee ); /* Number of rows */
01069     ncol = Bmat_numCT( thee ); /* Number of cols */
01070     numZ = Bmat_numZT( thee ); /* Number of entries */
01071
01072     nrowdigits = (int) (log( nrow )/log( 10 )) + 1;
01073     numZdigits = (int) (log( numZ )/log( 10 )) + 1;
01074
01075     nptrline = (int) ( 80 / (numZdigits + 1) );
01076     nindline = (int) ( 80 / (nrowdigits + 1) );
01077
01078     sprintf(ptrfmt, "(%dI%d)", nptrline,numZdigits+1);
01079     sprintf(ptrfmtstr,"%%%dd",numZdigits+1);
01080     sprintf(indfmt, "(%dI%d)",nindline,nrowdigits+1);
01081     sprintf(indfmtstr,"%%%dd",nrowdigits+1);
01082
01083     ptrc = (int) ( ( (ncol + 1) - 1 ) / nptrline ) + 1;
01084     indc = (int) ( ( numZ - 1 ) / nindline ) + 1;
01085     valc = (int) ( ( numZ - 1 ) / nvalline ) + 1;
01086
01087     totc = ptrc + indc + valc;
01088
01089     sprintf( mmtitle, "Sparse '%s' Matrix - Harwell-Boeing Format - '%s'",
01090             thee->name, fname );
01091
01092     /* Step 0: Open the file for writing */

```

```

01093
01094     fp = fopen( fname, "w" );
01095     if (fp == VNULL) {
01096         Vnm_Print(2,"Bmat_PrintHB: Ouch couldn't open file <%s>\n",fname);
01097         return;
01098     }
01099
01100    /* Step 1: Print the header information */
01101
01102    fprintf( fp, "%-72s%-8s\n", mmtitle, mmkey );
01103    fprintf( fp, "%14d%14d%14d%14d\n", totc, ptrc, indc, valc, 0 );
01104    fprintf( fp, "%3s%11s%14d%14d\n", mxtyp, " ", nrow, ncol, numZ );
01105    fprintf( fp, "%-16s%-16s%-20s%-20s\n", ptrfmt, indfmt, valfmt, "6E13.5" );
01106
01107    IA = Ablock->IA;
01108    JA = Ablock->JA;
01109    D = Ablock->diag;
01110    L = Ablock->offL;
01111    U = Ablock->offU;
01112
01113    if ( pqsym == IS_SYM ) {
01114
01115        /* Step 2: Print the pointer information */
01116
01117        for (i=0; i<(ncol+1); i++) {
01118            fprintf( fp, ptrfmtstr, Ablock->IA[i] + (i+1) );
01119            if ( (i+1) % nptrline ) == 0 ) {
01120                fprintf( fp, "\n" );
01121            }
01122        }
01123
01124        if ( (ncol+1) % nptrline ) != 0 ) {
01125            fprintf( fp, "\n" );
01126        }
01127
01128        /* Step 3: Print the index information */
01129
01130        j = 0;
01131        for (i=0; i<ncol; i++) {
01132            fprintf( fp, indfmtstr, i+1); /* diagonal */
01133            if ( (j+1) % nindline ) == 0 ) {
01134                fprintf( fp, "\n" );
01135            }
01136            j++;
01137            for (jj=IA[i]; jj<IA[i+1]; jj++) {
01138                fprintf( fp, indfmtstr, JA[jj] + 1 ); /* lower triangle */
01139                if ( (jj+1) % nindline ) == 0 ) {
01140                    fprintf( fp, "\n" );
01141                }
01142                j++;
01143            }
01144        }
01145
01146        if ( ( j % nindline ) != 0 ) {
01147            fprintf( fp, "\n" );
01148        }
01149
01150        /* Step 4: Print the value information */
01151
01152        j = 0;
01153        for (i=0; i<ncol; i++) {
01154            fprintf( fp, valfmtstr, D[i] );
01155            if ( (j+1) % nvalline ) == 0 ) {
01156                fprintf( fp, "\n" );
01157            }
01158            j++;
01159            for (jj=IA[i]; jj<IA[i+1]; jj++) {
01160                fprintf( fp, valfmtstr, L[jj] );
01161                if ( (jj+1) % nvalline ) == 0 ) {
01162                    fprintf( fp, "\n" );
01163                }
01164                j++;
01165            }
01166        }
01167
01168        if ( ( j % nvalline ) != 0 ) {
01169            fprintf( fp, "\n" );
01170        }
01171
01172    } else { /* ISNOT_SYM */
01173

```

```

01174     VASSERT( 0 ); /* NOT CODED YET */
01175 }
01176
01177 /* Step 5: Close the file */
01178 fclose( fp );
01179 }
01180
01181 VPUBLIC PDE* Vfetk_PDE_ctor(Vfetk *fetk) {
01182
01183     PDE *thee = VNULL;
01184
01185     thee = (PDE*)Vmem_malloc(fetk->vmem, 1, sizeof(PDE));
01186     VASSERT(thee != VNULL);
01187     VASSERT(Vfetk_PDE_ctor2(thee, fetk));
01188
01189     return thee;
01190 }
01191
01192 VPUBLIC int Vfetk_PDE_ctor2(PDE *thee,
01193                               Vfetk *fetk
01194                               ) {
01195
01196     int i;
01197
01198     if (thee == VNULL) {
01199         Vnm_print(2, "Vfetk_PDE_ctor2: Got NULL thee!\n");
01200         return 0;
01201     }
01202
01203     /* Store a local copy of the Vfetk class */
01204     var.fetk = fetk;
01205
01206     /* PDE-specific parameters and function pointers */
01207     thee->initAssemble = Vfetk_PDE_initAssemble;
01208     thee->initElement = Vfetk_PDE_initElement;
01209     thee->initFace = Vfetk_PDE_initFace;
01210     thee->initPoint = Vfetk_PDE_initPoint;
01211     thee->Fu = Vfetk_PDE_Fu;
01212     thee->Fu_v = Vfetk_PDE_Fu_v;
01213     thee->DFu_wv = Vfetk_PDE_DFu_wv;
01214     thee->delta = Vfetk_PDE_delta;
01215     thee->u_D = Vfetk_PDE_u_D;
01216     thee->u_T = Vfetk_PDE_u_T;
01217     thee->Ju = Vfetk_PDE_Ju;
01218     thee->vec = 1; /* FIX! */
01219     thee->sym[0][0] = 1;
01220     thee->est[0] = 1.0;
01221     for (i=0; i<VMAX_BDTYPE; i++) thee->bmap[0][i] = i;
01222
01223     /* Manifold-specific function pointers */
01224     thee->bisectEdge = Vfetk_PDE_bisectEdge;
01225     thee->mapBoundary = Vfetk_PDE_mapBoundary;
01226     thee->markSimplex = Vfetk_PDE_markSimplex;
01227     thee->oneChart = Vfetk_PDE_oneChart;
01228
01229     /* Element-specific function pointers */
01230     thee->simplexBasisInit = Vfetk_PDE_simplexBasisInit
01231 ;
01232     thee->simplexBasisForm = Vfetk_PDE_simplexBasisForm
01233 ;
01234
01235     return 1;
01236 }
01237
01238 VPUBLIC void Vfetk_PDE_dtor(PDE **thee) {
01239
01240     if ((*thee) != VNULL) {
01241         Vfetk_PDE_dtor2(*thee);
01242
01243     /* TODO: The following line is commented out because at the moment,
01244     there is a seg fault when deallocating at the end of a run. Since
01245     this routine is called only once at the very end, we'll leave it
01246     commented out. However, this could be a memory leak.
01247 */
01248     /* Vmem_free(var.fetk->vmem, 1, sizeof(PDE), (void **)thee); */
01249     (*thee) = VNULL;
01250 }
01251
01252 VPUBLIC void Vfetk_PDE_dtor2(PDE *thee) {
01253     var.fetk = VNULL;

```

```

01253 }
01254
01255 VPRIIVATE double smooth(int nverts, double dist[VAPBS_NVS], double
01256   coeff[VAPBS_NVS], int meth) {
01257   int i;
01258   double weight;
01259   double num = 0.0;
01260   double den = 0.0;
01261
01262   for (i=0; i<nverts; i++) {
01263     if (dist[i] < VSMALL) return coeff[i];
01264     weight = 1.0/dist[i];
01265     if (meth == 0) {
01266       num += (weight * coeff[i]);
01267       den += weight;
01268     } else if (meth == 1) {
01269       /* Small coefficients reset the average to 0; we need to break out
01270        * of the loop */
01271       if (coeff[i] < VSMALL) {
01272         num = 0.0;
01273         break;
01274       } else {
01275         num += weight; den += (weight/coeff[i]);
01276       }
01277     } else VASSERT(0);
01278   }
01279
01280   return (num/den);
01281
01282 }
01283
01284 VPRIIVATE double diel() {
01285
01286   int i, j;
01287   double eps, epsp, epsw, dist[5], coeff[5], srad, swin, *vx;
01288   Vsurf_Meth srfm;
01289   Vacc *acc;
01290   PBEparm *pbeparm;
01291
01292   epsp = Vpbe_getSoluteDiel(var.fetk->pbe);
01293   epsw = Vpbe_getSolventDiel(var.fetk->pbe);
01294   VASSERT(var.fetk->pbeparm != VNNULL);
01295   pbeparm = var.fetk->pbeparm;
01296   srfm = pbeparm->srfm;
01297   srad = pbeparm->srad;
01298   swin = pbeparm->swin;
01299   acc = var.fetk->pbe->acc;
01300
01301   eps = 0;
01302
01303   if (VABS(epsp - epsw) < VSMALL) return epsp;
01304   switch (srfm) {
01305     case VSM_MOL:
01306       eps = ((epsw-epsp)*Vacc_molAcc(acc, var.xq, srad) +
01307         epsp);
01308       break;
01309     case VSM_MOLSMOOTH:
01310       for (i=0; i<var.nverts; i++) {
01311         dist[i] = 0;
01312         vx = var.vx[i];
01313         for (j=0; j<3; j++) {
01314           dist[i] += VSQR(var.xq[j] - vx[j]);
01315         }
01316         dist[i] = VSQRT(dist[i]);
01317         coeff[i] = (epsw-epsp)*Vacc_molAcc(acc, var.xq,
01318           srad) + epsp;
01319       }
01320       eps = smooth(var.nverts, dist, coeff, 1);
01321       break;
01322     case VSM_SPLINE:
01323       eps = ((epsw-epsp)*Vacc_splineAcc(acc, var.xq, swin
01324         , 0.0) + epsp);
01325       break;
01326     default:
01327       Vnm_print(2, "Undefined surface method (%d)!\n", srfm);
01328       VASSERT(0);
01329   }
01330
01331   return eps;
01332 }
01333 }
```

```

01330
01331 VPRIVATE double ionacc() {
01332
01333     int i, j;
01334     double dist[5], coeff[5], irad, swin, *vx, accval;
01335     Vsurf_Meth srfm;
01336     Vac *acc = VNULL;
01337     PBEparm *pbeparm = VNULL;
01338
01339     VASSERT(var.fetk->pbeparm != VNULL);
01340     pbeparm = var.fetk->pbeparm;
01341     srfm = pbeparm->srfm;
01342     irad = Vpbe_getMaxIonRadius(var.fetk->pbe);
01343     swin = pbeparm->swin;
01344     acc = var.fetk->pbe->acc;
01345
01346     if (var.zks2 < VSMALL) return 0.0;
01347     switch (srfm) {
01348         case VSM_MOL:
01349             accval = Vacc_ivdwAcc(acc, var.xq, irad);
01350             break;
01351         case VSM_MOLSMOOTH:
01352             for (i=0; i<var.nverts; i++) {
01353                 dist[i] = 0;
01354                 vx = var.vx[i];
01355                 for (j=0; j<3; j++) {
01356                     dist[i] += VSQR(var.xq[j] - vx[j]);
01357                 }
01358                 dist[i] = VSQRT(dist[i]);
01359                 coeff[i] = Vacc_ivdwAcc(acc, var.xq, irad);
01360             }
01361             accval = smooth(var.nverts, dist, coeff, 1);
01362             break;
01363         case VSM_SPLINE:
01364             accval = Vacc_splineAcc(acc, var.xq, swin, irad);
01365             break;
01366         default:
01367             Vnm_print(2, "Undefined surface method (%d)!\n", srfm);
01368             VASSERT(0);
01369     }
01370
01371     return accval;
01372 }
01373
01374 VPRIVATE double debye_U(Vpbe *pbe, int d, double x[]) {
01375
01376     double size, *position, charge, xkappa, eps_w, dist, T, pot, val;
01377     int iatom, i;
01378     Valist *alist;
01379     Vatom *atom;
01380
01381     eps_w = Vpbe_getSolventDiel(pbe);
01382     xkappa = (1.0e10)*Vpbe_getXkappa(pbe);
01383     T = Vpbe_getTemperature(pbe);
01384     alist = Vpbe_getValist(pbe);
01385     val = 0;
01386     pot = 0;
01387
01388     for (iatom=0; iatom<Valist_getNumberAtoms(alist);
01389     iatom++) {
01390         atom = Valist_getAtom(alist, iatom);
01391         position = Vatom_getPosition(atom);
01392         charge = Vunit_ec*Vatom_getCharge(atom);
01393         size = (1e-10)*Vatom_getRadius(atom);
01394         dist = 0;
01395         for (i=0; i<d; i++) {
01396             dist += VSQR(position[i] - x[i]);
01397         }
01398         dist = (1.0e-10)*VSQRT(dist);
01399         val = (charge)/(4*VPI*Vunit_eps0*eps_w*dist);
01400         if (xkappa != 0.0) {
01401             val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
01402         }
01403         pot = pot + val;
01404     }
01405     pot = pot*Vunit_ec/(Vunit_kb*T);
01406
01407     return pot;
01408 }
01409 VPRIVATE double debye_Udiff(Vpbe *pbe, int d, double x[]) {

```

```

01410
01411     double size, *position, charge, eps_p, dist, T, pot, val;
01412     double Ufull;
01413     int iatom, i;
01414     Valist *alist;
01415     Vatom *atom;
01416
01417     Ufull = debye_U(pbe, d, x);
01418
01419     eps_p = Vpbe_getSoluteDiel(pbe);
01420     T = Vpbe_getTemperature(pbe);
01421     alist = Vpbe_getValist(pbe);
01422     val = 0;
01423     pot = 0;
01424
01425     for (iatom=0; iatom<Valist_getNumberAtoms(alist);
01426         iatom++) {
01427         atom = Valist_getAtom(alist, iatom);
01428         position = Vatom_getPosition(atom);
01429         charge = Vunit_ec*Vatom_getCharge(atom);
01430         size = (1e-10)*Vatom_getRadius(atom);
01431         dist = 0;
01432         for (i=0; i<d; i++) {
01433             dist += VSQR(position[i] - x[i]);
01434         }
01435         dist = (1.0e-10)*VSQRT(dist);
01436         val = (charge)/(4*VPI*Vunit_eps0*eps_p*dist);
01437         pot = pot + val;
01438     }
01439     pot = pot*Vunit_ec/(Vunit_kb*T);
01440
01441     pot = Ufull - pot;
01442
01443     return pot;
01444 }
01445
01446 VPRIvate void coulomb(Vpbe *pbe, int d, double pt[], double eps, double *U,
01447
01448     double dU[], double *d2U) {
01449
01450     int iatom, i;
01451     double T, pot, fx, fy, fz, x, y, z, scale;
01452     double *position, charge, dist, dist2, val, vec[3], dUold[3], Uold;
01453     Valist *alist;
01454     Vatom *atom;
01455
01456     /* Initialize variables */
01457     T = Vpbe_getTemperature(pbe);
01458     alist = Vpbe_getValist(pbe);
01459     pot = 0; fx = 0; fy = 0; fz = 0;
01460     x = pt[0]; y = pt[1]; z = pt[2];
01461
01462     /* Calculate */
01463     if (!Vgreen_coulombD(var.green, 1, &x, &y, &z, &pot, &
01464     fx, &fy, &fz)) {
01465         Vnm_prin(2, "Error calculating Green's function!\n");
01466         VASSErt(0);
01467     }
01468
01469     /* Scale the results */
01470     scale = Vunit_ec/(eps*Vunit_kb*T);
01471     *U = pot*scale;
01472     *d2U = 0.0;
01473     dU[0] = -fx*scale;
01474     dU[1] = -fy*scale;
01475     dU[2] = -fz*scale;
01476
01477 #if 0
01478     /* Compare with old results */
01479     val = 0.0;
01480     Uold = 0.0; dUold[0] = 0.0; dUold[1] = 0.0; dUold[2] = 0.0;
01481     for (iatom=0; iatom<Valist_getNumberAtoms(alist);
01482         iatom++) {
01483         atom = Valist_getAtom(alist, iatom);
01484         position = Vatom_getPosition(atom);
01485         charge = Vatom_getCharge(atom);
01486         dist2 = 0;
01487         for (i=0; i<d; i++) {
01488             vec[i] = (position[i] - pt[i]);
01489             dist2 += VSQR(vec[i]);
01490         }
01491         val += (charge)/(4*VPI*Vunit_eps0*eps_p*dist2);
01492     }
01493     pot = val;
01494 #endif

```

```

01487         }
01488         dist = VSQRT(dist2);
01489
01490         /* POTENTIAL */
01491         Uold = Uold + charge/dist;
01492
01493         /* GRADIENT */
01494         for (i=0; i<d; i++) dUold[i] = dUold[i] + vec[i]*charge/(dist2*dist);
01495
01496     }
01497     Uold = Uold*VSQR(Vunit_ec)*(1.0e10)/(4*VPI*Vunit_eps0*eps
01498     *Vunit_kb*T);
01499     for (i=0; i<d; i++) {
01500         dUold[i] = dUold[i]*VSQR(Vunit_ec)*(1.0e10)/(4*VPI*Vunit_eps0
01501         *eps*Vunit_kb*T);
01502         printf("Unew - Uold = %g - %g = %g\n", *U, Uold, (*U - Uold));
01503         printf("||dUnew - dUold||^2 = %g\n", (VSQR(dU[0] - dUold[0])
01504             + VSQR(dU[1] - dUold[1]) + VSQR(dU[2] - dUold[2])));
01505         printf("dUnew[0] = %g, dUold[0] = %g\n", dU[0], dUold[0]);
01506         printf("dUnew[1] = %g, dUold[1] = %g\n", dU[1], dUold[1]);
01507         printf("dUnew[2] = %g, dUold[2] = %g\n", dU[2], dUold[2]);
01508
01509 #endif
01510
01511 }
01512
01513 VPUBLIC void Vfetk_PDE_initAssemble(PDE *thee, int ip[],
01514     double rp[]) {
01515 #if 1
01516     /* Re-initialize the Green's function oracle in case the atom list has
01517      * changed */
01518     if (var.initGreen) {
01519         Vgreen_dtor(&(var.green));
01520         var.initGreen = 0;
01521     }
01522     var.green = Vgreen_ctor(var.fetk->pbe->alist);
01523     var.initGreen = 1;
01524 #else
01525     if (!var.initGreen) {
01526         var.green = Vgreen_ctor(var.fetk->pbe->alist
01527     );
01528     }
01529 #endif
01530
01531 }
01532
01533 VPUBLIC void Vfetk_PDE_initElement(PDE *thee, int
01534     elementType, int chart,
01535     double tvx[][3], void *data) {
01536
01537     int i, j;
01538     double epsp, epsw;
01539
01540     /* We assume that the simplex has been passed in as the void *data * *
01541      * argument. Store it */
01542     VASSERT(data != NULL);
01543     var.simp = (SS *)data;
01544
01545     /* save the element type */
01546     var.sType = elementType;
01547
01548     /* Grab the vertices from this simplex */
01549     var.nverts = thee->dim+1;
01550     for (i=0; i<thee->dim+1; i++) var.verts[i] = SS_vertex(var.simp, i
01551 );
01552
01553     /* Vertex locations of this simplex */
01554     for (i=0; i<thee->dim+1; i++) {
01555         for (j=0; j<thee->dim; j++) {
01556             var.vx[i][j] = tvx[i][j];
01557         }
01558
01559     /* Set the dielectric constant for this element for use in the jump term *
01560      * of the residual-based error estimator. The value is set to the average
01561      * * value of the vertices */
01562     var.jumpDiel = 0; /* NOT IMPLEMENTED YET! */

```

```

01562 }
01563
01564 VPUBLIC void Vfetk_PDE_initFace(PDE *thee, int faceType, int
01565   chart,
01566   double tnvec[]) {
01567   int i;
01568
01569   /* unit normal vector of this face */
01570   for (i=0; i<thee->dim; i++) var.nvec[i] = tnvec[i];
01571
01572   /* save the face type */
01573   var.fType = faceType;
01574 }
01575
01576 VPUBLIC void Vfetk_PDE_initPoint(PDE *thee, int pointType,
01577   int chart,
01578   double txq[], double tU[], double tdU[] [3]) {
01579   int i, j, ichop;
01580   double u2, coef2, eps_p;
01581   Vhal_PBEType pdetype;
01582   Vpbe *pbe = VNULL;
01583
01584   eps_p = Vpbe_getSoluteDiel(var.fetk->pbe);
01585   pdetype = var.fetk->type;
01586   pbe = var.fetk->pbe;
01587
01588   /* the point, the solution value and gradient, and the Coulomb value and *
01589   * gradient at the point */
01590   if ((pdetype == PBE_LRPBE) || (pdetype == PBE_NRPBE)) {
01591     coulomb(pbe, thee->dim, txq, eps_p, &(var.W), var.dW, &(var.d2W))
01592   }
01593   for (i=0; i<thee->vec; i++) {
01594     var.U[i] = tU[i];
01595     for (j=0; j<thee->dim; j++) {
01596       var.xq[j] = txq[j];
01597       var.dU[i][j] = tdU[i][j];
01598     }
01599   }
01600
01601 /* interior form case */
01602 if (pointType == 0) {
01603
01604   /* Get the dielectric values */
01605   var.diel = diel();
01606   var.ionacc = ionacc();
01607   var.A = var.diel;
01608   var.F = (var.diel - eps_p);
01609
01610   switch (pdetype) {
01611
01612     case PBE_LPBE:
01613       var.DB = var.ionacc*var.zkappa2*var.ionstr;
01614       var.B = var.DB*var.U[0];
01615       break;
01616
01617     case PBE_NPBE:
01618
01619       var.B = 0;
01620       var.DB = 0;
01621       if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01622         for (i=0; i<var.nion; i++) {
01623           u2 = -1.0 * var.U[0] * var.ionQ[i];
01624
01625           /* NONLINEAR TERM */
01626           coef2 = -1.0 * var.ionacc * var.zks2 * var.ionConc
01627           [i];
01628           var.B += (coef2 * Vcap_exp(u2, &ichop));
01629           /* LINEARIZED TERM */
01630           coef2 = -1.0 * var.ionQ[i] * coef2;
01631           var.DB += (coef2 * Vcap_exp(u2, &ichop));
01632         }
01633       }
01634       break;
01635
01636     case PBE_LRPBE:
01637       var.DB = var.ionacc*var.zkappa2*var.ionstr;
01638       var.B = var.DB*(var.U[0]+var.W);
01639       break;

```

```

01639
01640     case PBE_NRPBE:
01641
01642         var.B = 0;
01643         var.DB = 0;
01644         if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01645             for (i=0; i<var.nion; i++) {
01646                 u2 = -1.0 * (var.U[0] + var.W) * var.ionQ[i];
01647
01648                 /* NONLINEAR TERM */
01649                 coef2 = -1.0 * var.ionacc * var.zks2 * var.ionConc
01650                 [i];
01651                 var.B += (coef2 * Vcap_exp(u2, &ichop));
01652
01653                 /* LINEARIZED TERM */
01654                 coef2 = -1.0 * var.ionQ[i] * coef2;
01655                 var.DB += (coef2 * Vcap_exp(u2, &ichop));
01656             }
01657             break;
01658
01659     case PBE_SMPBE: /* SMPBE Temp */
01660
01661         var.B = 0;
01662         var.DB = 0;
01663         if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01664             for (i=0; i<var.nion; i++) {
01665                 u2 = -1.0 * var.U[0] * var.ionQ[i];
01666
01667                 /* NONLINEAR TERM */
01668                 coef2 = -1.0 * var.ionacc * var.zks2 * var.ionConc
01669                 [i];
01670                 var.B += (coef2 * Vcap_exp(u2, &ichop));
01671                 /* LINEARIZED TERM */
01672                 coef2 = -1.0 * var.ionQ[i] * coef2;
01673                 var.DB += (coef2 * Vcap_exp(u2, &ichop));
01674             }
01675             break;
01676         default:
01677             Vnm_print(2, "Vfetk_PDE_initPoint: Unknown PBE type (%d)!\n",
01678                     pdetype);
01679             VASSERT(0);
01680             break;
01681         }
01682
01683
01684     /* boundary form case */
01685 } else {
01686 #ifdef DONEUMANN
01687     ;
01688 #else
01689     Vnm_print(2, "Vfetk: Whoa! I just got a boundary point to evaluate
01690     (%d)!\n", pointType);
01691     Vnm_print(2, "Vfetk: Did you do that on purpose?\n");
01692 #endif
01693 }
01694 #if 0 /* THIS IS VERY NOISY! */
01695     Vfetk_dumpLocalVar();
01696 #endif
01697
01698 }
01699
01700 VPUBLIC void Vfetk_PDE_Fu(PDE *thee, int key, double F[]) {
01701
01702     //Vnm_print(2, "Vfetk_PDE_Fu: Setting error to zero!\n");
01703
01704     F[0] = 0.;
01705
01706 }
01707
01708 VPUBLIC double Vfetk_PDE_Fu_v(
01709     PDE *thee,
01710     int key,
01711     double V[],
01712     double dV[] [VAPBS_DIM]
01713 ) {
01714
01715     Vhal_PBEType type;
01716     int i;

```

```

01717     double value = 0.;
01718
01719     type = var.fetk->type;
01720
01721     /* interior form case */
01722     if (key == 0) {
01723
01724         for (i=0; i<thee->dim; i++) value += ( var.A * var.dU[0][i] * dV[0][
01725             i] );
01726         value += var.B * V[0];
01727
01728         if ((type == PBE_LRPBE) || (type == PBE_NRPBE)) {
01729             for (i=0; i<thee->dim; i++) {
01730                 if (var.F > VSMALL) value += (var.F * var.dW[i] * dV[0][i])
01731             }
01732         }
01733     /* boundary form case */
01734     } else {
01735 #ifdef DONEUMANN
01736         value = 0.0;
01737 #else
01738         Vnm_print(2, "Vfetk: Whoa! I was just asked to evaluate a boundary
01739         weak form for point type %d!\n", key);
01740     }
01741
01742     var.Fu_v = value;
01743     return value;
01744 }
01745
01746 VPUBLIC double Vfetk_PDE_DFu_wv(
01747     PDE *thee,
01748     int key,
01749     double W[],
01750     double dW[] [VAPBS_DIM],
01751     double V[],
01752     double dV[] [3]
01753 ) {
01754
01755     Vhal_PBEType type;
01756     int i;
01757     double value = 0. ;
01758
01759     type = var.fetk->type;
01760
01761     /* Interior form */
01762     if (key == 0) {
01763         value += var.DB * W[0] * V[0];
01764         for (i=0; i<thee->dim; i++) value += ( var.A * dW[0][i] * dV[0][i]
01765     );
01766
01767     /* boundary form case */
01768     } else {
01769 #ifdef DONEUMANN
01770         value = 0.0;
01771     } else
01772         Vnm_print(2, "Vfetk: Whoa! I was just asked to evaluate a boundary
01773         weak form for point type %d!\n", key);
01774     }
01775
01776     var.DFu_wv = value;
01777     return value;
01778 }
01779
01780 #define VRINGMAX 1000
01781
01782
01783 VPUBLIC void Vfetk_PDE_delta(PDE *thee, int type, int chart,
01784     double txq[],
01785     void *user, double F[] ) {
01786
01787     int iatom, jatom, natoms, atomIndex, atomList[VATOMMAX], nAtomList;
01788     int gotAtom, numString, isimp, invert, sid;
01789     double *position, charge, phi[VAPBS_NV], phix[VAPBS_NV]
01790     [3], value;
01791     Vatom *atom;
01792     Vhal_PBEType pdETYPE;
01793     SS *string[VRINGMAX];

```

```

01794     VV *vertex = (VV *)user;
01795
01796     pdetype = var.fetk->type;
01797
01798     F[0] = 0.0;
01799
01800     if ((pdetype == PBE_LPBE) || (pdetype == PBE_NPBE) || (
01801         pdetype == PBE_SMPBE) /* SMPBE Added */) {
01802         VASSERT( vertex != VNULL);
01803         numString = 0;
01804         sring[numString] = VV_firstSS(vertex);
01805         while (sring[numString] != VNULL) {
01806             numString++;
01807             sring[numString] = SS_link(sring[numString-1], vertex);
01808         }
01809         VASSERT( numString > 0 );
01810         VASSERT( numString <= VRINGMAX );
01811
01812         /* Move around the simplex ring and determine the charge locations */
01813         F[0] = 0.;
01814         charge = 0.;
01815         nAtomList = 0;
01816         for (isimp=0; isimp<numString; isimp++) {
01817             sid = SS_id(sring[isimp]);
01818             natoms = Vcsm_getNumberAtoms(Vfetk_getVcsm
01819                 (var.fetk), sid);
01820             for (iatom=0; iatom<natoms; iatom++) {
01821                 /* Get the delta function information */
01822                 atomIndex = Vcsm_getAtomIndex(Vfetk_getVcsm
01823                     (var.fetk),
01824                         iatom, sid);
01825                 gotAtom = 0;
01826                 for (jatom=0; jatom<nAtomList; jatom++) {
01827                     if (atomList[jatom] == atomIndex) {
01828                         gotAtom = 1;
01829                         break;
01830                     }
01831                 }
01832                 if (!gotAtom) {
01833                     VASSERT(nAtomList < VATOMMAX);
01834                     atomList[nAtomList] = atomIndex;
01835                     nAtomList++;
01836
01837                     atom = Vcsm_getAtom(Vfetk_getVcsm(
01838                         var.fetk), iatom, sid);
01839                     charge = Vatom_getCharge(atom);
01840                     position = Vatom_getPosition(atom);
01841
01842                     /* Get the test function value at the delta function I
01843                     * used to do a VASSERT to make sure the point was in the
01844                     * simplex (i.e., make sure round-off error isn't an
01845                     * issue), but round off errors became an issue */
01846                     if (!Gem_pointInSimplexVal(Vfetk_getGem(var.
01847                         fetk),
01848                             sring[isimp], position, phi, phix)) {
01849                         if (!Gem_pointInSimplex(Vfetk_getGem(var.
01850                             fetk),
01851                                 sring[isimp], position)) {
01852                                     Vnm_print(2, "delta: Both Gem_pointInSimplexVal \
01853 and Gem_pointInSimplex detected misplaced point charge!\n");
01854                                     Vnm_print(2, "delta: I think you have problems: \
01855 phi = \"%");
01856                                     for (ivert=0; ivert<Gem_dimVV(Vfetk_getGem
01857                                         (var.fetk)); ivert++) Vnm_print(2, "%e ", phi[ivert]);
01858
01859                                     Vnm_print(2, "}\n");
01860
01861                                     }
01862                                     value = 0;
01863                                     for (ivert=0; ivert<Gem_dimVV(Vfetk_getGem(var.
01864                                         fetk)); ivert++) {
01865                                         if (VV_id(SS_vertex(sring[isimp], ivert)) == VV_id(
01866                                             vertex)) value += phi[ivert];
01867                                         }
01868
01869                                         F[0] += (value * Vpbe_getZmagic(var.fetk
01870                                         ->pbe) * charge);
01871                                         } /* if !gotAtom */
01872                                         } /* for iatom */
01873                                         } /* for isimp */
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01960
01961
01962
01963
01964
01965
01966
01967
01968
01969
01970
01971
01972
01973
01974
01975
01976
01977
01978
01979
01980
01981
01982
01983
01984
01985
01986
01987
01988
01989
01990
01991
01992
01993
01994
01995
01996
01997
01998
01999
02000
02001
02002
02003
02004
02005
02006
02007
02008
02009
02010
02011
02012
02013
02014
02015
02016
02017
02018
02019
02020
02021
02022
02023
02024
02025
02026
02027
02028
02029
02030
02031
02032
02033
02034
02035
02036
02037
02038
02039
02040
02041
02042
02043
02044
02045
02046
02047
02048
02049
02050
02051
02052
02053
02054
02055
02056
02057
02058
02059
02060
02061
02062
02063
02064
02065
02066
02067
02068
02069
02070
02071
02072
02073
02074
02075
02076
02077
02078
02079
02080
02081
02082
02083
02084
02085
02086
02087
02088
02089
02090
02091
02092
02093
02094
02095
02096
02097
02098
02099
02100
02101
02102
02103
02104
02105
02106
02107
02108
02109
02110
02111
02112
02113
02114
02115
02116
02117
02118
02119
02120
02121
02122
02123
02124
02125
02126
02127
02128
02129
02130
02131
02132
02133
02134
02135
02136
02137
02138
02139
02140
02141
02142
02143
02144
02145
02146
02147
02148
02149
02150
02151
02152
02153
02154
02155
02156
02157
02158
02159
02160
02161
02162
02163
02164
02165
02166
02167
02168
02169
02170
02171
02172
02173
02174
02175
02176
02177
02178
02179
02180
02181
02182
02183
02184
02185
02186
02187
02188
02189
02190
02191
02192
02193
02194
02195
02196
02197
02198
02199
02200
02201
02202
02203
02204
02205
02206
02207
02208
02209
02210
02211
02212
02213
02214
02215
02216
02217
02218
02219
02220
02221
02222
02223
02224
02225
02226
02227
02228
02229
02230
02231
02232
02233
02234
02235
02236
02237
02238
02239
02240
02241
02242
02243
02244
02245
02246
02247
02248
02249
02250
02251
02252
02253
02254
02255
02256
02257
02258
02259
02260
02261
02262
02263
02264
02265
02266
02267
02268
02269
02270
02271
02272
02273
02274
02275
02276
02277
02278
02279
02280
02281
02282
02283
02284
02285
02286
02287
02288
02289
02290
02291
02292
02293
02294
02295
02296
02297
02298
02299
02300
02301
02302
02303
02304
02305
02306
02307
02308
02309
02310
02311
02312
02313
02314
02315
02316
02317
02318
02319
02320
02321
02322
02323
02324
02325
02326
02327
02328
02329
02330
02331
02332
02333
02334
02335
02336
02337
02338
02339
02340
02341
02342
02343
02344
02345
02346
02347
02348
02349
02350
02351
02352
02353
02354
02355
02356
02357
02358
02359
02360
02361
02362
02363
02364
02365
02366
02367
02368
02369
02370
02371
02372
02373
02374
02375
02376
02377
02378
02379
02380
02381
02382
02383
02384
02385
02386
02387
02388
02389
02390
02391
02392
02393
02394
02395
02396
02397
02398
02399
02400
02401
02402
02403
02404
02405
02406
02407
02408
02409
02410
02411
02412
02413
02414
02415
02416
02417
02418
02419
02420
02421
02422
02423
02424
02425
02426
02427
02428
02429
02430
02431
02432
02433
02434
02435
02436
02437
02438
02439
02440
02441
02442
02443
02444
02445
02446
02447
02448
02449
02450
02451
02452
02453
02454
02455
02456
02457
02458
02459
02460
02461
02462
02463
02464
02465
02466
02467
02468
02469
02470
02471
02472
02473
02474
02475
02476
02477
02478
02479
02480
02481
02482
02483
02484
02485
02486
02487
02488
02489
02490
02491
02492
02493
02494
02495
02496
02497
02498
02499
02500
02501
02502
02503
02504
02505
02506
02507
02508
02509
02510
02511
02512
02513
02514
02515
02516
02517
02518
02519
02520
02521
02522
02523
02524
02525
02526
02527
02528
02529
02530
02531
02532
02533
02534
02535
02536
02537
02538
02539
02540
02541
02542
02543
02544
02545
02546
02547
02548
02549
02550
02551
02552
02553
02554
02555
02556
02557
02558
02559
02560
02561
02562
02563
02564
02565
02566
02567
02568
02569
02570
02571
02572
02573
02574
02575
02576
02577
02578
02579
02580
02581
02582
02583
02584
02585
02586
02587
02588
02589
02590
02591
02592
02593
02594
02595
02596
02597
02598
02599
02600
02601
02602
02603
02604
02605
02606
02607
02608
02609
02610
02611
02612
02613
02614
02615
02616
02617
02618
02619
02620
02621
02622
02623
02624
02625
02626
02627
02628
02629
02630
02631
02632
02633
02634
02635
02636
02637
02638
02639
02640
02641
02642
02643
02644
02645
02646
02647
02648
02649
02650
02651
02652
02653
02654
02655
02656
02657
02658
02659
02660
02661
02662
02663
02664
02665
02666
02667
02668
02669
02670
02671
02672
02673
02674
02675
02676
02677
02678
02679
02680
02681
02682
02683
02684
02685
02686
02687
02688
02689
02690
02691
02692
02693
02694
02695
02696
02697
02698
02699
02700
02701
02702
02703
02704
02705
02706
02707
02708
02709
02710
02711
02712
02713
02714
02715
02716
02717
02718
02719
02720
02721
02722
02723
02724
02725
02726
02727
02728
02729
02730
02731
02732
02733
02734
02735
02736
02737
02738
02739
02740
02741
02742
02743
02744
02745
02746
02747
02748
02749
02750
02751
02752
02753
02754
02755
02756
02757
02758
02759
02760
02761
02762
02763
02764
02765
02766
02767
02768
02769
02770
02771
02772
02773
02774
02775
02776
02777
02778
02779
02780
02781
02782
02783
02784
02785
02786
02787
02788
02789
02790
02791
02792
02793
02794
02795
02796
02797
02798
02799
02800
02801
02802
02803
02804
02805
02806
02807
02808
02809
02810
02811
02812
02813
02814
02815
02816
02817
02818
02819
02820
02821
02822
02823
02824
02825
02826
02827
02828
02829
02830
02831
02832
02833
02834
02835
02836
02837
02838
02839
02840
02841
02842
02843
02844
02845
02846
02847
02848
02849
02850
02851
02852
02853
02854
02855
02856
02857
02858
02859
02860
02861
02862
02863
02864
02865
02866
02867
02868
02869
02870
02871
02872
02873
02874
02875
02876
02877
02878
02879
02880
02881
02882
02883
02884
02885
02886
02887
02888
02889
02890
02891
02892
02893
02894
02895
02896
02897
02898
02899
02900
02901
02902
02903
02904
02905
02906
02907
02908
02909
02910
02911
02912
02913
02914
02915
02916
02917
02918
02919
02920
02921
02922
02923
02924
02925
02926
02927
02928
02929
02930
02931
02932
02933
02934
02935
02936
02937
02938
02939
02940
02941
02942
02943
02944
02945
02946
02947
02948
02949
02950
02951
02952
02953
02954
02955
02956
02957
02958
02959
02960
02961
02962
02963
02964
02965
02966
02967
02968
02969
02970
02971
02972
02973
02974
02975
02976
02977
02978
02979
02980
02981
02982
02983
02984
02985
02986
02987
02988
02989
02990
02991
02992
02993
02994
02995
02996
02997
02998
02999
03000
03001
03002
03003
03004
03005
03006
03007
03008
03009
03010
03011
03012
03013
03014
03015
03016
03017
03018
03019
03020
03021
03022
03023
03024
03025
03026
03027
03028
03029
03030
03031
03032
03033
03034
03035
03036
03037
03038
03039
03040
03041
03042
03043
03044
03045
03046
03047
03048
03049
03050
03051
03052
03053
03054
03055
03056
03057
03058
03059
03060
03061
03062
03063
03064
03065
03066
03067
03068
03069
03070
03071
03072
03073
03074
03075
03076
03077
03078
03079
03080
03081
03082
03083
03084
03085
03086
03087
03088
03089
03090
03091
03092
03093
03094
03095
03096
03097
03098
03099
03100
03101
03102
03103
03104
03105
03106
03107
03108
03109
03110
03111
03112
03113
03114
03115
03116
03117
03118
03119
03120
03121
03122
03123
03124
03125
03126
03127
03128
03129
03130
03131
03132
03133
03134
03135
03136
03137
03138
03139
03140
03141
03142
03143
03144
03145
03146
03147
03148
03149
03150
03151
03152
03153
03154
03155
03156
03157
03158
03159
03160
03161
03162
03163
03164
03165
03166
03167
03168
03169
03170
03171
03172
03173
03174
03175
03176
03177
03178
03179
03180
03181
03182
03183
03184
03185
03186
03187
03188
03189
03190
03191
03192
03193
03194
03195
03196
03197
03198
03199
03200
03201
03202
03203
03204
03205
03206
03207
03208
03209
03210
03211
03212
03213
03214
03215
03216
03217
03218
03219
03220
03221
03222
03223
03224
03225
03226
03227
03228
03229
03230
03231
03232
03233
03234
03235
03236
03237
03238
03239
03240
03241
03242
03243
03244
03245
03246
03247
03248
03249
03250
03251
03252
03253
03254
03255
03256
03257
03258
03259
03260
03261
03262
03263
03264
03265
03266
03267
03268
03269
03270
03271
03272
03273
03274
03275
03276
03277
03278
03279
03280
03281
03282
03283
03284
03285
03286
03287
03288
03289
03290
03291
03292
03293
03294
03295
03296
03297
03298
03299
03300
03301
03302
03303
03304
03305
03306
03307
03308
03309
03310
03311
03312
03313
03314
03315
03316
03317
03318
03319
03320
03321
03322
03323
03324
03325
03326
03327
03328
03329
03330
03331
03332
03333
03334
03335
03336
03337
03338
03339
03340
03341
03342
03343
03344
03345
03346
03347
03348
03349
03350
03351
03352
03353
03354
03355
03356
03357
03358
03359
03360
03361
03362
03363
03364
03365
03366
03367
03368
03369
03370
03371
03372
03373
03374
03375
03376
03377
03378
03379
03380
03381
03382
03383
03384
03385
03386
03387
03388
03389
03390
03391
03392
03393
03394
03395
03396
03397
03398
03399
03400
03401
03402
03403
03404
03405
03406
03407
03408
03409
03410
03411
03412
03413
03414
03415
03416
03417
03418
03419
03420
03421
03422
03423
03424
03425
03426
03427
03428
03429
03430
03431
03432
03433
03434
03435
03436
03437
03438
03439
03440
03441
03442
03443
03444
03445
03446
03447
03448
03449
03450
03451
03452
03453
03454
03455
03456
03457
03458
03459
03460
03461
03462
03463
03464
03465
03466
03467
03468
03469
03470
03471
03472
03473
03474
03475
03476
03477
03478
03479
03480
03481
03482
03483
03484
03485
03486
03487
03488
03489
03490
03491
03492
03493
03494
03495
03496
03497
03498
03499
03500
03501
03502
03503
03504
03505
03506
03507
03508
03509
03510
03511
03512
03513
03514
03515
03516
03517
03518
03519
03520
03521
03522
03523
03524
03525
03526
03527
03528
03529
03530
03531
03532
03533
03534
03535
03536
03537
03538
03539
03540
03541
03542
03543
03544
03545
03546
03547
03548
03549
03550
03551
03552
03553
03
```

```

01864     } else if ((pdetype == PBE_LRPBE) || (pdetype == PBE_NRPBE)) {
01865         F[0] = 0.0;
01866     } else { VASSERT(0); }
01867
01868     var.delta = F[0];
01869
01870 }
01871
01872 VPUBLIC void Vfetk_PDE_u_D(PDE *thee, int type, int chart, double
01873     txq[],
01874     double F[]) {
01875
01876     if ((var.fetk->type == PBE_LPBE) || (var.fetk->type
01877         == PBE_NPBE) || (var.fetk->type == PBE_SMPBE) /* SMPBE
01878         Added */ ) {
01879         F[0] = debye_U(var.fetk->pbe, thee->dim, txq);
01880     } else if ((var.fetk->type == PBE_LRPBE) || (var.fetk
01881         ->type == PBE_NRPBE)) {
01882         F[0] = debye_Udiff(var.fetk->pbe, thee->dim, txq);
01883     } else VASSERT(0);
01884
01885     var.u_D = F[0];
01886
01887 }
01888
01889
01890 VPUBLIC void Vfetk_PDE_u_T(PDE *thee, int type, int chart, double
01891     txq[],
01892     double F[]) {
01893 /*VPUBLIC void Vfetk_PDE_u_T(sPDE *thee,
01894     int type,
01895     int chart,
01896     double txq[],
01897     double F[],
01898     double dF[][3]
01899 ) { */
01900
01901     F[0] = 0.0;
01902     var.u_T = F[0];
01903
01904 }
01905
01906
01907 VPUBLIC void Vfetk_PDE_bisectEdge(int dim, int dimII, int
01908     edgeType,
01909     int chart[], double vx[][3]) {
01910
01911     int i;
01912
01913     for (i=0; i<dimII; i++) vx[2][i] = .5 * (vx[0][i] + vx[1][i]);
01914     chart[2] = chart[0];
01915 }
01916
01917 VPUBLIC void Vfetk_PDE_mapBoundary(int dim, int dimII, int
01918     vertexType,
01919     int chart, double vx[3]) {
01920 }
01921
01922 VPUBLIC int Vfetk_PDE_markSimplex(int dim, int dimII, int
01923     simplexType,
01924     int faceType[VAPBS_NVS], int vertexType[VAPBS_NVS], int chart[], double vx[] [
01925     3],
01926     void *simplex) {
01927
01928     double targetRes, edgeLength, srad, swin, myAcc, refAcc;
01929     int i, natoms;
01930     Vsurf_Meth srfm;
01931     Vhal_PBEType type;
01932     FEmparm *feparm = VNULL;
01933     PBEparm *pbeparm = VNULL;
01934     Vpbe *pbe = VNULL;
01935     Vacc *acc = VNULL;
01936     Vcsm *csm = VNULL;
01937     SS *simp = VNULL;
01938
01939     VASSERT(var.fetk->feparm != VNULL);
01940     feparm = var.fetk->feparm;
01941     VASSERT(var.fetk->pbeparm != VNULL);
01942     pbeparm = var.fetk->pbeparm;
01943     pbe = var.fetk->pbe;

```

```

01942     csm = Vfetk_getVcsm(var.fetk);
01943     acc = pbe->acc;
01944     targetRes = feparm->targetRes;
01945     srfm = pbeparm->srfm;
01946     srad = pbeparm->srad;
01947     swin = pbeparm->swin;
01948     simp = (SS *)simplex;
01949     type = var.fetk->type;
01950
01951     /* Check to see if this simplex is smaller than the target size */
01952     /* NAB WARNING: I am providing face=-1 here to conform to the new MC API;
01953     however, I'm not sure if this is the correct behavior. */
01954     Gem_longestEdge(var.fetk->gm, simp, -1, &edgeLength);
01955     if (edgeLength < targetRes) return 0;
01956
01957     /* For non-regularized PBE, check charge-simplex map */
01958     if ((type == PBE_LPBE) || (type == PBE_NPBE) || (type ==
01959         PBE_SMPBE) /* SMPBE Added */) {
01960         natoms = Vcsm_getNumberAtoms(csm, SS_id(simp));
01961         if (natoms > 0) {
01962             return 1;
01963         }
01964
01965         /* We would like to resolve the mesh between the van der Waals surface the
01966         * max distance from this surface where there could be coefficient
01967         * changes */
01968         switch(srfm) {
01969             case VSM_MOL:
01970                 refAcc = Vacc_molAcc(acc, vx[0], srad);
01971                 for (i=1; i<(dim+1); i++) {
01972                     myAcc = Vacc_molAcc(acc, vx[i], srad);
01973                     if (myAcc != refAcc) {
01974                         return 1;
01975                     }
01976                 break;
01977             case VSM_MOLSMOOTH:
01978                 refAcc = Vacc_molAcc(acc, vx[0], srad);
01979                 for (i=1; i<(dim+1); i++) {
01980                     myAcc = Vacc_molAcc(acc, vx[i], srad);
01981                     if (myAcc != refAcc) {
01982                         return 1;
01983                     }
01984                 }
01985                 break;
01986             case VSM_SPLINE:
01987                 refAcc = Vacc_splineAcc(acc, vx[0], swin, 0.0);
01988                 for (i=1; i<(dim+1); i++) {
01989                     myAcc = Vacc_splineAcc(acc, vx[i], swin, 0.0);
01990                     if (myAcc != refAcc) {
01991                         return 1;
01992                     }
01993                 }
01994                 break;
01995             default:
01996                 VASSEERT(0);
01997                 break;
01998         }
01999     }
02000     return 0;
02001 }
02002
02003 VPUBLIC void Vfetk_PDE_oneChart(int dim, int dimII, int
02004     objType, int chart[],
02005     double vx[][][3], int dimV) {
02006 }
02007
02008 VPUBLIC double Vfetk_PDE_Ju(PDE *thee, int key) {
02009
02010     int i, ichop;
02011     double dielE, qmE, coef2, u2;
02012     double value = 0.;
02013     Vhal_PBEType type;
02014
02015     type = var.fetk->type;
02016
02017     /* interior form case */
02018     if (key == 0) {
02019         dielE = 0;

```

```

02020     for (i=0; i<3; i++) dielE += VSQR(var.dU[0][i]);
02021     dielE = dielE*var.diel;
02022
02023     switch (type) {
02024         case PBE_LPBE:
02025             if ((var.ionacc > VSMALL) && (var.zkappa2 > VSMALL)) {
02026                 qmE = var.ionacc*var.zkappa2*VSQR(var.U[0]);
02027             } else qmE = 0;
02028             break;
02029         case PBE_NPBE:
02030             if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
02031                 qmE = 0.;
02032                 for (i=0; i<var.nion; i++) {
02033                     coef2 = var.ionacc * var.zks2 * var.ionConc[
02034                         i] * var.ionQ[i];
02035                     u2 = -1.0 * (var.U[0]) * var.ionQ[i];
02036                     qmE += (coef2 * (Vcap_exp(u2, &ichop) - 1.0));
02037                 } else qmE = 0;
02038                 break;
02039         case PBE_LRPBE:
02040             if ((var.ionacc > VSMALL) && (var.zkappa2 > VSMALL)) {
02041                 qmE = var.ionacc*var.zkappa2*VSQR((var.U[0] + var.W
02042                     ));
02043             } else qmE = 0;
02044             break;
02045         case PBE_NRPBE:
02046             if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
02047                 qmE = 0.;
02048                 for (i=0; i<var.nion; i++) {
02049                     coef2 = var.ionacc * var.zks2 * var.ionConc[
02050                         i] * var.ionQ[i];
02051                     u2 = -1.0 * (var.U[0] + var.W) * var.ionQ[i];
02052                     qmE += (coef2 * (Vcap_exp(u2, &ichop) - 1.0));
02053                 } else qmE = 0;
02054                 break;
02055         case PBE_SMPBE: /* SMPBE Temp */
02056             if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
02057                 qmE = 0.;
02058                 for (i=0; i<var.nion; i++) {
02059                     coef2 = var.ionacc * var.zks2 * var.ionConc[
02060                         i] * var.ionQ[i];
02061                     u2 = -1.0 * (var.U[0]) * var.ionQ[i];
02062                     qmE += (coef2 * (Vcap_exp(u2, &ichop) - 1.0));
02063                 } else qmE = 0;
02064                 break;
02065             default:
02066                 Vnm_print(2, "Vfetk_PDE_Ju: Invalid PBE type (%d)!\n", type);
02067                 VASSERT(0);
02068                 break;
02069             }
02070             value = 0.5*(dielE + qmE) / Vpbe_getZmagic(var.fetk->
02071 pbe);
02072             /* boundary form case */
02073         } else if (key == 1) {
02074             value = 0.0;
02075
02076             /* how did we get here? */
02077         } else VASSERT(0);
02078
02079         return value;
02080     }
02081 }
02082
02083 VPUBLIC void Vfetk_externalUpdateFunction(SS **
02084     simps, int num) {
02085     Vcsm *csm = VNULL;
02086     int rc;
02087
02088     VASSERT(var.fetk != VNULL);
02089     csm = Vfetk_getVcsm(var.fetk);
02090     VASSERT(csm != VNULL);
02091
02092     rc = Vcsm_update(csm, simps, num);
02093
02094     if (!rc) {

```

```

02095     Vnm_print(2, "Error while updating charge-simplex map!\n");
02096     VASSERT(0);
02097 }
02098 }
02099
02100 VPRIIVATE void polyEval(int numP, double p[], double c[][VMAXP], double xv[]) {
02101     int i;
02102     double x, y, z;
02103
02104     x = xv[0];
02105     y = xv[1];
02106     z = xv[2];
02107     for (i=0; i<numP; i++) {
02108         p[i] = c[i][0]
02109             + c[i][1] * x
02110             + c[i][2] * y
02111             + c[i][3] * z
02112             + c[i][4] * x*x
02113             + c[i][5] * y*y
02114             + c[i][6] * z*z
02115             + c[i][7] * x*y
02116             + c[i][8] * x*z
02117             + c[i][9] * y*z
02118             + c[i][10] * x*x*x
02119             + c[i][11] * y*y*y
02120             + c[i][12] * z*z*z
02121             + c[i][13] * x*x*y
02122             + c[i][14] * x*x*z
02123             + c[i][15] * x*y*y
02124             + c[i][16] * y*y*z
02125             + c[i][17] * x*z*z
02126             + c[i][18] * y*z*z;
02127     }
02128 }
02129
02130 VPRIIVATE void setCoef(int numP, double c[][VMAXP], double cx[][VMAXP],
02131     double cy[][VMAXP], double cz[][VMAXP], int ic[][VMAXP], int icx[][VMAXP],
02132     int icy[], int icz[]) {
02133
02134     int i, j;
02135     for (i=0; i<numP; i++) {
02136         for (j=0; j<VMAXP; j++) {
02137             c[i][j] = 0.5 * (double)ic[i][j];
02138             cx[i][j] = 0.5 * (double)icx[i][j];
02139             cy[i][j] = 0.5 * (double)icy[i][j];
02140             cz[i][j] = 0.5 * (double)icz[i][j];
02141     }
02142 }
02143 }
02144
02145 VPUBLIC int Vfetk_PDE_simplexBasisInit(int key, int
dim, int comp, int *ndof,
02146     int dof[]) {
02147
02148     int qorder, bump, dimIS[VAPBS_NVS];
02149
02150     /* necessary quadrature order to return at the end */
02151     qorder = P_DEG;
02152
02153     /* deal with bump function requests */
02154     if ((key == 0) || (key == 1)) {
02155         bump = 0;
02156     } else if ((key == 2) || (key == 3)) {
02157         bump = 1;
02158     } else { VASSERT(0); }
02159
02160     /* for now use same element for all components, both trial and test */
02161     if (dim==2) {
02162         /* 2D simplex dimensions */
02163         dimIS[0] = 3; /* number of vertices */
02164         dimIS[1] = 3; /* number of edges */
02165         dimIS[2] = 0; /* number of faces (3D only) */
02166         dimIS[3] = 1; /* number of simplices (always=1) */
02167         if (bump==0) {
02168             if (P_DEG==1) {
02169                 init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
02170             } else if (P_DEG==2) {
02171                 init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
02172             } else if (P_DEG==3) {
02173                 init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
02174             } else Vnm_print(2, "..bad order..");
02175     }

```

```

02175      } else if (bump==1) {
02176          if (P_DEG==1) {
02177              init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
02178          } else Vnm_print(2, "..bad order..");
02179      } else Vnm_print(2, "..bad bump..");
02180  } else if (dim==3) {
02181      /* 3D simplex dimensions */
02182      dimIS[0] = 4; /* number of vertices */
02183      dimIS[1] = 6; /* number of edges */
02184      dimIS[2] = 4; /* number of faces (3D only) */
02185      dimIS[3] = 1; /* number of simplices (always=1) */
02186      if (bump==0) {
02187          if (P_DEG==1) {
02188              init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
02189          } else if (P_DEG==2) {
02190              init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
02191          } else if (P_DEG==3) {
02192              init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
02193          } else Vnm_print(2, "..bad order..");
02194      } else if (bump==1) {
02195          if (P_DEG==1) {
02196              init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
02197          } else Vnm_print(2, "..bad order..");
02198      } else Vnm_print(2, "..bad bump..");
02199  } else Vnm_print(2, "..bad dimension..");
02200
02201 /* save number of DF */
02202 numP = *ndof;
02203
02204 /* return the required quadrature order */
02205 return qorder;
02206 }
02207
02208 VPUBLIC void Vfetk_PDE_simplexBasisForm(int key, int
02209     dim, int comp, int pdkey,
02210     double xq[], double basis[]) {
02211
02212     if (pdkey == 0) {
02213         polyEval(numP, basis, c, xq);
02214     } else if (pdkey == 1) {
02215         polyEval(numP, basis, cx, xq);
02216     } else if (pdkey == 2) {
02217         polyEval(numP, basis, cy, xq);
02218     } else if (pdkey == 3) {
02219         polyEval(numP, basis, cz, xq);
02220     } else { VASSERT(0); }
02221
02222 VPRIVATE void init_2DP1(int dimIS[], int *ndof, int dof[], double c[][VMAXP],
02223     double cx[][VMAXP], double cy[][VMAXP], double cz[][VMAXP]) {
02224
02225     int i;
02226
02227     /* dof number and locations */
02228     dof[0] = 1;
02229     dof[1] = 0;
02230     dof[2] = 0;
02231     dof[3] = 0;
02232     *ndof = 0;
02233     for (i=0; i<VAPBS_NVS; i++) *ndof += dimIS[i] * dof[i];
02234     VASSERT( *ndof == dim_2DP1 );
02235     VASSERT( *ndof <= VMAXP );
02236
02237     /* coefficients of the polynomials */
02238     setCoef( *ndof, c, cx, cy, cz, lgr_2DP1, lgr_2DP1x, lgr_2DP1y, lgr_2DP1z );
02239 }
02240
02241 VPRIVATE void init_3DP1(int dimIS[], int *ndof, int dof[], double c[][VMAXP],
02242     double cx[][VMAXP], double cy[][VMAXP], double cz[][VMAXP]) {
02243
02244     int i;
02245
02246     /* dof number and locations */
02247     dof[0] = 1;
02248     dof[1] = 0;
02249     dof[2] = 0;
02250     dof[3] = 0;
02251     *ndof = 0;
02252     for (i=0; i<VAPBS_NVS; i++) *ndof += dimIS[i] * dof[i];
02253     VASSERT( *ndof == dim_3DP1 );
02254     VASSERT( *ndof <= VMAXP );

```

```

02255     /* coefficients of the polynomials */
02256     setCoef( *ndof, c, cx, cy, cz, lgr_3DP1x, lgr_3DP1y, lgr_3DP1z );
02258 }
02259
02260 VPUBLIC void Vfetk_dumpLocalVar() {
02261
02262     int i;
02263
02264     Vnm_print(1, "DEBUG: nvec = (%g, %g, %g)\n", var.nvec[0], var.nvec[
02265     1],
02266         var.nvec[2]);
02267     Vnm_print(1, "DEBUG: nverts = %d\n", var.nverts);
02268     for (i=0; i<var.nverts; i++) {
02269         Vnm_print(1, "DEBUG: verts[%d] ID = %d\n", i, VV_id(var.verts[i]));
02270
02271         Vnm_print(1, "DEBUG: vx[%d] = (%g, %g, %g)\n", i, var.vx[i][0],
02272             var.vx[i][1], var.vx[i][2]);
02273
02274         Vnm_print(1, "DEBUG: simp ID = %d\n", SS_id(var.simp));
02275         Vnm_print(1, "DEBUG: sType = %d\n", var.sType);
02276         Vnm_print(1, "DEBUG: fType = %d\n", var.fType);
02277         Vnm_print(1, "DEBUG: xq = (%g, %g, %g)\n", var.xq[0], var.xq[1], var.xq
02278 [2]);
02279         Vnm_print(1, "DEBUG: U[0] = %g\n", var.U[0]);
02280         Vnm_print(1, "DEBUG: dU[0] = (%g, %g, %g)\n", var.dU[0][0], var.dU[0][1]
02281     ],
02282         var.dU[0][2]);
02283         Vnm_print(1, "DEBUG: W = %g\n", var.W);
02284         Vnm_print(1, "DEBUG: d2W = %g\n", var.d2W);
02285         Vnm_print(1, "DEBUG: dW = (%g, %g, %g)\n", var.dW[0], var.dW[1], var.dW
02286 [2]);
02287         Vnm_print(1, "DEBUG: diel = %g\n", var.diel);
02288         Vnm_print(1, "DEBUG: ionacc = %g\n", var.ionacc);
02289         Vnm_print(1, "DEBUG: A = %g\n", var.A);
02290         Vnm_print(1, "DEBUG: F = %g\n", var.F);
02291         Vnm_print(1, "DEBUG: B = %g\n", var.B);
02292         Vnm_print(1, "DEBUG: DB = %g\n", var.DB);
02293         Vnm_print(1, "DEBUG: nion = %d\n", var.nion);
02294         for (i=0; i<var.nion; i++) {
02295             Vnm_print(1, "DEBUG: ionConc[%d] = %g\n", i, var.ionConc[i]);
02296             Vnm_print(1, "DEBUG: ionQ[%d] = %g\n", i, var.ionQ[i]);
02297             Vnm_print(1, "DEBUG: ionRadii[%d] = %g\n", i, var.ionRadii[i]);
02298
02299         Vnm_print(1, "DEBUG: zkappa2 = %g\n", var.zkappa2);
02300         Vnm_print(1, "DEBUG: zks2 = %g\n", var.zks2);
02301         Vnm_print(1, "DEBUG: Fu_v = %g\n", var.Fu_v);
02302         Vnm_print(1, "DEBUG: DFu_wv = %g\n", var.DFu_wv);
02303         Vnm_print(1, "DEBUG: delta = %g\n", var.delta);
02304         Vnm_print(1, "DEBUG: u_D = %g\n", var.u_D);
02305         Vnm_print(1, "DEBUG: u_T = %g\n", var.u_T);
02306     };
02307
02308     VPUBLIC int Vfetk_fillArray(Vfetk *thee, Bvec *vec,
02309         Vdata_Type type) {
02310
02311     int i, j, ichop;
02312     double coord[3], chi, q, conc, val;
02313     VV *vert;
02314     Bvec *u, *u_d;
02315     AM *am;
02316     Gem *gm;
02317     PBEparm *pbeparm;
02318     VAcc *acc;
02319     Vpbe *pbe;
02320
02321     gm = thee->gm;
02322     am = thee->am;
02323     pbe = thee->pbe;
02324     pbeparm = thee->pbeparm;
02325     acc = pbe->acc;
02326
02327     /* Make sure vec has enough rows to accomodate the vertex data */
02328     if (Bvec_numRB(vec, 0) != Gem_numVV(gm)) {
02329         Vnm_print(2, "Vfetk_fillArray: insufficient space in Bvec!\n");
02330         Vnm_print(2, "Vfetk_fillArray: Have %d, need %d!\n", Bvec_numRB(vec, 0
02331     ),
02332         Gem_numVV(gm));
02333     }
02334
02335     return 0;
02336 }

```

```

02329
02330     switch (type) {
02331
02332         case VDT_CHARGE:
02333             Vnm_print(2, "Vfetk_fillArray: can't write out charge
02334             distribution!\n");
02335             return 0;
02336             break;
02337
02338         case VDT_POT:
02339             u = am->u;
02340             u_d = am->ud;
02341             /* Copy in solution */
02342             Bvec_copy(vec, u);
02343             /* Add dirichlet condition */
02344             Bvec_axpy(vec, u_d, 1.0);
02345             break;
02346
02347         case VDT_SMOL:
02348             for (i=0; i<Gem_numVV(gm); i++) {
02349                 vert = Gem_VV(gm, i);
02350                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02351                 chi = Vacc_molAcc(acc, coord, pbe->solventRadius
02352             );
02353                 Bvec_set(vec, i, chi);
02354             break;
02355
02356         case VDT_SSPL:
02357             for (i=0; i<Gem_numVV(gm); i++) {
02358                 vert = Gem_VV(gm, i);
02359                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02360                 chi = Vacc_splineAcc(acc, coord, pbeparm->swin
02361             , 0.0);
02362                 Bvec_set(vec, i, chi);
02363             break;
02364
02365         case VDT_VDW:
02366             for (i=0; i<Gem_numVV(gm); i++) {
02367                 vert = Gem_VV(gm, i);
02368                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02369                 chi = Vacc_vdwAcc(acc, coord);
02370                 Bvec_set(vec, i, chi);
02371             }
02372             break;
02373
02374         case VDT_IVDW:
02375             for (i=0; i<Gem_numVV(gm); i++) {
02376                 vert = Gem_VV(gm, i);
02377                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02378                 chi = Vacc_ivdwAcc(acc, coord, pbe->maxIonRadius
02379             );
02380                 Bvec_set(vec, i, chi);
02381             }
02382             break;
02383
02384         case VDT_LAP:
02385             Vnm_print(2, "Vfetk_fillArray: can't write out Laplacian!\n");
02386             return 0;
02387             break;
02388
02389         case VDT_EDENS:
02390             Vnm_print(2, "Vfetk_fillArray: can't write out energy density!\n");
02391
02392             return 0;
02393             break;
02394
02395         case VDT_NDENS:
02396             u = am->u;
02397             u_d = am->ud;
02398             /* Copy in solution */
02399             Bvec_copy(vec, u);
02400             /* Add dirichlet condition */
02401             Bvec_axpy(vec, u_d, 1.0);
02402             /* Load up ions */
02403             ichop = 0;
02404             for (i=0; i<Gem_numVV(gm); i++) {
02405                 val = 0;
02406                 for (j=0; j<pbe->numIon; j++) {
02407                     q = pbe->ionQ[i];

```

```

02405             conc = pbe->ionConc[j];
02406             if (thee->type == PBE_NPBE || thee->type ==
02407     PBE_SMPBE /* SMPBE Added */ {
02408                 val += (conc*Vcap_exp(-q*Bvec_val(vec, i), &
02409                 ichop));
02410                 } else if (thee->type == PBE_LPBE) {
02411                     val += (conc * ( 1 - q*Bvec_val(vec, i)));
02412                 }
02413             Bvec_set(vec, i, val);
02414         }
02415         break;
02416     case VDT_QDENS:
02417         u = am->u;
02418         u_d = am->ud;
02419         /* Copy in solution */
02420         Bvec_copy(vec, u);
02421         /* Add dirichlet condition */
02422         Bvec_axpy(vec, u_d, 1.0);
02423         /* Load up ions */
02424         ichop = 0;
02425         for (i=0; i<Gem_numVV(gm); i++) {
02426             val = 0;
02427             for (j=0; j<pbe->numIon; j++) {
02428                 q = pbe->ionQ[j];
02429                 conc = pbe->ionConc[j];
02430                 if (thee->type == PBE_NPBE || thee->type ==
02431     PBE_SMPBE /* SMPBE Added */ {
02432                     val += (q*conc*Vcap_exp(-q*Bvec_val(vec, i), &
02433                     ichop));
02434                     } else if (thee->type == PBE_LPBE) {
02435                         val += (q*conc*(1 - q*Bvec_val(vec, i)));
02436                     }
02437                 Bvec_set(vec, i, val);
02438             }
02439             break;
02440         case VDT_DIELX:
02441             Vnm_print(2, "Vfetk_fillArray: can't write out x-shifted diel!\n");
02442             return 0;
02443             break;
02444         case VDT_DIELY:
02445             Vnm_print(2, "Vfetk_fillArray: can't write out y-shifted diel!\n");
02446             return 0;
02447             break;
02448         case VDT_DIELZ:
02449             Vnm_print(2, "Vfetk_fillArray: can't write out z-shifted diel!\n");
02450             return 0;
02451             break;
02452         case VDT_KAPPA:
02453             Vnm_print(2, "Vfetk_fillArray: can't write out kappa!\n");
02454             return 0;
02455             break;
02456         default:
02457             Vnm_print(2, "Vfetk_fillArray: invalid data type (%d)!\n", type);
02458             return 0;
02459             break;
02460         }
02461     }
02462     return 1;
02463 }
02464 }
02465
02466 VPUBLIC int Vfetk_write(Vfetk *thee, const char *iodev, const
02467     char *iofmt,
02468     const char *thost, const char *fname, Bvec *vec, Vdata_Format
02469     format) {
02470
02471     int i, j, ichop;
02472     Aprx *aprx;
02473     Gem *gm;
02474     Vio *sock;
02475
02476

```

```

02477     VASSERT(thee != VNULL);
02478     aprx = thee->aprx;
02479     gm = thee->gm;
02480
02481     sock = Vio_ctor(iodev, iofmt, thost, fname, "w");
02482     if (sock == VNULL) {
02483         Vnm_print(2, "Vfetk_write: Problem opening virtual socket %s\n",
02484             fname);
02485         return 0;
02486     }
02487     if (Vio_connect(sock, 0) < 0) {
02488         Vnm_print(2, "Vfetk_write: Problem connecting to virtual socket %s\n",
02489             fname);
02490         return 0;
02491     }
02492
02493     /* Make sure vec has enough rows to accomodate the vertex data */
02494     if (Bvec_numRB(vec, 0) != Gem_numVV(gm)) {
02495         Vnm_print(2, "Vfetk_fillArray: insufficient space in Bvec!\n");
02496         Vnm_print(2, "Vfetk_fillArray: Have %d, need %d!\n", Bvec_numRB(vec, 0
02497             ),
02498             Gem_numVV(gm));
02499     }
02500
02501     switch (format) {
02502
02503         case VDF_DX:
02504             Aprx_writeSOL(aprx, sock, vec, "DX");
02505             break;
02506         case VDF_AVIS:
02507             Aprx_writeSOL(aprx, sock, vec, "UCD");
02508             break;
02509         case VDF_UHBD:
02510             Vnm_print(2, "Vfetk_write: UHBD format not supported!\n");
02511             return 0;
02512         default:
02513             Vnm_print(2, "Vfetk_write: Invalid data format (%d)!\n", format);
02514             return 0;
02515     }
02516
02517     Vio_connectFree(sock);
02518     Vio_dtor(&sock);
02519
02520     return 1;
02521 }
02522 }
02523
02524 #endif

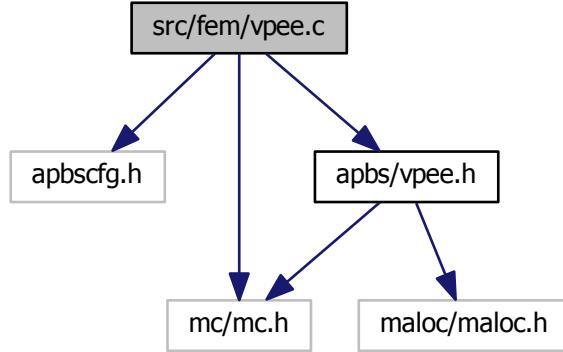
```

9.19 src/fem/vpee.c File Reference

Class Vpee methods.

```
#include "apbscfg.h"
#include "mc/mc.h"
#include "apbs/vpee.h"
```

Include dependency graph for vpee.c:



Functions

- VPRIVATE int **Vpee_userDefined** (*Vpee* **thee*, SS **sm*)
- VPRIVATE int **Vpee_ourSimp** (*Vpee* **thee*, SS **sm*, int *rcol*)
- VEXTERNC double **Aprx_estNonlinResid** (*Aprx* **thee*, SS **sm*, Bvec **u*, Bvec **ud*, Bvec **f*)
- VEXTERNC double **Aprx_estLocalProblem** (*Aprx* **thee*, SS **sm*, Bvec **u*, Bvec **ud*, Bvec **f*)
- VEXTERNC double **Aprx_estDualProblem** (*Aprx* **thee*, SS **sm*, Bvec **u*, Bvec **ud*, Bvec **f*)
- VPUBLIC *Vpee* * **Vpee_ctor** (*Gem* **gm*, int *localPartID*, int *killFlag*, double *killParam*)

Construct the Vpee object.

- VPUBLIC int **Vpee_ctor2** (*Vpee* **thee*, *Gem* **gm*, int *localPartID*, int *killFlag*, double *killParam*)

FORTRAN stub to construct the Vpee object.

- VPUBLIC void **Vpee_dtor** (*Vpee* ***thee*)

Object destructor.

- VPUBLIC void **Vpee_dtor2** (*Vpee* **thee*)

FORTRAN stub object destructor.

- VPUBLIC int **Vpee_markRefine** (*Vpee* **thee*, *AM* **am*, int *level*, int *akey*, int *rcol*, double *etol*, int *bkey*)

Mark simplices for refinement based on attenuated error estimates.

- VPUBLIC int **Vpee_numSS** (*Vpee* **thee*)

Returns the number of simplices in the local partition.

9.19.1 Detailed Description

Class Vpee methods.

Author

Nathan Baker

Version**Id:**[vpee.c](#) 1750 2012-07-18 18:34:27Z tuckerbeck**Attention**

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2011 Battelle Memorial Institute. Developed at the  
* Pacific Northwest National Laboratory, operated by Battelle Memorial  
* Institute, Pacific Northwest Division for the U.S. Department of Energy.  
*  
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.  
* Portions Copyright (c) 2002-2010, Nathan A. Baker.  
* Portions Copyright (c) 1999-2002, The Regents of the University of  
* California.  
* Portions Copyright (c) 1995, Michael Holst.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*  
* Redistributions of source code must retain the above copyright notice, this  
* list of conditions and the following disclaimer.  
*  
* Redistributions in binary form must reproduce the above copyright notice,  
* this list of conditions and the following disclaimer in the documentation  
* and/or other materials provided with the distribution.  
*  
* Neither the name of the developer nor the names of its contributors may be  
* used to endorse or promote products derived from this software without  
* specific prior written permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE  
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF  
* THE POSSIBILITY OF SUCH DAMAGE.  
*  
*
```

Definition in file [vpee.c](#).**9.20 vpee.c**

```
00001  
00058 #include "apbscfg.h"  
00059
```

```

00060 #if defined(HAVE_MC_H)
00061 #if defined(HAVE_MCX_H)
00062
00063 #include "mc/mc.h"
00064 #include "apbs/vpee.h"
00065
00066 VPRIIVATE int Vpee_userDefined(Vpee *thee,
00067                               SS *sm
00068                               );
00069 VPRIIVATE int Vpee_ourSimp(Vpee *thee,
00070                               SS *sm,
00071                               int rcol
00072                               );
00073 VEXTERNC double Aprx_estNonlinResid(Aprx *thee,
00074                               SS *sm,
00075                               Bvec *u,
00076                               Bvec *ud,
00077                               Bvec *f
00078                               );
00079 VEXTERNC double Aprx_estLocalProblem(Aprx *thee,
00080                               SS *sm,
00081                               Bvec *u,
00082                               Bvec *ud,
00083                               Bvec *f);
00084 VEXTERNC double Aprx_estDualProblem(Aprx *thee,
00085                               SS *sm,
00086                               Bvec *u,
00087                               Bvec *ud,
00088                               Bvec *f
00089                               );
00090
00091 /* /////////////////////////////////
00092 // Class Vpee: Non-inlineable methods
00093
00094 /* /////////////////////////////////
00095 // Routine: Vpee_ctor
00096 //
00097 //
00098 // Author: Nathan Baker
00100 VPUBLIC Vpee* Vpee_ctor(Gem *gm,
00101                               int localPartID,
00102                               int killFlag,
00103                               double killParam
00104                               ) {
00105
00106     Vpee *thee = VNULL;
00107
00108     /* Set up the structure */
00109     thee = Vmem_malloc(VNULL, 1, sizeof(Vpee) );
00110     VASSERT( thee != VNULL);
00111     VASSERT( Vpee_ctor2(thee, gm, localPartID, killFlag
00112 , killParam));
00113
00114     return thee;
00115 }
00116 /* /////////////////////////////////
00117 // Routine: Vpee_ctor2
00118 //
00119 // Author: Nathan Baker
00121 VPUBLIC int Vpee_ctor2(Vpee *thee,
00122                               Gem *gm,
00123                               int localPartID,
00124                               int killFlag,
00125                               double killParam
00126                               ) {
00127
00128     int invert,
00129     nLocalVerts;
00130     SS *simp;
00131     VV *vert;
00132     double radius,
00133     dx,
00134     dy,
00135     dz;
00136
00137     VASSERT(thee != VNULL);
00138
00139     /* Sanity check on input values */
00140     if (killFlag == 0) {
00141         Vnm_print(0, "Vpee_ctor2: No error attenuation outside partition.\n");
00142     } else if (killFlag == 1) {

```

```

00143     Vnm_print(0, "Vpee_ctor2: Error outside local partition ignored.\n");
00144 } else if (killFlag == 2) {
00145     Vnm_print(0, "Vpee_ctor2: Error ignored outside sphere with radius
00146 %4.3f times the radius of the circumscribing sphere\n", killParam);
00147     if (killParam < 1.0) {
00148         Vnm_print(2, "Vpee_ctor2: Warning! Parameter killParam = %4.3 < 1.0!
00149             \n",
00150             killParam);
00151         Vnm_print(2, "Vpee_ctor2: This may result in non-optimal marking and
00152 refinement!\n");
00153     } else if (killFlag == 3) {
00154         Vnm_print(0, "Vpee_ctor2: Error outside local partition and immediate
00155 neighbors ignored [NOT IMPLEMENTED].\n");
00156     } else {
00157         Vnm_print(2, "Vpee_ctor2: UNRECOGNIZED killFlag PARAMETER! BAILING!.n"
00158 );
00159         VASSERT(0);
00160     }
00161
00162     thee->gm = gm;
00163     thee->localPartID = localPartID;
00164     thee->killFlag = killFlag;
00165     thee->killParam = killParam;
00166     thee->mem = Vmem_ctor("APBS::VPEE");
00167
00168 /* Now, figure out the center of geometry for the local partition. The
00169 * general plan is to loop through the vertices, loop through the
00170 * vertices' simplex lists and find the vertices with simplices containing
00171 * chart values we're interested in. */
00172 thee->localPartCenter[0] = 0.0;
00173 thee->localPartCenter[1] = 0.0;
00174 thee->localPartCenter[2] = 0.0;
00175 nLocalVerts = 0;
00176 for (ivert=0; ivert<Gem_numVV(thee->gm); ivert++) {
00177     vert = Gem_VV(thee->gm, ivert);
00178     simp = VV_firstSS(vert);
00179     VASSERT(simp != VNNULL);
00180     while (simp != VNNULL) {
00181         if (SS_chart(simp) == thee->localPartID) {
00182             thee->localPartCenter[0] += VV_coord(vert, 0);
00183             thee->localPartCenter[1] += VV_coord(vert, 1);
00184             thee->localPartCenter[2] += VV_coord(vert, 2);
00185             nLocalVerts++;
00186             break;
00187         }
00188         simp = SS_link(simp, vert);
00189     }
00190     VASSERT(nLocalVerts > 0);
00191     thee->localPartCenter[0] =
00192         thee->localPartCenter[0]/((double)(nLocalVerts));
00193     thee->localPartCenter[1] =
00194         thee->localPartCenter[1]/((double)(nLocalVerts));
00195     thee->localPartCenter[2] =
00196         thee->localPartCenter[2]/((double)(nLocalVerts));
00197     Vnm_print(0, "Vpee_ctor2: Part %d centered at (%4.3f, %4.3f, %4.3f)\n",
00198             thee->localPartID, thee->localPartCenter[0], thee->localPartCenter[1],
00199             thee->localPartCenter[2]);
00200
00201 /* Now, figure out the radius of the sphere circumscribing the local
00202 * partition. We need to keep track of vertices so we don't double count
00203 * them. */
00204 thee->localPartRadius = 0.0;
00205 for (ivert=0; ivert<Gem_numVV(thee->gm); ivert++) {
00206     vert = Gem_VV(thee->gm, ivert);
00207     simp = VV_firstSS(vert);
00208     VASSERT(simp != VNNULL);
00209     while (simp != VNNULL) {
00210         if (SS_chart(simp) == thee->localPartID) {
00211             dx = thee->localPartCenter[0] - VV_coord(vert, 0);
00212             dy = thee->localPartCenter[1] - VV_coord(vert, 1);
00213             dz = thee->localPartCenter[2] - VV_coord(vert, 2);
00214             radius = dx*dx + dy*dy + dz*dz;
00215             if (radius > thee->localPartRadius) thee->localPartRadius =
00216                 radius;
00217             break;
00218         }
00219         simp = SS_link(simp, vert);
00220     }

```

```

00219      }
00220      theee->localPartRadius = VSQRT(theee->localPartRadius);
00221      Vnm_print(0, "Vpee_ctor2: Part %d has circumscribing sphere of radius %4.3f
00222 \n",
00223         theee->localPartID, theee->localPartRadius);
00224     return 1;
00225   }
00226
00227 /* /////////////////////////////////
00228 // Routine:  Vpee_dtor
00229 //
00230 // Author:  Nathan Baker
00231 VPUBLIC void Vpee_dtor(Vpee **thee) {
00232
00233   if ((*thee) != VNULL) {
00234     Vpee_dtor2(*thee);
00235     Vmem_free(VNULL, 1, sizeof(Vpee), (void **)thee);
00236     (*thee) = VNULL;
00237   }
00238 }
00239
00240 }
00241
00242 /* /////////////////////////////////
00243 // Routine:  Vpee_dtor2
00244 //
00245 // Author:  Nathan Baker
00246 VPUBLIC void Vpee_dtor2(Vpee *thee) {
00247   Vmem_dtor(&(thee->mem));
00248 }
00249
00250
00251 /* /////////////////////////////////
00252 // Routine:  Vpee_markRefine
00253 //
00254 // Author:  Nathan Baker (and Michael Holst: the author of AM_markRefine, on
00255 //           which this is based)
00256 VPUBLIC int Vpee_markRefine(Vpee *thee,
00257   AM *am,
00258   int level,
00259   int akey,
00260   int rcol,
00261   double etol,
00262   int bkey
00263   ) {
00264
00265   Aprx *aprx;
00266   int marked = 0,
00267   markMe,
00268   i,
00269   smid,
00270   count,
00271   currentQ;
00272   double minError = 0.0,
00273   maxError = 0.0,
00274   errEst = 0.0,
00275   mlevel,
00276   barrier;
00277   SS *sm;
00278
00279
00280   VASSERT(thee != VNULL);
00281
00282   /* Get the Aprx object from AM */
00283   aprx = am->aprx;
00284
00285   /* input check and some i/o */
00286   if ( ! ((-1 <= akey) && (akey <= 4)) ) {
00287     Vnm_print(0,"Vpee_markRefine: bad refine key; simplices marked = %d\n",
00288               marked);
00289     return marked;
00290   }
00291
00292   /* For uniform markings, we have no effect */
00293   if ((-1 <= akey) && (akey <= 0)) {
00294     marked = Gem_markRefine(thee->gm, akey, rcol);
00295     return marked;
00296   }
00297
00298   /* Informative I/O */
00299   if (akey == 2) {
00300     Vnm_print(0,"Vpee_estRefine: using Aprx_estNonlinResid()\n");
00301

```

```

00302     } else if (akey == 3) {
00303         Vnm_print(0, "Vpee_estRefine: using Aprx_estLocalProblem().\n");
00304     } else if (akey == 4) {
00305         Vnm_print(0, "Vpee_estRefine: using Aprx_estDualProblem().\n");
00306     } else {
00307         Vnm_print(0, "Vpee_estRefine: bad key given; simplices marked = %d\n",
00308                 marked);
00309         return marked;
00310     }
00311     if (thee->killFlag == 0) {
00312         Vnm_print(0, "Vpee_markRefine: No error attenuation -- simplices in all
00313 partitions will be marked.\n");
00314     } else if (thee->killFlag == 1) {
00315         Vnm_print(0, "Vpee_markRefine: Maximum error attenuation -- only
00316 simplices in local partition will be marked.\n");
00317     } else if (thee->killFlag == 2) {
00318         Vnm_print(0, "Vpee_markRefine: Spherical error attenuation --
00319 simplices within a sphere of %4.3f times the size of the partition will be marked\n",
00320                 thee->killParam);
00321     } else if (thee->killFlag == 2) {
00322         Vnm_print(0, "Vpee_markRefine: Neighbor-based error attenuation --
00323 simplices in the local and neighboring partitions will be marked [NOT IMPLEMENTED]!
00324 \n");
00325         VASSERT(0);
00326     } else {
00327         Vnm_print(2, "Vpee_markRefine: bogus killFlag given; simplices marked =
00328 %d\n",
00329                 marked);
00330         return marked;
00331     }
00332
00333 /* set the barrier type */
00334 mlevel = (etol*etol) / Gem_numSS(thee->gm);
00335 if (bkey == 0) {
00336     barrier = (etol*etol);
00337     Vnm_print(0, "Vpee_estRefine: forcing [err per S] < [TOL] = %g\n",
00338                 barrier);
00339 } else if (bkey == 1) {
00340     barrier = mlevel;
00341     Vnm_print(0, "Vpee_estRefine: forcing [err per S] < [(TOL^2/numS)^{1/2}]
00342 = %g\n",
00343                 VSQRT(barrier));
00344 } else {
00345     Vnm_print(0, "Vpee_estRefine: bad bkey given; simplices marked = %d\n",
00346                 marked);
00347     return marked;
00348 }
00349
00350 /* timer */
00351 Vnm_tstart(30, "error estimation");
00352
00353 /* count = num generations to produce from marked simplices (minimally) */
00354 count = 1; /* must be >= 1 */
00355
00356 /* check the refinement Q for emptiness */
00357 currentQ = 0;
00358 if (Gem_numSQ(thee->gm, currentQ) > 0) {
00359     Vnm_print(0, "Vpee_markRefine: non-empty refinement Q%d....clearing..",
00360                 currentQ);
00361     Gem_resetSQ(thee->gm, currentQ);
00362     Vnm_print(0, "..done.\n");
00363 }
00364 if (Gem_numSQ(thee->gm, !currentQ) > 0) {
00365     Vnm_print(0, "Vpee_markRefine: non-empty refinement Q%d....clearing..",
00366                 !currentQ);
00367     Gem_resetSQ(thee->gm, !currentQ);
00368     Vnm_print(0, "..done.\n");
00369 }
00370 VASSERT( Gem_numSQ(thee->gm, currentQ) == 0 );
00371 VASSERT( Gem_numSQ(thee->gm, !currentQ) == 0 );
00372
00373 /* clear everyone's refinement flags */
00374 Vnm_print(0, "Vpee_markRefine: clearing all simplex refinement flags..");
00375 for (i=0; i<Gem_numSS(thee->gm); i++) {
00376     if ( (i>0) && (i % VVRTKEY) == 0 ) Vnm_print(0, "[MS:%d]", i);
00377     sm = Gem_SS(thee->gm, i);
00378     SS_setRefineKey(sm, currentQ, 0);
00379     SS_setRefineKey(sm, !currentQ, 0);
00380     SS_setRefinementCount(sm, 0);
00381 }
00382 Vnm_print(0, "..done.\n");

```

```

00376
00377     /* NON-ERROR-BASED METHODS */
00378     /* Simplex flag clearing */
00379     if (akey == -1) return marked;
00380     /* Uniform & user-defined refinement*/
00381     if ((akey == 0) || (akey == 1)) {
00382         smid = 0;
00383         while ( smid < Gem_numSS(thee->gm) ) {
00384             /* Get the simplex and find out if it's markable */
00385             sm = Gem_SS(thee->gm,smid);
00386             markMe = Vpee_ourSimp(thee, sm, rcol);
00387             if (markMe) {
00388                 if (akey == 0) {
00389                     marked++;
00390                     Gem_appendSQ(thee->gm,currentQ, sm);
00391                     SS_setRefineKey(sm,currentQ,1);
00392                     SS_setRefinementCount (sm, count);
00393                 } else if (Vpee_userDefined(thee, sm)) {
00394                     marked++;
00395                     Gem_appendSQ(thee->gm,currentQ, sm);
00396                     SS_setRefineKey(sm,currentQ,1);
00397                     SS_setRefinementCount (sm, count);
00398                 }
00399             }
00400             smid++;
00401         }
00402     }
00403
00404     /* ERROR-BASED METHODS */
00405     /* gerror = global error accumulation */
00406     aprx->gerror = 0.;
00407
00408     /* traverse the simplices and process the error estimates */
00409     Vnm_print(0,"Vpee_markRefine: estimating error..");
00410     smid = 0;
00411     while ( smid < Gem_numSS(thee->gm) ) {
00412
00413         /* Get the simplex and find out if it's markable */
00414         sm = Gem_SS(thee->gm,smid);
00415         markMe = Vpee_ourSimp(thee, sm, rcol);
00416
00417         if ( (smid>0) && (smid % VPRTKEY) == 0 ) Vnm_print(0,[MS:%d],smid);
00418
00419         /* Produce an error estimate for this element if it is in the set */
00420         if (markMe) {
00421             if (akey == 2) {
00422                 errEst = Aprx_estNonlinResid(aprx, sm, am->u,am->ud,am->f);
00423             } else if (akey == 3) {
00424                 errEst = Aprx_estLocalProblem(aprx, sm, am->u,am->ud,am->f);
00425             } else if (akey == 4) {
00426                 errEst = Aprx_estDualProblem(aprx, sm, am->u,am->ud,am->f);
00427             }
00428             VASSERT( errEst >= 0. );
00429
00430             /* if error estimate above tol, mark element for refinement */
00431             if (errEst > barrier) {
00432                 marked++;
00433                 Gem_appendSQ(thee->gm,currentQ, sm); /*add to refinement Q*/
00434                 SS_setRefineKey(sm,currentQ,1); /* note now on refine Q */
00435                 SS_setRefinementCount (sm, count); /* refine X many times? */
00436             }
00437
00438             /* keep track of min/max errors over the mesh */
00439             minError = VMIN2( VSQRT(VABS(errEst)), minError );
00440             maxError = VMAX2( VSQRT(VABS(errEst)), maxError );
00441
00442             /* store the estimate */
00443             Bvec_set( aprx->wew, smid, errEst );
00444
00445             /* accumulate into global error (errEst is SQUARED already) */
00446             aprx->gerror += errEst;
00447
00448             /* otherwise store a zero for the estimate */
00449             } else {
00450                 Bvec_set( aprx->wew, smid, 0. );
00451             }
00452
00453             smid++;
00454         }
00455
00456     /* do some i/o */

```

```

00457     Vnm_print(0, "..done. [marked=<%d/%d>]\n", marked, Gem_numSS(thee->gm));
00458     Vnm_print(0, "Vpee_estRefine: TOL=<%g> Global_Error=<%g>\n",
00459                 etol, aprx->error);
00460     Vnm_print(0, "Vpee_estRefine: (TOL^2/numS)^(1/2)=<%g> Max_Ele_Error=<%g>\n"
00461                 VSQRT(mlevel),maxError);
00462     Vnm_tstop(30, "error estimation");
00463
00464     /* check for making the error tolerance */
00465     if ((bkey == 1) && (aprx->error <= etol)) {
00466         Vnm_print(0,
00467             "Vpee_estRefine: *****\n");
00468         Vnm_print(0,
00469             "Vpee_estRefine: Global Error criterion met; setting marked=0.\n");
00470         Vnm_print(0,
00471             "Vpee_estRefine: *****\n");
00472         marked = 0;
00473     }
00474
00475
00476     /* return */
00477     return marked;
00478
00479 }
00480
00481 /* /////////////////////////////////
00482 // Routine: Vpee_numSS
00483 //
00484 // Author: Nathan Baker
00485 VPUBLIC int Vpee_numSS(Vpee *thee) {
00486     int num = 0;
00487     int isimp;
00488
00489     for (isimp=0; isimp<Gem_numSS(thee->gm); isimp++) {
00490         if (SS_chart(Gem_SS(thee->gm, isimp)) == thee->localPartID) num++;
00491     }
00492
00493     return num;
00494 }
00495
00496
00497 /* /////////////////////////////////
00498 // Routine: Vpee_userDefined
00499 //
00500 // Purpose: Reduce code bloat by wrapping up the common steps for getting the
00501 // user-defined error estimate
00502 //
00503 // Author: Nathan Baker
00504 VPRIIVATE int Vpee_userDefined(Vpee *thee,
00505                         SS *sm
00506                         )
00507 {
00508
00509     int invert,
00510         icoord,
00511         chart[4],
00512         fType[4],
00513         vType[4];
00514     double vx[4][3];
00515
00516     for (invert=0; invert<Gem_dimVV(thee->gm); invert++) {
00517         fType[invert] = SS_faceType(sm,invert);
00518         vType[invert] = VV_type(SS_vertex(sm,invert) );
00519         chart[invert] = VV_chart(SS_vertex(sm,invert) );
00520         for (icoord=0; icoord<Gem_dimII(thee->gm); icoord++) {
00521             vx[invert][icoord] = VV_coord(SS_vertex(sm,invert), icoord );
00522         }
00523     }
00524     return thee->gm->pde->markSimplex(Gem_dim(thee->gm), Gem_dimII(thee->gm),
00525                                         SS_type(sm), fType, vType, chart, vx, sm);
00526 }
00527
00528 /* /////////////////////////////////
00529 // Routine: Vpee_ourSimp
00530 //
00531 // Purpose: Reduce code bloat by wrapping up the common steps for determining
00532 // whether the given simplex can be marked (i.e., belongs to our
00533 // partition or overlap region)
00534 //
00535 // Returns: 1 if could be marked, 0 otherwise
00536 //
00537 // Author: Nathan Baker
00538 VPRIIVATE int Vpee_ourSimp(Vpee *thee,

```

```

00540                     SS *sm,
00541                     int rcol
00542                 ) {
00543
00544     int invert;
00545     double dist,
00546         dx,
00547         dy,
00548         dz;
00549
00550     if (thee->killFlag == 0) return 1;
00551     else if (thee->killFlag == 1) {
00552         if ((SS_chart(sm) == rcol) || (rcol < 0)) return 1;
00553     } else if (thee->killFlag == 2) {
00554         if (rcol < 0) return 1;
00555     } else {
00556         /* We can only do distance-based searches on the local partition */
00557         VASSERT(rcol == thee->localPartID);
00558         /* Find the closest distance between this simplex and the
00559          * center of the local partition and check it against
00560          * (thee->localPartRadius*thee->killParam) */
00561         dist = 0;
00562         for (invert=0; invert<SS_dimVV(sm); invert++) {
00563             dx = VV_coord(SS_vertex(sm, invert), 0) -
00564                 thee->localPartCenter[0];
00565             dy = VV_coord(SS_vertex(sm, invert), 1) -
00566                 thee->localPartCenter[1];
00567             dz = VV_coord(SS_vertex(sm, invert), 2) -
00568                 thee->localPartCenter[2];
00569             dist = VSQRT((dx*dx + dy*dy + dz*dz));
00570         }
00571         if (dist < thee->localPartRadius*thee->killParam)
00572             return 1;
00573         }
00574     } else if (thee->killFlag == 3) VASSERT(0);
00575     else VASSERT(0);
00576
00577     return 0;
00578 }
00579
00580 #endif
00581 #endif

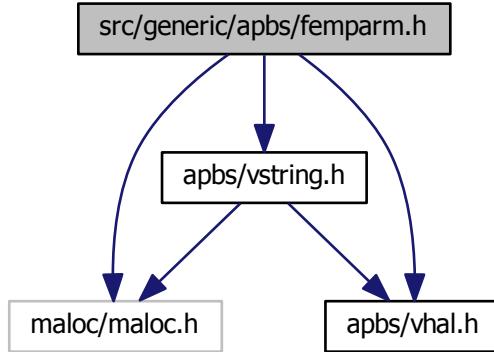
```

9.21 src/generic/apbs/femparm.h File Reference

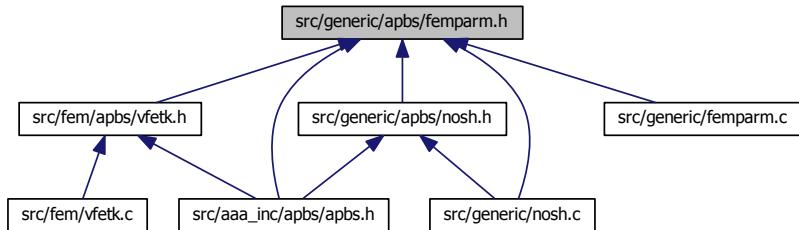
Contains declarations for class APOLparm.

```
#include "malloc/malloc.h"
#include "apbs/vhal.h"
#include "apbs/vstring.h"
```

Include dependency graph for femparm.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [sFMParm](#)
Parameter structure for FEM-specific variables from input files.

TypeDefs

- typedef enum [eFMParm_EtolType](#) `FMParm_EtolType`
Declare FMParm_EtolType type.
- typedef enum [eFMParm_EstType](#) `FMParm_EstType`
Declare FMParm_EstType type.
- typedef enum [eFMParm_CalcType](#) `FMParm_CalcType`
Declare FMParm_CalcType type.
- typedef struct [sFMParm](#) `FMParm`

Declaration of the FEMparm class as the FEMparm structure.

Enumerations

- enum `eFEMparm_EtolType` { `FET_SIMP` = 0, `FET_GLOB` = 1, `FET_FRAC` = 2 }

Adaptive refinement error estimate tolerance key.

- enum `eFEMparm_EstType` {
`FRT_UNIF` = 0, `FRT_GEOM` = 1, `FRT_RESI` = 2, `FRT_DUAL` = 3,
`FRT_LOCA` = 4 }

Adaptive refinement error estimator method.

- enum `eFEMparm_CalcType` { `FCT_MANUAL`, `FCT_NONE` }

Calculation type.

Functions

- VEXTERNC `FEMparm *` `FEMparm_ctor` (`FEMparm_CalcType` type)
Construct FEMparm.
- VEXTERNC int `FEMparm_ctor2` (`FEMparm *thee`, `FEMparm_CalcType` type)
FORTRAN stub to construct FEMparm.
- VEXTERNC void `FEMparm_dtor` (`FEMparm **thee`)
Object destructor.
- VEXTERNC void `FEMparm_dtor2` (`FEMparm *thee`)
FORTRAN stub for object destructor.
- VEXTERNC int `FEMparm_check` (`FEMparm *thee`)
Consistency check for parameter values stored in object.
- VEXTERNC void `FEMparm_copy` (`FEMparm *thee`, `FEMparm *source`)
Copy target object into thee.
- VEXTERNC Vrc_Codes `FEMparm_parseToken` (`FEMparm *thee`, char `tok[VMAX_BUFSIZE]`, Vio `*sock`)
Parse an MG keyword from an input file.

9.21.1 Detailed Description

Contains declarations for class APOLparm. Contains declarations for class FEMparm.

Version

Id:

`apolparm.h` 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2012 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.  
*  
*
```

Version

Id:

[femparm.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 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](#).

9.22 femparm.h

```

00001
00063 #ifndef _FEMPARM_H_
00064 #define _FEMPARM_H_
00065
00066 /* Generic header files */
00067 #include "maloc/maloc.h"
00068 #include "apbs/vhal.h"
00069 #include "apbs/vstring.h"
00070
00076 enum eFEMParm_EtoIType {
00077     FET_SIMP=0,
00078     FET_GLOB=1,
00079     FET_FRAC=2
00080 };

```

```

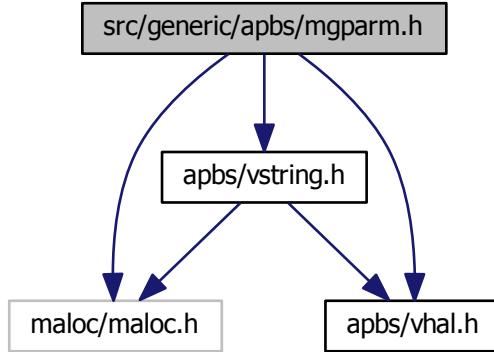
00081
00087     typedef enum eFEMparm_EtolType FEMparm_EtolType
00088     ;
00089
00095     enum eFEMparm_EstType {
00096         FRT_UNIF=0,
00097         FRT_GEOM=1,
00098         FRT_RESI=2,
00099         FRT_DUAL=3,
00101         FRT_LOCA=4
00102     };
00103
00108     typedef enum eFEMparm_EstType FEMparm_EstType;
00109
00114     enum eFEMparm_CalcType {
00115         FCT_MANUAL,
00116         FCT_NONE
00117     };
00118
00123     typedef enum eFEMparm_CalcType FEMparm_CalcType
00124     ;
00130     struct sFEMparm {
00131
00132         int parsed;
00135         FEMparm_CalcType type;
00136         int settype;
00137         double glen[3];
00138         int setglen;
00139         double etol;
00140         int setetol;
00141         FEMparm_EtolType ekey;
00144         int setekey;
00145         FEMparm_EstType akeyPRE;
00148         int setakeyPRE;
00149         FEMparm_EstType akeySOLVE;
00151         int setakeySOLVE;
00152         int targetNum;
00156         int settargetNum;
00157         double targetRes;
00161         int settargetRes;
00162         int maxsolve;
00163         int setmaxsolve;
00164         int maxvert;
00166         int setmaxvert;
00167         int pkey;
00170         int useMesh;
00171         int meshID;
00173     };
00174
00179     typedef struct sFEMparm FEMparm;
00180
00181 /* //////////////////////////////// */
00182 // Class NOsh: Non-inlineable methods (nosh.c)
00184
00191 VEXTERNC FEMparm* FEMparm_ctor(FEMparm_CalcType type);
00192
00200 VEXTERNC int FEMparm_ctor2(FEMparm *thee, FEMparm_CalcType type);
00201
00207 VEXTERNC void FEMparm_dtor(FEMparm **thee);
00208
00214 VEXTERNC void FEMparm_dtor2(FEMparm *thee);
00215
00223 VEXTERNC int FEMparm_check(FEMparm *thee);
00224
00231 VEXTERNC void FEMparm_copy(FEMparm *thee, FEMparm *source);
00232
00243 VEXTERNC Vrc_Codes FEMparm_parseToken(FEMparm *thee, char tok[VMAX_BUFSIZE],
00244     Vio *sock);
00245
00246 #endif
00247

```

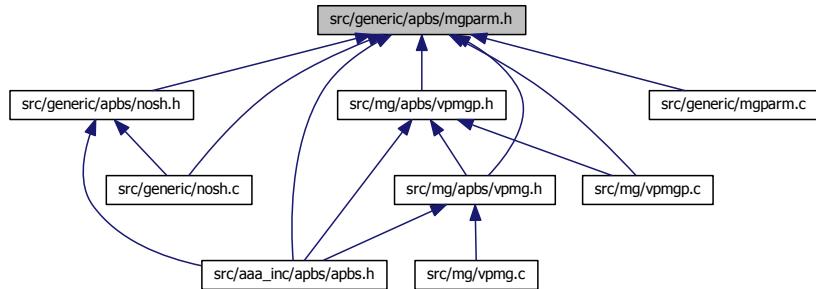
9.23 src/generic/apbs/mgparm.h File Reference

Contains declarations for class MGparm.

```
#include "maloc/malloc.h"
#include "apbs/vhal.h"
#include "apbs/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)**

FORTRAN stub to construct MGparm object.
- **VEXTERNC void MGparm_dtor (MGparm **thee)**

Object destructor.
- **VEXTERNC void MGparm_dtor2 (MGparm *thee)**

FORTRAN stub for object destructor.

- VEXTERNC Vrc_Codes [MGparm_check](#) ([MGparm](#) *thee)
Consistency check for parameter values stored in object.
- VEXTERNC void [MGparm_copy](#) ([MGparm](#) *thee, [MGparm](#) *parm)
Copy MGparm object into thee.
- VEXTERNC Vrc_Codes [MGparm_parseToken](#) ([MGparm](#) *thee, char tok[VMAX_BUFSIZE], [Vio](#) *sock)
Parse an MG keyword from an input file.

9.23.1 Detailed Description

Contains declarations for class MGparm.

Version

Id:

[mgparm.h](#) 1763 2012-07-18 22:04:34Z tuckerbeck

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](#).

9.24 mgparm.h

```

00001
00064 #ifndef _MGPARM_H_
00065 #define _MGPARM_H_
00066
00067 #include "malloc/malloc.h"
00068 #include "apbs/vhal.h"
00069 #include "apbs/vstring.h"
00070
00075 enum eMgpParm_CalcType {
00076     MCT_MANUAL=0,
00077     MCT_AUTO=1,
00078     MCT_PARALLEL=2,
00079     MCT_DUMMY=3,
00080     MCT_NONE=4
00081 };
00082
00087 typedef enum eMgpParm_CalcType MGparm_CalcType;
00088
00093 enum eMgpParm_CentMeth {
00094     MCM_POINT=0,
00095     MCM_MOLECULE=1,
00096     MCM_FOCUS=2
00097 };
00098
00103 typedef enum eMgpParm_CentMeth MGparm_CentMeth;
00112 struct sMgpParm {
00113
00114     MGparm_CalcType type;
00115     int parsed;
00117     /* *** GENERIC PARAMETERS *** */
00118     int dime[3];
00119     int setdime;
00120     Vchrg_Meth chgm;
00121     int setchgm;
00122     Vchrg_Src chgs;
00125     /* *** TYPE 0 PARAMETERS (SEQUENTIAL MANUAL) *** */
00126     int nlev;
00128     int setnlev;
00129     double etol;
00130     int setetol;
00131     double grid[3];
00132     int setgrid;
00133     double glen[3];
00134     int setglen;
00135     MGparm_CentMeth cmeth;
00136     double center[3];
00144     int centmol;
00147     int setgcent;
00149     /* ***** TYPE 1 & 2 PARAMETERS (SEQUENTIAL & PARALLEL AUTO-FOCUS) *** */
00150     double cglen[3];
00151     int setcglen;
00152     double fglen[3];
00153     int setfglen;
00154     MGparm_CentMeth ccmeth;
00155     double ccenter[3];
00156     int ccentmol;
00159     int setcgcent;
00160     MGparm_CentMeth fcmeth;

```

```

00161     double fcenter[3];
00162     int fcenmol;
00163     int setfgcent;
00168     /* ***** TYPE 2 PARAMETERS (PARALLEL AUTO-FOCUS) ***** */
00169     double partDisjCenter[3];
00171     double partDisjLength[3];
00173     int partDisjOwnSide[6];
00176     int pdime[3];
00177     int setpdime;
00178     int proc_rank;
00179     int setrank;
00180     int proc_size;
00181     int setsize;
00182     double ofrac;
00183     int setofrac;
00184     int async;
00185     int setasync;
00187     int nonlintype;
00188     int setnonlintype;
00190     int method;
00191     int setmethod;
00193     int useAqua;
00194     int setUseAqua;
00195 };
00196
00201     typedef struct sMparm MGparm;
00202
00209 VEXTERNC int MGparm_getNx(MGparm *thee);
00210
00217 VEXTERNC int MGparm_getNy(MGparm *thee);
00218
00225 VEXTERNC int MGparm_getNz(MGparm *thee);
00226
00233 VEXTERNC double MGparm_getHx(MGparm *thee);
00234
00241 VEXTERNC double MGparm_getHy(MGparm *thee);
00242
00249 VEXTERNC double MGparm_getHz(MGparm *thee);
00250
00257 VEXTERNC void MGparm_setCenterX(MGparm *thee, double x);
00258
00265 VEXTERNC void MGparm_setCenterY(MGparm *thee, double y);
00266
00273 VEXTERNC void MGparm_setCenterZ(MGparm *thee, double z);
00274
00281 VEXTERNC double MGparm_getCenterX(MGparm *thee);
00282
00289 VEXTERNC double MGparm_getCenterY(MGparm *thee);
00290
00297 VEXTERNC double MGparm_getCenterZ(MGparm *thee);
00298
00305 VEXTERNC MGparm* MGparm_ctor(MGparm_CalcType
    type);
00306
00314 VEXTERNC Vrc_Codes      MGparm_ctor2(MGparm *thee,
    MGparm_CalcType type);
00315
00321 VEXTERNC void      MGparm_dtor(MGparm **thee);
00322
00328 VEXTERNC void      MGparm_dtor2(MGparm *thee);
00329
00336 VEXTERNC Vrc_Codes      MGparm_check(MGparm *thee);
00337
00344 VEXTERNC void      MGparm_copy(MGparm *thee, MGparm *parm
    );
00345
00355 VEXTERNC Vrc_Codes      MGparm_parseToken(MGparm *thee,
    char tok[VMAX_BUFSIZE],
    Vio *sock);
00356
00357
00358 #endif
00359

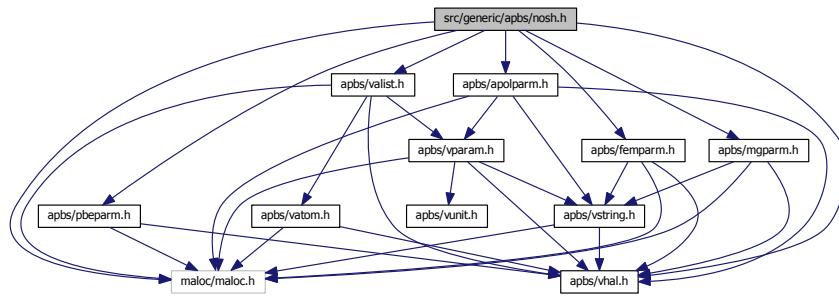
```

9.25 src/generic/apbs/nosh.h File Reference

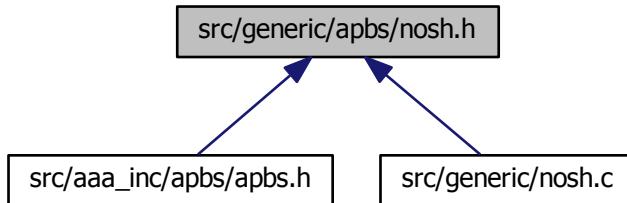
Contains declarations for class NOsh.

```
#include "maloc/malloc.h"
#include "apbs/vhal.h"
#include "apbs/pbeparm.h"
#include "apbs/mgparm.h"
#include "apbs/femparm.h"
#include "apbs/apolparm.h"
#include "apbs/valist.h"

Include dependency graph for nosh.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `sNOsh_calc`
Calculation class for use when parsing fixed format input files.
- struct `sNOsh`
Class for parsing fixed format input files.

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 }`
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)`

- **VEXTERNC char * NOsh_getPotpath (NOsh *thee, int imap)**

Returns path to specified kappa map.
- **VEXTERNC char * NOsh_getChargepath (NOsh *thee, int imap)**

Returns path to specified potential map.
- **VEXTERNC NOsh_calc * NOsh_getCalc (NOsh *thee, int icalc)**

Returns path to specified charge distribution map.
- **VEXTERNC int NOsh_getDielfmt (NOsh *thee, int imap)**

Returns specified calculation object.
- **VEXTERNC int NOsh_getKappafmt (NOsh *thee, int imap)**

Returns format of specified dielectric map.
- **VEXTERNC int NOsh_getPotfmt (NOsh *thee, int imap)**

Returns format of specified kappa map.
- **VEXTERNC int NOsh_getChargefmt (NOsh *thee, int imap)**

Returns format of specified potential map.
- **VEXTERNC NOsh_PrintType NOsh_printWhat (NOsh *thee, int iprint)**

Return an integer ID of the observable to print .
- **VEXTERNC char * NOsh_elecname (NOsh *thee, int ielec)**

Return an integer mapping of an ELEC statement to a calculation ID .
- **VEXTERNC int NOsh_elec2calc (NOsh *thee, int icalc)**

Return the name of an elec statement.
- **VEXTERNC int NOsh_apol2calc (NOsh *thee, int icalc)**

Return the name of an apol statement.
- **VEXTERNC int NOsh_printNarg (NOsh *thee, int iprint)**

Return number of arguments to PRINT statement .
- **VEXTERNC int NOsh_printOp (NOsh *thee, int iprint, int iarg)**

Return integer ID for specified operation .
- **VEXTERNC int NOsh_printCalc (NOsh *thee, int iprint, int iarg)**

Return calculation ID for specified PRINT statement .
- **VEXTERNC NOsh * NOsh_ctor (int rank, int size)**

Construct NOsh.
- **VEXTERNC NOsh_calc * NOsh_calc_ctor (NOsh_CalcType calcType)**

Construct NOsh_calc.
- **VEXTERNC int NOsh_calc_copy (NOsh_calc *thee, NOsh_calc *source)**

Copy NOsh_calc object into thee.
- **VEXTERNC void NOsh_calc_dtor (NOsh_calc **thee)**

Object destructor.
- **VEXTERNC int NOsh_ctor2 (NOsh *thee, int rank, int size)**

FORTRAN stub to construct NOsh.
- **VEXTERNC void NOsh_dtor (NOsh **thee)**

Object destructor.
- **VEXTERNC void NOsh_dtor2 (NOsh *thee)**

FORTRAN stub for object destructor.
- **VEXTERNC int NOsh_parselInput (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.

9.25.1 Detailed Description

Contains declarations for class NOsh.

Version

Id:

[nosh.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 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](#).

9.26 nosh.h

```
00001
00062 #ifndef _NOSH_H_
00063 #define _NOSH_H_
00064
00065 /* Generic headers */
00066 #include "maloc/maloc.h"
00067 #include "apbs/vhal.h"
00068
00069 /* Headers specific to this file */
00070 #include "apbs/pbeparm.h"
00071 #include "apbs/mgparm.h"
00072 #include "apbs/femparm.h"
00073 #include "apbs/apolparm.h"
00074 #include "apbs/valist.h"
00075
00076 #define NOSH_MAXMOL 20
00077
00082 #define NOSH_MAXCALC 20
00083
00086 #define NOSH_MAXPRINT 20
00087
00090 #define NOSH_MAXPOP 20
00091
00096 enum eNosh_MolFormat {
00097     NMF_PQR=0,
00098     NMF_PDB=1,
00099     NMF_XML=2
00100 };
00101
00106 typedef enum eNosh_MolFormat NOsh_MolFormat;
00107
00112 enum eNosh_CalcType {
00113     NCT_MG=0,
00114     NCT_FEM=1,
00115     NCT_APOL=2
00116 };
00117
00122 typedef enum eNosh_CalcType NOsh_CalcType;
00123
00128 enum eNosh_ParmFormat {
00129     NPF_FLAT=0,
00130     NPF_XML=1
00131 };
00132
00137 typedef enum eNosh_ParmFormat NOsh_ParmFormat;
00138
00143 enum eNosh_PrintType {
00144     NPT_ENERGY=0,
00145     NPT_FORCE=1,
00146     NPT_ELECENERGY,
00147     NPT_ELECFORCE,
00148     NPT_APOLENERGY,
00149     NPT_APOLFORCE
00150 };
00151
00156 typedef enum eNosh_PrintType NOsh_PrintType;
00157
00163 struct sNOsh_calc {
00164     MGparm *mgparm;
00165     FEMparm *femparm;
00166     PBEparm *pbeparm;
```

```

00167 APOLparm *apolparm;
00168 NOsh_CalcType calctype;
00169 };
00170
00175 typedef struct sNOsh_calc NOsh_calc;
00176
00182 struct sNOsh {
00183
00184 NOsh_calc *calc[NOSH_MAXCALC];
00185     int ncalc;
00186 NOsh_calc *elec[NOSH_MAXCALC];
00187     int nelec;
00188 NOsh_calc *apol[NOSH_MAXCALC];
00189     int napol;
00190     int ispara;
00191     int proc_rank;
00192     int proc_size;
00193     int bogus;
00194     int elec2calc[NOSH_MAXCALC];
00195     int apol2calc[NOSH_MAXCALC];
00196     int nmol;
00197     char molpath[NOSH_MAXMOL][VMAX_ARGLEN];
00198     NOsh_MolFormat molfmt[NOSH_MAXMOL];
00199 Valist *alist[NOSH_MAXMOL];
00200     int gotparm;
00201     char parmpath[VMAX_ARGLEN];
00202     NOsh_ParmFormat parmfmt;
00203     int ndiel;
00204     char dielxpath[NOSH_MAXMOL][VMAX_ARGLEN];
00205     char dielypath[NOSH_MAXMOL][VMAX_ARGLEN];
00206     char dielzpath[NOSH_MAXMOL][VMAX_ARGLEN];
00207     Vdata_Format dielfmt[NOSH_MAXMOL];
00208     int nkappa;
00209     char kappapath[NOSH_MAXMOL][VMAX_ARGLEN];
00210     Vdata_Format kappafmt[NOSH_MAXMOL];
00211     int npot;
00212     char potpath[NOSH_MAXMOL][VMAX_ARGLEN];
00213     Vdata_Format potfmt[NOSH_MAXMOL];
00214     int ncharge;
00215     char chargepath[NOSH_MAXMOL][VMAX_ARGLEN];
00216     Vdata_Format chargefmt[NOSH_MAXMOD];
00217     int nmesh;
00218     char meshpath[NOSH_MAXMOL][VMAX_ARGLEN];
00219     Vdata_Format meshfmt[NOSH_MAXMOL];
00220     int nprint;
00221     NOsh_PrintType printwhat[NOSH_MAXPRINT];
00222     int printnarg[NOSH_MAXPRINT];
00223     int printcalc[NOSH_MAXPRINT][NOSH_MAXPOP];
00224     int printtop[NOSH_MAXPRINT][NOSH_MAXPOP];
00225     int parsed;
00226     char elecname[NOSH_MAXCALC][VMAX_ARGLEN];
00227     char apolname[NOSH_MAXCALC][VMAX_ARGLEN];
00228 };
00229
00230 typedef struct sNOsh NOsh;
00231
00232 /* //////////////////////////////// */
00233 // Class NOsh: Inlineable methods (mcsh.c)
00234 #if !defined(VINLINE_NOsh)
00235 VEXTERNC char* NOsh_getMolpath(NOsh *thee, int imol);
00236
00237 VEXTERNC char* NOsh_getDielxpath(NOsh *thee, int imap);
00238
00239 VEXTERNC char* NOsh_getDielypath(NOsh *thee, int imap);
00240
00241 VEXTERNC char* NOsh_getDielzpath(NOsh *thee, int imap);
00242
00243 VEXTERNC char* NOsh_getKappapath(NOsh *thee, int imap);
00244
00245 VEXTERNC char* NOsh_getPotpath(NOsh *thee, int imap);
00246
00247 VEXTERNC char* NOsh_getChargepath(NOsh *thee, int imap);
00248
00249 VEXTERNC NOsh_calc* NOsh_getCalc(NOsh *thee, int icalc);
00250
00251 VEXTERNC int NOsh_getDielfmt(NOsh *thee, int imap);
00252
00253 VEXTERNC int NOsh_getKappafmt(NOsh *thee, int imap);
00254
00255
00256
00257
00258
00259
00260
00261
00262
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
0194
```

```

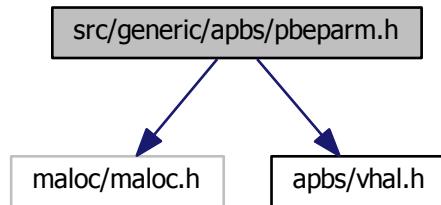
00367 VEXTERNC int NOsh_getPotfmt(NOsh *thee, int imap);
00368
00376 VEXTERNC int NOsh_getChargefmt(NOsh *thee, int imap);
00377
00378 #else
00379
00380 # define NOsh_getMolpath(thee, imol) ((thee)->molpath[(imol)])
00381 # define NOsh_getDielXpath(thee, imol) ((thee)->dielXpath[(imol)])
00382 # define NOsh_getDielYpath(thee, imol) ((thee)->dielYpath[(imol)])
00383 # define NOsh_getDielZpath(thee, imol) ((thee)->dielZpath[(imol]))
00384 # define NOsh_getKappapath(thee, imol) ((thee)->kappapath[(imol)])
00385 # define NOsh_getPotpath(thee, imol) ((thee)->potpath[(imol)])
00386 # define NOsh_getChargepath(thee, imol) ((thee)->chargepath[(imol)])
00387 # define NOsh_getCalc(thee, icalc) ((thee)->calc[(icalc)])
00388 # define NOsh_getDielfmt(thee, imap) ((thee)->dielfmt[(imap)])
00389 # define NOsh_getKappafmt(thee, imap) ((thee)->kappafmt[(imap)])
00390 # define NOsh_getPotfmt(thee, imap) ((thee)->potfmt[(imap]))
00391 # define NOsh_getChargefmt(thee, imap) ((thee)->chargefmt[(imap]))
00392
00393 #endif
00394
00395
00396 /* //////////////////////////////// Class NOsh: Non-inlineable methods (mcsh.c)
00397 // Class NOsh: Non-inlineable methods (mcsh.c)
00398
00399 VEXTERNC NOsh_PrintType NOsh_printWhat(NOsh *thee, int iprint);
00400
00418 VEXTERNC char* NOsh_elecname(NOsh *thee, int ielec);
00419
00427 VEXTERNC int NOsh_eleccalc(NOsh *thee, int icalc);
00428
00436 VEXTERNC int NOsh_apol2calc(NOsh *thee, int icalc);
00437
00445 VEXTERNC int NOsh_printNarg(NOsh *thee, int iprint);
00446
00455 VEXTERNC int NOsh_printOp(NOsh *thee, int iprint, int iarg);
00456
00467 VEXTERNC int NOsh_printCalc(NOsh *thee, int iprint, int iarg);
00468
00478 VEXTERNC NOsh* NOsh_ctor(int rank, int size);
00479
00486 VEXTERNC NOsh_calc* NOsh_calc_ctor(
00487     NOsh_CalcType calcType
00488 );
00489
00496 VEXTERNC int NOsh_calc_copy(
00497     NOsh_calc *thee,
00498     NOsh_calc *source
00499 );
00500
00506 VEXTERNC void NOsh_calc_dtor(NOsh_calc **thee);
00507
00518 VEXTERNC int NOsh_ctor2(NOsh *thee, int rank, int size);
00519
00525 VEXTERNC void NOsh_dtor(NOsh **thee);
00526
00532 VEXTERNC void NOsh_dtor2(NOsh *thee);
00533
00542 VEXTERNC int NOsh_parseInput(NOsh *thee, Vio *sock);
00543
00553 VEXTERNC int NOsh_parseInputFile(NOsh *thee, char *filename);
00554
00564 VEXTERNC int NOsh_setupElecCalc(
00565     NOsh *thee,
00566     Valist *alist[NOSH_MAXMOL]
00567 );
00568
00578 VEXTERNC int NOsh_setupApolCalc(
00579     NOsh *thee,
00580     Valist *alist[NOSH_MAXMOL]
00581 );
00582
00583 #endif
00584

```

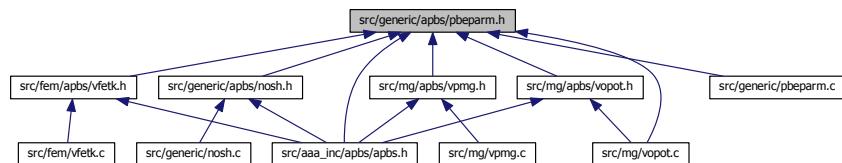
9.27 src/generic/apbs/pbeparm.h File Reference

Contains declarations for class PBParm.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
Include dependency graph for pbeparm.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [sPBParm](#)

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 [ePBParm_calcEnergy](#) [PBParm_calcEnergy](#)
Define ePBParm_calcEnergy enumeration as PBParm_calcEnergy.
- typedef enum [ePBParm_calcForce](#) [PBParm_calcForce](#)

Define ePBParm_calcForce enumeration as PBParm_calcForce.

- **typedef struct sPBParm PBParm**

Declaration of the PBParm class as the PBParm structure.

Enumerations

- **enum ePBParm_calcEnergy { PCE_NO = 0, PCE_TOTAL = 1, PCE_COMPS = 2 }**
Define energy calculation enumeration.
- **enum ePBParm_calcForce { PCF_NO = 0, PCF_TOTAL = 1, PCF_COMPS = 2 }**
Define force calculation enumeration.

Functions

- **VEXTERNC double PBParm_getIonCharge (PBParm *thee, int iion)**
Get charge (e) of specified ion species.
- **VEXTERNC double PBParm_getIonConc (PBParm *thee, int iion)**
Get concentration (M) of specified ion species.
- **VEXTERNC double PBParm_getIonRadius (PBParm *thee, int iion)**
Get radius (A) of specified ion species.
- **VEXTERNC PBParm * PBParm_ctor ()**
Construct PBParm object.
- **VEXTERNC int PBParm_ctor2 (PBParm *thee)**
FORTRAN stub to construct PBParm object.
- **VEXTERNC void PBParm_dtor (PBParm **thee)**
Object destructor.
- **VEXTERNC void PBParm_dtor2 (PBParm *thee)**
FORTRAN stub for object destructor.
- **VEXTERNC int PBParm_check (PBParm *thee)**
Consistency check for parameter values stored in object.
- **VEXTERNC void PBParm_copy (PBParm *thee, PBParm *parm)**
Copy PBParm object into thee.
- **VEXTERNC int PBParm_parseToken (PBParm *thee, char tok[VMAX_BUFSIZE], Vio *sock)**
Parse a keyword from an input file.

9.27.1 Detailed Description

Contains declarations for class PBParm.

Version

Id:

[pbeparm.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 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](#).

9.28 pbeparm.h

```

00001
00062 #ifndef _PBEPARM_H_
00063 #define _PBEPARM_H_
00064
00065 /* Generic headers */
00066 #include "malloc/malloc.h"
00067
00068 /* Headers specific to this file */

```

```
00069 #include "apbs/vhal.h"
00070
00074 #define PBEPARM_MAXWRITE 20
00075
00080 enum ePBEParm_calcEnergy {
00081     PCE_NO=0,
00082     PCE_TOTAL=1,
00083     PCE_COMPS=2
00084 };
00085
00090 typedef enum ePBEParm_calcEnergy PBEParm_calcEnergy
00091 ;
00096 enum ePBEParm_calcForce {
00097     PCF_NO=0,
00098     PCF_TOTAL=1,
00099     PCF_COMPS=2
00100 };
00101
00106 typedef enum ePBEParm_calcForce PBEParm_calcForce
00107 ;
00116 struct sPBEParm {
00117
00118     int molid;
00119     int setmolid;
00120     int useDielMap;
00122     int dielMapID;
00123     int useKappaMap;
00125     int kappaMapID;
00126     int usePotMap;
00128     int potMapID;
00130     int useChargeMap;
00132     int chargeMapID;
00133     Vhal_PBEType pbetype;
00134     int setpbetype;
00135     Vbcfl bcfl;
00136     int setbcfl;
00137     int nion;
00138     int setnion;
00139     double iong[MAXION];
00140     double ionc[MAXION];
00141     double ionr[MAXION];
00142     int setion[MAXION];
00143     double pdie;
00144     int setpdie;
00145     double sdens;
00146     int setsdens;
00147     double sdie;
00148     int setsdie;
00149     Vsurf_Meth srfm;
00150     int setsrfm;
00151     double srad;
00152     int setsrad;
00153     double swin;
00154     int setswin;
00155     double temp;
00156     int settemp;
00158     double smsize;
00159     int setsmsize;
00161     double smvolume;
00162     int setsmvolume;
00164     PBEParm_calcEnergy calcenergy;
00165     int setcalcenergy;
00166     PBEParm_calcForce calcforce;
00167     int setcalcforce;
00169 /*-----*/
00170 /* Added by Michael Grabe */
00171 /*-----*/
00172
00173     double zmem;
00174     int setzmem;
00175     double lmem;
00176     int setlmem;
00177     double mdie;
00178     int setmdie;
00179     double memv;
00180     int setmemv;
00182 /*-----*/
00183
00184     int numwrite;
```

```

00185     char writestem[PBEPARAM_MAXWRITE] [VMAX_ARGLEN];
00187     Vdata_Type writetype[PBEPARAM_MAXWRITE];
00188     Vdata_Format writefmt[PBEPARAM_MAXWRITE]
00189     ;
00190     int writemat;
00193     int setwritemat;
00194     char writematemstem[VMAX_ARGLEN];
00195     int writematflag;
00200     int parsed;
00202   };
00203
00208   typedef struct sPBEparm PBEparm;
00209
00210 /* //////////////////////////////// */
00211 // Class NOsh: Non-inlineable methods (mcsh.c)
00213
00219 VEXTERNC double PBEparm_getIonCharge(
00220     PBEparm *thee,
00221     int iion
00222 );
00223
00229 VEXTERNC double PBEparm_getIonConc(
00230     PBEparm *thee,
00231     int iion
00232 );
00233
00239 VEXTERNC double PBEparm_getIonRadius(
00240     PBEparm *thee,
00241     int iion
00242 );
00243
00244
00250 VEXTERNC PBEparm* PBEparm_ctor();
00251
00257 VEXTERNC int PBEparm_ctor2(
00258     PBEparm *thee
00259 );
00260
00265 VEXTERNC void PBEparm_dtor(
00266     PBEparm **thee
00267 );
00268
00273 VEXTERNC void PBEparm_dtor2(
00274     PBEparm *thee
00275 );
00276
00282 VEXTERNC int PBEparm_check(
00283     PBEparm *thee
00284 );
00285
00290 VEXTERNC void PBEparm_copy(
00291     PBEparm *thee,
00292     PBEparm *parm
00293 );
00294
00301 VEXTERNC int PBEparm_parseToken(
00302     PBEparm *thee,
00303     char tok[VMAX_BUFSIZE],
00304     Vio *sock
00305 );
00306
00307
00308 #endif
00309

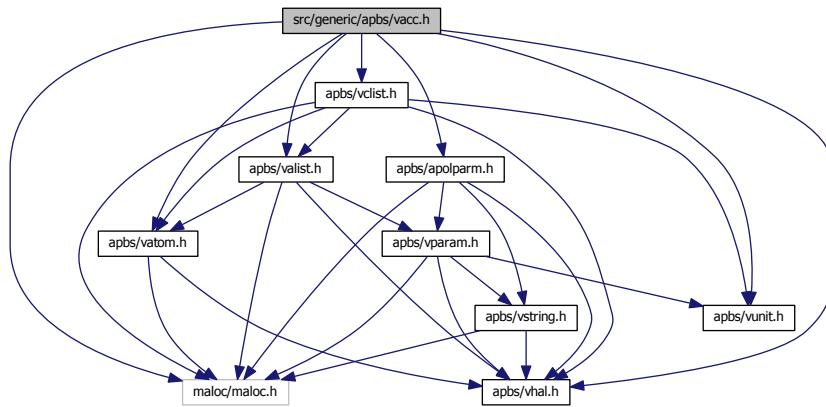
```

9.29 src/generic/apbs/vacc.h File Reference

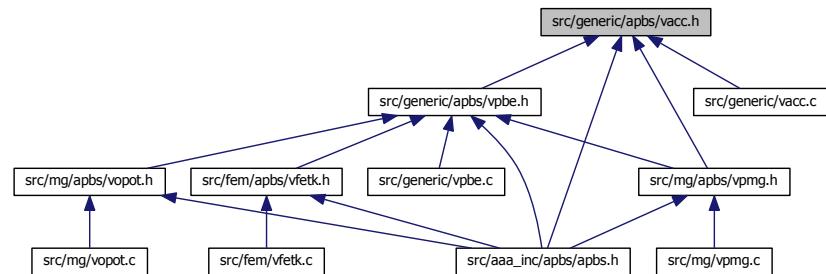
Contains declarations for class Vacc.

```
#include "maloc/malloc.h"
#include "apbs/vhal.h"
#include "apbs/valist.h"
#include "apbs/vclist.h"
#include "apbs/vatom.h"
#include "apbs/vunit.h"
#include "apbs/apolparm.h"
```

Include dependency graph for vacc.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct **sVaccSurf**
Surface object list of per-atom surface points.
- struct **sVacc**
Oracle for solvent- and ion-accessibility around a biomolecule.

Typedefs

- **typedef struct sVaccSurf VaccSurf**
Declaration of the VaccSurf class as the VaccSurf structure.
- **typedef struct sVacc Vacc**
Declaration of the Vacc class as the Vacc structure.

Functions

- **VEXTERNC unsigned long int Vacc_memChk (Vacc *thee)**
Get number of bytes in this object and its members.
- **VEXTERNC VaccSurf * VaccSurf_ctor (Vmem *mem, double probe_radius, int nsphere)**
Allocate and construct the surface object; do not assign surface points to positions.
- **VEXTERNC int VaccSurf_ctor2 (VaccSurf *thee, Vmem *mem, double probe_radius, int nsphere)**
Construct the surface object using previously allocated memory; do not assign surface points to positions.
- **VEXTERNC void VaccSurf_dtor (VaccSurf **thee)**
Destroy the surface object and free its memory.
- **VEXTERNC void VaccSurf_dtor2 (VaccSurf *thee)**
Destroy the surface object.
- **VEXTERNC VaccSurf * VaccSurf_refSphere (Vmem *mem, int npts)**
Set up an array of points for a reference sphere of unit radius.
- **VEXTERNC VaccSurf * Vacc_atomSurf (Vacc *thee, Vatom *atom, VaccSurf *ref, double probe_radius)**
Set up an array of points corresponding to the SAS due to a particular atom.
- **VEXTERNC Vacc * Vacc_ctor (Valist *alist, Vclist *clist, double surf_density)**
Construct the accessibility object.
- **VEXTERNC int Vacc_ctor2 (Vacc *thee, Valist *alist, Vclist *clist, double surf_density)**
FORTRAN stub to construct the accessibility object.
- **VEXTERNC void Vacc_dtor (Vacc **thee)**
Destroy object.
- **VEXTERNC void Vacc_dtor2 (Vacc *thee)**
FORTRAN stub to destroy object.
- **VEXTERNC double Vacc_vdwAcc (Vacc *thee, double center[VAPBS_DIM])**
Report van der Waals accessibility.
- **VEXTERNC double Vacc_ivdwAcc (Vacc *thee, double center[VAPBS_DIM], double radius)**
Report inflated van der Waals accessibility.
- **VEXTERNC double Vacc_molAcc (Vacc *thee, double center[VAPBS_DIM], double radius)**
Report molecular accessibility.
- **VEXTERNC double Vacc_fastMolAcc (Vacc *thee, double center[VAPBS_DIM], double radius)**
Report molecular accessibility quickly.
- **VEXTERNC double Vacc_splineAcc (Vacc *thee, double center[VAPBS_DIM], double win, double infrad)**
Report spline-based accessibility.
- **VEXTERNC void Vacc_splineAccGrad (Vacc *thee, double center[VAPBS_DIM], double win, double infrad, double *grad)**
Report gradient of spline-based accessibility.
- **VEXTERNC double Vacc_splineAccAtom (Vacc *thee, double center[VAPBS_DIM], double win, double infrad, Vatom *atom)**
Report spline-based accessibility for a given atom.

- VEXTERNC void `Vacc_splineAccGradAtomUnnorm` (`Vacc *thee`, double `center[VAPBS_DIM]`, double `win`, double `infrad`, `Vatom *atom`, double `*force`)
Report gradient of spline-based accessibility with respect to a particular atom (see `Vpmg_splineAccAtom`)
- VEXTERNC void `Vacc_splineAccGradAtomNorm` (`Vacc *thee`, double `center[VAPBS_DIM]`, double `win`, double `infrad`, `Vatom *atom`, double `*force`)
Report gradient of spline-based accessibility with respect to a particular atom normalized by the accessibility value due to that atom at that point (see `Vpmg_splineAccAtom`)
- VEXTERNC void `Vacc_splineAccGradAtomNorm4` (`Vacc *thee`, double `center[VAPBS_DIM]`, double `win`, double `infrad`, `Vatom *atom`, double `*force`)
Report gradient of spline-based accessibility with respect to a particular atom normalized by a 4th order accessibility value due to that atom at that point (see `Vpmg_splineAccAtom`)
- VEXTERNC void `Vacc_splineAccGradAtomNorm3` (`Vacc *thee`, double `center[VAPBS_DIM]`, double `win`, double `infrad`, `Vatom *atom`, double `*force`)
Report gradient of spline-based accessibility with respect to a particular atom normalized by a 3rd order accessibility value due to that atom at that point (see `Vpmg_splineAccAtom`)
- VEXTERNC double `Vacc_SASA` (`Vacc *thee`, double `radius`)
Build the solvent accessible surface (SAS) and calculate the solvent accessible surface area.
- VEXTERNC double `Vacc_totalSASA` (`Vacc *thee`, double `radius`)
Return the total solvent accessible surface area (SASA)
- VEXTERNC double `Vacc_atomSASA` (`Vacc *thee`, double `radius`, `Vatom *atom`)
Return the atomic solvent accessible surface area (SASA)
- VEXTERNC `VaccSurf * Vacc_atomSASPoints` (`Vacc *thee`, double `radius`, `Vatom *atom`)
Get the set of points for this atom's solvent-accessible surface.
- VEXTERNC void `Vacc_atomdSAV` (`Vacc *thee`, double `radius`, `Vatom *atom`, double `*dSA`)
Get the derivative of solvent accessible volume.
- VEXTERNC void `Vacc_atomdSASA` (`Vacc *thee`, double `dpos`, double `radius`, `Vatom *atom`, double `*dSA`)
Get the derivative of solvent accessible area.
- VEXTERNC void `Vacc_totalAtomdSASA` (`Vacc *thee`, double `dpos`, double `radius`, `Vatom *atom`, double `*dSA`)
Testing purposes only.
- VEXTERNC void `Vacc_totalAtomdSAV` (`Vacc *thee`, double `dpos`, double `radius`, `Vatom *atom`, double `*dSA`, `Vclist *clist`)
Total solvent accessible volume.
- VEXTERNC double `Vacc_totalSAV` (`Vacc *thee`, `Vclist *clist`, `APOLparm *apolparm`, double `radius`)
Return the total solvent accessible volume (SAV)
- VEXTERNC int `Vacc_wcaEnergy` (`Vacc *thee`, `APOLparm *apolparm`, `Valist *alist`, `Vclist *clist`)
Return the WCA integral energy.
- VEXTERNC int `Vacc_wcaForceAtom` (`Vacc *thee`, `APOLparm *apolparm`, `Vclist *clist`, `Vatom *atom`, double `*force`)
Return the WCA integral force.
- VEXTERNC int `Vacc_wcaEnergyAtom` (`Vacc *thee`, `APOLparm *apolparm`, `Valist *alist`, `Vclist *clist`, int `iatom`, double `*value`)
Calculate the WCA energy for an atom.

9.29.1 Detailed Description

Contains declarations for class `Vacc`.

Version**Id:**[vacc.h](#) 1750 2012-07-18 18:34:27Z tuckerbeck**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](#).

9.30 vacc.h

```

00001
00062 #ifndef _VACC_H_
00063 #define _VACC_H_
00064
00065 /* Generic headers */
00066 #include "maloc/maloc.h"
00067 #include "apbs/vhal.h"
00068
00069 /* Headers specific to this file */
00070 #include "apbs/valist.h"
00071 #include "apbs/vclist.h"
00072 #include "apbs/vatom.h"
00073 #include "apbs/vunit.h"
00074 #include "apbs/apolparm.h"
00075
00081 struct sVaccSurf {
00082     Vmem *mem;
00083     double *xpts;
00084     double *ypts;
00085     double *zpts;
00086     char *bpts;
00088     double area;
00089     int npts;
00090     double probe_radius;
00092 };
00093
00098 typedef struct sVaccSurf VaccSurf;
00099
00105 struct sVacc {
00106
00107     Vmem *mem;
00108     Valist *alist;
00109     Vclist *clist;
00110     int *atomFlags;
00113     VaccSurf *refSphere;
00114     VaccSurf **surf;
00117     Vset acc;
00119     double surf_density;
00122 };
00123
00128 typedef struct sVacc Vacc;
00129
00130 #if !defined(VINLINE_VACC)
00131
00137     VEXTERNC unsigned long int Vacc_memChk(
00138         Vacc *thee
00139     );
00140
00141 #else /* if defined(VINLINE_VACC) */
00142
00143 #    define Vacc_memChk(thee) (Vmem_bytes((thee)->mem))
00144
00145 #endif /* if !defined(VINLINE_VACC) */
00146
00154 VEXTERNC VaccSurf* VaccSurf_ctor(
00155     Vmem *mem,
00156     double probe_radius,
00157     int nsphere
00158 );
00159
00167 VEXTERNC int VaccSurf_ctor2(
00168     VaccSurf *thee,
00169     Vmem *mem,
00170     double probe_radius,
00171     int nsphere
00172 );
00173
00179 VEXTERNC void VaccSurf_dtor(
00180     VaccSurf **thee
00181 );
00182
00188 VEXTERNC void VaccSurf_dtor2(
00189     VaccSurf *thee
00190 );
00191
00206 VEXTERNC VaccSurf* VaccSurf_refSphere(
00207     Vmem *mem,
00208     int npts

```

```
00209      );
00210
00218 VEXTERNC VaccSurf* Vacc_atomSurf(
00219     Vacc *thee,
00220     Vatom *atom,
00221     VaccSurf *ref,
00223     double probe_radius
00224 );
00225
00230 VEXTERNC Vacc* Vacc_ctor(
00231     Valist *alist,
00232     Vclist *clist,
00234     double surf_density
00236 );
00237
00242 VEXTERNC int Vacc_ctor2(
00243     Vacc *thee,
00244     Valist *alist,
00245     Vclist *clist,
00247     double surf_density
00249 );
00250
00255 VEXTERNC void Vacc_dtors(
00256     Vacc **thee
00257 );
00258
00263 VEXTERNC void Vacc_dtors2(
00264     Vacc *thee
00265 );
00266
00277 VEXTERNC double Vacc_vdwAcc(
00278     Vacc *thee,
00279     double center[VAPBS_DIM]
00280 );
00281
00293 VEXTERNC double Vacc_ivdwAcc(
00294     Vacc *thee,
00295     double center[VAPBS_DIM],
00296     double radius
00297 );
00298
00313 VEXTERNC double Vacc_molAcc(
00314     Vacc *thee,
00315     double center[VAPBS_DIM],
00316     double radius
00317 );
00318
00337 VEXTERNC double Vacc_fastMolAcc(
00338     Vacc *thee,
00339     double center[VAPBS_DIM],
00340     double radius
00341 );
00342
00354 VEXTERNC double Vacc_splineAcc(
00355     Vacc *thee,
00356     double center[VAPBS_DIM],
00357     double win,
00358     double inftrad
00359 );
00360
00366 VEXTERNC void Vacc_splineAccGrad(
00367     Vacc *thee,
00368     double center[VAPBS_DIM],
00369     double win,
00370     double inftrad,
00371     double *grad
00372 );
00373
00385 VEXTERNC double Vacc_splineAccAtom(
00386     Vacc *thee,
00387     double center[VAPBS_DIM],
00388     double win,
00389     double inftrad,
00390     Vatom *atom
00391 );
00392
00403 VEXTERNC void Vacc_splineAccGradAtomUnnorm(
00404     Vacc *thee,
00405     double center[VAPBS_DIM],
00406     double win,
00407     double inftrad,
```

```
00408     Vatom *atom,
00409     double *force
00410   );
00411
00412 VEXTERNC void Vacc_splineAccGradAtomNorm(
00413   Vacc *thee,
00414   double center[VAPBS_DIM],
00415   double win,
00416   double infrad,
00417   Vatom *atom,
00418   double *force
00419 );
00420
00421 VEXTERNC void Vacc_splineAccGradAtomNorm4(
00422   Vacc *thee,
00423   double center[VAPBS_DIM],
00424   double win,
00425   double infrad,
00426   Vacc *atom,
00427   double *force
00428 );
00429
00430 VEXTERNC void Vacc_splineAccGradAtomNorm3(
00431   Vacc *thee,
00432   double center[VAPBS_DIM],
00433   double win,
00434   double infrad,
00435   Vatom *atom,
00436   double *force
00437 );
00438
00439 VEXTERNC void Vacc_SASA(
00440   Vacc *thee,
00441   double radius
00442 );
00443
00444 VEXTERNC double Vacc_totalSASA(
00445   Vacc *thee,
00446   double radius
00447 );
00448
00449 VEXTERNC double Vacc_atomSASA(
00450   Vacc *thee,
00451   double radius,
00452   Vatom *atom
00453 );
00454
00455 VEXTERNC VaccSurf* Vacc_atomSASPoints(
00456   Vacc *thee,
00457   double radius,
00458   Vatom *atom
00459 );
00460
00461 VEXTERNC void Vacc_atomdSAV(
00462   Vacc *thee,
00463   double radius,
00464   Vatom *atom,
00465   double *dSA
00466 );
00467
00468 VEXTERNC void Vacc_atomdSASA(
00469   Vacc *thee,
00470   double dpos,
00471   double radius,
00472   Vatom *atom,
00473   double *dSA
00474 );
00475
00476 VEXTERNC void Vacc_totalAtomdSASA(
00477   Vacc *thee,
00478   double dpos,
00479   double radius,
00480   Vatom *atom,
00481   double *dSA
00482 );
00483
00484 VEXTERNC void Vacc_totalAtomdSAV(
00485   Vacc *thee,
00486   double dpos,
00487   double radius,
```

```

00563     Vatom *atom,
00564     double *dsA,
00565     Vclist *clist
00566 );
00567
00575 VEXTERNC double Vacc_totalsAV(
00576     Vacc *thee,
00577     Vclist *clist,
00578     APOLparm *apolparm,
00579     double radius
00580 );
00581
00582
00589 VEXTERNC int Vacc_wcaEnergy(
00590     Vacc *thee,
00591     APOLparm *apolparm,
00592     Valist *alist,
00593     Vclist *clist
00594 );
00601 VEXTERNC int Vacc_wcaForceAtom(Vacc *thee,
00602     APOLparm *apolparm,
00603     Vclist *clist,
00604     Vatom *atom,
00605     double *force
00606 );
00607
00613 VEXTERNC int Vacc_wcaEnergyAtom(
00614     Vacc *thee,
00615     APOLparm *apolparm,
00616     Valist *alist,
00617     Vclist *clist,
00618     int iatom,
00619     double *value
00620 );
00621
00622 #endif /* ifndef _VACC_H_ */

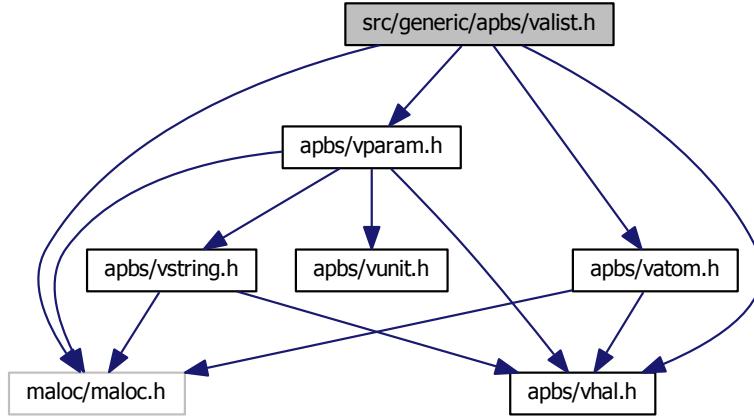
```

9.31 src/generic/apbs/valist.h File Reference

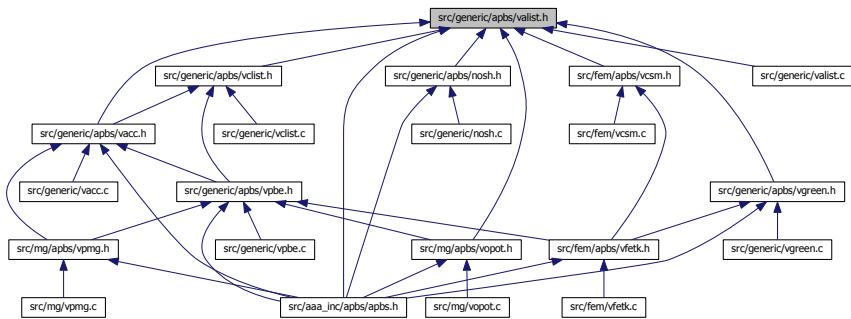
Contains declarations for class Valist.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/vatom.h"
#include "apbs/vparam.h"
```

Include dependency graph for valist.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `sValist`

Container class for list of atom objects.

TypeDefs

- typedef struct `sValist` `Valist`

Declaration of the Valist class as the Valist structure.

Functions

- VEXTERNC `Vatom * Valist_getAtomList (Valist *thee)`

- VEXTERNC double `Valist_getCenterX (Valist *thee)`
Get x-coordinate of molecule center.
- VEXTERNC double `Valist_getCenterY (Valist *thee)`
Get y-coordinate of molecule center.
- VEXTERNC double `Valist_getCenterZ (Valist *thee)`
Get z-coordinate of molecule center.
- VEXTERNC int `Valist_getNumberAtoms (Valist *thee)`
Get number of atoms in the list.
- VEXTERNC `Vatom * Valist_getAtom (Valist *thee, int i)`
Get pointer to particular atom in list.
- VEXTERNC unsigned long int `Valist_memChk (Valist *thee)`
Get total memory allocated for this object and its members.
- VEXTERNC `Valist * Valist_ctor ()`
Construct the atom list object.
- VEXTERNC Vrc_Codes `Valist_ctor2 (Valist *thee)`
FORTRAN stub to construct the atom list object.
- VEXTERNC void `Valist_dtor (Valist **thee)`
Destroys atom list object.
- VEXTERNC void `Valist_dtor2 (Valist *thee)`
FORTRAN stub to destroy atom list object.
- VEXTERNC Vrc_Codes `Valist_readPQR (Valist *thee, Vparam *param, Vio *sock)`
Fill atom list with information from a PQR file.
- VEXTERNC Vrc_Codes `Valist_readPDB (Valist *thee, Vparam *param, Vio *sock)`
Fill atom list with information from a PDB file.
- VEXTERNC Vrc_Codes `Valist_readXML (Valist *thee, Vparam *param, Vio *sock)`
Fill atom list with information from an XML file.
- VEXTERNC Vrc_Codes `Valist_getStatistics (Valist *thee)`
Load up Valist with various statistics.

9.31.1 Detailed Description

Contains declarations for class Valist.

Version

Id:

`valist.h` 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 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](#).

9.32 valist.h

```

00001
00062 #ifndef _VALIST_H_
00063 #define _VALIST_H_
00064
00065 /* Generic headers */
00066 #include "maloc/maloc.h"
00067 #include "apbs/vhal.h"
00068
00069 /* Headers specific to this file */
00070 #include "apbs/vatom.h"
00071 #include "apbs/vparam.h"
00072
00073 struct sValist {
00074

```

```

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 };
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_ */

```

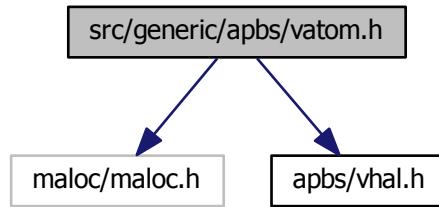
9.33 src/generic/apbs/vatom.h File Reference

Contains declarations for class Vatom.

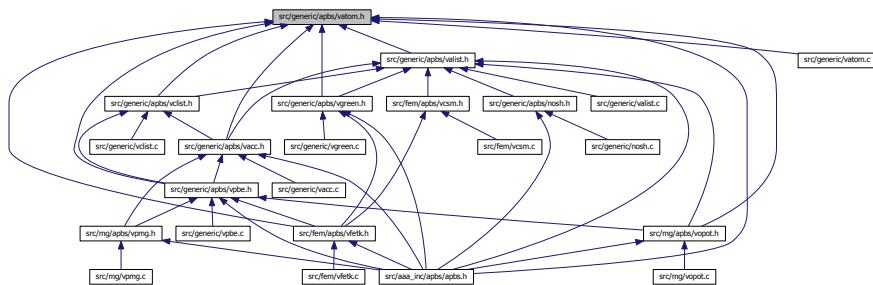
```
#include "maloc/malloc.h"
```

```
#include "apbs/vhal.h"
```

Include dependency graph for vatom.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [sVatom](#)

Contains public data members for Vatom class/module.

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.

9.33.1 Detailed Description

Contains declarations for class Vatom.

Version

Id:

[vatom.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2012 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](#).

9.34 vatom.h

```

00001
00062 #ifndef _VATOM_H_
00063 #define _VATOM_H_
00064
00065 #include "malloc/malloc.h"
00066 #include "apbs/vhal.h"
00067
00074 #define VMAX_RECLEN      64
00075
00081 struct sVatom {
00082
00083     double position[3];
00084     double radius;
00085     double charge;
00086     double partID;
00088     double epsilon;
00090     int id;
00091     char resName[VMAX_RECLEN];
00095     char atomName[VMAX_RECLEN];
00097 #if defined(WITH_TINKER)
00098
00099     double dipole[3];
00100     double quadrupole[9];
00101     double inducedDipole[3];
00102     double nLIInducedDipole[3];
00104 #endif /* if defined(WITH_TINKER) */
00105 };
00106
00111 typedef struct sVatom Vatom;
00112
00113 #if !defined(VINLINE_VATOM)
00114
00121     VEXTERNC double* Vatom_getPosition(Vatom *thee);
00122
00129     VEXTERNC void      Vatom_setRadius(Vatom *thee, double
00130         radius);
00137     VEXTERNC double    Vatom_getRadius(Vatom *thee);
00138
00146     VEXTERNC void      Vatom_setPartID(Vatom *thee, int
00147         partID);
00155     VEXTERNC double    Vatom_getPartID(Vatom *thee);
00156
00163     VEXTERNC void      Vatom_setAtomID(Vatom *thee, int id);
00164
00171     VEXTERNC double    Vatom_getAtomID(Vatom *thee);
00172
00179     VEXTERNC void      Vatom_setCharge(Vatom *thee, double
00180         charge);

```

```

00180
00187     VEXTERNC double  Vatom_getCharge(Vatom *thee);
00188
00195     VEXTERNC void      Vatom_setEpsilon(Vatom *thee, double
00196     epsilon);
00196
00203     VEXTERNC double  Vatom_getEpsilon(Vatom *thee);
00204
00212     VEXTERNC unsigned long int Vatom_memChk(Vatom *thee);
00213
00214 #else /* if defined(VINLINE_VATOM) */
00215 #  define Vatom_getPosition(thee) ((thee)->position)
00216 #  define Vatom_setRadius(thee, tRadius) ((thee)->radius = (tRadius))
00217 #  define Vatom_getRadius(thee) ((thee)->radius)
00218 #  define Vatom_setPartID(thee, tpartID) ((thee)->partID = (double)(tpartID))
00219 #  define Vatom_getPartID(thee) ((thee)->partID)
00220 #  define Vatom_setAtomID(thee, tatomID) ((thee)->id = (tatomID))
00221 #  define Vatom_getAtomID(thee) ((thee)->id)
00222 #  define Vatom_setCharge(thee, tCharge) ((thee)->charge = (tCharge))
00223 #  define Vatom_getCharge(thee) ((thee)->charge)
00224 #  define Vatom_setEpsilon(thee, tEpsilon) ((thee)->epsilon = (tEpsilon))
00225 #  define Vatom_getEpsilon(thee) ((thee)->epsilon)
00226 #  define Vatom_memChk(thee) (sizeof(Vatom))
00227 #endif /* if !defined(VINLINE_VATOM) */
00228
00229 /* //////////////////////////////// */
00230 // Class Vatom: Non-Inlineable methods (vatom.c)
00232
00239 VEXTERNC void      Vatom_setResName(Vatom *thee, char resName[VMAX_RECLEN]);
00240
00245 VEXTERNC void      Vatom_setAtomName(
00246     Vatom *thee,
00247     char atomName[VMAX_RECLEN]
00248 );
00249
00256 VEXTERNC void      Vatom_getResName(Vatom *thee, char resName[VMAX_RECLEN]);
00257
00262 VEXTERNC void      Vatom_getAtomName(
00263     Vatom *thee,
00264     char atomName[VMAX_RECLEN]
00265 );
00266
00272 VEXTERNC Vatom* Vatom_ctor();
00273
00280 VEXTERNC int       Vatom_ctor2(Vatom *thee);
00281
00287 VEXTERNC void      Vatom_dtor(Vatom **thee);
00288
00294 VEXTERNC void      Vatom_dtor2(Vatom *thee);
00295
00302 VEXTERNC void      Vatom_setPosition(Vatom *thee, double position[3]);
00303
00311 VEXTERNC void Vatom_copyTo(Vatom *thee, Vatom *dest);
00312
00320 VEXTERNC void Vatom_copyFrom(Vatom *thee, Vatom *src);
00321
00322 #if defined(WITH_TINKER)
00323
00330 VEXTERNC void      Vatom_setInducedDipole(Vatom *thee,
00331                               double inducedDipole[3]);
00332
00339 VEXTERNC void      Vatom_setNLInducedDipole(Vatom *thee,
00340                               double nlInducedDipole[3]);
00341
00348 VEXTERNC void      Vatom_setDipole(Vatom *thee, double dipole[3]);
00349
00356 VEXTERNC void      Vatom_setQuadrupole(Vatom *thee, double quadrupole[9]);
00357
00363 VEXTERNC double*   Vatom_getDipole(Vatom *thee);
00364
00370 VEXTERNC double*   Vatom_getQuadrupole(Vatom *thee);
00371
00377 VEXTERNC double*   Vatom_getInducedDipole(Vatom *thee);
00378
00384 VEXTERNC double*   Vatom_getNLInducedDipole(Vatom *thee);
00385 #endif /* if defined(WITH_TINKER) */
00386
00387 #endif /* ifndef _VATOM_H_ */

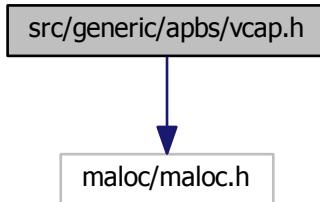
```

9.35 src/generic/apbs/vcap.h File Reference

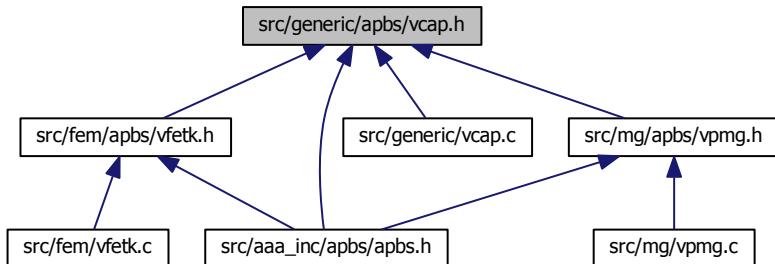
Contains declarations for class Vcap.

```
#include "maloc/maloc.h"
```

Include dependency graph for vcap.h:



This graph shows which files directly or indirectly include this file:



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.

9.35.1 Detailed Description

Contains declarations for class Vcap.

Version

Id:

[vcap.h](#) 1667 2011-12-02 23:22:02Z pcells

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2012 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](#).

9.36 vcap.h

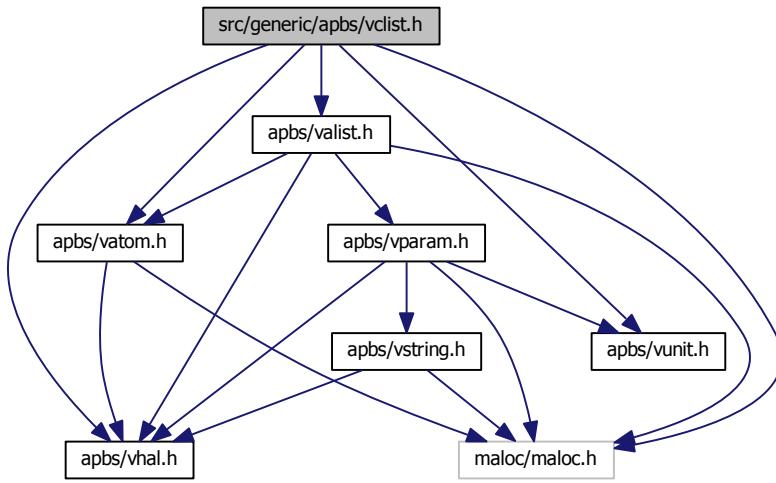
```
00001
00064 #ifndef _VCAP_H_
00065 #define _VCAP_H_
00066
00070 #define EXPMAX 85.00
00071
00075 #define EXPMIN -85.00
00076
00077 #include "malloc/malloc.h"
00078
00097 VEXTERNC double Vcap_exp(
00098     double x,
00099     int *ichop
00100 );
00101
00102
00121 VEXTERNC double Vcap_sinh(
00122     double x,
00123     int *ichop
00124 );
00125
00144 VEXTERNC double Vcap_cosh(
00145     double x,
00146     int *ichop
00147 );
00148
00149 #endif /* ifndef _VCAP_H_ */
```

9.37 src/generic/apbs/vclist.h File Reference

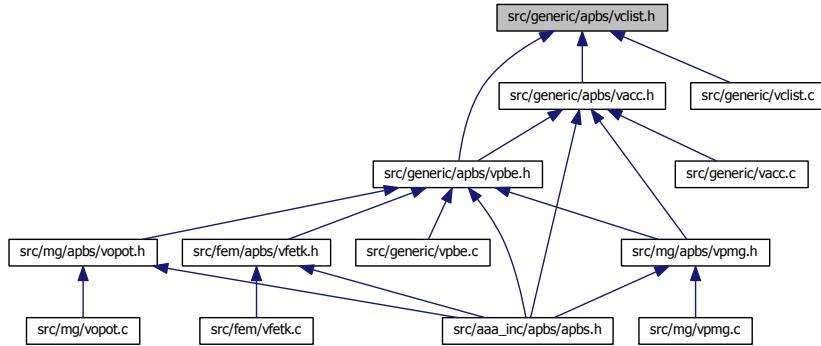
Contains declarations for class Vclist.

```
#include "malloc/malloc.h"
#include "apbs/vhal.h"
#include "apbs/valist.h"
#include "apbs/vatom.h"
#include "apbs/vunit.h"
```

Include dependency graph for vclist.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `sVclistCell`
Atom cell list cell.
- struct `sVclist`
Atom cell list.

Typedefs

- typedef enum `eVclist_DomainMode` `Vclist_DomainMode`

Declaration of Vclist_DomainMode enumeration type.

- **typedef struct sVclistCell VclistCell**

Declaration of the VclistCell class as the VclistCell structure.

- **typedef struct sVclist Vclist**

Declaration of the Vclist class as the Vclist structure.

Enumerations

- enum **eVclist_DomainMode { CLIST_AUTO_DOMAIN, CLIST_MANUAL_DOMAIN }**

Atom cell list domain setup mode.

Functions

- VEXTERNC unsigned long int **Vclist_memChk (Vclist *thee)**

Get number of bytes in this object and its members.

- VEXTERNC double **Vclist_maxRadius (Vclist *thee)**

Get the max probe radius value (in A) the cell list was constructed with.

- VEXTERNC **Vclist * Vclist_ctor (Valist *alist, double max_radius, int npts[VAPBS_DIM], Vclist_DomainMode mode, double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM])**

Construct the cell list object.

- VEXTERNC **Vrc_Codes Vclist_ctor2 (Vclist *thee, Valist *alist, double max_radius, int npts[VAPBS_DIM], Vclist_DomainMode mode, double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM])**

FORTRAN stub to construct the cell list object.

- VEXTERNC void **Vclist_dtor (Vclist **thee)**

Destroy object.

- VEXTERNC void **Vclist_dtor2 (Vclist *thee)**

FORTRAN stub to destroy object.

- VEXTERNC **VclistCell * Vclist_getCell (Vclist *thee, double position[VAPBS_DIM])**

Return cell corresponding to specified position or return VNULL.

- VEXTERNC **VclistCell * VclistCell_ctor (int natoms)**

Allocate and construct a cell list cell object.

- VEXTERNC **Vrc_Codes VclistCell_ctor2 (VclistCell *thee, int natoms)**

Construct a cell list object.

- VEXTERNC void **VclistCell_dtor (VclistCell **thee)**

Destroy object.

- VEXTERNC void **VclistCell_dtor2 (VclistCell *thee)**

FORTRAN stub to destroy object.

9.37.1 Detailed Description

Contains declarations for class Vclist.

Version

Id:

[vclist.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2012 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](#).

9.38 vclist.h

```
00001  
00062 #ifndef _VCLIST_H_  
00063 #define _VCLIST_H_  
00064  
00065 /* Generic headers */  
00066 #include "malloc/malloc.h"  
00067 #include "apbs/vhal.h"  
00068
```

```

00069 /* Headers specific to this file */
00070 #include "apbs/valist.h"
00071 #include "apbs/vatom.h"
00072 #include "apbs/vunit.h"
00073
00079 enum eVclist_DomainMode {
00080     CLIST_AUTO_DOMAIN,
00082     CLIST_MANUAL_DOMAIN
00084 };
00085
00091 typedef enum eVclist_DomainMode Vclist_DomainMode
00092 ;
00092
00098 struct sVclistCell {
00099     Vatom **atoms;
00100     int natoms;
00101 };
00102
00107 typedef struct sVclistCell VclistCell;
00108
00114 struct sVclist {
00115
00116     Vmem *vmem;
00117     Valist *alist;
00118     Vclist_DomainMode mode;
00119     int npts[VAPBS_DIM];
00120     int n;
00121     double max_radius;
00122     VclistCell *cells;
00123     double lower_corner[VAPBS_DIM];
00124     double upper_corner[VAPBS_DIM];
00125     double spacs[VAPBS_DIM];
00127 };
00128
00133 typedef struct sVclist Vclist;
00134
00135 #if !defined(VINLINE_VCLIST)
00136
00142     VEXTERNC unsigned long int Vclist_memChk(
00143         Vclist *thee
00144     );
00145
00153     VEXTERNC double Vclist_maxRadius(
00154         Vclist *thee
00155     );
00156
00157 #else /* if defined(VINLINE_VCLIST) */
00158
00159 #    define Vclist_memChk(thee) (Vmem_bytes((thee)->vmem))
00160 #    define Vclist_maxRadius(thee) ((thee)->max_radius)
00161
00162 #endif /* if !defined(VINLINE_VCLIST) */
00163
00164 /* //////////////////////////////// */
00165 // Class Vclist: Non-Inlineable methods (vclist.c)
00167
00172 VEXTERNC Vclist* Vclist_ctor(
00173     Valist *alist,
00174     double max_radius,
00175     int npts[VAPBS_DIM],
00177     Vclist_DomainMode mode,
00178     double lower_corner[VAPBS_DIM],
00181     double upper_corner[VAPBS_DIM]
00184 );
00185
00190 VEXTERNC Vrc_Codes Vclist_ctor2(
00191     Vclist *thee,
00192     Valist *alist,
00193     double max_radius,
00194     int npts[VAPBS_DIM],
00196     Vclist_DomainMode mode,
00197     double lower_corner[VAPBS_DIM],
00200     double upper_corner[VAPBS_DIM]
00203 );
00204
00209 VEXTERNC void Vclist_dtor(
00210     Vclist **thee
00211 );
00212
00217 VEXTERNC void Vclist_dtor2(
00218     Vclist *thee

```

```

00219      );
00220
00228 VEXTERNC VclistCell* Vclist_getCell(
00229     Vclist *thee,
00230     double position[VAPBS_DIM]
00231 );
00232
00239 VEXTERNC VclistCell* VclistCell_ctor(
00240     int natoms
00241 );
00242
00249 VEXTERNC Vrc_Codes VclistCell_ctor2(
00250     VclistCell *thee,
00251     int natoms
00252 );
00253
00258 VEXTERNC void VclistCell_dtor(
00259     VclistCell **thee
00260 );
00261
00266 VEXTERNC void VclistCell_dtor2(
00267     VclistCell *thee
00268 );
00269
00270 #endif /* ifndef _VCLIST_H_ */

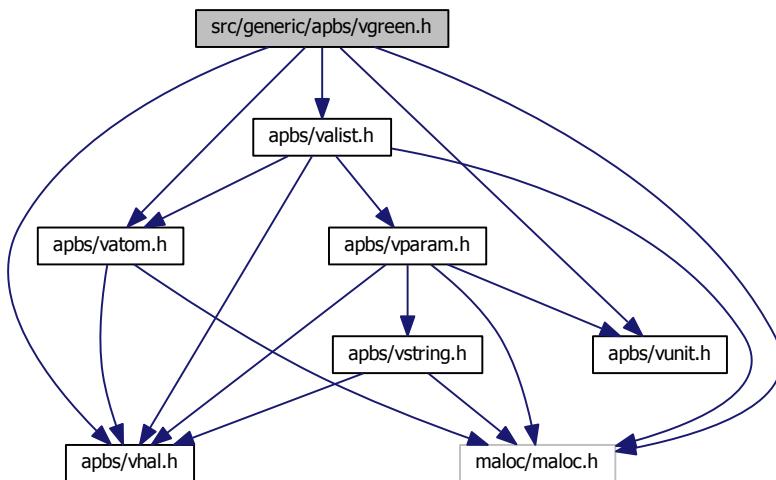
```

9.39 src/generic/apbs/vgreen.h File Reference

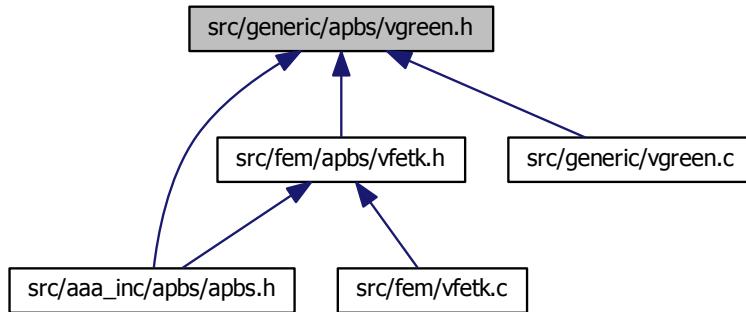
Contains declarations for class Vgreen.

```
#include "maloc/malloc.h"
#include "apbs/vhal.h"
#include "apbs/vunit.h"
#include "apbs/vatom.h"
#include "apbs/valist.h"
```

Include dependency graph for vgreen.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [sVgreen](#)
Contains public data members for Vgreen class/module.

Typedefs

- typedef struct [sVgreen](#) [Vgreen](#)
Declaration of the Vgreen class as the Vgreen structure.

Functions

- VEXTERNC [Valist](#) * [Vgreen_getValist](#) ([Vgreen](#) *thee)
Get the atom list associated with this Green's function object.
- VEXTERNC unsigned long int [Vgreen_memChk](#) ([Vgreen](#) *thee)
Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC [Vgreen](#) * [Vgreen_ctor](#) ([Valist](#) *alist)
Construct the Green's function oracle.
- VEXTERNC int [Vgreen_ctor2](#) ([Vgreen](#) *thee, [Valist](#) *alist)
FORTRAN stub to construct the Green's function oracle.
- VEXTERNC void [Vgreen_dtor](#) ([Vgreen](#) **thee)
Destruct the Green's function oracle.
- VEXTERNC void [Vgreen_dtor2](#) ([Vgreen](#) *thee)
FORTRAN stub to destruct the Green's function oracle.
- VEXTERNC int [Vgreen_helmholtz](#) ([Vgreen](#) *thee, int npos, double *x, double *y, double *z, double *val, double kappa)
Get the Green's function for Helmholtz's equation integrated over the atomic point charges.
- VEXTERNC int [Vgreen_helmholtzD](#) ([Vgreen](#) *thee, int npos, double *x, double *y, double *z, double *gradx, double *grady, double *gradz, double kappa)

Get the gradient of Green's function for Helmholtz's equation integrated over the atomic point charges.

- VEXTERNC int [Vgreen_coulomb_direct](#) ([Vgreen](#) *thee, int npos, double *x, double *y, double *z, double *val)
Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.
- VEXTERNC int [Vgreen_coulomb](#) ([Vgreen](#) *thee, int npos, double *x, double *y, double *z, double *val)
Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation or H. E. Johnston, R. Krasny FMM library (if available)
- VEXTERNC int [Vgreen_coulombD_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)

9.39.1 Detailed Description

Contains declarations for class Vgreen.

Version

Id:

[vgreen.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Definition in file [vgreen.h](#).

9.40 vgreen.h

```

00001
00065 #ifndef _VGREEN_H_
00066 #define _VGREEN_H_
00067
00068 /* Generic headers */
00069 #include "maloc/maloc.h"
00070 #include "apbs/vhal.h"
00071
00072 /* Specific headers */
00073 #include "apbs/vunit.h"
00074 #include "apbs/vatom.h"
00075 #include "apbs/valist.h"
00076
00077
00083 struct sVgreen {
00084
00085   Valist *alist;
00086   Vmem *vmem;
00087   double *xp;
00088   double *yp;
00089   double *zp;
00090   double *qp;
00091   int np;

```

```

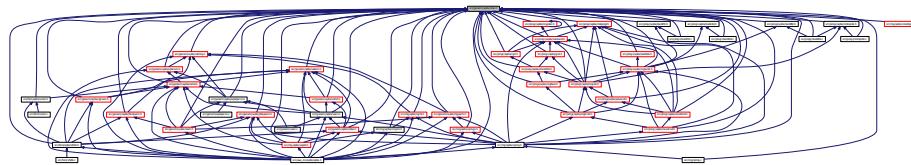
00096 };
00097
00102 typedef struct sVgreen Vgreen;
00103
00104 /* //////////////////////////////// Class Vgreen: Inlineable methods (vgreen.c) */
00105 // Class Vgreen: Inlineable methods (vgreen.c)
00107
00108 #if !defined(VINLINE_VGREEN)
00109
00117     VEXTERNC Valist* Vgreen_getValist(Vgreen *thee);
00118
00126     VEXTERNC unsigned long int Vgreen_memChk(Vgreen *thee);
00127
00128 #else /* if defined(VINLINE_VGREEN) */
00129 #    define Vgreen_getValist(thee) ((thee)->alist)
00130 #    define Vgreen_memChk(thee) (Vmem_bytes((thee)->vmem))
00131 #endif /* if !defined(VINLINE_VGREEN) */
00132
00133 /* //////////////////////////////// Class Vgreen: Non-Inlineable methods (vgreen.c) */
00134 // Class Vgreen: Non-Inlineable methods (vgreen.c)
00136
00143 VEXTERNC Vgreen* Vgreen_ctor(Valist *alist);
00144
00152 VEXTERNC int Vgreen_ctor2(Vgreen *thee, Valist *alist);
00153
00159 VEXTERNC void Vgreen_dtor(Vgreen **thee);
00160
00166 VEXTERNC void Vgreen_dtor2(Vgreen *thee);
00167
00192 VEXTERNC int Vgreen_helmholtz(Vgreen *thee, int npos, double *x, double *y,
00193     double *z, double *val, double kappa);
00194
00222 VEXTERNC int Vgreen_helmholtzD(Vgreen *thee, int npos, double *x, double *y,
00223     double *z, double *gradx, double *grady, double *gradz, double kappa);
00224
00245 VEXTERNC int Vgreen_coulomb_direct(Vgreen *thee, int npos, double *x,
00246     double *y, double *z, double *val);
00247
00268 VEXTERNC int Vgreen_coulomb(Vgreen *thee, int npos, double *x, double *y,
00269     double *z, double *val);
00270
00294 VEXTERNC int Vgreen_coulombD_direct(Vgreen *thee, int npos, double *x,
00295     double *y, double *z, double *pot, double *gradx, double *grady, double
00296     *gradz);
00297
00322 VEXTERNC int Vgreen_coulombD(Vgreen *thee, int npos, double *x, double *y,
00323     double *z, double *pot, double *gradx, double *grady, double *gradz);
00324
00325 #endif /* ifndef _VGREEN_H_ */

```

9.41 src/generic/apbs/vhal.h File Reference

Contains generic macro definitions for APBS.

This graph shows which files directly or indirectly include this file:



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 PRINT_FUNC __PRETTY_FUNCTION__

- #define **OS_SEP_STR** "/"
 - #define **OS_SEP_CHAR** '/'
- #define **ANNOUNCE_FUNCTION**
- #define **WARN_FORTRAN**
- #define **WARN_UNTESTED**
- #define **WARN_PARTTESTED**
- #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 **VMESSAGE0**(msg)
- #define **VMESSAGE1**(msg, arg)
- #define **VMESSAGE2**(msg, arg0, arg1)
- #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, arg)
- #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,
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_TRI = 0, VCM_BSPL2 = 1, VCM_BSPL4 = 2 }**
Types of charge discretization methods.
- **enum eVchrg_Src { VCM_CHARGE = 0, VCM_PERMANENT = 1, VCM_INDUCED = 2, VCM_NLINDUCED = 3
}**
Charge source.
- **enum eVdata_Type { VDT_CHARGE, VDT_POT, VDT_ATOMPOT, VDT_SMOL,
VDT_SSPL, VDT_VDW, VDT_IVDW, VDT_LAP,
VDT_EDENS, VDT_NDENS, VDT_QDENS, VDT_DIELX,
VDT_DIELY, VDT_DIELZ, VDT_KAPPA }**
Types of (scalar) data that can be written out of APBS.

- enum **eVdata_Format** {
 VDF_DX = 0, **VDF_UHBD** = 1, **VDF_AVIS** = 2, **VDF_MCSF** = 3,
 VDF_GZ = 4, **VDF_FLAT** = 5 }

Format of data for APBS I/O.

Functions

- char * **wrap_text** (char *str, int right_margin, int left_padding)

9.41.1 Detailed Description

Contains generic macro definitions for APBS.

Version

Id:

[vhal.h](#) 1767 2012-07-19 00:53:54Z sobolevnrm

Author

Nathan A. Baker

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pn1.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](#).

9.41.2 Macro Definition Documentation

9.41.2.1 #define PRINT_DBL(*expr*)

Value:

```
do {
    Vnm_print(2, "%s:%d [%s(): %s == %f\n\n",
              __FILE__, __LINE__, __FUNCTION__, #expr, expr); \
} while(0)
```

Definition at line 769 of file [vhal.h](#).

9.41.2.2 #define PRINT_FUNC __PRETTY_FUNCTION__

OS specific flags and etcetera

Definition at line 520 of file [vhal.h](#).

9.41.2.3 #define PRINT_INT(*expr*)

Value:

```
do {
    Vnm_print(2, "%s:%d [%s(): %s == %d\n",
              __FILE__, __LINE__, __FUNCTION__, #expr, expr); \
} while(0)
```

Definition at line 763 of file [vhal.h](#).

9.41.2.4 #define VCOPY(*srcvec*, *dstvec*, *i*, *n*)

Value:

```
do {
    for (i = 0; i < n; i++) \
        dstvec[i] = srcvec[i]; \
} while(0)
```

Definition at line 786 of file [vhal.h](#).

9.41.2.5 #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 779 of file [vhal.h](#).

9.41.2.6 #define VMESSAGE0(msg)

Value:

```

do {
    Vnm_print(2, "%s:%d [%s()]: MESSAGE:\n" \
              " %s\n\n", \
              __FILE__, __LINE__, __FUNCTION__, msg); \
} while(0)

```

Definition at line 597 of file [vhal.h](#).

9.41.2.7 #define VMESSAGE1(msg, arg)

Value:

```

do {
    char buff[1000];
    snprintf( buff, 1000, msg, arg );
    Vnm_print(2, "%s:%d [%s()]: MESSAGE:\n" \
              " %s\n\n", \
              __FILE__, __LINE__, __FUNCTION__, buff); \
} while(0)

```

Definition at line 604 of file [vhal.h](#).

9.41.2.8 #define VMESSAGE2(msg, arg0, arg1)

Value:

```

do {
    char buff[1000];
    snprintf( buff, 1000, msg, arg0, arg1 );
    Vnm_print(2, "%s:%d [%s()]: MESSAGE:\n" \
              " %s\n\n", \
              __FILE__, __LINE__, __FUNCTION__, buff); \
} while(0)

```

Definition at line 613 of file [vhal.h](#).

9.42 vhal.h

```

00001
00056 #ifndef _VAPBSHAL_H_
00057 #define _VAPBSHAL_H_
00058
00065 enum eVrc_Codes {
00066
00067     VRC_WARNING=-1,
00068     VRC_FAILURE=0,
00069     VRC_SUCCESS=1

```

```
00071 };
00072 typedef enum eVrc_Codes Vrc_Codes;
00073
00080 enum eVsol_Meth {
00081
00082     VSOL_CGMG, /* 0: conjugate gradient multigrid */
00083     VSOL_Newton, /* 1: newton */
00084     VSOL_MG, /* 2: multigrid */
00085     VSOL(CG, /* 3: conjugate gradient */
00086     VSOL_SOR, /* 4: sucessive overrelaxation */
00087     VSOL_RBGS, /* 5: red-black gauss-seidel */
00088     VSOL_WJ, /* 6: weighted jacobi */
00089     VSOL_Richardson, /* 7: richardson */
00090     VSOL_CGMGAqua, /* 8: conjugate gradient multigrid aqua */
00091     VSOL_NewtonAqua /* 9: newton aqua */
00092
00093 };
00094 typedef enum eVsol_Meth Vsol_Meth;
00095
00101 enum eVsurf_Meth {
00102     VSM_MOL=0,
00106         VSM_MOLSMOOTH=1,
00108     VSM_SPLINE=2,
00118     VSM_SPLINE3=3,
00122     VSM_SPLINE4=4
00126 };
00127
00132 typedef enum eVsurf_Meth Vsurf_Meth;
00133
00138 enum eVhal_PBEType {
00139     PBE_LPBE,
00140     PBE_NPBE,
00141     PBE_LRPBE,
00142     PBE_NRPBE,
00143     PBE_SMPBE
00144 };
00145
00150 typedef enum eVhal_PBEType Vhal_PBEType;
00151
00156 enum eVhal_IPKEYType {
00157     IPKEY_SMPBE = -2,
00158     IPKEY_LPBE,
00159     IPKEY_NPBE
00160 };
00161
00166 typedef enum eVhal_IPKEYType Vhal_IPKEYType;
00167
00172 enum eVhal_NONLINType {
00173     NONLIN_LPBE = 0,
00174     NONLIN_NPBE,
00175         NONLIN_SMPBE,
00176     NONLIN_LPBEAQUA,
00177     NONLIN_NPBEAQUA
00178 };
00179
00184 typedef enum eVhal_NONLINType Vhal_NONLINType;
00185
00190 enum eVoutput_Format {
00191     OUTPUT_NULL,
00192     OUTPUT_FLAT,
00193 };
00194
00199 typedef enum eVoutput_Format Voutput_Format;
00200
00206 enum eVbcfl {
00207     BCFL_ZERO=0,
00208     BCFL_SDH=1,
00210     BCFL_MDH=2,
00212     BCFL_UNUSED=3,
00213     BCFL_FOCUS=4,
00214     BCFL_MEM=5,
00215     BCFL_MAP=6
00216 };
00217
00222 typedef enum eVbcfl Vbcfl;
00223
00229 enum eVchrg_Meth {
00230     VCM_TRIIL=0,
00233     VCM_BSPL2=1,
00236     VCM_BSPL4=2
00237 };
```

```
00238
00243 typedef enum eVchrg_Meth Vchrg_Meth;
00244
00250 enum eVchrg_Src {
00251     VCM_CHARGE=0,
00252     VCM_PERMANENT=1,
00253     VCM_INDUCED=2,
00254     VCM_NLINDUCED=3
00255 };
00256
00261 typedef enum eVchrg_Src Vchrg_Src;
00262
00268 enum eVdata_Type {
00269     VDT_CHARGE,
00270     VDT_POT,
00271     VDT_ATOMPOT,
00272     VDT_SMOL,
00274     VDT_SSPL,
00276     VDT_VDW,
00278     VDT_IVDW,
00280     VDT_LAP,
00281     VDT_EDENS,
00283     VDT_NDENS,
00285     VDT_QDENS,
00287     VDT_DIELX,
00289     VDT_DIELY,
00291     VDT_DIELZ,
00293     VDT_KAPPA
00295 };
00296
00301 typedef enum eVdata_Type Vdata_Type;
00302
00308 enum eVdata_Format {
00309     VDF_DX=0,
00310     VDF_UHBD=1,
00311     VDF_AVIS=2,
00312     VDF_MCSF=3,
00313     VDF_GZ=4,
00314     VDF_FLAT=5
00315 };
00316
00321 typedef enum eVdata_Format Vdata_Format;
00322
00327 #define APBS_TIMER_WALL_CLOCK 26
00328
00333 #define APBS_TIMER_SETUP 27
00334
00339 #define APBS_TIMER_SOLVER 28
00340
00345 #define APBS_TIMER_ENERGY 29
00346
00351 #define APBS_TIMER_FORCE 30
00352
00357 #define APBS_TIMER_TEMP1 31
00358
00363 #define APBS_TIMER_TEMP2 32
00364
00369 #define MAXMOL 5
00370
00375 #define MAXION 10
00376
00380 #define MAXFOCUS 5
00381
00385 #define VMGNLEV 4
00386
00390 #define VREDFRAC 0.25
00391
00395 #define VAPBS_NVS 4
00396
00400 #define VAPBS_DIM 3
00401
00406 #define VAPBS_RIGHT 0
00407
00412 #define MAX_SPHERE PTS 50000
00413
00418 #define VAPBS_FRONT 1
00419
00424 #define VAPBS_UP 2
00425
00430 #define VAPBS_LEFT 3
00431
```

```
00436 #define VAPBS_BACK 4
00437
00442 #define VAPBS_DOWN 5
00443
00448 #define VPMGSMALL 1e-12
00449
00454 #define SINH_MIN -85.0
00455
00460 #define SINH_MAX 85.0
00461
00462 #if defined(VDEBUG)
00463 #  if !defined(APBS_NOINLINE)
00464 #    define APBS_NOINLINE 1
00465 #  endif
00466 #endif
00467
00468 #if !defined(APBS_NOINLINE)
00469
00473 #  define VINLINE_VACC
00474
00478 #  define VINLINE_VATOM
00479
00483 #  define VINLINE_VCSM
00484
00488 #  define VINLINE_VPBE
00489
00493 #  define VINLINE_VPEE
00494
00498 #  define VINLINE_VGREEN
00499
00503 #  define VINLINE_VFETK
00504
00508 #  define VINLINE_VPMG
00509
00514 #  define MAX_HASH_DIM 75
00515
00516 #endif
00517
00519 #if !defined(WIN32) || defined(__MINGW32__)
00520 #define PRINT_FUNC __PRETTY_FUNCTION__
00521 #define OS_SEP_STR "/"
00522 #define OS_SEP_CHAR '/'
00523 #else
00524 #define OS_SEP_STR "\\"
00525 #define OS_SEP_CHAR '\\'
00526 #define PRINT_FUNC __FUNCSIG__
00527 #endif
00528
00529 #ifdef VERBOSE_DEBUG
00530 #define ANNOUNCE_FUNCTION do{fprintf(stderr, "%s() [%s:%d]\\
n", PRINT_FUNC, __FILE__, __LINE__);}while(0)
00531 #define WARN_FORTRAN do{fprintf(stderr, "%s() [%s:%d]: Calling Fortran
Subroutine!!!\n", __FUNCTION__, __FILE__, __LINE__);}while(0)
00532 #define WARN_UNTESTED do{fprintf(stderr, "%s() [%s:%d]: Untested
Translation!\n", __FUNCTION__, __FILE__, __LINE__);}while(0)
00533 #define WARN_PARTTESTED do{fprintf(stderr, "%s() [%s:%d]: Partially Tested
Translation.\n", __FUNCTION__, __FILE__, __LINE__);}while(0)
00534 #else
00535 #define ANNOUNCE_FUNCTION
00536 #define WARN_FORTRAN
00537 #define WARN_UNTESTED
00538 #define WARN_PARTTESTED
00539 #endif
00540
00541 /* Fortran name mangling */
00542 #if defined(VF77_UPPERCASE)
00543 #  if defined(VF77_NOUNDERSCORE)
00544 #    define VF77_MANGLE(name,NAME) NAME
00545 #  elif defined(VF77_ONEUNDERSCORE)
00546 #    define VF77_MANGLE(name,NAME) NAME ## _
00547 #  else
00548 #    define VF77_MANGLE(name,NAME) name
00549 #  endif
00550 #else
00551 #  if defined(VF77_NOUNDERSCORE)
00552 #    define VF77_MANGLE(name,NAME) name
00553 #  elif defined(VF77_ONEUNDERSCORE)
00554 #    define VF77_MANGLE(name,NAME) name ## _
00555 #  else
00556
00559 #    define VF77_MANGLE(name,NAME) name
```



```

00665     char buff[1000];
00666     snprintf( buff, 1000, msg, arg0, arg1 );
00667     Vnm_print(2, "%s:%d [%s()]: ERROR:\n"
00668             "    Assertion Failed (%s): %s\n\n",
00669             __FILE__, __LINE__, __FUNCTION__, #cnd, buff);
00670     abort();
00671 }
00672 } while(0)
00673
00679 #define VWARN_MSG0(cnd, msg)
00680 do {
00681     if( (cnd) == 0 ) {
00682         Vnm_print(2, "%s:%d [%s()]: WARNING:\n"
00683                 "    Condition Failed (%s): %s\n\n",
00684                 __FILE__, __LINE__, __FUNCTION__, #cnd, msg);
00685     }
00686 } while(0)
00687
00693 #define VWARN_MSG1(cnd, msg, arg)
00694 do {
00695     if( (cnd) == 0 ) {
00696         char buff[1000];
00697         snprintf( buff, 1000, msg, arg );
00698         Vnm_print(2, "%s:%d [%s()]: WARNING:\n"
00699                 "    Condition Failed (%s): %s\n\n",
00700                 __FILE__, __LINE__, __FUNCTION__, #cnd, buff);
00701     }
00702 } while(0)
00703
00709 #define VWARN_MSG2(cnd, msg, arg0, arg1)
00710 do {
00711     if( (cnd) == 0 ) {
00712         char buff[1000];
00713         snprintf( buff, 1000, msg, arg0, arg1 );
00714         Vnm_print(2, "%s:%d [%s()]: WARNING:\n"
00715                 "    Condition Failed (%s): %s\n\n",
00716                 __FILE__, __LINE__, __FUNCTION__, #cnd, buff);
00717     }
00718 } while(0)
00719
00725 #define VABORT_MSG0(msg)
00726 do {
00727     Vnm_print(2, "%s:%d [%s()]: ABORTING:\n"
00728             "    %s\n\n",
00729             __FILE__, __LINE__, __FUNCTION__, msg);
00730     abort();
00731 } while(0)
00732
00738 #define VABORT_MSG1(msg, arg)
00739 do {
00740     char buff[1000];
00741     snprintf( buff, 1000, msg, arg );
00742     Vnm_print(2, "%s:%d [%s()]: ABORTING:\n"
00743             "    %s\n\n",
00744             __FILE__, __LINE__, __FUNCTION__, buff);
00745     abort();
00746 } while(0)
00747
00753 #define VABORT_MSG2(msg, arg0, arg1)
00754 do {
00755     char buff[1000];
00756     snprintf( buff, 1000, msg, arg0, arg1 );
00757     Vnm_print(2, "%s:%d [%s()]: ABORTING:\n"
00758             "    %s\n\n",
00759             __FILE__, __LINE__, __FUNCTION__, buff);
00760     abort();
00761 } while(0)
00762
00763 #define PRINT_INT(expr)
00764 do {
00765     Vnm_print(2, "%s:%d [%s()]: %s == %d\n",
00766             __FILE__, __LINE__, __FUNCTION__, #expr, expr);
00767 } while(0)
00768
00769 #define PRINT_DBL(expr)
00770 do {
00771     Vnm_print(2, "%s:%d [%s()]: %s == %f\n\n",
00772             __FILE__, __LINE__, __FUNCTION__, #expr, expr);
00773 } while(0)
00774
00775 #define VMALLOC(vmem, n, type) ((type*)Vmem_malloc(vmem, n, sizeof(type)))

```

```

00776
00777 #define VFREE(vmem, n, type, ptr) (Vmem_free(vmem, n, sizeof(type), (void
00778     **)&(ptr)))
00778
00779 #define VFILL(vec, n, val)
00780     do {
00781         int fill_idx;
00782         for (fill_idx = 0; fill_idx < n; fill_idx++) \
00783             vec[fill_idx] = val;
00784     } while(0)
00785
00786 #define VCOPY(srcvec, dstvec, i, n) \
00787     do {
00788         for (i = 0; i < n; i++) \
00789             dstvec[i] = srcvec[i];
00790     } while(0)
00791
00792
00793 char* wrap_text( char* str, int right_margin, int left_padding );
00794
00795 #define VAT(array, i) ((array)[(i) - 1])
00796 #define RAT(array, i) ((array) + i - 1)
00797
00798 #endif /* #ifndef _VAPBSHAL_H_ */

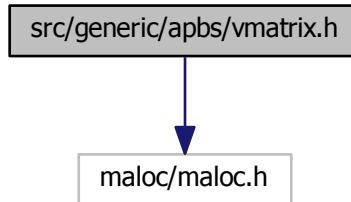
```

9.43 src/generic/apbs/vmatrix.h File Reference

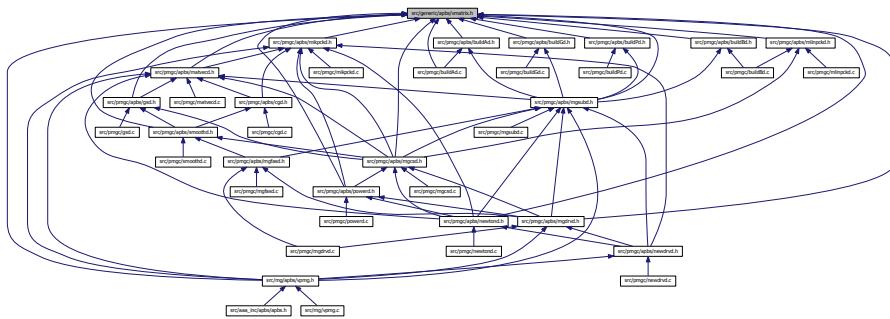
Contains inclusions for matrix data wrappers.

```
#include "maloc/maloc.h"
```

Include dependency graph for vmatrix.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define **MAT2**(mat, rows, cols)
- #define **RAT2**(mat, i, j) (mat + ((j - 1) * rows_##mat + (i - 1)))
- #define **VAT2**(mat, i, j) mat[(j - 1) * rows_##mat + (i - 1)]
- #define **MAT3**(mat, rows, cols, levs)
- #define **RAT3**(mat, i, j, k) (mat + ((k - 1) * rows_##mat * cols_##mat + (j - 1) * rows_##mat + (i - 1)))
- #define **VAT3**(mat, i, j, k) mat[(k - 1) * rows_##mat * cols_##mat + (j - 1) * rows_##mat + (i - 1)]

9.43.1 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](#).

9.43.2 Macro Definition Documentation

9.43.2.1 #define MAT2(mat, rows, cols)

Value:

```
int rows_##mat = rows;      \
int cols_##mat = rows
```

Definition at line [67](#) of file [vmatrix.h](#).

9.43.2.2 #define MAT3(mat, rows, cols, levs)

Value:

```
int rows_##mat = rows;          \
int cols_##mat = cols;          \
int levs_##mat = levs
```

Definition at line [77](#) of file [vmatrix.h](#).

9.44 vmatrix.h

```
00001
00061 #ifndef _VMATRIX_H_
00062 #define _VMATRIX_H_
00063
00064 /* Generic headers */
```

```

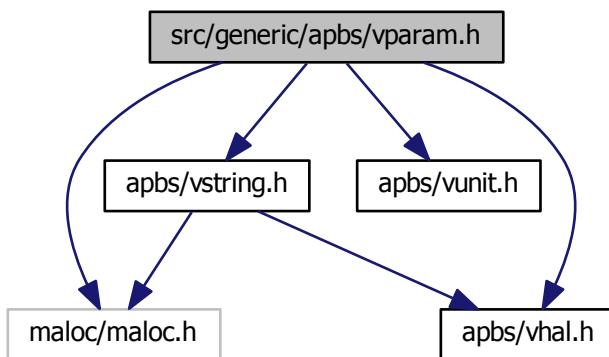
00065 #include "malloc/malloc.h"
00066
00067 #define MAT2(mat, rows, cols) \
00068     int rows_##mat = rows; \
00069     int cols_##mat = rows
00070
00071 #define RAT2(mat, i, j) \
00072     (mat + ((j - 1) * rows_##mat + (i - 1)))
00073
00074 #define VAT2(mat, i, j) \
00075     mat[(j - 1) * rows_##mat + (i - 1)]
00076
00077 #define MAT3(mat, rows, cols, levs) \
00078     int rows_##mat = rows; \
00079     int cols_##mat = cols; \
00080     int levs_##mat = levs
00081
00082 #define RAT3(mat, i, j, k) \
00083     (mat + ((k - 1) * rows_##mat * cols_##mat + (j - 1) * rows_##mat + (i - 1)))
00084
00085 #define VAT3(mat, i, j, k) \
00086     mat[(k - 1) * rows_##mat * cols_##mat + (j - 1) * rows_##mat + (i - 1)]
00087
00088 #endif /* _VMATRIX_H_ */

```

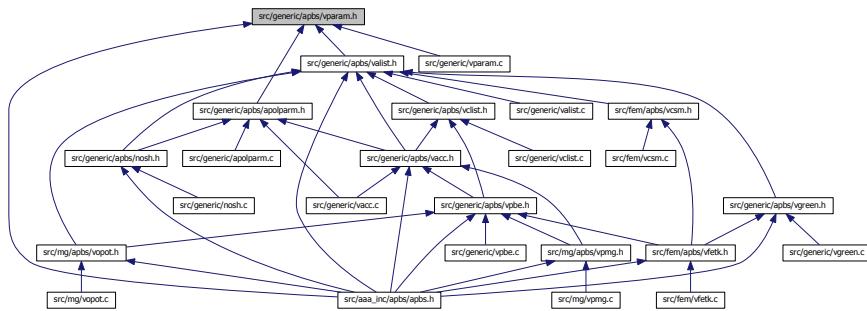
9.45 src/generic/apbs/vparam.h File Reference

Contains declarations for class [Vparam](#).

```
#include "malloc/malloc.h"
#include "apbs/vhal.h"
#include "apbs/vunit.h"
#include "apbs/vstring.h"
Include dependency graph for vparam.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct **sVparam_AtomData**
AtomData sub-class; stores atom data.
- struct **Vparam_ResData**
ResData sub-class; stores residue data.
- struct **Vparam**
Reads and assigns charge/radii parameters.

Typedefs

- typedef struct **sVparam_AtomData** **Vparam_AtomData**
Declaration of the Vparam_AtomData class as the sVparam_AtomData structure.
- typedef struct **Vparam_ResData** **Vparam_ResData**
Declaration of the Vparam_ResData class as the Vparam_ResData structure.
- typedef struct **Vparam** **Vparam**
Declaration of the Vparam class as the Vparam structure.

Functions

- VEXTERNC unsigned long int **Vparam_memChk** (**Vparam** *thee)
Get number of bytes in this object and its members.
- VEXTERNC **Vparam_AtomData** * **Vparam_AtomData_ctor** ()
Construct the object.
- VEXTERNC int **Vparam_AtomData_ctor2** (**Vparam_AtomData** *thee)
FORTRAN stub to construct the object.
- VEXTERNC void **Vparam_AtomData_dtor** (**Vparam_AtomData** **thee)
Destroy object.
- VEXTERNC void **Vparam_AtomData_dtor2** (**Vparam_AtomData** *thee)
FORTRAN stub to destroy object.
- VEXTERNC void **Vparam_AtomData_copyTo** (**Vparam_AtomData** *thee, **Vparam_AtomData** *dest)
Copy current atom object to destination.
- VEXTERNC void **Vparam_ResData_copyTo** (**Vparam_ResData** *thee, **Vparam_ResData** *dest)

Copy current residue object to destination.

- VEXTERNC void [Vparam_AtomData_copyFrom](#) (Vparam_AtomData *thee, Vparam_AtomData *src)

Copy current atom object from another.

- VEXTERNC [Vparam_ResData * Vparam_ResData_ctor](#) (Vmem *mem)

Construct the object.

- VEXTERNC int [Vparam_ResData_ctor2](#) (Vparam_ResData *thee, Vmem *mem)

FORTRAN stub to construct the object.

- VEXTERNC void [Vparam_ResData_dtor](#) (Vparam_ResData **thee)

Destroy object.

- VEXTERNC void [Vparam_ResData_dtor2](#) (Vparam_ResData *thee)

FORTRAN stub to destroy object.

- VEXTERNC [Vparam * Vparam_ctor](#) ()

Construct the object.

- VEXTERNC int [Vparam_ctor2](#) (Vparam *thee)

FORTRAN stub to construct the object.

- VEXTERNC void [Vparam_dtor](#) (Vparam **thee)

Destroy object.

- VEXTERNC void [Vparam_dtor2](#) (Vparam *thee)

FORTRAN stub to destroy object.

- VEXTERNC [Vparam_ResData * Vparam_getResData](#) (Vparam *thee, char resName[VMAX_ARGLEN])

Get residue data.

- VEXTERNC [Vparam_AtomData * Vparam_getAtomData](#) (Vparam *thee, char resName[VMAX_ARGLEN], char atomName[VMAX_ARGLEN])

Get atom data.

- VEXTERNC int [Vparam_readFlatFile](#) (Vparam *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname)

Read a flat-file format parameter database.

- VEXTERNC int [Vparam_readXMLFile](#) (Vparam *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname)

Read an XML format parameter database.

9.45.1 Detailed Description

Contains declarations for class [Vparam](#).

Version

Id:

vparam.h 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 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](#).

9.46 vparam.h

```

00001
00062 #ifndef _VPARAM_H_
00063 #define _VPARAM_H_
00064
00065 /* Generic headers */
00066 #include "maloc/maloc.h"
00067 #include "apbs/vhal.h"
00068 #include "apbs/vunit.h"
00069 #include "apbs/vstring.h"
00070
00087 struct sVparam_AtomData {
00088     char atomName[VMAX_ARGLEN];
00089     char resName[VMAX_ARGLEN];
00090     double charge;

```

```

00091     double radius;
00092     double epsilon;
00094 };
00095
00101 typedef struct sVparam_AtomData Vparam_AtomData;
00102
00109 struct Vparam_ResData {
00110     Vmem *vmem;
00111     char name[VMAX_ARGLEN];
00112     int nAtomData;
00114     Vparam_AtomData *atomData;
00115 };
00116
00122 typedef struct Vparam_ResData Vparam_ResData;
00123
00130 struct Vparam {
00131     Vmem *vmem;
00133     int nResData;
00135     Vparam_ResData *resData;
00136 };
00137
00142 typedef struct Vparam Vparam;
00143
00144 /* //////////////////////////////// */
00145 // Class Vparam: Inlineable methods (vparam.c)
00147
00148 #if !defined(VINLINE_VPARAM)
00149
00156     VEXTERNC unsigned long int Vparam_memChk(Vparam *thee);
00157
00158 #else /* if defined(VINLINE_VPARAM) */
00159
00160 #    define Vparam_memChk(thee) (Vmem_bytes((thee)->vmem))
00161
00162 #endif /* if !defined(VINLINE_VPARAM) */
00163
00164 /* //////////////////////////////// */
00165 // Class Vparam: Non-Inlineable methods (vparam.c)
00167
00172 VEXTERNC Vparam_AtomData* Vparam_AtomData_ctor();
00173
00179 VEXTERNC int Vparam_AtomData_ctor2(Vparam_AtomData *thee);
00180
00185 VEXTERNC void Vparam_AtomData_dtor(Vparam_AtomData **thee);
00186
00191 VEXTERNC void Vparam_AtomData_dtor2(Vparam_AtomData *thee);
00192
00200 VEXTERNC void Vparam_AtomData_copyTo(Vparam_AtomData *thee,
00201     Vparam_AtomData *dest);
00202
00210 VEXTERNC void Vparam_ResData_copyTo(Vparam_ResData *thee,
00211     Vparam_ResData *dest);
00212
00220 VEXTERNC void Vparam_AtomData_copyFrom(Vparam_AtomData *thee,
00221     Vparam_AtomData *src);
00222
00228 VEXTERNC Vparam_ResData* Vparam_ResData_ctor(Vmem *mem);
00229
00236 VEXTERNC int Vparam_ResData_ctor2(Vparam_ResData *thee, Vmem *mem);
00237
00242 VEXTERNC void Vparam_ResData_dtor(Vparam_ResData **thee);
00243
00248 VEXTERNC void Vparam_ResData_dtor2(Vparam_ResData *thee);
00249
00254 VEXTERNC Vparam* Vparam_ctor();
00255
00261 VEXTERNC int Vparam_ctor2(Vparam *thee);
00262
00267 VEXTERNC void Vparam_dtor(Vparam **thee);
00268
00273 VEXTERNC void Vparam_dtor2(Vparam *thee);
00274
00285 VEXTERNC Vparam_ResData* Vparam_getResData(Vparam *thee,
00286     char resName[VMAX_ARGLEN]);
00287
00299 VEXTERNC Vparam_AtomData* Vparam_getAtomData(Vparam *thee,
00300     char resName[VMAX_ARGLEN], char atomName[VMAX_ARGLEN]);
00301
00330 VEXTERNC int Vparam_readFlatFile(Vparam *thee, const char *iodev,
00331     const char *iofmt, const char *thost, const char *fname);

```

```

00343 VEXTERNC int Vparam_readXMLFile(Vparam *thee, const char *iodev,
00344     const char *iofmt, const char *thost, const char *fname);
00345
00346 #endif /* ifndef _VPARAM_H_ */

```

9.47 src/generic/apbs/vpbe.h File Reference

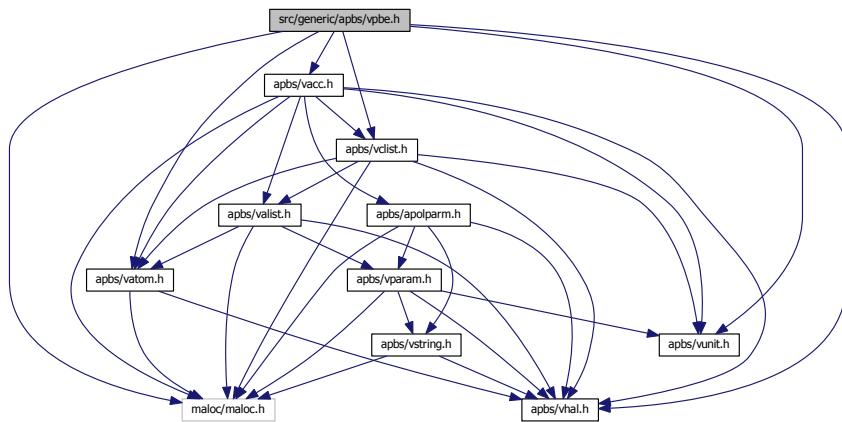
Contains declarations for class Vpbe.

```

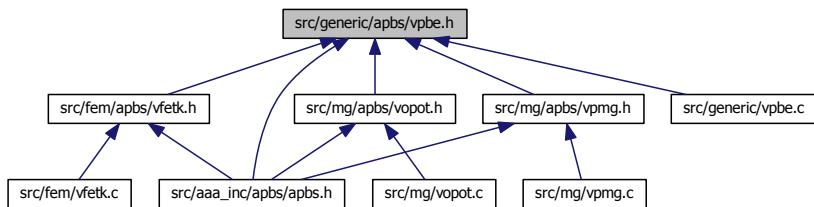
#include "maloc/malloc.h"
#include "apbs/vhal.h"
#include "apbs/vunit.h"
#include "apbs/vatom.h"
#include "apbs/vacc.h"
#include "apbs/vclist.h"

```

Include dependency graph for vpbe.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [Svpbe](#)

Contains public data members for Vpbe class/module.

Typedefs

- `typedef struct sVpbe Vpbe`

Declaration of the Vpbe class as the Vpbe structure.

Functions

- `VEXTERNC Valist * Vpbe_getValist (Vpbe *thee)`
Get atom list.
- `VEXTERNC Vacc * Vpbe_getVacc (Vpbe *thee)`
Get accessibility oracle.
- `VEXTERNC double Vpbe_getBulkIonicStrength (Vpbe *thee)`
Get bulk ionic strength.
- `VEXTERNC double Vpbe_getMaxIonRadius (Vpbe *thee)`
Get maximum radius of ion species.
- `VEXTERNC double Vpbe_getTemperature (Vpbe *thee)`
Get temperature.
- `VEXTERNC double Vpbe_getSoluteDiel (Vpbe *thee)`
Get solute dielectric constant.
- `VEXTERNC double Vpbe_getGamma (Vpbe *thee)`
Get apolar coefficient.
- `VEXTERNC double Vpbe_getSoluteRadius (Vpbe *thee)`
Get sphere radius which bounds biomolecule.
- `VEXTERNC double Vpbe_getSoluteXlen (Vpbe *thee)`
Get length of solute in x dimension.
- `VEXTERNC double Vpbe_getSoluteYlen (Vpbe *thee)`
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)`

- VEXTERNC double `Vpbe_getzmem (Vpbe *thee)`

Get charge scaling factor.
- VEXTERNC double `Vpbe_getLmem (Vpbe *thee)`

Get z position of the membrane bottom.
- VEXTERNC double `Vpbe_getmembraneDiel (Vpbe *thee)`

*Get length of the membrane (A)
aauthor Michael Grabe.*
- VEXTERNC double `Vpbe_getmemenv (Vpbe *thee)`

Get membrane dielectric constant.
- VEXTERNC double `Vpbe_getmemv (Vpbe *thee)`

Get membrane potential (kT)
- VEXTERNC `Vpbe * Vpbe_ctor (Valist *alist, int ionNum, double *ionConc, double *ionRadii, double *ionQ, double T, double soluteDiel, double solventDiel, double solventRadius, int focusFlag, double sdens, double z_mem, double L, double membraneDiel, double V)`

Construct Vpbe object.
- VEXTERNC int `Vpbe_ctor2 (Vpbe *thee, Valist *alist, int ionNum, double *ionConc, double *ionRadii, double *ionQ, double T, double soluteDiel, double solventDiel, double solventRadius, int focusFlag, double sdens, double z_mem, double L, double membraneDiel, double V)`

FORTRAN stub to construct Vpbe objct.
- VEXTERNC int `Vpbe_getIons (Vpbe *thee, int *nion, double ionConc[MAXION], double ionRadii[MAXION], double ionQ[MAXION])`

Get information about the counterion species present.
- VEXTERNC void `Vpbe_dtor (Vpbe **thee)`

Object destructor.
- VEXTERNC void `Vpbe_dtor2 (Vpbe *thee)`

FORTRAN stub object destructor.
- VEXTERNC double `Vpbe_getCoulombEnergy1 (Vpbe *thee)`

Calculate coulombic energy of set of charges.
- VEXTERNC unsigned long int `Vpbe_memChk (Vpbe *thee)`

Return the memory used by this structure (and its contents) in bytes.

9.47.1 Detailed Description

Contains declarations for class Vpbe.

Version

Id:

`vpbe.h` 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 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](#).

9.48 vpbe.h

```

00001
00066 #ifndef _VPBE_H_
00067 #define _VPBE_H_
00068
00069 /* Generic headers */
00070 #include "maloc/maloc.h"
00071 #include "apbs/vhal.h"
00072
00073 /* Specific headers */
00074 #include "apbs/vunit.h"
00075 #include "apbs/vatom.h"
00076 #include "apbs/vacc.h"
00077 #include "apbs/vclist.h"
00078

```

```

00084 struct sVpbe {
00085     Vmem *vmem;
00086     Valist *alist;
00087     Vclist *clist;
00088     Vacc *acc;
00089     double T;
00090     double soluteDiel;
00091     double solventDiel;
00092     double solventRadius;
00093     double bulkIonicStrength;
00094     double maxIonRadius;
00095     int numIon;
00096     double ionConc[MAXION];
00097     double ionRadii[MAXION];
00098     double ionQ[MAXION];
00099     double xkappa;
00100    double deblen;
00101    double zkappa2;
00102    double zmagic;
00103    double soluteCenter[3];
00104    double soluteRadius;
00105    double soluteXlen;
00106    double soluteYlen;
00107    double soluteZlen;
00108    double soluteCharge;
00109    double smvolume;
00110    double smsize;
00111    int ipkey;
00112    int paramFlag;
00113    /*-----*/
00114    /* Added by Michael Grabe */
00115    /*-----*/
00116    double z_mem;
00117    double L;
00118    double membraneDiel;
00119    double V;
00120    int param2Flag;
00121    /*-----*/
00122    };
00123
00124 typedef struct sVpbe Vpbe;
00125
00126 /* //////////////////////////////// */
00127 // Class Vpbe: Inlineable methods (vpbe.c)
00128
00129 #if !defined(VINLINE_VPBE)
00130
00131 VEXTERNC Valist* Vpbe_getValist(Vpbe *thee);
00132
00133 VEXTERNC Vacc* Vpbe_getVacc(Vpbe *thee);
00134
00135 VEXTERNC double Vpbe_getBulkIonicStrength(Vpbe *thee);
00136
00137 VEXTERNC double Vpbe_getMaxIonRadius(Vpbe *thee);
00138
00139 VEXTERNC double Vpbe_getTemperature(Vpbe *thee);
00140
00141 VEXTERNC double Vpbe_getSoluteDiel(Vpbe *thee);
00142
00143 VEXTERNC double Vpbe_getGamma(Vpbe *thee);
00144
00145 VEXTERNC double Vpbe_getSoluteRadius(Vpbe *thee);
00146
00147 VEXTERNC double Vpbe_getSoluteXlen(Vpbe *thee);
00148
00149 VEXTERNC double Vpbe_getSoluteYlen(Vpbe *thee);
00150
00151 VEXTERNC double Vpbe_getSoluteZlen(Vpbe *thee);
00152
00153 VEXTERNC double* Vpbe_getSoluteCenter(Vpbe *thee);
00154
00155 VEXTERNC double Vpbe_getSoluteCharge(Vpbe *thee);
00156
00157 VEXTERNC double Vpbe_getSolventDiel(Vpbe *thee);
00158
00159 VEXTERNC double Vpbe_getSolventRadius(Vpbe *thee);
00160
00161 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_ */

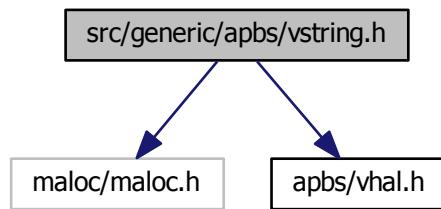
```

9.49 src/generic/apbs/vstring.h File Reference

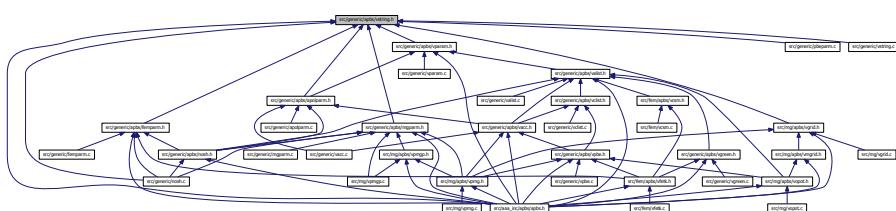
Contains declarations for class Vstring.

```
#include "maloc/malloc.h"
#include "apbs/vhal.h"
```

Include dependency graph for vstring.h:



This graph shows which files directly or indirectly include this file:



Functions

- VEXTERNC int [Vstring_strcasecmp](#) (const char *s1, const char *s2)
Case-insensitive string comparison (BSD standard)
- VEXTERNC int [Vstring_isdigit](#) (const char *tok)
A modified sscanf that examines the complete string.
- VEXTERNC char * [Vstring_wrappedtext](#) (const char *str, int right_margin, int left_padding)

9.49.1 Detailed Description

Contains declarations for class Vstring.

Version

Id:

[vstring.h](#) 1750 2012-07-18 18:34:27Z tuckerbeck

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 [vstring.h](#).

9.50 vstring.h

```

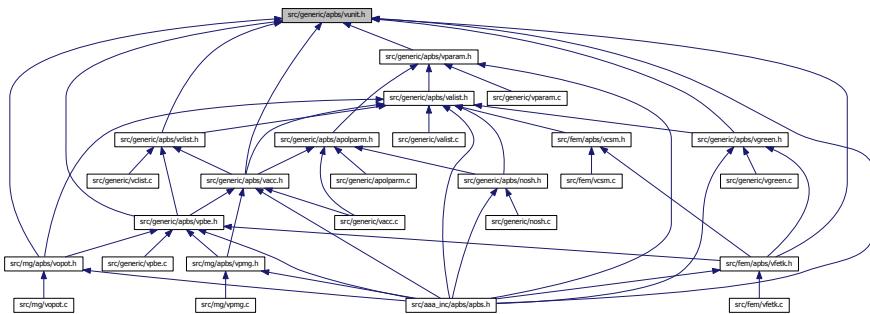
00001
00078 #ifndef _VSTRING_H_
00079 #define _VSTRING_H_
00080
00081 #include "malloc/malloc.h"
00082 #include "apbs/vhal.h"
00083
00096 VEXTERNC int Vstring_strcasecmp(const char *s1, const char *
00097           s2);
00098
00104 VEXTERNC int Vstring_isdigit(const char *tok);
00105
00111 VEXTERNC char* Vstring_wrappedtext(
00112           const char* str,
00113           int right_margin,
00114           int left_padding
00115           );
00116
00117 #endif /* ifndef _VSTRING_H_ */

```

9.51 src/generic/apbs/vunit.h File Reference

Contains a collection of useful constants and conversion factors.

This graph shows which files directly or indirectly include this file:



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
 $e_c^2 / \text{in ESU units} \Rightarrow \text{kcal/mol}$
 - #define [Vunit_esu_kb](#) 1.9871913e-03
 $k_b \text{ in ESU units} \Rightarrow \text{kcal/mol}$

9.51.1 Detailed Description

Contains a collection of useful constants and conversion factors.

Author

Nathan Baker
Nathan A. Baker

Version

Id:

[vunit.h](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2012 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](#).

9.52 vunit.h

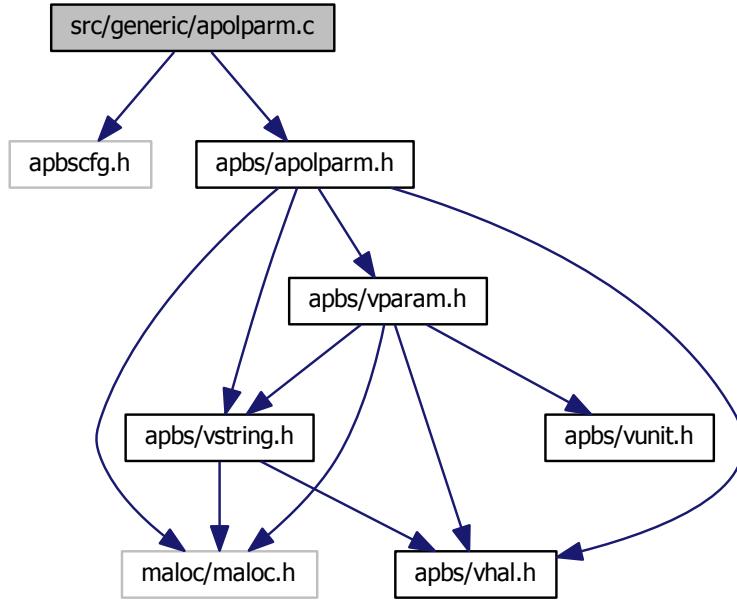
```
00001
00063 #ifndef _VUNIT_H_
00064 #define _VUNIT_H_
00065
00068 #define Vunit_J_to_cal  4.1840000e+00
00069
00072 #define Vunit_cal_to_J  2.3900574e-01
00073
00076 #define Vunit_amu_to_kg  1.6605402e-27
00077
00080 #define Vunit_kg_to_amu  6.0221367e+26
00081
00084 #define Vunit_ec_to_C   1.6021773e-19
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_ */
```

9.53 src/generic/apolparm.c File Reference

Class APOLparm methods.

```
#include "apbscfg.h"
#include "apbs/apolparm.h"
```

Include dependency graph for apolparm.c:



Functions

- VPUBLIC [APOLparm](#) * [APOLparm_ctor](#) ()

Construct APOLparm.
- VPUBLIC Vrc_Codes [APOLparm_ctor2](#) ([APOLparm](#) *thee)

FORTRAN stub to construct APOLparm.
- VPUBLIC void [APOLparm_copy](#) ([APOLparm](#) *thee, [APOLparm](#) *source)

Copy target object into thee.
- VPUBLIC void [APOLparm_dtor](#) ([APOLparm](#) **thee)

Object destructor.
- VPUBLIC void [APOLparm_dtor2](#) ([APOLparm](#) *thee)

FORTRAN stub for object destructor.
- VPUBLIC Vrc_Codes [APOLparm_check](#) ([APOLparm](#) *thee)

Consistency check for parameter values stored in object.
- VPRIVATE Vrc_Codes [APOLparm_parseGRID](#) ([APOLparm](#) *thee, Vio *sock)
- VPRIVATE Vrc_Codes [APOLparm_parseMOL](#) ([APOLparm](#) *thee, Vio *sock)
- VPRIVATE Vrc_Codes [APOLparm_parseSRFM](#) ([APOLparm](#) *thee, Vio *sock)
- VPRIVATE Vrc_Codes [APOLparm_parseSRAD](#) ([APOLparm](#) *thee, Vio *sock)
- VPRIVATE Vrc_Codes [APOLparm_parseSWIN](#) ([APOLparm](#) *thee, Vio *sock)
- VPRIVATE Vrc_Codes [APOLparm_parseTEMP](#) ([APOLparm](#) *thee, Vio *sock)
- VPRIVATE Vrc_Codes [APOLparm_parseGAMMA](#) ([APOLparm](#) *thee, Vio *sock)
- VPRIVATE Vrc_Codes [APOLparm_parseCALCENERGY](#) ([APOLparm](#) *thee, Vio *sock)

- VPRIVATE Vrc_Codes **APOLparm_parseCALCFORCE** (**APOLparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **APOLparm_parseBCONC** (**APOLparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **APOLparm_parseSDENS** (**APOLparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **APOLparm_parseDPOS** (**APOLparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **APOLparm_parsePRESS** (**APOLparm** *thee, Vio *sock)
- VPUBLIC Vrc_Codes **APOLparm_parseToken** (**APOLparm** *thee, char tok[VMAX_BUFSIZE], Vio *sock)

Parse an MG keyword from an input file.

9.53.1 Detailed Description

Class APOLparm methods.

Author

David Gohara

Version

Id:

[apolparm.c](#) 1750 2012-07-18 18:34:27Z tuckerbeck

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](#).

9.54 apolparm.c

```

00001
00058 #include "apbscfg.h"
00059 #include "apbs/apolparm.h"
00060
00061 VEMBED(rcsid="$Id: apolparm.c 1750 2012-07-18 18:34:27Z tuckerbeck $" )
00062
00063 #if !defined(VINLINE_MGPARM)
00064
00065 #endif /* if !defined(VINLINE_MGPARM) */
00066
00067 VPUBLIC APOLparm* APOLparm_ctor() {
00068
00069     /* Set up the structure */
00070     APOLparm *thee = VNULL;
00071     thee = (APOLparm*)Vmem_malloc(VNULL, 1, sizeof(APOLparm));
00072     VASSERT( thee != VNULL );
00073     VASSERT( APOLparm_ctor2(thee) == VRC_SUCCESS );
00074
00075     return thee;
00076 }
00077
00078 VPUBLIC Vrc_Codes APOLparm_ctor2(APOLparm *thee) {
00079
00080     int i;
00081
00082     if (thee == VNULL) return VRC_FAILURE;
00083
00084     thee->parsed = 0;
00085
00086     thee->setgrid = 0;
00087     thee->setmolid = 0;
00088     thee->setbconc = 0;
00089     thee->setsdens = 0;
00090     thee->setdpos = 0;
00091     thee->setpress = 0;
00092     thee->setsrfm = 0;
00093     thee->setsrad = 0;
00094     thee->setswin = 0;
00095
00096     thee->settemp = 0;
00097     thee->setgamma = 0;
00098
00099     thee->setwat = 0;
00100
00101     thee->sav = 0.0;
00102     thee->sasa = 0.0;
00103     thee->wcaEnergy = 0.0;
00104
00105     for(i=0;i<3;i++) thee->totForce[i] = 0.0;
00106
00107     return VRC_SUCCESS;
00108 }
00109
00110 VPUBLIC void APOLparm_copy(
00111     APOLparm *thee,
00112     APOLparm *source
00113 ) {

```

```

00114
00115     int i;
00116
00117     thee->parsed = source->parsed;
00118
00119     for (i=0; i<3; i++) thee->grid[i] = source->grid[i];
00120     thee->setgrid = source->setgrid;
00121
00122     thee->molid = source->molid;
00123     thee->setmolid = source->setmolid;
00124
00125     thee->bconc = source->bconc ;
00126     thee->setbconc= source->setbconc ;
00127
00128     thee->sdens = source->sdens ;
00129     thee->setsdens= source->setsdens ;
00130
00131     thee->dpos = source->dpos ;
00132     thee->setdpos= source->setdpos ;
00133
00134     thee->press = source->press ;
00135     thee->setpress = source->setpress ;
00136
00137     thee->srfm = source->srfm ;
00138     thee->setsrfm = source->setsrfm ;
00139
00140     thee->srad = source->srad ;
00141     thee->setsrad = source->setsrad ;
00142
00143     thee->swin = source->swin ;
00144     thee->setswin = source->setswin ;
00145
00146     thee->temp = source->temp ;
00147     thee->settemp = source->settemp ;
00148
00149     thee->gamma = source->gamma ;
00150     thee->setgamma = source->setgamma ;
00151
00152     thee->calcenergy = source->calcenergy ;
00153     thee->setcalcenergy = source->setcalcenergy ;
00154
00155     thee->calccforce = source->calccforce ;
00156     thee->setcalccforce = source->setcalccforce ;
00157
00158     thee->setwat = source->setwat ;
00159
00160     thee->sav = source->sav;
00161     thee->sasa = source->sasa;
00162     thee->wcaEnergy = source->wcaEnergy;
00163
00164     for(i=0;i<3;i++) thee->totForce[i] = source->totForce[i];
00165
00166     return;
00167 }
00168
00169 VPUBLIC void APOLparm_dtor(APOLparm **thee) {
00170     if ((*thee) != VNULL) {
00171         APOLparm_dtor2(*thee);
00172         Vmem_free(VNULL, 1, sizeof(APOLparm), (void **)thee);
00173         (*thee) = VNULL;
00174     }
00175
00176     return;
00177 }
00178
00179 VPUBLIC void APOLparm_dtor2(APOLparm *thee) { ; }
00180
00181 VPUBLIC Vrc_Codes APOLparm_check(APOLparm *thee) {
00182
00183
00184     Vrc_Codes rc;
00185     rc = VRC_SUCCESS;
00186
00187     if (!thee->parsed) {
00188         Vnm_print(2, "APOLparm_check: not filled!\n");
00189         return VRC_FAILURE;
00190     }
00191     if (!thee->setgrid) {
00192         Vnm_print(2, "APOLparm_check: grid not set!\n");
00193         rc = VRC_FAILURE;
00194     }

```

```

00195     if (!thee->setmolid) {
00196         Vnm_print(2, "APOLparm_check: molid not set!\n");
00197         rc = VRC_FAILURE;
00198     }
00199     if (!thee->setbconc) {
00200         Vnm_print(2, "APOLparm_check: bconc not set!\n");
00201         rc = VRC_FAILURE;
00202     }
00203     if (!thee->setsdens) {
00204         Vnm_print(2, "APOLparm_check: sdens not set!\n");
00205         rc = VRC_FAILURE;
00206     }
00207     if (!thee->setdpos) {
00208         Vnm_print(2, "APOLparm_check: dpos not set!\n");
00209         rc = VRC_FAILURE;
00210     }
00211     if (!thee->setpress) {
00212         Vnm_print(2, "APOLparm_check: press not set!\n");
00213         rc = VRC_FAILURE;
00214     }
00215     if (!thee->setsrfm) {
00216         Vnm_print(2, "APOLparm_check: srfm not set!\n");
00217         rc = VRC_FAILURE;
00218     }
00219     if (!thee->setsrad) {
00220         Vnm_print(2, "APOLparm_check: srad not set!\n");
00221         rc = VRC_FAILURE;
00222     }
00223     if (!thee->setswin) {
00224         Vnm_print(2, "APOLparm_check: swin not set!\n");
00225         rc = VRC_FAILURE;
00226     }
00227     if (!thee->settemp) {
00228         Vnm_print(2, "APOLparm_check: temp not set!\n");
00229         rc = VRC_FAILURE;
00230     }
00231     if (!thee->setgamma) {
00232         Vnm_print(2, "APOLparm_check: gamma not set!\n");
00233         rc = VRC_FAILURE;
00234     }
00235     return rc;
00236 }
00237 }
00238
00239 VPRIVATE Vrc_Codes APOLparm_parseGRID(APOLparm *thee, Vio *sock) {
00240
00241     char tok[VMAX_BUFSIZE];
00242     double tf;
00243
00244     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00245     if (sscanf(tok, "%lf", &tf) == 0) {
00246         Vnm_print(2, "Nosh: Read non-float (%s) while parsing GRID \
00247 keyword!\n", tok);
00248         return VRC_WARNING;
00249     } else thee->grid[0] = tf;
00250     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00251     if (sscanf(tok, "%lf", &tf) == 0) {
00252         Vnm_print(2, "Nosh: Read non-float (%s) while parsing GRID \
00253 keyword!\n", tok);
00254         return VRC_WARNING;
00255     } else thee->grid[1] = tf;
00256     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00257     if (sscanf(tok, "%lf", &tf) == 0) {
00258         Vnm_print(2, "Nosh: Read non-float (%s) while parsing GRID \
00259 keyword!\n", tok);
00260         return VRC_WARNING;
00261     } else thee->grid[2] = tf;
00262     thee->setgrid = 1;
00263     return VRC_SUCCESS;
00264 }
00265 VERROR1:
00266     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00267     return VRC_WARNING;
00268 }
00269
00270 VPRIVATE Vrc_Codes APOLparm_parseMOL(APOLparm *thee, Vio *sock) {
00271     int ti;
00272     char tok[VMAX_BUFSIZE];
00273
00274     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00275     if (sscanf(tok, "%d", &ti) == 0) {

```

```

00276         Vnm_print(2, "NOsh: Read non-int (%s) while parsing MOL \
00277 keyword!\n", tok);
00278         return VRC_WARNING;
00279     }
00280     thee->molid = ti;
00281     thee->setmolid = 1;
00282     return VRC_SUCCESS;
00283
00284 VERROR1:
00285     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00286     return VRC_WARNING;
00287 }
00288
00289 VPRIIVATE Vrc_Codes APOLparm_parseSRFM(APOLparm *thee, Vio *sock) {
00290     char tok[VMAX_BUFSIZE];
00291
00292     VJMPERR1(Vio_scant(sock, "%s", tok) == 1);
00293
00294     if (Vstring_strcasecmp(tok, "sacc") == 0) {
00295         thee->srfm = VSM_MOL;
00296         thee->setsrfm = 1;
00297         return VRC_SUCCESS;
00298     } else {
00299         printf("parseAPOL: Unrecognized keyword (%s) when parsing srfm!\n",
00300             tok);
00300     printf("parseAPOL: Accepted values for srfm = sacc\n");
00301     return VRC_WARNING;
00302 }
00303
00304     return VRC_FAILURE;
00305
00306 VERROR1:
00307     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00308     return VRC_WARNING;
00309 }
00310
00311 VPRIIVATE Vrc_Codes APOLparm_parseSRAD(APOLparm *thee, Vio *sock) {
00312     char tok[VMAX_BUFSIZE];
00313     double tf;
00314
00315     VJMPERR1(Vio_scant(sock, "%s", tok) == 1);
00316     if (sscanf(tok, "%lf", &tf) == 0) {
00317         Vnm_print(2, "NOsh: Read non-float (%s) while parsing SRAD \
00318 keyword!\n", tok);
00319         return VRC_WARNING;
00320     }
00321     thee->srad = tf;
00322     thee->setsrad = 1;
00323     return VRC_SUCCESS;
00324
00325 VERROR1:
00326     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00327     return VRC_WARNING;
00328 }
00329
00330 VPRIIVATE Vrc_Codes APOLparm_parseSWIN(APOLparm *thee, Vio *sock) {
00331     char tok[VMAX_BUFSIZE];
00332     double tf;
00333
00334     VJMPERR1(Vio_scant(sock, "%s", tok) == 1);
00335     if (sscanf(tok, "%lf", &tf) == 0) {
00336         Vnm_print(2, "NOsh: Read non-float (%s) while parsing SWIN \
00337 keyword!\n", tok);
00338         return VRC_WARNING;
00339     }
00340     thee->swin = tf;
00341     thee->setswin = 1;
00342     return VRC_SUCCESS;
00343
00344 VERROR1:
00345     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00346     return VRC_WARNING;
00347 }
00348
00349 VPRIIVATE Vrc_Codes APOLparm_parseTEMP(APOLparm *thee, Vio *sock) {
00350     char tok[VMAX_BUFSIZE];
00351     double tf;
00352
00353     VJMPERR1(Vio_scant(sock, "%s", tok) == 1);
00354     if (sscanf(tok, "%lf", &tf) == 0) {
00355         Vnm_print(2, "NOsh: Read non-float (%s) while parsing TEMP \

```

```

00356 keyword!\n", tok);
00357     return VRC_WARNING;
00358 }
00359 thee->temp = tf;
00360 thee->settemp = 1;
00361 return VRC_SUCCESS;
00362
00363 VERROR1:
00364     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00365     return VRC_WARNING;
00366 }
00367
00368 VPRIPRIVATE Vrc_Codes APOLparm_parseGAMMA(APOLparm *thee, Vio *sock) {
00369     char tok[VMAX_BUFSIZE];
00370     double tf;
00371
00372     VJMPERR1(Vio_scantf(sock, "%s", tok) == 1);
00373     if (sscanf(tok, "%lf", &tf) == 0) {
00374         Vnm_print(2, "Nosh: Read non-float (%s) while parsing GAMMA \
00375 keyword!\n", tok);
00376         return VRC_WARNING;
00377     }
00378     thee->gamma = tf;
00379     thee->setgamma = 1;
00380     return VRC_SUCCESS;
00381
00382 VERROR1:
00383     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00384     return VRC_WARNING;
00385 }
00386
00387 VPRIPRIVATE Vrc_Codes APOLparm_parseCALCENERGY(APOLparm *thee, Vio *sock)
00388 {
00389     char tok[VMAX_BUFSIZE];
00390     int ti;
00391
00392     VJMPERR1(Vio_scantf(sock, "%s", tok) == 1);
00393     /* Parse number */
00394     if (sscanf(tok, "%d", &ti) == 1) {
00395         thee->calcencegy = (APOLparm_calcEnergy)ti
00396 ;
00397         thee->setcalcencegy = 1;
00398
00399         Vnm_print(2, "parseAPOL: Warning -- parsed deprecated \"calcenergy \
00400 %d\" statement.\n", ti);
00401         Vnm_print(2, "parseAPOL: Please use \"calcenergy \"");
00402         switch (thee->calcencegy) {
00403             case ACE_NO:
00404                 Vnm_print(2, "no");
00405                 break;
00406             case ACE_TOTAL:
00407                 Vnm_print(2, "total");
00408                 break;
00409             case ACE_COMPS:
00410                 Vnm_print(2, "comps");
00411                 break;
00412             default:
00413                 Vnm_print(2, "UNKNOWN");
00414                 break;
00415         }
00416         Vnm_print(2, "\" instead.\n");
00417         return VRC_SUCCESS;
00418     } else if (Vstring_strcasecmp(tok, "no") == 0) {
00419         thee->calcencegy = ACE_NO;
00420         thee->setcalcencegy = 1;
00421         return VRC_SUCCESS;
00422     } else if (Vstring_strcasecmp(tok, "total") == 0) {
00423         thee->calcencegy = ACE_TOTAL;
00424         thee->setcalcencegy = 1;
00425         return VRC_SUCCESS;
00426     } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00427         thee->calcencegy = ACE_COMPS;
00428         thee->setcalcencegy = 1;
00429         return VRC_SUCCESS;
00430     } else {
00431         Vnm_print(2, "Nosh: Unrecognized parameter (%s) while parsing \
00432 calcenergy!\n", tok);
00433         return VRC_WARNING;
00434     }
00435     return VRC_FAILURE;
00436

```

```

00435 VERROR1:
00436     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00437     return VRC_WARNING;
00438 }
00439
00440 VPRIVATE Vrc_Codes APOLparm_parseCALCFORCE(APOLparm *thee, Vio *sock) {
00441     char tok[VMAX_BUFSIZE];
00442     int ti;
00443
00444     VJMPERR1(Vio_scantok(sock, "%s", tok) == 1);
00445     /* Parse number */
00446     if (sscanf(tok, "%d", &ti) == 1) {
00447         thee->calcforce = (APOLparm_calcForce)ti;
00448         thee->setcalcforce = 1;
00449
00450         Vnm_print(2, "parseAPOL: Warning -- parsed deprecated \"calcforce \
00451 %d\" statement.\n", ti);
00452         Vnm_print(2, "parseAPOL: Please use \"calcforce \"");
00453         switch (thee->calcenergy) {
00454             case ACF_NO:
00455                 Vnm_print(2, "no");
00456                 break;
00457             case ACF_TOTAL:
00458                 Vnm_print(2, "total");
00459                 break;
00460             case ACF_COMPS:
00461                 Vnm_print(2, "comps");
00462                 break;
00463             default:
00464                 Vnm_print(2, "UNKNOWN");
00465                 break;
00466         }
00467         Vnm_print(2, "\ instead.\n");
00468         return VRC_SUCCESS;
00469     } else if (Vstring_strcasecmp(tok, "no") == 0) {
00470         thee->calcforce = ACF_NO;
00471         thee->setcalcforce = 1;
00472         return VRC_SUCCESS;
00473     } else if (Vstring_strcasecmp(tok, "total") == 0) {
00474         thee->calcforce = ACF_TOTAL;
00475         thee->setcalcforce = 1;
00476         return VRC_SUCCESS;
00477     } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00478         thee->calcforce = ACF_COMPS;
00479         thee->setcalcforce = 1;
00480         return VRC_SUCCESS;
00481     } else {
00482         Vnm_print(2, "NOsh: Unrecognized parameter (%s) while parsing \
00483 calcforce!\n", tok);
00484         return VRC_WARNING;
00485     }
00486     return VRC_FAILURE;
00487 }
00488 VERROR1:
00489     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00490     return VRC_WARNING;
00491 }
00492
00493 VPRIVATE Vrc_Codes APOLparm_parseBCONC(APOLparm *thee, Vio *sock) {
00494     char tok[VMAX_BUFSIZE];
00495     double tf;
00496
00497     VJMPERR1(Vio_scantok(sock, "%s", tok) == 1);
00498     if (sscanf(tok, "%lf", &tf) == 0) {
00499         Vnm_print(2, "NOsh: Read non-float (%s) while parsing BCONC \
00500 keyword!\n", tok);
00501         return VRC_WARNING;
00502     }
00503     thee->bconc = tf;
00504     thee->setbconc = 1;
00505     return VRC_SUCCESS;
00506 }
00507 VERROR1:
00508     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00509     return VRC_WARNING;
00510 }
00511
00512 VPRIVATE Vrc_Codes APOLparm_parseSDENS(APOLparm *thee, Vio *sock) {
00513     char tok[VMAX_BUFSIZE];
00514     double tf;
00515

```

```

00516     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00517     if (sscanf(tok, "%lf", &tf) == 0) {
00518         Vnm_print(2, "NOsh: Read non-float (%s) while parsing SDENS \
00519 keyword!\n", tok);
00520         return VRC_WARNING;
00521     }
00522     thee->sdens = tf;
00523     thee->setsdens = 1;
00524     return VRC_SUCCESS;
00525
00526 VERROR1:
00527     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00528     return VRC_WARNING;
00529 }
00530
00531 VPRIPRIVATE Vrc_Codes APOLparm_parseDPOS(APOLparm *thee, Vio *sock) {
00532     char tok[VMAX_BUFSIZE];
00533     double tf;
00534
00535     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00536     if (sscanf(tok, "%lf", &tf) == 0) {
00537         Vnm_print(2, "NOsh: Read non-float (%s) while parsing SDENS \
00538 keyword!\n", tok);
00539         return VRC_WARNING;
00540     }
00541     thee->dpos = tf;
00542     thee->setdpos = 1;
00543
00544     if (thee->dpos < 0.001){
00545         Vnm_print(1,"\\nWARNING WARNING WARNING WARNING WARNING\\n");
00546         Vnm_print(1,"NOsh: dpos is set to a very small value.\n");
00547         Vnm_print(1,"NOsh: If you are not using a PQR file, you can \
00548 safely ignore this message.\n");
00549         Vnm_print(1,"NOsh: Otherwise please choose a value greater than \
00550 or equal to 0.001.\n\\n");
00551     }
00552
00553     return VRC_SUCCESS;
00554
00555 VERROR1:
00556     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00557     return VRC_WARNING;
00558 }
00559
00560 VPRIPRIVATE Vrc_Codes APOLparm_parsePRESS(APOLparm *thee, Vio *sock) {
00561     char tok[VMAX_BUFSIZE];
00562     double tf;
00563
00564     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00565     if (sscanf(tok, "%lf", &tf) == 0) {
00566         Vnm_print(2, "NOsh: Read non-float (%s) while parsing PRESS \
00567 keyword!\n", tok);
00568         return VRC_WARNING;
00569     }
00570     thee->press = tf;
00571     thee->setpress = 1;
00572     return VRC_SUCCESS;
00573
00574 VERROR1:
00575     Vnm_print(2, "parseAPOL: ran out of tokens!\n");
00576     return VRC_WARNING;
00577 }
00578
00579 VPUBLIC Vrc_Codes APOLparm_parseToken(APOLparm *thee
00580 , char tok[VMAX_BUFSIZE],
00581 Vio *sock) {
00582
00583     if (thee == VNULL) {
00584         Vnm_print(2, "parseAPOL: got NULL thee!\n");
00585         return VRC_WARNING;
00586     }
00587
00588     if (sock == VNULL) {
00589         Vnm_print(2, "parseAPOL: got NULL socket!\n");
00590         return VRC_WARNING;
00591     }
00592     if (Vstring_strcasecmp(tok, "mol") == 0) {
00593         return APOLparm_parseMOL(thee, sock);
00594     } else if (Vstring_strcasecmp(tok, "grid") == 0) {
00595         return APOLparm_parseGRID(thee, sock);

```

```

00596 } else if (Vstring_strcmp(tok, "dime") == 0) {
00597 Vnm_print(2, "APOLparm_parseToken: The DIME and GLEN keywords for APOLAR
have been replaced with GRID.\n");
00598 Vnm_print(2, "APOLparm_parseToken: Please see the APBS User Guide for more
information.\n");
00599 return VRC_WARNING;
00600 } else if (Vstring_strcmp(tok, "glen") == 0) {
00601 Vnm_print(2, "APOLparm_parseToken: The DIME and GLEN keywords for APOLAR
have been replaced with GRID.\n");
00602 Vnm_print(2, "APOLparm_parseToken: Please see the APBS User Guide for more
information.\n");
00603 return VRC_WARNING;
00604 } else if (Vstring_strcmp(tok, "bconc") == 0) {
00605 return APOLparm_parseBCONC(thee, sock);
00606 } else if (Vstring_strcmp(tok, "sdens") == 0) {
00607 return APOLparm_parseSDENS(thee, sock);
00608 } else if (Vstring_strcmp(tok, "dpos") == 0) {
00609 return APOLparm_parseDPOS(thee, sock);
00610 } else if (Vstring_strcmp(tok, "srfm") == 0) {
00611 return APOLparm_parseSRFM(thee, sock);
00612 } else if (Vstring_strcmp(tok, "srad") == 0) {
00613 return APOLparm_parseSRAD(thee, sock);
00614 } else if (Vstring_strcmp(tok, "swin") == 0) {
00615 return APOLparm_parseSWIN(thee, sock);
00616 } else if (Vstring_strcmp(tok, "temp") == 0) {
00617 return APOLparm_parseTEMP(thee, sock);
00618 } else if (Vstring_strcmp(tok, "gamma") == 0) {
00619 return APOLparm_parseGAMMA(thee, sock);
00620 } else if (Vstring_strcmp(tok, "press") == 0) {
00621 return APOLparm_parsePRESS(thee, sock);
00622 } else if (Vstring_strcmp(tok, "calcenergy") == 0) {
00623 return APOLparm_parseCALCENERGY(thee, sock);
00624 } else if (Vstring_strcmp(tok, "calcforce") == 0) {
00625 return APOLparm_parseCALCFORCE(thee, sock);
00626 } else {
00627 Vnm_print(2, "parseAPOL: Unrecognized keyword (%s)!\n", tok);
00628 return VRC_WARNING;
00629 }
00630
00631
00632 return VRC_FAILURE;
00633
00634 }

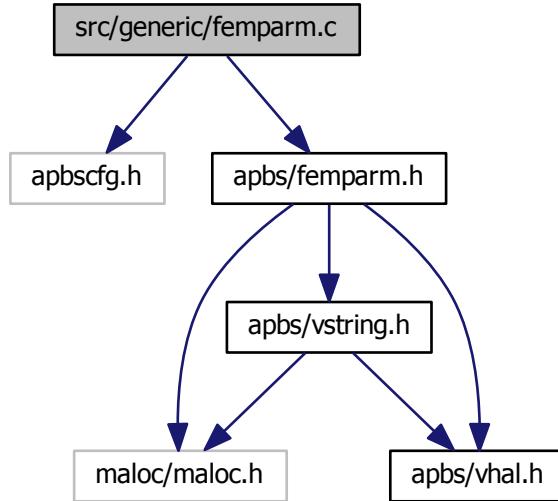
```

9.55 src/generic/femparm.c File Reference

Class FEMparm methods.

```
#include "apbscfg.h"
#include "apbs/femparm.h"
```

Include dependency graph for femparm.c:



Functions

- VPUBLIC **FEMparm** * **FEMparm_ctor** (**FEMparm_CalcType** type)
Construct FEMparm.
- VPUBLIC int **FEMparm_ctor2** (**FEMparm** *thee, **FEMparm_CalcType** type)
FORTRAN stub to construct FEMparm.
- VPUBLIC void **FEMparm_copy** (**FEMparm** *thee, **FEMparm** *source)
Copy target object into thee.
- VPUBLIC void **FEMparm_dtor** (**FEMparm** **thee)
Object destructor.
- VPUBLIC void **FEMparm_dtor2** (**FEMparm** *thee)
FORTRAN stub for object destructor.
- VPUBLIC int **FEMparm_check** (**FEMparm** *thee)
Consistency check for parameter values stored in object.
- VPRIVATE Vrc_Codes **FEMparm_parseDOMAINLENGTH** (**FEMparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **FEMparm_parseETOL** (**FEMparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **FEMparm_parseEKEY** (**FEMparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **FEMparm_parseAKEYPRE** (**FEMparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **FEMparm_parseAKEYSOLVE** (**FEMparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **FEMparm_parseTARGETNUM** (**FEMparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **FEMparm_parseTARGETRES** (**FEMparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **FEMparm_parseMAXSOLVE** (**FEMparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **FEMparm_parseMAXVERT** (**FEMparm** *thee, Vio *sock)
- VPRIVATE Vrc_Codes **FEMparm_parseUSEMESH** (**FEMparm** *thee, Vio *sock)
- VPUBLIC Vrc_Codes **FEMparm_parseToken** (**FEMparm** *thee, char tok[VMAX_BUFSIZE], Vio *sock)
Parse an MG keyword from an input file.

9.55.1 Detailed Description

Class FEMparm methods.

Author

Nathan Baker

Version

Id:

[femparm.c](#) 1750 2012-07-18 18:34:27Z tuckerbeck

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](#).

9.56 femparm.c

```

00001
00058 #include "apbscfg.h"
00059 #include "apbs/femparm.h"
00060
00061 VEMBED(rcsid="$Id: femparm.c 1750 2012-07-18 18:34:27Z tuckerbeck $")
00062
00063 #if !defined(VINLINE_MGPARM)
00064
00065 #endif /* if !defined(VINLINE_MGPARM) */
00066
00067 VPUBLIC FEMparm* FEMparm_ctor(FEMparm_CalcType
00068 type) {
00069     /* Set up the structure */
00070     FEMparm *thee = VNULL;
00071     thee = (FEMparm*)Vmem_malloc(VNULL, 1, sizeof(FEMparm));
00072     VASSERT( thee != VNULL );
00073     VASSERT( FEMparm_ctor2(thee, type) );
00074
00075     return thee;
00076 }
00077
00078 VPUBLIC int FEMparm_ctor2(FEMparm *thee,
00079                             FEMparm_CalcType type
00080                             ) {
00081
00082     if (thee == VNULL) return 0;
00083
00084     thee->parsed = 0;
00085     thee->type = type;
00086     thee->settotype = 1;
00087
00088     thee->setglen = 0;
00089     thee->setetol = 0;
00090     thee->setekey = 0;
00091     thee->setakeyPRE = 0;
00092     thee->setakeySOLVE = 0;
00093     thee->settargtNum = 0;
00094     thee->settargtRes = 0;
00095     thee->setmaxsolve = 0;
00096     thee->setmaxvert = 0;
00097     thee->useMesh = 0;
00098
00099     return 1;
00100 }
00101
00102 VPUBLIC void FEMparm_copy(
00103     FEMparm *thee,
00104     FEMparm *source
00105     ) {
00106
00107     int i;
00108
00109     thee->parsed = source->parsed;
00110     thee->type = source->type;
00111     thee->settotype = source->settotype;
00112     for (i=0; i<3; i++) thee->glen[i] = source->glen[i];
00113     thee->setglen = source->setglen;
00114     thee->setol = source->setol;
00115     thee->setetol = source->setetol;
00116     thee->ekey = source->ekey;
00117     thee->setekey = source->setekey;
00118     thee->akeyPRE = source->akeyPRE;
00119     thee->setakeyPRE = source->setakeyPRE;
00120     thee->akeySOLVE = source->akeySOLVE;
00121     thee->setakeySOLVE = source->setakeySOLVE;
00122     thee->targtNum = source->targtNum;
00123     thee->settargtNum = source->settargtNum;
00124     thee->targtRes = source->targtRes;
00125     thee->settargtRes = source->settargtRes;
00126     thee->maxsolve = source->maxsolve;
00127     thee->setmaxsolve = source->setmaxsolve;
00128     thee->maxvert = source->maxvert;
00129     thee->setmaxvert = source->setmaxvert;
00130     thee->pkey = source->pkey;
00131     thee->useMesh = source->useMesh;
00132     thee->meshID = source->meshID;
00133 }
```

```

00134
00135 VPUBLIC void FEMparm_dtor(FEMparm **thee) {
00136     if ((*thee) != VNULL) {
00137         FEMparm_dtor2(*thee);
00138         Vmem_free(VNULL, 1, sizeof(FEMparm), (void **)thee);
00139         (*thee) = VNULL;
00140     }
00141 }
00142
00143 VPUBLIC void FEMparm_dtor2(FEMparm *thee) { ; }
00144
00145 VPUBLIC int FEMparm_check(FEMparm *thee) {
00146
00147     int rc;
00148     rc = 1;
00149
00150     if (!thee->parsed) {
00151         Vnm_print(2, "FEMparm_check: not filled!\n");
00152         return 0;
00153     }
00154     if (!thee->setttype) {
00155         Vnm_print(2, "FEMparm_check: type not set!\n");
00156         rc = 0;
00157     }
00158     if (!thee->setglen) {
00159         Vnm_print(2, "FEMparm_check: glen not set!\n");
00160         rc = 0;
00161     }
00162     if (!thee->setetol) {
00163         Vnm_print(2, "FEMparm_check: etol not set!\n");
00164         rc = 0;
00165     }
00166     if (!thee->setekey) {
00167         Vnm_print(2, "FEMparm_check: ekey not set!\n");
00168         rc = 0;
00169     }
00170     if (!thee->setakeyPRE) {
00171         Vnm_print(2, "FEMparm_check: akeyPRE not set!\n");
00172         rc = 0;
00173     }
00174     if (!thee->setakeySOLVE) {
00175         Vnm_print(2, "FEMparm_check: akeySOLVE not set!\n");
00176         rc = 0;
00177     }
00178     if (!thee->settargetNum) {
00179         Vnm_print(2, "FEMparm_check: targetNum not set!\n");
00180         rc = 0;
00181     }
00182     if (!thee->settargetRes) {
00183         Vnm_print(2, "FEMparm_check: targetRes not set!\n");
00184         rc = 0;
00185     }
00186     if (!thee->setmaxsolve) {
00187         Vnm_print(2, "FEMparm_check: maxsolve not set!\n");
00188         rc = 0;
00189     }
00190     if (!thee->setmaxvert) {
00191         Vnm_print(2, "FEMparm_check: maxvert not set!\n");
00192         rc = 0;
00193     }
00194
00195     return rc;
00196 }
00197
00198 VPRIVATE Vrc_Codes FEMparm_parseDOMAINLENGTH(FEMparm *thee,
00199                                         Vio *sock
00200                                         ) {
00201
00202     int i;
00203     double tf;
00204     char tok[VMAX_BUFSIZE];
00205
00206     for (i=0; i<3; i++) {
00207         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00208         if (sscanf(tok, "%lf", &tf) == 0) {
00209             Vnm_print(2, "parseFE: Read non-double (%s) while parsing \
00210 DOMAINLENGTH keyword!\n", tok);
00211             return VRC_FAILURE;
00212         }
00213         thee->glen[i] = tf;
00214     }

```

```

00215     thee->setglen = 1;
00216     return VRC_SUCCESS;
00217 VERROR1:
00218     Vnm_print(2, "parseFE: ran out of tokens!\n");
00219     return VRC_FAILURE;
00220
00221 }
00222
00223 VPRIPRIVATE Vrc_Codes FEMparm_parseETOL(FEMparm *thee,
00224                                     Vio *sock
00225                                     ) {
00226
00227     double tf;
00228     char tok[VMAX_BUFSIZE];
00229
00230     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00231     if (sscanf(tok, "%lf", &tf) == 0) {
00232         Vnm_print(2, "parseFE: Read non-double (%s) while parsing \
00233 ETOL keyword!\n", tok);
00234         return VRC_FAILURE;
00235     }
00236     thee->etol = tf;
00237     thee->setetol = 1;
00238     return VRC_SUCCESS;
00239 VERROR1:
00240     Vnm_print(2, "parseFE: ran out of tokens!\n");
00241     return VRC_FAILURE;
00242
00243
00244 }
00245
00246 VPRIPRIVATE Vrc_Codes FEMparm_parseEKEY(FEMparm *thee,
00247                                     Vio *sock
00248                                     ) {
00249
00250     char tok[VMAX_BUFSIZE];
00251     Vrc_Codes vrc = VRC_FAILURE;
00252
00253     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00254     if (Vstring_strcasecmp(tok, "simp") == 0) {
00255         thee->ekey = FET_SIMP;
00256         thee->setekey = 1;
00257         vrc = VRC_SUCCESS;
00258     } else if (Vstring_strcasecmp(tok, "glob") == 0) {
00259         thee->ekey = FET_GLOB;
00260         thee->setekey = 1;
00261         vrc = VRC_SUCCESS;
00262     } else if (Vstring_strcasecmp(tok, "frac") == 0) {
00263         thee->ekey = FET_FRAC;
00264         thee->setekey = 1;
00265         vrc = VRC_SUCCESS;
00266     } else {
00267         Vnm_print(2, "parseFE: undefined value (%s) for ekey!\n", tok);
00268         vrc = VRC_FAILURE;
00269     }
00270
00271     return vrc;
00272 VERROR1:
00273     Vnm_print(2, "parseFE: ran out of tokens!\n");
00274     return VRC_FAILURE;
00275
00276 }
00277
00278 VPRIPRIVATE Vrc_Codes FEMparm_parseAKEYPRE(FEMparm *thee, Vio *sock) {
00279
00280     char tok[VMAX_BUFSIZE];
00281     Vrc_Codes vrc = VRC_FAILURE;
00282
00283     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00284     if (Vstring_strcasecmp(tok, "unif") == 0) {
00285         thee->akeyPRE = FRT_UNIF;
00286         thee->setakeyPRE = 1;
00287         vrc = VRC_SUCCESS;
00288     } else if (Vstring_strcasecmp(tok, "geom") == 0) {
00289         thee->akeyPRE = FRT_GEOM;
00290         thee->setakeyPRE = 1;
00291         vrc = VRC_SUCCESS;
00292     } else {
00293         Vnm_print(2, "parseFE: undefined value (%s) for akeyPRE!\n", tok);
00294         vrc = VRC_FAILURE;
00295     }

```

```

00296
00297     return vrc;
00298
00299 VERROR1:
00300     Vnm_print(2, "parseFE: ran out of tokens!\n");
00301     return VRC_FAILURE;
00302
00303 }
00304
00305 VPRIVATE Vrc_Codes FEMparm_parseAKEYSOLVE(FEMparm *thee, Vio *sock) {
00306
00307     char tok[VMAX_BUFSIZE];
00308     Vrc_Codes vrc = VRC_FAILURE;
00309
00310     VJMPERR1(Vio_scantok(sock, "%s", tok) == 1);
00311     if (Vstring_strcasecmp(tok, "resi") == 0) {
00312         thee->akeySOLVE = FRT_RESI;
00313         thee->setakeySOLVE = 1;
00314         vrc = VRC_SUCCESS;
00315     } else if (Vstring_strcasecmp(tok, "dual") == 0) {
00316         thee->akeySOLVE = FRT_DUAL;
00317         thee->setakeySOLVE = 1;
00318         vrc = VRC_SUCCESS;
00319     } else if (Vstring_strcasecmp(tok, "loca") == 0) {
00320         thee->akeySOLVE = FRT_LOCA;
00321         thee->setakeySOLVE = 1;
00322         vrc = VRC_SUCCESS;
00323     } else {
00324         Vnm_print(2, "parseFE: undefined value (%s) for akeyPRE!\n", tok);
00325         vrc = VRC_FAILURE;
00326     }
00327
00328     return vrc;
00329 VERROR1:
00330     Vnm_print(2, "parseFE: ran out of tokens!\n");
00331     return VRC_SUCCESS;
00332
00333 }
00334
00335 VPRIVATE Vrc_Codes FEMparm_parseTARGETNUM(FEMparm *thee, Vio *sock) {
00336
00337     char tok[VMAX_BUFSIZE];
00338     int ti;
00339
00340     VJMPERR1(Vio_scantok(sock, "%s", tok) == 1);
00341     if (sscanf(tok, "%d", &ti) == 0) {
00342         Vnm_print(2, "parseFE: read non-int (%s) for targetNum!\n", tok);
00343         return VRC_FAILURE;
00344     }
00345     thee->targetNum = ti;
00346     thee->settargetNum = 1;
00347     return VRC_SUCCESS;
00348 VERROR1:
00349     Vnm_print(2, "parseFE: ran out of tokens!\n");
00350     return VRC_FAILURE;
00351
00352 }
00353
00354 VPRIVATE Vrc_Codes FEMparm_parseTARGETRES(FEMparm *thee, Vio *sock) {
00355
00356     char tok[VMAX_BUFSIZE];
00357     double tf;
00358
00359     VJMPERR1(Vio_scantok(sock, "%s", tok) == 1);
00360     if (sscanf(tok, "%lf", &tf) == 0) {
00361         Vnm_print(2, "parseFE: read non-double (%s) for targetNum!\n",
00362                 tok);
00363         return VRC_FAILURE;
00364     }
00365     thee->targetRes = tf;
00366     thee->settargetRes = 1;
00367     return VRC_SUCCESS;
00368 VERROR1:
00369     Vnm_print(2, "parseFE: ran out of tokens!\n");
00370     return VRC_FAILURE;
00371
00372 }
00373
00374 VPRIVATE Vrc_Codes FEMparm_parseMAXSOLVE(FEMparm *thee, Vio *sock) {
00375
00376     char tok[VMAX_BUFSIZE];

```

```

00377     int ti;
00378
00379     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00380     if (sscanf(tok, "%d", &ti) == 0) {
00381         Vnm_print(2, "parseFE:  read non-int (%s) for maxsolve!\n", tok);
00382         return VRC_FAILURE;
00383     }
00384     thee->maxsolve = ti;
00385     thee->setmaxsolve = 1;
00386     return VRC_SUCCESS;
00387 VERROR1:
00388     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00389     return VRC_FAILURE;
00390
00391 }
00392
00393 VPRIPRIVATE Vrc_Codes FEMparm_parseMAXVERT(FEMparm *thee, Vio *sock) {
00394
00395     char tok[VMAX_BUFSIZE];
00396     int ti;
00397
00398     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00399     if (sscanf(tok, "%d", &ti) == 0) {
00400         Vnm_print(2, "parseFE:  read non-int (%s) for maxvert!\n", tok);
00401         return VRC_FAILURE;
00402     }
00403     thee->maxvert = ti;
00404     thee->setmaxvert = 1;
00405     return VRC_SUCCESS;
00406
00407 VERROR1:
00408     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00409     return VRC_FAILURE;
00410
00411 }
00412
00413 VPRIPRIVATE Vrc_Codes FEMparm_parseUSEMESH(FEMparm *thee, Vio *sock) {
00414     char tok[VMAX_BUFSIZE];
00415     int ti;
00416
00417     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00418     if (sscanf(tok, "%d", &ti) == 0) {
00419         Vnm_print(2, "parseFE:  read non-int (%s) for usemesh!\n", tok);
00420         return VRC_FAILURE;
00421     }
00422     thee->useMesh = 1;
00423     thee->meshID = ti;
00424
00425     return VRC_SUCCESS;
00426
00427 VERROR1:
00428     Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00429     return VRC_FAILURE;
00430 }
00431
00432
00433 VPUBLIC Vrc_Codes FEMparm_parseToken(FEMparm *thee,
00434     char tok[VMAX_BUFSIZE],
00435     Vio *sock) {
00436
00437     //int i, ti; // gcc says unused
00438     //double tf; // gcc says unused
00439
00440     if (thee == VNULL) {
00441         Vnm_print(2, "parseFE:  got NULL thee!\n");
00442         return VRC_FAILURE;
00443     }
00444
00445     if (sock == VNULL) {
00446         Vnm_print(2, "parseFE:  got NULL socket!\n");
00447         return VRC_FAILURE;
00448     }
00449
00450     if (Vstring_strcasecmp(tok, "domainLength") == 0) {
00451         return FEMparm_parseDOMAINLENGTH(thee, sock);
00452     } else if (Vstring_strcasecmp(tok, "etol") == 0) {
00453         return FEMparm_parseETOL(thee, sock);
00454     } else if (Vstring_strcasecmp(tok, "ekey") == 0) {
00455         return FEMparm_parseEKEY(thee, sock);
00456     } else if (Vstring_strcasecmp(tok, "akeyPRE") == 0) {
00457         return FEMparm_parseAKEYPRE(thee, sock);

```

```

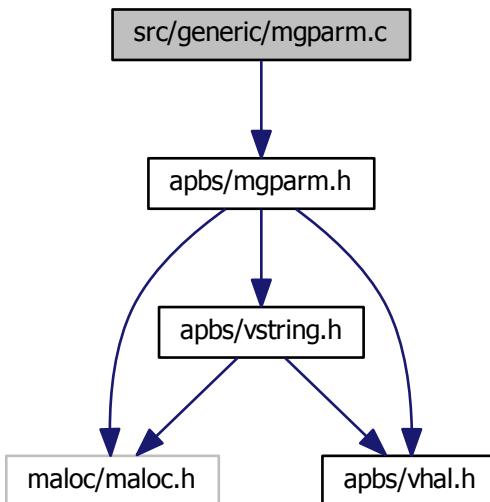
00457     } else if (Vstring_strcasecmp(tok, "akeySOLVE") == 0) {
00458         return FEMparm_parseAKEYSOLVE(thee, sock);
00459     } else if (Vstring_strcasecmp(tok, "targetNum") == 0) {
00460         return FEMparm_parseTARGETNUM(thee, sock);
00461     } else if (Vstring_strcasecmp(tok, "targetRes") == 0) {
00462         return FEMparm_parseTARGETRES(thee, sock);
00463     } else if (Vstring_strcasecmp(tok, "maxsolve") == 0) {
00464         return FEMparm_parseMAXSOLVE(thee, sock);
00465     } else if (Vstring_strcasecmp(tok, "maxvert") == 0) {
00466         return FEMparm_parseMAXVERT(thee, sock);
00467     } else if (Vstring_strcasecmp(tok, "usemesh") == 0) {
00468         return FEMparm_parseUSEMESH(thee, sock);
00469     }
00470     return VRC_WARNING;
00471 }
00472
00473 }
```

9.57 src/generic/mgparm.c File Reference

Class MGparm methods.

```
#include "apbs/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)
 - FORTRAN stub to construct MGparm object.
- VPUBLIC void **MGparm_dtor** (**MGparm** **thee)
 - Object destructor.
- VPUBLIC void **MGparm_dtor2** (**MGparm** *thee)
 - FORTRAN stub for object destructor.
- VPUBLIC Vrc_Codes **MGparm_check** (**MGparm** *thee)
 - Consistency check for parameter values stored in object.
- VPUBLIC void **MGparm_copy** (**MGparm** *thee, **MGparm** *parm)
 - Copy MGparm object into thee.
- VPRIVATE Vrc_Codes **MGparm_parseDIME** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseCHGM** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseNLEV** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseETOL** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseGRID** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseGLEN** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseGAMMA** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseGCENT** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseCGLEN** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseFGLEN** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseCGCENT** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseFGCENT** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parsePDIME** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseOFRAC** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseASYNC** (**MGparm** *thee, **Vio** *sock)
- VPRIVATE Vrc_Codes **MGparm_parseUSEAQUA** (**MGparm** *thee, **Vio** *sock)
- VPUBLIC Vrc_Codes **MGparm_parseToken** (**MGparm** *thee, **char** tok[VMAX_BUFSIZE], **Vio** *sock)
 - Parse an MG keyword from an input file.

9.57.1 Detailed Description

Class MGparm methods.

Author

Nathan Baker

Version

Id:

[mgparm.c](#) 1763 2012-07-18 22:04:34Z tuckerbeck

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](#).

9.58 mgparm.c

```

00001
00057 #include "apbs/mgparm.h"
00058
00059 VEMBED(rcsid="$Id: mgparm.c 1763 2012-07-18 22:04:34Z tuckerbeck $")
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->setchgm = 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->setcgcen = 0;
00156     thee->setfgcen = 0;
00157
00158 /* *** TYPE 2 PARAMETERS *** */
00159     thee->setpdime = 0;
00160     thee->setrank = 0;
00161     thee->setsiz = 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
00187     Vrc_Codes rc;
00188     int i, tdim[3], ti, tnlev[3], nlev;
00189
00190     rc = VRC_SUCCESS;
00191
00192     Vnm_print(0, "MGparm_check: checking MGparm object of type %d.\n",
00193             thee->type);
00194
00195     /* Check to see if we were even filled... */
00196     if (!thee->parsed) {
00197         Vnm_print(2, "MGparm_check: not filled!\n");
00198         return VRC_FAILURE;
00199     }
00200
00201     /* Check generic settings */
00202     if (!thee->setdime) {
00203         Vnm_print(2, "MGparm_check: DIME not set!\n");
00204         rc = VRC_FAILURE;
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 */

```

```

00213     if ((thee->type == MCT_MANUAL) || (thee->type ==
00214         MCT_DUMMY)) {
00215         if ((!thee->setgrid) && (!thee->setglen)) {
00216             Vnm_print(2, "MGparm_check: Neither GRID nor GLEN set!\n");
00217             rc = VRC_FAILURE;
00218         }
00219         if ((thee->setgrid) && (thee->setglen)) {
00220             Vnm_print(2, "MGparm_check: Both GRID and GLEN set!\n");
00221             rc = VRC_FAILURE;
00222         }
00223         if (!thee->setgcnt) {
00224             Vnm_print(2, "MGparm_check: GCENT not set!\n");
00225             rc = VRC_FAILURE;
00226         }
00227     }
00228
00229     /* Check sequential and parallel automatic focusing settings */
00230     if ((thee->type == MCT_AUTO) || (thee->type == MCT_PARALLEL)
00231 ) {
00232         if (!thee->setcglen) {
00233             Vnm_print(2, "MGparm_check: CGLEN not set!\n");
00234             rc = VRC_FAILURE;
00235         }
00236         if (!thee->setfglen) {
00237             Vnm_print(2, "MGparm_check: FGLEN not set!\n");
00238             rc = VRC_FAILURE;
00239         }
00240         if (!thee->setcgcent) {
00241             Vnm_print(2, "MGparm_check: CGCENT not set!\n");
00242             rc = VRC_FAILURE;
00243         }
00244         if (!thee->setfgcent) {
00245             Vnm_print(2, "MGparm_check: FGCENT not set!\n");
00246             rc = VRC_FAILURE;
00247         }
00248
00249     /* Check parallel automatic focusing settings */
00250     if (thee->type == MCT_PARALLEL) {
00251         if (!thee->setpdime) {
00252             Vnm_print(2, "MGparm_check: PDIME not set!\n");
00253             rc = VRC_FAILURE;
00254         }
00255         if (!thee->setrank) {
00256             Vnm_print(2, "MGparm_check: PROC_RANK not set!\n");
00257             rc = VRC_FAILURE;
00258         }
00259         if (!thee->setsize) {
00260             Vnm_print(2, "MGparm_check: PROC_SIZE not set!\n");
00261             rc = VRC_FAILURE;
00262         }
00263         if (!thee->setofrac) {
00264             Vnm_print(2, "MGparm_check: OFRAC not set!\n");
00265             rc = VRC_FAILURE;
00266         }
00267     }
00268
00269     /* Perform a sanity check on nlev and dime, resetting values as necessary
00270 */
00271     if (rc == 1) {
00272
00273     /* Calculate the actual number of grid points and nlev to satisfy the
00274      * formula: n = c * 2^(l+1) + 1, where n is the number of grid points,
00275      * c is an integer, and l is the number of levels */
00276     if (thee->type != MCT_DUMMY) {
00277         for (i=0; i<3; i++) {
00278             /* See if the user picked a reasonable value, if not then fix it */
00279             ti = thee->dime[i] - 1;
00280             if (ti == VPOW(2, (thee->nlev+1))) {
00281                 tlev[i] = thee->nlev;
00282                 tdim[i] = thee->dime[i];
00283             } else {
00284                 tdim[i] = thee->dime[i];
00285                 ti = tdim[i] - 1;
00286                 tlev[i] = 0;
00287                 /* Find the maximum number of times this dimension can be
00288                  * divided by two */
00289                 while (VEVEN(ti)) {
00290                     (tlev[i])++;
00291                     ti = (int)ceil(0.5*ti);
00292                 }
00293                 (tlev[i])--;
00294             }
00295         }
00296     }
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01960
01961
01962
01963
01964
01965

```

```

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, tdim[i], tnlev[i]);
00297                 ti = (int)(tdime[i]/VPOW(2.,(VMGNLEV+1)));
00298                 if (ti < 1) ti = 1;
00299                 tdim[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).
00302                               \n", i, tdim[i], VMGNLEV);
00303             }
00304         }
00305     } else { /* We are a dummy calculation, but we still need positive numbers of
00306     * points */
00307         for (i=0; i<3; i++) {
00308             tnlev[i] = thee->nlev;
00309             tdim[i] = thee->dime[i];
00310             if (thee->dime[i] <= 0) {
00311                 Vnm_print(2, "NOsh: Resetting dime[%d] from %d to 3.\n", i, thee->dime
00312                           [i]);
00313                 thee->dime[i] = 3;
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, tdim[0],
00322               tdim[1], tdim[2]);
00323     thee->nlev = nlev;
00324     if (thee->nlev <= 0) {
00325         Vnm_print(2, "MGparm_check: illegal nlev (%d); check your grid dimensions!
00326                               \n", thee->nlev);
00327         rc = VRC_FAILURE;
00328     }
00329     if (thee->nlev < 2) {
00330         Vnm_print(2, "MGparm_check: you're using a very small nlev (%d) and
00331 therefore\n", thee->nlev);
00332         Vnm_print(2, "MGparm_check: will not get the optimal performance of the
00333 multigrid\n");
00334         Vnm_print(2, "MGparm_check: algorithm. Please check your grid dimensions.
00335                               \n");
00336     }
00337     for (i=0; i<3; i++) thee->dime[i] = tdim[i];
00338 }
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
00349     thee->type = parm->type;
00350     thee->parsed = parm->parsed;
00351
00352     /* *** GENERIC PARAMETERS *** */
00353     for (i=0; i<3; i++) thee->dime[i] = parm->dime[i];
00354     thee->setdime = parm->setdime;
00355     thee->chgm = parm->chgm;
00356     thee->setchgm = parm->setchgm;
00357     thee->chgs = parm->chgs;
00358
00359     /* *** TYPE 0 PARMs *** */
00360     thee->nlev = parm->nlev;
00361     thee->setnlev = parm->setnlev;
00362     thee->etol = parm->etol;
00363     thee->setetol = parm->setetol;
00364     for (i=0; i<3; i++) thee->grid[i] = parm->grid[i];

```

```

00365     thee->setgrid = parm->setgrid;
00366     for (i=0; i<3; i++) thee->glen[i] = parm->glen[i];
00367     thee->setglen = parm->setglen;
00368     thee->cmeth = parm->cmeth;
00369     for (i=0; i<3; i++) thee->center[i] = parm->center[i];
00370     thee->setgcent = parm->setgcent;
00371     thee->centmol = parm->centmol;
00372
00373     /* *** TYPE 1 & 2 PARMs *** */
00374     for (i=0; i<3; i++) thee->crlen[i] = parm->crlen[i];
00375     thee->setcrlen = parm->setcrlen;
00376     for (i=0; i<3; i++) thee->flen[i] = parm->flen[i];
00377     thee->setflen = parm->setflen;
00378     thee->ccmeth = parm->ccmeth;
00379     for (i=0; i<3; i++) thee->ccenter[i] = parm->ccenter[i];
00380     thee->setcgcent = parm->setcgcent;
00381     thee->ccentmol = parm->ccentmol;
00382     thee->fcmeth = parm->fcmeth;
00383     for (i=0; i<3; i++) thee->fcenter[i] = parm->fcenter[i];
00384     thee->setfgcent = parm->setfgcent;
00385     thee->fcenmol = parm->fcenmol;
00386
00387     /* *** TYPE 2 PARMs *** */
00388     for (i=0; i<3; i++)
00389         thee->partDisjCenter[i] = parm->partDisjCenter
00390         [i];
00391         for (i=0; i<3; i++)
00392             thee->partDisjLength[i] = parm->partDisjLength
00393             [i];
00394             for (i=0; i<6; i++)
00395                 thee->partDisjOwnSide[i] = parm->partDisjOwnSide
00396                 [i];
00397                 for (i=0; i<3; i++) thee->pdime[i] = parm->pdime[i];
00398                 thee->setpdime = parm->setpdime;
00399                 thee->proc_rank = parm->proc_rank;
00400                 thee->setrank = parm->setrank;
00401                 thee->proc_size = parm->proc_size;
00402                 thee->setszie = parm->setszie;
00403                 thee->ofrac = parm->ofrac;
00404                 thee->setofrac = parm->setofrac;
00405                 thee->setasync = parm->setasync;
00406                 thee->async = parm->async;
00407
00408                 thee->nonlintype = parm->nonlintype;
00409                 thee->setnonlintype = parm->setnonlintype;
00410
00411                 thee->method = parm->method;
00412                 thee->method = parm->method;
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_scant(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_scant(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_scant(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     VERROR1:
00442         Vnm_print(2, "parseMG: ran out of tokens!\n");

```

```

00443         return VRC_WARNING;
00444     }
00445
00446 VPRIPRIVATE Vrc_Codes MGparm_parseCHGM(MGparm *thee, Vio *sock) {
00447
00448     char tok[VMAX_BUFSIZE];
00449     Vchrg_Meth ti;
00450
00451     VJMPERR1(Vio_sccanf(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 statement \"chgm %d\".\n", ti);
00456         Vnm_print(2, "NOsh: Please use \"chgm \"");
00457         switch (thee->chgm) {
00458             case VCM_TRI1:
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_TRI1;
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     VERROR1:
00493         Vnm_print(2, "parseMG: ran out of tokens!\n");
00494         return VRC_WARNING;
00495 }
00496
00497 VPRIPRIVATE Vrc_Codes MGparm_parseNLEV(MGparm *thee, Vio *sock) {
00498
00499     char tok[VMAX_BUFSIZE];
00500     int ti;
00501
00502     VJMPERR1(Vio_sccanf(sock, "%s", tok) == 1);
00503     if (sscanf(tok, "%d", &ti) == 0) {
00504         Vnm_print(2, "NOsh: Read non-integer (%s) while parsing NLEV \
00505 keyword!\n", tok);
00506         return VRC_WARNING;
00507     } else thee->nlev = ti;
00508     thee->setnlev = 1;
00509     return VRC_SUCCESS;
00510
00511     VERROR1:
00512         Vnm_print(2, "parseMG: ran out of tokens!\n");
00513         return VRC_WARNING;
00514 }
00515
00516 VPRIPRIVATE Vrc_Codes MGparm_parseETOL(MGparm *thee, Vio *sock) {
00517
00518     char tok[VMAX_BUFSIZE];
00519     double tf;
00520
00521     VJMPERR1(Vio_sccanf(sock, "%s", tok) == 1);
00522     if (sscanf(tok, "%lf", &tf) == 0) {

```

```

00523     Vnm_print(2, "NOsh: Read non-float (%s) while parsing etol \
00524 keyword!\n", tok);
00525     return VRC_WARNING;
00526 } else if (tf <= 0.0) {
00527     Vnm_print(2, "parseMG: etol must be greater than 0!\n");
00528     return VRC_WARNING;
00529 } else thee->etol = tf;
00530 thee->setetol = 1;
00531 return VRC_SUCCESS;
00532
00533 VERROR1:
00534     Vnm_print(2, "parseMG: ran out of tokens!\n");
00535     return VRC_WARNING;
00536 }
00537
00538
00539 VPRIVATE Vrc_Codes MGparm_parseGRID(MGparm *thee, Vio *sock) {
00540
00541     char tok[VMAX_BUFSIZE];
00542     double tf;
00543
00544     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00545     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 VERROR1:
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 VERROR1:
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
00608 message,\n");
00609     Vnm_print(2, "parseMG: please contact the developers of those programs
00610 regarding this message.\n");
00611     return VRC_SUCCESS;
00612
00613 VERROR1:
00614     Vnm_print(2, "parseMG: ran out of tokens!\n");
00615     return VRC_WARNING;
00616 }
00617
00618 VPRIVATE Vrc_Codes MGparm_parseGCENT (MGparm *thee, Vio *sock) {
00619
00620     char tok[VMAX_BUFSIZE];
00621     double tf;
00622     int ti;
00623
00624     /* If the next token isn't a float, it probably means we want to
00625      * center on a molecule */
00626     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00627     if (sscanf(tok, "%lf", &tf) == 0) {
00628         if (Vstring_stcasecmp(tok, "mol") == 0) {
00629             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00630             Vnm_print(2, "Nosh: Read non-int (%s) while parsing \
00631 GCENT MOL keyword!\n", tok);
00632             return VRC_WARNING;
00633         } else {
00634             thee->cmeth = MCM_MOLECULE;
00635             /* Subtract 1 here to convert user numbering (1, 2, 3, ...) into
00636              array index */
00637             thee->centmol = ti - 1;
00638         }
00639     } else {
00640         Vnm_print(2, "Nosh: Unexpected keyword (%s) while parsing \
00641 GCENT!\n", tok);
00642         return VRC_WARNING;
00643     }
00644     else {
00645         thee->center[0] = tf;
00646         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00647         if (sscanf(tok, "%lf", &tf) == 0) {
00648             Vnm_print(2, "Nosh: Read non-float (%s) while parsing \
00649 GCENT keyword!\n", tok);
00650             return VRC_WARNING;
00651         }
00652         thee->center[1] = tf;
00653         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00654         if (sscanf(tok, "%lf", &tf) == 0) {
00655             Vnm_print(2, "Nosh: Read non-float (%s) while parsing \
00656 GCENT keyword!\n", tok);
00657             return VRC_WARNING;
00658         }
00659         thee->center[2] = tf;
00660     }
00661     thee->setgcent = 1;
00662     return VRC_SUCCESS;
00663
00664 VERROR1:
00665     Vnm_print(2, "parseMG: ran out of tokens!\n");
00666     return VRC_WARNING;
00667 }
00668
00669 VPRIVATE Vrc_Codes MGparm_parseCGLEN (MGparm *thee, Vio *sock) {
00670
00671     char tok[VMAX_BUFSIZE];
00672     double tf;
00673
00674     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00675     if (sscanf(tok, "%lf", &tf) == 0) {
00676         Vnm_print(2, "Nosh: Read non-float (%s) while parsing CGLEN \
00677 keyword!\n", tok);
00678         return VRC_WARNING;
00679     } else thee->crlen[0] = tf;
00680     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00681     if (sscanf(tok, "%lf", &tf) == 0) {
00682         Vnm_print(2, "Nosh: Read non-float (%s) while parsing CGLEN \
00683 keyword!\n", tok);

```

```

00683     return VRC_WARNING;
00684 } else thee->crlen[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->crlen[2] = tf;
00691 thee->setcrlen = 1;
00692 return VRC_SUCCESS;
00693
00694 VERROR1:
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 VERROR1:
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 FGCENT keyword!\n", tok);
00770             return VRC_WARNING;
00771         }
00772         thee->ccenter[2] = tf;
00773     }
00774     thee->setcgcen = 1;
00775     return VRC_SUCCESS;
00776
00777     VERROR1:
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->setfgcen = 1;
00827     return VRC_SUCCESS;
00828
00829     VERROR1:
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         theee->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         theee->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         theee->pdime[2] = ti;
00863     }
00864     theee->setpdime = 1;
00865     return VRC_SUCCESS;
00866
00867     VERROR1:
00868     Vnm_print(2, "parseMG:  ran out of tokens!\n");
00869     return VRC_WARNING;
00870 }
00871
00872 VPUBLIC Vrc_Codes MGparm_parseOFRAC(MGparm *theee, Vio *sock) {
00873
00874     char tok[VMAX_BUFSIZE];
00875     double tf;
00876
00877     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00878     if (sscanf(tok, "%lf", &tf) == 0) {
00879         Vnm_print(2, "Nosh:  Read non-int (%s) while parsing OFRAC \
00880 keyword!\n", tok);
00881         return VRC_WARNING;
00882     }
00883     theee->ofrac = tf;
00884     theee->setofrac = 1;
00885     return VRC_SUCCESS;
00886
00887     VERROR1:
00888     Vnm_print(2, "parseMG:  ran out of tokens!\n");
00889     return VRC_WARNING;
00890 }
00891
00892 VPUBLIC Vrc_Codes MGparm_parseASYNC(MGparm *theee, Vio *sock) {
00893
00894     char tok[VMAX_BUFSIZE];
00895     int ti;
00896
00897     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00898     if (sscanf(tok, "%i", &ti) == 0) {
00899         Vnm_print(2, "Nosh:  Read non-integer (%s) while parsing ASYNC \
00900 keyword!\n", tok);
00901         return VRC_WARNING;
00902     }
00903     theee->async = ti;
00904     theee->setasync = 1;
00905     return VRC_SUCCESS;
00906
00907     VERROR1:
00908     Vnm_print(2, "parseMG:  ran out of tokens!\n");
00909     return VRC_WARNING;
00910 }
00911
00912 VPUBLIC Vrc_Codes MGparm_parseUSEAQUA(MGparm *theee, Vio *sock) {
00913     Vnm_print(0, "Nosh:  parsed useaqua\n");
00914     theee->useAqua = 1;
00915     theee->setUseAqua = 1;
00916     return VRC_SUCCESS;
00917 }
00918
00919 VPUBLIC Vrc_Codes MGparm_parseToken(MGparm *theee, char
00920     tok[VMAX_BUFSIZE],
00921     Vio *sock) {
00922     if (theee == VNNULL) {
00923         Vnm_print(2, "parseMG:  got NULL thee!\n");
00924         return VRC_WARNING;

```

```

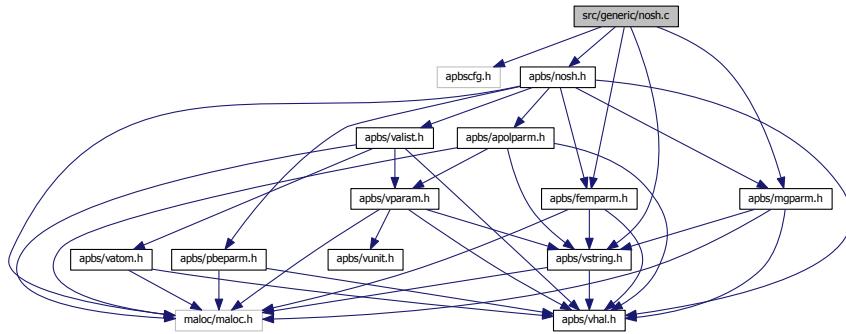
00925      }
00926      if (sock == VNULL) {
00927          Vnm_print(2, "parseMG: got NULL socket!\n");
00928          return VRC_WARNING;
00929      }
00930
00931  Vnm_print(0, "MGparm_parseToken: trying %s...\n", tok);
00932
00933
00934      if (Vstring_strcasecmp(tok, "dime") == 0) {
00935          return MGparm_parseDIME(thee, sock);
00936      } else if (Vstring_strcasecmp(tok, "chgm") == 0) {
00937          return MGparm_parseCHGM(thee, sock);
00938      } else if (Vstring_strcasecmp(tok, "nlev") == 0) {
00939          Vnm_print(2, "Warning: The 'nlev' keyword is now deprecated!\n");
00940          return MGparm_parseNLEV(thee, sock);
00941      } else if (Vstring_strcasecmp(tok, "etol") == 0) {
00942          return MGparm_parseETOL(thee, sock);
00943      } else if (Vstring_strcasecmp(tok, "grid") == 0) {
00944          return MGparm_parseGRID(thee, sock);
00945      } else if (Vstring_strcasecmp(tok, "glen") == 0) {
00946          return MGparm_parseGLEN(thee, sock);
00947      } else if (Vstring_strcasecmp(tok, "gcent") == 0) {
00948          return MGparm_parseGCENT(thee, sock);
00949      } else if (Vstring_strcasecmp(tok, "cglen") == 0) {
00950          return MGparm_parseCGLEN(thee, sock);
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 }
```

9.59 src/generic/nosh.c File Reference

Class NOsh methods.

```
#include "apbscfg.h"
#include "apbs/nosh.h"
#include "apbs/vstring.h"
#include "apbs/mgparm.h"
#include "apbs/femparm.h"
```

Include dependency graph for nosh.c:



Functions

- VPRIVATE int **NOsh_parseREAD** (NOsh *thee, Vio *sock)
 - VPRIVATE int **NOsh_parsePRINT** (NOsh *thee, Vio *sock)
 - VPRIVATE int **NOsh_parseELEC** (NOsh *thee, Vio *sock)
 - VPRIVATE int **NOsh_parseAPOLAR** (NOsh *thee, Vio *sock)
 - VEXTERNC int **NOsh_parseFEM** (NOsh *thee, Vio *sock, NOsh_calc *elec)
 - VEXTERNC int **NOsh_parseMG** (NOsh *thee, Vio *sock, NOsh_calc *elec)
 - VEXTERNC int **NOsh_parseAPOL** (NOsh *thee, Vio *sock, NOsh_calc *elec)
 - VPRIVATE int **NOsh_setupCalcMG** (NOsh *thee, NOsh_calc *elec)
 - VPRIVATE int **NOsh_setupCalcMGAUTO** (NOsh *thee, NOsh_calc *elec)
 - VPRIVATE int **NOsh_setupCalcMGMANUAL** (NOsh *thee, NOsh_calc *elec)
 - VPRIVATE int **NOsh_setupCalcMGPARA** (NOsh *thee, NOsh_calc *elec)
 - VPRIVATE int **NOsh_setupCalcFEM** (NOsh *thee, NOsh_calc *elec)
 - VPRIVATE int **NOsh_setupCalcFEMANUAL** (NOsh *thee, NOsh_calc *elec)
 - VPRIVATE int **NOsh_setupCalcAPOL** (NOsh *thee, NOsh_calc *elec)
 - VPUBLIC char * **NOsh_getMolpath** (NOsh *thee, int imol)

Returns path to specified molecule.
 - VPUBLIC char * **NOsh_getDielXpath** (NOsh *thee, int imol)

Returns path to specified x-shifted dielectric map.
 - VPUBLIC char * **NOsh_getDielYpath** (NOsh *thee, int imol)

Returns path to specified y-shifted dielectric map.
 - VPUBLIC char * **NOsh_getDielZpath** (NOsh *thee, int imol)

Returns path to specified z-shifted dielectric map.
 - VPUBLIC char * **NOsh_getKappapath** (NOsh *thee, int imol)

Returns path to specified kappa map.
 - VPUBLIC char * **NOsh_getPotpath** (NOsh *thee, int imol)

Returns path to specified potential map.
 - VPUBLIC char * **NOsh_getChargepath** (NOsh *thee, int imol)

Returns path to specified charge distribution map.
 - VPUBLIC NOsh_calc * **NOsh_getCalc** (NOsh *thee, int icalc)

Returns specified calculation object.
 - VPUBLIC int **NOsh_getDielfmt** (NOsh *thee, int i)

- **VPUBLIC int NOsh_getKappafmt (NOsh *thee, int i)**

Returns format of specified dielectric map.
- **VPUBLIC int NOsh_getPotfmt (NOsh *thee, int i)**

Returns format of specified kappa map.
- **VPUBLIC int NOsh_getChargefmt (NOsh *thee, int i)**

Returns format of specified potential map.
- **VPUBLIC NOsh_PrintType NOsh_printWhat (NOsh *thee, int iprint)**

Return an integer ID of the observable to print .
- **VPUBLIC int NOsh_printNarg (NOsh *thee, int iprint)**

Return number of arguments to PRINT statement .
- **VPUBLIC int NOsh_elec2calc (NOsh *thee, int icalc)**

Return the name of an elec statement.
- **VPUBLIC int NOsh_apol2calc (NOsh *thee, int icalc)**

Return the name of an apol statement.
- **VPUBLIC char * NOsh_elecname (NOsh *thee, int ielec)**

Return an integer mapping of an ELEC statement to a calculation ID .
- **VPUBLIC int NOsh_printOp (NOsh *thee, int iprint, int iarg)**

Return integer ID for specified operation .
- **VPUBLIC int NOsh_printCalc (NOsh *thee, int iprint, int iarg)**

Return calculation ID for specified PRINT statement .
- **VPUBLIC NOsh * NOsh_ctor (int rank, int size)**

Construct NOsh.
- **VPUBLIC int NOsh_ctor2 (NOsh *thee, int rank, int size)**

FORTRAN stub to construct NOsh.
- **VPUBLIC void Nosh_dtor (NOsh **thee)**

Object destructor.
- **VPUBLIC void Nosh_dtor2 (NOsh *thee)**

FORTRAN stub for object destructor.
- **VPUBLIC NOsh_calc * NOsh_calc_ctor (NOsh_CalcType calctype)**

Construct NOsh_calc.
- **VPUBLIC void Nosh_calc_dtor (NOsh_calc **thee)**

Object destructor.
- **VPUBLIC int NOsh_calc_copy (NOsh_calc *thee, NOsh_calc *source)**

Copy NOsh_calc object into thee.
- **VPUBLIC int NOsh_parseInputFile (NOsh *thee, char *filename)**

Parse an input file only from a file.
- **VPUBLIC int NOsh_parseInput (NOsh *thee, Vio *sock)**

Parse an input file from a socket.
- **VPRIVATE int Nosh_parseREAD_MOL (NOsh *thee, Vio *sock)**
- **VPRIVATE int Nosh_parseREAD_PARM (NOsh *thee, Vio *sock)**
- **VPRIVATE int Nosh_parseREAD_DIEL (NOsh *thee, Vio *sock)**
- **VPRIVATE int Nosh_parseREAD_KAPPA (NOsh *thee, Vio *sock)**
- **VPRIVATE int Nosh_parseREAD_POTENTIAL (NOsh *thee, Vio *sock)**
- **VPRIVATE int Nosh_parseREAD_CHARGE (NOsh *thee, Vio *sock)**
- **VPRIVATE int Nosh_parseREAD_MESH (NOsh *thee, Vio *sock)**
- **VPUBLIC int NOsh_setupElecCalc (NOsh *thee, Valist *alist[NOSH_MAXMOL])**

Setup the series of electrostatics calculations.
- **VPUBLIC int NOsh_setupApolCalc (NOsh *thee, Valist *alist[NOSH_MAXMOL])**

Setup the series of non-polar calculations.

9.59.1 Detailed Description

Class NOsh methods.

Author

Nathan Baker

Version

Id:

[nosh.c](#) 1750 2012-07-18 18:34:27Z tuckerbeck

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](#).

9.60 nosh.c

```
00001
00058 #include "apbscfg.h"
00059 #include "apbs/nosh.h"
00060 #include "apbs/vstring.h"
00061 #include "apbs/mgparm.h"
00062 #include "apbs/femparm.h"
00063
00064 VEMBED(rcsid="$Id: nosh.c 1750 2012-07-18 18:34:27Z tuckerbeck $")
00065
00066
00067 VPRIvATE int NOsh_parseREAD(
00068     NOsh *thee,
00069     Vio *sock);
00070
00071 VPRIvATE int NOsh_parsePRINT(
00072     NOsh *thee,
00073     Vio *sock);
00074
00075 VPRIvATE int NOsh_parseELEC(
00076     NOsh *thee,
00077     Vio *sock
00078 );
00079
00080 VPRIvATE int NOsh_parseAPOLAR(
00081     NOsh *thee,
00082     Vio *sock
00083 );
00084
00085 VEXTERNC int NOsh_parseFEM(
00086     NOsh *thee,
00087     Vio *sock,
00088     NOsh_calc *elec
00089 );
00090
00091 VEXTERNC int NOsh_parseMG(
00092     NOsh *thee,
00093     Vio *sock,
00094     NOsh_calc *elec
00095 );
00096
00097 VEXTERNC int NOsh_parseAPOL(
00098     NOsh *thee,
00099     Vio *sock,
00100     NOsh_calc *elec
00101 );
00102
00103 VPRIvATE int NOsh_setupCalcMG(
00104     NOsh *thee,
00105     NOsh_calc *elec
00106 );
00107
00108 VPRIvATE int NOsh_setupCalcMGAUTO(
00109     NOsh *thee,
00110     NOsh_calc *elec
00111 );
00112
00113 VPRIvATE int NOsh_setupCalcMGMANUAL(
00114     NOsh *thee,
00115     NOsh_calc *elec
00116 );
00117
00118 VPRIvATE int NOsh_setupCalcMGPARA(
00119     NOsh *thee,
00120     NOsh_calc *elec
00121 );
00122
00123 VPRIvATE int NOsh_setupCalcFEM(
00124     NOsh *thee,
00125     NOsh_calc *elec
00126 );
00127
00128 VPRIvATE int NOsh_setupCalcFEMANUAL(
00129     NOsh *thee,
00130     NOsh_calc *elec
00131 );
00132
00133 VPRIvATE int NOsh_setupCalcAPOL(
00134     NOsh *thee,
```

```

00135         NOsh_calc *elec
00136     );
00137
00138 #if !defined(VINLINE_NOSH)
00139
00140 VPUBLIC char* NOsh_getMolpath(NOsh *thee, int imol) {
00141     VASSERT(thee != VNULL);
00142     VASSERT(imol < thee->nmol);
00143     return thee->molpath[imol];
00144 }
00145 VPUBLIC char* NOsh_getDielXpath(NOsh *thee, int imol) {
00146     VASSERT(thee != VNULL);
00147     VASSERT(imol < thee->nmol);
00148     return thee->dielxpath[imol];
00149 }
00150 VPUBLIC char* NOsh_getDielYpath(NOsh *thee, int imol) {
00151     VASSERT(thee != VNULL);
00152     VASSERT(imol < thee->nmol);
00153     return thee->dielypath[imol];
00154 }
00155 VPUBLIC char* NOsh_getDielZpath(NOsh *thee, int imol) {
00156     VASSERT(thee != VNULL);
00157     VASSERT(imol < thee->nmol);
00158     return thee->dielzpath[imol];
00159 }
00160 VPUBLIC char* NOsh_getKappapath(NOsh *thee, int imol) {
00161     VASSERT(thee != VNULL);
00162     VASSERT(imol < thee->nmol);
00163     return thee->kappapath[imol];
00164 }
00165 VPUBLIC char* NOsh_getPotpath(NOsh *thee, int imol) {
00166     VASSERT(thee != VNULL);
00167     VASSERT(imol < thee->nmol);
00168     return thee->potpath[imol];
00169 }
00170 VPUBLIC char* NOsh_getChargepath(NOsh *thee, int imol) {
00171     VASSERT(thee != VNULL);
00172     VASSERT(imol < thee->nmol);
00173     return thee->chargepath[imol];
00174 }
00175 VPUBLIC NOsh_calc* NOsh_getCalc(NOsh *thee, int icalc)
00176 {
00177     VASSERT(thee != VNULL);
00178     VASSERT(icalc < thee->ncalc);
00179     return thee->calc[icalc];
00180 }
00181 VPUBLIC int NOsh_getDielfmt(NOsh *thee, int i) {
00182     VASSERT(thee != VNULL);
00183     VASSERT(i < thee->ndiel);
00184     return (thee->dielfmt[i]);
00185 }
00186 VPUBLIC int NOsh_getKappafmt(NOsh *thee, int i) {
00187     VASSERT(thee != VNULL);
00188     VASSERT(i < thee->nkappa);
00189     return (thee->kappafmt[i]);
00190 }
00191 VPUBLIC int NOsh_getPotfmt(NOsh *thee, int i) {
00192     VASSERT(thee != VNULL);
00193     VASSERT(i < thee->npot);
00194     return (thee->potfmt[i]);
00195 }
00196 VPUBLIC int NOsh_getChargefmt(NOsh *thee, int i) {
00197     VASSERT(thee != VNULL);
00198     VASSERT(i < thee->ncharge);
00199     return (thee->chargefmt[i]);
00200
00201
00202 #endif /* if !defined(VINLINE_NOSH) */
00203
00204 VPUBLIC NOsh_PrintType NOsh_printWhat(NOsh *
00205     thee, int iprint) {
00206     VASSERT(thee != VNULL);
00207     VASSERT(iprint < thee->nprint);
00208     return thee->printwhat[iprint];
00209 }
00210 VPUBLIC int NOsh_printNarg(NOsh *thee, int iprint) {
00211     VASSERT(thee != VNULL);
00212     VASSERT(iprint < thee->nprint);
00213     return thee->printnarg[iprint];

```

```
00214 }
00215
00216 VPUBLIC int NOsh_elec2calc(NOsh *thee, int icalc) {
00217     VASSERT(thee != VNULL);
00218     VASSERT(icalc < thee->nalc);
00219     return thee->elec2calc[icalc];
00220 }
00221
00222 VPUBLIC int NOsh_apol2calc(NOsh *thee, int icalc) {
00223     VASSERT(thee != VNULL);
00224     VASSERT(icalc < thee->nalc);
00225     return thee->apol2calc[icalc];
00226 }
00227
00228 VPUBLIC char* NOsh_elecname(NOsh *thee, int ielec) {
00229     VASSERT(thee != VNULL);
00230     VASSERT(ielec < thee->nlec + 1);
00231     return thee->elecname[ielec];
00232 }
00233
00234 VPUBLIC int NOsh_printOp(NOsh *thee, int iprint, int iarg) {
00235     VASSERT(thee != VNULL);
00236     VASSERT(iprint < thee->nprint);
00237     VASSERT(iarg < thee->printnarg[iprint]);
00238     return thee->printop[iprint][iarg];
00239 }
00240
00241 VPUBLIC int NOsh_printCalc(NOsh *thee, int iprint, int iarg)
00242 {
00243     VASSERT(thee != VNULL);
00244     VASSERT(iprint < thee->nprint);
00245     VASSERT(iarg < thee->printnarg[iprint]);
00246     return thee->printcalc[iprint][iarg];
00247 }
00248 VPUBLIC NOsh* NOsh_ctor(int rank, int size) {
00249
00250     /* Set up the structure */
00251     NOsh *thee = VNULL;
00252     thee = (NOsh*)Vmemp_malloc(VNULL, 1, sizeof(NOsh));
00253     VASSERT(thee != VNULL);
00254     VASSERT(NOsh_ctor2(thee, rank, size));
00255
00256     return thee;
00257 }
00258
00259 VPUBLIC int NOsh_ctor2(NOsh *thee, int rank, int size) {
00260
00261     int i;
00262
00263     if (thee == VNULL) return 0;
00264
00265     thee->proc_rank = rank;
00266     thee->proc_size = size;
00267
00268     thee->ispara = 0;
00269     thee->parsed = 0;
00270
00271     thee->nmol = 0;
00272     thee->gotparm = 0;
00273     thee->ncharge = 0;
00274     thee->nndiel = 0;
00275     thee->nkappa = 0;
00276     thee->npot = 0;
00277     thee->nprint = 0;
00278
00279     for (i=0; i<NOSH_MAXCALC; i++) {
00280         thee->calc[i] = VNULL;
00281         thee->elec[i] = VNULL;
00282         thee->apol[i] = VNULL;
00283     }
00284     for (i=0; i<NOSH_MAXMOL; i++) {
00285         thee->alist[i] = VNULL;
00286     }
00287     thee->nalc = 0;
00288     thee->nlec = 0;
00289     thee->npol = 0;
00290
00291     return 1;
00292 }
```

```

00294 VPUBLIC void NOsh_dtor(NOsh **thee) {
00295     if ((*thee) != VNULL) {
00296         NOsh_dtor2(*thee);
00297         Vmem_free(VNULL, 1, sizeof(NOsh), (void **)thee);
00298         (*thee) = VNULL;
00299     }
00300 }
00301
00302 VPUBLIC void NOsh_dtor2(NOsh *thee) {
00303
00304     int i;
00305
00306     if (thee != VNULL) {
00307         for (i=0; i<(thee->nalc); i++) NOsh_calc_dtor(&(thee->
00308             calc[i]));
00309         for (i=0; i<(thee->nelec); i++) NOsh_calc_dtor(&(thee->
00310             elec[i]));
00311         for (i=0; i<(thee->napol); i++) NOsh_calc_dtor(&(thee->
00312             apol[i]));
00313     }
00314 VPUBLIC NOsh_calc* NOsh_calc_ctor(
00315     NOsh_CalcType calctype
00316     ) {
00317     NOsh_calc *thee;
00318     thee = (NOsh_calc *)Vmem_malloc(VNULL, 1, sizeof(NOsh_calc))
00319     ;
00320     thee->calctype = calctype;
00321     switch (calctype) {
00322         case NCT_MG:
00323             thee->mgparm = MGparm_ctor(MCT_NONE);
00324             thee->femparm = VNULL;
00325             thee->apolparm = VNULL;
00326             break;
00327         case NCT_FEM:
00328             thee->mgparm = FEMparm_ctor(FCT_NONE);
00329             thee->apolparm = VNULL;
00330             break;
00331         case NCT_APOL:
00332             thee->mgparm = VNULL;
00333             thee->femparm = VNULL;
00334             thee->apolparm = APOLparm_ctor();
00335             break;
00336         default:
00337             Vnm_print(2, "NOsh_calc_ctor: unknown calculation type (%d)!\n",
00338                     calctype);
00339             VASSERT(0);
00340     }
00341     thee->pbeparm = PBEparm_ctor();
00342
00343     return thee;
00344 }
00345
00346 VPUBLIC void NOsh_calc_dtor(
00347     NOsh_calc **thee
00348     ) {
00349
00350     NOsh_calc *calc = VNULL;
00351     calc = *thee;
00352     if (calc == VNULL) return;
00353
00354     switch (calc->calctype) {
00355         case NCT_MG:
00356             MGparm_dtor(&(calc->mgparm));
00357             break;
00358         case NCT_FEM:
00359             FEMparm_dtor(&(calc->femparm));
00360             break;
00361         case NCT_APOL:
00362             APOLparm_dtor(&(calc->apolparm));
00363             break;
00364         default:
00365             Vnm_print(2, "NOsh_calc_dtor: unknown calculation type (%d)!\n",
00366                     calc->calctype);
00367             VASSERT(0);
00368     }
00369     PBEparm_dtor(&(calc->pbeparm));
00370 }
```

```

00371 Vmem_free(VNULL, 1, sizeof(NoSh_calc), (void **)thee);
00372 calc = VNULL;
00373
00374 }
00375
00376 VPUBLIC int NoSh_calc_copy(
00377     NoSh_calc *thee,
00378     NoSh_calc *source
00379 ) {
00380
00381 VASSERT(thee != VNULL);
00382 VASSERT(source != VNULL);
00383 VASSERT(thee->calctype == source->calctype);
00384 if (source->mgparm != VNULL)
00385 MGparm_copy(thee->mgparm, source->mgparm);
00386 if (source->femparm != VNULL)
00387 FEMparm_copy(thee->femparm, source->femparm);
00388 if (source->pbeparm != VNULL)
00389 PBEparm_copy(thee->pbeparm, source->pbeparm);
00390 if (source->apolparm != VNULL)
00391 APOLparm_copy(thee->apolparm, source->apolparm);
00392
00393 return 1;
00394
00395 }
00396
00397 VPUBLIC int NoSh_parseInputFile(
00398     NoSh *thee,
00399     char *filename
00400 ) {
00401
00402 Vio *sock;
00403 int rc;
00404
00405 sock = Vio_ctor("FILE", "ASC", VNULL, filename, "r");
00406 rc = NoSh_parseInput(thee, sock);
00407 Vio_dtor(&sock);
00408
00409 return rc;
00410 }
00411
00412 VPUBLIC int NoSh_parseInput(
00413     NoSh *thee,
00414     Vio *sock
00415 ) {
00416
00417 char *MCwhiteChars = " =,\t\r\n";
00418 char *MCcommChars = "%#";
00419 char tok[VMAX_BUFSIZE];
00420
00421 if (thee == VNULL) {
00422 Vnm_print(2, "NoSh_parseInput: Got NULL thee!\n");
00423 return 0;
00424 }
00425
00426 if (sock == VNULL) {
00427 Vnm_print(2, "NoSh_parseInput: Got pointer to NULL socket!\n");
00428 Vnm_print(2, "NoSh_parseInput: The specified input file was not found!\n");
00429 return 0;
00430 }
00431
00432 if (thee->parsed) {
00433 Vnm_print(2, "NoSh_parseInput: Already parsed an input file!\n");
00434 return 0;
00435 }
00436
00437 if (Vio_accept(sock, 0) < 0) {
00438 Vnm_print(2, "NoSh_parseInput: Problem reading from socket!\n");
00439 return 0;
00440 }
00441
00442 /* Set up the whitespace and comment character definitions */
00443 Vio_setWhiteChars(sock, MCwhiteChars);
00444 Vio_setCommChars(sock, MCcommChars);
00445
00446 /* We parse the file until we run out of tokens */
00447 Vnm_print(0, "NoSh_parseInput: Starting file parsing...\n");
00448 while (Vio_scanf(sock, "%s", tok) == 1) {
00449 /* At the highest level, we look for keywords that indicate functions like:
00450 read => Read in a molecule file

```

```

00452     elec => Do an electrostatics calculation
00453     print => Print some results
00454     apolar => do a non-polar calculation
00455     quit => Quit
00456
00457 These cause the code to go to a lower-level parser routine which
00458 handles keywords specific to the particular function. Each
00459 lower-level parser routine then returns when it hits the "end"
00460 keyword. Due to this simple layout, no nesting of these "function"
00461 sections is allowed.
00462 */
00463 if (Vstring_strcasecmp(tok, "read") == 0) {
00464     Vnm_print(0, "NOsh: Parsing READ section\n");
00465     if (!NOsh_parseREAD(thee, sock)) return 0;
00466     Vnm_print(0, "NOsh: Done parsing READ section \
00467 (nmol=%d, ndiel=%d, nkappa=%d, ncharge=%d, npot=%d)\n", thee->nmol, thee->
00468     ndiel,
00469     thee->nkappa, thee->ncharge, thee->npot);
00470 } else if (Vstring_strcasecmp(tok, "print") == 0) {
00471     Vnm_print(0, "NOsh: Parsing PRINT section\n");
00472     if (!NOsh_parsePRINT(thee, sock)) return 0;
00473     Vnm_print(0, "NOsh: Done parsing PRINT section\n");
00474 } else if (Vstring_strcasecmp(tok, "elec") == 0) {
00475     Vnm_print(0, "NOsh: Parsing ELEC section\n");
00476     if (!NOsh_parseELEC(thee, sock)) return 0;
00477     Vnm_print(0, "NOsh: Done parsing ELEC section (nelec = %d)\n",
00478     thee->nelec);
00479 } else if (Vstring_strcasecmp(tok, "apolar") == 0) {
00480     Vnm_print(0, "NOsh: Parsing APOLAR section\n");
00481     if (!NOsh_parseAPOLAR(thee, sock)) return 0;
00482     Vnm_print(0, "NOsh: Done parsing APOLAR section (nelec = %d)\n",
00483     thee->nelec);
00484 } else if (Vstring_strcasecmp(tok, "quit") == 0) {
00485     Vnm_print(0, "NOsh: Done parsing file (got QUIT)\n");
00486     break;
00487 } else {
00488     Vnm_print(2, "NOsh_parseInput: Ignoring undefined keyword %s!\n", tok);
00489 }
00490
00491 thee->parsed = 1;
00492 return 1;
00493
00494 }
00495
00496 VPRIVATE int NOsh_parseREAD_MOL(NOsh *thee, Vio *sock) {
00497
00498     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00499     NOsh_MolFormat molfmt;
00500
00501     VJMPERR1(Vio_scantf(sock, "%s", tok) == 1);
00502     if (Vstring_strcasecmp(tok, "pqr") == 0) {
00503         molfmt = NMF_PQR;
00504         VJMPERR1(Vio_scantf(sock, "%s", tok) == 1);
00505         if (tok[0]== '/') {
00506             strcpy(strnew, "");
00507             while (tok[strlen(tok)-1] != '/') {
00508                 strcat(str, tok);
00509                 strcat(str, " ");
00510                 VJMPERR1(Vio_scantf(sock, "%s", tok) == 1);
00511             }
00512             strcat(str, tok);
00513             strncpy(strnew, str+1, strlen(str)-2);
00514             strcpy(tok, strnew);
00515         }
00516         Vnm_print(0, "NOsh: Storing molecule %d path %s\n",
00517         thee->nmol, tok);
00518         thee->molfmt[thee->nmol] = molfmt;
00519         strncpy(thee->molpath[thee->nmol], tok, VMAX_ARGLEN);
00520         (thee->nmol)++;
00521     } else if (Vstring_strcasecmp(tok, "pdb") == 0) {
00522         molfmt = NMF_PDB;
00523         VJMPERR1(Vio_scantf(sock, "%s", tok) == 1);
00524         if (tok[0]== '/') {
00525             strcpy(strnew, "");
00526             while (tok[strlen(tok)-1] != '/') {
00527                 strcat(str, tok);
00528                 strcat(str, " ");
00529                 VJMPERR1(Vio_scantf(sock, "%s", tok) == 1);
00530             }
00531             strcat(str, tok);

```

```

00532         strncpy(strnew, str+1, strlen(str)-2);
00533         strcpy(tok, strnew);
00534     }
00535     Vnm_print(0, "NOsh: Storing molecule %d path %s\n",
00536     thee->nmol, tok);
00537     thee->molfmt[thee->nmol] = molfmt;
00538     strncpy(thee->molpath[thee->nmol], tok, VMAX_ARGLEN);
00539     (thee->nmol)++;
00540 } else if (Vstring_strcasecmp(tok, "xml") == 0) {
00541     molfmt = NMF_XML;
00542     VJMPERR1(Vio_scnf(sock, "%s", tok) == 1);
00543     if (tok[0]== '/') {
00544         strcpy(strnew, "");
00545         while (tok[strlen(tok)-1] != '/') {
00546             strcat(str, tok);
00547             strcat(str, " ");
00548             VJMPERR1(Vio_scnf(sock, "%s", tok) == 1);
00549         }
00550         strcat(str, tok);
00551         strncpy(strnew, str+1, strlen(str)-2);
00552         strcpy(tok, strnew);
00553     }
00554     Vnm_print(0, "NOsh: Storing molecule %d path %s\n",
00555     thee->nmol, tok);
00556     thee->molfmt[thee->nmol] = molfmt;
00557     strncpy(thee->molpath[thee->nmol], tok, VMAX_ARGLEN);
00558     (thee->nmol)++;
00559 } else {
00560     Vnm_print(2, "NOsh_parseREAD: Ignoring undefined mol format \
00561 %s!\n", tok);
00562 }
00563
00564     return 1;
00565
00566
00567 VERROR1:
00568     Vnm_print(2, "NOsh_parseREAD_MOL: Ran out of tokens while parsing READ
00569 section!\n");
00570     return 0;
00571 }
00572
00573 VPRIVATE int NOsh_parseREAD_PARM(NOsh *thee, Vio *sock) {
00574
00575     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00576     NOsh_ParmFormat parmfmt;
00577
00578     VJMPERR1(Vio_scnf(sock, "%s", tok) == 1);
00579     if (Vstring_strcasecmp(tok, "flat") == 0) {
00580         parmfmt = NPF_FLAT;
00581         VJMPERR1(Vio_scnf(sock, "%s", tok) == 1);
00582         if (tok[0]== '/') {
00583             strcpy(strnew, "");
00584             while (tok[strlen(tok)-1] != '/') {
00585                 strcat(str, tok);
00586                 strcat(str, " ");
00587                 VJMPERR1(Vio_scnf(sock, "%s", tok) == 1);
00588             }
00589             strcat(str, tok);
00590             strncpy(strnew, str+1, strlen(str)-2);
00591             strcpy(tok, strnew);
00592         }
00593         if (thee->gotparm) {
00594             Vnm_print(2, "NOsh: Hey! You already specified a parameterfile
00595 (%s)!\n", thee->parmpath);
00596             Vnm_print(2, "NOsh: I'm going to ignore this one (%s)!\n", tok);
00597         } else {
00598             thee->parmfmt = parmfmt;
00599             thee->gotparm = 1;
00600             strncpy(thee->parmpath, tok, VMAX_ARGLEN);
00601         }
00602     } else if(Vstring_strcasecmp(tok, "xml") == 0) {
00603         parmfmt = NPF_XML;
00604         VJMPERR1(Vio_scnf(sock, "%s", tok) == 1);
00605         if (tok[0]== '/') {
00606             strcpy(strnew, "");
00607             while (tok[strlen(tok)-1] != '/') {
00608                 strcat(str, tok);
00609                 strcat(str, " ");
00610                 VJMPERR1(Vio_scnf(sock, "%s", tok) == 1);
00611             }
00612         }

```

```

00611         strcat(str, tok);
00612         strncpy(strnew, str+1, strlen(str)-2);
00613         strcpy(tok, strnew);
00614     }
00615     if (thee->gotparm) {
00616         Vnm_print(2, "NOsh: Hey! You already specified a parameterfile
00617 (%s)\n", thee->parmpath);
00618     } else {
00619         thee->parmfmt = parmfmt;
00620         thee->gotparm = 1;
00621         strncpy(thee->parmpath, tok, VMAX_ARGLEN);
00622     }
00623
00624 } else {
00625     Vnm_print(2, "NOsh_parseREAD: Ignoring undefined parm format \
00626 %s!\n", tok);
00627 }
00628
00629     return 1;
00630
00631 VERROR1:
00632     Vnm_print(2, "NOsh_parseREAD_PARM: Ran out of tokens while parsing
00633 READ section!\n");
00634     return 0;
00635 }
00636
00637 VPRIPRIVATE int NOsh_parseREAD_DIEL(NOsh *thee, Vio *sock) {
00638
00639     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00640     Vdata_Format dielfmt;
00641
00642     VJMPERR1(Vio_scnf(sock, "%s", tok) == 1);
00643     if (Vstring_strcasecmp(tok, "dx") == 0) {
00644         dielfmt = VDF_DX;
00645     } else if (Vstring_strcasecmp(tok, "gz") == 0) {
00646         dielfmt = VDF_GZ;
00647     } else {
00648         Vnm_print(2, "NOsh_parseREAD: Ignoring undefined format \
00649 %s!\n", tok);
00650     return VRC_FAILURE;
00651 }
00652
00653 VJMPERR1(Vio_scnf(sock, "%s", tok) == 1);
00654 if (tok[0]== '/') {
00655     strcpy(strnew, "");
00656     while (tok[strlen(tok)-1] != '/') {
00657         strcat(str, tok);
00658         strcat(str, " ");
00659         VJMPERR1(Vio_scnf(sock, "%s", tok) == 1);
00660     }
00661     strcat(str, tok);
00662     strncpy(strnew, str+1, strlen(str)-2);
00663     strcpy(tok, strnew);
00664 }
00665 Vnm_print(0, "NOsh: Storing x-shifted dielectric map %d path \
00666 %s\n", thee->ndiel, tok);
00667 strncpy(thee->dielXpath[thee->ndiel], tok, VMAX_ARGLEN);
00668 VJMPERR1(Vio_scnf(sock, "%s", tok) == 1);
00669 Vnm_print(0, "NOsh: Storing y-shifted dielectric map %d path \
00670 %s\n", thee->ndiel, tok);
00671 strncpy(thee->dielYpath[thee->ndiel], tok, VMAX_ARGLEN);
00672 VJMPERR1(Vio_scnf(sock, "%s", tok) == 1);
00673 Vnm_print(0, "NOsh: Storing z-shifted dielectric map %d path \
00674 %s\n", thee->ndiel, tok);
00675 strncpy(thee->dielZpath[thee->ndiel], tok, VMAX_ARGLEN);
00676 thee->dielfmt[thee->ndiel] = dielfmt;
00677 (thee->ndiel)++;
00678
00679     return 1;
00680
00681 VERROR1:
00682     Vnm_print(2, "NOsh_parseREAD_DIEL: Ran out of tokens while parsing
00683 READ \
00684 section!\n");
00685     return 0;
00686 }
00687
00688 VPRIPRIVATE int NOsh_parseREAD_KAPPA(NOsh *thee, Vio *sock) {

```

```

00689     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00690     Vdata_Format kappafmt;
00691
00692     VJMPERR1(Vio_scnf(sock, "%s", tok) == 1);
00693     if (Vstring_strcasecmp(tok, "dx") == 0) {
00694         kappafmt = VDF_DX;
00695     } else if (Vstring_strcasecmp(tok, "gz") == 0) {
00696         kappafmt = VDF_GZ;
00697     } else {
00698         Vnm_print(2, "NOsh_parseREAD: Ignoring undefined format \
00699             %s\n", tok);
00700         return VRC_FAILURE;
00701     }
00702
00703     VJMPERR1(Vio_scnf(sock, "%s", tok) == 1);
00704     if (tok[0]==')') {
00705         strcpy(strnew, "");
00706         while (tok[strlen(tok)-1] != ')') {
00707             strcat(str, tok);
00708             strcat(str, " ");
00709             VJMPERR1(Vio_scnf(sock, "%s", tok) == 1);
00710         }
00711         strcat(str, tok);
00712         strncpy(strnew, str+1, strlen(str)-2);
00713         strcpy(tok, strnew);
00714     }
00715     Vnm_print(0, "NOsh: Storing kappa map %d path %s\n",
00716             thee->nkappa, tok);
00717     thee->kappafmt[thee->nkappa] = kappafmt;
00718     strncpy(thee->kappapath[thee->nkappa], tok, VMAX_ARGLEN);
00719     (thee->nkappa)++;
00720
00721     return 1;
00722
00723
00724 VERROR1:
00725     Vnm_print(2, "NOsh_parseREAD: Ran out of tokens while parsing READ \
00726 section!\n");
00727     return 0;
00728
00729 }
00730
00731 VPRIPRIVATE int NOsh_parseREAD_POTENTIAL(NOsh *thee, Vio *sock) {
00732
00733     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00734     Vdata_Format potfmt;
00735
00736     VJMPERR1(Vio_scnf(sock, "%s", tok) == 1);
00737     if (Vstring_strcasecmp(tok, "dx") == 0) {
00738         potfmt = VDF_DX;
00739     } else if (Vstring_strcasecmp(tok, "gz") == 0) {
00740         potfmt = VDF_GZ;
00741     } else {
00742         Vnm_print(2, "NOsh_parseREAD: Ignoring undefined format \
00743             %s\n", tok);
00744         return VRC_FAILURE;
00745     }
00746
00747     VJMPERR1(Vio_scnf(sock, "%s", tok) == 1);
00748     if (tok[0]==')') {
00749         strcpy(strnew, "");
00750         while (tok[strlen(tok)-1] != ')') {
00751             strcat(str, tok);
00752             strcat(str, " ");
00753             VJMPERR1(Vio_scnf(sock, "%s", tok) == 1);
00754         }
00755         strcat(str, tok);
00756         strncpy(strnew, str+1, strlen(str)-2);
00757         strcpy(tok, strnew);
00758     }
00759     Vnm_print(0, "NOsh: Storing potential map %d path %s\n",
00760             thee->npot, tok);
00761     thee->potfmt[thee->npot] = potfmt;
00762     strncpy(thee->potpath[thee->npot], tok, VMAX_ARGLEN);
00763     (thee->npot)++;
00764
00765     return 1;
00766
00767 VERROR1:
00768     Vnm_print(2, "NOsh_parseREAD: Ran out of tokens while parsing READ \
00769             section!\n");

```

```

00770     return 0;
00771
00772 }
00773
00774 VPRIVATE int NOsh_parseREAD_CHARGE(NOsh *thee, Vio *sock) {
00775
00776     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00777     Vdata_Format chargefmt;
00778
00779     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00780     if (Vstring_strcasecmp(tok, "dx") == 0) {
00781         chargefmt = VDF_DX;
00782     } else if (Vstring_strcasecmp(tok, "gz") == 0) {
00783         chargefmt = VDF_GZ;
00784     } else {
00785         Vnm_print(2, "NOsh_parseREAD: Ignoring undefined format \
00786         %s!\n", tok);
00787         return VRC_FAILURE;
00788     }
00789
00790     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00791     if (tok[0]==')') {
00792         strcpy(strnew, "");
00793         while (tok[strlen(tok)-1] != ')') {
00794             strcat(str, tok);
00795             strcat(str, " ");
00796             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00797         }
00798         strcat(str, tok);
00799         strncpy(strnew, str+1, strlen(str)-2);
00800         strcpy(tok, strnew);
00801     }
00802     Vnm_print(0, "NOsh: Storing charge map %d path %s\n",
00803             thee->ncharge, tok);
00804     thee->chargefmt[thee->ncharge] = chargefmt;
00805     strncpy(thee->chargepath[thee->ncharge], tok, VMAX_ARGLEN);
00806     (thee->ncharge)++;
00807
00808     return 1;
00809
00810 VERROR1:
00811     Vnm_print(2, "NOsh_parseREAD: Ran out of tokens while parsing READ \
00812 section!\n");
00813     return 0;
00814
00815 }
00816
00817 VPRIVATE int NOsh_parseREAD_MESH(NOsh *thee, Vio *sock) {
00818
00819     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00820     Vdata_Format meshfmt;
00821
00822     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00823     if (Vstring_strcasecmp(tok, "mcsf") == 0) {
00824         meshfmt = VDF_MCSF;
00825         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00826         if (tok[0]==')') {
00827             strcpy(strnew, "");
00828             while (tok[strlen(tok)-1] != ')') {
00829                 strcat(str, tok);
00830                 strcat(str, " ");
00831                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00832             }
00833             strcat(str, tok);
00834             strncpy(strnew, str+1, strlen(str)-2);
00835             strcpy(tok, strnew);
00836         }
00837         Vnm_print(0, "NOsh: Storing mesh %d path %s\n",
00838             thee->nmesh, tok);
00839         thee->meshfmt[thee->nmesh] = meshfmt;
00840         strncpy(thee->meshpath[thee->nmesh], tok, VMAX_ARGLEN);
00841         (thee->nmesh)++;
00842     } else {
00843         Vnm_print(2, "NOsh_parseREAD: Ignoring undefined mesh format \
00844         %s!\n", tok);
00845     }
00846
00847     return 1;
00848
00849 VERROR1:
00850     Vnm_print(2, "NOsh_parseREAD: Ran out of tokens while parsing READ \

```

```

00851     section!\n");
00852     return 0;
00853 }
00855
00856
00857 VPRIIVATE int NOsh_parseREAD(NOsh *thee, Vio *sock) {
00858     char tok[VMAX_BUFSIZE];
00859
00860     if (thee == VNULL) {
00861         Vnm_print(2, "NOsh_parseREAD: Got NULL thee!\n");
00862         return 0;
00863     }
00864
00865     if (sock == VNULL) {
00866         Vnm_print(2, "NOsh_parseREAD: Got pointer to NULL socket!\n");
00867         return 0;
00868     }
00869
00870     if (thee->parsed) {
00871         Vnm_print(2, "NOsh_parseREAD: Already parsed an input file!\n");
00872         return 0;
00873     }
00874
00875
00876     /* Read until we run out of tokens (bad) or hit the "END" keyword (good) */
00877     while (Vio_scanf(sock, "%s", tok) == 1) {
00878         if (Vstring_strcasecmp(tok, "end") == 0) {
00879             Vnm_print(0, "NOsh: Done parsing READ section\n");
00880             return 1;
00881         } else if (Vstring_strcasecmp(tok, "mol") == 0) {
00882             NOsh_parseREAD_MOL(thee, sock);
00883         } else if (Vstring_strcasecmp(tok, "parm") == 0) {
00884             NOsh_parseREAD_PARM(thee, sock);
00885         } else if (Vstring_strcasecmp(tok, "diel") == 0) {
00886             NOsh_parseREAD_DIEL(thee, sock);
00887         } else if (Vstring_strcasecmp(tok, "kappa") == 0) {
00888             NOsh_parseREAD_KAPPA(thee, sock);
00889         } else if (Vstring_strcasecmp(tok, "pot") == 0) {
00890             NOsh_parseREAD_POTENTIAL(thee, sock);
00891         } else if (Vstring_strcasecmp(tok, "charge") == 0) {
00892             NOsh_parseREAD_CHARGE(thee, sock);
00893         } else if (Vstring_strcasecmp(tok, "mesh") == 0) {
00894             NOsh_parseREAD_MESH(thee, sock);
00895         } else {
00896             Vnm_print(2, "NOsh_parseREAD: Ignoring undefined keyword %s!\n",
00897             tok);
00898         }
00899     }
00900
00901     /* We ran out of tokens! */
00902     Vnm_print(2, "NOsh_parseREAD: Ran out of tokens while parsing READ \
00903 section!\n");
00904     return 0;
00905
00906 }
00907
00908 VPRIIVATE int NOsh_parsePRINT(NOsh *thee, Vio *sock) {
00909
00910     char tok[VMAX_BUFSIZE];
00911     char name[VMAX_BUFSIZE];
00912     int ti, idx, expect, ielec, iapol;
00913
00914     if (thee == VNULL) {
00915         Vnm_print(2, "NOsh_parsePRINT: Got NULL thee!\n");
00916         return 0;
00917     }
00918
00919     if (sock == VNULL) {
00920         Vnm_print(2, "NOsh_parsePRINT: Got pointer to NULL socket!\n");
00921         return 0;
00922     }
00923
00924     if (thee->parsed) {
00925         Vnm_print(2, "NOsh_parsePRINT: Already parsed an input file!\n");
00926         return 0;
00927     }
00928
00929     idx = thee->nprint;
00930     if (thee->nprint >= NOSH_MAXPRINT) {
00931         Vnm_print(2, "NOsh_parsePRINT: Exceeded max number (%d) of PRINT \

```

```

00932 sections\n",
00933     NOSH_MAXPRINT);
00934     return 0;
00935 }
00936
00937
00938 /* The first thing we read is the thing we want to print */
00939 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00940 if (Vstring_strcasecmp(tok, "energy") == 0) {
00941     thee->printwhat[idx] = NPT_ENERGY;
00942     thee->printnarg[idx] = 0;
00943 } else if (Vstring_strcasecmp(tok, "force") == 0) {
00944     thee->printwhat[idx] = NPT_FORCE;
00945     thee->printnarg[idx] = 0;
00946 } else if (Vstring_strcasecmp(tok, "elecEnergy") == 0) {
00947     thee->printwhat[idx] = NPT_ELECENERGY;
00948     thee->printnarg[idx] = 0;
00949 } else if (Vstring_strcasecmp(tok, "elecForce") == 0) {
00950     thee->printwhat[idx] = NPT_ELECFORCE;
00951     thee->printnarg[idx] = 0;
00952 } else if (Vstring_strcasecmp(tok, "apolEnergy") == 0) {
00953     thee->printwhat[idx] = NPT_APOLENERGY;
00954     thee->printnarg[idx] = 0;
00955 } else if (Vstring_strcasecmp(tok, "apolForce") == 0) {
00956     thee->printwhat[idx] = NPT_APOLFORCE;
00957     thee->printnarg[idx] = 0;
00958 } else {
00959     Vnm_print(2, "NOsh_parsePRINT: Undefined keyword %s while parsing \
00960 PRINT section!\n", tok);
00961     return 0;
00962 }
00963
00964 expect = 0; /* We first expect a calculation ID (0) then an op (1) */
00965
00966 /* Read until we run out of tokens (bad) or hit the "END" keyword (good) */
00967 while (Vio_scanf(sock, "%s", tok) == 1) {
00968
00969 /* The next thing we read is either END or an ARG OP ARG statement */
00970 if (Vstring_strcasecmp(tok, "end") == 0) {
00971     if (expect != 0) {
00972         (thee->nprint)++;
00973         (thee->printnarg[idx])++;
00974         Vnm_print(0, "NOsh: Done parsing PRINT section\n");
00975         return 1;
00976     } else {
00977         Vnm_print(2, "NOsh_parsePRINT: Got premature END to PRINT!\n")
00978     }
00979     return 0;
00980 } else {
00981
00982     /* Grab a calculation ID */
00983     if ((sscanf(tok, "%d", &ti) == 1) &&
00984     (Vstring_isdigit(tok) == 1)) {
00985         if (expect == 0) {
00986             thee->printcalc[idx][thee->printnarg[idx]
00987 ] = ti-1;
00988             expect = 1;
00989         } else {
00990             Vnm_print(2, "NOsh_parsePRINT: Syntax error in PRINT \
00991 section while reading %s!\n", tok);
00992             return 0;
00993         }
00994     /* Grab addition operation */
00995     } else if (Vstring_strcasecmp(tok, "+") == 0) {
00996         if (expect == 1) {
00997             thee->printop[idx][thee->printnarg[idx]] =
00998                 0;
00999             (thee->printnarg[idx])++;
01000             expect = 0;
01001             if (thee->printnarg[idx] >= NOSH_MAXPOP
01002 ) {
01003                 Vnm_print(2, "NOsh_parsePRINT: Exceeded max number \
01004 (%d) of arguments for PRINT section!\n",
01005                 NOSH_MAXPOP);
01006                 return 0;
01007             }
01008         } else {
01009             Vnm_print(2, "NOsh_parsePRINT: Syntax error in PRINT \
01010 section while reading %s!\n", tok);
01011             return 0;
01012         }
01013     }
01014 }
```

```

01009             }
01100         /* Grab subtraction operation */
01101         } else if (Vstring_strcasecmp(tok, "-") == 0) {
01102             if (expect == 1) {
01103                 thee->printop[idx] [thee->printnarg[idx]] =
01104                     (thee->printnarg[idx])++;
01105                 expect = 0;
01106                 if (thee->printnarg[idx] >= NOSH_MAXPOP
01107             ) {
01108                 Vnm_print(2, "NOsh_parseREAD: Exceeded max number \
01109 (%d) of arguments for PRINT section!\n",
01110                 NOSH_MAXPOP);
01111                 return 0;
01112             }
01113             } else {
01114                 Vnm_print(2, "NOsh_parsePRINT: Syntax error in PRINT \
01115 section while reading %s!\n", tok);
01116                 return 0;
01117             }
01118             /* Grab a calculation name from elec ID */
01119         } else if (sscanf(tok, "%s", name) == 1) {
01120             if (expect == 0) {
01121                 for (ielec=0; ielec<thee->nelec; ielec++) {
01122                     if (Vstring_strcasecmp(thee->elecname[ielec]
01123 , name) == 0) {
01124                         thee->printcalc[idx][thee->printnarg[idx]] = ielec;
01125                         expect = 1;
01126                         break;
01127                     }
01128                     for (iapol=0; iapol<thee->napol; iapol++) {
01129                         if (Vstring_strcasecmp(thee->apolname[iapol]
01130 , name) == 0) {
01131                             thee->printcalc[idx][thee->printnarg[idx]] = iapol;
01132                             expect = 1;
01133                             break;
01134                         }
01135                     }
01136                 }
01137                 if (expect == 0) {
01138                     Vnm_print(2, "No ELEC or APOL statement has been named %s!\n",
01139                     name);
01140                     return 0;
01141                 }
01142             } else {
01143                 Vnm_print(2, "NOsh_parsePRINT: Syntax error in PRINT \
01144 section while reading %s!\n", tok);
01145                 return 0;
01146             }
01147         }
01148     }
01149     /* Got bad operation */
01150     } else {
01151         Vnm_print(2, "NOsh_parsePRINT: Undefined keyword %s while \
01152 parsing PRINT section!\n", tok);
01153         return 0;
01154     }
01155     } /* end parse token */
01156 }
01157 } /* end while */
01158
01159 /* We ran out of tokens! */
01160 VERROR1:
01161 Vnm_print(2, "NOsh_parsePRINT: Ran out of tokens while parsing PRINT \
01162 section!\n");
01163 return 0;
01164
01165 }
01166
01167 VPRIPRIVATE int NOsh_parseELEC(NOsh *thee, Vio *sock) {
01168     NOsh_calc *calc = VNULL;
01169     char tok[VMAX_BUFSIZE];
01170
01171     if (thee == VNULL) {
01172         Vnm_print(2, "NOsh_parseELEC: Got NULL thee!\n");
01173         return 0;
01174     }
01175     if (sock == VNULL) {
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01960
01961
01962
01963
01964
01965
01966
01967
01968
01969
01970
01971
01972
01973
01974
01975
01976
01977
01978
01979
01980
01981
01982
01983
01984
01985
01986
01987
01988
01989
01990
01991
01992
01993
01994
01995
01996
01997
01998
01999
02000
02001
02002
02003
02004
02005
02006
02007
02008
02009
02010
02011
02012
02013
02014
02015
02016
02017
02018
02019
02020
02021
02022
02023
02024
02025
02026
02027
02028
02029
02030
02031
02032
02033
02034
02035
02036
02037
02038
02039
02040
02041
02042
02043
02044
02045
02046
02047
02048
02049
02050
02051
02052
02053
02054
02055
02056
02057
02058
02059
02060
02061
02062
02063
02064
02065
02066
02067
02068
02069
02070
02071
02072
02073
02074
02075
02076
02077
02078
02079
02080
02081
02082
02083
02084
02085
02086
02087
02088
02089
02090
02091
02092
02093
02094
02095
02096
02097
02098
02099
02100
02101
02102
02103
02104
02105
02106
02107
02108
02109
02110
02111
02112
02113
02114
02115
02116
02117
02118
02119
02120
02121
02122
02123
02124
02125
02126
02127
02128
02129
02130
02131
02132
02133
02134
02135
02136
02137
02138
02139
02140
02141
02142
02143
02144
02145
02146
02147
02148
02149
02150
02151
02152
02153
02154
02155
02156
02157
02158
02159
02160
02161
02162
02163
02164
02165
02166
02167
02168
02169
02170
02171
02172
02173
02174
02175
02176
02177
02178
02179
02180
02181
02182
02183
02184
02185
02186
02187
02188
02189
02190
02191
02192
02193
02194
02195
02196
02197
02198
02199
02200
02201
02202
02203
02204
02205
02206
02207
02208
02209
02210
02211
02212
02213
02214
02215
02216
02217
02218
02219
02220
02221
02222
02223
02224
02225
02226
02227
02228
02229
02230
02231
02232
02233
02234
02235
02236
02237
02238
02239
02240
02241
02242
02243
02244
02245
02246
02247
02248
02249
02250
02251
02252
02253
02254
02255
02256
02257
02258
02259
02260
02261
02262
02263
02264
02265
02266
02267
02268
02269
02270
02271
02272
02273
02274
02275
02276
02277
02278
02279
02280
02281
02282
02283
02284
02285
02286
02287
02288
02289
02290
02291
02292
02293
02294
02295
02296
02297
02298
02299
02300
02301
02302
02303
02304
02305
02306
02307
02308
02309
02310
02311
02312
02313
02314
02315
02316
02317
02318
02319
02320
02321
02322
02323
02324
02325
02326
02327
02328
02329
02330
02331
02332
02333
02334
02335
02336
02337
02338
02339
02340
02341
02342
02343
02344
02345
02346
02347
02348
02349
02350
02351
02352
02353
02354
02355
02356
02357
02358
02359
02360
02361
02362
02363
02364
02365
02366
02367
02368
02369
02370
02371
02372
02373
02374
02375
02376
02377
02378
02379
02380
02381
02382
02383
02384
02385
02386
02387
02388
02389
02390
02391
02392
02393
02394
02395
02396
02397
02398
02399
02400
02401
02402
02403
02404
02405
02406
02407
02408
02409
02410
02411
02412
02413
02414
02415
02416
02417
02418
02419
02420
02421
02422
02423
02424
02425
02426
02427
02428
02429
02430
02431
02432
02433
02434
02435
02436
02437
02438
02439
02440
02441
02442
02443
02444
02445
02446
02447
02448
02449
02450
02451
02452
02453
02454
02455
02456
02457
02458
02459
02460
02461
02462
02463
02464
02465
02466
02467
02468
02469
02470
02471
02472
02473
02474
02475
02476
02477
02478
02479
02480
02481
02482
02483
02484
02485
02486
02487
02488
02489
02490
02491
02492
02493
02494
02495
02496
02497
02498
02499
02500
02501
02502
02503
02504
02505
02506
02507
02508
02509
02510
02511
02512
02513
02514
02515
02516
02517
02518
02519
02520
02521
02522
02523
02524
02525
02526
02527
02528
02529
02530
02531
02532
02533
02534
02535
02536
02537
02538
02539
02540
02541
02542
02543
02544
02545
02546
02547
02548
02549
02550
02551
02552
02553
02554
02555
02556
02557
02558
02559
02560
02561
02562
02563
02564
02565
02566
02567
02568
02569
02570
02571
02572
02573
02574
02575
02576
02577
02578
02579
02580
02581
02582
02583
02584
02585
02586
02587
02588
02589
02590
02591
02592
02593
02594
02595
02596
02597
02598
02599
02600
02601
02602
02603
02604
02605
02606
02607
02608
02609
02610
02611
02612
02613
02614
02615
02616
02617
02618
02619
02620
02621
02622
02623
02624
02625
02626
02627
02628
02629
02630
02631
02632
02633
02634
02635
02636
02637
02638
02639
02640
02641
02642
02643
02644
02645
02646
02647
02648
02649
02650
02651
02652
02653
02654
02655
02656
02657
02658
02659
02660
02661
02662
02663
02664
02665
02666
02667
02668
02669
02670
02671
02672
02673
02674
02675
02676
02677
02678
02679
02680
02681
02682
02683
02684
02685
02686
02687
02688
02689
02690
02691
02692
02693
02694
02695
02696
02697
02698
02699
02700
02701
02702
02703
02704
02705
02706
02707
02708
02709
02710
02711
02712
02713
02714
02715
02716
02717
02718
02719
02720
02721
02722
02723
02724
02725
02726
02727
02728
02729
02730
02731
02732
02733
02734
02735
02736
02737
02738
02739
02740
02741
02742
02743
02744
02745
02746
02747
02748
02749
02750
02751
02752
02753
02754
02755
02756
02757
02758
02759
02760
02761
02762
02763
02764
02765
02766
02767
02768
02769
02770
02771
02772
02773
02774
02775
02776
02777
02778
02779
02780
02781
02782
02783
02784
02785
02786
02787
02788
02789
02790
02791
02792
02793
02794
02795
02796
02797
02798
02799
02800
02801
02802
02803
02804
02805
02806
02807
02808
02809
02810
02811
02812
02813
02814
02815
02816
02817
02818
02819
02820
02821
02822
02823
02824
02825
02826
02827
02828
02829
02830
02831
02832
02833
02834
02835
02836
02837
02838
02839
02840
02841
02842
02843
02844
02845
02846
02847
02848
02849
02850
02851
02852
02853
02854
02855
02856
02857
02858
02859
02860
02861
02862
02863
02864
02865
02866
02867
02868
02869
02870
02871
02872
02873
02874
02875
02876
02877
02878
02879
02880
02881
02882
02
```

```

01086      Vnm_print(2, "NOsh_parseELEC: Got pointer to NULL socket!\n");
01087      return 0;
01088  }
01089
01090  if (thee->parsed) {
01091    Vnm_print(2, "NOsh_parseELEC: Already parsed an input file!\n");
01092    return 0;
01093  }
01094
01095  /* Get a pointer to the latest ELEC calc object and update the ELEC
01096 statement number */
01097  if (thee->nelec >= NOSH_MAXCALC) {
01098    Vnm_print(2, "NOsh: Too many electrostatics calculations in this \
01099 run!\n");
01100    Vnm_print(2, "NOsh: Current max is %d; ignoring this calculation\n",
01101      NOSH_MAXCALC);
01102    return 1;
01103  }
01104
01105  /* The next token HAS to be the method OR "name" */
01106 if (Vio_scanf(sock, "%s", tok) == 1) {
01107  if (Vstring_strcasecmp(tok, "name") == 0) {
01108    Vio_scanf(sock, "%s", tok);
01109    strncpy(thee->elecname[thee->nelec], tok, VMAX_ARGLEN)
01110  ;
01111  if (Vio_scanf(sock, "%s", tok) != 1) {
01112    Vnm_print(2, "NOsh_parseELEC: Ran out of tokens while reading \
01113 ELEC section!\n");
01114    return 0;
01115  }
01116  if (Vstring_strcasecmp(tok, "mg-manual") == 0) {
01117    thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_MG
01118  );
01119    calc = thee->elec[thee->nelec];
01120    (thee->nelec)++;
01121    calc->mparm->type = MCT_MANUAL;
01122  } else if (Vstring_strcasecmp(tok, "mg-auto") == 0) {
01123    thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_MG
01124  );
01125    calc = thee->elec[thee->nelec];
01126    (thee->nelec)++;
01127    calc->mparm->type = MCT_AUTO;
01128  } else if (Vstring_strcasecmp(tok, "mg-para") == 0) {
01129    thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_MG
01130  );
01131    calc = thee->elec[thee->nelec];
01132    (thee->nelec)++;
01133    calc->mparm->type = MCT_PARALLEL;
01134  } else if (Vstring_strcasecmp(tok, "mg-dummy") == 0) {
01135    thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_MG
01136  );
01137    calc = thee->elec[thee->nelec];
01138    calc->mparm->type = MCT_DUMMY;
01139    return NOsh_parseMG(thee, sock, calc);
01140  } else if (Vstring_strcasecmp(tok, "fe-manual") == 0) {
01141    thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_FEM
01142  );
01143    calc = thee->elec[thee->nelec];
01144    (thee->nelec)++;
01145    calc->femparm->type = FCT_MANUAL;
01146  } else {
01147    Vnm_print(2, "NOsh_parseELEC: The method (\\"mg\\" or \\"fem\\") or \
01148 \\"name\\" must be the first keyword in the ELEC section\n");
01149    return 0;
01150  }
01151  }
01152
01153  Vnm_print(2, "NOsh_parseELEC: Ran out of tokens while reading ELEC
01154 section!\n");
01155  return 0;
01156}
01157
01158 VPRIVATE int NOsh_parseAPOLAR(NOsh *thee, Vio *sock) {
01159

```

```

01160 NOsh_calc *calc = VNULL;
01161     char tok[VMAX_BUFSIZE];
01163
01164     if (thee == VNULL) {
01165         Vnm_print(2, "NOsh_parseAPOLAR: Got NULL thee!\n");
01166         return 0;
01167     }
01168
01169     if (sock == VNULL) {
01170         Vnm_print(2, "NOsh_parseAPOLAR: Got pointer to NULL socket!\n");
01171         return 0;
01172     }
01173
01174     if (thee->parsed) {
01175         Vnm_print(2, "NOsh_parseAPOLAR: Already parsed an input file!\n");
01176         return 0;
01177     }
01178
01179 /* Get a pointer to the latest ELEC calc object and update the ELEC
01180 statement number */
01181     if (thee->napol >= NOSH_MAXCALC) {
01182         Vnm_print(2, "NOsh: Too many non-polar calculations in this \
01183 run!\n");
01184         Vnm_print(2, "NOsh: Current max is %d; ignoring this calculation\n",
01185         NOSH_MAXCALC);
01186         return 1;
01187     }
01188
01189 /* The next token HAS to be the method OR "name" */
01190     if (Vio_scanf(sock, "%s", tok) == 1) {
01191         if (Vstring_strcasecmp(tok, "name") == 0) {
01192             Vio_scanf(sock, "%s", tok);
01193             strncpy(thee->apolname[thee->napol], tok, VMAX_ARGLEN)
01194 ;
01195     /* Parse the non-polar parameters */
01196     thee->apol[thee->napol] = NOsh_calc_ctor(NCT_APOL
01197 );
01197     calc = thee->apol[thee->napol];
01198     (thee->napol)++;
01199     return NOsh_parseAPOL(thee, sock, calc);
01200
01201     if (Vio_scanf(sock, "%s", tok) != 1) {
01202         Vnm_print(2, "NOsh_parseAPOLAR: Ran out of tokens while reading \
01203 APOLAR section!\n");
01204         return 0;
01205     }
01206 }
01207 }
01208
01209 return 1;
01210
01211 }
01212
01213 VPUBLIC int NOsh_setupElecCalc(
01214     NOsh *thee,
01215     Valist *alist[NOSH_MAXMOL]
01216     ) {
01217     int ielec, imol, i;
01218     NOsh_calc *elec = VNULL;
01219     MGparm *mgparm = VNULL;
01220     Valist *mymol = VNULL;
01221
01222     VASSERT(thee != VNULL);
01223     for (imol=0; imol<thee->nmol; imol++) {
01224         thee->alist[imol] = alist[imol];
01225     }
01226
01227
01228     for (ielec=0; ielec<(thee->nelec); ielec++) {
01229         /* Unload the calculation object containing the ELEC information */
01230         elec = thee->elec[ielec];
01231
01232         if (((thee->ndiel != 0) || (thee->nkappa != 0) ||
01233             (thee->ncharge != 0) || (thee->npot != 0)) &&
01234             (elec->pbeparm->calcrel != PCF_NO)) {
01235             Vnm_print(2, "NOsh_setupElecCalc: Calculation of forces disabled because
01236 surface \
01237 map is used!\n");
01238             elec->pbeparm->calcrel = PCF_NO;

```

```

01238 }
01239 /* Setup the calculation */
01240 switch (elec->calctype) {
01241 case NCT_MG:
01242   /* Center on the molecules, if requested */
01243   mgparm = elec->mgparm;
01244   VASSERT(mgparm != VNULL);
01245   if (elec->mgparm->cmeth == MCM_MOLECULE) {
01246     VASSERT(mgparm->centmol >= 0);
01247     VASSERT(mgparm->centmol < thee->nmol);
01248     mymol = thee->alist[mgparm->centmol];
01249     VASSERT(mymol != VNULL);
01250     for (i=0; i<3; i++) {
01251       mgparm->center[i] = mymol->center[i];
01252     }
01253   }
01254   if (elec->mgparm->fmeth == MCM_MOLECULE) {
01255     VASSERT(mgparm->fcentmol >= 0);
01256     VASSERT(mgparm->fcentmol < thee->nmol);
01257     mymol = thee->alist[mgparm->fcentmol];
01258     VASSERT(mymol != VNULL);
01259     for (i=0; i<3; i++) {
01260       mgparm->fcenter[i] = mymol->center[i];
01261     }
01262   }
01263   if (elec->mgparm->ccmeth == MCM_MOLECULE) {
01264     VASSERT(mgparm->ccentmol >= 0);
01265     VASSERT(mgparm->ccentmol < thee->nmol);
01266     mymol = thee->alist[mgparm->ccentmol];
01267     VASSERT(mymol != VNULL);
01268     for (i=0; i<3; i++) {
01269       mgparm->ccenter[i] = mymol->center[i];
01270     }
01271   }
01272   NOsh_setupCalcMG(thee, elec);
01273   break;
01274 case NCT_FEM:
01275   NOsh_setupCalcFEM(thee, elec);
01276   break;
01277 default:
01278   Vnm_print(2, "NOsh_setupCalc: Invalid calculation type (%d)!\n",
01279             elec->calctype);
01280   return 0;
01281 }
01282 /* At this point, the most recently-created NOsh_calc object should be the
01283 one we use for results for this ELEC statement. Assign it. */
01284 /* Associate ELEC statement with the calculation */
01285 thee->elec2calc[ielec] = thee->ncalc-1;
01286 Vnm_print(0, "NOsh_setupCalc: Mapping ELEC statement %d (%d) to \
01287 calculation %d (%d)\n", ielec, ielec+1, thee->elec2calc[ielec],
01288           thee->elec2calc[ielec]+1);
01289 }
01290
01291 return 1;
01292 }
01293
01294 }
01295
01296 VPUBLIC int NOsh_setupApolCalc(
01297   NOsh *thee,
01298   Valist *alist[NOSH_MAXMOL]
01299 )
01300 int iapol, imol;
01301 int doCalc = ACD_NO;
01302 NOsh_calc *calc = VNULL;
01303
01304 VASSERT(thee != VNULL);
01305 for (imol=0; imol<thee->nmol; imol++) {
01306   thee->alist[imol] = alist[imol];
01307 }
01308
01309 for (iapol=0; iapol<(thee->nopol); iapol++) {
01310   /* Unload the calculation object containing the APOL information */
01311   calc = thee->apol[iapol];
01312
01313   /* Setup the calculation */
01314   switch (calc->calctype) {
01315   case NCT_APOL:
01316     NOsh_setupCalcAPOL(thee, calc);
01317     doCalc = ACD_YES;
01318     break;

```

```

01319     default:
01320     Vnm_print(2, "NOsh_setupCalc: Invalid calculation type (%d)!\n", calc->
01321     calcType);
01322     return ACD_ERROR;
01323   }
01324   /* At this point, the most recently-created NOsh_calc object should be the
01325   one we use for results for this APOL statement. Assign it. */
01326   /* Associate APOL statement with the calculation */
01327   thee->apol2calc[iapol] = thee->nCalc-1;
01328   Vnm_print(0, "NOsh_setupCalc: Mapping APOL statement %d (%d) to calculation
01329   %d (%d)\n", iapol, iapol+1, thee->apol2calc[iapol], thee->apol2calc
01330   [iapol]+1);
01331 }
01332 if (doCalc == ACD_YES) {
01333   return ACD_YES;
01334 } else{
01335   return ACD_NO;
01336 }
01337 VPUBLIC int NOsh_parseMG(
01338   NOsh *thee,
01339   Vio *sock,
01340   NOsh_calc *elec
01341   ) {
01342
01343   char tok[VMAX_BUFSIZE];
01344   MGparm *mgparm = VNULL;
01345   PBEparm *pbeparm = VNULL;
01346   int rc;
01347
01348   /* Check the arguments */
01349   if (thee == VNULL) {
01350     Vnm_print(2, "NOsh: Got NULL thee!\n");
01351     return 0;
01352   }
01353   if (sock == VNULL) {
01354     Vnm_print(2, "NOsh: Got pointer to NULL socket!\n");
01355     return 0;
01356   }
01357   if (elec == VNULL) {
01358     Vnm_print(2, "NOsh: Got pointer to NULL elec object!\n");
01359     return 0;
01360   }
01361   mgparm = elec->mgparm;
01362   if (mgparm == VNULL) {
01363     Vnm_print(2, "NOsh: Got pointer to NULL mgparm object!\n");
01364     return 0;
01365   }
01366   pbeparm = elec->pbeparm;
01367   if (pbeparm == VNULL) {
01368     Vnm_print(2, "NOsh: Got pointer to NULL pbeparm object!\n");
01369     return 0;
01370   }
01371
01372   Vnm_print(0, "NOsh_parseMG: Parsing parameters for MG calculation\n");
01373
01374   /* Parallel stuff */
01375   if (mgparm->type == MCT_PARALLEL) {
01376     mgparm->proc_rank = thee->proc_rank;
01377     mgparm->proc_size = thee->proc_size;
01378     mgparm->setrank = 1;
01379     mgparm->setszie = 1;
01380   }
01381
01382
01383   /* Start snarfing tokens from the input stream */
01384   rc = 1;
01385   while (Vio_scanf(sock, "%s", tok) == 1) {
01386
01387     Vnm_print(0, "NOsh_parseMG: Parsing %s...\n", tok);
01388
01389     /* See if it's an END token */
01390     if (Vstring_strcasecmp(tok, "end") == 0) {
01391       mgparm->parsed = 1;
01392       pbeparm->parsed = 1;
01393       rc = 1;
01394       break;
01395     }
01396

```

```

01397 /* Pass the token through a series of parsers */
01398 rc = PBEparm_parseToken(pbeparm, tok, sock);
01399 if (rc == -1) {
01400 Vnm_print(0, "NOsh_parseMG: parsePBE error!\n");
01401 break;
01402 } else if (rc == 0) {
01403 /* Pass the token to the generic MG parser */
01404 rc = MGparm_parseToken(mgparm, tok, sock);
01405 if (rc == -1) {
01406 Vnm_print(0, "NOsh_parseMG: parseMG error!\n");
01407 break;
01408 } else if (rc == 0) {
01409 /* We ran out of parsers! */
01410 Vnm_print(2, "NOsh: Unrecognized keyword: %s\n", tok);
01411 break;
01412 }
01413 }
01414 }
01415
01416 /* Handle various errors arising in the token-snarfing loop -- these all
01417 just result in simple returns right now */
01418 if (rc == -1) return 0;
01419 if (rc == 0) return 0;
01420
01421 /* Check the status of the parameter objects */
01422 if ((MGparm_check(mgparm) == VRC_FAILURE) || (
01423 PBEparm_check(pbeparm))) {
01424 Vnm_print(2, "NOsh: MG parameters not set correctly!\n");
01425 return 0;
01426 }
01427 return 1;
01428 }
01429
01430 VPRIPRIVATE int NOsh_setupCalcMG(
01431     NOsh *thee,
01432     NOsh_calc *calc
01433 ) {
01434
01435 MGparm *mgparm = VNULL;
01436
01437 VASSERT(thee != VNULL);
01438 VASSERT(calc != VNULL);
01439 mgparm = calc->mgparm;
01440 VASSERT(mgparm != VNULL);
01441
01442
01443 /* Now we're ready to whatever sorts of post-processing operations that are
01444 necessary for the various types of calculations */
01445 switch (mgparm->type) {
01446 case MCT_MANUAL:
01447     return NOsh_setupCalcMGMANUAL(thee, calc);
01448 case MCT_DUMMY:
01449     return NOsh_setupCalcMGMANUAL(thee, calc);
01450 case MCT_AUTO:
01451     return NOsh_setupCalcMGAUTO(thee, calc);
01452 case MCT_PARALLEL:
01453     return NOsh_setupCalcMGPARA(thee, calc);
01454 default:
01455     Vnm_print(2, "NOsh_setupCalcMG: undefined MG calculation type (%d)!\n",
01456             mgparm->type);
01457     return 0;
01458 }
01459
01460 /* Shouldn't get here */
01461 return 0;
01462 }
01463
01464 VPRIPRIVATE int NOsh_setupCalcFEM(
01465     NOsh *thee,
01466     NOsh_calc *calc
01467 ) {
01468
01469 VASSERT(thee != VNULL);
01470 VASSERT(calc != VNULL);
01471 VASSERT(calc->femparm != VNULL);
01472
01473 /* Now we're ready to whatever sorts of post-processing operations that are
01474 * necessary for the various types of calculations */
01475 switch (calc->femparm->type) {
01476 case FCT_MANUAL:

```

```

01477     return NOsh_setupCalcFEMANUAL(thee, calc);
01478   default:
01479     Vnm_print(2, "NOsh_parseFEM:  unknown calculation type (%d)!\n",
01480               calc->femparm->type);
01481     return 0;
01482   }
01483
01484 /* Shouldn't get here */
01485   return 0;
01486 }
01487
01488
01489 VPRIVATE int NOsh_setupCalcMGMANUAL(
01490   NOsh *thee,
01491   NOsh_calc *elec
01492 ) {
01493
01494   MGparm *mgparm = VNULL;
01495   PBEparm *pbeparm = VNULL;
01496   NOsh_calc *calc = VNULL;
01497
01498   if (thee == VNULL) {
01499     Vnm_print(2, "NOsh_setupCalcMGMANUAL:  Got NULL thee!\n");
01500     return 0;
01501   }
01502   if (elec == VNULL) {
01503     Vnm_print(2, "NOsh_setupCalcMGMANUAL:  Got NULL calc!\n");
01504     return 0;
01505   }
01506   mgparm = elec->mgparm;
01507   if (mgparm == VNULL) {
01508     Vnm_print(2, "NOsh_setupCalcMGMANUAL:  Got NULL mgparm -- was this
calculation \
01509 set up?\n");
01510     return 0;
01511   }
01512   pbeparm = elec->pbeparm;
01513   if (pbeparm == VNULL) {
01514     Vnm_print(2, "NOsh_setupCalcMGMANUAL:  Got NULL pbeparm -- was this
calculation \
01515 set up?\n");
01516     return 0;
01517   }
01518
01519 /* Set up missing MG parameters */
01520   if (mgparm->setgrid == 0) {
01521     VASSERT(mgparm->setglen);
01522     mgparm->grid[0] = mgparm->glen[0]/((double)(mgparm->dime[0]-1));
01523     mgparm->grid[1] = mgparm->glen[1]/((double)(mgparm->dime[1]-1));
01524     mgparm->grid[2] = mgparm->glen[2]/((double)(mgparm->dime[2]-1));
01525   }
01526   if (mgparm->setglen == 0) {
01527     VASSERT(mgparm->setgrid);
01528     mgparm->glen[0] = mgparm->grid[0]*((double)(mgparm->dime[0]-1));
01529     mgparm->glen[1] = mgparm->grid[1]*((double)(mgparm->dime[1]-1));
01530     mgparm->glen[2] = mgparm->grid[2]*((double)(mgparm->dime[2]-1));
01531   }
01532
01533 /* Check to see if he have any room left for this type of calculation, if
01534 so: set the calculation type, update the number of calculations of this type,
01535 and parse the rest of the section */
01536   if (thee->nalc >= NOSH_MAXCALC) {
01537     Vnm_print(2, "NOsh:  Too many calculations in this run!\n");
01538     Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
01539               NOSH_MAXCALC);
01540     return 0;
01541   }
01542
01543   /* Get the next calculation object and increment the number of calculations
*/
01544   thee->calc[thee->nalc] = NOsh_calc_ctor(NCT_MG);
01545   calc = thee->calc[thee->nalc];
01546   (thee->nalc)++;
01547
01548
01549
01550 /* Copy over contents of ELEC */
01551 NOsh_calc_copy(calc, elec);
01552
01553

```

```

01554     return 1;
01555 }
01556
01557 VPUBLIC int NOsh_setupCalcMGAUTO(
01558     NOsh *thee,
01559     NOsh_calc *elec
01560 ) {
01561
01562     NOsh_calc *calcf = VNULL;
01563     NOsh_calc *calcc = VNULL;
01564     double fgrid[3], cgrid[3];
01565     double d[3], minf[3], maxf[3], minc[3], maxc[3];
01566     double redfrac, redrat[3], td;
01567     int ifocus, nfocus, tnfocus[3];
01568     int j;
01569     int icalc;
01570     int dofix;
01571
01572 /* A comment about the coding style in this function. I use lots and lots
01573 and lots of pointer referencing. I could (and probably should) save
01574 these in temporary variables. However, since there are so many MGparm,
01575 etc. and NOsh_calc, etc. objects running around in this function, the
01576 current scheme is easiest to debug. */
01577
01578 if (thee == VNULL) {
01579     Vnm_print(2, "NOsh_setupCalcMGAUTO: Got NULL thee!\n");
01580     return 0;
01581 }
01582 if (elec == VNULL) {
01583     Vnm_print(2, "NOsh_setupCalcMGAUTO: Got NULL elec!\n");
01584     return 0;
01585 }
01586 if (elec->mgparm == VNULL) {
01587     Vnm_print(2, "NOsh_setupCalcMGAUTO: Got NULL mgparm!\n");
01588     return 0;
01589 }
01590 if (elec->pbeparm == VNULL) {
01591     Vnm_print(2, "NOsh_setupCalcMGAUTO: Got NULL pbeparm!\n");
01592     return 0;
01593 }
01594 }
01595
01596 Vnm_print(0, "NOsh_setupCalcMGAUTO(%s, %d): coarse grid center = %g %g %g\n",
01597     __FILE__, __LINE__,
01598     elec->mgparm->ccenter[0],
01599     elec->mgparm->ccenter[1],
01600     elec->mgparm->ccenter[2]);
01601 Vnm_print(0, "NOsh_setupCalcMGAUTO(%s, %d): fine grid center = %g %g %g\n",
01602     __FILE__, __LINE__,
01603     elec->mgparm->fccenter[0],
01604     elec->mgparm->fccenter[1],
01605     elec->mgparm->fccenter[2]);
01606
01607 /* Calculate the grid spacing on the coarse and fine levels */
01608 for (j=0; j<3; j++) {
01609     cgrid[j] = (elec->mgparm->crlen[j])/((double)(elec->mgparm->
01610         dime[j]-1));
01611     fgrid[j] = (elec->mgparm->flen[j])/((double)(elec->mgparm->
01612         dime[j]-1));
01613     d[j] = elec->mgparm->fccenter[j] - elec->mgparm->ccenter
01614         [j];
01615 }
01616 Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): Coarse grid spacing = %g, %g, %g
01617 \n",
01618     __FILE__, __LINE__, cgrid[0], cgrid[1], cgrid[2]);
01619 Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): Fine grid spacing = %g, %g, %g\n
01620 ",
01621     __FILE__, __LINE__, fgrid[0], fgrid[1], fgrid[2]);
01622 Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): Displacement between fine and \
01623 coarse grids = %g, %g, %g\n", __FILE__, __LINE__, d[0], d[1], d[2]);
01624
01625 /* Now calculate the number of focusing levels, never reducing the grid
01626 spacing by more than redfrac at each level */
01627 for (j=0; j<3; j++) {
01628     if (fgrid[j]/cgrid[j] < VREDFRAC) {
01629         redfrac = fgrid[j]/cgrid[j];
01630         td = log(redfrac)/log(VREDFRAC);
01631         tnfocus[j] = (int)ceil(td) + 1;
01632     } else tnfocus[j] = 2;
01633 }
01634 nfocus = VMAX2(VMAX2(tnfocus[0], tnfocus[1]), tnfocus[2]);

```

```

01630
01631 /* Now set redrat to the actual value by which the grid spacing is reduced
01632 at each level of focusing */
01633 for (j=0; j<3; j++) {
01634   redrat[j] = VPOW((fgrid[j]/cgrid[j]), 1.0/((double)nfocus-1.0));
01635 }
01636 Vnm_print(0, "NOSH: %d levels of focusing with %g, %g, %g reductions\n",
01637   nfocus, redrat[0], redrat[1], redrat[2]);
01638
01639 /* Now that we know how many focusing levels to use, we're ready to set up
01640 the parameter objects */
01641 if (nfocus > (NOSH_MAXCALC-(thee->ncalc))) {
01642   Vnm_print(2, "NOSH: Require more calculations than max (%d)!\n",
01643             NOSH_MAXCALC);
01644   return 0;
01645 }
01646
01647 for (ifocus=0; ifocus<nfocus; ifocus++) {
01648
01649 /* Generate the new calc object */
01650 icalc = thee->nalc;
01651 thee->calc[icalc] = NOsh_calc_ctor(NCT_MG);
01652 (thee->nalc)++;
01653
01654 /* This is the _current_ NOsh_calc object */
01655 calcf = thee->calc[icalc];
01656 /* This is the _previous_ Nosh_calc object */
01657 if (ifocus != 0) {
01658   calcc = thee->calc[icalc-1];
01659 } else {
01660   calcc = VNULL;
01661 }
01662
01663 /* Copy over most of the parameters from the ELEC object */
01664 NOsh_calc_copy(calcf, elec);
01665
01666 /* Set up the grid lengths and spacings */
01667 if (ifocus == 0) {
01668   for (j=0; j<3; j++) {
01669     calcf->mgparm->grid[j] = cgrid[j];
01670     calcf->mgparm->glen[j] = elec->mgparm->cglen[j];
01671   }
01672 } else {
01673   for (j=0; j<3; j++) {
01674     calcf->mgparm->grid[j] = redrat[j]*(calcc->mgparm->grid
01675 [j]);
01676     calcf->mgparm->glen[j] = redrat[j]*(calcc->mgparm
01677 ->glen[j]);
01678   }
01679   calcf->mgparm->setgrid = 1;
01680   calcf->mgparm->setglen = 1;
01681
01682 /* Get centers and centering method from coarse and fine meshes */
01683 if (ifocus == 0) {
01684   calcf->mgparm->cmeth = elec->mgparm->ccmeth;
01685   calcf->mgparm->centmol = elec->mgparm->ccentmol;
01686   for (j=0; j<3; j++) {
01687     calcf->mgparm->center[j] = elec->mgparm->ccenter[j];
01688   }
01689 } else if (ifocus == (nfocus-1)) {
01690   calcf->mgparm->cmeth = elec->mgparm->fcmeth;
01691   calcf->mgparm->centmol = elec->mgparm->fcntmol;
01692   for (j=0; j<3; j++) {
01693     calcf->mgparm->center[j] = elec->mgparm->fcenter[j];
01694   }
01695 } else {
01696   calcf->mgparm->cmeth = MCM_FOCUS;
01697   /* TEMPORARILY move the current grid center
01698 to the fine grid center. In general, this will move portions of
01699 the current mesh off the immediately-coarser mesh. We'll fix that
01700 in the next step. */
01701   for (j=0; j<3; j++) {
01702     calcf->mgparm->center[j] = elec->mgparm->fcenter[j];
01703   }
01704
01705

```

```

01706 /* As mentioned above, it is highly likely that the previous "jump"
01707   to the fine grid center put portions of the current mesh off the
01708   previous (coarser) mesh. Fix this by displacing the current mesh
01709   back onto the previous coarser mesh. */
01710 if (ifocus != 0) {
01711   Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): starting mesh \
01712 repositioning.\n", __FILE__, __LINE__);
01713   Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): coarse mesh center = \
01714 %g %g %g\n", __FILE__, __LINE__,
01715   calcc->mgparm->center[0],
01716   calcc->mgparm->center[1],
01717   calcc->mgparm->center[2]);
01718   Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): coarse mesh upper corner = \
01719 %g %g %g\n", __FILE__, __LINE__,
01720   calcc->mgparm->center[0]+0.5*(calcc->mgparm->glen[
01721   0]),
01722   calcc->mgparm->center[1]+0.5*(calcc->mgparm->glen
01723   [1]),
01724   calcc->mgparm->center[2]+0.5*(calcc->mgparm->glen
01725   [2]));
01726   Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): coarse mesh lower corner = \
01727 %g %g %g\n", __FILE__, __LINE__,
01728   calcc->mgparm->center[0]-0.5*(calcc->mgparm->glen[
01729   0]),
01730   calcc->mgparm->center[1]-0.5*(calcc->mgparm->glen
01731   [1]),
01732   calcc->mgparm->center[2]-0.5*(calcc->mgparm->glen
01733   [2]));
01734   Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): initial fine mesh upper corner \
01735 = \
01736 %g %g %g\n", __FILE__, __LINE__,
01737   calcf->mgparm->center[0]+0.5*(calcf->mgparm->glen[
01738   0]),
01739   calcf->mgparm->center[1]+0.5*(calcf->mgparm->glen
01740   [1]),
01741   calcf->mgparm->center[2]+0.5*(calcf->mgparm->glen
01742   [2]));
01743   Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): initial fine mesh lower corner \
01744 = \
01745 %g %g %g\n", __FILE__, __LINE__,
01746   calcf->mgparm->center[0]-0.5*(calcf->mgparm->glen[
01747   0]),
01748   calcf->mgparm->center[1]-0.5*(calcf->mgparm->glen
01749   [1]),
01750   calcf->mgparm->center[2]-0.5*(calcf->mgparm->glen
01751   [2]));
01752   for (j=0; j<3; j++) {
01753     /* Check if we've fallen off of the lower end of the mesh */
01754     dofix = 0;
01755     minf[j] = calcf->mgparm->center[j]
01756     - 0.5*(calcf->mgparm->glen[j]);
01757     minc[j] = calcc->mgparm->center[j]
01758     - 0.5*(calcc->mgparm->glen[j]);
01759     d[j] = minc[j] - minf[j];
01760     if (d[j] >= VSMALL) {
01761       if (ifocus == (nfocus-1)) {
01762         Vnm_print(2, "NOsh_setupCalcMGAUTO: Error! Finest \
01763 mesh has fallen off the coarser meshes!\n");
01764         Vnm_print(2, "NOsh_setupCalcMGAUTO: difference in min %d-\
01765 direction = %g\n", j, d[j]);
01766         Vnm_print(2, "NOsh_setupCalcMGAUTO: min fine = %g %g %g\n",
01767         minf[0], minf[1], minf[2]);
01768         Vnm_print(2, "NOsh_setupCalcMGAUTO: min coarse = %g %g %g\n",
01769         minc[0], minc[1], minc[2]);
01770         VASSERT(0);
01771       } else {
01772         Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): ifocus = %d, \
01773 fixing mesh min violation (%g in %d-direction).\n", __FILE__, __LINE__, ifocus,
01774         d[j], j);
01775         calcf->mgparm->center[j] += d[j];
01776         dofix = 1;
01777       }
01778     }
01779     /* Check if we've fallen off of the upper end of the mesh */
01780     maxf[j] = calcf->mgparm->center[j] \
01781     + 0.5*(calcf->mgparm->glen[j]);
01782     maxc[j] = calcc->mgparm->center[j] \
01783     + 0.5*(calcc->mgparm->glen[j]);
01784     d[j] = maxf[j] - maxc[j];
01785     if (d[j] >= VSMALL) {

```

```

01772     if (ifocus == (nfocus-1)) {
01773         Vnm_print(2, "NOsh_setupCalcMGAUTO: Error! Finest \
01774 mesh has fallen off the coarser meshes!\n");
01775         Vnm_print(2, "NOsh_setupCalcMGAUTO: difference in %d\
01776 direction = %g\n", j, d[j]);
01777         VASSERT(0);
01778     } else {
01779         /* If we already fixed the lower boundary and we now need
01780         to fix the upper boundary, we have a serious problem. */
01781         if (dofix) {
01782             Vnm_print(2, "NOsh_setupCalcMGAUTO: Error! Both \
01783 ends of the finer mesh do not fit in the bigger mesh!\n");
01784             VASSERT(0);
01785         }
01786         Vnm_print(0, "NOsh_setupCalcMGAUTO(%s, %d): ifocus = %d, \
01787 fixing mesh max violation (%g in %d-direction).\n", __FILE__, __LINE__, ifocus,
01788
01789             d[j], j);
01790         calcf->mgparm->center[j] -= d[j];
01791         dofisx = 1;
01792     }
01793 }
01794 Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): final fine mesh upper corner =
01795 \n%g %g %g\n", __FILE__, __LINE__,
01796         calcf->mgparm->center[0]+0.5*(calcf->mgparm->glen[0]),
01797         calcf->mgparm->center[1]+0.5*(calcf->mgparm->glen[1]),
01798         calcf->mgparm->center[2]+0.5*(calcf->mgparm->glen[2]));
01799 Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d): final fine mesh lower corner =
01800 \n%g %g %g\n", __FILE__, __LINE__,
01801         calcf->mgparm->center[0]-0.5*(calcf->mgparm->glen[0]),
01802         calcf->mgparm->center[1]-0.5*(calcf->mgparm->glen[1]),
01803         calcf->mgparm->center[2]-0.5*(calcf->mgparm->glen[2]));
01804 }
01805 /* Finer levels have focusing boundary conditions */
01806 if (ifocus != 0) calcf->pbeparm->bclf1 = BCFL_FOCUS;
01807
01808 /* Only the finest level handles I/O and needs to worry about disjoint
01809 partitioning */
01810 if (ifocus != (nfocus-1)) calcf->pbeparm->numwrite = 0;
01811
01812 /* Reset boundary flags for everything except parallel focusing */
01813 if (calcf->mgparm->type != MCT_PARALLEL) {
01814     Vnm_print(0, "NOsh_setupMGAUTO: Resetting boundary flags\n");
01815     for (j=0; j<6; j++) calcf->mgparm->partDisjOwnSide[j] =
01816         0;
01817     for (j=0; j<3; j++) {
01818         calcf->mgparm->partDisjCenter[j] = 0;
01819         calcf->mgparm->partDisjLength[j] = calcf->mgparm
01820             ->glen[j];
01821     }
01822
01823     calcf->mgparm->parsed = 1;
01824 }
01825
01826
01827 return 1;
01828 }
01829 }
01830
01831 /* Author: Nathan Baker and Todd Dolinsky */
01832 VPUBLIC int NOsh_setupCalcMGPARA(
01833     NOsh *thee,
01834     NOsh_calc *elec
01835 ) {
01836
01837     /* NEW (25-Jul-2006): This code should produce modify the ELEC statement
01838     and pass it on to MGAUTO for further processing. */
01839
01840     MGparm *mgparm = VNULL;
01841     double ofrac;

```

```

01842 double hx, hy, hzed;
01843 double xofrac, yofrac, zofrac;
01844 int rank, size, npx, npy, nproc, ip, jp, kp;
01845 int xeffGlob, yeffGlob, zeffGlob, xDisj, yDisj, zDisj;
01846 int xigminDisj, xigmaxDisj, yigminDisj, yigmaxDisj, zigminDisj, zigmaxDisj;
01847 int xigminOlap, xigmaxOlap, yigminOlap, yigmaxOlap, zigminOlap, zigmaxOlap;
01848 int xOlapReg, yOlapReg, zOlapReg;
01849 double xlenDisj, ylenDisj, zlenDisj;
01850 double xcentDisj, ycentDisj, zcentDisj;
01851 double xcentOlap, ycentOlap, zcentOlap;
01852 double xlenOlap, ylenOlap, zlenOlap;
01853 double xminOlap, xmaxOlap, yminOlap, ymaxOlap, zminOlap, zmaxOlap;
01854 double xminDisj, xmaxDisj, yminDisj, ymaxDisj, zminDisj, zmaxDisj;
01855 double xcent, ycent, zcent;
01856
01857 /* Grab some useful variables */
01858 VASSERT(thee != VNULL);
01859 VASSERT(elec != VNULL);
01860 mgparm = elec->mgparm;
01861 VASSERT(mgparm != VNULL);
01862
01863 /* Grab some useful variables */
01864 ofrac = mgparm->ofrac;
01865 npx = mgparm->pdime[0];
01866 npy = mgparm->pdime[1];
01867 npz = mgparm->pdime[2];
01868 nproc = npx*npy*npz;
01869
01870 /* If this is not an asynchronous calculation, then we need to make sure we
01871 have all the necessary MPI information */
01872 if (mgparm->setasync == 0) {
01873
01874 #ifndef HAVE_MPI_H
01875
01876     Vnm_tprint(2, "NOsh_setupCalcMGPARA: Oops! You're trying to perform \
01877 an 'mg-para' (parallel) calculation\n");
01878     Vnm_tprint(2, "NOsh_setupCalcMGPARA: with a version of APBS that
01879 wasn't \
01880 compiled with MPI!\n");
01881     Vnm_tprint(2, "NOsh_setupCalcMGPARA: Perhaps you meant to use the \
01882 'async' flag?\n");
01883     Vnm_tprint(2, "NOsh_setupCalcMGPARA: Bailing out!\n");
01884
01885     return 0;
01886 #endif
01887
01888 rank = thee->proc_rank;
01889 size = thee->proc_size;
01890 Vnm_print(0, "NOsh_setupCalcMGPARA: Hello from processor %d of %d\n",
01891 size);
01892
01893 /* Check to see if we have too many processors. If so, then simply set
01894 this processor to duplicating the work of processor 0. */
01895 if (rank > (nproc-1)) {
01896     Vnm_print(2, "NOsh_setupMGPARA: There are more processors available than\
01897 the %d you requested.\n", nproc);
01898     Vnm_print(2, "NOsh_setupMGPARA: Eliminating processor %d\n", rank);
01899     thee->bogus = 1;
01900     rank = 0;
01901 }
01902
01903 /* Check to see if we have too few processors. If so, this is a fatal
01904 error. */
01905 if (size < nproc) {
01906     Vnm_print(2, "NOsh_setupMGPARA: There are too few processors (%d) to \
01907 satisfy requirements (%d)\n", size, nproc);
01908     return 0;
01909 }
01910
01911 Vnm_print(0, "NOsh_setupMGPARA: Hello (again) from processor %d of %d\n",
01912 rank, size);
01913
01914 } else { /* Setting up for an asynchronous calculation. */
01915
01916 rank = mgparm->async;
01917
01918 thee->ispara = 1;
01919 thee->proc_rank = rank;
01920
01921 /* Check to see if the async id is greater than the number of

```

```

01922     * processors. If so, this is a fatal error. */
01923     if (rank > (nproc-1)) {
01924         Vnm_print(2, "NOsh_setupMGPARA: The processor id you requested
01925 (%d) \
01926 is not within the range of processors available (0-%d)\n", rank, (nproc-1));
01927         return 0;
01928     }
01929
01930     /* Calculate the processor's coordinates in the processor grid */
01931 kp = (int)floor(rank/(npx*npy));
01932 jp = (int)floor((rank-kp*npx*npy)/npx);
01933 ip = rank - kp*npx*npy - jp*npx;
01934 Vnm_print(0, "NOsh_setupMGPARA: Hello world from PE (%d, %d, %d)\n",
01935     ip, jp, kp);
01936
01937     /* Calculate effective overlap fractions for uneven processor distributions */
01938 if (npx == 1) xofrac = 0.0;
01939 else xofrac = ofrac;
01940 if (npy == 1) yofrac = 0.0;
01941 else yofrac = ofrac;
01942 if (npz == 1) zofrac = 0.0;
01943 else zofrac = ofrac;
01944
01945     /* Calculate the global grid size and spacing */
01946 xDisj = (int)VFLLOOR(mgparm->dime[0]/(1 + 2*xofrac) + 0.5);
01947 xeffGlob = npx*xDisj;
01948 hx = mgparm->fglen[0]/(double)(xeffGlob-1);
01949 yDisj = (int)VFLLOOR(mgparm->dime[1]/(1 + 2*yofrac) + 0.5);
01950 yeffGlob = npy*yDisj;
01951 hy = mgparm->fglen[1]/(double)(yeffGlob-1);
01952 zDisj = (int)VFLLOOR(mgparm->dime[2]/(1 + 2*zofrac) + 0.5);
01953 zeffGlob = npz*zDisj;
01954 hzed = mgparm->fglen[2]/(double)(zeffGlob-1);
01955 Vnm_print(0, "NOsh_setupMGPARA: Global Grid size = (%d, %d, %d)\n",
01956     xeffGlob, yeffGlob, zeffGlob);
01957 Vnm_print(0, "NOsh_setupMGPARA: Global Grid Spacing = (%.3f, %.3f, %.3f)\n
",
01958     hx, hy, hzed);
01959 Vnm_print(0, "NOsh_setupMGPARA: Processor Grid Size = (%d, %d, %d)\n",
01960     xDisj, yDisj, zDisj);
01961
01962     /* Calculate the maximum and minimum processor grid points */
01963 xigminDisj = ip*xDisj;
01964 xigmaxDisj = xigminDisj + xDisj - 1;
01965 yigminDisj = jp*yDisj;
01966 yigmaxDisj = yigminDisj + yDisj - 1;
01967 zigminDisj = kp*zDisj;
01968 zigmaxDisj = zigminDisj + zDisj - 1;
01969 Vnm_print(0, "NOsh_setupMGPARA: Min Grid Points for this proc. (%d, %d,
01970     %d)\n",
01971     xigminDisj, yigminDisj, zigminDisj);
01972 Vnm_print(0, "NOsh_setupMGPARA: Max Grid Points for this proc. (%d, %d,
01973     %d)\n",
01974     xigmaxDisj, yigmaxDisj, zigmaxDisj);
01975
01976     /* Calculate the disjoint partition length and center displacement */
01977 xminDisj = VMAX2(hx*(xigminDisj-0.5), 0.0);
01978 xmaxDisj = VMIN2(hx*(xigmaxDisj+0.5), mgparm->fglen[0]);
01979 xlenDisj = xmaxDisj - xminDisj;
01980 yminDisj = VMAX2(hy*(yigminDisj-0.5), 0.0);
01981 ymaxDisj = VMIN2(hy*(yigmaxDisj+0.5), mgparm->fglen[1]);
01982 ylenDisj = ymaxDisj - yminDisj;
01983 zminDisj = VMAX2(hzed*(zigminDisj-0.5), 0.0);
01984 zmaxDisj = VMIN2(hzed*(zigmaxDisj+0.5), mgparm->fglen[2]);
01985 zlenDisj = zmaxDisj - zminDisj;
01986
01987 xcent = 0.5*mgparm->fglen[0];
01988 ycent = 0.5*mgparm->fglen[1];
01989 zcent = 0.5*mgparm->fglen[2];
01990
01991 xcentDisj = xminDisj + 0.5*xlenDisj - xcent;
01992 ycentDisj = yminDisj + 0.5*ylenDisj - ycent;
01993 zcentDisj = zminDisj + 0.5*zlenDisj - zcent;
01994 if (VABS(xcentDisj) < VSMALL) xcentDisj = 0.0;
01995 if (VABS(ycentDisj) < VSMALL) ycentDisj = 0.0;
01996 if (VABS(zcentDisj) < VSMALL) zcentDisj = 0.0;
01997
01998 Vnm_print(0, "NOsh_setupMGPARA: Disj part length = (%g, %g, %g)\n",
01999     xlenDisj, ylenDisj, zlenDisj);

```

```

01999     Vnm_print(0, "NOsh_setupMGPARA:  Disj part center displacement = (%g, %g,
02000             %g)\n",
02001             xcentDisj, ycentDisj, zcentDisj);
02002
02003     /* Calculate the overlapping partition length and center displacement */
02004     xOlapReg = 0;
02005     yOlapReg = 0;
02006     zOlapReg = 0;
02007     if (npx != 1) xOlapReg = (int)VFLLOOR(xofrac*mgparm->fglen[0]/npx
02008 /hx + 0.5) + 1;
02009     if (npy != 1) yOlapReg = (int)VFLLOOR(yofrac*mgparm->fglen[1]/npy
02010 /hy + 0.5) + 1;
02011     if (npz != 1) zOlapReg = (int)VFLLOOR(zofrac*mgparm->fglen[2]/npz
02012 /hzed + 0.5) + 1;
02013
02014     Vnm_print(0, "NOsh_setupMGPARA:  No. of Grid Points in Overlap (%d, %d, %d)
02015             \n",
02016             xOlapReg, yOlapReg, zOlapReg);
02017
02018     if (ip == 0) xigminOlap = 0;
02019     else if (ip == (npx - 1)) xigminOlap = xeffGlob - mgparm->dime[0];
02020     else xigminOlap = xigminDisj - xOlapReg;
02021     xigmaxOlap = xigminOlap + mgparm->dime[0] - 1;
02022
02023     if (jp == 0) yigminOlap = 0;
02024     else if (jp == (npy - 1)) yigminOlap = yeffGlob - mgparm->dime[1];
02025     else yigminOlap = yigminDisj - yOlapReg;
02026     yigmaxOlap = yigminOlap + mgparm->dime[1] - 1;
02027
02028     Vnm_print(0, "NOsh_setupMGPARA:  Min Grid Points with Overlap (%d, %d, %d)
02029             \n",
02030             xigminOlap, yigminOlap, zigminOlap);
02031     Vnm_print(0, "NOsh_setupMGPARA:  Max Grid Points with Overlap (%d, %d, %d)
02032             \n",
02033             xigmaxOlap, yigmaxOlap, zigmaxOlap);
02034
02035     xminOlap = hx * xigminOlap;
02036     xmaxOlap = hx * xigmaxOlap;
02037     yminOlap = hy * yigminOlap;
02038     ymaxOlap = hy * yigmaxOlap;
02039     zminOlap = hzed * zigminOlap;
02040     zmaxOlap = hzed * zigmaxOlap;
02041
02042     xlenOlap = xmaxOlap - xminOlap;
02043     ylenOlap = ymaxOlap - yminOlap;
02044     zlenOlap = zmaxOlap - zminOlap;
02045
02046     xcentOlap = (xminOlap + 0.5*xlenOlap) - xcent;
02047     ycentOlap = (yminOlap + 0.5*ylenOlap) - ycent;
02048     zcentOlap = (zminOlap + 0.5*zlenOlap) - zcent;
02049     if (VABS(xcentOlap) < VSMALL) xcentOlap = 0.0;
02050     if (VABS(ycentOlap) < VSMALL) ycentOlap = 0.0;
02051     if (VABS(zcentOlap) < VSMALL) zcentOlap = 0.0;
02052
02053     Vnm_print(0, "NOsh_setupMGPARA:  Olap part length = (%g, %g, %g)\n",
02054             xlenOlap, ylenOlap, zlenOlap);
02055     Vnm_print(0, "NOsh_setupMGPARA:  Olap part center displacement = (%g, %g,
02056             %g)\n",
02057             xcentOlap, ycentOlap, zcentOlap);
02058
02059     /* Calculate the boundary flags:
02060     Flags are set to 1 when another processor is present along the boundary
02061     Flags are otherwise set to 0. */
02062
02063     if (ip == 0) mgparm->partDisjOwnSide[VAPBS_LEFT] =
0;
02064     else mgparm->partDisjOwnSide[VAPBS_LEFT] = 1;
02065     if (ip == (npx-1)) mgparm->partDisjOwnSide[VAPBS_RIGHT]
0] = 0;
02066     else mgparm->partDisjOwnSide[VAPBS_RIGHT] = 1;
02067     if (jp == 0) mgparm->partDisjOwnSide[VAPBS_BACK] = 0;
02068     else mgparm->partDisjOwnSide[VAPBS_BACK] = 1;
02069     if (jp == (npy-1)) mgparm->partDisjOwnSide[VAPBS_FRONT]
0] = 0;
02070     else mgparm->partDisjOwnSide[VAPBS_FRONT] = 1;

```

```

02069 if (kp == 0) mgparm->partDisjOwnSide[VAPBS_DOWN] = 0;
02070 else mgparm->partDisjOwnSide[VAPBS_DOWN] = 1;
02071 if (kp == (npz-1)) mgparm->partDisjOwnSide[VAPBS_UP] =
0;
02072 else mgparm->partDisjOwnSide[VAPBS_UP] = 1;
02073
02074 Vnm_print(0, "Nosh_setupMGPARA: partDisjOwnSide[LEFT] = %d\n",
02075     mgparm->partDisjOwnSide[VAPBS_LEFT]);
02076 Vnm_print(0, "Nosh_setupMGPARA: partDisjOwnSide[RIGHT] = %d\n",
02077     mgparm->partDisjOwnSide[VAPBS_RIGHT]);
02078 Vnm_print(0, "Nosh_setupMGPARA: partDisjOwnSide[FRONT] = %d\n",
02079     mgparm->partDisjOwnSide[VAPBS_FRONT]);
02080 Vnm_print(0, "Nosh_setupMGPARA: partDisjOwnSide[BACK] = %d\n",
02081     mgparm->partDisjOwnSide[VAPBS_BACK]);
02082 Vnm_print(0, "Nosh_setupMGPARA: partDisjOwnSide[UP] = %d\n",
02083     mgparm->partDisjOwnSide[VAPBS_UP]);
02084 Vnm_print(0, "Nosh_setupMGPARA: partDisjOwnSide[DOWN] = %d\n",
02085     mgparm->partDisjOwnSide[VAPBS_DOWN]);
02086
02087 /* Set the mesh parameters */
02088 mgparm->fglen[0] = xlenOlap;
02089 mgparm->fglen[1] = ylenOlap;
02090 mgparm->fglen[2] = zlenOlap;
02091 mgparm->partDisjLength[0] = xlenDisj;
02092 mgparm->partDisjLength[1] = ylenDisj;
02093 mgparm->partDisjLength[2] = zlenDisj;
02094 mgparm->partDisjCenter[0] = mgparm->fcenter[0] +
    xcentDisj;
02095 mgparm->partDisjCenter[1] = mgparm->fcenter[1] +
    ycentDisj;
02096 mgparm->partDisjCenter[2] = mgparm->fcenter[2] +
    zcentDisj;
02097 mgparm->fcenter[0] += xcentOlap;
02098 mgparm->fcenter[1] += ycentOlap;
02099 mgparm->fcenter[2] += zcentOlap;
02100
02101 Vnm_print(0, "Nosh_setupCalcMGPARA (%s, %d): Set up *relative* partition \
02102 centers...\n", __FILE__, __LINE__);
02103 Vnm_print(0, "Nosh_setupCalcMGPARA (%s, %d): Absolute centers will be set \
02104 in NOsh_setupMGAUTO\n", __FILE__, __LINE__);
02105 Vnm_print(0, "Nosh_setupCalcMGPARA (%s, %d): partDisjCenter = %g %g %g\n",
02106     __FILE__, __LINE__,
02107     mgparm->partDisjCenter[0],
02108     mgparm->partDisjCenter[1],
02109     mgparm->partDisjCenter[2]);
02110 Vnm_print(0, "Nosh_setupCalcMGPARA (%s, %d): ccenter = %g %g %g\n",
02111     __FILE__, __LINE__,
02112     mgparm->ccenter[0],
02113     mgparm->ccenter[1],
02114     mgparm->ccenter[2]);
02115 Vnm_print(0, "Nosh_setupCalcMGPARA (%s, %d): fcenter = %g %g %g\n",
02116     __FILE__, __LINE__,
02117     mgparm->fcenter[0],
02118     mgparm->fcenter[1],
02119     mgparm->fcenter[2]);
02120
02121 /* Setup the automatic focusing calculations associated with this processor */
02122 return NOsh_setupCalcMGAUTO(thee, elec);
02123
02124
02125 }
02126
02127 VPUBLIC int NOsh_parseFEM(
02128     NOsh *thee,
02129     Vio *sock,
02130     NOsh_calc *elec
02131     ) {
02132
02133     char tok[VMAX_BUFSIZE];
02134     FEMparm *feparm = VNULL;
02135     PBEparm *pbeparm = VNULL;
02136     int rc;
02137     Vrc_Codes vrc;
02138
02139     /* Check the arguments */
02140     if (thee == VNULL) {
02141         Vnm_print(2, "NOsh_parseFEM: Got NULL thee!\n");
02142         return 0;
02143     }
02144     if (sock == VNULL) {
02145         Vnm_print(2, "NOsh_parseFEM: Got pointer to NULL socket!\n");

```

```

02146     return 0;
02147 }
02148 if (elec == VNULL) {
02149   Vnm_print(2, "NOsh_parseFEM: Got pointer to NULL elec object!\n");
02150   return 0;
02151 }
02152 feparm = elec->feparm;
02153 if (feparm == VNULL) {
02154   Vnm_print(2, "NOsh_parseFEM: Got pointer to NULL feparm object!\n");
02155   return 0;
02156 }
02157 pbeparm = elec->pbeparm;
02158 if (pbeparm == VNULL) {
02159   Vnm_print(2, "NOsh_parseFEM: Got pointer to NULL pbeparm object!\n");
02160   return 0;
02161 }
02162
02163 Vnm_print(0, "NOsh_parseFEM: Parsing parameters for FEM calculation\n");
02164
02165 /* Start snarfing tokens from the input stream */
02166 rc = 1;
02167 while (Vio_scanf(sock, "%s", tok) == 1) {
02168
02169   Vnm_print(0, "NOsh_parseFEM: Parsing %s...\n", tok);
02170
02171   /* See if it's an END token */
02172   if (Vstring_strcasecmp(tok, "end") == 0) {
02173     feparm->parsed = 1;
02174     pbeparm->parsed = 1;
02175     rc = 1;
02176     break;
02177   }
02178
02179   /* Pass the token through a series of parsers */
02180   rc = PBEparm_parseToken(pbeparm, tok, sock);
02181   if (rc == -1) {
02182     Vnm_print(0, "NOsh_parseFEM: parsePBE error!\n");
02183     break;
02184   } else if (rc == 0) {
02185     /* Pass the token to the generic MG parser */
02186     vrc = FEMPARM_parseToken(feparm, tok, sock);
02187     if (vrc == VRC_FAILURE) {
02188       Vnm_print(0, "NOsh_parseFEM: parseMG error!\n");
02189       break;
02190     } else if (vrc == VRC_WARNING) {
02191       /* We ran out of parsers! */
02192       Vnm_print(2, "NOsh: Unrecognized keyword: %s\n", tok);
02193       break;
02194     }
02195   }
02196 }
02197
02198 /* Handle various errors arising in the token-snarfing loop -- these all
02199 * just result in simple returns right now */
02200 if (rc == -1) return 0;
02201 if (rc == 0) return 0;
02202
02203 /* Check the status of the parameter objects */
02204 if ((!FEMPARM_check(feparm)) || (!PBEparm_check(
02205   pbeparm))) {
02206   Vnm_print(2, "NOsh: FEM parameters not set correctly!\n");
02207   return 0;
02208 }
02209 return 1;
02210
02211 }
02212
02213 VPRIVATE int NOsh_setupCalcFEMANUAL(
02214   NOsh *thee,
02215   NOsh_calc *elec
02216 ) {
02217
02218   FEMPARM *feparm = VNULL;
02219   PBEparm *pbeparm = VNULL;
02220   NOsh_calc *calc = VNULL;
02221
02222   VASSERT(thee != VNULL);
02223   VASSERT(elec != VNULL);
02224   feparm = elec->feparm;
02225   VASSERT(feparm != VNULL);

```

```

02226 pbeparm = elec->pbeparm;
02227 VASSERT(pbeparm);
02228
02229 /* Check to see if he have any room left for this type of
0230 * calculation, if so: set the calculation type, update the number
0231 * of calculations of this type, and parse the rest of the section
0232 */
0233 if (thee->nalc >= NOSH_MAXCALC) {
0234 Vnm_print(2, "Nosh: Too many calculations in this run!\n");
0235 Vnm_print(2, "Nosh: Current max is %d; ignoring this calculation\n",
0236 NOSH_MAXCALC);
0237 return 0;
0238 }
0239 thee->calc[thee->nalc] = NOsh_calc_ctor(NCT_FEM
0240 );
0241 calc = thee->calc[thee->nalc];
0242 (thee->nalc)++;
0243 /* Copy over contents of ELEC */
0244 NOsh_calc_copy(calc, elec);
0245
0246
0247 return 1;
0248 }
0249
0250 VPUBLIC int NOsh_parseAPOL(
0251     Nosh *thee,
0252     Vio *sock,
0253     NOsh_calc *elec
0254     ) {
0255
0256 char tok[VMAX_BUFSIZE];
0257 APOLparm *apolparm = VNULL;
0258 int rc;
0259
0260 /* Check the arguments */
0261 if (thee == VNULL) {
0262 Vnm_print(2, "NOsh_parseAPOL: Got NULL thee!\n");
0263 return 0;
0264 }
0265 if (sock == VNULL) {
0266 Vnm_print(2, "NOsh_parseAPOL: Got pointer to NULL socket!\n");
0267 return 0;
0268 }
0269 if (elec == VNULL) {
0270 Vnm_print(2, "NOsh_parseAPOL: Got pointer to NULL elec object!\n");
0271 return 0;
0272 }
0273 apolparm = elec->apolparm;
0274 if (apolparm == VNULL) {
0275 Vnm_print(2, "NOsh_parseAPOL: Got pointer to NULL feparm object!\n");
0276 return 0;
0277 }
0278
0279 Vnm_print(0, "NOsh_parseAPOL: Parsing parameters for APOL calculation\n");
0280
0281 /* Start snarfing tokens from the input stream */
0282 rc = 1;
0283 while (Vio_scanf(sock, "%s", tok) == 1) {
0284
0285 Vnm_print(0, "NOsh_parseAPOL: Parsing %s...\n", tok);
0286 /* See if it's an END token */
0287 if (Vstring_strcasecmp(tok, "end") == 0) {
0288     apolparm->parsed = 1;
0289     rc = 1;
0290     break;
0291 }
0292
0293 /* Pass the token through a series of parsers */
0294 /* Pass the token to the generic non-polar parser */
0295 rc = APOLparm_parseToken(apolparm, tok, sock);
0296 if (rc == -1) {
0297     Vnm_print(0, "NOsh_parseFEM: parseMG error!\n");
0298     break;
0299 } else if (rc == 0) {
0300     /* We ran out of parsers! */
0301     Vnm_print(2, "NOsh: Unrecognized keyword: %s\n", tok);
0302     break;
0303 }
0304
0305 }

```

```

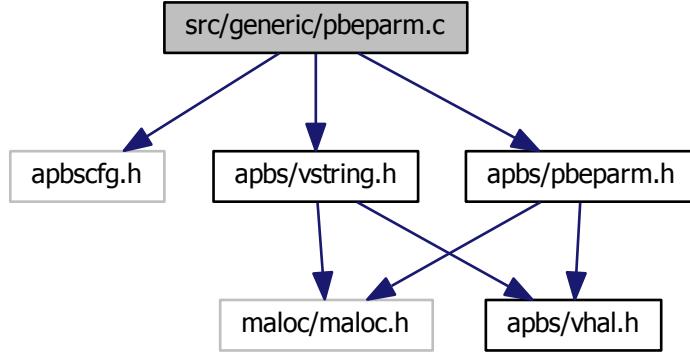
02306  /* Handle various errors arising in the token-snarfing loop -- these all
02307  * just result in simple returns right now */
02308  if (rc == -1) return 0;
02309  if (rc == 0) return 0;
02310
02311
02312  /* Check the status of the parameter objects */
02313  if (!APOLparm_check(apolparm)) {
02314    Vnm_print(2, "NOsh: APOL parameters not set correctly!\n");
02315    return 0;
02316  }
02317
02318  return 1;
02319
02320 }
02321
02322 VPRIIVATE int NOsh_setupCalcAPOL(
02323     NOsh *thee,
02324     NOsh_calc *apol
02325 ) {
02326
02327     NOsh_calc *calc = VNULL;
02328
02329     VASSERT(thee != VNULL);
02330     VASSERT(apol != VNULL);
02331
02332     /* Check to see if he have any room left for this type of
02333      * calculation, if so: set the calculation type, update the number
02334      * of calculations of this type, and parse the rest of the section
02335      */
02336     if (thee->nCalc >= NOSH_MAXCALC) {
02337       Vnm_print(2, "NOsh: Too many calculations in this run!\n");
02338       Vnm_print(2, "NOsh: Current max is %d; ignoring this calculation\n",
02339                  NOSH_MAXCALC);
02340       return 0;
02341     }
02342     thee->calc[thee->nCalc] = NOsh_calc_ctor(NCT_APOL
02343 );
02343     calc = thee->calc[thee->nCalc];
02344     (thee->nCalc)++;
02345
02346     /* Copy over contents of APOL */
02347     NOsh_calc_copy(calc, apol);
02348
02349     return 1;
02350 }
```

9.61 src/generic/pbeparm.c File Reference

Class PBParm methods.

```
#include "apbscfg.h"
#include "apbs/pbeparm.h"
#include "apbs/vstring.h"
```

Include dependency graph for pbeparm.c:



Functions

- VPUBLIC double `PBEparm_getIonCharge (PBEparm *thee, int i)`
Get charge (e) of specified ion species.
- VPUBLIC double `PBEparm_getIonConc (PBEparm *thee, int i)`
Get concentration (M) of specified ion species.
- VPUBLIC double `PBEparm_getIonRadius (PBEparm *thee, int i)`
Get radius (A) of specified ion species.
- VPUBLIC double `PBEparm_getZmem (PBEparm *thee)`
- VPUBLIC double `PBEparm_getLmem (PBEparm *thee)`
- VPUBLIC double `PBEparm_getMembraneDiel (PBEparm *thee)`
- VPUBLIC double `PBEparm_getMemv (PBEparm *thee)`
- VPUBLIC `PBEparm * PBEparm_ctor ()`
Construct PBEparm object.
- VPUBLIC int `PBEparm_ctor2 (PBEparm *thee)`
FORTRAN stub to construct PBEparm object.
- VPUBLIC void `PBEparm_dtor (PBEparm **thee)`
Object destructor.
- VPUBLIC void `PBEparm_dtor2 (PBEparm *thee)`
FORTRAN stub for object destructor.
- VPUBLIC int `PBEparm_check (PBEparm *thee)`
Consistency check for parameter values stored in object.
- VPUBLIC void `PBEparm_copy (PBEparm *thee, PBEparm *parm)`
Copy PBEparm object into thee.
- VPRIVATE int `PBEparm_parseLPBE (PBEparm *thee, Vio *sock)`
- VPRIVATE int `PBEparm_parseNPBE (PBEparm *thee, Vio *sock)`
- VPRIVATE int `PBEparm_parseMOL (PBEparm *thee, Vio *sock)`
- VPRIVATE int `PBEparm_parseLRPBE (PBEparm *thee, Vio *sock)`

- VPRIVATE int **PBEparm_parseNRPBE** (**PBEparm** *thee, **Vio** *sock)
- VPRIVATE int **PBEparm_parseSMPBE** (**PBEparm** *thee, **Vio** *sock)
- VPRIVATE int **PBEparm_parseBCFL** (**PBEparm** *thee, **Vio** *sock)
- VPRIVATE int **PBEparm_parseISON** (**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.

9.61.1 Detailed Description

Class PBEparm methods.

Author

Nathan Baker

Version

Id:

[pbeparm.c](#) 1750 2012-07-18 18:34:27Z tuckerbeck

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](#).

9.62 pbeparm.c

```

00001
00057 #include "apbscfg.h"
00058 #include "apbs/pbeparm.h"
00059 #include "apbs/vstring.h"
00060
00061 VEMBED(rcsid="$Id: pbeparm.c 1750 2012-07-18 18:34:27Z tuckerbeck $" )
00062
00063 #if !defined(VINLINE_MGPARM)
00064
00065 #endif /* if !defined(VINLINE_MGPARM) */
00066
00067 VPUBLIC double PBParm_getIonCharge(PBParm *thee,
00068     int i) {
00069     VASSERT(thee != VNULL);
00070     VASSERT(i < thee->nion);
00071     return thee->ionq[i];
00072 }
00073 VPUBLIC double PBParm_getIonConc(PBParm *thee, int i
00074 ) {
00075     VASSERT(thee != VNULL);
00076     VASSERT(i < thee->nion);
00077     return thee->ionc[i];
00078 }
00079 VPUBLIC double PBParm_getIonRadius(PBParm *thee,
00080     int i) {
00081     VASSERT(thee != VNULL);
00082     VASSERT(i < thee->nion);
00083     return thee->ionr[i];
00084 }
00085 /*-----*/

```

```

00086 /* Added by Michael Grabe */  

00087 /*-----*/  

00088  

00089 VPUBLIC double PBparm_getzmem(PBparm *thee) {  

00090     VASSERT(thee != VNULL);  

00091     return thee->zmem;  

00092 }  

00093 VPUBLIC double PBparm_getLmem(PBparm *thee) {  

00094     VASSERT(thee != VNULL);  

00095     return thee->Lmem;  

00096 }  

00097 VPUBLIC double PBparm_getmembraneDiel(PBparm *thee) {  

00098     VASSERT(thee != VNULL);  

00099     return thee->mdie;  

00100 }  

00101 VPUBLIC double PBparm_getmemv(PBparm *thee) {  

00102     VASSERT(thee != VNULL);  

00103     return thee->memv;  

00104 }  

00105  

00106 VPUBLIC PBparm* PBparm_ctor() {  

00107     /* Set up the structure */  

00108     PBparm *thee = VNULL;  

00109     thee = (PBparm*)Vmem_malloc(VNULL, 1, sizeof(PBparm));  

00110     VASSERT( thee != VNULL );  

00111     VASSERT( PBparm_ctor2(thee) );  

00112  

00113     return thee;  

00114 }  

00115 }  

00116  

00117 VPUBLIC int PBparm_ctor2(PBparm *thee) {  

00118     int i;  

00119  

00120     if (thee == VNULL) return 0;  

00121  

00122     thee->parsed = 0;  

00123  

00124     thee->setmolid = 0;  

00125     thee->setpbtype = 0;  

00126     thee->setbcfl = 0;  

00127     thee->setnion = 0;  

00128     for (i=0; i<MAXION; i++) {  

00129         thee->seotion[i] = 0;  

00130         thee->ionq[i] = 0.0;  

00131         thee->ionc[i] = 0.0;  

00132         thee->ionr[i] = 0.0;  

00133     }  

00134     thee->setpdie = 0;  

00135     thee->setsdie = 0;  

00136     thee->setsrfm = 0;  

00137     thee->setsrad = 0;  

00138     thee->setswin = 0;  

00139     thee->settemp = 0;  

00140     thee->setcalcenergy = 0;  

00141     thee->setcalcforce = 0;  

00142     thee->setsdens = 0;  

00143     thee->numwrite = 0;  

00144     thee->setwritemat = 0;  

00145     thee->nion = 0;  

00146     thee->sdens = 0;  

00147     thee->swin = 0;  

00148     thee->srad = 1.4;  

00149     thee->useDie1Map = 0;  

00150     thee->useKappaMap = 0;  

00151     thee->usePotMap = 0;  

00152     thee->useChargeMap = 0;  

00153  

00154     /*-----*/  

00155     /* Added by Michael Grabe */  

00156     /*-----*/  

00157  

00158     thee->setzmem = 0;  

00159     thee->setLmem = 0;  

00160     thee->setmdie = 0;  

00161     thee->setmemv = 0;  

00162  

00163     /*-----*/  

00164     thee->smsize = 0.0;  

00165  

00166

```

```

00167     thee->smvolume = 0.0;
00168
00169     thee->setsmsize = 0;
00170     thee->setsmvolume = 0;
00171
00172     return 1;
00173 }
00174
00175 VPUBLIC void PBParm_dtor(PBParm **thee) {
00176     if ((*thee) != VNULL) {
00177         PBParm_dtor2(*thee);
00178         Vmem_free(VNULL, 1, sizeof(PBParm), (void **)thee);
00179         (*thee) = VNULL;
00180     }
00181 }
00182
00183 VPUBLIC void PBParm_dtor2(PBParm *thee) { ; }
00184
00185 VPUBLIC int PBParm_check(PBParm *thee) {
00186
00187     int i;
00188
00189     /* Check to see if we were even filled... */
00190     if (!thee->parsed) {
00191         Vnm_print(2, "PBParm_check: not filled!\n");
00192         return 0;
00193     }
00194
00195     if (!thee->setmolid) {
00196         Vnm_print(2, "PBParm_check: MOL not set!\n");
00197         return 0;
00198     }
00199     if (!thee->setpbtype) {
00200         Vnm_print(2, "PBParm_check: LPBE/NPBE/LRPBE/NRPBE/SMPBE not set!\n");
00201         return 0;
00202     }
00203     if (!thee->setbcfl) {
00204         Vnm_print(2, "PBParm_check: BCFL not set!\n");
00205         return 0;
00206     }
00207     if (!thee->setnion) {
00208         thee->setnion = 1;
00209         thee->nion = 0;
00210     }
00211     for (i=0; i<thee->nion; i++) {
00212         if (!thee->seion[i]) {
00213             Vnm_print(2, "PBParm_check: ION # %d not set!\n", i);
00214             return 0;
00215         }
00216     }
00217     if (!thee->setpdie) {
00218         Vnm_print(2, "PBParm_check: PDIE not set!\n");
00219         return 0;
00220     }
00221     if (((thee->srfm==VSM_MOL) || (thee->srfm==VSM_MOLSMOOTH
00222 )) \
00223         && (!thee->setsdens) && (thee->srad > VSMALL)) {
00224         Vnm_print(2, "PBParm_check: SDENS not set!\n");
00225         return 0;
00226     }
00227     if (!thee->setsdie) {
00228         Vnm_print(2, "PBParm_check: SDIE not set!\n");
00229         return 0;
00230     }
00231     if (!thee->setsrfm) {
00232         Vnm_print(2, "PBParm_check: SRFM not set!\n");
00233         return 0;
00234     }
00235     if (((thee->srfm==VSM_MOL) || (thee->srfm==VSM_MOLSMOOTH
00236 )) \
00237         && (!thee->setsrad)) {
00238         Vnm_print(2, "PBParm_check: SRAD not set!\n");
00239     }
00240     if ((thee->srfm==VSM_SPLINE) && (!thee->setswin)) {
00241         Vnm_print(2, "PBParm_check: SWIN not set!\n");
00242         return 0;
00243     }
00244     if ((thee->srfm==VSM_SPLINE3) && (!thee->setswin)) {
00245         Vnm_print(2, "PBParm_check: SWIN not set!\n");
00246         return 0;

```

```

00246     }
00247     if ((thee->srfm==VSM_SPLINE4) && (!thee->setswin)) {
00248         Vnm_print(2, "PBEparm_check: SWIN not set!\n");
00249         return 0;
00250     }
00251     if (!thee->settemp) {
00252         Vnm_print(2, "PBEparm_check: TEMP not set!\n");
00253         return 0;
00254     }
00255     if (!thee->setcalcenergy) thee->calcenergy = PCE_NO
00256 ;
00257     if (!thee->setcalcforce) thee->calcforce = PCF_NO
00258 ;
00259 /*-----*/
00260 /* Added by Michael Grabe */
00261 /*-----*/
00262
00263     if ((!thee->setzmem) && (thee->bcfl == 3)){
00264         Vnm_print(2, "PBEparm_check: ZMEM not set!\n");
00265         return 0;
00266     }
00267     if ((!thee->setLmem) && (thee->bcfl == 3)){
00268         Vnm_print(2, "PBEparm_check: LMEM not set!\n");
00269         return 0;
00270     }
00271     if ((!thee->setmdie) && (thee->bcfl == 3)){
00272         Vnm_print(2, "PBEparm_check: MDIE not set!\n");
00273         return 0;
00274     }
00275     if ((!thee->setmemv) && (thee->bcfl == 3)){
00276         Vnm_print(2, "PBEparm_check: MEMV not set!\n");
00277         return 0;
00278     }
00279 /*-----*/
00280
00281     return 1;
00282 }
00283 }
00284
00285 VPUBLIC void PBEparm_copy(PBEparm *thee, PBEparm *
00286     parm) {
00287     int i, j;
00288
00289     VASSERT(thee != VNULL);
00290     VASSERT(parm != VNULL);
00291
00292     thee->molid = parm->molid;
00293     thee->setmolid = parm->setmolid;
00294     thee->useDielMap = parm->useDielMap;
00295     thee->dielMapID = parm->dielMapID;
00296     thee->useKappaMap = parm->useKappaMap;
00297     thee->kappaMapID = parm->kappaMapID;
00298     thee->usePotMap = parm->usePotMap;
00299     thee->potMapID = parm->potMapID;
00300     thee->useChargeMap = parm->useChargeMap;
00301     thee->chargeMapID = parm->chargeMapID;
00302     thee->pbtetype = parm->pbtetype;
00303     thee->setpbtype = parm->setpbtype;
00304     thee->bcfl = parm->bcfl;
00305     thee->setbcfl = parm->setbcfl;
00306     thee->nion = parm->nion;
00307     thee->setnion = parm->setnion;
00308     for (i=0; i<MAXION; i++) {
00309         thee->ionq[i] = parm->ionq[i];
00310         thee->ionc[i] = parm->ionc[i];
00311         thee->ionr[i] = parm->ionr[i];
00312         thee->setion[i] = parm->setion[i];
00313     };
00314     thee->pdie = parm->pdie;
00315     thee->setpdie = parm->setpdie;
00316     thee->sdens = parm->sdens;
00317     thee->setsdens = parm->setsdens;
00318     thee->sdie = parm->sdie;
00319     thee->setsdie = parm->setsdie;
00320     thee->srfm = parm->srfm;
00321     thee->setsrfm = parm->setsrfm;
00322     thee->srad = parm->srad;
00323     thee->setsrad = parm->setsrad;

```

```

00324     thee->swin = parm->swin;
00325     thee->setswin = parm->setswin;
00326     thee->temp = parm->temp;
00327     thee->settemp = parm->settemp;
00328     thee->calcenergy = parm->calcenergy;
00329     thee->setcalcenergy = parm->setcalcenergy;
00330     thee->calcforce = parm->calcforce,
00331     thee->setcalcforce = parm->setcalcforce;
00332
00333 /*-----*/
00334 /* Added by Michael Grabe */
00335 /*-----*/
00336
00337     thee->zmem = parm->zmem;
00338     thee->setzmem = parm->setzmem;
00339     thee->Lmem = parm->Lmem;
00340     thee->setLmem = parm->setLmem;
00341     thee->mdie = parm->mdie;
00342     thee->setmdie = parm->setmdie;
00343     thee->memv = parm->memv;
00344     thee->setmemv = parm->setmemv;
00345
00346 /*-----*/
00347
00348     thee->numwrite = parm->numwrite;
00349     for (i=0; i<PBEPPARM_MAXWRITE; i++) {
00350         thee->writetype[i] = parm->writetype[i];
00351         thee->writefmt[i] = parm->writefmt[i];
00352         for (j=0; j<VMAX_ARGLEN; j++)
00353             thee->writestem[i][j] = parm->writestem[i][j];
00354     }
00355     thee->writemat = parm->writemat;
00356     thee->setwritemat = parm->setwritemat;
00357     for (i=0; i<VMAX_ARGLEN; i++) thee->writematstem[i] = parm->
00358     writematstem[i];
00359     thee->writematflag = parm->writematflag;
00360
00361     thee->smsize = parm->smsize;
00362     thee->smvolume = parm->smvolume;
00363
00364     thee->setsmsize = parm->setsmsize;
00365     thee->setsmvolume = parm->setsmvolume;
00366
00367     thee->parsed = parm->parsed;
00368 }
00369
00370 VPRIPRIVATE int PBEParm_parseLPBE(PBEParm *thee, Vio *sock) {
00371     Vnm_print(0, "Nosh: parsed lpbe\n");
00372     thee->pbetype = PBE_LPBE;
00373     thee->setpbetype = 1;
00374     return 1;
00375 }
00376
00377 VPRIPRIVATE int PBEParm_parseNPBE(PBEParm *thee, Vio *sock) {
00378     Vnm_print(0, "Nosh: parsed npbe\n");
00379     thee->pbetype = PBE_NPBE;
00380     thee->setpbetype = 1;
00381     return 1;
00382 }
00383
00384 VPRIPRIVATE int PBEParm_parseMOL(PBEParm *thee, Vio *sock) {
00385     int ti;
00386     char tok[VMAX_BUFSIZE];
00387
00388     VJMPERR1(Vio_scantf(sock, "%s", tok) == 1);
00389     if (sscanf(tok, "%d", &ti) == 0) {
00390         Vnm_print(2, "Nosh: Read non-int (%s) while parsing MOL \
00391 keyword!\n", tok);
00392         return -1;
00393     }
00394     thee->molid = ti;
00395     thee->setmolid = 1;
00396     return 1;
00397
00398     VERROR1:
00399         Vnm_print(2, "parsePBE: ran out of tokens!\n");
00400         return -1;
00401 }
00402
00403 VPRIPRIVATE int PBEParm_parseLRPBE(PBEParm *thee, Vio *sock) {

```

```

00404     Vnm_print(0, "NOsh: parsed lrpbe\n");
00405     thee->pbetype = PBE_LRPBE;
00406     thee->setpbetype = 1;
00407     return 1;
00408 }
00409
00410 VPRIIVATE int PBEparm_parseNRPBE(PBEparm *thee, Vio *sock) {
00411     Vnm_print(0, "NOsh: parsed nrpbe\n");
00412     thee->pbetype = PBE_NRPBE;
00413     thee->setpbetype = 1;
00414     return 1;
00415 }
00416
00417 VPRIIVATE int PBEparm_parseSMPBE(PBEparm *thee, Vio *sock) {
00418
00419     int i;
00420
00421     char type[VMAX_BUFSIZE]; /* vol or size (keywords) */
00422     char value[VMAX_BUFSIZE]; /* floating point value */
00423
00424     char setVol = 1;
00425     char setSize = 1;
00426     char keyValuePairs = 2;
00427
00428     double size, volume;
00429
00430     for(i=0;i<keyValuePairs;i++) {
00431
00432     /* The line two tokens at a time */
00433     VJMPERR1(Vio_sccanf(sock, "%s", type) == 1);
00434     VJMPERR1(Vio_sccanf(sock, "%s", value) == 1);
00435
00436     if(!strcmp(type,"vol")){
00437         if ((setVol = sscanf(value, "%lf", &volume)) == 0){
00438             Vnm_print(2,"NOsh: Read non-float (%s) while parsing smpbe keyword!\n",
00439                         value);
00440             return VRC_FAILURE;
00441         }
00442     }else if(!strcmp(type,"size")){
00443         if ((setSize = sscanf(value, "%lf", &size)) == 0){
00444             Vnm_print(2,"NOsh: Read non-float (%s) while parsing smpbe keyword!\n",
00445                         value);
00446             return VRC_FAILURE;
00447         }
00448     }else{
00449         Vnm_print(2,"NOsh: Read non-float (%s) while parsing smpbe keyword!\n",
00450                         value);
00451         return VRC_FAILURE;
00452     }
00453
00454     /* If either the volume or size isn't set, throw an error */
00455     if((setVol == 0) || (setSize == 0)){
00456         Vnm_print(2,"NOsh: Error while parsing smpbe keywords! Only size or vol was
00457                     specified.\n");
00458         return VRC_FAILURE;
00459     }
00460
00461     Vnm_print(0, "NOsh: parsed smpbe\n");
00462     thee->pbetype = PBE_SMPBE;
00463     thee->setpbetype = 1;
00464
00465     thee->smsize = size;
00466     thee->setsmsize = 1;
00467
00468     return VRC_SUCCESS;
00469
00470 VERROR1:
00471     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00472     return VRC_FAILURE;
00473
00474 }
00475
00476 VPRIIVATE int PBEparm_parseBCFL(PBEparm *thee, Vio *sock) {
00477     char tok[VMAX_BUFSIZE];
00478     int ti;
00479
00480     VJMPERR1(Vio_sccanf(sock, "%s", tok) == 1);

```

```

00481     /* We can either parse int flag... */
00482     if (sscanf(tok, "%d", &ti) == 1) {
00483
00484         thee->bcfl = (Vbcfl)ti;
00485         thee->setbcfl = 1;
00486         /* Warn that this usage is deprecated */
00487         Vnm_print(2, "parsePBE: Warning -- parsed deprecated \"bcfl %d\" \
00488 statement\n", ti);
00489         Vnm_print(2, "parsePBE: Please use \"bcfl ");
00490         switch (thee->bcfl) {
00491             case BCFL_ZERO:
00492                 Vnm_print(2, "zero");
00493                 break;
00494             case BCFL_SDH:
00495                 Vnm_print(2, "sdh");
00496                 break;
00497             case BCFL_MDH:
00498                 Vnm_print(2, "mdh");
00499                 break;
00500             case BCFL_FOCUS:
00501                 Vnm_print(2, "focus");
00502                 break;
00503             case BCFL_MEM:
00504                 Vnm_print(2, "mem");
00505                 break;
00506             case BCFL_MAP:
00507                 Vnm_print(2, "map");
00508                 break;
00509             default:
00510                 Vnm_print(2, "UNKNOWN");
00511                 break;
00512         }
00513     }
00514     Vnm_print(2, "\" instead.\n");
00515     return 1;
00516
00517     /* ...or the word */
00518 } else {
00519
00520     if (Vstring_strcasecmp(tok, "zero") == 0) {
00521         thee->bcfl = BCFL_ZERO;
00522         thee->setbcfl = 1;
00523         return 1;
00524     } else if (Vstring_strcasecmp(tok, "sdh") == 0) {
00525         thee->bcfl = BCFL_SDH;
00526         thee->setbcfl = 1;
00527         return 1;
00528     } else if (Vstring_strcasecmp(tok, "mdh") == 0) {
00529         thee->bcfl = BCFL_MDH;
00530         thee->setbcfl = 1;
00531         return 1;
00532     } else if (Vstring_strcasecmp(tok, "focus") == 0) {
00533         thee->bcfl = BCFL_FOCUS;
00534         thee->setbcfl = 1;
00535         return 1;
00536     } else if (Vstring_strcasecmp(tok, "mem") == 0) {
00537         thee->bcfl = BCFL_MEM;
00538         thee->setbcfl = 1;
00539         return 1;
00540     } else if (Vstring_strcasecmp(tok, "map") == 0) {
00541         thee->bcfl = BCFL_MAP;
00542         thee->setbcfl = 1;
00543         return 1;
00544     } else {
00545         Vnm_print(2, "NOsh: parsed unknown BCFL parameter (%s)!\\n",
00546                 tok);
00547         return -1;
00548     }
00549 }
00550     return 0;
00551
00552     VERROR1:
00553         Vnm_print(2, "parsePBE: ran out of tokens!\\n");
00554         return -1;
00555 }
00556
00557 VPRIVATE int PBParm_parseION(PBParm *thee, Vio *sock) {
00558
00559     int i;
00560     int meth = 0;
00561

```

```

00562 char tok[VMAX_BUFSIZE];
00563 char value[VMAX_BUFSIZE];
00564
00565 double tf;
00566 double charge, conc, radius;
00567
00568 int setCharge = 0;
00569 int setConc = 0;
00570 int setRadius = 0;
00571 int keyValuePairs = 3;
00572
00573 /* Get the initial token for the ION statement */
00574 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00575
00576 /* Scan the token once to determine the type (old style or new keyvalue pair)
 */
00577 meth = sscanf(tok, "%lf", &tf);
00578 /* If tok is a non-zero float value, we are using the old method */
00579 if(meth != 0){
00580
00581 Vnm_print(2, "NOsh: Deprecated use of ION keyword! Use key-value pairs\n",
tok);
00582
00583 if (sscanf(tok, "%lf", &tf) == 0) {
00584 Vnm_print(2, "NOsh: Read non-float (%s) while parsing ION keyword!\n", tok)
;
00585 return VRC_FAILURE;
00586 }
00587 thee->ionq[thee->nion] = tf;
00588 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00589 if (sscanf(tok, "%lf", &tf) == 0) {
00590 Vnm_print(2, "NOsh: Read non-float (%s) while parsing ION keyword!\n", tok)
;
00591 return VRC_FAILURE;
00592 }
00593 thee->ionc[thee->nion] = tf;
00594 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00595 if (sscanf(tok, "%lf", &tf) == 0) {
00596 Vnm_print(2, "NOsh: Read non-float (%s) while parsing ION keyword!\n", tok)
;
00597 return VRC_FAILURE;
00598 }
00599 thee->ionr[thee->nion] = tf;
00600
00601 }else{
00602
00603 /* Three key-value pairs (charge, radius and conc) */
00604 for(i=0;i<keyValuePairs;i++){
00605
00606 /* Now scan for the value (float) to be used with the key token parsed
 * above the if-else statement */
00607 VJMPERR1(Vio_scanf(sock, "%s", value) == 1);
00608 if(!strcmp(tok,"charge")){
00609 setCharge = sscanf(value, "%lf", &charge);
00610 if (setCharge == 0){
00611 Vnm_print(2,"NOsh: Read non-float (%s) while parsing ION %s keyword!\n",
value, tok);
00612 return VRC_FAILURE;
00613 }
00614 thee->ionq[thee->nion] = charge;
00615 }else if(!strcmp(tok,"radius")){
00616 setRadius = sscanf(value, "%lf", &radius);
00617 if (setRadius == 0){
00618 Vnm_print(2,"NOsh: Read non-float (%s) while parsing ION %s keyword!\n",
value, tok);
00619 return VRC_FAILURE;
00620 }
00621 thee->ionr[thee->nion] = radius;
00622 }else if(!strcmp(tok,"conc")){
00623 setConc = sscanf(value, "%lf", &conc);
00624 if (setConc == 0){
00625 Vnm_print(2,"NOsh: Read non-float (%s) while parsing ION %s keyword!\n",
value, tok);
00626 return VRC_FAILURE;
00627 }
00628 thee->ionc[thee->nion] = conc;
00629 }else{
00630 Vnm_print(2,"NOsh: Illegal or missing key-value pair for ION keyword!\n");
00631 return VRC_FAILURE;
00632 }
00633 }
00634 }
```

```

00635     /* If all three values haven't be set (setValue = 0) then read the next
00636      token */
00637     if((setCharge != 1) || (setConc != 1) || (setRadius != 1)){
00638         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00639     }
00640 } /* end for */
00641 } /* end if */
00642
00643 /* Finally set the setion, nion and setnion flags and return success */
00644 theee->setion[thee->nion] = 1;
00645 (thee->nion)++;
00646 thee->setnion = 1;
00647 return VRC_SUCCESS;
00648
00649 VERROR1:
00650     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00651     return VRC_FAILURE;
00652 }
00653
00654 VPRIPRIVATE int PBparm_parsePDIE(PBparm *thee, Vio *sock) {
00655     char tok[VMAX_BUFSIZE];
00656     double tf;
00657
00658     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00659     if (sscanf(tok, "%lf", &tf) == 0) {
00660         Vnm_print(2, "NOsh: Read non-float (%s) while parsing PDIE \
00661 keyword!\n", tok);
00662         return -1;
00663     }
00664     thee->pdie = tf;
00665     thee->setpdie = 1;
00666     return 1;
00667
00668 VERROR1:
00669     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00670     return -1;
00671 }
00672
00673 VPRIPRIVATE int PBparm_parseSDENS(PBparm *thee, Vio *sock) {
00674     char tok[VMAX_BUFSIZE];
00675     double tf;
00676
00677     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00678     if (sscanf(tok, "%lf", &tf) == 0) {
00679         Vnm_print(2, "NOsh: Read non-float (%s) while parsing SDENS \
00680 keyword!\n", tok);
00681         return -1;
00682     }
00683     thee->sdens = tf;
00684     thee->setsdens = 1;
00685     return 1;
00686
00687 VERROR1:
00688     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00689     return -1;
00690 }
00691
00692 VPRIPRIVATE int PBparm_parseSDIE(PBparm *thee, Vio *sock) {
00693     char tok[VMAX_BUFSIZE];
00694     double tf;
00695
00696     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00697     if (sscanf(tok, "%lf", &tf) == 0) {
00698         Vnm_print(2, "NOsh: Read non-float (%s) while parsing SDIE \
00699 keyword!\n", tok);
00700         return -1;
00701     }
00702     thee->sdie = tf;
00703     thee->setsdie = 1;
00704     return 1;
00705
00706 VERROR1:
00707     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00708     return -1;
00709 }
00710
00711 VPRIPRIVATE int PBparm_parseSRFM(PBparm *thee, Vio *sock) {
00712     char tok[VMAX_BUFSIZE];
00713     int ti;
00714

```

```

00715     VJMPERR1(Vio_scantf(sock, "%s", tok) == 1);
00716
00717     /* Parse old-style int arg */
00718     if (sscanf(tok, "%d", &ti) == 1) {
00719         thee->srfm = (Vsurf_Meth)ti;
00720         thee->setsrfm = 1;
00721
00722         Vnm_print(2, "parsePBE: Warning -- parsed deprecated \"srfm %d\" \
00723 statement.\n", ti);
00724         Vnm_print(2, "parsePBE: Please use \"srfm \"");
00725         switch (thee->srfm) {
00726             case VSM_MOL:
00727                 Vnm_print(2, "mol");
00728                 break;
00729             case VSM_MOLSMOOTH:
00730                 Vnm_print(2, "smol");
00731                 break;
00732             case VSM_SPLINE:
00733                 Vnm_print(2, "spl2");
00734                 break;
00735             case VSM_SPLINE3:
00736                 Vnm_print(2, "spl3");
00737                 break;
00738             case VSM_SPLINE4:
00739                 Vnm_print(2, "spl4");
00740                 break;
00741             default:
00742                 Vnm_print(2, "UNKNOWN");
00743                 break;
00744         }
00745         Vnm_print(2, "\" instead.\n");
00746         return 1;
00747
00748     /* Parse newer text-based args */
00749     } else if (Vstring_strcasecmp(tok, "mol") == 0) {
00750         thee->srfm = VSM_MOL;
00751         thee->setsrfm = 1;
00752         return 1;
00753     } else if (Vstring_strcasecmp(tok, "smol") == 0) {
00754         thee->srfm = VSM_MOLSMOOTH;
00755         thee->setsrfm = 1;
00756         return 1;
00757     } else if (Vstring_strcasecmp(tok, "spl2") == 0) {
00758         thee->srfm = VSM_SPLINE;
00759         thee->setsrfm = 1;
00760         return 1;
00761     } else if (Vstring_strcasecmp(tok, "spl3") == 0) {
00762         thee->srfm = VSM_SPLINE3;
00763         thee->setsrfm = 1;
00764         return 1;
00765     } else if (Vstring_strcasecmp(tok, "spl4") == 0) {
00766         thee->srfm = VSM_SPLINE4;
00767         thee->setsrfm = 1;
00768         return 1;
00769     } else {
00770         Vnm_print(2, "NOsh: Unrecognized keyword (%s) when parsing \
00771 srfm!\n", tok);
00772         return -1;
00773     }
00774
00775     return 0;
00776
00777     VERROR1:
00778     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00779     return -1;
00780 }
00781
00782 VPRIPRIVATE int PBEparm_parseSRAD(PBEparm *thee, Vio *sock) {
00783     char tok[VMAX_BUFSIZE];
00784     double tf;
00785
00786     VJMPERR1(Vio_scantf(sock, "%s", tok) == 1);
00787     if (sscanf(tok, "%lf", &tf) == 0) {
00788         Vnm_print(2, "NOsh: Read non-float (%s) while parsing SRAD \
00789 keyword!\n", tok);
00790         return -1;
00791     }
00792     thee->srad = tf;
00793     thee->setsrad = 1;
00794     return 1;
00795 }
```

```

00796     VERROR1:
00797         Vnm_print(2, "parsePBE: ran out of tokens!\n");
00798         return -1;
00799     }
00800
00801 VPRIVATE int PBParm_parseSWIN(PBParm *thee, Vio *sock) {
00802     char tok[VMAX_BUFSIZE];
00803     double tf;
00804
00805     VJMPERR1(Vio_scantok(sock, "%s", tok) == 1);
00806     if (sscanf(tok, "%lf", &tf) == 0) {
00807         Vnm_print(2, "NOsh: Read non-float (%s) while parsing SWIN \
00808 keyword!\n", tok);
00809         return -1;
00810     }
00811     thee->swin = tf;
00812     thee->setswin = 1;
00813     return 1;
00814
00815     VERROR1:
00816         Vnm_print(2, "parsePBE: ran out of tokens!\n");
00817         return -1;
00818     }
00819
00820 VPRIVATE int PBParm_parseTEMP(PBParm *thee, Vio *sock) {
00821     char tok[VMAX_BUFSIZE];
00822     double tf;
00823
00824     VJMPERR1(Vio_scantok(sock, "%s", tok) == 1);
00825     if (sscanf(tok, "%lf", &tf) == 0) {
00826         Vnm_print(2, "NOsh: Read non-float (%s) while parsing TEMP \
00827 keyword!\n", tok);
00828         return -1;
00829     }
00830     thee->temp = tf;
00831     thee->settemp = 1;
00832     return 1;
00833
00834     VERROR1:
00835         Vnm_print(2, "parsePBE: ran out of tokens!\n");
00836         return -1;
00837     }
00838
00839 VPRIVATE int PBParm_parseUSEMAP(PBParm *thee, Vio *sock) {
00840     char tok[VMAX_BUFSIZE];
00841     int ti;
00842
00843     VJMPERR1(Vio_scantok(sock, "%s", tok) == 1);
00844     Vnm_print(0, "PBParm_parseToken: Read %s...\n", tok);
00845     if (Vstring_strcasecmp(tok, "diel") == 0) {
00846         thee->useDielMap = 1;
00847         VJMPERR1(Vio_scantok(sock, "%s", tok) == 1);
00848         if (sscanf(tok, "%d", &ti) == 0) {
00849             Vnm_print(2, "NOsh: Read non-int (%s) while parsing \
00850 USEMAP DIEL keyword!\n", tok);
00851             return -1;
00852         }
00853         thee->dielMapID = ti;
00854         return 1;
00855     } else if (Vstring_strcasecmp(tok, "kappa") == 0) {
00856         thee->useKappaMap = 1;
00857         VJMPERR1(Vio_scantok(sock, "%s", tok) == 1);
00858         if (sscanf(tok, "%d", &ti) == 0) {
00859             Vnm_print(2, "NOsh: Read non-int (%s) while parsing \
00860 USEMAP KAPPA keyword!\n", tok);
00861             return -1;
00862         }
00863         thee->kappaMapID = ti;
00864         return 1;
00865     } else if (Vstring_strcasecmp(tok, "pot") == 0) {
00866         thee->usePotMap = 1;
00867         VJMPERR1(Vio_scantok(sock, "%s", tok) == 1);
00868         if (sscanf(tok, "%d", &ti) == 0) {
00869             Vnm_print(2, "NOsh: Read non-int (%s) while parsing \
00870 USEMAP POT keyword!\n", tok);
00871             return -1;
00872         }
00873         thee->potMapID = ti;
00874         return 1;
00875     } else if (Vstring_strcasecmp(tok, "charge") == 0) {
00876         thee->useChargeMap = 1;

```

```

00877     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00878     if (sscanf(tok, "%d", &ti) == 0) {
00879         Vnm_print(2, "NOsh: Read non-int (%s) while parsing \
00880 USEMAP CHARGE keyword!\n", tok);
00881         return -1;
00882     }
00883     thee->chargeMapID = ti;
00884     return 1;
00885 } else {
00886     Vnm_print(2, "NOsh: Read undefined keyword (%s) while parsing \
00887 USEMAP statement!\n", tok);
00888     return -1;
00889 }
00890 return 0;
00891
00892 VERROR1:
00893     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00894     return -1;
00895 }
00896
00897 VPRIPRIVATE int PBEparm_parseCALCENERGY(PBEparm *thee, Vio *sock) {
00898     char tok[VMAX_BUFSIZE];
00899     int ti;
00900
00901     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00902     /* Parse number */
00903     if (sscanf(tok, "%d", &ti) == 1) {
00904         thee->calcenergy = (PBEparm_calcEnergy)ti;
00905         thee->setcalcenergy = 1;
00906
00907         Vnm_print(2, "parsePBE: Warning -- parsed deprecated \"calcenergy \
00908 %d\" statement.\n", ti);
00909         Vnm_print(2, "parsePBE: Please use \"calcenergy \"");
00910         switch (thee->calcenergy) {
00911             case PCE_NO:
00912                 Vnm_print(2, "no");
00913                 break;
00914             case PCE_TOTAL:
00915                 Vnm_print(2, "total");
00916                 break;
00917             case PCE_COMPS:
00918                 Vnm_print(2, "comps");
00919                 break;
00920             default:
00921                 Vnm_print(2, "UNKNOWN");
00922                 break;
00923         }
00924         Vnm_print(2, "\" instead.\n");
00925         return 1;
00926     } else if (Vstring_strcasecmp(tok, "no") == 0) {
00927         thee->calcenergy = PCE_NO;
00928         thee->setcalcenergy = 1;
00929         return 1;
00930     } else if (Vstring_strcasecmp(tok, "total") == 0) {
00931         thee->calcenergy = PCE_TOTAL;
00932         thee->setcalcenergy = 1;
00933         return 1;
00934     } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00935         thee->calcenergy = PCE_COMPS;
00936         thee->setcalcenergy = 1;
00937         return 1;
00938     } else {
00939         Vnm_print(2, "NOsh: Unrecognized parameter (%s) while parsing \
00940 calcenergy!\n", tok);
00941         return -1;
00942     }
00943     return 0;
00944
00945 VERROR1:
00946     Vnm_print(2, "parsePBE: ran out of tokens!\n");
00947     return -1;
00948 }
00949
00950 VPRIPRIVATE int PBEparm_parseCALCFORCE(PBEparm *thee, Vio *sock) {
00951     char tok[VMAX_BUFSIZE];
00952     int ti;
00953
00954     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00955     /* Parse number */
00956     if (sscanf(tok, "%d", &ti) == 1) {
00957         thee->calcforce = (PBEparm_calcForce)ti;

```

```

00958     thee->setcalcforce = 1;
00959
00960     Vnm_print(2, "parsePBE:  Warning -- parsed deprecated \"calcforce \
00961 %d\" statement.\n", ti);
00962     Vnm_print(2, "parsePBE:  Please use \"calcforce \"");
00963     switch (thee->calcenergy) {
00964         case PCF_NO:
00965             Vnm_print(2, "no");
00966             break;
00967         case PCF_TOTAL:
00968             Vnm_print(2, "total");
00969             break;
00970         case PCF_COMPS:
00971             Vnm_print(2, "comps");
00972             break;
00973         default:
00974             Vnm_print(2, "UNKNOWN");
00975             break;
00976     }
00977     Vnm_print(2, "\" instead.\n");
00978     return 1;
00979 } else if (Vstring_strcasecmp(tok, "no") == 0) {
00980     thee->calcforce = PCF_NO;
00981     thee->setcalcforce = 1;
00982     return 1;
00983 } else if (Vstring_strcasecmp(tok, "total") == 0) {
00984     thee->calcforce = PCF_TOTAL;
00985     thee->setcalcforce = 1;
00986     return 1;
00987 } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00988     thee->calcforce = PCF_COMPS;
00989     thee->setcalcforce = 1;
00990     return 1;
00991 } else {
00992     Vnm_print(2, "NOsh:  Unrecognized parameter (%s) while parsing \
00993 calcforce!\n", tok);
00994     return -1;
00995 }
00996 return 0;
00997
00998 VERROR1:
00999     Vnm_print(2, "parsePBE:  ran out of tokens!\n");
01000     return -1;
01001 }
01002
01003 /*-----*/
01004 /* Added by Michael Grabe */
01005 /*-----*/
01006
01007 VPRIPRIVATE int PBParm_parseZMEM(PBParm *thee, Vio *sock) {
01008     char tok[VMAX_BUFSIZE];
01009     double tf;
01010
01011     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01012     if (sscanf(tok, "%lf", &tf) == 0) {
01013         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing ZMEM \
01014 keyword!\n", tok);
01015         return -1;
01016     }
01017     thee->zmem = tf;
01018     thee->setzmem = 1;
01019     return 1;
01020
01021 VERROR1:
01022     Vnm_print(2, "parsePBE:  ran out of tokens!\n");
01023     return -1;
01024 }
01025
01026
01027 VPRIPRIVATE int PBParm_parseLMEM(PBParm *thee, Vio *sock) {
01028     char tok[VMAX_BUFSIZE];
01029     double tf;
01030
01031     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01032     if (sscanf(tok, "%lf", &tf) == 0) {
01033         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing LMEM \
01034 keyword!\n", tok);
01035         return -1;
01036     }
01037     thee->Lmem = tf;
01038     thee->setLmem = 1;

```

```

01039     return 1;
01040
01041 VERROR1:
01042 Vnm_print(2, "parsePBE: ran out of tokens!\n");
01043     return -1;
01044 }
01045
01046 VPRIPRIVATE int PBEparm_parseMDIE(PBEparm *thee, Vio *sock) {
01047     char tok[VMAX_BUFSIZE];
01048     double tf;
01049
01050     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01051     if (sscanf(tok, "%lf", &tf) == 0) {
01052         Vnm_print(2, "NOSH: Read non-float (%s) while parsing MDIE \
01053         keyword!\n", tok);
01054         return -1;
01055     }
01056     thee->mdie = tf;
01057     thee->setmdie = 1;
01058     return 1;
01059
01060 VERROR1:
01061 Vnm_print(2, "parsePBE: ran out of tokens!\n");
01062     return -1;
01063 }
01064
01065 VPRIPRIVATE int PBEparm_parseMEMV(PBEparm *thee, Vio *sock) {
01066     char tok[VMAX_BUFSIZE];
01067     double tf;
01068
01069     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01070     if (sscanf(tok, "%lf", &tf) == 0) {
01071         Vnm_print(2, "NOSH: Read non-float (%s) while parsing MEMV \
01072         keyword!\n", tok);
01073         return -1;
01074     }
01075     thee->memv = tf;
01076     thee->setmemv = 1;
01077     return 1;
01078
01079 VERROR1:
01080 Vnm_print(2, "parsePBE: ran out of tokens!\n");
01081     return -1;
01082 }
01083
01084 /-----*/
01085
01086 VPRIPRIVATE int PBEparm_parseWRITE(PBEparm *thee, Vio *sock) {
01087     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]++;
01088     Vdata_Type writetype;
01089     Vdata_Format writefmt;
01090
01091     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01092     if (Vstring_strcasecmp(tok, "pot") == 0) {
01093         writetype = VDT_POT;
01094     } else if (Vstring_strcasecmp(tok, "atompot") == 0) {
01095         writetype = VDT_ATOMPOT;
01096     } else if (Vstring_strcasecmp(tok, "charge") == 0) {
01097         writetype = VDT_CHARGE;
01098     } else if (Vstring_strcasecmp(tok, "smol") == 0) {
01099         writetype = VDT_SMOL;
01100     } else if (Vstring_strcasecmp(tok, "dielx") == 0) {
01101         writetype = VDT_DIELX;
01102     } else if (Vstring_strcasecmp(tok, "diely") == 0) {
01103         writetype = VDT_DIELY;
01104     } else if (Vstring_strcasecmp(tok, "dielz") == 0) {
01105         writetype = VDT_DIELZ;
01106     } else if (Vstring_strcasecmp(tok, "kappa") == 0) {
01107         writetype = VDT_KAPPA;
01108     } else if (Vstring_strcasecmp(tok, "sspl") == 0) {
01109         writetype = VDT_SSPL;
01110     } else if (Vstring_strcasecmp(tok, "vdw") == 0) {
01111         writetype = VDT_VDW;
01112     } else if (Vstring_strcasecmp(tok, "ivdw") == 0) {
01113         writetype = VDT_IVDW;
01114     } else if (Vstring_strcasecmp(tok, "lap") == 0) {
01115         writetype = VDT_LAP;
01116     } else if (Vstring_strcasecmp(tok, "edens") == 0) {
01117         writetype = VDT_EDENS;
01118     } else if (Vstring_strcasecmp(tok, "ndens") == 0) {
01119         writetype = VDT_NDENS;

```

```

01120     } else if (Vstring_strcasecmp(tok, "qdens") == 0) {
01121         writetype = VDT_QDENS;
01122     } else {
01123         Vnm_print(2, "PBParm_parse: Invalid data type (%s) to write!\n",
01124             tok);
01125         return -1;
01126     }
01127     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01128     if (Vstring_strcasecmp(tok, "dx") == 0) {
01129         writefmt = VDF_DX;
01130     } else if (Vstring_strcasecmp(tok, "uhbd") == 0) {
01131         writefmt = VDF_UHBD;
01132     } else if (Vstring_strcasecmp(tok, "avs") == 0) {
01133         writefmt = VDF_AVF;
01134     } else if (Vstring_strcasecmp(tok, "gz") == 0) {
01135         writefmt = VDF_GZ;
01136     } else if (Vstring_strcasecmp(tok, "flat") == 0) {
01137         writefmt = VDF_FLAT;
01138     } else {
01139         Vnm_print(2, "PBParm_parse: Invalid data format (%s) to write!\n",
01140             tok);
01141         return -1;
01142     }
01143     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01144     if (tok[0]=='"') {
01145         strcpy(strnew, "");
01146         while (tok[strlen(tok)-1] != '"') {
01147             strcat(str, tok);
01148             strcat(str, " ");
01149             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01150         }
01151         strcat(str, tok);
01152         strncpy(strnew, str+1, strlen(str)-2);
01153         strcpy(tok, strnew);
01154     }
01155     if (thee->numwrite < (PBEParam_MAXWRITE-1)) {
01156         strncpy(thee->writestem[thee->numwrite], tok, VMAX_ARGLEN);
01157         thee->writetype[thee->numwrite] = writetype;
01158         thee->writefmt[thee->numwrite] = writefmt;
01159         (thee->numwrite)++;
01160     } else {
01161         Vnm_print(2, "PBParm_parse: You have exceeded the maximum number of write
01162 statements!\n");
01163         Vnm_print(2, "PBParm_parse: Ignoring additional write statements!\n");
01164     }
01165     return 1;
01166     VERROR1:
01167     Vnm_print(2, "parsePBE: ran out of tokens!\n");
01168     return -1;
01169 }
01170
01171 VPRIVATE int PBParm_parseWITEMAT(PBParm *thee, Vio *sock) {
01172     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]++;
01173
01174     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01175     if (Vstring_strcasecmp(tok, "poisson") == 0) {
01176         thee->witematflag = 0;
01177     } else if (Vstring_strcasecmp(tok, "full") == 0) {
01178         thee->witematflag = 1;
01179     } else {
01180         Vnm_print(2, "NOsh: Invalid format (%s) while parsing \
01181 WITEMAT keyword!\n", tok);
01182         return -1;
01183     }
01184
01185     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01186     if (tok[0]=='"') {
01187         strcpy(strnew, "");
01188         while (tok[strlen(tok)-1] != '"') {
01189             strcat(str, tok);
01190             strcat(str, " ");
01191             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01192         }
01193         strcat(str, tok);
01194         strncpy(strnew, str+1, strlen(str)-2);
01195         strcpy(tok, strnew);
01196     }
01197     strncpy(thee->witematstem, tok, VMAX_ARGLEN);
01198     thee->setwitemat = 1;
01199     thee->witemat = 1;

```

```

01200     return 1;
01201
01202 VERROR1:
01203     Vnm_print(2, "parsePBE: ran out of tokens!\n");
01204     return -1;
01205
01206 }
01207
01208 VPUBLIC int PBEparm_parseToken(PBEparm *thee, char tok
01209   [VMAX_BUFSIZE],
01210   Vio *sock) {
01211
01212     if (thee == VNULL) {
01213       Vnm_print(2, "parsePBE: got NULL thee!\n");
01214       return -1;
01215     }
01216     if (sock == VNULL) {
01217       Vnm_print(2, "parsePBE: got NULL socket!\n");
01218       return -1;
01219     }
01220
01221     Vnm_print(0, "PBEparm_parseToken: trying %s...\n", tok);
01222
01223     if (Vstring_strcasecmp(tok, "mol") == 0) {
01224       return PBEparm_parseMOL(thee, sock);
01225     } else if (Vstring_strcasecmp(tok, "lpbe") == 0) {
01226       return PBEparm_parseLPBE(thee, sock);
01227     } else if (Vstring_strcasecmp(tok, "npbe") == 0) {
01228       return PBEparm_parseNPBE(thee, sock);
01229     } else if (Vstring_strcasecmp(tok, "lrbpe") == 0) {
01230       return PBEparm_parseLRPBE(thee, sock);
01231     } else if (Vstring_strcasecmp(tok, "nrpb") == 0) {
01232       return PBEparm_parseNRPBE(thee, sock);
01233     } else if (Vstring_strcasecmp(tok, "smpbe") == 0) {
01234       return PBEparm_parseSMPBE(thee, sock);
01235     } else if (Vstring_strcasecmp(tok, "bcfl") == 0) {
01236       return PBEparm_parseBCFL(thee, sock);
01237     } else if (Vstring_strcasecmp(tok, "ion") == 0) {
01238       return PBEparm_parseION(thee, sock);
01239     } else if (Vstring_strcasecmp(tok, "pdie") == 0) {
01240       return PBEparm_parsePDIE(thee, sock);
01241     } else if (Vstring_strcasecmp(tok, "sdens") == 0) {
01242       return PBEparm_parseSDENS(thee, sock);
01243     } else if (Vstring_strcasecmp(tok, "sdie") == 0) {
01244       return PBEparm_parseSDIE(thee, sock);
01245     } else if (Vstring_strcasecmp(tok, "srfm") == 0) {
01246       return PBEparm_parseSRFM(thee, sock);
01247     } else if (Vstring_strcasecmp(tok, "srad") == 0) {
01248       return PBEparm_parseSRAD(thee, sock);
01249     } else if (Vstring_strcasecmp(tok, "swin") == 0) {
01250       return PBEparm_parseSWIN(thee, sock);
01251     } else if (Vstring_strcasecmp(tok, "temp") == 0) {
01252       return PBEparm_parseTEMP(thee, sock);
01253     } else if (Vstring_strcasecmp(tok, "usemap") == 0) {
01254       return PBEparm_parseUSEMAP(thee, sock);
01255     } else if (Vstring_strcasecmp(tok, "calcenergy") == 0) {
01256       return PBEparm_parseCALCENERGY(thee, sock);
01257     } else if (Vstring_strcasecmp(tok, "calcforce") == 0) {
01258       return PBEparm_parseCALCFORCE(thee, sock);
01259     } else if (Vstring_strcasecmp(tok, "write") == 0) {
01260       return PBEparm_parseWRITE(thee, sock);
01261     } else if (Vstring_strcasecmp(tok, "writemat") == 0) {
01262       return PBEparm_parseWITEMAT(thee, sock);
01263
01264 /*-----*/
01265 /* Added by Michael Grabe */
01266 /*-----*/
01267
01268     } else if (Vstring_strcasecmp(tok, "zmem") == 0) {
01269       return PBEparm_parseZMEM(thee, sock);
01270     } else if (Vstring_strcasecmp(tok, "Lmem") == 0) {
01271       return PBEparm_parseLMEM(thee, sock);
01272     } else if (Vstring_strcasecmp(tok, "mdie") == 0) {
01273       return PBEparm_parseMDIE(thee, sock);
01274     } else if (Vstring_strcasecmp(tok, "memv") == 0) {
01275       return PBEparm_parseMEMV(thee, sock);
01276     }
01277
01278 /*-----*/
01279     return 0;

```

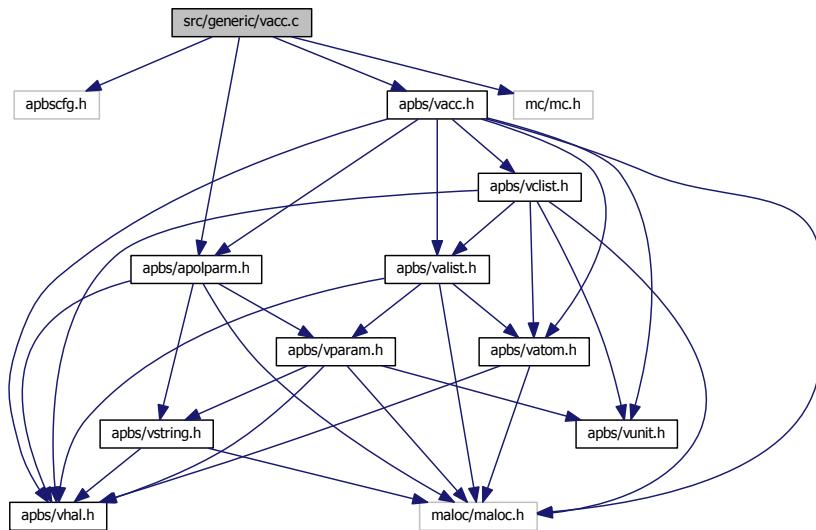
```
01280
01281 }
```

9.63 src/generic/vacc.c File Reference

Class Vacc methods.

```
#include "apbscfg.h"
#include "apbs/vacc.h"
#include "apbs/apolparm.h"
#include "mc/mc.h"
```

Include dependency graph for vacc.c:



Functions

- VPUBLIC unsigned long int [Vacc_memChk \(Vacc *thee\)](#)
Get number of bytes in this object and its members.
- VPRIVATE int [ivdwAccExclus \(Vacc *thee, double center\[3\], double radius, int atomID\)](#)
Determines if a point is within the union of the spheres centered at the atomic centers with radii equal to the sum of their van der Waals radii and the probe radius. Does not include contributions from the specified atom.
- VPUBLIC Vacc * [Vacc_ctor \(Valist *alist, Vclist *clist, double surf_density\)](#)
Construct the accessibility object.
- VPRIVATE int [Vacc_storeParms \(Vacc *thee, Valist *alist, Vclist *clist, double surf_density\)](#)
- VPRIVATE int [Vacc_allocate \(Vacc *thee\)](#)
- VPUBLIC int [Vacc_ctor2 \(Vacc *thee, Valist *alist, Vclist *clist, double surf_density\)](#)
FORTRAN stub to construct the accessibility object.
- VPUBLIC void [Vacc_dtor \(Vacc **thee\)](#)
Destroy object.
- VPUBLIC void [Vacc_dtor2 \(Vacc *thee\)](#)

FORTRAN stub to destroy object.

- VPUBLIC double **Vacc_vdwAcc** (*Vacc *thee*, double *center[3]*)
- VPUBLIC double **Vacc_ivdwAcc** (*Vacc *thee*, double *center[3]*, double *radius*)
- VPUBLIC void **Vacc_splineAccGradAtomNorm** (*Vacc *thee*, double *center[VAPBS_DIM]*, double *win*, double *infrad*, *Vatom *atom*, double **grad*)

*Report gradient of spline-based accessibility with respect to a particular atom normalized by the accessibility value due to that atom at that point (see *Vpmg_splineAccAtom*)*

- VPUBLIC void **Vacc_splineAccGradAtomUnnorm** (*Vacc *thee*, double *center[VAPBS_DIM]*, double *win*, double *infrad*, *Vatom *atom*, double **grad*)

*Report gradient of spline-based accessibility with respect to a particular atom (see *Vpmg_splineAccAtom*)*

- VPUBLIC double **Vacc_splineAccAtom** (*Vacc *thee*, double *center[VAPBS_DIM]*, double *win*, double *infrad*, *Vatom *atom*)

Report spline-based accessibility for a given atom.

- VPRIVATE double **splineAcc** (*Vacc *thee*, double *center[VAPBS_DIM]*, double *win*, double *infrad*, *VclistCell *cell*)

Fast spline-based surface computation subroutine.

- VPUBLIC double **Vacc_splineAcc** (*Vacc *thee*, double *center[VAPBS_DIM]*, double *win*, double *infrad*)

Report spline-based accessibility.

- VPUBLIC void **Vacc_splineAccGrad** (*Vacc *thee*, double *center[VAPBS_DIM]*, double *win*, double *infrad*, double **grad*)

Report gradient of spline-based accessibility.

- VPUBLIC double **Vacc_molAcc** (*Vacc *thee*, double *center[VAPBS_DIM]*, double *radius*)

Report molecular accessibility.

- VPUBLIC double **Vacc_fastMolAcc** (*Vacc *thee*, double *center[VAPBS_DIM]*, double *radius*)

Report molecular accessibility quickly.

- VPUBLIC void **Vacc_writeGMV** (*Vacc *thee*, double *radius*, int *meth*, *Gem *gm*, *char *iodev*, *char *iofmt*, *char *iohost*, *char *iofile*)

- VPUBLIC double **Vacc_SASA** (*Vacc *thee*, double *radius*)

Build the solvent accessible surface (SAS) and calculate the solvent accessible surface area.

- VPUBLIC double **Vacc_totalSASA** (*Vacc *thee*, double *radius*)

Return the total solvent accessible surface area (SASA)

- VPUBLIC double **Vacc_atomSASA** (*Vacc *thee*, double *radius*, *Vatom *atom*)

Return the atomic solvent accessible surface area (SASA)

- VPUBLIC **VaccSurf * VaccSurf_ctor** (*Vmem *mem*, double *probe_radius*, int *nsphere*)

Allocate and construct the surface object; do not assign surface points to positions.

- VPUBLIC int **VaccSurf_ctor2** (*VaccSurf *thee*, *Vmem *mem*, double *probe_radius*, int *nsphere*)

Construct the surface object using previously allocated memory; do not assign surface points to positions.

- VPUBLIC void **VaccSurf_dtor** (*VaccSurf **thee*)

Destroy the surface object and free its memory.

- VPUBLIC void **VaccSurf_dtor2** (*VaccSurf *thee*)

Destroy the surface object.

- VPUBLIC **VaccSurf * Vacc_atomSurf** (*Vacc *thee*, *Vatom *atom*, *VaccSurf *ref*, double *prad*)

Set up an array of points corresponding to the SAS due to a particular atom.

- VPUBLIC **VaccSurf * VaccSurf_refSphere** (*Vmem *mem*, int *npts*)

Set up an array of points for a reference sphere of unit radius.

- VPUBLIC **VaccSurf * Vacc_atomSASPoints** (*Vacc *thee*, double *radius*, *Vatom *atom*)

Get the set of points for this atom's solvent-accessible surface.

- VPUBLIC void **Vacc_splineAccGradAtomNorm4** (*Vacc *thee*, double *center[VAPBS_DIM]*, double *win*, double *infrad*, *Vatom *atom*, double **grad*)

Report gradient of spline-based accessibility with respect to a particular atom normalized by a 4th order accessibility value due to that atom at that point (see Vpmg_splineAccAtom)

- VPUBLIC void `Vacc_splineAccGradAtomNorm3` (`Vacc *thee`, double `center[VAPBS_DIM]`, double `win`, double `infrad`, `Vatom *atom`, double `*grad`)

Report gradient of spline-based accessibility with respect to a particular atom normalized by a 3rd order accessibility value due to that atom at that point (see Vpmg_splineAccAtom)

- VPUBLIC void `Vacc_atomdSAV` (`Vacc *thee`, double `srad`, `Vatom *atom`, double `*dSA`)

Get the derivative of solvent accessible volume.

- VPRIVATE double `Vacc_SASAPOS` (`Vacc *thee`, double `radius`)

- VPRIVATE double `Vacc_atomSASAPOS` (`Vacc *thee`, double `radius`, `Vatom *atom`, int `mode`)

- VPUBLIC void `Vacc_atomdSASA` (`Vacc *thee`, double `dpos`, double `srad`, `Vatom *atom`, double `*dSA`)

Get the derivative of solvent accessible area.

- VPUBLIC void `Vacc_totalAtomdSASA` (`Vacc *thee`, double `dpos`, double `srad`, `Vatom *atom`, double `*dSA`)

Testing purposes only.

- VPUBLIC void `Vacc_totalAtomdSAV` (`Vacc *thee`, double `dpos`, double `srad`, `Vatom *atom`, double `*dSA`, `Vclist *clist`)

Total solvent accessible volume.

- VPUBLIC double `Vacc_totalSAV` (`Vacc *thee`, `Vclist *clist`, `APOLparm *apolparm`, double `radius`)

Return the total solvent accessible volume (SAV)

- int `Vacc_wcaEnergyAtom` (`Vacc *thee`, `APOLparm *apolparm`, `Valist *alist`, `Vclist *clist`, int `iatom`, double `*value`)

Calculate the WCA energy for an atom.

- VPUBLIC int `Vacc_wcaEnergy` (`Vacc *acc`, `APOLparm *apolparm`, `Valist *alist`, `Vclist *clist`)

Return the WCA integral energy.

- VPUBLIC int `Vacc_wcaForceAtom` (`Vacc *thee`, `APOLparm *apolparm`, `Vclist *clist`, `Vatom *atom`, double `*force`)

Return the WCA integral force.

9.63.1 Detailed Description

Class Vacc methods.

Author

Nathan Baker

Version

Id:

`vacc.c` 1750 2012-07-18 18:34:27Z tuckerbeck

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](#).

9.63.2 Function Documentation

9.63.2.1 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

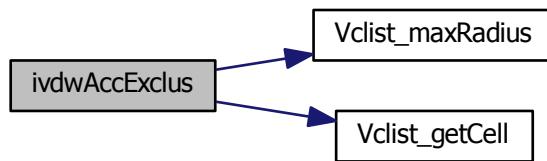
Nathan Baker

Parameters

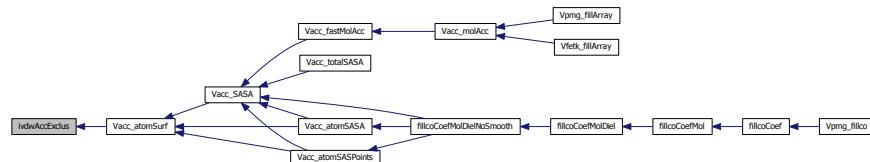
| | |
|---------------|--------------------------------------|
| <i>center</i> | Accessibility object |
| <i>radius</i> | Position to test |
| <i>atomID</i> | Radius of probe ID of atom to ignore |

Definition at line 86 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.63.2.2 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

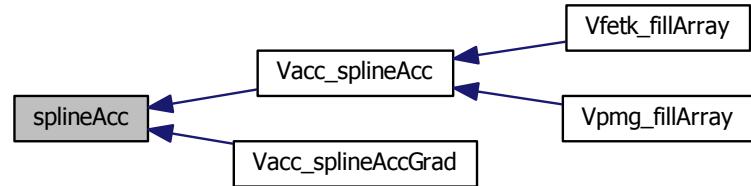
| | |
|---------------|--|
| <i>center</i> | Accessibility object |
| <i>win</i> | Point at which the acc is to be evaluated |
| <i>infrad</i> | Spline window |
| <i>cell</i> | Radius to inflate atomic radius Cell of atom objects |

Definition at line 499 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.63.2.3 VPRIVATE int Vacc_allocate (Vacc *thee)

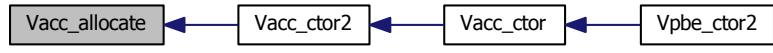
Allocate (and clear) space for storage

Definition at line 199 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

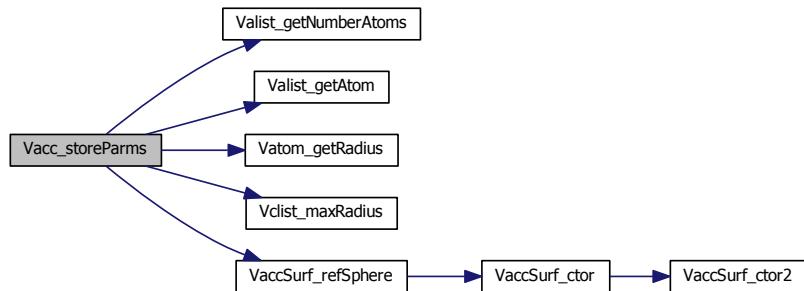


9.63.2.4 VPRIVATE int Vacc_storeParms (Vacc * *thee*, Valist * *alist*, Vclist * *clist*, double *surf_density*)

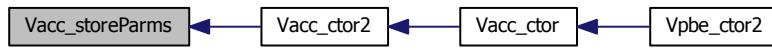
Check and store parameters passed to constructor

Definition at line 154 of file [vacc.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.64 vacc.c

```

00001
00057 #include "apbscfg.h"
00058 #include "apbs/vacc.h"
00059 #include "apbs/apolparm.h"
00060
00061 #if defined(HAVE_MC_H)
00062 #include "mc/mc.h"
00063 #endif
00064
00065 VEMBED(rcsid="$Id: vacc.c 1750 2012-07-18 18:34:27Z tuckerbeck $" )
00066
00067 #if !defined(VINLINE_VACC)
00068
00069 VPUBLIC unsigned long int Vacc_memChk (Vacc *thee) {
00070     if (thee == VNULL)
00071         return 0;
00072     return Vmem_bytes(thee->mem);
00073 }
00074
00075 #endif /* if !defined(VINLINE_VACC) */
00076
00086 VPRIIVATE int ivdwAccExclus(
00087     Vacc *thee,
00088     double center[3],
00089     double radius,
00090     int atomID
00091 ) {
00092
00093     int iatom;
00094     double dist2,
00095             *apos;
00096     Vatom *atom;
00097     VclistCell *cell;
00098
00099     VASSERT(thee != VNULL);
00100
00101     /* We can only test probes with radii less than the max specified */
00102     if (radius > Vclist_maxRadius(thee->clist)) {
00103         Vnm_print(2,
00104             "Vacc_ivdwAcc: got radius (%g) bigger than max radius (%g)\n",
00105             radius, Vclist_maxRadius(thee->clist));
00106     VASSERT(0);
00107     }
00108
00109     /* Get the relevant cell from the cell list */
00110     cell = Vclist_getCell(thee->clist, center);
00111
00112     /* If we have no cell, then no atoms are nearby and we're definitely
00113      * accessible */
00114     if (cell == VNULL) {
00115         return 1;
00116     }
00117
00118     /* Otherwise, check for overlap with the atoms in the cell */
00119     for (iatom=0; iatom<cell->natoms; iatom++) {
00120         atom = cell->atoms[iatom];
00121
00122         // We don't actually need to test this if the atom IDs do match; don't
00123         // compute this if we're comparing atom against itself.
00124         if (atom->id == atomID) continue;
00125     }
  
```

```

00125     apos = atom->position;
00126     dist2 = VSQR(center[0]-apos[0]) + VSQR(center[1]-apos[1])
00127     + VSQR(center[2]-apos[2]);
00128     if (dist2 < VSQR(atom->radius+radius)) {
00129         return 0;
00130     }
00131 }
00132
00133     /* If we're still here, then the point is accessible */
00134     return 1;
00135
00136 }
00137
00138 VPUBLIC Vacc* Vacc_ctor(Valist *alist,
00139                         Vclist *clist,
00140                         double surf_density /* Surface density */)
00141 {
00142
00143
00144     Vacc *thee = VNULL;
00145
00146     /* Set up the structure */
00147     thee = (Vacc*)Vmem_malloc(VNULL, 1, sizeof(Vacc));
00148     VASSERT( thee != VNULL );
00149     VASSERT( Vacc_ctor2(thee, alist, clist, surf_density) );
00150     return thee;
00151 }
00152
00153 VPRIPRIVATE int Vacc_storeParms(Vacc *thee,
00154                                   Valist *alist,
00155                                   Vclist *clist,
00156                                   double surf_density /* Surface density */)
00157 {
00158
00159     int nsphere,
00160         iatom;
00161     double maxrad = 0.0,
00162             maxarea,
00163             rad;
00164     Vatom *atom;
00165
00166     if (alist == VNULL) {
00167         Vnm_print(2, "Vacc_storeParms: Got NULL Valist!\n");
00168         return 0;
00169     } else thee->alist = alist;
00170     if (clist == VNULL) {
00171         Vnm_print(2, "Vacc_storeParms: Got NULL Vclist!\n");
00172         return 0;
00173     } else thee->clist = clist;
00174     thee->surf_density = surf_density;
00175
00176     /* Loop through the atoms to determine the maximum radius */
00177     maxrad = 0.0;
00178     for (iatom=0; iatom<Valist_getNumberAtoms(alist);
00179           iatom++) {
00180         atom = Valist_getAtom(alist, iatom);
00181         rad = Vatom_getRadius(atom);
00182         if (rad > maxrad) maxrad = rad;
00183     }
00184     maxrad = maxrad + Vclist_maxRadius(thee->clist);
00185
00186     maxarea = 4.0*VPI*maxrad*maxrad;
00187     nsphere = (int)ceil(maxarea*surf_density);
00188
00189     Vnm_print(0, "Vacc_storeParms: Surf. density = %g\n", surf_density);
00190     Vnm_print(0, "Vacc_storeParms: Max area = %g\n", maxarea);
00191     thee->refSphere = VaccSurf_refSphere(thee->mem
00192 , nsphere);
00193     Vnm_print(0, "Vacc_storeParms: Using %d-point reference sphere\n",
00194               thee->refSphere->npts);
00195
00196     return 1;
00197 }
00198
00199 VPRIPRIVATE int Vacc_allocate(Vacc *thee) {
00200
00201     int i,
00202         natoms;
00203
00204     natoms = Valist_getNumberAtoms(thee->alist);
00205

```

```

00206     thee->atomFlags = (int*)Vmem_malloc(thee->mem, natoms, sizeof(
00207         int));
00208     if (thee->atomFlags == VNULL) {
00209         Vnm_print(2,
00210             "Vacc_allocate: Failed to allocate %d (int)s for atomFlags!\n",
00211             natoms);
00212     }
00213     for (i=0; i<natoms; i++) (thee->atomFlags)[i] = 0;
00214
00215     return 1;
00216 }
00217
00218
00219 VPUBLIC int Vacc_ctor2(Vacc *thee,
00220                         Valist *alist,
00221                         Vclist *clist,
00222                         double surf_density
00223                         ) {
00224
00225     /* Check and store parameters */
00226     if (!Vacc_storeParms(thee, alist, clist, surf_density)) {
00227         Vnm_print(2, "Vacc_ctor2: parameter check failed!\n");
00228         return 0;
00229     }
00230
00231     /* Set up memory management object */
00232     thee->mem = Vmem_ctor("APBS::VACC");
00233     if (thee->mem == VNULL) {
00234         Vnm_print(2, "Vacc_ctor2: memory object setup failed!\n");
00235         return 0;
00236     }
00237
00238     /* Setup and check probe */
00239     thee->srf = VNULL;
00240
00241     /* Allocate space */
00242     if (!Vacc_allocate(thee)) {
00243         Vnm_print(2, "Vacc_ctor2: memory allocation failed!\n");
00244         return 0;
00245     }
00246
00247     return 1;
00248 }
00249
00250
00251 VPUBLIC void Vacc_dtors(Vacc **thee) {
00252
00253     if ((*thee) != VNULL) {
00254         Vacc_dtors(*thee);
00255         Vmem_free(VNULL, 1, sizeof(Vacc), (void **)thee);
00256         (*thee) = VNULL;
00257     }
00258
00259 }
00260
00261 VPUBLIC void Vacc_dtors2(Vacc *thee) {
00262
00263     int i,
00264         natoms;
00265
00266     natoms = Valist_getNumberAtoms(thee->alist);
00267     Vmem_free(thee->mem, natoms, sizeof(int), (void **)&(thee->atomFlags
00268     ));
00269
00270     if (thee->refSphere != VNULL) {
00271         VaccSurf_dtors(&(thee->refSphere));
00272         thee->refSphere = VNULL;
00273     }
00274     if (thee->srf != VNULL) {
00275         for (i=0; i<natoms; i++) VaccSurf_dtors(&(thee->srf[i]
00276     ));
00277         Vmem_free(thee->mem, natoms, sizeof(VaccSurf *),
00278             (void **)&(thee->srf));
00279         thee->srf = VNULL;
00280     }
00281
00282     Vmem_dtors(&(thee->mem));
00283 }
00284

```

```

00283 VPUBLIC double Vacc_vdwAcc(Vacc *thee,
00284                               double center[3]
00285                               ) {
00286
00287     VclistCell *cell;
00288     Vatom *atom;
00289     int iatom;
00290     double *apos,
00291             dist2;
00292
00293     /* Get the relevant cell from the cell list */
00294     cell = Vclist_getCell(thee->clist, center);
00295
00296     /* If we have no cell, then no atoms are nearby and we're definitely
00297      * accessible */
00298     if (cell == VNULL) return 1.0;
00299
00300     /* Otherwise, check for overlap with the atoms in the cell */
00301     for (iatom=0; iatom<cell->natoms; iatom++) {
00302         atom = cell->atoms[iatom];
00303         apos = VatomGetPosition(atom);
00304         dist2 = VSQR(center[0]-apos[0]) + VSQR(center[1]-apos[1])
00305             + VSQR(center[2]-apos[2]);
00306         if (dist2 < VSQR(Vatom_getRadius(atom))) return 0.0;
00307     }
00308
00309     /* If we're still here, then the point is accessible */
00310     return 1.0;
00311 }
00312
00313 VPUBLIC double Vacc_ivdwAcc(Vacc *thee,
00314                               double center[3],
00315                               double radius
00316                               ) {
00317
00318     return (double)ivdwAccExclus(thee, center, radius, -1);
00319
00320 }
00321
00322 VPUBLIC void Vacc_splineAccGradAtomNorm(Vacc *
00323     thee,
00324                               double center[VAPBS_DIM],
00325                               double win,
00326                               double infrad,
00327                               Vatom *atom,
00328                               double *grad
00329                               ) {
00330
00331     int i;
00332     double dist,
00333             *apos,
00334             arad,
00335             sm,
00336             sm2,
00337             w2i, /* inverse of win squared */
00338             w3i, /* inverse of win cubed */
00339             mygrad,
00340             mychi = 1.0; /* Char. func. value for given atom */
00341
00342     VASSERT(thee != NULL);
00343
00344     /* Inverse squared window parameter */
00345     w2i = 1.0/(win*win);
00346     w3i = 1.0/(win*win*win);
00347
00348     /* The grad is zero by default */
00349     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
00350
00351     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
00352      * *** MAGNITUDE OF THE FORCE *** */
00353     apos = VatomGetPosition(atom);
00354     /* Zero-radius atoms don't contribute */
00355     if (Vatom_getRadius(atom) > 0.0) {
00356         arad = Vatom_getRadius(atom) + infrad;
00357         dist = VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
00358             + VSQR(apos[2]-center[2]);
00359         /* If we're inside an atom, the entire characteristic function
00360          * will be zero and the grad will be zero, so we can stop */
00361         if (dist < (arad - win)) return;
00362         /* Likewise, if we're outside the smoothing window, the characteristic
00363          * function is unity and the grad will be zero, so we can stop */
00364

```

```

00363     else if (dist > (arad + win)) return;
00364     /* Account for floating point error at the border
00365      * NAB: COULDN'T THESE TESTS BE COMBINED AS BELOW
00366      * (Vacc_splineAccAtom)? */
00367     else if ((VABS(dist - (arad - win)) < VSMALL) ||
00368              (VABS(dist - (arad + win)) < VSMALL)) return;
00369     /* If we're inside the smoothing window */
00370     else {
00371         sm = dist - arad + win;
00372         sm2 = VSQR(sm);
00373         mychi = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
00374         mygrad = 1.5*sm*w2i - 0.75*sm2*w3i;
00375     }
00376     /* Now assemble the grad vector */
00377     VASSERT(mychi > 0.0);
00378     for (i=0; i<VAPBS_DIM; i++)
00379         grad[i] = -(mygrad/mychi)*((center[i] - apos[i])/dist);
00380     }
00381 }
00382
00383 VPUBLIC void Vacc_splineAccGradAtomUnnorm(Vacc
00384     *thee,
00385             double center[VAPBS_DIM],
00386             double win,
00387             double infrad,
00388             Vatom *atom,
00389             double *grad
00390     ) {
00391     int i;
00392     double dist,
00393         *apos,
00394         arad,
00395         sm,
00396         sm2,
00397         w2i, /* Inverse of win squared */
00398         w3i, /* Inverse of win cubed */
00399         mygrad,
00400         mychi = 1.0;           /* Char. func. value for given atom */
00401
00402     VASSERT(thee != NULL);
00403
00404     /* Inverse squared window parameter */
00405     w2i = 1.0/(win*win);
00406     w3i = 1.0/(win*win*win);
00407
00408     /* The grad is zero by default */
00409     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
00410
00411     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
00412      * *** MAGNITUDE OF THE FORCE *** */
00413     apos = Vatom_getPosition(atom);
00414
00415     /* Zero-radius atoms don't contribute */
00416     if (Vatom_getRadius(atom) > 0.0) {
00417         arad = Vatom_getRadius(atom) + infrad;
00418         dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
00419                     + VSQR(apos[2]-center[2]));
00420         /* If we're inside an atom, the entire characteristic function
00421          * will be zero and the grad will be zero, so we can stop */
00422         if (dist < (arad - win)) return;
00423         /* Likewise, if we're outside the smoothing window, the characteristic
00424          * function is unity and the grad will be zero, so we can stop */
00425         else if (dist > (arad + win)) return;
00426         /* Account for floating point error at the border
00427          * NAB: COULDN'T THESE TESTS BE COMBINED AS BELOW
00428          * (Vacc_splineAccAtom)? */
00429         else if ((VABS(dist - (arad - win)) < VSMALL) ||
00430                  (VABS(dist - (arad + win)) < VSMALL)) return;
00431         /* If we're inside the smoothing window */
00432         else {
00433             sm = dist - arad + win;
00434             sm2 = VSQR(sm);
00435             mychi = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
00436             mygrad = 1.5*sm*w2i - 0.75*sm2*w3i;
00437         }
00438         /* Now assemble the grad vector */
00439         VASSERT(mychi > 0.0);
00440         for (i=0; i<VAPBS_DIM; i++)
00441             grad[i] = -(mygrad)*((center[i] - apos[i])/dist);
00442     }
00443 }
```

```

00443
00444 VPUBLIC double Vacc_splineAccAtom(Vacc *thee,
00445                               double center[VAPBS_DIM],
00446                               double win,
00447                               double infrad,
00448                               Vatom *atom
00449 ) {
00450
00451     double dist,
00452         *apos,
00453         arad,
00454         sm,
00455         sm2,
00456         w2i, /* Inverse of win squared */
00457         w3i, /* Inverse of win cubed */
00458         value,
00459         stot,
00460         sctot;
00461
00462     VASSERT(thee != NULL);
00463
00464     /* Inverse squared window parameter */
00465     w2i = 1.0/(win*win);
00466     w3i = 1.0/(win*win*win);
00467
00468     apos = Vatom_getPosition(atom);
00469     /* Zero-radius atoms don't contribute */
00470     if (Vatom_getRadius(atom) > 0.0) {
00471         arad = Vatom_getRadius(atom) + infrad;
00472         stot = arad + win;
00473         sctot = VMAX2(0, (arad - win));
00474         dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
00475             + VSQR(apos[2]-center[2]));
00476         /* If we're inside an atom, the entire characteristic function
00477          * will be zero */
00478         if ((dist < sctot) || (VABS(dist - stot) < VSMALL)) {
00479             value = 0.0;
00480             /* We're outside the smoothing window */
00481         } else if ((dist > stot) || (VABS(dist - stot) < VSMALL)) {
00482             value = 1.0;
00483             /* We're inside the smoothing window */
00484         } else {
00485             sm = dist - arad + win;
00486             sm2 = VSQR(sm);
00487             value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
00488         }
00489     } else value = 1.0;
00490
00491     return value;
00492 }
00493
00494 VPRIVATE double splineAcc(
00500     Vacc *thee,
00501     double center[VAPBS_DIM],
00502     double win,
00503     double infrad,
00504     VclistCell *cell
00505 ) {
00506
00507     int atomID, iatom;
00508     Vatom *atom;
00509     double value = 1.0;
00510
00511     VASSERT(thee != NULL);
00512
00513     /* Now loop through the atoms assembling the characteristic function */
00514     for (iatom=0; iatom<cell->natoms; iatom++) {
00515
00516         atom = cell->atoms[iatom];
00517         atomID = atom->id;
00518
00519         /* Check to see if we've counted this atom already */
00520         if ( !(thee->atomFlags[atomID]) ) {
00521
00522             thee->atomFlags[atomID] = 1;
00523             value *= Vacc_splineAccAtom(thee, center, win,
00524                 infrad, atom);
00525
00526             if (value < VSMALL) return value;
00527         }
00528     }

```

```

00529     return value;
00530 }
00531 }
00532 }
00533 }
00534 VPUBLIC double Vacc_splineAcc(Vacc *thee, double center[
00535     VAPBS_DIM], double win,
00536     double infrad) {
00537     VclistCell *cell;
00538     Vatom *atom;
00539     int iatom, atomID;
00540 }
00541 }
00542 VASSERT(thee != NULL);
00543 }
00544 if (Vclist_maxRadius(thee->clist) < (win + infrad)) {
00545     Vnm_print(2, "Vacc_splineAcc: Vclist has max_radius=%g;\n",
00546             Vclist_maxRadius(thee->clist));
00547     Vnm_print(2, "Vacc_splineAcc: Insufficient for win=%g, infrad=%g\n",
00548             win, infrad);
00549     VASSERT(0);
00550 }
00551 }
00552 /* Get a cell or VNULL; in the latter case return 1.0 */
00553 cell = Vclist_getCell(thee->clist, center);
00554 if (cell == VNULL) return 1.0;
00555 }
00556 /* First, reset the list of atom flags
00557  * NAB: THIS SEEMS VERY INEFFICIENT */
00558 for (iatom=0; iatom<cell->natoms; iatom++) {
00559     atom = cell->atoms[iatom];
00560     atomID = atom->id;
00561     thee->atomFlags[atomID] = 0;
00562 }
00563 }
00564 return splineAcc(thee, center, win, infrad, cell);
00565 }
00566 }
00567 VPUBLIC void Vacc_splineAccGrad(Vacc *thee, double center
00568 [VAPBS_DIM],
00569     double win, double infrad, double *grad) {
00570     int iatom, i, atomID;
00571     double acc = 1.0;
00572     double tgrad[VAPBS_DIM];
00573     VclistCell *cell;
00574     Vatom *atom = VNULL;
00575 }
00576 VASSERT(thee != NULL);
00577 }
00578 if (Vclist_maxRadius(thee->clist) < (win + infrad)) {
00579     Vnm_print(2, "Vacc_splineAccGrad: Vclist max_radius=%g;\n",
00580             Vclist_maxRadius(thee->clist));
00581     Vnm_print(2, "Vacc_splineAccGrad: Insufficient for win=%g, infrad=%g\n"
00582             win, infrad);
00583     VASSERT(0);
00584 }
00585 }
00586 /* Reset the gradient */
00587 for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
00588 }
00589 /* Get the cell; check for nullity */
00590 cell = Vclist_getCell(thee->clist, center);
00591 if (cell == VNULL) return;
00592 }
00593 /* Reset the list of atom flags */
00594 for (iatom=0; iatom<cell->natoms; iatom++) {
00595     atom = cell->atoms[iatom];
00596     atomID = atom->id;
00597     thee->atomFlags[atomID] = 0;
00598 }
00599 }
00600 /* Get the local accessibility */
00601 acc = splineAcc(thee, center, win, infrad, cell);
00602 }
00603 /* Accumulate the gradient of all local atoms */
00604 if (acc > VSMALL) {
00605     for (iatom=0; iatom<cell->natoms; iatom++) {
00606         atom = cell->atoms[iatom];
00607     }
00608 }
```

```

00607         Vacc_splineAccGradAtomNorm(thee, center,
00608             win, infrad, atom, tgrad);
00609             for (i=0; i<VAPBS_DIM; i++) grad[i] += tgrad[i];
00610     }
00611     for (i=0; i<VAPBS_DIM; i++) grad[i] *= -acc;
00612 }
00613
00614 VPUBLIC double Vacc_molAcc(Vacc *thee, double center[VAPBS_DIM]
00615     , double radius) {
00616
00617     double rc;
00618
00619     /* ***** CHECK IF OUTSIDE ATOM+PROBE RADIUS SURFACE ***** */
00620     if (Vacc_ivdwAcc(thee, center, radius) == 1.0) {
00621
00622         /* Vnm_print(2, "DEBUG: ivdwAcc = 1.0\n"); */
00623         rc = 1.0;
00624
00625     /* ***** CHECK IF INSIDE ATOM RADIUS SURFACE ***** */
00626     } else if (Vacc_vdwAcc(thee, center) == 0.0) {
00627
00628         /* Vnm_print(2, "DEBUG: vdwAcc = 0.0\n"); */
00629         rc = 0.0;
00630
00631     /* ***** CHECK IF OUTSIDE MOLECULAR SURFACE ***** */
00632     } else {
00633
00634         /* Vnm_print(2, "DEBUG: calling fastMolAcc...\n"); */
00635         rc = Vacc_fastMolAcc(thee, center, radius);
00636
00637     }
00638
00639     return rc;
00640
00641 }
00642
00643 VPUBLIC double Vacc_fastMolAcc(Vacc *thee, double center[
00644     VAPBS_DIM],
00645     double radius) {
00646
00647     Vatom *atom;
00648     VaccSurf *surf;
00649     VclistCell *cell;
00650     int ipt, iatom, atomID;
00651     double dist2, rad2;
00652
00653     rad2 = radius*radius;
00654
00655     /* Check to see if the SAS has been defined */
00656     if (thee->surf == VNULL) Vacc_SASA(thee, radius);
00657
00658     /* Get the cell associated with this point */
00659     cell = Vclist_getCell(thee->clist, center);
00660     if (cell == VNULL) {
00661         Vnm_print(2, "Vacc_fastMolAcc: unexpected VNULL VclistCell!\n");
00662         return 1.0;
00663     }
00664
00665     /* Loop through all the atoms in the cell */
00666     for (iatom=0; iatom<cell->natoms; iatom++) {
00667         atom = cell->atoms[iatom];
00668         atomID = Vatom_getAtomID(atom);
00669         surf = thee->surf[atomID];
00670         /* Loop through all SAS points associated with this atom */
00671         for (ipt=0; ipt<surf->npts; ipt++) {
00672             /* See if we're within a probe radius of the point */
00673             dist2 = VSQR(center[0]-(surf->xpts[ipt]))
00674                 + VSQR(center[1]-(surf->ypts[ipt]))
00675                 + VSQR(center[2]-(surf->zpts[ipt]));
00676             if (dist2 < rad2) return 1.0;
00677         }
00678
00679     /* If all else failed, we are not inside the molecular surface */
00680     return 0.0;
00681 }
00682
00683
00684 #if defined(HAVE_MC_H)

```

```

00685 VPUBLIC void Vacc_writeGMV(Vacc *thee, double radius, int meth, Gem *gm,
00686     char *iodev, char *iofmt, char *iohost, char *iofile) {
00687
00688     double *accVals[MAXV], coord[3];
00689     Vio *sock;
00690     int ivert, icoord;
00691
00692     for (ivert=0; ivert<MAXV; ivert++) accVals[ivert] = VNULL;
00693     accVals[0] = (void *)Vmem_malloc(thee->mem, Gem_numVV(gm), sizeof(double
00694     ));
00694     accVals[1] = (void *)Vmem_malloc(thee->mem, Gem_numVV(gm), sizeof(double
00695     ));
00695     for (ivert=0; ivert<Gem_numVV(gm); ivert++) {
00696         for (icoord=0; icoord<3; icoord++)
00697             coord[icoord] = VV_coord(Gem_VV(gm, ivert), icoord);
00698         if (meth == 0) {
00699             accVals[0][ivert] = Vacc_molAcc(thee, coord, radius);
00700             accVals[1][ivert] = Vacc_molAcc(thee, coord, radius);
00701         } else if (meth == 1) {
00702             accVals[0][ivert] = Vacc_ivdwAcc(thee, coord, radius);
00703             accVals[1][ivert] = Vacc_ivdwAcc(thee, coord, radius);
00704         } else if (meth == 2) {
00705             accVals[0][ivert] = Vacc_vdwAcc(thee, coord);
00706             accVals[1][ivert] = Vacc_vdwAcc(thee, coord);
00707         } else VASSERT(0);
00708     }
00709     sock = Vio_ctor(iodev, iofmt, iohost, iofile, "w");
00710     Gem_writeGMV(gm, sock, 1, accVals);
00711     Vio_dtor(&sock);
00712     Vmem_free(thee->mem, Gem_numVV(gm), sizeof(double),
00713     (void **) &(accVals[0]));
00714     Vmem_free(thee->mem, Gem_numVV(gm), sizeof(double),
00715     (void **) &(accVals[1]));
00716 }
00717 #endif /* defined(HAVE_MC_H) */
00718
00719 VPUBLIC double Vacc_SASA(Vacc *thee,
00720                             double radius
00721                             ) {
00722
00723     int i,
00724         natom;
00725     double area;
00726     /* *apos; // gcc says unused
00727     Vatom *atom;
00728     VaccSurf *asurf;
00729
00730     time_t ts; // PCE: temp
00731     ts = clock();
00732
00733     //unsigned long long mbeg; // gcc says unused
00734
00735     natom = Valist_getNumberAtoms(thee->alist);
00736
00737     /* Check to see if we need to build the surface */
00738     if (thee->surf == VNULL) {
00739         thee->surf = Vmem_malloc(thee->mem, natom, sizeof(VaccSurf
00740         *));
00741     }
00742 #if defined(DEBUG_MAC OSX_OCL) || defined(DEBUG_MAC OSX_STANDARD)
00743 #include "mach_chud.h"
00744     machm_(&mbeg);
00745 #pragma omp parallel for private(i,atom)
00746 #endif
00747     for (i=0; i<natom; i++) {
00748         atom = Valist_getAtom(thee->alist, i);
00749         /* NOTE: RIGHT NOW WE DO THIS FOR THE ENTIRE MOLECULE WHICH IS
00750          * INCREDIBLY INEFFICIENT, PARTICULARLY DURING FOCUSING!!! */
00751         thee->surf[i] = Vacc_atomSurf(thee, atom, thee->
00752         refSphere,
00753         radius);
00754     }
00755
00756     /* Calculate the area */
00757     area = 0.0;
00758     for (i=0; i<natom; i++) {
00759         atom = Valist_getAtom(thee->alist, i);
00760         asurf = thee->surf[i];
00761         /* See if this surface needs to be rebuilt */
00762         if (asurf->probe_radius != radius) {

```

```

00762     Vnm_print(2, "Vacc_SASA: Warning -- probe radius changed from %g
00763         to %g!\n",
00764         asurf->probe_radius, radius);
00765         VaccSurf_dtor2(asurf);
00766         thee->surf[i] = Vacc_atomSurf(thee, atom, thee->
00767             refSphere, radius);
00768         asurf = thee->surf[i];
00769     }
00770     area += (asurf->area);
00771 #if defined(DEBUG_MAC OSX_OCL) || defined(DEBUG_MAC OSX_STANDARD)
00772 mets_(&mbeg, "Vacc_SASA - Parallel");
00773 #endif
00774
00775     printf("Vacc_SASA: Time elapsed: %f\n", ((double)clock() - ts) /
00776         CLOCKS_PER_SEC);
00777     return area;
00778 }
00779
00780 VPUBLIC double Vacc_totalSASA(Vacc *thee, double radius) {
00781     return Vacc_SASA(thee, radius);
00782 }
00783
00784 }
00785
00786 VPUBLIC double Vacc_atomSASA(Vacc *thee, double radius, Vatom
00787     *atom) {
00788     VaccSurf *asurf;
00789     int id;
00790
00791     if (thee->surf == VNULL) Vacc_SASA(thee, radius);
00792
00793     id = Vatom_getAtomID(atom);
00794     asurf = thee->surf[id];
00795
00796     /* See if this surface needs to be rebuilt */
00797     if (asurf->probe_radius != radius) {
00798         Vnm_print(2, "Vacc_SASA: Warning -- probe radius changed from %g to
00799             %g!\n",
00800             asurf->probe_radius, radius);
00801         VaccSurf_dtor2(asurf);
00802         thee->surf[id] = Vacc_atomSurf(thee, atom, thee->
00803             refSphere, radius);
00804         asurf = thee->surf[id];
00805     }
00806
00807     return asurf->area;
00808 }
00809
00810 VPUBLIC VaccSurf* VaccSurf_ctor(Vmem *mem, double
00811     probe_radius, int nsphere) {
00812     VaccSurf *thee;
00813
00814     //thee = Vmem_malloc(mem, 1, sizeof(Vacc) );
00815     if (nsphere >= MAX_SPHERE PTS) {
00816         Vnm_print(2, "VaccSurf_ctor: Error! The requested number of grid points
00817         (%d) exceeds the maximum (%d)!\n", nsphere, MAX_SPHERE PTS);
00818         Vnm_print(2, "VaccSurf_ctor: Please check the variable MAX_SPHERE PTS to
00819         reset.\n");
00820         VASSEERT(0);
00821     }
00822     thee = (VaccSurf*)calloc(1,sizeof(Vacc));
00823     VASSEERT( VaccSurf_ctor2(thee, mem, probe_radius, nsphere) );
00824
00825     return thee;
00826 }
00827
00828 VPUBLIC int VaccSurf_ctor2(VaccSurf *thee, Vmem *mem,
00829     double probe_radius,
00830     int nsphere) {
00831
00832     if (thee == VNULL)
00833         return 0;
00834
00835     thee->mem = mem;
00836     thee->npts = nsphere;
00837     thee->probe_radius = probe_radius;

```

```

00833     thee->area = 0.0;
00834
00835     if (thee->npts > 0) {
00836         thee->xpts = Vmem_malloc(thee->mem, thee->npts, sizeof(
00837             double));
00838         thee->ypts = Vmem_malloc(thee->mem, thee->npts, sizeof(
00839             double));
00840         thee->zpts = Vmem_malloc(thee->mem, thee->npts, sizeof(
00841             double));
00842         thee->bpts = Vmem_malloc(thee->mem, thee->npts, sizeof(char)
00843     );
00844     } else {
00845         thee->xpts = VNULL;
00846         thee->ypts = VNULL;
00847         thee->zpts = VNULL;
00848         thee->bpts = VNULL;
00849     }
00850 VPUBLIC void VaccSurf_dtor(VaccSurf **thee) {
00851
00852     Vmem *mem;
00853
00854     if ((*thee) != VNULL) {
00855         mem = (*thee)->mem;
00856         VaccSurf_dtor2(*thee);
00857         //Vmem_free(mem, 1, sizeof(VaccSurf), (void **)thee);
00858         free(*thee);
00859         (*thee) = VNULL;
00860     }
00861
00862 }
00863
00864 VPUBLIC void VaccSurf_dtor2(VaccSurf *thee) {
00865
00866     if (thee->npts > 0) {
00867         Vmem_free(thee->mem, thee->npts, sizeof(double),
00868                 (void **)(&(thee->xpts)));
00869         Vmem_free(thee->mem, thee->npts, sizeof(double),
00870                 (void **)(&(thee->ypts)));
00871         Vmem_free(thee->mem, thee->npts, sizeof(double),
00872                 (void **)(&(thee->zpts)));
00873         Vmem_free(thee->mem, thee->npts, sizeof(char),
00874                 (void **)(&(thee->bpts)));
00875     }
00876
00877 }
00878
00879 VPUBLIC VaccSurf* Vacc_atomSurf(Vacc *thee,
00880                                     Vatom *atom,
00881                                     VaccSurf *ref,
00882                                     double prad
00883                                     ) {
00884
00885     VaccSurf *surf;
00886     int i,
00887         j,
00888         npts,
00889         atomID;
00890     double arad,
00891         rad,
00892         pos[3],
00893         *apos;
00894     char bpts[MAX_SPHERE PTS];
00895
00896     /* Get atom information */
00897     arad = Vatom_getRadius(atom);
00898     apos = Vatom_getPosition(atom);
00899     atomID = Vatom_getAtomID(atom);
00900
00901     if (arad < VSMALL) {
00902         return VaccSurf_ctor(thee->mem, prad, 0);
00903     }
00904
00905     rad = arad + prad;
00906
00907     /* Determine which points will contribute */
00908     npts = 0;
00909     for (i=0; i<ref->npts; i++) {

```

```

00910     /* Reset point flag: zero-radius atoms do not contribute */
00911     pos[0] = rad*(ref->xpts[i]) + apos[0];
00912     pos[1] = rad*(ref->ypts[i]) + apos[1];
00913     pos[2] = rad*(ref->zpts[i]) + apos[2];
00914     if (ivdwAccExclus(thee, pos, prad, atomID)) {
00915         npts++;
00916         bpts[i] = 1;
00917     } else {
00918         bpts[i] = 0;
00919     }
00920 }
00921
00922 /* Allocate space for the points */
00923 surf = VaccSurf_ctor(thee->mem, prad, npts);
00924
00925 /* Assign the points */
00926 j = 0;
00927 for (i=0; i<ref->npts; i++) {
00928     if (bpts[i]) {
00929         surf->bpts[j] = 1;
00930         surf->xpts[j] = rad*(ref->xpts[i]) + apos[0];
00931         surf->ypts[j] = rad*(ref->ypts[i]) + apos[1];
00932         surf->zpts[j] = rad*(ref->zpts[i]) + apos[2];
00933         j++;
00934     }
00935 }
00936
00937 /* Assign the area */
00938 surf->area = 4.0*VPI*rad*rad*((double)(surf->npts))/((double)(ref->
npts));
00939
00940     return surf;
00941
00942 }
00943
00944 VPUBLIC VaccSurf* VaccSurf_refSphere(Vmem *mem, int
npts) {
00945
00946     VaccSurf *surf;
00947     int nactual, i, itheta, ntheta, iphi, nphimax, nphi;
00948     double frac;
00949     double sintheta, costheta, theta, dtheta;
00950     double sinphi, cosphi, phi, dphi;
00951
00952     /* Setup "constants" */
00953     frac = ((double)(npts))/4.0;
00954     ntheta = VRINT(VSQRT(Vunit_pi*frac));
00955     dtheta = Vunit_pi/((double)(ntheta));
00956     nphimax = 2*ntheta;
00957
00958     /* Count the actual number of points to be used */
00959     nactual = 0;
00960     for (itheta=0; itheta<ntheta; itheta++) {
00961         theta = dtheta*((double)(itheta));
00962         sintheta = VSIN(theta);
00963         costheta = VCOS(theta);
00964         nphi = VRINT(sintheta*nphimax);
00965         nactual += nphi;
00966     }
00967
00968     /* Allocate space for the points */
00969     surf = VaccSurf_ctor(mem, 1.0, nactual);
00970
00971     /* Clear out the boolean array */
00972     for (i=0; i<nactual; i++) surf->bpts[i] = 1;
00973
00974     /* Assign the points */
00975     nactual = 0;
00976     for (itheta=0; itheta<ntheta; itheta++) {
00977         theta = dtheta*((double)(itheta));
00978         sintheta = VSIN(theta);
00979         costheta = VCOS(theta);
00980         nphi = VRINT(sintheta*nphimax);
00981         if (nphi != 0) {
00982             dphi = 2*Vunit_pi/((double)(nphi));
00983             for (iphi=0; iphi<nphi; iphi++) {
00984                 phi = dphi*((double)(iphi));
00985                 sinphi = VSIN(phi);
00986                 cosphi = VCOS(phi);
00987                 surf->xpts[nactual] = cosphi * sintheta;
00988                 surf->ypts[nactual] = sinphi * sintheta;

```

```

00989             surf->zpts[nactual] = costheta;
00990             nactual++;
00991         }
00992     }
00993 }
00994
00995 surf->npts = nactual;
00996
00997 return surf;
00998 }
00999
01000 VPUBLIC VaccSurf* Vacc_atomSASPoints(Vacc *thee,
01001     double radius,
01002     Vatom *atom) {
01003
01004     VaccSurf *asurf = VNULL;
01005     int id;
01006
01007     if (thee->surf == VNULL) Vacc_SASA(thee, radius);
01008     id = Vatom_getAtomID(atom);
01009
01010     asurf = thee->surf[id];
01011
01012     /* See if this surface needs to be rebuilt */
01013     if (asurf->probe_radius != radius) {
01014         Vnm_print(2, "Vacc_SASA: Warning -- probe radius changed from %g to
01015 %g!\n",
01016                 asurf->probe_radius, radius);
01017         VaccSurf_dtor2(asurf);
01018         thee->surf[id] = Vacc_atomSurf(thee, atom, thee->
01019                                         refSphere, radius);
01020         asurf = thee->surf[id];
01021     }
01022
01023     return asurf;
01024 }
01025
01026 VPUBLIC void Vacc_splineAccGradAtomNorm4(Vacc *
01027     thee, double center[VAPBS_DIM],
01028     double win, double infrad, Vatom *atom, double *grad) {
01029
01030     int i;
01031     double dist, *apos, arad, sm, sm2, sm3, sm4, sm5, sm6, sm7,
01032     e, e2, e3, e4, e5, e6, e7;
01033     double b, b2, b3, b4, b5, b6, b7;
01034     double c0, c1, c2, c3, c4, c5, c6, c7;
01035     double denom, mygrad;
01036     double mychi = 1.0;           /* Char. func. value for given atom */
01037
01038     VASSERT(thee != NULL);
01039
01040     /* The grad is zero by default */
01041     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
01042
01043     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
01044    * *** MAGNITUDE OF THE FORCE *** */
01045     apos = VatomGetPosition(atom);
01046     /* Zero-radius atoms don't contribute */
01047     if (Vatom_getRadius(atom) > 0.0) {
01048
01049         arad = Vatom_getRadius(atom);
01050         arad = arad + infrad;
01051         b = arad - win;
01052         e = arad + win;
01053
01054         e2 = e * e;
01055         e3 = e2 * e;
01056         e4 = e3 * e;
01057         e5 = e4 * e;
01058         e6 = e5 * e;
01059         e7 = e6 * e;
01060         b2 = b * b;
01061         b3 = b2 * b;
01062         b4 = b3 * b;
01063         b5 = b4 * b;
01064         b6 = b5 * b;
01065         b7 = b6 * b;
01066
01067         denom = e7 - 7.0*b*e6 + 21.0*b2*e5 - 35.0*e4*b3
01068         + 35.0*e3*b4 - 21.0*b5*e2 + 7.0*e*b6 - b7;

```

```

01066      c0 = b4*(35.0*e3 - 21.0*b*e2 + 7*e*b2 - b3)/denom;
01067      c1 = -140.0*b3*e3/denom;
01068      c2 = 210.0*e2*b2*(e + b)/denom;
01069      c3 = -140.0*e*b*(e2 + 3.0*b*e + b2)/denom;
01070      c4 = 35.0*(e3 + 9.0*b*e2 + 9.0*e*b2 + b3)/denom;
01071      c5 = -84.0*(e2 + 3.0*b*e + b2)/denom;
01072      c6 = 70.0*(e + b)/denom;
01073      c7 = -20.0/denom;
01074
01075      dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
01076      + VSQR(apos[2]-center[2]));
01077
01078      /* If we're inside an atom, the entire characteristic function
01079      * will be zero and the grad will be zero, so we can stop */
01080      if (dist < (arad - win)) return;
01081      /* Likewise, if we're outside the smoothing window, the characteristic
01082      * function is unity and the grad will be zero, so we can stop */
01083      else if (dist > (arad + win)) return;
01084      /* Account for floating point error at the border
01085      * NAB: COULDNT THESE TESTS BE COMBINED AS BELOW
01086      * (Vacc_splineAccAtom) ? */
01087      else if ((VABS(dist - (arad - win)) < VSMALL) ||
01088      (VABS(dist - (arad + win)) < VSMALL)) return;
01089      /* If we're inside the smoothing window */
01090      else {
01091          sm = dist;
01092          sm2 = sm * sm;
01093          sm3 = sm2 * sm;
01094          sm4 = sm3 * sm;
01095          sm5 = sm4 * sm;
01096          sm6 = sm5 * sm;
01097          sm7 = sm6 * sm;
01098          mychi = c0 + c1*sm + c2*sm2 + c3*sm3
01099          + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
01100          mygrad = c1 + 2.0*c2*sm + 3.0*c3*sm2 + 4.0*c4*sm3
01101          + 5.0*c5*sm4 + 6.0*c6*sm5 + 7.0*c7*sm6;
01102          if (mychi <= 0.0) {
01103          /* Avoid numerical round off errors */
01104          return;
01105          } else if (mychi > 1.0) {
01106          /* Avoid numerical round off errors */
01107          mychi = 1.0;
01108          }
01109          }
01110          /* Now assemble the grad vector */
01111          VASSERT(mychi > 0.0);
01112          for (i=0; i<VAPBS_DIM; i++)
01113              grad[i] = -(mygrad/mychi)*((center[i] - apos[i])/dist);
01114      }
01115  }
01116
01117 VPUBLIC void Vacc_splineAccGradAtomNorm3(Vacc *
01118     thee, double center[VAPBS_DIM],
01119     double win, double infrad, Vatom *atom, double *grad) {
01120
01121     int i;
01122     double dist, *apos, arad, sm, sm2, sm3, sm4, sm5;
01123     double e, e2, e3, e4, e5;
01124     double b, b2, b3, b4, b5;
01125     double c0, c1, c2, c3, c4, c5;
01126     double denom, mygrad;
01127     double mychi = 1.0;           /* Char. func. value for given atom */
01128
01129     VASSERT(thee != NULL);
01130
01131     /* The grad is zero by default */
01132     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
01133
01134     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
01135     * *** MAGNITUDE OF THE FORCE *** */
01136     apos = Vatom_getPosition(atom);
01137     /* Zero-radius atoms don't contribute */
01138     if (Vatom_getRadius(atom) > 0.0) {
01139
01140         arad = Vatom_getRadius(atom);
01141         arad = arad + infrad;
01142         b = arad - win;
01143         e = arad + win;
01144
01145         e2 = e * e;
01146         e3 = e2 * e;

```

```

01146     e4 = e3 * e;
01147     e5 = e4 * e;
01148     b2 = b * b;
01149     b3 = b2 * b;
01150     b4 = b3 * b;
01151     b5 = b4 * b;
01152
01153     denom = pow((e - b), 5.0);
01154     c0 = -10.0*e2*b3 + 5.0*e*b4 - b5;
01155     c1 = 30.0*e2*b2;
01156     c2 = -30.0*(e2*b + e*b2);
01157     c3 = 10.0*(e2 + 4.0*e*b + b2);
01158     c4 = -15.0*(e + b);
01159     c5 = 6;
01160     c0 = c0/denom;
01161     c1 = c1/denom;
01162     c2 = c2/denom;
01163     c3 = c3/denom;
01164     c4 = c4/denom;
01165     c5 = c5/denom;
01166
01167     dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
01168     + VSQR(apos[2]-center[2]));
01169
01170     /* If we're inside an atom, the entire characteristic function
01171 * will be zero and the grad will be zero, so we can stop */
01172     if (dist < (arad - win)) return;
01173     /* Likewise, if we're outside the smoothing window, the characteristic
01174 * function is unity and the grad will be zero, so we can stop */
01175     else if (dist > (arad + win)) return;
01176     /* Account for floating point error at the border
01177 * NAB: COULDNT THESE TESTS BE COMBINED AS BELOW
01178 * (Vacc_splineAccAtom)? */
01179     else if ((VABS(dist - (arad - win)) < VSMALL) ||
01180               (VABS(dist - (arad + win)) < VSMALL)) return;
01181     /* If we're inside the smoothing window */
01182     else {
01183         sm = dist;
01184         sm2 = sm * sm;
01185         sm3 = sm2 * sm;
01186         sm4 = sm3 * sm;
01187         sm5 = sm4 * sm;
01188         mychi = c0 + c1*sm + c2*sm2 + c3*sm3
01189         + c4*sm4 + c5*sm5;
01190         mygrad = c1 + 2.0*c2*sm + 3.0*c3*sm2 + 4.0*c4*sm3
01191         + 5.0*c5*sm4;
01192         if (mychi <= 0.0) {
01193             /* Avoid numerical round off errors */
01194             return;
01195         } else if (mychi > 1.0) {
01196             /* Avoid numerical round off errors */
01197             mychi = 1.0;
01198         }
01199     }
01200     /* Now assemble the grad vector */
01201     VASSERT(mychi > 0.0);
01202     for (i=0; i<VAPBS_DIM; i++)
01203         grad[i] = -(mygrad/mychi)*((center[i] - aps[i])/dist);
01204     }
01205 }
01206
01207 /* /////////////////////////////////
01208 // Routine: Vacc_atomdSAV
01209 //
01210 // Purpose: Calculates the vector valued atomic derivative of volume
01211 //
01212 // Args:      radius   The radius of the solvent probe in Angstroms
01213 //           iatom    Index of the atom in thee->alist
01214 //
01215 // Author:    Jason Wagoner
01216 //           Nathan Baker (original FORTRAN routine from UHBD by Brock Luty)
01217 VPUBLIC void Vacc_atomdSAV(Vacc *thee,
01218                             double srad,
01219                             Vatom *atom,
01220                             double *dSA
01221                             ) {
01222
01223     int ipt, iatom;
01224
01225     double area;
01226     double *tPos, tRad, vec[3];

```

```

01228     double dx,dy,dz;
01229     VAccSurf *ref;
01230     dx = 0.0;
01231     dy = 0.0;
01232     dz = 0.0;
01233     /* Get the atom information */
01234     ref = thee->refSphere;
01235     iatom = Vatom_getAtomID(atom);
01236
01237     dSA[0] = 0.0;
01238     dSA[1] = 0.0;
01239     dSA[2] = 0.0;
01240
01241     tPos = VatomGetPosition(atom);
01242     tRad = Vatom_getRadius(atom);
01243
01244 if(tRad == 0.0) return;
01245
01246 area = 4.0*VPI*(tRad+srad)*(tRad+srad)/((double)(ref->npts));
01247 for (ipt=0; ipt<ref->npts; ipt++) {
01248     vec[0] = (tRad+srad)*ref->xpts[ipt] + tPos[0];
01249     vec[1] = (tRad+srad)*ref->ypts[ipt] + tPos[1];
01250     vec[2] = (tRad+srad)*ref->zpts[ipt] + tPos[2];
01251     if (!vdwAccExclus(thee, vec, srad, iatom)) {
01252         dx = dx+vec[0]-tPos[0];
01253         dy = dy+vec[1]-tPos[1];
01254         dz = dz+vec[2]-tPos[2];
01255     }
01256 }
01257
01258 if ((tRad+srad) != 0){
01259     dSA[0] = dx*area/(tRad+srad);
01260     dSA[1] = dy*area/(tRad+srad);
01261     dSA[2] = dz*area/(tRad+srad);
01262 }
01263
01264 }
01265
01266 /* Note: This is purely test code to make certain that the dSASA code is
01267 behaving properly. This function should NEVER be called by anyone
01268 other than an APBS developer at Wash U.
01269 */
01270 VPRIVATE double Vacc_SASAPos(Vacc *thee, double radius) {
01271
01272     int i, natom;
01273     double area;
01274     Vatom *atom;
01275     VAccSurf *asurf;
01276
01277     natom = Valist_getNumberAtoms(thee->alist);
01278
01279     /* Calculate the area */
01280     area = 0.0;
01281     for (i=0; i<natom; i++) {
01282         atom = Valist_getAtom(thee->alist, i);
01283         asurf = thee->surf[i];
01284
01285         VaccSurf_dtOr2(asurf);
01286         thee->surf[i] = Vacc_atomSurf(thee, atom, thee->
01287             refSphere, radius);
01288         asurf = thee->surf[i];
01289         area += (asurf->area);
01290     }
01291
01292     return area;
01293 }
01294
01295 VPRIVATE double Vacc_atomSASAPos(Vacc *thee,
01296                                     double radius,
01297                                     Vatom *atom, /* The atom being
01298 manipulated */
01299                                     int mode
01300                                     ) {
01301
01302     VAccSurf *asurf;
01303     int id;
01304     static int warned = 0;
01305
01306     if ((thee->surf == VNULL) || (mode == 1)){
01307         if (!warned){

```

```

01307     printf("WARNING: Recalculating entire surface!!!!\n");
01308     warned = 1;
01309 }
01310 Vacc_SASAPos(thee, radius); // reinitialize before we can do anything about
01311 // doing a calculation on a repositioned atom
01312 }
01313 id = Vatom_getAtomID(atom);
01314 asurf = thee->surf[id];
01315
01316 VaccSurf_dtor(&asurf);
01317 thee->surf[id] = Vacc_atomSurf(thee, atom, thee->refSphere
01318 , radius);
01319 asurf = thee->surf[id];
01320 //printf("%s: Time elapsed: %f\n", __func__, ((double)clock() - ts) /
01321 CLOCKs_PER_SEC);
01322
01323 return asurf->area;
01324 }
01325 /* /////////////////////////////////
01326 // Routine: Vacc_atomdSASA
01327 //
01328 // Purpose: Calculates the derivative of surface area with respect to
01329 atomic
01330 // displacement using finite difference methods.
01331 //
01332 // Args: radius The radius of the solvent probe in Angstroms
01333 // iatom Index of the atom in thee->alist
01334 //
01335 // Author: Jason Wagoner
01336 // David Gohara
01337 // Nathan Baker (original FORTRAN routine from UHBD by Brock Luty)
01338 VPUBLIC void Vacc_atomdSASA(Vacc *thee,
01339                               double dpos,
01340                               double srad,
01341                               Vatom *atom,
01342                               double *dSA
01343 ) {
01344
01345     double *temp_Pos,
01346             tPos[3],
01347             axbl,
01348             axtl,
01349             aybl,
01350             ayt1,
01351             azbl,
01352             azt1;
01353     VaccSurf *ref;
01354
01355     //printf("%s: entering\n", __func__);
01356     time_t ts;
01357     ts = clock();
01358
01359     /* Get the atom information */
01360     ref = thee->refSphere;
01361     temp_Pos = Vatom_getPosition(atom); // Get a pointer to
01362     // the position object. You actually manipulate the atom doing this...
01363     tPos[0] = temp_Pos[0];
01364     tPos[1] = temp_Pos[1];
01365     tPos[2] = temp_Pos[2];
01366
01367     /* Shift by pos -/+ on x */
01368     temp_Pos[0] -= dpos;
01369     axbl = Vacc_atomSASAPos(thee, srad, atom, 0);
01370     temp_Pos[0] = tPos[0];
01371
01372     temp_Pos[0] += dpos;
01373     axtl = Vacc_atomSASAPos(thee, srad, atom, 0);
01374     temp_Pos[0] = tPos[0];
01375
01376     /* Shift by pos -/+ on y */
01377     temp_Pos[1] -= dpos;
01378     aybl = Vacc_atomSASAPos(thee, srad, atom, 0);
01379     temp_Pos[1] = tPos[1];
01380
01381     temp_Pos[1] += dpos;
01382     ayt1 = Vacc_atomSASAPos(thee, srad, atom, 0);
01383     temp_Pos[1] = tPos[1];

```

```

01384 /* Shift by pos -/+ on z */
01385     temp_Pos[2] -= dpos;
01386     azbl = Vacc_atomSASAPos(thee, srad, atom, 0);
01387     temp_Pos[2] = tPos[2];
01388
01389     temp_Pos[2] += dpos;
01390     aztl = Vacc_atomSASAPos(thee, srad, atom, 0);
01391     temp_Pos[2] = tPos[2];
01392
01393 /* Reset the atom SASA to zero displacement */
01394 Vacc_atomSASAPos(thee, srad, atom, 0);
01395
01396 /* Calculate the final value */
01397     dSA[0] = (axt1-axb1)/(2.0 * dpos);
01398     dSA[1] = (ayt1/ayb1)/(2.0 * dpos);
01399     dSA[2] = (azt1-azb1)/(2.0 * dpos);
01400 }
01401
01402 /* Note: This is purely test code to make certain that the dSASA code is
01403    behaving properly. This function should NEVER be called by anyone
01404    other than an APBS developer at Wash U.
01405 */
01406
01407 VPUBLIC void Vacc_totalAtomdSASA(Vacc *thee, double dpos
01408 , double srad, Vatom *atom, double *dSA) {
01409     int iatom;
01410     double *temp_Pos, tRad;
01411     double tPos[3];
01412     double axb1, axt1, ayb1, ayt1, azb1, aztl;
01413     VaccSurf *ref;
01414
01415     /* Get the atom information */
01416     ref = thee->refSphere;
01417     temp_Pos = Vatom_getPosition(atom);
01418     tRad = Vatom_getRadius(atom);
01419     iatom = Vatom_getAtomID(atom);
01420
01421     dSA[0] = 0.0;
01422     dSA[1] = 0.0;
01423     dSA[2] = 0.0;
01424
01425     tPos[0] = temp_Pos[0];
01426     tPos[1] = temp_Pos[1];
01427     tPos[2] = temp_Pos[2];
01428
01429     /* Shift by pos -/+ on x */
01430     temp_Pos[0] -= dpos;
01431     axb1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01432     temp_Pos[0] = tPos[0];
01433
01434     temp_Pos[0] += dpos;
01435     axt1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01436     temp_Pos[0] = tPos[0];
01437
01438     /* Shift by pos -/+ on y */
01439     temp_Pos[1] -= dpos;
01440     ayb1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01441     temp_Pos[1] = tPos[1];
01442
01443     temp_Pos[1] += dpos;
01444     ayt1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01445     temp_Pos[1] = tPos[1];
01446
01447     /* Shift by pos -/+ on z */
01448     temp_Pos[2] -= dpos;
01449     azb1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01450     temp_Pos[2] = tPos[2];
01451
01452     temp_Pos[2] += dpos;
01453     aztl = Vacc_atomSASAPos(thee, srad, atom, 1);
01454     temp_Pos[2] = tPos[2];
01455
01456     /* Calculate the final value */
01457     dSA[0] = (axt1-axb1)/(2.0 * dpos);
01458     dSA[1] = (ayt1-ayb1)/(2.0 * dpos);
01459     dSA[2] = (azt1-azb1)/(2.0 * dpos);
01460 }
01461
01462 /* Note: This is purely test code to make certain that the dSASA code is
01463    behaving properly. This function should NEVER be called by anyone

```

```

01464     other than an APBS developer at Wash U.
01465 */
01466 VPUBLIC void Vacc_totalAtomdSAV(Vacc *thee, double dpos,
01467     double srad, Vatom *atom, double *dSA, Vclist *clist) {
01468     int iatom;
01469     double *temp_Pos, tRad;
01470     double tPos[3];
01471     double axb1, axt1, ayb1, ayt1, azb1, azt1;
01472     VaccSurf *ref;
01473
01474     /* Get the atom information */
01475     ref = thee->refSphere;
01476     temp_Pos = Vatom_getPosition(atom);
01477     tRad = Vatom_getRadius(atom);
01478     iatom = Vatom_getAtomID(atom);
01479
01480     dSA[0] = 0.0;
01481     dSA[1] = 0.0;
01482     dSA[2] = 0.0;
01483
01484     tPos[0] = temp_Pos[0];
01485     tPos[1] = temp_Pos[1];
01486     tPos[2] = temp_Pos[2];
01487
01488     /* Shift by pos -/+ on x */
01489     temp_Pos[0] -= dpos;
01490     axb1 = Vacc_totalSAV(thee,clist, VNULL, srad);
01491     temp_Pos[0] = tPos[0];
01492
01493     temp_Pos[0] += dpos;
01494     axt1 = Vacc_totalSAV(thee,clist, VNULL, srad);
01495     temp_Pos[0] = tPos[0];
01496
01497     /* Shift by pos -/+ on y */
01498     temp_Pos[1] -= dpos;
01499     ayb1 = Vacc_totalSAV(thee,clist, VNULL, srad);
01500     temp_Pos[1] = tPos[1];
01501
01502     temp_Pos[1] += dpos;
01503     ayt1 = Vacc_totalSAV(thee,clist, VNULL, srad);
01504     temp_Pos[1] = tPos[1];
01505
01506     /* Shift by pos -/+ on z */
01507     temp_Pos[2] -= dpos;
01508     azb1 = Vacc_totalSAV(thee,clist, VNULL, srad);
01509     temp_Pos[2] = tPos[2];
01510
01511     temp_Pos[2] += dpos;
01512     azt1 = Vacc_totalSAV(thee,clist, VNULL, srad);
01513     temp_Pos[2] = tPos[2];
01514
01515     /* Calculate the final value */
01516     dSA[0] = (axt1-axb1)/(2.0 * dpos);
01517     dSA[1] = (ayt1-ayb1)/(2.0 * dpos);
01518     dSA[2] = (azt1-azb1)/(2.0 * dpos);
01519 }
01520
01521 VPUBLIC double Vacc_totalSAV(Vacc *thee, Vclist *clist
01522 , APOLparm *apolparm, double radius) {
01523     int i;
01524     int npts[3];
01525
01526     double spacs[3], vec[3];
01527     double w, wx, wy, wz, len, fn, x, y, z, vol;
01528     double vol_density,sav;
01529     double *lower_corner, *upper_corner;
01530
01531     sav = 0.0;
01532     vol = 1.0;
01533     vol_density = 2.0;
01534
01535     lower_corner = clist->lower_corner;
01536     upper_corner = clist->upper_corner;
01537
01538     for (i=0; i<3; i++) {
01539         len = upper_corner[i] - lower_corner[i];
01540         vol *= len;
01541         fn = len*vol_density + 1;
01542         npts[i] = (int)ceil(fn);

```

```

01543     spacs[i] = len/((double)(npts[i])-1.0);
01544     if (apolparm != VNULL) {
01545         if (apolparm->setgrid) {
01546             if (apolparm->grid[i] > spacs[i]) {
01547                 Vnm_print(2, "Vacc_totalSAV: Warning, your GRID value (%g) is larger than
the recommended value (%g)!\n",
01548                         apolparm->grid[i], spacs[i]);
01549             }
01550             spacs[i] = apolparm->grid[i];
01551         }
01552     }
01553 }
01554 }
01555
01556 for (x=lower_corner[0]; x<=upper_corner[0]; x=x+spacs[0]) {
01557     if ( VABS(x - lower_corner[0]) < VSMALL) {
01558         wx = 0.5;
01559     } else if ( VABS(x - upper_corner[0]) < VSMALL) {
01560         wx = 0.5;
01561     } else {
01562         wx = 1.0;
01563     }
01564     vec[0] = x;
01565     for (y=lower_corner[1]; y<=upper_corner[1]; y=y+spacs[1]) {
01566         if ( VABS(y - lower_corner[1]) < VSMALL) {
01567             wy = 0.5;
01568         } else if ( VABS(y - upper_corner[1]) < VSMALL) {
01569             wy = 0.5;
01570         } else {
01571             wy = 1.0;
01572         }
01573     vec[1] = y;
01574     for (z=lower_corner[2]; z<=upper_corner[2]; z=z+spacs[2]) {
01575         if ( VABS(z - lower_corner[2]) < VSMALL) {
01576             wz = 0.5;
01577         } else if ( VABS(z - upper_corner[2]) < VSMALL) {
01578             wz = 0.5;
01579         } else {
01580             wz = 1.0;
01581         }
01582     vec[2] = z;
01583
01584     w = wx*wy*wz;
01585
01586     sav += (w*(1.0-Vacc_ivdwAcc(thee, vec, radius)));
01587
01588     } /* z loop */
01589 } /* y loop */
01590 } /* x loop */
01591
01592 w = spacs[0]*spacs[1]*spacs[2];
01593 sav *= w;
01594
01595 return sav;
01596 }
01597
01598 int Vacc_wcaEnergyAtom(Vacc *thee, APOLparm *
01599 apolparm, Valist *alist,
01600             Vclist *clist, int iatom, double *value) {
01600
01601     int i;
01602     int npts[3];
01603     int pad = 14;
01604
01605         int xmin, ymin, zmin;
01606         int xmax, ymax, zmax;
01607
01608         double sigma6, sigma12;
01609
01610     double spacs[3], vec[3];
01611     double w, wx, wy, wz, len, fn, x, y, z, vol;
01612     double x2,y2,z2,r;
01613     double vol_density, energy, rho, srad;
01614     double psig, epsilon, watepsilon, sigma, watsigma, eni, chi;
01615
01616     double *pos;
01617     double *lower_corner, *upper_corner;
01618
01619     Vatom *atom = VNULL;
01620     VASSERT(apolparm != VNULL);
01621

```

```

01622 energy = 0.0;
01623 vol = 1.0;
01624 vol_density = 2.0;
01625
01626 lower_corner = clist->lower_corner;
01627 upper_corner = clist->upper_corner;
01628
01629 atom = Valist_getAtom(alist, iatom);
01630 pos = Vatom_getPosition(atom);
01631
01632 /* Note: these are the original temporary water parameters... they have been
01633 replaced by entries in a parameter file:
01634 watsigma = 1.7683;
01635 watepsilon = 0.1521;
01636 watepsilon = watepsilon*4.184;
01637 */
01638
01639 srad = apolparm->srad;
01640 rho = apolparm->bconc;
01641 watsigma = apolparm->watsigma;
01642 watepsilon = apolparm->watepsilon;
01643 psig = atom->radius;
01644 epsilon = atom->epsilon;
01645 sigma = psig + watsigma;
01646 epsilon = VSQRT((epsilon * watepsilon));
01647
01648 /* parameters */
01649 sigma6 = VPOW(sigma,6);
01650 sigma12 = VPOW(sigma,12);
01651 /* OPLS-style radius: double sigmar = sigma*VPOW(2, (1.0/6.0)); */
01652
01653 xmin = pos[0] - pad;
01654 xmax = pos[0] + pad;
01655 ymin = pos[1] - pad;
01656 ymax = pos[1] + pad;
01657 zmin = pos[2] - pad;
01658 zmax = pos[2] + pad;
01659
01660 for (i=0; i<3; i++) {
01661 len = (upper_corner[i] + pad) - (lower_corner[i] - pad);
01662 vol *= len;
01663 fn = len*vol_density + 1;
01664 npts[i] = (int)ceil(fn);
01665 spacs[i] = 0.5;
01666 if (apolparm->setgrid) {
01667 if (apolparm->grid[i] > spacs[i]) {
01668 Vnm_print(2, "Vacc_totalsAV: Warning, your GRID value (%g) is larger than
the recommended value (%g)!\n",
01669 apolparm->grid[i], spacs[i]);
01670 }
01671 spacs[i] = apolparm->grid[i];
01672 }
01673 }
01674
01675 for (x=xmin; x<=xmax; x=x+spacs[0]) {
01676 if ( VABS(x - xmin) < VSMALL) {
01677 wx = 0.5;
01678 } else if ( VABS(x - xmax) < VSMALL) {
01679 wx = 0.5;
01680 } else {
01681 wx = 1.0;
01682 }
01683 vec[0] = x;
01684 for (y=ymin; y<=ymax; y=y+spacs[1]) {
01685 if ( VABS(y - ymin) < VSMALL) {
01686 wy = 0.5;
01687 } else if ( VABS(y - ymax) < VSMALL) {
01688 wy = 0.5;
01689 } else {
01690 wy = 1.0;
01691 }
01692 vec[1] = y;
01693 for (z=zmin; z<=zmax; z=z+spacs[2]) {
01694 if ( VABS(z - zmin) < VSMALL) {
01695 wz = 0.5;
01696 } else if ( VABS(z - zmax) < VSMALL) {
01697 wz = 0.5;
01698 } else {
01699 wz = 1.0;
01700 }
01701 vec[2] = z;

```

```

01702
01703     w = wx*wy*wz;
01704
01705     chi = Vacc_ivdwAcc(thee, vec, srad);
01706
01707     if (VABS(chi) > VSMALL) {
01708
01709         x2 = VSQR(vec[0]-pos[0]);
01710         y2 = VSQR(vec[1]-pos[1]);
01711         z2 = VSQR(vec[2]-pos[2]);
01712         r = VSQRT(x2+y2+z2);
01713
01714         if (r <= 14 && r >= sigma) {
01715             eni = chi*rho*epsilon*(-2.0*sigma6/VPOW(r,6)+sigma12/VPOW(r,12));
01716         } else if (r <= 14){
01717             eni = -1.0*epsilon*chi*rho;
01718         }else{
01719             eni = 0.0;
01720         }
01721     }else{
01722         eni = 0.0;
01723     }
01724
01725     energy += eni*w;
01726
01727     } /* z loop */
01728     } /* y loop */
01729 } /* x loop */
01730
01731 w = spacs[0]*spacs[1]*spacs[2];
01732 energy *= w;
01733
01734 *value = energy;
01735
01736 return VRC_SUCCESS;
01737 }
01738
01739 VPUBLIC int Vacc_wcaEnergy(Vacc *acc, APOLparm *
01740     apolparm, Valist *alist,
01741     Vclist *clist){
01742
01743     int iatom;
01744     int rc = 0;
01745
01746     double energy = 0.0;
01747     double tenergy = 0.0;
01748     double rho = apolparm->bconc;
01749
01750     /* Do a sanity check to make sure that watepsilon and watsigma are set
01751      * If not, return with an error. */
01752     if(apolparm->setwat == 0){
01753         Vnm_print(2,"Vacc_wcaEnergy: Error. No value was set for watsigma and
01754         watepsilon.\n");
01755         return VRC_FAILURE;
01756     }
01757
01758     if (VABS(rho) < VSMALL) {
01759         apolparm->wcaEnergy = tenergy;
01760         return 1;
01761     }
01762     for (iatom=0; iatom<Valist_getNumberAtoms(alist);
01763         iatom++){
01764         rc = Vacc_wcaEnergyAtom(acc, apolparm, alist, clist,
01765             iatom, &energy);
01766         if(rc == 0) return 0;
01767
01768         tenergy += energy;
01769     }
01770
01771     apolparm->wcaEnergy = tenergy;
01772
01773     return VRC_SUCCESS;
01774 }
01775
01776 VPUBLIC int Vacc_wcaForceAtom(Vacc *thee,
01777     APOLparm *apolparm,
01778     Vclist *clist,
01779     Vatom *atom,
01780     double *force

```

```

01779
01780     int i,
01781         si,
01782         npts[3],
01783         pad = 14,
01784         xmin,
01785         ymin,
01786         zmin,
01787         xmax,
01788         ymax,
01789         zmax;
01790
01791     double sigma6,
01792         sigma12,
01793         spacs[3],
01794         vec[3],
01795         fpt[3],
01796         w,
01797         wx,
01798         wy,
01799         wz,
01800         len,
01801         fn,
01802         x,
01803         y,
01804         z,
01805         vol,
01806         x2,
01807         y2,
01808         z2,
01809         r,
01810         vol_density,
01811         fo,
01812         rho,
01813         srad,
01814         psig,
01815         epsilon,
01816         watepsilon,
01817         sigma,
01818         watsigma,
01819         chi,
01820         *pos,
01821         *lower_corner,
01822         *upper_corner,
01823
01824     /* Allocate needed variables now that we've asserted required conditions.
01825 */
01826     time_t ts;
01827     ts = clock();
01828
01829     VASSERT(apolparm != VNULL);
01830
01831     /* Do a sanity check to make sure that watepsilon and watsigma are set
01832      * If not, return with an error. */
01833     if(apolparm->setwat == 0){
01834         Vnm_print(2,"Vacc_wcaEnergy: Error. No value was set for watsigma and
01835         watepsilon.\n");
01836         return VRC_FAILURE;
01837     }
01838
01839     vol = 1.0;
01840     vol_density = 2.0;
01841
01842     lower_corner = clist->lower_corner;
01843     upper_corner = clist->upper_corner;
01844
01845     pos = Vatom_getPosition(atom);
01846
01847     srad = apolparm->srad;
01848     rho = apolparm->bconc;
01849     watsigma = apolparm->watsigma;
01850     watepsilon = apolparm->watepsilon;
01851
01852     psig = atom->radius;
01853     epsilon = atom->epsilon;
01854     sigma = psig + watsigma;
01855     epsilon = VSQRT((epsilon * watepsilon));
01856
01857     /* parameters */
01858     sigma6 = VPOW(sigma,6);
01859     sigma12 = VPOW(sigma,12);

```

```

01858     /* OPLS-style radius:  double sigmar = sigma*VPOW(2, (1.0/6.0)); */
01859
01860     for (i=0; i<3; i++) {
01861         len = (upper_corner[i] + pad) - (lower_corner[i] - pad);
01862         vol *= len;
01863         fn = len*vol_density + 1;
01864         npts[i] = (int)ceil(fn);
01865         spacs[i] = 0.5;
01866         force[i] = 0.0;
01867         if (apolparm->setgrid) {
01868             if (apolparm->grid[i] > spacs[i]) {
01869                 Vnm_pprint(2, "Vacc_totalSAV: Warning, your GRID value (%g) is larger than
the recommended value (%g)!\n",
01870                 apolparm->grid[i], spacs[i]);
01871             }
01872             spacs[i] = apolparm->grid[i];
01873         }
01874     }
01875
01876     xmin = pos[0] - pad;
01877     xmax = pos[0] + pad;
01878     ymin = pos[1] - pad;
01879     ymax = pos[1] + pad;
01880     zmin = pos[2] - pad;
01881     zmax = pos[2] + pad;
01882
01883     for (x=xmin; x<=xmax; x=x+spacs[0]) {
01884         if ( VABS(x - xmin) < VSMALL) {
01885             wx = 0.5;
01886         } else if ( VABS(x - xmax) < VSMALL) {
01887             wx = 0.5;
01888         } else {
01889             wx = 1.0;
01890         }
01891         vec[0] = x;
01892         for (y=ymin; y<=ymax; y=y+spacs[1]) {
01893             if ( VABS(y - ymin) < VSMALL) {
01894                 wy = 0.5;
01895             } else if ( VABS(y - ymax) < VSMALL) {
01896                 wy = 0.5;
01897             } else {
01898                 wy = 1.0;
01899             }
01900             vec[1] = y;
01901             for (z=zmin; z<=zmax; z=z+spacs[2]) {
01902                 if ( VABS(z - zmin) < VSMALL) {
01903                     wz = 0.5;
01904                 } else if ( VABS(z - zmax) < VSMALL) {
01905                     wz = 0.5;
01906                 } else {
01907                     wz = 1.0;
01908                 }
01909                 vec[2] = z;
01910
01911                 w = wx*wy*wz;
01912
01913                 chi = Vacc_ivdwAcc(thee, vec, srad);
01914
01915                 if (chi != 0.0) {
01916
01917                     x2 = VSQR(vec[0]-pos[0]);
01918                     y2 = VSQR(vec[1]-pos[1]);
01919                     z2 = VSQR(vec[2]-pos[2]);
01920                     r = VSQRT(x2+y2+z2);
01921
01922                     if (r <= 14 && r >= sigma){
01923
01924                         fo = 12.0*chi*rho*epsilon*(sigma6/VPOW(r,7)-sigma12/VPOW(r,13));
01925
01926                         fpt[0] = -1.0*(pos[0]-vec[0])*fo/r;
01927                         fpt[1] = -1.0*(pos[1]-vec[1])*fo/r;
01928                         fpt[2] = -1.0*(pos[2]-vec[2])*fo/r;
01929
01930                     }else {
01931                         for (si=0; si < 3; si++) fpt[si] = 0.0;
01932                     }
01933                 }else {
01934                     for (si=0; si < 3; si++) fpt[si] = 0.0;
01935                 }
01936
01937             for(i=0;i<3;i++) {

```

```

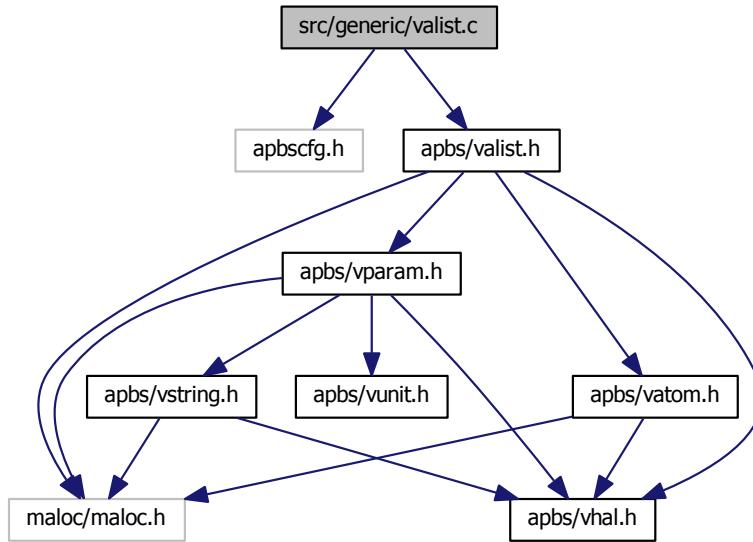
01938     force[i] += (w*fpt[i]);
01939 }
01940
01941 } /* z loop */
01942 } /* y loop */
01943 } /* x loop */
01944
01945 w = spacs[0]*spacs[1]*spacs[2];
01946 for(i=0;i<3;i++) force[i] *= w;
01947
01948 return VRC_SUCCESS;
01949 }
01950

```

9.65 src/generic/valist.c File Reference

Class Valist methods.

```
#include "apbscfg.h"
#include "apbs/valist.h"
Include dependency graph for valist.c:
```



Functions

- VPUBLIC double [Valist_getCenterX](#) ([Valist](#) *thee)
Get x-coordinate of molecule center.
- VPUBLIC double [Valist_getCenterY](#) ([Valist](#) *thee)
Get y-coordinate of molecule center.
- VPUBLIC double [Valist_getCenterZ](#) ([Valist](#) *thee)
Get z-coordinate of molecule center.
- VPUBLIC [Vatom](#) * [Valist_getAtomList](#) ([Valist](#) *thee)

- **VPUBLIC int Valist_getNumberAtoms (Valist *thee)**

Get actual array of atom objects from the list.
- **VPUBLIC Vatom * Valist_getAtom (Valist *thee, int i)**

Get number of atoms in the list.
- **VPUBLIC unsigned long int Valist_memChk (Valist *thee)**

Get pointer to particular atom in list.
- **VPUBLIC Valist * Valist_ctor ()**

Get total memory allocated for this object and its members.
- **VPUBLIC Vrc_Codes Valist_ctor2 (Valist *thee)**

Construct the atom list object.
- **VPUBLIC void Valist_dtor (Valist **thee)**

FORTRAN stub to construct the atom list object.
- **VPUBLIC void Valist_dtor2 (Valist *thee)**

Destroys atom list object.
- **VPUBLIC void Valist_dtor3 (Valist *thee)**

FORTRAN stub to destroy atom list object.
- **VPRIVATE Vrc_Codes Valist_readPDBSerial (Valist *thee, Vio *sock, int *serial)**
- **VPRIVATE Vrc_Codes Valist_readPDBAtomName (Valist *thee, Vio *sock, char atomName[VMAX_ARGLEN])**
- **VPRIVATE Vrc_Codes Valist_readPDBResidueName (Valist *thee, Vio *sock, char resName[VMAX_ARGLEN])**
- **VPRIVATE Vrc_Codes Valist_readPDBResidueNumber (Valist *thee, Vio *sock, int *resSeq)**
- **VPRIVATE Vrc_Codes Valist_readPDBAtomCoord (Valist *thee, Vio *sock, double *coord)**
- **VPRIVATE Vrc_Codes Valist_readPDBChargeRadius (Valist *thee, Vio *sock, double *charge, double *radius)**
- **VPRIVATE Vrc_Codes Valist_readPDB_throughXYZ (Valist *thee, Vio *sock, int *serial, char atomName[VMAX_ARGLEN], char resName[VMAX_ARGLEN], int *resSeq, double *x, double *y, double *z)**
- **VPRIVATE Vatom * Valist_getAtomStorage (Valist *thee, Vatom **plist, int *pnlist, int *pnatoms)**
- **VPRIVATE Vrc_Codes Valist_setAtomArray (Valist *thee, Vatom **plist, int nlist, int natoms)**
- **VPUBLIC Vrc_Codes Valist_readPDB (Valist *thee, Vparam *param, Vio *sock)**

Fill atom list with information from a PDB file.
- **VPUBLIC Vrc_Codes Valist_readPQR (Valist *thee, Vparam *params, Vio *sock)**

Fill atom list with information from a PQR file.
- **VPUBLIC Vrc_Codes Valist_readXML (Valist *thee, Vparam *params, Vio *sock)**

Fill atom list with information from an XML file.
- **VPUBLIC Vrc_Codes Valist_getStatistics (Valist *thee)**

Load up Valist with various statistics.

Variables

- **VPRIVATE char * Valist_whiteChars = " \t\r\n"**
- **VPRIVATE char * Valist_commChars = "#%"**
- **VPRIVATE char * Valist_xmlwhiteChars = " \t\r\n<>"**

9.65.1 Detailed Description

Class Valist methods.

Author

Nathan Baker

Version

Id:

[valist.c](#) 1750 2012-07-18 18:34:27Z tuckerbeck

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](#).

9.66 valist.c

```
00001  
00056 #include "apbscfg.h"  
00057 #include "apbs/valist.h"
```

```
00058
00059 VEMBED(rcsid="$Id: valist.c 1750 2012-07-18 18:34:27Z tuckerbeck $")
00060
00061 VPRIvATE char *Valist_whiteChars = " \t\r\n";
00062 VPRIvATE char *Valist_commChars = "#%";
00063 VPRIvATE char *Valist_xmlwhiteChars = " \t\r\n<>";
00064
00065 #if !defined(VINLINE_VATOM)
00066
00067 VPUBLIC double Valist_getCenterX(Valist *thee) {
00068
00069     if (thee == NULL) {
00070         Vnm_print(2, "Valist_getCenterX: Found null pointer when getting the center
00071         of X coordinate!\n");
00072         VASSErT(0);
00073     }
00074     return thee->center[0];
00075 }
00076
00077 VPUBLIC double Valist_getCenterY(Valist *thee) {
00078
00079     if (thee == NULL) {
00080         Vnm_print(2, "Valist_getCenterY: Found null pointer when getting the center
00081         of Y coordinate!\n");
00082         VASSErT(0);
00083     }
00084     return thee->center[1];
00085 }
00086 VPUBLIC double Valist_getCenterZ(Valist *thee) {
00087
00088     if (thee == NULL) {
00089         Vnm_print(2, "Valist_getCenterZ: Found null pointer when getting the center
00090         of Z coordinate!\n");
00091         VASSErT(0);
00092     }
00093     return thee->center[2];
00094 }
00095
00096 VPUBLIC Vatom* Valist_getAtomList(Valist *thee) {
00097
00098     if (thee == NULL) {
00099         Vnm_print(2, "Valist_getAtomList: Found null pointer when getting the atom
00100         list!\n");
00101         VASSErT(0);
00102     }
00103     return thee->atoms;
00104 }
00105
00106 VPUBLIC int Valist_getNumberAtoms(Valist *thee) {
00107
00108     if (thee == NULL) {
00109         Vnm_print(2, "Valist_getNumberAtoms: Found null pointer when getting the
00110         number of atoms!\n");
00111         VASSErT(0);
00112     }
00113     return thee->number;
00114 }
00115
00116 VPUBLIC Vatom* Valist_getAtom(Valist *thee, int i) {
00117
00118     if (thee == NULL) {
00119         Vnm_print(2, "Valist_getAtom: Found null pointer when getting atoms!\n");
00120         VASSErT(0);
00121     }
00122     if (i >= thee->number) {
00123         Vnm_print(2, "Valist_getAtom: Requested atom number (%d) outside of atom
00124         list range (%d)!\n", i, thee->number);
00125         VASSErT(0);
00126     }
00127     return &(thee->atoms[i]);
00128 }
00129
00130 VPUBLIC unsigned long int Valist_memChk(Valist *thee) {
00131
00132     if (thee == NULL) return 0;
```

```

00133     return Vmem_bytes(thee->vmem);
00134
00135 }
00136
00137 #endif /* if !defined(VINLINE_VATOM) */
00138
00139 VPUBLIC Valist* Valist_ctor() {
00140
00141     /* Set up the structure */
00142     Valist *thee = VNULL;
00143     thee = (Valist*)Vmem_malloc(VNULL, 1, sizeof(Valist));
00144     if (thee == VNULL) {
00145         Vnm_print(2, "Valist_ctor: Got NULL pointer when constructing the atom list
object!\n");
00146         VASSERT(0);
00147     }
00148     if (Valist_ctor2(thee) != VRC_SUCCESS) {
00149         Vnm_print(2, "Valist_ctor: Error in constructing the atom list object!\n");
00150         VASSERT(0);
00151     }
00152
00153     return thee;
00154 }
00155
00156 VPUBLIC Vrc_Codes Valist_ctor2(Valist *thee) {
00157
00158     thee->atoms = VNULL;
00159     thee->number = 0;
00160
00161     /* Initialize the memory management object */
00162     thee->vmem = Vmem_ctor("APBS:VALIST");
00163
00164     return VRC_SUCCESS;
00165
00166 }
00167
00168 VPUBLIC void Valist_dtor(Valist **thee)
00169 {
00170     if ((*thee) != VNULL) {
00171         Valist_dtor2(*thee);
00172         Vmem_free(VNULL, 1, sizeof(Valist), (void **)thee);
00173         (*thee) = VNULL;
00174     }
00175 }
00176
00177 VPUBLIC void Valist_dtor2(Valist *thee) {
00178
00179     Vmem_free(thee->vmem, thee->number, sizeof(Vatom), (void **)
&(thee->atoms));
00180     thee->atoms = VNULL;
00181     thee->number = 0;
00182
00183     Vmem_dtor(&(thee->vmem));
00184 }
00185
00186 /* Read serial number from PDB ATOM/HETATM field */
00187 VPRIVATE Vrc_Codes Valist_readPDBSerial(Valist *thee, Vio *sock, int *
serial) {
00188
00189     char tok[VMAX_BUFSIZE];
00190     int ti = 0;
00191
00192     if (Vio_scanf(sock, "%s", tok) != 1) {
00193         Vnm_print(2, "Valist_readPDB: Ran out of tokens while parsing serial!
\n");
00194         return VRC_FAILURE;
00195     }
00196     if (sscanf(tok, "%d", &ti) != 1) {
00197         Vnm_print(2, "Valist_readPDB: Unable to parse serial token (%s) as
int!\n",
00198                 tok);
00199         return VRC_FAILURE;
00200     }
00201     *serial = ti;
00202
00203     return VRC_SUCCESS;
00204 }
00205
00206 /* Read atom name from PDB ATOM/HETATM field */
00207 VPRIVATE Vrc_Codes Valist_readPDBAtomName(Valist *thee, Vio *sock,
00208                                         char atomName[VMAX_ARGLEN]) {

```

```

00209
00210     char tok[VMAX_BUFSIZE];
00211
00212     if (Vio_scanf(sock, "%s", tok) != 1) {
00213         Vnm_print(2, "Valist_readPDB: Ran out of tokens while parsing atom
name!\n");
00214         return VRC_FAILURE;
00215     }
00216     if (strlen(tok) < VMAX_ARGLEN) strcpy(atomName, tok);
00217     else {
00218         Vnm_print(2, "Valist_readPDB: Atom name (%s) too long!\n", tok);
00219         return VRC_FAILURE;
00220     }
00221     return VRC_SUCCESS;
00222 }
00223
00224 /* Read residue name from PDB ATOM/HETATM field */
00225 VPRIPRIVATE Vrc_Codes Valist_readPDBResidueName(Valist *thee, Vio *sock,
00226         char resName[VMAX_ARGLEN]) {
00227
00228     char tok[VMAX_BUFSIZE];
00229
00230     if (Vio_scanf(sock, "%s", tok) != 1) {
00231         Vnm_print(2, "Valist_readPDB: Ran out of tokens while parsing residue
name!\n");
00232         return VRC_FAILURE;
00233     }
00234     if (strlen(tok) < VMAX_ARGLEN) strcpy(resName, tok);
00235     else {
00236         Vnm_print(2, "Valist_readPDB: Residue name (%s) too long!\n", tok);
00237         return VRC_FAILURE;
00238     }
00239     return VRC_SUCCESS;
00240 }
00241
00242 /* Read residue number from PDB ATOM/HETATM field */
00243 VPRIPRIVATE Vrc_Codes Valist_readPDBResidueNumber(
00244     Valist *thee, Vio *sock, int *resSeq) {
00245
00246     char tok[VMAX_BUFSIZE];
00247     char *resstring;
00248     int ti = 0;
00249
00250     if (Vio_scanf(sock, "%s", tok) != 1) {
00251         Vnm_print(2, "Valist_readPDB: Ran out of tokens while parsing resSeq!
\n");
00252         return VRC_FAILURE;
00253     }
00254     if (sscanf(tok, "%d", &ti) != 1) {
00255
00256         /* One of three things can happen here:
00257         1) There is a chainID in the line:    THR A    1
00258         2) The chainID is merged with resSeq: THR A1001
00259         3) An actual error:                  THR foo
00260
00261     */
00262
00263     if (strlen(tok) == 1) {
00264         /* Case 1: Chain ID Present
00265             Read the next field and hope its a float */
00266
00267         if (Vio_scanf(sock, "%s", tok) != 1) {
00268             Vnm_print(2, "Valist_readPDB: Ran out of tokens while parsing
resSeq!\n");
00269             return VRC_FAILURE;
00270         }
00271         if (sscanf(tok, "%d", &ti) != 1) {
00272             Vnm_print(2, "Valist_readPDB: Unable to parse resSeq token (%s) as int!\n"
00273
00274                     tok);
00275             return VRC_FAILURE;
00276         }
00277     } else {
00278         /* Case 2: Chain ID, merged string.
00279             Move pointer forward past the chainID and check
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
00288 (%s) as int!\n",
00289             resstring);
00290     return VRC_FAILURE;
00291   }
00292 }
00293 *resSeq = ti;
00294
00295 return VRC_SUCCESS;
00296 }
00297
00298 /* Read atom coordinate from PDB ATOM/HETATOM field */
00299 VPRIVATE Vrc_Codes Valist_readPDBAtomCoord(Valist *thee, Vio *sock,
00300     double *coord) {
00301     char tok[VMAX_BUFSIZE];
00302     double tf = 0;
00303
00304     if (Vio_scanf(sock, "%s", tok) != 1) {
00305         Vnm_print(2, "Valist_readPDB: Ran out of tokens while parsing atom
00306 coordinate!\n");
00307         return VRC_FAILURE;
00308     }
00309     if (sscanf(tok, "%lf", &tf) != 1) {
00310         return VRC_FAILURE;
00311     }
00312     *coord = tf;
00313
00314     return VRC_SUCCESS;
00315 }
00316 /* Read charge and radius from PQR ATOM/HETATOM field */
00317 VPRIVATE Vrc_Codes Valist_readPDBChargeRadius(Valist *thee, Vio *sock,
00318     double *charge, double *radius) {
00319
00320     char tok[VMAX_BUFSIZE];
00321     double tf = 0;
00322
00323     if (Vio_scanf(sock, "%s", tok) != 1) {
00324         Vnm_print(2, "Valist_readPQR: Ran out of tokens while parsing charge!
00325 \n");
00326         return VRC_FAILURE;
00327     }
00328     if (sscanf(tok, "%lf", &tf) != 1) {
00329         return VRC_FAILURE;
00330     }
00331     *charge = tf;
00332
00333     if (Vio_scanf(sock, "%s", tok) != 1) {
00334         Vnm_print(2, "Valist_readPQR: Ran out of tokens while parsing radius!
00335 \n");
00336         return VRC_FAILURE;
00337     }
00338     if (sscanf(tok, "%lf", &tf) != 1) {
00339         return VRC_FAILURE;
00340     }
00341     *radius = tf;
00342
00343     return VRC_SUCCESS;
00344 }
00345 /* Read ATOM/HETATOM field of PDB through the X/Y/Z fields */
00346 VPRIVATE Vrc_Codes Valist_readPDB_throughXYZ(
00347     Valist *thee,
00348     Vio *sock, /* Socket ready for reading */
00349     int *serial, /* Set to atom number */
00350     char atomName[VMAX_ARGLEN], /* Set to atom name */
00351     char resName[VMAX_ARGLEN], /* Set to residue name */
00352     int *resSeq, /* Set to residue number */
00353     double *x, /* Set to x-coordinate */
00354     double *y, /* Set to y-coordinate */
00355     double *z /* Set to z-coordinate */
00356 )
00357
00358     int i, njunk, gotit;
00359
00360     /* Grab serial */

```

```

00361     if (Valist_readPDBSerial(thee, sock, serial) == VRC_FAILURE) {
00362         Vnm_print(2, "Valist_readPDB: Error while parsing serial!\n");
00363     }
00364
00365     /* Grab atom name */
00366     if (Valist_readPDBAtomName(thee, sock, atomName) == VRC_FAILURE)
00367     {
00368         Vnm_print(2, "Valist_readPDB: Error while parsing atom name!\n");
00369         return VRC_FAILURE;
00370     }
00371
00372     /* Grab residue name */
00373     if (Valist_readPDBResidueName(thee, sock, resName) == VRC_FAILURE)
00374     {
00375         Vnm_print(2, "Valist_readPDB: Error while parsing residue name!\n");
00376         return VRC_FAILURE;
00377     }
00378
00379     /* Grab residue number */
00380     if (Valist_readPDBResidueNumber(thee, sock, resSeq) == VRC_FAILURE)
00381     {
00382         Vnm_print(2, "Valist_readPDB: Error while parsing residue name!\n");
00383         return VRC_FAILURE;
00384     }
00385
00386     /* Read tokens until we find one that can be parsed as an atom
00387      * x-coordinate. We will allow njunk=1 intervening field that
00388      * cannot be parsed as a coordinate */
00389     njunk = 1;
00390     gotit = 0;
00391     for (i=0; i<(njunk+1); i++) {
00392         if (Valist_readPDBAtomCoord(thee, sock, x) == VRC_SUCCESS) {
00393             gotit = 1;
00394             break;
00395         }
00396     if (!gotit) {
00397         Vnm_print(2, "Valist_readPDB: Can't find x!\n");
00398         return VRC_FAILURE;
00399     }
00400
00401     /* Read y-coordinate */
00402     if (Valist_readPDBAtomCoord(thee, sock, y) == VRC_FAILURE) {
00403         Vnm_print(2, "Valist_readPDB: Can't find y!\n");
00404         return VRC_FAILURE;
00405     }
00406     /* Read z-coordinate */
00407     if (Valist_readPDBAtomCoord(thee, sock, z) == VRC_FAILURE) {
00408         Vnm_print(2, "Valist_readPDB: Can't find z!\n");
00409         return VRC_FAILURE;
00410     }
00411 #if 0 /* Set to 1 if you want to debug */
00412     Vnm_print(1, "Valist_readPDB: serial = %d\n", *serial);
00413     Vnm_print(1, "Valist_readPDB: atomName = %s\n", atomName);
00414     Vnm_print(1, "Valist_readPDB: resName = %s\n", resName);
00415     Vnm_print(1, "Valist_readPDB: resSeq = %d\n", *resSeq);
00416     Vnm_print(1, "Valist_readPDB: pos = (%g, %g, %g)\n",
00417               *x, *y, *z);
00418 #endif
00419
00420     return VRC_SUCCESS;
00421 }
00422
00423 /* Get a the next available atom storage location, increasing the storage
00424   * space if necessary. Return VNULL if something goes wrong. */
00425 VPRIVATE Vatom* Valist_getAtomStorage(
00426     Valist *thee,
00427     Vatom **plist, /* Pointer to existing list of atoms */
00428     int *pnlist, /* Size of existing list, may be changed */
00429     int *pnatoms /* Existing number of atoms in list; incremented
00430                  before exit */
00431 )
00432 {
00433     Vatom *oldList, *newList, *theList;
00434     Vatom *oldAtom, *newAtom;
00435     int iatom, inext, oldLength, newLength, natoms;
00436
00437     newList = VNULL;
00438

```

```

00439 /* See if we need more space */
00440 if (*pnatoms >= *pnlist) {
00441
00442     /* Double the storage space */
00443     oldLength = *pnlist;
00444     newLength = 2*oldLength;
00445     newList = (Vatom*)Vmem_malloc(thee->vmem, newLength, sizeof(
00446         Vatom));
00447     oldList = *plist;
00448
00449     /* Check the allocation */
00450     if (newList == VNNULL) {
00451         Vnm_print(2, "Valist_readPDB: failed to allocate space for %d
00452             (%Vatom)s!\n", newLength);
00453         return VNNULL;
00454     }
00455
00456     /* Copy the atoms over */
00457     natoms = *pnatoms;
00458     for (iatom=0; iatom<natoms; iatom++) {
00459         oldAtom = &(oldList[iatom]);
00460         newAtom = &(newList[iatom]);
00461         Vatom_copyTo(oldAtom, newAtom);
00462         Vatom_dtor2(oldAtom);
00463     }
00464
00465     /* Free the old list */
00466     Vmem_free(thee->vmem, oldLength, sizeof(Vatom), (void **)plist
00467     );
00468
00469     /* Copy new list to plist */
00470     *plist = newList;
00471     *pnlist = newLength;
00472 }
00473
00474 theList = *plist;
00475 inext = *pnatoms;
00476
00477 /* Get the next available spot and increment counters */
00478 newAtom = &(theList[inext]);
00479 *pnatoms = inext + 1;
00480
00481     return newAtom;
00482 }
00483
00484 VPRIVATE Vrc_Codes Valist_setAtomArray(Valist *thee,
00485     Vatom **plist, /* Pointer to list of atoms to store */
00486     int nlist, /* Length of list */
00487     int natoms /* Number of real atom entries in list */
00488 ) {
00489
00490     Vatom *list, *newAtom, *oldAtom;
00491     int i;
00492
00493     list = *plist;
00494
00495     /* Allocate necessary space */
00496     thee->number = 0;
00497     thee->atoms = (Vatom*)Vmem_malloc(thee->vmem, natoms, sizeof(
00498         Vatom));
00499     if (thee->atoms == VNNULL) {
00500         Vnm_print(2, "Valist_readPDB: Unable to allocate space for %d
00501             (%Vatom)s!\n",
00502             natoms);
00503         return VRC_FAILURE;
00504     }
00505     thee->number = natoms;
00506
00507     /* Copy over data */
00508     for (i=0; i<thee->number; i++) {
00509         newAtom = &(thee->atoms[i]);
00510         oldAtom = &(list[i]);
00511         Vatom_copyTo(oldAtom, newAtom);
00512         Vatom_dtor2(oldAtom);
00513     }
00514
00515     /* Free old array */
00516     Vmem_free(thee->vmem, nlist, sizeof(Vatom), (void **)plist);
00517
00518     return VRC_SUCCESS;
00519 }
```

```

00515
00516 VPUBLIC Vrc_Codes Valist_readPDB(Valist *thee, Vparam
00517     *param, Vio *sock) {
00518     /* WE DO NOT DIRECTLY CONFORM TO PDB STANDARDS -- TO ALLOW LARGER FILES, WE
00519      * REQUIRE ALL FIELDS TO BE WHITESPACE DELIMITED */
00520
00521     Vatom *atoms = VNULL;
00522     Vatom *nextAtom = VNULL;
00523     Vparam_AtomData *atomData = VNULL;
00524
00525     char tok[VMAX_BUFSIZE];
00526     char atomName[VMAX_ARGLEN], resName[VMAX_ARGLEN];
00527
00528     int nlist, natoms, serial, resSeq;
00529
00530     double x, y, z, charge, radius, epsilon;
00531     double pos[3];
00532
00533     if (thee == VNULL) {
00534         Vnm_print(2, "Valist_readPDB: Got NULL pointer when reading PDB file!\n");
00535         VASSERT(0);
00536     }
00537     thee->number = 0;
00538
00539     Vio_setWhiteChars(sock, Valist_whiteChars);
00540     Vio_setCommChars(sock, Valist_commChars);
00541
00542     /* Allocate some initial space for the atoms */
00543     nlist = 200;
00544     atoms = (Vatom*)Vmem_malloc(thee->vmem, nlist, sizeof(Vatom));
00545 ;
00546     natoms = 0;
00547     /* Read until we run out of lines */
00548     while (Vio_scanf(sock, "%s", tok) == 1) {
00549
00550         /* Parse only ATOM/HETATOM fields */
00551         if ((Vstring_strcasecmp(tok, "ATOM") == 0) ||
00552             (Vstring_strcasecmp(tok, "HETATOM") == 0)) {
00553
00554             /* Read ATOM/HETATOM field of PDB through the X/Y/Z fields */
00555             if (Valist_readPDB_throughXYZ(thee, sock, &serial, atomName,
00556                 resName, &resSeq, &x, &y, &z) == VRC_FAILURE)
00557             {
00558                 Vnm_print(2, "Valist_readPDB: Error parsing atom %d!\n",
00559                         serial);
00560                 return VRC_FAILURE;
00561             }
00562
00563             /* Try to find the parameters. */
00564             atomData = Vparam_getAtomData(param, resName,
00565                 atomName);
00566             if (atomData == VNULL) {
00567                 Vnm_print(2, "Valist_readPDB: Couldn't find parameters for \
00568 atom = %s, residue = %s\n", atomName, resName);
00569                 return VRC_FAILURE;
00570             }
00571             charge = atomData->charge;
00572             radius = atomData->radius;
00573             epsilon = atomData->epsilon;
00574
00575             /* Get pointer to next available atom position */
00576             nextAtom = Valist_getAtomStorage(thee, &atoms, &nlist, &natoms);
00577             if (nextAtom == VNULL) {
00578                 Vnm_print(2, "Valist_readPDB: Error in allocating spacing for
atoms!\n");
00579                 return VRC_FAILURE;
00580             }
00581
00582             /* Store the information */
00583             pos[0] = x; pos[1] = y; pos[2] = z;
00584             Vatom_setPosition(nextAtom, pos);
00585             Vatom_setCharge(nextAtom, charge);
00586             Vatom_setRadius(nextAtom, radius);
00587             Vatom_setEpsilon(nextAtom, epsilon);
00588             Vatom_setAtomID(nextAtom, natoms-1);
00589             Vatom_setResName(nextAtom, resName);
00590             Vatom_setAtomName(nextAtom, atomName);
00591
00592         } /* if ATOM or HETATOM */

```

```

00591     } /* while we haven't run out of tokens */
00592
00593     Vnm_print(0, "Valist_readPDB: Counted %d atoms\n", natoms);
00594     fflush(stdout);
00595
00596     /* Store atoms internally */
00597     if (Valist_setAtomArray(thee, &atoms, nlist, natoms) == VRC_FAILURE
00598     ) {
00599         Vnm_print(2, "Valist_readPDB: unable to store atoms!\n");
00600         return VRC_FAILURE;
00601     }
00602     return Valist_getStatistics(thee);
00603
00604
00605 }
00606
00607 VPUBLIC Vrc_Codes Valist_readPQR(Valist *thee, Vparam
00608 *params, Vio *sock) {
00609
00610     /* WE DO NOT DIRECTLY CONFORM TO PDB STANDARDS -- TO ALLOW LARGER FILES, WE
00611     * REQUIRE ALL FIELDS TO BE WHITESPACE DELIMITED */
00612
00613     Vatom *atoms = VNULL;
00614     Vatom *nextAtom = VNULL;
00615     Vparam_AtomData *atomData = VNULL;
00616
00617     char tok[VMAX_BUFSIZE];
00618     char atomName[VMAX_ARGLEN], resName[VMAX_ARGLEN];
00619
00620     int use_params = 0;
00621     int nlist, natoms, serial, resSeq;
00622
00623     double x, y, z, charge, radius, epsilon;
00624     double pos[3];
00625
00626     epsilon = 0.0;
00627
00628     if (thee == VNULL) {
00629         Vnm_print(2, "Valist_readPQR: Got NULL pointer when reading PQR file!\n");
00630         VASSERT(0);
00631     }
00632     thee->number = 0;
00633
00634     Vio_setWhiteChars(sock, Valist_whiteChars);
00635     Vio_setCommChars(sock, Valist_commChars);
00636
00637     /* Allocate some initial space for the atoms */
00638     nlist = 200;
00639     atoms = (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/HETATOM field of PDB through the X/Y/Z fields */
00653             if (Valist_readPDB_throughXYZ(thee, sock, &serial, atomName,
00654                 resName, &resSeq, &x, &y, &z) == VRC_FAILURE
00655             ) {
00656                 Vnm_print(2, "Valist_readPQR: Error parsing atom %d!\n", serial
00657             );
00658                 Vnm_print(2, "Please double check this atom in the pqr file,
00659                 e.g., make sure there are no concatenated fields.\n");
00660                 return VRC_FAILURE;
00661             }
00662             /* Read Q/R fields */
00663             if (Valist_readPDBChargeRadius(thee, sock, &charge, &radius) ==
00664 VRC_FAILURE) {
00665                 Vnm_print(2, "Valist_readPQR: Error parsing atom %d!\n",
00666                         serial);
00667                 Vnm_print(2, "Please double check this atom in the pqr file,

```

```

e.g., make sure there are no concatenated fields.\n");
00665         return VRC_FAILURE;
00666     }
00667
00668     if(use_params){
00669         /* Try to find the parameters. */
0070 atomData = Vparam_getAtomData(params, resName, atomName);
0071     if (atomData == VNULL) {
0072         Vnm_print(2, "Valist_readPDB: Couldn't find parameters for \
0073 atom = %s, residue = %s\n", atomName, resName);
0074         return VRC_FAILURE;
0075     }
0076     charge = atomData->charge;
0077     radius = atomData->radius;
0078     epsilon = atomData->epsilon;
0079 }
0080
0081     /* Get pointer to next available atom position */
0082 nextAtom = Valist_getAtomStorage(thee, &atoms, &nlist, &natoms);
0083     if (nextAtom == VNULL) {
0084         Vnm_print(2, "Valist_readPQR: Error in allocating spacing for
atoms!\n");
0085         return VRC_FAILURE;
0086     }
0087
0088     /* Store the information */
0089 pos[0] = x; pos[1] = y; pos[2] = z;
0090 Vatom_setPosition(nextAtom, pos);
0091 Vatom_setCharge(nextAtom, charge);
0092 Vatom_setRadius(nextAtom, radius);
0093 Vatom_setEpsilon(nextAtom, epsilon);
0094 Vatom_setAtomID(nextAtom, natoms-1);
0095 Vatom_setResName(nextAtom, resName);
0096     Vatom_setAtomName(nextAtom, atomName);
0097
0098 } /* if ATOM or HETATM */
0099 } /* while we haven't run out of tokens */
0100
0101 Vnm_print(0, "Valist_readPQR: Counted %d atoms\n", natoms);
0102 fflush(stdout);
0103
0104 /* Store atoms internally */
0105     if (Valist_setAtomArray(thee, &atoms, nlist, natoms) == VRC_FAILURE
0106 ) {
0107         Vnm_print(2, "Valist_readPDB: unable to store atoms!\n");
0108         return VRC_FAILURE;
0109     }
0110
0111     return Valist_getStatistics(thee);
0112
0113 }
0114
0115 VPUBLIC Vrc_Codes Valist_readXML(Valist *thee, Vparam
*params, Vio *sock) {
0116
0117     Vatom *atoms = VNULL;
0118     Vatom *nextAtom = VNULL;
0119
0120     char tok[VMAX_BUFSIZE];
0121     char endtag[VMAX_BUFSIZE];
0122
0123     int nlist, natoms;
0124     int xset, yset, zset, chgset, radset;
0125
0126     double x, y, z, charge, radius, dtmp;
0127     double pos[3];
0128
0129     if (thee == VNULL) {
0130         Vnm_print(2, "Valist_readXML: Got NULL pointer when reading XML file!\n");
0131         VASSERT(0);
0132     }
0133     thee->number = 0;
0134
0135     Vio_setWhiteChars(sock, Valist_xmlwhiteChars);
0136     Vio_setCommChars(sock, Valist_commChars);
0137
0138     /* Allocate some initial space for the atoms */
0139     nlist = 200;
0140     atoms = (Vatom*)Vmem_malloc(thee->vmem, nlist, sizeof(Vatom))
0141 ;

```

```

00741     /* Initialize some variables */
00742     natoms = 0;
00743     xset = 0;
00744     yset = 0;
00745     zset = 0;
00746     chgset = 0;
00747     radset = 0;
00748     strcpy(endtag, "/");
00749
00750     if(params == VNULL) {
00751         Vnm_print(1, "\nValist_readXML: Warning Warning Warning Warning Warning\n");
00752         Vnm_print(1, "Valist_readXML: The use of XML input files with parameter\n");
00753         Vnm_print(1, "Valist_readXML: files is currently not supported.\n");
00754         Vnm_print(1, "Valist_readXML: Warning Warning Warning\n\n");
00755     }
00756 }
00757
00758     /* Read until we run out of lines */
00759     while (Vio_scanf(sock, "%s", tok) == 1) {
00760
00761         /* The first tag taken is the start tag - save it to detect end */
00762         if (Vstring_strcasecmp(endtag, "/") == 0) strcat(
00763             endtag, tok);
00764
00765         if (Vstring_strcasecmp(tok, "x") == 0) {
00766             Vio_scanf(sock, "%s", tok);
00767             if (sscanf(tok, "%lf", &dtmp) != 1) {
00768                 Vnm_print(2, "Valist_readXML: Unexpected token (%s) while \
00769 reading x!\n", tok);
00770                 return VRC_FAILURE;
00771             }
00772             x = dtmp;
00773             xset = 1;
00774         } else if (Vstring_strcasecmp(tok, "y") == 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 y!\n", tok);
00779                 return VRC_FAILURE;
00780             }
00781             y = dtmp;
00782             yset = 1;
00783         } else if (Vstring_strcasecmp(tok, "z") == 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 z!\n", tok);
00788                 return VRC_FAILURE;
00789             }
00790             z = dtmp;
00791             zset = 1;
00792         } else if (Vstring_strcasecmp(tok, "charge") == 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 charge!\n", tok);
00797                 return VRC_FAILURE;
00798             }
00799             charge = dtmp;
00800             chgset = 1;
00801         } else if (Vstring_strcasecmp(tok, "radius") == 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 radius!\n", tok);
00806                 return VRC_FAILURE;
00807             }
00808             radius = dtmp;
00809             radset = 1;
00810         } else if (Vstring_strcasecmp(tok, "/atom") == 0) {
00811
00812             /* Get pointer to next available atom position */
00813             nextAtom = Valist_getAtomStorage(thee, &atoms, &nlist, &natoms);
00814             if (nextAtom == VNULL) {
00815                 Vnm_print(2, "Valist_readXML: Error in allocating spacing for
00816 atoms!\n");
00817                 return VRC_FAILURE;
00818             }
00819
00820             if (xset && yset && zset && chgset && radset) {

```

```

00820             /* Store the information */
00821             pos[0] = x; pos[1] = y; pos[2] = z;
00822             Vatom_setPosition(nextAtom, pos);
00823             Vatom_setCharge(nextAtom, charge);
00824             Vatom_setRadius(nextAtom, radius);
00825             Vatom_setAtomID(nextAtom, natoms-1);
00826
00827             /* Reset the necessary flags */
00828             xset = 0;
00829             yset = 0;
00830             zset = 0;
00831             chgset = 0;
00832             radset = 0;
00833         } else {
00834             Vnm_print(2, "Valist_readXML: Missing field(s) in atom tag:\n");
00835         };
00836         if (!xset) Vnm_print(2, "\tx value not set!\n");
00837         if (!yset) Vnm_print(2, "\ty value not set!\n");
00838         if (!zset) Vnm_print(2, "\tz value not set!\n");
00839         if (!chgset) Vnm_print(2, "\tcharge value not set!\n");
00840         if (!radset) Vnm_print(2, "\tradius value not set!\n");
00841     }
00842     } else if (Vstring_strcasecmp(tok, endtag) == 0)
00843     break;
00844
00845     Vnm_print(0, "Valist_readXML: Counted %d atoms\n", natoms);
00846     fflush(stdout);
00847
00848     /* Store atoms internally */
00849     if (Valist_setAtomArray(thee, &atoms, nlist, natoms) == VRC_FAILURE
00850     ) {
00851         Vnm_print(2, "Valist_readXML: unable to store atoms!\n");
00852         return VRC_FAILURE;
00853     }
00854     return Valist_getStatistics(thee);
00855
00856 }
00857
00858 /* Load up Valist with various statistics */
00859 VPUBLIC Vrc_Codes Valist_getStatistics(Valist *thee)
00860 {
00861     Vatom *atom;
00862     int i, j;
00863
00864     if (thee == VNULL) {
00865     Vnm_print(2, "Valist_getStatistics: Got NULL pointer when loading up Valist
00866     with various statistics!\n");
00867     VASSERT(0);
00868
00869     thee->center[0] = 0.;
00870     thee->center[1] = 0.;
00871     thee->center[2] = 0.;
00872     thee->maxrad = 0.;
00873     thee->charge = 0.;

00874     if (thee->number == 0) return VRC_FAILURE;
00875
00876     /* Reset stat variables */
00877     atom = &(thee->atoms[0]);
00878     for (i=0; i<3; i++) {
00879         thee->maxcrd[i] = thee->mincrd[i] = atom->position[
00880         i];
00881     }
00882     thee->maxrad = atom->radius;
00883     thee->charge = 0.0;
00884
00885     for (i=0; i<thee->number; i++) {
00886
00887         atom = &(thee->atoms[i]);
00888         for (j=0; j<3; j++) {
00889             if (atom->position[j] < thee->mincrd[j])
00890                 thee->mincrd[j] = atom->position[j];
00891             if (atom->position[j] > thee->maxcrd[j])
00892                 thee->maxcrd[j] = atom->position[j];
00893         }
00894         if (atom->radius > thee->maxrad) thee->maxrad = atom

```

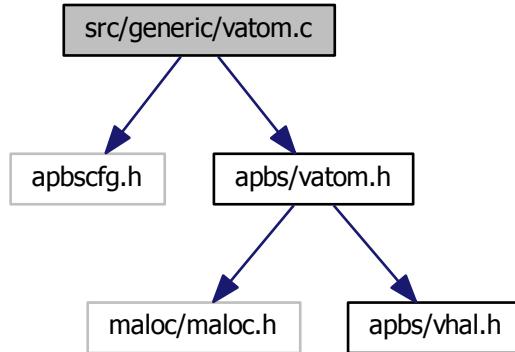
```

->radius;
00895     thee->charge = thee->charge + atom->charge;
00896 }
00897
00898     thee->center[0] = 0.5*(thee->maxcrd[0] + thee->mincrd[0])
00899 ;
00900     thee->center[1] = 0.5*(thee->maxcrd[1] + thee->mincrd[1])
00901 ;
00902     thee->center[2] = 0.5*(thee->maxcrd[2] + thee->mincrd[2])
00903
00904 Vnm_print(0, "Valist_getStatistics: Max atom coordinate: (%g, %g, %g)\n",
00905     thee->maxcrd[0], thee->maxcrd[1], thee->maxcrd[2]);
00906 Vnm_print(0, "Valist_getStatistics: Min atom coordinate: (%g, %g, %g)\n",
00907     thee->mincrd[0], thee->mincrd[1], thee->mincrd[2]);
00908 Vnm_print(0, "Valist_getStatistics: Molecule center: (%g, %g, %g)\n",
00909     thee->center[0], thee->center[1], thee->center[2]);
00910
00911     return VRC_SUCCESS;
00912 }
```

9.67 src/generic/vatom.c File Reference

Class Vatom methods.

```
#include "apbscfg.h"
#include "apbs/vatom.h"
Include dependency graph for vatom.c:
```



Functions

- VPUBLIC double * [VatomGetPosition \(Vatom *thee\)](#)
Get atomic position.
- VPUBLIC double [Vatom_getPartID \(Vatom *thee\)](#)
Get partition ID.
- VPUBLIC void [Vatom_setPartID \(Vatom *thee, int partID\)](#)
Set partition ID.
- VPUBLIC double [Vatom_getAtomID \(Vatom *thee\)](#)

- *Get atom ID.*
 - VPUBLIC void `Vatom_setAtomID` (`Vatom *thee`, int `atomID`)
Set atom ID.
 - VPUBLIC void `Vatom_setRadius` (`Vatom *thee`, double `radius`)
Set atomic radius.
 - VPUBLIC double `Vatom_getRadius` (`Vatom *thee`)
Get atomic position.
 - VPUBLIC void `Vatom_setCharge` (`Vatom *thee`, double `charge`)
Set atomic charge.
 - VPUBLIC double `Vatom_getCharge` (`Vatom *thee`)
Get atomic charge.
 - VPUBLIC unsigned long int `Vatom_memChk` (`Vatom *thee`)
Return the memory used by this structure (and its contents) in bytes.
 - VPUBLIC `Vatom * Vatom_ctor` ()
Constructor for the Vatom class.
 - VPUBLIC int `Vatom_ctor2` (`Vatom *thee`)
FORTRAN stub constructor for the Vatom class.
 - VPUBLIC void `Vatom_dtor` (`Vatom **thee`)
Object destructor.
 - VPUBLIC void `Vatom_dtor2` (`Vatom *thee`)
FORTRAN stub object destructor.
 - VPUBLIC void `Vatom_setPosition` (`Vatom *thee`, double `position[3]`)
Set the atomic position.
 - VPUBLIC void `Vatom_copyTo` (`Vatom *thee`, `Vatom *dest`)
Copy information to another atom.
 - VPUBLIC void `Vatom_copyFrom` (`Vatom *thee`, `Vatom *src`)
Copy information to another atom.
 - VPUBLIC void `Vatom_setResName` (`Vatom *thee`, char `resName[VMAX_RECLEN]`)
Set residue name.
 - VPUBLIC void `Vatom_getResName` (`Vatom *thee`, char `resName[VMAX_RECLEN]`)
Retrieve residue name.
 - VPUBLIC void `Vatom_setAtomName` (`Vatom *thee`, char `atomName[VMAX_RECLEN]`)
Set atom name.
 - VPUBLIC void `Vatom_getAtomName` (`Vatom *thee`, char `atomName[VMAX_RECLEN]`)
Retrieve atom name.

9.67.1 Detailed Description

Class Vatom methods.

Author

Nathan Baker

Version**Id:**[vatom.c](#) 1667 2011-12-02 23:22:02Z pcellis**Attention**

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2012 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](#).

9.68 vatom.c

```
00001  
00057 #include "apbscfg.h"  
00058 #include "apbs/vatom.h"
```

```
00059
00060 VEMBED(rcsid="$Id: vatom.c 1667 2011-12-02 23:22:02Z pcellis $")
00061
00062 #if !defined(VINLINE_VATOM)
00063
00064 VPUBLIC double *Vatom_getPosition(Vatom *thee) {
00065     VASSERT(thee != VNULL);
00066     return thee->position;
00067 }
00068
00069 }
00070
00071 VPUBLIC double Vatom_getPartID(Vatom *thee) {
00072
00073     VASSERT(thee != VNULL);
00074     return thee->partID;
00075
00076 }
00077
00078 VPUBLIC void Vatom_setPartID(Vatom *thee, int partID) {
00079
00080     VASSERT(thee != VNULL);
00081     thee->partID = (double)partID;
00082
00083 }
00084
00085 VPUBLIC double Vatom_getAtomID(Vatom *thee) {
00086
00087     VASSERT(thee != VNULL);
00088     return thee->atomID;
00089
00090 }
00091
00092 VPUBLIC void Vatom_setAtomID(Vatom *thee, int atomID) {
00093
00094     VASSERT(thee != VNULL);
00095     thee->atomID = atomID;
00096
00097 }
00098
00099 VPUBLIC void Vatom_setRadius(Vatom *thee, double radius) {
00100
00101     VASSERT(thee != VNULL);
00102     thee->radius = radius;
00103
00104 }
00105
00106 VPUBLIC double Vatom_getRadius(Vatom *thee) {
00107
00108     VASSERT(thee != VNULL);
00109     return thee->radius;
00110
00111 }
00112
00113 VPUBLIC void Vatom_setCharge(Vatom *thee, double charge) {
00114
00115     VASSERT(thee != VNULL);
00116     thee->charge = charge;
00117
00118 }
00119
00120 VPUBLIC double Vatom_getCharge(Vatom *thee) {
00121
00122     VASSERT(thee != VNULL);
00123     return thee->charge;
00124
00125 }
00126
00127 VPUBLIC unsigned long int Vatom_memChk(Vatom *thee) { return
00128     sizeof(Vatom); }
00129 #endif /* if !defined(VINLINE_VATOM) */
00130
00131 VPUBLIC Vatom* Vatom_ctor() {
00132
00133     /* Set up the structure */
00134     Vatom *thee = VNULL;
00135     thee = (Vatom *)Vmem_malloc( VNULL, 1, sizeof(Vatom) );
00136     VASSERT( thee != VNULL );
00137     VASSERT( Vatom_ctor2(thee));
00138 }
```

```

00139     return thee;
00140 }
00141
00142 VPUBLIC int Vatom_ctor2(Vatom *thee) {
00143     thee->partID = -1;
00144     return 1;
00145 }
00146
00147 VPUBLIC void Vatom_dtor(Vatom **thee) {
00148     if ((*thee) != VNULL) {
00149         Vatom_dtor2(*thee);
00150         Vmem_free(VNULL, 1, sizeof(Vatom), (void **)thee);
00151         (*thee) = VNULL;
00152     }
00153 }
00154
00155 VPUBLIC void Vatom_dtor2(Vatom *thee) { ; }
00156
00157 VPUBLIC void Vatom_setPosition(Vatom *thee, double
00158     position[3]) {
00159     VASSERT(thee != VNULL);
00160     (thee->position)[0] = position[0];
00161     (thee->position)[1] = position[1];
00162     (thee->position)[2] = position[2];
00163 }
00164 }
00165
00166 VPUBLIC void Vatom_copyTo(Vatom *thee, Vatom *dest) {
00167
00168     VASSERT(thee != VNULL);
00169     VASSERT(dest != VNULL);
00170
00171     memcpy(dest, thee, sizeof(Vatom));
00172
00173 }
00174
00175 VPUBLIC void Vatom_copyFrom(Vatom *thee, Vatom *src) {
00176
00177     Vatom_copyTo(src, thee);
00178
00179 }
00180
00181 VPUBLIC void Vatom_setResName(Vatom *thee, char resName[
00182     VMAX_RECLEN]) {
00183
00184     VASSERT(thee != VNULL);
00185     strcpy(thee->resName, resName);
00186 }
00187
00188 VPUBLIC void Vatom_getResName(Vatom *thee, char resName[
00189     VMAX_RECLEN]) {
00190
00191     VASSERT(thee != VNULL);
00192     strcpy(resName, thee->resName);
00193
00194 }
00195
00196 VPUBLIC void Vatom_setAtomName(Vatom *thee, char atomName
00197     [VMAX_RECLEN]) {
00198
00199     VASSERT(thee != VNULL);
00200     strcpy(thee->atomName, atomName);
00201 }
00202
00203 VPUBLIC void Vatom_getAtomName(Vatom *thee, char atomName
00204     [VMAX_RECLEN]) {
00205
00206     VASSERT(thee != VNULL);
00207     strcpy(atomName, thee->atomName);
00208 }
00209
00210 #if defined(WITH_TINKER)
00211
00212 VPUBLIC void Vatom_setDipole(Vatom *thee, double dipole[3]) {
00213
00214     VASSERT(thee != VNULL);

```

```

00215     (thee->dipole)[0] = dipole[0];
00216     (thee->dipole)[1] = dipole[1];
00217     (thee->dipole)[2] = dipole[2];
00218 }
00220
00221 VPUBLIC void Vatom_setQuadrupole(Vatom *thee, double quadrupole[9]) {
00222
00223     int i;
00224     VASSERT(thee != VNULL);
00225     for (i=0; i<9; i++) (thee->quadrupole)[i] = quadrupole[i];
00226 }
00227
00228 VPUBLIC void Vatom_setInducedDipole(Vatom *thee, double dipole[3]) {
00229
00230     VASSERT(thee != VNULL);
00231     (thee->inducedDipole)[0] = dipole[0];
00232     (thee->inducedDipole)[1] = dipole[1];
00233     (thee->inducedDipole)[2] = dipole[2];
00234 }
00235
00236 VPUBLIC void Vatom_setNLInducedDipole(Vatom *thee, double dipole[3]) {
00237
00238     VASSERT(thee != VNULL);
00239     (thee->nLInducedDipole)[0] = dipole[0];
00240     (thee->nLInducedDipole)[1] = dipole[1];
00241     (thee->nLInducedDipole)[2] = dipole[2];
00242 }
00243 }
00244
00245 VPUBLIC double *Vatom_getDipole(Vatom *thee) {
00246
00247     VASSERT(thee != VNULL);
00248     return thee->dipole;
00249
00250 }
00251
00252 VPUBLIC double *Vatom_getQuadrupole(Vatom *thee) {
00253
00254     VASSERT(thee != VNULL);
00255     return thee->quadrupole;
00256
00257 }
00258
00259 VPUBLIC double *Vatom_getInducedDipole(Vatom *thee) {
00260
00261     VASSERT(thee != VNULL);
00262     return thee->inducedDipole;
00263
00264 }
00265
00266 VPUBLIC double *Vatom_getNLInducedDipole(Vatom *thee) {
00267
00268     VASSERT(thee != VNULL);
00269     return thee->nLInducedDipole;
00270
00271 }
00272
00273 #endif /* if defined(WITH_TINKER) */

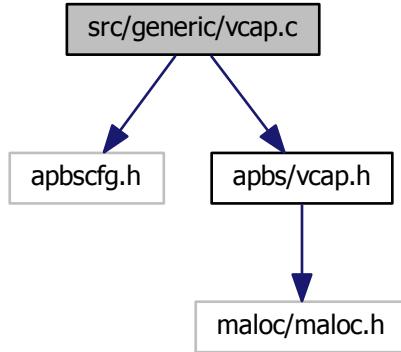
```

9.69 src/generic/vcap.c File Reference

Class Vcap methods.

```
#include "apbscfg.h"
#include "apbs/vcap.h"
```

Include dependency graph for vcap.c:



Functions

- VPUBLIC double [Vcap_exp](#) (double x, int *ichop)
Provide a capped exp() function.
- VPUBLIC double [Vcap_sinh](#) (double x, int *ichop)
Provide a capped sinh() function.
- VPUBLIC double [Vcap_cosh](#) (double x, int *ichop)
Provide a capped cosh() function.

9.69.1 Detailed Description

Class Vcap methods.

Author

Nathan Baker

Version

Id:

[vcap.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 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](#).

9.70 vcap.c

```

00001
00057 #include "apbscfg.h"
00058 #include "apbs/vcap.h"
00059
00060 VPUBLIC double Vcap_exp(double x, int *ichop) {
00061
00062     /* The two chopped arguments */
00063     if (x > EXPMAX) {
00064         (*ichop) = 1;
00065         return VEXP(EXPMAX);
00066     } else if (x < EXPMIN) {
00067         (*ichop) = 1;
00068         return VEXP(EXPMIN);
00069     }

```

```

00070
00071     /* The normal EXP */
00072     (*ichop) = 0;
00073     return VEXP(x);
00074 }
00075
00076 VPUBLIC double Vcap_sinh(double x, int *ichop) {
00077
00078     /* The two chopped arguments */
00079     if (x > EXPMAX) {
00080         (*ichop) = 1;
00081         return VSINH(EXPMAX);
00082     } else if (x < EXPMIN) {
00083         (*ichop) = 1;
00084         return VSINH(EXPMIN);
00085     }
00086
00087     /* The normal SINH */
00088     (*ichop) = 0;
00089     return VSINH(x);
00090 }
00091
00092 VPUBLIC double Vcap_cosh(double x, int *ichop) {
00093
00094     /* The two chopped arguments */
00095     if (x > EXPMAX) {
00096         (*ichop) = 1;
00097         return VCOSH(EXPMAX);
00098     } else if (x < EXPMIN) {
00099         (*ichop) = 1;
00100         return VCOSH(EXPMIN);
00101     }
00102
00103     /* The normal COSH */
00104     (*ichop) = 0;
00105     return VCOSH(x);
00106 }
00107

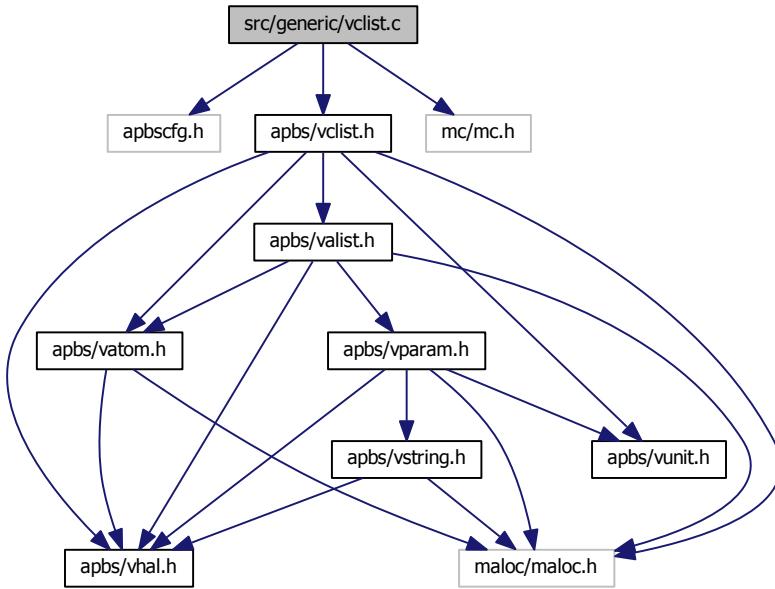
```

9.71 src/generic/vclist.c File Reference

Class Vclist methods.

```
#include "apbscfg.h"
#include "apbs/vclist.h"
#include "mc/mc.h"
```

Include dependency graph for vclist.c:



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 A) the cell list was constructed with.
- VPUBLIC **Vclist** * **Vclist_ctor** (**Valist** *alist, double max_radius, int npts[VAPBS_DIM], **Vclist_DomainMode** mode, double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM])
Construct the cell list object.
- VPRIVATE void **Vclist_getMolDims** (**Vclist** *thee, double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM], double *r_max)
- VPRIVATE Vrc_Codes **Vclist_setupGrid** (**Vclist** *thee)
- VPRIVATE Vrc_Codes **Vclist_storeParms** (**Vclist** *thee, **Valist** *alist, double max_radius, int npts[VAPBS_DIM], **Vclist_DomainMode** mode, double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM])
- VPRIVATE void **Vclist_gridSpan** (**Vclist** *thee, **Vatom** *atom, int imin[VAPBS_DIM], int imax[VAPBS_DIM])
- VPRIVATE int **Vclist_arrayIndex** (**Vclist** *thee, int i, int j, int k)
- VPRIVATE Vrc_Codes **Vclist_assignAtoms** (**Vclist** *thee)
- VPUBLIC Vrc_Codes **Vclist_ctor2** (**Vclist** *thee, **Valist** *alist, double max_radius, int npts[VAPBS_DIM], **Vclist_DomainMode** mode, double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM])
FORTRAN stub to construct the cell list object.

- VPUBLIC void [Vclist_dtor](#) (*Vclist* ***thee*)
Destroy object.
- VPUBLIC void [Vclist_dtor2](#) (*Vclist* **thee*)
FORTRAN stub to destroy object.
- VPUBLIC *VclistCell* * [Vclist_getCell](#) (*Vclist* **thee*, double *pos*[[VAPBS_DIM](#)])
Return cell corresponding to specified position or return VNULL.
- VPUBLIC *VclistCell* * [VclistCell_ctor](#) (int *natoms*)
Allocate and construct a cell list cell object.
- VPUBLIC *Vrc_Codes* [VclistCell_ctor2](#) (*VclistCell* **thee*, int *natoms*)
Construct a cell list object.
- VPUBLIC void [VclistCell_dtor](#) (*VclistCell* ***thee*)
Destroy object.
- VPUBLIC void [VclistCell_dtor2](#) (*VclistCell* **thee*)
FORTRAN stub to destroy object.

9.71.1 Detailed Description

Class *Vclist* methods.

Author

Nathan Baker

Version

Id:

[vclist.c](#) 1750 2012-07-18 18:34:27Z tuckerbeck

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](#).

9.72 vclist.c

```

00001
00057 #include "apbscfg.h"
00058 #include "apbs/vclist.h"
00059
00060 #if defined(HAVE_MC_H)
00061 #include "mc/mc.h"
00062 #endif
00063
00064 VEMBED(rcsid="$Id: vclist.c 1750 2012-07-18 18:34:27Z tuckerbeck $" )
00065
00066 #if !defined(VINLINE_VCLIST)
00067
00068 VPUBLIC unsigned long int Vclist_memChk(Vclist *thee) {
00069     if (thee == VNULL) return 0;
00070     return Vmem_bytes(thee->vmem);
00071 }
00072
00073 VPUBLIC double Vclist_maxRadius(Vclist *thee) {
00074     VASSERT(thee != VNULL);
00075     return thee->max_radius;
00076 }
00077
00078 #endif /* if !defined(VINLINE_VCLIST) */
00079
00080 VPUBLIC Vclist* Vclist_ctor(Valist *alist, double
    max_radius,
00081     int npts[VAPBS_DIM], Vclist_DomainMode mode,
00082     double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM]) {
00083
00084     Vclist *thee = VNULL;
00085
00086     /* Set up the structure */
00087     thee = (Vclist*)Vmem_malloc(VNULL, 1, sizeof(Vclist));
00088     VASSERT( thee != VNULL );
00089     VASSERT( Vclist_ctor2(thee, alist, max_radius, npts, mode,
        lower_corner,
        upper_corner) == VRC_SUCCESS );
00090     return thee;
00091 }
00092
00093 /* Get the dimensions of the molecule stored in thee->alist */
00094 VPRIPRIVATE void Vclist_getMolDims(
00095     Vclist *thee,
00096     double lower_corner[VAPBS_DIM], /* Set to lower corner of
        molecule */

```

```

00098     double upper_corner[VAPBS_DIM], /* Set to lower corner of molecule */
00099     double *r_max /* Set to max atom radius */
00100   ) {
00101
00102   int i, j;
00103   double pos;
00104   Valist *alist;
00105   Vatom *atom;
00106
00107   alist = thee->alist;
00108
00109   /* Initialize */
00110   for (i=0; i<VAPBS_DIM; i++) {
00111     lower_corner[i] = VLARGE;
00112     upper_corner[i] = -VLARGE;
00113   }
00114   *r_max = -1.0;
00115
00116   /* Check each atom */
00117   for (i=0; i<Valist_getNumberAtoms(alist); i++) {
00118     atom = Valist_getAtom(alist, i);
00119     for (j=0; j<VAPBS_DIM; j++) {
00120       pos = (Vatom_getPosition(atom))[j];
00121       if (pos < lower_corner[j]) lower_corner[j] = pos;
00122       if (pos > upper_corner[j]) upper_corner[j] = pos;
00123     }
00124     if (Vatom_getRadius(atom) > *r_max) *r_max =
00125       Vatom_getRadius(atom);
00126   }
00127 }
00128
00129 /* Setup lookup grid */
00130 VPRIPRIVATE Vrc_Codes Vclist_setupGrid(Vclist *thee) {
00131
00132   /* Inflation factor ~ sqrt(2) */
00133   #define VCLIST_INFLATE 1.42
00134
00135   int i;
00136   double length[VAPBS_DIM], r_max;
00137
00138   /* Set up the grid corners */
00139   switch (thee->mode) {
00140     case CLIST_AUTO_DOMAIN:
00141       /* Get molecule dimensions */
00142       Vclist_getMolDims(thee, thee->lower_corner, thee->
00143         upper_corner,
00144         &r_max);
00145       /* Set up grid spacings */
00146       for (i=0; i<VAPBS_DIM; i++) {
00147         thee->upper_corner[i] = thee->upper_corner
00148           [i]
00149             + VCLIST_INFLATE*(r_max+thee->max_radius);
00150         thee->lower_corner[i] = thee->lower_corner
00151           [i]
00152             - VCLIST_INFLATE*(r_max+thee->max_radius);
00153       }
00154       break;
00155     case CLIST_MANUAL_DOMAIN:
00156       /* Grid corners established in constructor */
00157       break;
00158     default:
00159       Vnm_print(2, "Vclist_setupGrid: invalid setup mode (%d)!\n",
00160                 thee->mode);
00161       return VRC_FAILURE;
00162   }
00163
00164   /* Set up the grid lengths and spacings */
00165   for (i=0; i<VAPBS_DIM; i++) {
00166     length[i] = thee->upper_corner[i] - thee->lower_corner
00167       [i];
00168     thee->spacs[i] = length[i]/((double)(thee->npts[i] - 1));
00169   }
00170   Vnm_print(0, "Vclist_setupGrid: Grid lengths = (%g, %g, %g)\n",
00171             length[0], length[1], length[2]);
00172
00173   Vnm_print(0, "Vclist_setupGrid: Grid lower corner = (%g, %g, %g)\n",
00174             (thee->lower_corner)[0], (thee->lower_corner)[1],
00175             (thee->lower_corner)[2]);
00176
00177   return VRC_SUCCESS;

```

```

00174
00175     #undef VCLIST_INFLATE
00176 }
00177
00178 /* Check and store parameters passed to constructor */
00179 VPRIVATE Vrc_Codes Vclist_storeParms(Vclist *thee, Valist *alist,
00180     double max_radius, int npts[VAPBS_DIM], Vclist_DomainMode
00181     mode,
00182     double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM] ) {
00183
00184     int i = 0;
00185
00186     if (alist == VNULL) {
00187         Vnm_print(2, "Vclist_ctor2: Got NULL Valist!\n");
00188         return VRC_FAILURE;
00189     } else thee->alist = alist;
00190
00191     thee->n = 1;
00192     for (i=0; i<VAPBS_DIM; i++) {
00193         if (npts[i] < 3) {
00194             Vnm_print(2,
00195                 "Vclist_ctor2: n[%d] (%d) must be greater than 2!\n",
00196                 i, npts[i]);
00197             return VRC_FAILURE;
00198         }
00199         thee->npts[i] = npts[i];
00200     }
00201     Vnm_print(0, "Vclist_ctor2: Using %d x %d x %d hash table\n",
00202             npts[0], npts[1], npts[2]);
00203
00204     thee->mode = mode;
00205     switch (thee->mode) {
00206         case CLIST_AUTO_DOMAIN:
00207             Vnm_print(0, "Vclist_ctor2: automatic domain setup.\n");
00208             break;
00209         case CLIST_MANUAL_DOMAIN:
00210             Vnm_print(0, "Vclist_ctor2: manual domain setup.\n");
00211             Vnm_print(0, "Vclist_ctor2: lower corner = [ \n");
00212             for (i=0; i<VAPBS_DIM; i++) {
00213                 thee->lower_corner[i] = lower_corner[i];
00214                 Vnm_print(0, "%g ", lower_corner[i]);
00215             }
00216             Vnm_print(0, "]\n");
00217             Vnm_print(0, "Vclist_ctor2: upper corner = [ \n");
00218             for (i=0; i<VAPBS_DIM; i++) {
00219                 thee->upper_corner[i] = upper_corner[i];
00220                 Vnm_print(0, "%g ", upper_corner[i]);
00221             }
00222             Vnm_print(0, "]\n");
00223             break;
00224         default:
00225             Vnm_print(2, "Vclist_ctor2: invalid setup mode (%d)!\n", mode);
00226             return VRC_FAILURE;
00227     }
00228
00229     thee->max_radius = max_radius;
00230     Vnm_print(0, "Vclist_ctor2: Using %g max radius\n", max_radius);
00231
00232     return VRC_SUCCESS;
00233 }
00234
00235 /* Calculate the gridpoints an atom spans */
00236 VPRIVATE void Vclist_gridSpan(Vclist *thee,
00237     Vatom *atom, /* Atom */
00238     int imin[VAPBS_DIM], /* Set to min grid indices */
00239     int imax[VAPBS_DIM] /* Set to max grid indices */
00240 ) {
00241
00242     int i;
00243     double *coord, dc, idc, rtot;
00244
00245     /* Get the position in the grid's frame of reference */
00246     coord = Vatom_getPosition(atom);
00247
00248     /* Get the range the atom radius + probe radius spans */
00249     rtot = Vatom_getRadius(atom) + thee->max_radius;
00250
00251     /* Calculate the range of grid points the inflated atom spans in the x
00252      * direction. */
00253     for (i=0; i<VAPBS_DIM; i++) {

```

```

00254     dc = coord[i] - (thee->lower_corner)[i];
00255     idc = (dc + rtot)/(thee->spacs[i]);
00256     imax[i] = (int)(ceil(idc));
00257     imax[i] = VMIN2(imax[i], thee->npts[i]-1);
00258     idc = (dc - rtot)/(thee->spacs[i]);
00259     imin[i] = (int)(floor(idc));
00260     imin[i] = VMAX2(imin[i], 0);
00261 }
00262
00263 }
00264
00265 /* Get the array index for a particular cell based on its i,j,k
00266 * coordinates */
00267 VPRIIVATE int Vclist_arrayIndex(Vclist *thee, int i, int j, int k) {
00268
00269     return (thee->npts[2])* (thee->npts[1])*i + (thee->npts[2])*j +
k;
00270
00271 }
00272
00273
00274 /* Assign atoms to cells */
00275 VPRIIVATE Vrc_Codes Vclist_assignAtoms(Vclist *thee) {
00276
00277     int iatom, i, j, k, ui, inext;
00278     int imax[VAPBS_DIM], imin[VAPBS_DIM];
00279     int totatoms;
00280     Vatom *atom;
00281     VclistCell *cell;
00282
00283
00284     /* Find out how many atoms are associated with each grid point */
00285     totatoms = 0;
00286     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist
), iatom++) {
00287
00288         /* Get grid span for atom */
00289         atom = Valist_getAtom(thee->alist, iatom);
00290         Vclist_gridSpan(thee, atom, imin, imax);
00291
00292         /* Now find and assign the grid points */
00293         VASSERT(VAPBS_DIM == 3);
00294         for ( i = imin[0]; i <= imax[0]; i++) {
00295             for ( j = imin[1]; j <= imax[1]; j++) {
00296                 for ( k = imin[2]; k <= imax[2]; k++) {
00297                     /* Get index to array */
00298                     ui = Vclist_arrayIndex(thee, i, j, k);
00299                     /* Increment number of atoms for this grid point */
00300                     cell = &(thee->cells[ui]);
00301                     (cell->natoms)++;
00302                     totatoms++;
00303                 }
00304             }
00305         }
00306     }
00307     Vnm_print(0, "Vclist_assignAtoms: Have %d atom entries\n", totatoms);
00308
00309     /* Allocate the space to store the pointers to the atoms */
00310     for (ui=0; ui<thee->n; ui++) {
00311         cell = &(thee->cells[ui]);
00312         if ( VclistCell_ctor2(cell, cell->natoms) ==
VRC_FAILURE ) {
00313             Vnm_print(2, "Vclist_assignAtoms: cell error!\n");
00314             return VRC_FAILURE;
00315         }
00316         /* Clear the counter for later use */
00317         cell->natoms = 0;
00318     }
00319
00320     /* Assign the atoms to grid points */
00321     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist
), iatom++) {
00322
00323         /* Get grid span for atom */
00324         atom = Valist_getAtom(thee->alist, iatom);
00325         Vclist_gridSpan(thee, atom, imin, imax);
00326
00327         /* Now find and assign the grid points */
00328         for ( i = imin[0]; i <= imax[0]; i++) {
00329             for ( j = imin[1]; j <= imax[1]; j++) {
00330                 for ( k = imin[2]; k <= imax[2]; k++) {

```

```

00331             /* Get index to array */
00332             ui = Vclist_arrayIndex(thee, i, j, k);
00333             cell = &(thee->cells[ui]);
00334             /* Index of next available array location */
00335             inext = cell->natoms;
00336             cell->atoms[inext] = atom;
00337             /* Increment number of atoms */
00338             (cell->natoms)++;
00339         }
00340     }
00341 }
00342 }
00343
00344     return VRC_SUCCESS;
00345 }
00346
00347 /* Main (FORTRAN stub) constructor */
00348 VPUBLIC Vrc_Codes Vclist_ctor2(Vclist *thee, Valist *
00349     alist, double max_radius,
00350     int npts[VAPBS_DIM], Vclist_DomainMode mode,
00351     double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM]) {
00352
00353     int i;
00354     VclistCell *cell;
00355
00356     /* Check and store parameters */
00357     if (Vclist_storeParms(thee, alist, max_radius, npts, mode, lower_corner,
00358                           upper_corner) == VRC_FAILURE) {
00359         Vnm_print(2, "Vclist_ctor2: parameter check failed!\n");
00360         return VRC_FAILURE;
00361     }
00362
00363     /* Set up memory */
00364     thee->vmem = Vmem_ctor("APBS::VCLIST");
00365     if (thee->vmem == VNULL) {
00366         Vnm_print(2, "Vclist_ctor2: memory object setup failed!\n");
00367         return VRC_FAILURE;
00368     }
00369
00370     /* Set up cells */
00371     thee->cells = (VclistCell*)Vmem_malloc( thee->vmem, thee
00372     ->n, sizeof(VclistCell) );
00373     if (thee->cells == VNULL) {
00374         Vnm_print(2,
00375                 "Vclist_ctor2: Failed allocating %d VclistCell objects!\n",
00376                 thee->n);
00377         return VRC_FAILURE;
00378     }
00379     for (i=0; i<thee->n; i++) {
00380         cell = &(thee->cells[i]);
00381         cell->natoms = 0;
00382     }
00383
00384     /* Set up the grid */
00385     if (Vclist_setupGrid(thee) == VRC_FAILURE) {
00386         Vnm_print(2, "Vclist_ctor2: grid setup failed!\n");
00387         return VRC_FAILURE;
00388     }
00389
00390     /* Assign atoms to grid cells */
00391     if (Vclist_assignAtoms(thee) == VRC_FAILURE) {
00392         Vnm_print(2, "Vclist_ctor2: atom assignment failed!\n");
00393         return VRC_FAILURE;
00394     }
00395
00396
00397
00398     return VRC_SUCCESS;
00399 }
00400
00401 /* Destructor */
00402 VPUBLIC void Vclist_dtor(Vclist **thee) {
00403
00404     if ((*thee) != VNULL) {
00405         Vclist_dtor2(*thee);
00406         Vmem_free(VNULL, 1, sizeof(Vclist), (void **)thee);
00407         (*thee) = VNULL;
00408     }
00409 }
```

```

00410 }
00411
00412 /* Main (stub) destructor */
00413 VPUBLIC void Vclist_dtor2(Vclist *thee) {
00414
00415     VclistCell *cell;
00416     int i;
00417
00418     for (i=0; i<thee->n; i++) {
00419         cell = &(thee->cells[i]);
00420         VclistCell_dtor2(cell);
00421     }
00422     Vmem_free(thee->vmem, thee->n, sizeof(VclistCell),
00423             (void **) &(thee->cells));
00424     Vmem_dtor(&(thee->vmem));
00425
00426 }
00427
00428 VPUBLIC VclistCell* Vclist_getCell(Vclist *thee,
00429                                     double pos[VAPBS_DIM]
00430                                     )
00431 {
00432     int i,
00433         ic[VAPBS_DIM],
00434         ui;
00435     double c[VAPBS_DIM];
00436
00437     /* Assert this before we do anything else, since its failure should fail
00438     the function */
00439     VASSERT(VAPBS_DIM == 3);
00440
00441     /* Convert to grid based coordinates */
00442     for (i=0; i<VAPBS_DIM; i++) {
00443         c[i] = pos[i] - (thee->lower_corner)[i];
00444         ic[i] = (int)(c[i]/thee->spacs[i]);
00445
00446         if (ic[i] < 0 || ic[i] >= thee->npts[i]) {
00447             return VNULL;
00448         }
00449     }
00450
00451     /* Get the array index */
00452     ui = Vclist_arrayIndex(thee, ic[0], ic[1], ic[2]);
00453
00454     return &(thee->cells[ui]);
00455 }
00456
00457 VPUBLIC VclistCell* VclistCell_ctor(int natoms) {
00458
00459     VclistCell *thee = VNULL;
00460
00461     /* Set up the structure */
00462     thee = (VclistCell*)Vmem_malloc(VNULL, 1, sizeof(VclistCell)
00463 );
00464     VASSERT( thee != VNULL);
00465     VASSERT( VclistCell_ctor2(thee, natoms) == VRC_SUCCESS
00466 );
00467
00468     return thee;
00469 }
00470
00471 VPUBLIC Vrc_Codes VclistCell_ctor2(VclistCell *thee,
00472                                     int natoms) {
00473
00474     if (thee == VNULL) {
00475         Vnm_print(2, "VclistCell_ctor2: NULL thee!\n");
00476         return VRC_FAILURE;
00477     }
00478
00479     thee->natoms = natoms;
00480     if (thee->natoms > 0) {
00481         thee->atoms = (Vatom**)Vmem_malloc(VNULL, natoms, sizeof(
00482             Vatom *));
00483         if (thee->atoms == VNULL) {
00484             Vnm_print(2,
00485                     "VclistCell_ctor2: unable to allocate space for %d atom pointers!\n"
00486                     natoms);
00487         }
00488         return VRC_FAILURE;
00489     }

```

```

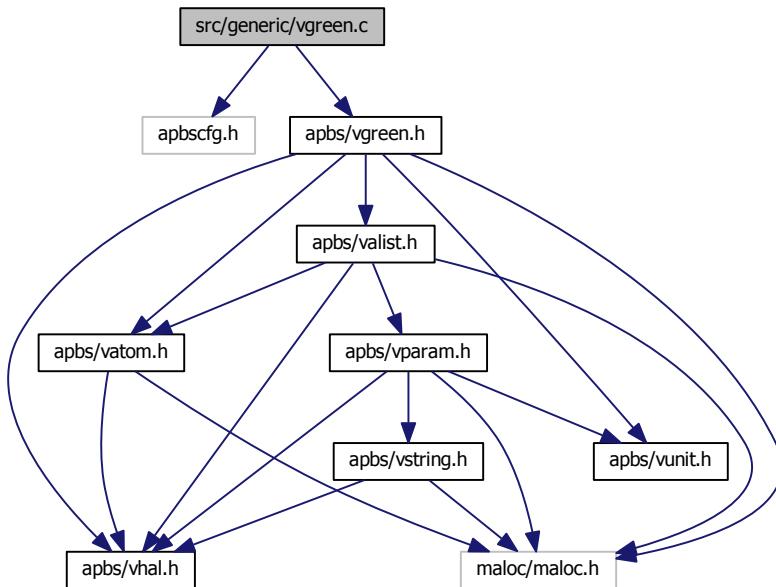
00485     }
00486
00487     return VRC_SUCCESS;
00488 }
00489 }
00490
00491 VPUBLIC void VclistCell_dtor(VclistCell **thee) {
00492
00493     if ((*thee) != VNULL) {
00494         VclistCell_dtor2(*thee);
00495         Vmem_free(VNULL, 1, sizeof(VclistCell), (void **)thee);
00496         (*thee) = VNULL;
00497     }
00498
00499 }
00500
00501 /* Main (stub) destructor */
00502 VPUBLIC void VclistCell_dtor2(VclistCell *thee) {
00503
00504     if (thee->natoms > 0) {
00505         Vmem_free(VNULL, thee->natoms, sizeof(Vatom *),
00506                   (void **)&(thee->atoms));
00507     }
00508
00509 }
00510

```

9.73 src/generic/vgreen.c File Reference

Class Vgreen methods.

```
#include "apbscfg.h"
#include "apbs/vgreen.h"
Include dependency graph for vgreen.c:
```



Functions

- VPRIVATE int **treesetup** (*Vgreen* **thee*)
- VPRIVATE int **treetcleanup** (*Vgreen* **thee*)
- VPRIVATE int **treecalc** (*Vgreen* **thee*, double **xtar*, double **ytar*, double **ztar*, double **qtar*, int *numtars*, double **tpengtar*, double **x*, double **y*, double **z*, double **q*, int *numpars*, double **fx*, double *, double **fz*, int *iflag*, int *farrdim*, int *arrdim*)
- VPUBLIC *Valist* * **Vgreen_getValist** (*Vgreen* **thee*)

Get the atom list associated with this Green's function object.
- VPUBLIC unsigned long int **Vgreen_memChk** (*Vgreen* **thee*)

Return the memory used by this structure (and its contents) in bytes.
- VPUBLIC *Vgreen* * **Vgreen_ctor** (*Valist* **alist*)

Construct the Green's function oracle.
- VPUBLIC int **Vgreen_ctor2** (*Vgreen* **thee*, *Valist* **alist*)

FORTRAN stub to construct the Green's function oracle.
- VPUBLIC void **Vgreen_dtor** (*Vgreen* ***thee*)

Destruct the Green's function oracle.
- VPUBLIC void **Vgreen_dtor2** (*Vgreen* **thee*)

FORTRAN stub to destruct the Green's function oracle.
- VPUBLIC int **Vgreen_helmholtz** (*Vgreen* **thee*, int *npos*, double **x*, double **y*, double **z*, double **val*, double *kappa*)

Get the Green's function for Helmholtz's equation integrated over the atomic point charges.
- VPUBLIC int **Vgreen_helmholtzD** (*Vgreen* **thee*, int *npos*, double **x*, double **y*, double **z*, double **gradx*, double **grady*, double **gradz*, double *kappa*)

Get the gradient of Green's function for Helmholtz's equation integrated over the atomic point charges.
- VPUBLIC int **Vgreen_coulomb_direct** (*Vgreen* **thee*, int *npos*, double **x*, double **y*, double **z*, double **val*)

Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.
- VPUBLIC int **Vgreen_coulomb** (*Vgreen* **thee*, int *npos*, double **x*, double **y*, double **z*, double **val*)

Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation or H. E. Johnston, R. Krasny FMM library (if available)
- VPUBLIC int **Vgreen_coulombD_direct** (*Vgreen* **thee*, int *npos*, double **x*, double **y*, double **z*, double **pot*, double **gradx*, double **grady*, double **gradz*)

Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.
- VPUBLIC int **Vgreen_coulombD** (*Vgreen* **thee*, int *npos*, double **x*, double **y*, double **z*, double **pot*, double **gradx*, double **grady*, double **gradz*)

Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using either direct summation or H. E. Johnston/R. Krasny FMM library (if available)

9.73.1 Detailed Description

Class *Vgreen* methods.

Author

Nathan Baker

Version**Id:**[vgreen.c](#) 1667 2011-12-02 23:22:02Z pcellis**Attention**

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2012 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](#).**9.74 vgreen.c**

```
00001  
00057 #include "apbscfg.h"  
00058 #include "apbs/vgreen.h"
```

```

00059
00060 /* Define wrappers for F77 treecode routines */
00061 #ifdef HAVE_TREE
00062 # define F77TREEPEFORCE VF77_MANGLE(treepeforce, TREEPEFORCE)
00063 # define F77DIRECT_ENG_FORCE VF77_MANGLE(direct_eng_force, DIRECT_ENG_FORCE)
00064 # define F77CLEANUP VF77_MANGLE(mycleanup, MYCLEANUP)
00065 # define F77TREE_COMP VF77_MANGLE(mytree_compp, MYTREE_COMP)
00066 # define F77TREE_COMPPFP VF77_MANGLE(mytree_comppfp, MYTREE_COMPPFP)
00067 # define F77CREATE_TREE VF77_MANGLE(mycreate_tree, MYCREATE_TREE)
00068 # define F77INITLEVELS VF77_MANGLE(myinitlevels, MYINITLEVELS)
00069 # define F77SETUP VF77_MANGLE(mysetup, MYSETUP)
00070#endif /* ifdef HAVE_TREE */
00071
00072 /* Some constants associated with the tree code */
00073 #ifdef HAVE_TREE
00074
00075 # define FMM_DIST_TOL VSMALL
00076
00077 # define FMM_IFLAG 2
00078
00079 # define FMM_ORDER 4
00080
00081 # define FMM_THETA 0.5
00082
00083 # define FMM_MAXPARNODE 150
00084
00085 # define FMM_SHRINK 1
00086
00087 # define FMM_MINLEVEL 50000
00088
00089 # define FMM_MAXLEVEL 0
00090#endif /* ifdef HAVE_TREE */
00091
00092
00093 /*
00094 * @brief Setup treecode internal structures
00095 * @ingroup Vgreen
00096 * @author Nathan Baker
00097 * @param thee Vgreen object
00098 * @return 1 if successful, 0 otherwise
00099 */
00100 VPRIIVATE int treesetup(Vgreen *thee);
00101
00102 /*
00103 * @brief Clean up treecode internal structures
00104 * @ingroup Vgreen
00105 * @author Nathan Baker
00106 * @param thee Vgreen object
00107 * @return 1 if successful, 0 otherwise
00108 */
00109 VPRIIVATE int treecleanup(Vgreen *thee);
00110
00111 /*
00112 * @brief Calculate forces or potential
00113 * @ingroup Vgreen
00114 * @author Nathan Baker
00115 * @param thee Vgreen object
00116 * @return 1 if successful, 0 otherwise
00117 */
00118 VPRIIVATE int treecalc(Vgreen *thee, double *xtar, double *ytar, double *
00119 ztar,
00120 double *qtar, int numtar, double *tpengtar, double *x, double *y,
00121 double *z, double *g, int numpars, double *fx, double *fy, double *fz,
00122 int iflag, int farrdim, int arrdim);
00123
00124 #if !defined(VINLINE_VGREEN)
00125
00126 VPUBLIC Valist* Vgreen_getValist(Vgreen *thee) {
00127
00128     VASSERT(thee != VNULL);
00129     return thee->alist;
00130 }
00131
00132 VPUBLIC unsigned long int Vgreen_memChk(Vgreen *thee) {
00133     if (thee == VNULL) return 0;
00134     return Vmem_bytes(thee->vmem);
00135 }
00136#endif /* if !defined(VINLINE_VGREEN) */
00137

```

```

00157 VPUBLIC Vgreen* Vgreen_ctor(Valist *alist) {
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 /* ifdef 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,
00210     double *x, double *y,
00211     double *z, double *val, double kappa) {
00212
00213     Vnm_print(2, "Error -- Vgreen_helmholtz not implemented yet!\n");
00214     return 0;
00215 }
00216
00217 VPUBLIC int Vgreen_helmholtzD(Vgreen *thee, int npos,
00218     double *x, double *y,
00219     double *z, double *gradx, double *grady, double *gradz, double kappa) {
00220
00221     Vnm_print(2, "Error -- Vgreen_helmholtzD not implemented yet!\n");
00222     return 0;
00223 }
00224
00225 VPUBLIC int Vgreen_coulomb_direct(Vgreen *thee, int
00226     npos, double *x,
00227     double *y, double *z, double *val) {
00228
00229     Vatom *atom;
00230     double *apos, charge, dist, dx, dy, dz, scale;
00231     double *q, qtemp, fx, fy, fz;
00232     int iatom, ipos;
00233
00234     if (thee == VNULL) {

```

```

00234     Vnm_print(2, "Vgreen_coulomb: Got NULL thee!\n");
00235     return 0;
00236 }
00237
00238     for (ipos=0; ipos<npos; ipos++) val[ipos] = 0.0;
00239
00240     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist
00241 ); iatom++) {
00242         atom = Valist_getAtom(thee->alist, iatom);
00243         apos = Vatom_getPosition(atom);
00244         charge = Vatom_getCharge(atom);
00245         for (ipos=0; ipos<npos; ipos++) {
00246             dx = apos[0] - x[ipos];
00247             dy = apos[1] - y[ipos];
00248             dz = apos[2] - z[ipos];
00249             dist = VSQRT(VSQR(dx) + VSQR(dy) + VSQR(dz));
00250             if (dist > VSMALL) val[ipos] += (charge/dist);
00251         }
00252
00253         scale = Vunit_ec/(4*Vunit_pi*Vunit_eps0*1.0e-10);
00254         for (ipos=0; ipos<npos; ipos++) val[ipos] = val[ipos]*scale;
00255
00256         return 1;
00257     }
00258
00259 VPUBLIC int Vgreen_coulomb(Vgreen *thee, int npos, double *
00260     x, double *y,
00261     double *z, double *val) {
00262
00263     Vatom *atom;
00264     double *apos, charge, dist, dx, dy, dz, scale;
00265     double *q, qtemp, fx, fy, fz;
00266     int iatom, ipos;
00267
00268     if (thee == VNULL) {
00269         Vnm_print(2, "Vgreen_coulomb: Got NULL thee!\n");
00270         return 0;
00271     }
00272
00273     for (ipos=0; ipos<npos; ipos++) val[ipos] = 0.0;
00274 #ifdef HAVE_TREE
00275
00276     /* Allocate charge array (if necessary) */
00277     if (Valist_getNumberAtoms(thee->alist) > 1) {
00278         if (npos > 1) {
00279             q = VNULL;
00280             q = Vmem_malloc(thee->vmem, npos, sizeof(double));
00281             if (q == VNULL) {
00282                 Vnm_print(2, "Vgreen_coulomb: Error allocating charge array!\n
00283 ");
00284                 return 0;
00285             }
00286         } else {
00287             q = &(qtemp);
00288         }
00289         for (ipos=0; ipos<npos; ipos++) q[ipos] = 1.0;
00290
00291         /* Calculate */
00292         treecalc(thee, x, y, z, q, npos, val, thee->xp, thee->yp, thee->zp
00293
00294             thee->qp, thee->np, &fx, &fy, &fz, 1, 1, thee->np);
00295     } else return Vgreen_coulomb_direct(thee, npos, x, y,
00296     z, val);
00297
00298     /* De-allocate charge array (if necessary) */
00299     if (npos > 1) Vmem_free(thee->vmem, npos, sizeof(double), (void **) &q);
00300
00301     scale = Vunit_ec/(4*Vunit_pi*Vunit_eps0*1.0e-10);
00302     for (ipos=0; ipos<npos; ipos++) val[ipos] = val[ipos]*scale;
00303
00304     return 1;
00305 #else /* ifdef HAVE_TREE */
00306     return Vgreen_coulomb_direct(thee, npos, x, y, z, val)
00307 ;
00308 #endif
00309

```

```

00309 }
00310
00311 VPUBLIC int Vgreen_coulombD_direct(Vgreen *thee,
00312     int npos,
00313     double *x, double *y, double *z, double *pot, double *gradx,
00314     double *grady, double *gradz) {
00315
00316     Vatom *atom;
00317     double *apos, charge, dist, dist2, idist3, dy, dz, dx, scale;
00318     double *q, qtemp;
00319     int iatom, ipos;
00320
00321     if (thee == VNULL) {
00322         Vnm_print(2, "Vgreen_coulombD: Got VNULL thee!\n");
00323         return 0;
00324     }
00325
00326     for (ipos=0; ipos<npos; ipos++) {
00327         pot[ipos] = 0.0;
00328         gradx[ipos] = 0.0;
00329         grady[ipos] = 0.0;
00330         gradz[ipos] = 0.0;
00331     }
00332
00333     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist
00334     ); iatom++) {
00335         atom = Valist_getAtom(thee->alist, iatom);
00336         apos = VatomGetPosition(atom);
00337         charge = Vatom_getCharge(atom);
00338         for (ipos=0; ipos<npos; ipos++) {
00339             dx = apos[0] - x[ipos];
00340             dy = apos[1] - y[ipos];
00341             dz = apos[2] - z[ipos];
00342             dist2 = VSQR(dx) + VSQR(dy) + VSQR(dz);
00343             dist = VSQRT(dist2);
00344             if (dist > VSMALL) {
00345                 idist3 = 1.0/(dist*dist2);
00346                 gradx[ipos] -= (charge*dx*idist3);
00347                 grady[ipos] -= (charge*dy*idist3);
00348                 gradz[ipos] -= (charge*dz*idist3);
00349                 pot[ipos] += (charge/dist);
00350             }
00351         }
00352         scale = Vunit_ec/(4*VPI*Vunit_eps0*(1.0e-10));
00353         for (ipos=0; ipos<npos; ipos++) {
00354             gradx[ipos] = gradx[ipos]*scale;
00355             grady[ipos] = grady[ipos]*scale;
00356             gradz[ipos] = gradz[ipos]*scale;
00357             pot[ipos] = pot[ipos]*scale;
00358         }
00359     }
00360     return 1;
00361 }
00362
00363 VPUBLIC int Vgreen_coulombD(Vgreen *thee, int npos, double
00364     *x, double *y,
00365     double *z, double *pot, double *gradx, double *grady, double *gradz) {
00366
00367     Vatom *atom;
00368     double *apos, charge, dist, dist2, idist3, dy, dz, dx, scale;
00369     double *q, qtemp;
00370     int iatom, ipos;
00371
00372     if (thee == VNULL) {
00373         Vnm_print(2, "Vgreen_coulombD: Got VNULL thee!\n");
00374         return 0;
00375     }
00376
00377     for (ipos=0; ipos<npos; ipos++) {
00378         pot[ipos] = 0.0;
00379         gradx[ipos] = 0.0;
00380         grady[ipos] = 0.0;
00381         gradz[ipos] = 0.0;
00382     }
00383 #ifdef HAVE_TREE
00384
00385     if (Valist_getNumberAtoms(thee->alist) > 1) {
00386         if (npos > 1) {

```

```

00387             q = VNULL;
00388             q = Vmem_malloc(thee->vmem, npos, sizeof(double));
00389             if (q == VNULL) {
00390                 Vnm_print(2, "Vgreen_coulomb: Error allocating charge array!\n"
00391                         );
00392                     return 0;
00393                 }
00394             } else {
00395                 q = &(qtemp);
00396                 for (ipos=0; ipos<npos; ipos++) q[ipos] = 1.0;
00397
00398             /* Calculate */
00399             treecalc(thee, x, y, z, q, npos, pot, thee->xp, thee->yp, thee->zp
00400
00401             thee->qp, thee->np, gradx, grady, gradz, 2, npos, thee->np
00402         );
00403             /* De-allocate charge array (if necessary) */
00404             if (npos > 1) Vmem_free(thee->vmem, npos, sizeof(double), (void **)
00405             &q);
00406             } else return Vgreen_coulombD_direct(thee, npos, x, y
00407             , z, pot,
00408             gradx, grady, gradz);
00409             scale = Vunit_ec/(4*VPI*Vunit_eps0*(1.0e-10));
00410             for (ipos=0; ipos<npos; ipos++) {
00411                 gradx[ipos] = gradx[ipos]*scale;
00412                 grady[ipos] = grady[ipos]*scale;
00413                 gradz[ipos] = gradz[ipos]*scale;
00414                 pot[ipos] = pot[ipos]*scale;
00415             }
00416             return 1;
00417 #else /* ifdef HAVE_TREE */
00418
00419     return Vgreen_coulombD_direct(thee, npos, x, y, z,
00420             pot,
00421             gradx, grady, gradz);
00422 #endif
00423
00424 }
00425
00426 VPRIIVATE int treesetup(Vgreen *thee) {
00427
00428 #ifdef HAVE_TREE
00429
00430     double dist_tol = FMM_DIST_TOL;
00431     int iflag = FMM_IFLAG;
00432     double order = FMM_ORDER;
00433     int theta = FMM_THETA;
00434     int shrink = FMM_SHRINK;
00435     int maxparnode = FMM_MAXPARNODE;
00436     int minlevel = FMM_MINLEVEL;
00437     int maxlevel = FMM_MAXLEVEL;
00438     int level = 0;
00439     int one = 1;
00440     Vatom *atom;
00441     double xyzminmax[6], *pos;
00442     int i;
00443
00444     /* Set up particle arrays with atomic coordinates and charges */
00445     Vnm_print(0, "treesetup: Initializing FMM particle arrays...\n");
00446     thee->np = Valist_getNumberAtoms(thee->alist);
00447     thee->xp = VNULL;
00448     thee->xp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(
00449         double));
00450     if (thee->xp == VNULL) {
00451         Vnm_print(2, "Vgreen_ctor2: Failed to allocate %d*sizeof(double)!\n",
00452             thee->np);
00453         return 0;
00454     }
00455     thee->yp = VNULL;
00456     thee->yp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(
00457         double));
00458     if (thee->yp == VNULL) {
00459         Vnm_print(2, "Vgreen_ctor2: Failed to allocate %d*sizeof(double)!\n",
00460             thee->np);
00461         return 0;

```

```

00460      }
00461      thee->zp = VNULL;
00462      thee->zp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(
00463          double));
00464      if (thee->zp == VNULL) {
00465          Vnm_print(2, "Vgreen_ctor2: Failed to allocate %d*sizeof(double)!\n",
00466              thee->np);
00467          return 0;
00468      thee->qp = VNULL;
00469      thee->qp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(
00470          double));
00471      if (thee->qp == VNULL) {
00472          Vnm_print(2, "Vgreen_ctor2: Failed to allocate %d*sizeof(double)!\n",
00473              thee->np);
00474          return 0;
00475      for (i=0; i<thee->np; i++) {
00476          atom = Valist_getAtom(thee->alist, i);
00477          pos = Vatom_getPosition(atom);
00478          thee->xp[i] = pos[0];
00479          thee->yp[i] = pos[1];
00480          thee->zp[i] = pos[2];
00481          thee->qp[i] = Vatom_getCharge(atom);
00482      }
00483
00484      Vnm_print(0, "treesetup: Setting things up...\n");
00485      F77SETUP(thee->xp, thee->yp, thee->zp, &(thee->np), &order, &theta,
00486      &iflag,
00487          &dist_tol, xyzminmax, &(thee->np));
00488
00489      Vnm_print(0, "treesetup: Initializing levels...\n");
00490      F77INITLEVELS(&minlevel, &maxlevel);
00491
00492      Vnm_print(0, "treesetup: Creating tree...\n");
00493      F77CREATE_TREE(&one, &(thee->np), thee->xp, thee->yp, thee->zp,
00494      thee->qp,
00495          &shrink, &maxparnode, xyzminmax, &level, &(thee->np));
00496
00497      return 1;
00498 #else /* ifdef HAVE_TREE */
00499
00500     Vnm_print(2, "treesetup: Error! APBS not linked with treecode!\n");
00501     return 0;
00502
00503 #endif /* ifdef HAVE_TREE */
00504 }
00505
00506 VPRIVATE int treecleanup(Vgreen *thee) {
00507
00508 #ifdef HAVE_TREE
00509
00510     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **) &(thee->xp
00511     ));
00512     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **) &(thee->yp
00513     ));
00514     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **) &(thee->zp
00515     ));
00516     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **) &(thee->qp
00517     ));
00518     F77CLEANUP();
00519
00520 #else /* ifdef HAVE_TREE */
00521
00522     Vnm_print(2, "treecleanup: Error! APBS not linked with treecode!\n");
00523
00524 #endif /* ifdef HAVE_TREE */
00525
00526 VPRIVATE int treecalc(Vgreen *thee, double *xtar, double *ytar, double *
00527     ztar,
00528         double *qtar, int numtars, double *tpengtar, double *x, double *y,
00529         double *z, double *q, int numpars, double *fx, double *fy, double * fz,
00530         int iflag, int farrdim, int arrdim) {
00531 #ifdef HAVE_TREE

```

```

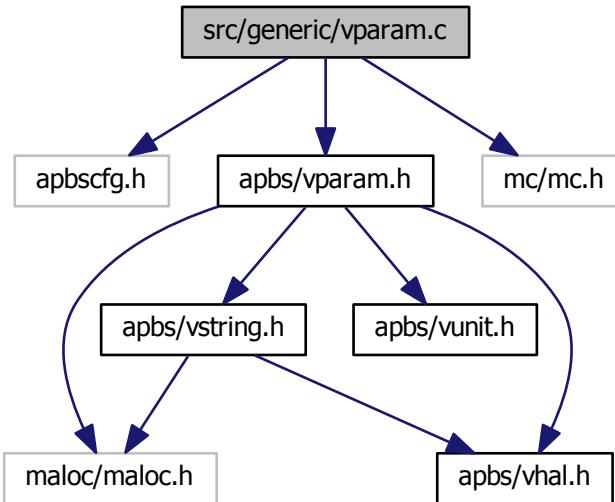
00532     int i, level, err, maxlevel, minlevel, one;
00533     double xyzminmax[6];
00534
00535
00536     if (iflag != 1) {
00537         F77TREE_COMPFP(xtar, ytar, ztar, qtar, &numtars, tpengtar, x, y, z, q,
00538                         fx, fy, fz, &numpars, &farrdim, &arrdim);
00539     } else {
00540         F77TREE_COMPP(xtar, ytar, ztar, qtar, &numtars, tpengtar, &farrdim, x,
00541                         y, z, q, &numpars, &arrdim);
00542     }
00543
00544     return 1;
00545
00546 #else /* ifdef HAVE_TREE */
00547     Vnm_Print(2, "treecalc: Error! APBS not linked with treecode!\n");
00548
00549     return 0;
00550
00551
00552 #endif /* ifdef HAVE_TREE */
00553 }
00554

```

9.75 src/generic/vparam.c File Reference

Class [Vparam](#) methods.

```
#include "apbscfg.h"
#include "apbs/vparam.h"
#include "mc/mc.h"
Include dependency graph for vparam.c:
```



Functions

- VPRIVATE int [readFlatFileLine](#) (Vio *sock, [Vparam_AtomData](#) *atom)

Read a single line of the flat file database.
- VPRIVATE int [readXMLFileAtom](#) (Vio *sock, [Vparam_AtomData](#) *atom)

Read atom information from an XML file.
- VPUBLIC unsigned long int [Vparam_memChk](#) ([Vparam](#) *thee)

Get number of bytes in this object and its members.
- VPUBLIC [Vparam_AtomData](#) * [Vparam_AtomData_ctor](#) ()

Construct the object.
- VPUBLIC int [Vparam_AtomData_ctor2](#) ([Vparam_AtomData](#) *thee)

FORTRAN stub to construct the object.
- VPUBLIC void [Vparam_AtomData_dtor](#) ([Vparam_AtomData](#) **thee)

Destroy object.
- VPUBLIC void [Vparam_AtomData_dtor2](#) ([Vparam_AtomData](#) *thee)

FORTRAN stub to destroy object.
- VPUBLIC [Vparam_ResData](#) * [Vparam_ResData_ctor](#) ([Vmem](#) *mem)

Construct the object.
- VPUBLIC int [Vparam_ResData_ctor2](#) ([Vparam_ResData](#) *thee, [Vmem](#) *mem)

FORTRAN stub to construct the object.
- VPUBLIC void [Vparam_ResData_dtor](#) ([Vparam_ResData](#) **thee)

Destroy object.
- VPUBLIC void [Vparam_ResData_dtor2](#) ([Vparam_ResData](#) *thee)

FORTRAN stub to destroy object.
- VPUBLIC [Vparam](#) * [Vparam_ctor](#) ()

Construct the object.
- VPUBLIC int [Vparam_ctor2](#) ([Vparam](#) *thee)

FORTRAN stub to construct the object.
- VPUBLIC void [Vparam_dtor](#) ([Vparam](#) **thee)

Destroy object.
- VPUBLIC void [Vparam_dtor2](#) ([Vparam](#) *thee)

FORTRAN stub to destroy object.
- VPUBLIC [Vparam_ResData](#) * [Vparam_getResData](#) ([Vparam](#) *thee, char resName[VMAX_ARGLEN])

Get residue data.
- VPUBLIC [Vparam_AtomData](#) * [Vparam_getAtomData](#) ([Vparam](#) *thee, char resName[VMAX_ARGLEN], char atomName[VMAX_ARGLEN])

Get atom data.
- VPUBLIC int [Vparam_readXMLFile](#) ([Vparam](#) *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname)

Read an XML format parameter database.
- VPUBLIC int [Vparam_readFlatFile](#) ([Vparam](#) *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname)

Read a flat-file format parameter database.
- VEXTERNC void [Vparam_AtomData_copyTo](#) ([Vparam_AtomData](#) *thee, [Vparam_AtomData](#) *dest)

Copy current atom object to destination.
- VEXTERNC void [Vparam_ResData_copyTo](#) ([Vparam_ResData](#) *thee, [Vparam_ResData](#) *dest)

Copy current residue object to destination.
- VEXTERNC void [Vparam_AtomData_copyFrom](#) ([Vparam_AtomData](#) *thee, [Vparam_AtomData](#) *src)

Copy current atom object from another.

Variables

- VPRIVATE char * MCwhiteChars = " =,\t\n\r"

Whitespace characters for socket reads.
- VPRIVATE char * MCcommChars = "#%"

Comment characters for socket reads.
- VPRIVATE char * MCxmlwhiteChars = " =,\t\n\r<>"

Whitespace characters for XML socket reads.

9.75.1 Detailed Description

Class [Vparam](#) methods.

Author

Nathan Baker

Version

Id:

[vparam.c](#) 1750 2012-07-18 18:34:27Z tuckerbeck

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](#).

9.76 vparam.c

```

00001
00057 #include "apbscfg.h"
00058 #include "apbs/vparam.h"
00059
00060 #if defined(HAVE_MC_H)
00061 #include "mc/mc.h"
00062 #endif
00063
00064 VEMBED(rcsid="$Id: vparam.c 1750 2012-07-18 18:34:27Z tuckerbeck $" )
00065
00066
00070 VPRIIVATE char *MCwhiteChars = " =,;\\t\\n\\r";
00071
00076 VPRIIVATE char *MCcommChars = "#%";
00077
00082 VPRIIVATE char *MCxmlwhiteChars = " =,;\\t\\n\\r>";
00083
00092 VPRIIVATE int readFlatFileLine(Vio *sock, Vparam_AtomData
    *atom);
00093
00102 VPRIIVATE int readXMLFileAtom(Vio *sock, Vparam_AtomData
    *atom);
00103
00104
00105 #if !defined(VINLINE_VPARAM)
00106
00107 VPUBLIC unsigned long int Vparam_memChk(Vparam *thee) {
00108     if (thee == VNULL) return 0;
00109     return Vmem_bytes(thee->vmem);
00110 }
00111
00112 #endif /* if !defined(VINLINE_VPARAM) */
00113
00114 VPUBLIC Vparam_AtomData* Vparam_AtomData_ctor
() {
00115
00116     Vparam_AtomData *thee = VNULL;
00117
00118     /* Set up the structure */
00119     thee = (Vparam_AtomData*)Vmem_malloc(VNULL, 1, sizeof(
        Vparam_AtomData));
00120     VASSERT(thee != VNULL);
00121     VASSERT(Vparam_AtomData_ctor2(thee));
00122
00123     return thee;
00124 }
00125
00126 VPUBLIC int Vparam_AtomData_ctor2(Vparam_AtomData
    *thee) { return 1; }
00127
00128 VPUBLIC void Vparam_AtomData_dtor(Vparam_AtomData
    **thee) {
00129
00130     if ((*thee) != VNULL) {
00131         Vparam_AtomData_dtor2(*thee);

```

```

00132         Vmem_free(VNULL, 1, sizeof(Vparam_AtomData), (void **)thee);
00133             (*thee) = VNULL;
00134     }
00135
00136 }
00137
00138 VPUBLIC void Vparam_AtomData_dtor2(Vparam_AtomData
00139     *thee) { ; }
00140 VPUBLIC Vparam_ResData* Vparam_ResData_ctor(
00141     Vmem *mem) {
00142     Vparam_ResData *thee = VNULL;
00143
00144     /* Set up the structure */
00145     thee = (Vparam_ResData*)Vmem_malloc(mem, 1, sizeof(
00146         Vparam_ResData) );
00147     VASSERT(thee != VNULL);
00148     VASSERT(Vparam_ResData_ctor2(thee, mem));
00149
00150     return thee;
00151 }
00152 VPUBLIC int Vparam_ResData_ctor2(Vparam_ResData
00153     *thee, Vmem *mem) {
00154     if (thee == VNULL) {
00155         Vnm_print(2, "Vparam_ResData_ctor2: Got VNULL thee!\n");
00156         return 0;
00157     }
00158     thee->vmem = mem;
00159     thee->nAtomData = 0;
00160     thee->atomData = VNULL;
00161
00162     return 1;
00163 }
00164
00165 VPUBLIC void Vparam_ResData_dtor(Vparam_ResData
00166     **thee) {
00167     if ((*thee) != VNULL) {
00168         Vparam_ResData_dtor2(*thee);
00169         Vmem_free((*thee)->vmem, 1, sizeof(Vparam_ResData), (void
00170             **)thee);
00171             (*thee) = VNULL;
00172     }
00173 }
00174
00175 VPUBLIC void Vparam_ResData_dtor2(Vparam_ResData
00176     *thee) {
00177     if (thee == VNULL) return;
00178     if (thee->nAtomData > 0) {
00179         Vmem_free(thee->vmem, thee->nAtomData, sizeof(
00180             Vparam_AtomData),
00181             (void **)(&(thee->atomData)));
00182     }
00183     thee->nAtomData = 0;
00184     thee->atomData = VNULL;
00185
00186 VPUBLIC Vparam* Vparam_ctor() {
00187
00188     Vparam *thee = VNULL;
00189
00190     /* Set up the structure */
00191     thee = (Vparam*)Vmem_malloc(VNULL, 1, sizeof(Vparam) );
00192     VASSERT(thee != VNULL);
00193     VASSERT(Vparam_ctor2(thee));
00194
00195     return thee;
00196 }
00197
00198 VPUBLIC int Vparam_ctor2(Vparam *thee) {
00199     if (thee == VNULL) {
00200         Vnm_print(2, "Vparam_ctor2: got VNULL thee!\n");
00201         return 0;
00202     }

```

```

00204
00205     thee->vmem = VNULL;
00206     thee->vmem = Vmem_ctor("APBS:VPARAM");
00207     if (thee->vmem == VNULL) {
00208         Vnm_print(2, "Vparam_ctor2: failed to init Vmem!\n");
00209         return 0;
00210     }
00211
00212     thee->nResData = 0;
00213     thee->resData = VNULL;
00214
00215     return 1;
00216 }
00217
00218 VPUBLIC void Vparam_dtor(Vparam **thee) {
00219
00220     if ((*thee) != VNULL) {
00221         Vparam_dtor2(*thee);
00222         Vmem_free(VNULL, 1, sizeof(Vparam), (void **)thee);
00223         (*thee) = VNULL;
00224     }
00225
00226 }
00227
00228 VPUBLIC void Vparam_dtor2(Vparam *thee) {
00229
00230     int i;
00231
00232     if (thee == VNULL) return;
00233
00234     /* Destroy the residue data */
00235     for (i=0; i<thee->nResData; i++) Vparam_ResData_dtor2
00236     (&(thee->resData[i]));
00237     if (thee->nResData > 0) Vmem_free(thee->vmem, thee->nResData
00238
00239         sizeof(Vparam_ResData), (void **)&(thee->resData));
00240     thee->nResData = 0;
00241     thee->resData = VNULL;
00242
00243     if (thee->vmem != VNULL) Vmem_dtor(&(thee->vmem));
00244     thee->vmem = VNULL;
00245
00246 VPUBLIC Vparam_ResData* Vparam_getResData(Vparam
00247     *thee,
00248     char resName[VMAX_ARGLEN]) {
00249
00250     int i;
00251     Vparam_ResData *res = VNULL;
00252
00253     VASSERT(thee != VNULL);
00254
00255     if ((thee->nResData == 0) || (thee->resData == VNULL)) {
00256         res = VNULL;
00257         return res;
00258     }
00259
00260     /* Look for the matching residue */
00261     for (i=0; i<thee->nResData; i++) {
00262         res = &(thee->resData[i]);
00263         if (Vstring_strcasecmp(resName, res->name) == 0)
00264             return res;
00265     }
00266
00267     /* Didn't find a matching residue */
00268     res = VNULL;
00269     Vnm_print(2, "Vparam_getResData: unable to find res=%s\n", resName);
00270
00271
00272 VPUBLIC Vparam_AtomData* Vparam_getAtomData(
00273     Vparam *thee,
00274     char resName[VMAX_ARGLEN], char atomName[VMAX_ARGLEN]) {
00275
00276     int i;
00277     Vparam_ResData *res = VNULL;
00278     Vparam_AtomData *atom = VNULL;
00279
00280     VASSERT(thee != VNULL);

```

```

00280
00281     if ((thee->nResData == 0) || (thee->resData == VNULL)) {
00282         atom = VNULL;
00283         return atom;
00284     }
00285
00286     /* Look for the matching residue */
00287     res = Vparam_getResData(thee, resName);
00288     if (res == VNULL) {
00289         atom = VNULL;
00290         Vnm_print(2, "Vparam_getAtomData: Unable to find residue %s!\n", resName);
00291         return atom;
00292     }
00293     for (i=0; i<res->nAtomData; i++) {
00294         atom = &(res->atomData[i]);
00295     if (atom == VNULL) {
00296         Vnm_print(2, "Vparam_getAtomData: got NULL atom!\n");
00297         return VNULL;
00298     }
00299     if (Vstring_strcasecmp(atomName, atom->atomName
00300 ) == 0) {
00301         return atom;
00302     }
00303
00304     /* Didn't find a matching atom/residue */
00305     atom = VNULL;
00306     Vnm_print(2, "Vparam_getAtomData: unable to find atom '%s', res '%s'\n",
00307             atomName, resName);
00308     return atom;
00309 }
00310
00311 VPUBLIC int Vparam_readXMLFile(Vparam *thee, const char
*iodev,
00312     const char *iofmt, const char *thost, const char *fname) {
00313
00314     int i, ires, natoms, nalloc, ralloc;
00315     Vparam_AtomData *atoms = VNULL;
00316     Vparam_AtomData *statoms = VNULL;
00317     Vparam_AtomData *atom = VNULL;
00318     Vparam_ResData *res = VNULL;
00319     Vparam_ResData *residues = VNULL;
00320     Vparam_ResData *tresidues = VNULL;
00321     Vio *sock = VNULL;
00322     char currResName[VMAX_ARGLEN];
00323     char tok[VMAX_ARGLEN];
00324     char endtag[VMAX_ARGLEN];
00325
00326     VASSERT(thee != VNULL);
00327
00328     /* Setup communication */
00329     sock = Vio_ctor(iodev, iofmt, thost, fname, "r");
00330     if (sock == VNULL) {
00331         Vnm_print(2, "Vparam_readXMLFile: Problem opening virtual socket %s\n",
00332                 fname);
00333         return 0;
00334     }
00335     if (Vio_accept(sock, 0) < 0) {
00336         Vnm_print(2, "Vparam_readXMLFile: Problem accepting virtual socket %s\n
00337                 ", fname);
00338         return 0;
00339     }
00340     Vio_setWhiteChars(sock, MCxmlwhiteChars);
00341     Vio_setCommChars(sock, MCcommChars);
00342
00343     /* Clear existing parameters */
00344     if (thee->nResData > 0) {
00345         Vnm_print(2, "WARNING -- CLEARING PARAMETER DATABASE!\n");
00346         for (i=0; i<thee->nResData; i++) {
00347             Vparam_ResData_dtor2(&(thee->resData[i])
00348         );
00349         }
00350         Vmem_free(thee->vmem, thee->nResData,
00351             sizeof(Vparam_ResData), (void **) &(thee->resData
00352         )));
00353     }
00354     strcpy(endtag, "/");
00355
00356     /* Set up temporary residue list */

```

```

00356
00357     ralloc = 50;
00358     residues = (Vparam_ResData*)Vmem_malloc(thee->vmem,
00359         ralloc, sizeof(Vparam_ResData));
00360
00361     /* Read until we run out of entries, allocating space as needed */
00362     while (1) {
00363
00364         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00365
00366         /* The first token should be the start tag */
00367
00368         if (Vstring_strcasecmp(endtag, "/") == 0) strcat(
00369             endtag, tok);
00370
00371         if (Vstring_strcasecmp(tok, "residue") == 0) {
00372             if (thee->nResData >= ralloc) {
00373                 residues = (Vparam_ResData*)Vmem_malloc(thee->
00374                     vmem, 2*ralloc, sizeof(Vparam_ResData));
00375                 VASSERT(residues != VNULL);
00376                 for (i=0; i<thee->nResData; i++) {
00377                     Vparam_ResData_copyTo(&(residues[i]),
00378                         &(residues[i]));
00379                 }
00380                 Vmem_free(thee->vmem, ralloc, sizeof(Vparam_ResData
00381             ),
00382                         (void **) &(residues));
00383                 residues = residues;
00384                 residues = VNULL;
00385                 ralloc = 2*ralloc;
00386             }
00387
00388             /* Initial space for this residue's atoms */
00389             nalloc = 20;
00390             natoms = 0;
00391             atoms = (Vparam_AtomData*)Vmem_malloc(thee->vmem,
00392                 nalloc, sizeof(Vparam_AtomData));
00393
00394             } else if (Vstring_strcasecmp(tok, "name") == 0) {
00395                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1); /* value */
00396                 strcpy(currResName, tok);
00397                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1); /* </name> */
00398             } else if (Vstring_strcasecmp(tok, "atom") == 0) {
00399                 if (natoms >= nalloc) {
00400                     atoms = (Vparam_AtomData*)Vmem_malloc(thee->
00401                         vmem, 2*nalloc, sizeof(Vparam_AtomData));
00402                     VASSERT(atoms != VNULL);
00403                     for (i=0; i<natoms; i++) {
00404                         Vparam_AtomData_copyTo(&(atoms[i]), &
00405                             (atoms[i]));
00406                     }
00407                     Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData
00408             ),
00409                         (void **) &(atoms));
00410                     atoms = atoms;
00411                     atoms = VNULL;
00412                     nalloc = 2*nalloc;
00413                 }
00414                 atom = &(atoms[natoms]);
00415                 if (!readXMLFileAtom(sock, atom)) break;
00416                 natoms++;
00417
00418             } else if (Vstring_strcasecmp(tok, "/residue") == 0)
00419             {
00420                 res = &(residues[thee->nResData]);
00421                 Vparam_ResData_ctor2(res, thee->vmem);
00422                 res->atomData = (Vparam_AtomData*)Vmem_malloc(
00423                     thee->vmem, natoms,
00424                         sizeof(Vparam_AtomData));
00425                 res->nAtomData = natoms;
00426                 strcpy(res->name, currResName);
00427                 for (i=0; i<natoms; i++) {
00428                     strcpy(atoms[i].resName, currResName);
00429                     Vparam_AtomData_copyTo(&(atoms[i]), &(res->
00430                         atomData[i]));
00431                 }
00432                 Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData
00433             ), (void **) &(atoms));
00434             (thee->nResData)++;
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01960
01961
01962
01963
01964
01965
01966
01967
01968
01969
01970
01971
01972
01973
01974
01975
01976
01977
01978
01979
01980
01981
01982
01983
01984
01985
01986
01987
01988
01989
01990
01991
01992
01993
01994
01995
01996
01997
01998
01999
02000
02001
02002
02003
02004
02005
02006
02007
02008
02009
02010
02011
02012
02013
02014
02015
02016
02017
02018
02019
02020
02021
02022
02023
02024
02025
02026
02027
02028
02029
02030
02031
02032
02033
02034
02035
02036
02037
02038
02039
02040
02041
02042
02043
02044
02045
02046
02047
02048
02049
02050
02051
02052
02053
02054
02055
02056
02057
02058
02059
02060
02061
02062
02063
02064
02065
02066
02067
02068
02069
02070
02071
02072
02073
02074
02075
02076
02077
02078
02079
02080
02081
02082
02083
02084
02085
02086
02087
02088
02089
02090
02091
02092
02093
02094
02095
02096
02097
02098
02099
02100
02101
02102
02103
02104
02105
02106
02107
02108
0210
```

```

00424         } else if (Vstring_strcasecmp(tok, endtag) == 0)
00425             break;
00426
00427     /* Initialize and copy the residues into the Vparam object */
00428
00429     thee->resData = (Vparam_ResData*)Vmem_malloc(thee->
00430                                                 vmem, thee->nResData,
00431                                                 sizeof(Vparam_ResData));
00432     for (ires=0; ires<thee->nResData; ires++) {
00433         Vparam_ResData_copyTo(&(residues[ires]), &(thee->
00434             resData[ires]));
00435     }
00436
00437     /* Destroy temporary atom space */
00438     Vmem_free(thee->vmem, ralloc, sizeof(Vparam_ResData), (
00439         void **) &(residues));
00440
00441     /* Shut down communication */
00442     Vio_acceptFree(sock);
00443     Vio_dtor(&sock);
00444
00445     return 1;
00446
00447 VERROR1:
00448     Vnm_print(2, "Vparam_readXMLFile: Got unexpected EOF reading parameter
00449     file!\n");
00450     return 0;
00451 }
00452
00453 VPUBLIC int Vparam_readFlatFile(Vparam *thee, const
00454     char *iodev,
00455     const char *iofmt, const char *thost, const char *fname) {
00456
00457     int i, iatom, jatom, ires, natoms, nalloc;
00458     Vparam_AtomData *atoms = VNULL;
00459     Vparam_AtomData *statoms = VNULL;
00460     Vparam_AtomData *atom = VNULL;
00461     Vparam_ResData *res = VNULL;
00462     Vio *sock = VNULL;
00463     char currResName[VMAX_ARGLEN];
00464
00465     VASSERT(thee != VNULL);
00466
00467     /* Setup communication */
00468     sock = Vio_ctor(iodev, iofmt, thost, fname, "r");
00469     if (sock == VNULL) {
00470         Vnm_print(2, "Vparam_readFlatFile: Problem opening virtual socket %s\n"
00471             "                %s",
00472             fname);
00473     }
00474     if (Vio_accept(sock, 0) < 0) {
00475         Vnm_print(2, "Vparam_readFlatFile: Problem accepting virtual socket %s
00476             %s",
00477             fname);
00478     }
00479     Vio_setWhiteChars(sock, MCwhiteChars);
00480     Vio_setCommChars(sock, MCcommChars);
00481
00482     /* Clear existing parameters */
00483     if (thee->nResData > 0) {
00484         Vnm_print(2, "WARNING -- CLEARING PARAMETER DATABASE!\n");
00485         for (i=0; i<thee->nResData; i++) {
00486             Vparam_ResData_dtor2(&(thee->resData[i]))
00487         }
00488         Vmem_free(thee->vmem, thee->nResData,
00489             sizeof(Vparam_ResData), (void **) &(thee->resData
00490             ));
00491     }
00492
00493     /* Initial space for atoms */
00494     nalloc = 200;
00495     natoms = 0;
00496     atoms = (Vparam_AtomData*)Vmem_malloc(thee->vmem, nalloc
00497         , sizeof(Vparam_AtomData));
00498
00499     /* Read until we run out of entries, allocating space as needed */

```

```

00494     while (1) {
00495         if (natomsms >= nalloc) {
00496             tatoms = (Vparam_AtomData*)Vmem_malloc(thee->vmem
00497             , 2*nalloc, sizeof(Vparam_AtomData));
00498             VASSERT(tatoms != VNULL);
00499             for (i=0; i<natoms; i++) {
00500                 Vparam_AtomData_copyTo(&(atoms[i]), &
00501                     tatoms[i]);
00502             }
00503             Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData
00504             )),
00505             (void **)&(atoms));
00506             atoms = tatoms;
00507             tatoms = VNULL;
00508             nalloc = 2*nalloc;
00509         }
00510         atom = &(atoms[natoms]);
00511         if (!readFlatFileLine(sock, atom)) break;
00512         natoms++;
00513     }
00514     if (natoms == 0) return 0;
00515     /* Count the number of residues */
00516     thee->nResData = 1;
00517     strcpy(currResName, atoms[0].resName);
00518     for (i=1; i<natoms; i++) {
00519         if (Vstring_strcasecmp(atoms[i].resName, currResName)
00520             != 0) {
00521             strcpy(currResName, atoms[i].resName);
00522             (thee->nResData)++;
00523         }
00524     }
00525     /* Create the residues */
00526     thee->resData = (Vparam_ResData*)Vmem_malloc(thee->
00527         vmem, thee->nResData,
00528         sizeof(Vparam_ResData));
00529     VASSERT(thee->resData != VNULL);
00530     for (i=0; i<(thee->nResData); i++) {
00531         res = &(thee->resData[i]);
00532         Vparam_ResData_ctor2(res, thee->vmem);
00533     }
00534     /* Count the number of atoms per residue */
00535     ires = 0;
00536     res = &(thee->resData[ires]);
00537     res->nAtomData = 1;
00538     strcpy(res->name, atoms[0].resName);
00539     for (i=1; i<natoms; i++) {
00540         if (Vstring_strcasecmp(atoms[i].resName, res->name
00541             ) != 0) {
00542             (ires)++;
00543             res = &(thee->resData[ires]);
00544             res->nAtomData = 1;
00545             strcpy(res->name, atoms[i].resName);
00546         } else (res->nAtomData)++;
00547     }
00548     /* Allocate per-residue space for atoms */
00549     for (ires=0; ires<thee->nResData; ires++) {
00550         res = &(thee->resData[ires]);
00551         res->atomData = (Vparam_AtomData*)Vmem_malloc(thee->
00552             vmem, res->nAtomData,
00553             sizeof(Vparam_AtomData));
00554         iatom = 0;
00555         Vparam_AtomData_copyTo(&(atoms[0]), &(res->atomData
00556             [iatom]));
00557         for (ires=0; ires<thee->nResData; ires++) {
00558             res = &(thee->resData[ires]);
00559             for (jatom=0; jatom<res->nAtomData; jatom++) {
00560                 Vparam_AtomData_copyTo(&(atoms[iatom]), &(res
00561                 ->atomData[jatom]));
00562                 iatom++;
00563             }
00564         }
00565     }
00566     /* Shut down communication */

```

```

00566     Vio_acceptFree(sock);
00567     Vio_dtor(&sock);
00568
00569     /* Destroy temporary atom space */
00570     Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData), (
00571         void **)(&(atoms)));
00572     return 1;
00573 }
00575
00576 VEXTERNC void Vparam_AtomData_copyTo(Vparam_AtomData
00577     *thee,
00578     Vparam_AtomData *dest) {
00579
00580     VASSERT(thee != VNULL);
00581     VASSERT(dest != VNULL);
00582
00583     strcpy(dest->atomName, thee->atomName);
00584     strcpy(dest->resName, thee->resName);
00585     dest->charge = thee->charge;
00586     dest->radius = thee->radius;
00587     dest->epsilon = thee->epsilon;
00588 }
00589
00590 VEXTERNC void Vparam_ResData_copyTo(Vparam_ResData
00591     *thee,
00592     Vparam_ResData *dest) {
00593
00594     int i;
00595
00596     VASSERT(thee != VNULL);
00597     VASSERT(dest != VNULL);
00598
00599     strcpy(dest->name, thee->name);
00600     dest->vmem = thee->vmem;
00601     dest->nAtomData = thee->nAtomData;
00602
00603     dest->atomData = (Vparam_AtomData*)Vmem_malloc(thee
00604         ->vmem, dest->nAtomData,
00605             sizeof(Vparam_AtomData));
00606
00607     for (i=0; i<dest->nAtomData; i++) {
00608         Vparam_AtomData_copyTo(&(thee->atomData[i])
00609             , &(dest->atomData[i]));
00610     }
00611     Vmem_free(thee->vmem, thee->nAtomData, sizeof(Vparam_AtomData)
00612         ),
00613         (void **)(&(thee->atomData));
00614 }
00615
00616 VPRIVATE int readXMLFileAtom(Vio *sock, Vparam_AtomData
00617     *atom) {
00618
00619     double dtmp;
00620     char tok[VMAX_BUFSIZE];
00621     int chgflag, radflag, nameflag;
00622
00623     VASSERT(atom != VNULL);
00624
00625     if (Vio_scanf(sock, "%s", tok) != 1) return 0;
00626
00627     chgflag = 0;
00628     radflag = 0;
00629     nameflag = 0;
00630
00631     while (1)
00632     {
00633         if (Vstring_strcasecmp(tok, "name") == 0) {
00634             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00635             if (strlen(tok) > VMAX_ARGLEN) {
00636                 Vnm_print(2, "Vparam_readXMLFileAtom: string (%s) too long \
00637 (%d)!\n", tok, strlen(tok));
00638                 return 0;
00639             }

```

```

00638         nameflag = 1;
00639         strcpy(atom->atomName, tok);
00640     } else if (Vstring_strcasecmp(tok, "charge") == 0)
00641     {
00642         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00643         if (sscanf(tok, "%lf", &dtmp) != 1) {
00644             Vnm_print(2, "Vparam_readXMLFileAtom: Unexpected token (%s"
00645             while \
00646             parsing charge!\n", tok);
00647             return 0;
00648         }
00649         chgflag = 1;
00650         atom->charge = dttmp;
00651     } else if (Vstring_strcasecmp(tok, "radius") == 0)
00652     {
00653         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00654         if (sscanf(tok, "%lf", &dtmp) != 1) {
00655             Vnm_print(2, "Vparam_readXMLFileAtom: Unexpected token (%s"
00656             while \
00657             parsing radius!\n", tok);
00658             return 0;
00659         }
00660         radflag = 1;
00661         atom->radius = dttmp;
00662     } else if (Vstring_strcasecmp(tok, "epsilon") == 0)
00663     {
00664         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00665         if (sscanf(tok, "%lf", &dtmp) != 1) {
00666             Vnm_print(2, "Vparam_readXMLFileAtom: Unexpected token (%s"
00667             while \
00668             parsing epsilon!\n", tok);
00669             return 0;
00670         }
00671         atom->epsilon = dttmp;
00672     } else if ((Vstring_strcasecmp(tok, "/atom") == 0)
00673 || \
00674         (Vstring_strcasecmp(tok, "atom") == 0)) {
00675         if (chgflag && radflag && nameflag) return 1;
00676         else if (!chgflag) {
00677             Vnm_print(2, "Vparam_readXMLFileAtom: Reached end of atom
00678             without \
00679             setting the charge!\n");
00680             return 0;
00681         } else if (!radflag) {
00682             Vnm_print(2, "Vparam_readXMLFileAtom: Reached end of atom
00683             without \
00684             setting the radius!\n");
00685             return 0;
00686         } else if (!nameflag) {
00687             Vnm_print(2, "Vparam_readXMLFileAtom: Reached end of atom
00688             without \
00689             setting the name!\n");
00690             return 0;
00691         }
00692     }
00693     /* If we get here something wrong has happened */
00694     VJMPERR1(1);
00695     VERROR1:
00696     Vnm_print(2, "Vparam_readXMLFileAtom: Got unexpected EOF reading parameter
00697     file!\n");
00698     return 0;
00699 }
00700
00701 VPRIPRIVATE int readFlatFileLine(Vio *sock, Vparam_AtomData
00702 *atom) {
00703     double dtmp;
00704     char tok[VMAX_BUFSIZE];
00705     VASSERT(atom != VNULL);
00706     if (Vio_scanf(sock, "%s", tok) != 1) return 0;
00707     if (strlen(tok) > VMAX_ARGLEN) {
00708         Vnm_print(2, "Vparam_readFlatFile: string (%s) too long (%d)!\n",
00709             tok, strlen(tok));

```

```

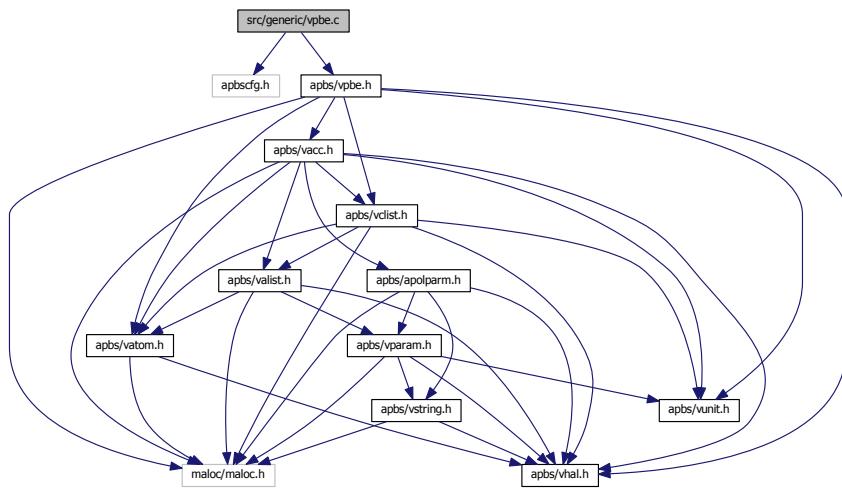
00707         return 0;
00708     }
00709     strcpy(atom->resName, tok);
00710     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00711     if (strlen(tok) > VMAX_ARGLEN) {
00712         Vnm_print(2, "Vparam_readFlatFile: string (%s) too long (%d)!\n",
00713                 tok, strlen(tok));
00714         return 0;
00715     }
00716     strcpy(atom->atomName, tok);
00717     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00718     if (sscanf(tok, "%lf", &dtmp) != 1) {
00719         Vnm_print(2, "Vparam_readFlatFile: Unexpected token (%s) while \
00720 parsing charge!\n", tok);
00721         return 0;
00722     }
00723     atom->charge = dttmp;
00724     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00725     if (sscanf(tok, "%lf", &dtmp) != 1) {
00726         Vnm_print(2, "Vparam_readFlatFile: Unexpected token (%s) while \
00727 parsing radius!\n", tok);
00728         return 0;
00729     }
00730     atom->radius = dttmp;
00731     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00732     if (sscanf(tok, "%lf", &dtmp) != 1) {
00733         Vnm_print(2, "Vparam_readFlatFile: Unexpected token (%s) while \
00734 parsing radius!\n", tok);
00735         return 0;
00736     }
00737     atom->epsilon = dttmp;
00738
00739     return 1;
00740
00741 VERROR1:
00742     Vnm_print(2, "Vparam_readFlatFile: Got unexpected EOF reading parameter
file!\n");
00743     return 0;
00744 }
```

9.77 src/generic/vpbe.c File Reference

Class Vpbe methods.

```
#include "apbscfg.h"
#include "apbs/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-Hückel parameter.
 - VPUBLIC double Vpbe_getDeblen (Vpbe *thee)

- VPUBLIC double `Vpbe_getZkappa2 (Vpbe *thee)`

Get modified squared Debye-Hückel parameter.
- VPUBLIC double `Vpbe_getZmagic (Vpbe *thee)`

Get charge scaling factor.
- VPUBLIC double `Vpbe_getSoluteRadius (Vpbe *thee)`

Get sphere radius which bounds biomolecule.
- VPUBLIC double `Vpbe_getSoluteXlen (Vpbe *thee)`

Get length of solute in x dimension.
- VPUBLIC double `Vpbe_getSoluteYlen (Vpbe *thee)`

Get length of solute in y dimension.
- VPUBLIC double `Vpbe_getSoluteZlen (Vpbe *thee)`

Get length of solute in z dimension.
- VPUBLIC double `Vpbe_getSoluteCharge (Vpbe *thee)`

Get total solute charge.
- VPUBLIC double `Vpbe_getzmem (Vpbe *thee)`

Get z position of the membrane bottom.
- VPUBLIC double `Vpbe_getLmem (Vpbe *thee)`

*Get length of the membrane (A)
author Michael Grabe.*
- VPUBLIC double `Vpbe_getmembraneDiel (Vpbe *thee)`

Get membrane dielectric constant.
- VPUBLIC double `Vpbe_getmemv (Vpbe *thee)`

Get membrane potential (kT)
- VPUBLIC `Vpbe * Vpbe_ctor (Valist *alist, int ionNum, double *ionConc, double *ionRadii, double *ionQ, double T, double soluteDiel, double solventDiel, double solventRadius, int focusFlag, double sdens, double z_mem, double L, double membraneDiel, double V)`

Construct Vpbe object.
- VPUBLIC int `Vpbe_ctor2 (Vpbe *thee, Valist *alist, int ionNum, double *ionConc, double *ionRadii, double *ionQ, double T, double soluteDiel, double solventDiel, double solventRadius, int focusFlag, double sdens, double z_mem, double L, double membraneDiel, double V)`

FORTRAN stub to construct Vpbe objct.
- VPUBLIC void `Vpbe_dtor (Vpbe **thee)`

Object destructor.
- VPUBLIC void `Vpbe_dtor2 (Vpbe *thee)`

FORTRAN stub object destructor.
- VPUBLIC double `Vpbe_getCoulombEnergy1 (Vpbe *thee)`

Calculate coulombic energy of set of charges.
- VPUBLIC unsigned long int `Vpbe_memChk (Vpbe *thee)`

Return the memory used by this structure (and its contents) in bytes.
- VPUBLIC int `Vpbe_getIons (Vpbe *thee, int *nion, double ionConc[MAXION], double ionRadii[MAXION], double ionQ[MAXION])`

Get information about the counterion species present.

9.77.1 Detailed Description

Class Vpbe methods.

Author

Nathan Baker

Version

Id:

[vpbe.c](#) 1750 2012-07-18 18:34:27Z tuckerbeck

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](#).

9.78 vpbe.c

```

00001
00057 #include "apbscfg.h"
00058 #include "apbs/vpbe.h"
00059
00060 /* //////////////////////////////// */
00061 // Class Vpbe: Private method declaration
00063 #define MAX_SPLINE_WINDOW 0.5
00064
00065 /* //////////////////////////////// */
00066 // Class Vpbe: Inlineable methods
00068 #if !defined(VINLINE_VPBE)
00069
00070 VPUBLIC Valist* Vpbe_getValist(Vpbe *thee) {
00071
00072     VASSERT(thee != VNULL);
00073     return thee->alist;
00074
00075 }
00076
00077 VPUBLIC Vacc* Vpbe_getVacc(Vpbe *thee) {
00078
00079     VASSERT(thee != VNULL);
00080     VASSERT(thee->paramFlag);
00081     return thee->acc;
00082
00083 }
00084
00085 VPUBLIC double Vpbe_getBulkIonicStrength(Vpbe *thee) {
00086
00087     VASSERT(thee != VNULL);
00088     VASSERT(thee->paramFlag);
00089     return thee->bulkIonicStrength;
00090 }
00091
00092 VPUBLIC double Vpbe_getTemperature(Vpbe *thee) {
00093
00094     VASSERT(thee != VNULL);
00095     VASSERT(thee->paramFlag);
00096     return thee->T;
00097
00098 }
00099
00100 VPUBLIC double Vpbe_getSoluteDiel(Vpbe *thee) {
00101
00102     VASSERT(thee != VNULL);
00103     VASSERT(thee->paramFlag);
00104     return thee->soluteDiel;
00105
00106 }
00107
00108 VPUBLIC double* Vpbe_getSoluteCenter(Vpbe *thee) {
00109
00110     VASSERT(thee != VNULL);
00111     return thee->soluteCenter;
00112 }
00113
00114 VPUBLIC double Vpbe_getSolventDiel(Vpbe *thee) {
00115
00116     VASSERT(thee != VNULL);
00117     VASSERT(thee->paramFlag);
00118     return thee->solventDiel;
00119 }
00120
00121 VPUBLIC double Vpbe_getSolventRadius(Vpbe *thee) {
00122
00123     VASSERT(thee != VNULL);
00124     VASSERT(thee->paramFlag);
00125     return thee->solventRadius;
00126 }
00127
00128 VPUBLIC double Vpbe_getMaxIonRadius(Vpbe *thee) {
00129
00130     VASSERT(thee != VNULL);
00131     VASSERT(thee->paramFlag);
00132     return thee->maxIonRadius;
00133 }
00134
00135 VPUBLIC double Vpbe_getXkappa(Vpbe *thee) {

```

```

00136
00137     VASSERT(thee != VNULL);
00138     VASSERT(thee->paramFlag);
00139     return thee->xkappa;
00140 }
00141
00142 VPUBLIC double Vpbe_getDeblen(Vpbe *thee) {
00143
00144     VASSERT(thee != VNULL);
00145     VASSERT(thee->paramFlag);
00146     return thee->deblen;
00147 }
00148
00149 VPUBLIC double Vpbe_getZkappa2(Vpbe *thee) {
00150
00151     VASSERT(thee != VNULL);
00152     VASSERT(thee->paramFlag);
00153     return thee->z kappa2;
00154 }
00155
00156 VPUBLIC double Vpbe_getZmagic(Vpbe *thee) {
00157
00158     VASSERT(thee != VNULL);
00159     VASSERT(thee->paramFlag);
00160     return thee->zmagic;
00161 }
00162
00163 VPUBLIC double Vpbe_getSoluteRadius(Vpbe *thee) {
00164
00165     VASSERT(thee != VNULL);
00166     return thee->soluteRadius;
00167 }
00168
00169 VPUBLIC double Vpbe_getSoluteXlen(Vpbe *thee) {
00170
00171     VASSERT(thee != VNULL);
00172     return thee->soluteXlen;
00173 }
00174
00175 VPUBLIC double Vpbe_getSoluteYlen(Vpbe *thee) {
00176
00177     VASSERT(thee != VNULL);
00178     return thee->soluteYlen;
00179 }
00180
00181 VPUBLIC double Vpbe_getSoluteZlen(Vpbe *thee) {
00182
00183     VASSERT(thee != VNULL);
00184     return thee->soluteZlen;
00185 }
00186
00187 VPUBLIC double Vpbe_getSoluteCharge(Vpbe *thee) {
00188
00189     VASSERT(thee != VNULL);
00190     return thee->soluteCharge;
00191 }
00192
00193 /* /////////////////////////////////
00194 // Routine: Vpbe_getzmem
00195 // Purpose: This routine returns values stored in the structure thee.
00196 // Author: Michael Grabe
00197 VPUBLIC double Vpbe_getzmem(Vpbe *thee) {
00198
00199     VASSERT(thee != VNULL);
00200     VASSERT(thee->param2Flag);
00201     return thee->z_mem;
00202 }
00203
00204
00205 /* /////////////////////////////////
00206 // Routine: Vpbe_getLmem
00207 // Purpose: This routine returns values stored in the structure thee.
00208 // Author: Michael Grabe
00209 VPUBLIC double Vpbe_getLmem(Vpbe *thee) {
00210
00211     VASSERT(thee != VNULL);
00212     VASSERT(thee->param2Flag);
00213     return thee->L;
00214 }
00215
00216
00217 /* /////////////////////////////////
00218 // Routine: Vpbe_getmembraneDiel

```

```

00219 // Purpose: This routine returns values stored in the structure thee.
00220 // Author: Michael Grabe
00221 VPUBLIC double Vpbe_getmembraneDiel(Vpbe *thee) {
00222
00223     VASSERT(thee != VNULL);
00224     VASSERT(thee->param2Flag);
00225     return thee->membraneDiel;
00226 }
00227
00228 /* /////////////////////////////////
00229 // Routine: Vpbe_getmemv
00230 // Purpose: This routine returns values stored in the structure thee.
00231 // Author: Michael Grabe
00232 VPUBLIC double Vpbe_getmemv(Vpbe *thee) {
00233
00234     VASSERT(thee != VNULL);
00235     VASSERT(thee->param2Flag);
00236     return thee->V;
00237 }
00238
00239
00240 #endif /* if !defined(VINLINE_VPBE) */
00241
00242 /* /////////////////////////////////
00243 // Class Vpbe: Non-inlineable methods
00244
00245 VPUBLIC Vpbe* Vpbe_ctor(Valist *alist, int ionNum, double *ionConc,
00246     double *ionRadii, double *ionQ, double T,
00247     double soluteDiel, double solventDiel,
00248     double solventRadius, int focusFlag, double sdens,
00249     double z_mem, double L, double membraneDiel, double V ) {
00250
00251     /* Set up the structure */
00252     Vpbe *thee = VNULL;
00253     thee = (Vpbe*)Vmem_malloc(VNULL, 1, sizeof(Vpbe) );
00254     VASSERT( thee != VNULL );
00255     VASSERT( Vpbe_ctor2(thee, alist, ionNum, ionConc,
00256         ionRadii, ionQ,
00257         T, soluteDiel, solventDiel, solventRadius
00258         , focusFlag, sdens,
00259         z_mem, L, membraneDiel, V) );
00260
00261     return thee;
00262 }
00263
00264
00265 VPUBLIC int Vpbe_ctor2(Vpbe *thee, Valist *alist, int
00266     ionNum,
00267     double *ionConc, double *ionRadii,
00268     double *ionQ, double T, double soluteDiel,
00269     double solventDiel, double solventRadius, int
00270     focusFlag,
00271     double sdens, double z_mem, double L, double membraneDiel
00272     ,
00273     double V) {
00274
00275     int i, iatom, inhash[3];
00276     double atomRadius;
00277     Vatom *atom;
00278     double center[3] = {0.0, 0.0, 0.0};
00279     double lower_corner[3] = {0.0, 0.0, 0.0};
00280     double upper_corner[3] = {0.0, 0.0, 0.0};
00281     double disp[3], dist, radius, charge, xmin, xmax, ymin, ymax, zmin, zmax;
00282     double x, y, z, netCharge;
00283     double nhash[3];
00284     const double N_A = 6.022045000e+23;
00285     const double e_c = 4.803242384e-10;
00286     const double k_B = 1.380662000e-16;
00287     const double pi = 4. * VATAN(1.);
00288
00289     /* Set up memory management object */
00290     thee->vmem = Vmem_ctor("APBS::VPBE");
00291
00292     VASSERT(thee != VNULL);
00293     if (alist == VNULL) {
00294         Vnm_print(2, "Vpbe_ctor2: Got null pointer to Valist object!\n");
00295         return 0;
00296     }
00297
00298     /* **** STUFF THAT GETS DONE FOR EVERYONE **** */
00299     /* Set pointers */
00300     thee->alist = alist;

```

```

00298     thee->paramFlag = 0;
00299
00300     /* Determine solute center */
00301     center[0] = thee->alist->center[0];
00302     center[1] = thee->alist->center[1];
00303     center[2] = thee->alist->center[2];
00304     thee->soluteCenter[0] = center[0];
00305     thee->soluteCenter[1] = center[1];
00306     thee->soluteCenter[2] = center[2];
00307
00308     /* Determine solute length and charge*/
00309     radius = 0;
00310     atom = Valist_getAtom(thee->alist, 0);
00311     xmin = Vatom_getPosition(atom)[0];
00312     xmax = Vatom_getPosition(atom)[0];
00313     ymin = Vatom_getPosition(atom)[1];
00314     ymax = Vatom_getPosition(atom)[1];
00315     zmin = Vatom_getPosition(atom)[2];
00316     zmax = Vatom_getPosition(atom)[2];
00317     charge = 0;
00318     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist
00319     ); iatom++) {
00320         atom = Valist_getAtom(thee->alist, iatom);
00321         atomRadius = Vatom_getRadius(atom);
00322         x = Vatom_getPosition(atom)[0];
00323         y = Vatom_getPosition(atom)[1];
00324         z = Vatom_getPosition(atom)[2];
00325         if ((x+atomRadius) > xmax) xmax = x + atomRadius;
00326         if ((x-atomRadius) < xmin) xmin = x - atomRadius;
00327         if ((y+atomRadius) > ymax) ymax = y + atomRadius;
00328         if ((y-atomRadius) < ymin) ymin = y - atomRadius;
00329         if ((z+atomRadius) > zmax) zmax = z + atomRadius;
00330         if ((z-atomRadius) < zmin) zmin = z - atomRadius;
00331         disp[0] = (x - center[0]);
00332         disp[1] = (y - center[1]);
00333         disp[2] = (z - center[2]);
00334         dist = (disp[0]*disp[0]) + (disp[1]*disp[1]) + (disp[2]*disp[2]);
00335         dist = VSQRT(dist) + atomRadius;
00336         if (dist > radius) radius = dist;
00337         charge += Vatom_getCharge(Valist_getAtom(
00338             thee->alist, iatom));
00339     }
00340     thee->soluteRadius = radius;
00341     Vnm_print(0, "Vpbe_ctor2: solute radius = %g\n", radius);
00342     thee->soluteXlen = xmax - xmin;
00343     thee->soluteYlen = ymax - ymin;
00344     thee->soluteZlen = zmax - zmin;
00345     Vnm_print(0, "Vpbe_ctor2: solute dimensions = %g x %g x %g\n",
00346             thee->soluteXlen, thee->soluteYlen, thee->
00347             soluteZlen);
00348     thee->soluteCharge = charge;
00349     Vnm_print(0, "Vpbe_ctor2: solute charge = %g\n", charge);
00350
00351     /* Set parameters */
00352     thee->numIon = ionNum;
00353     if (thee->numIon >= MAXION) {
00354         Vnm_print(2, "Vpbe_ctor2: Too many ion species (MAX = %d) !\n",
00355                 MAXION);
00356         return 0;
00357     }
00358     thee->bulkIonicStrength = 0.0;
00359     thee->maxIonRadius = 0.0;
00360     netCharge = 0.0;
00361     for (i=0; i<thee->numIon; i++) {
00362         thee->ionConc[i] = ionConc[i];
00363         thee->ionRadii[i] = ionRadii[i];
00364         if (ionRadii[i] > thee->maxIonRadius) thee->maxIonRadius
00365             = ionRadii[i];
00366         thee->ionQ[i] = ionQ[i];
00367         thee->bulkIonicStrength += (0.5*ionConc[i]*VSQR(ionQ[i
00368     ]));
00369     }
00370     netCharge += (ionConc[i]*ionQ[i]);
00371
00372 #ifndef VAPBSQUIET
00373     Vnm_print(1, " Vpbe_ctor: Using max ion radius (%g A) for exclusion \
00374 function\n", thee->maxIonRadius);
00375 #endif
00376     if (VABS(netCharge) > VSMALL) {
00377         Vnm_print(2, "Vpbe_ctor2: You have a counterion charge imbalance!\n");
00378         Vnm_print(2, "Vpbe_ctor2: Net charge conc. = %g M\n", netCharge);
00379     }
00380     return 0;

```

```

00374     }
00375     thee->T = T;
00376     thee->soluteDiel = soluteDiel;
00377     thee->solventDiel = solventDiel;
00378     thee->solventRadius = solventRadius;
00379
00380     /* Compute parameters:
00381      *
00382      * kappa^2 = (8 pi N_A e_c^2) I_s / (1000 eps_w k_B T)
00383      * kappa = 0.325567 * I_s^{1/2} angstroms^{-1}
00384      * deblen = 1 / kappa
00385      *         = 3.071564378 * I_s^{1/2} angstroms
00386      * \bar{kappa}^2 = eps_w * kappa^2
00387      * zmagic = (4 * pi * e_c^2) / (k_B T) (we scale the diagonal later)
00388      *         = 7046.528838
00389      */
00390     if (thee->T == 0.0) {
00391         Vnm_print(2, "Vpbe_ctor2: You set the temperature to 0 K.\n");
00392         Vnm_print(2, "Vpbe_ctor2: That violates the 3rd Law of Thermo.!");
00393         return 0;
00394     }
00395     if (thee->bulkIonicStrength == 0.) {
00396         thee->xkappa = 0.;
00397         thee->deblen = 0.;
00398         thee->zkappa2 = 0.;
00399     } else {
00400         thee->xkappa = VSQRT( thee->bulkIonicStrength *
00401             1.0e-16 *
00402             ((8.0 * pi * N_A * e_c*e_c) /
00403              (1000.0 * thee->solventDiel * k_B * T))
00404         );
00405         thee->deblen = 1. / thee->xkappa;
00406         thee->zkappa2 = thee->solventDiel * VSQR(thee->xkappa
00407     );
00408     }
00409     Vnm_print(0, "Vpbe_ctor2: bulk ionic strength = %g\n",
00410             thee->bulkIonicStrength);
00411     Vnm_print(0, "Vpbe_ctor2: xkappa = %g\n", thee->xkappa);
00412     Vnm_print(0, "Vpbe_ctor2: Debye length = %g\n", thee->deblen);
00413     Vnm_print(0, "Vpbe_ctor2: zkappa2 = %g\n", thee->zkappa2);
00414     thee->zmagic = ((4.0 * pi * e_c*e_c) / (k_B * thee->T)) * 1.0e+8;
00415     Vnm_print(0, "Vpbe_ctor2: zmagic = %g\n", thee->zmagic);
00416
00417     /* Compute accessibility objects:
00418      * - Allow for extra room in the case of spline windowing
00419      * - Place some limits on the size of the hash table in the case of very
00420      *   large molecules
00421      */
00422     if (thee->maxIonRadius > thee->solventRadius)
00423         radius = thee->maxIonRadius + MAX_SPLINE_WINDOW;
00424     else radius = thee->solventRadius + MAX_SPLINE_WINDOW;
00425
00426     nhash[0] = (thee->soluteXlen)/0.5;
00427     nhash[1] = (thee->soluteYlen)/0.5;
00428     nhash[2] = (thee->soluteZlen)/0.5;
00429     for (i=0; i<3; i++) inhash[i] = (int)(nhash[i]);
00430
00431     for (i=0;i<3;i++){
00432         if (inhash[i] < 3) inhash[i] = 3;
00433         if (inhash[i] > MAX_HASH_DIM) inhash[i] = MAX_HASH_DIM;
00434     }
00435     Vnm_print(0, "Vpbe_ctor2: Constructing Vclist with %d x %d x %d table\n",
00436             inhash[0], inhash[1], inhash[2]);
00437
00438     thee->clist = Vclist_ctor(thee->alist, radius, inhash,
00439                               CLIST_AUTO_DOMAIN, lower_corner, upper_corner);
00440
00441     VASSERT(thee->clist != VNNULL);
00442     thee->acc = Vacc_ctor(thee->alist, thee->clist, sdens
00443 );
00444
00445     VASSERT(thee->acc != VNNULL);
00446
00447     /* SMPBE Added */
00448     thee->smsize = 0.0;
00449     thee->smvolume = 0.0;
00450     thee->ipkey = 0;
00451
00452     thee->paramFlag = 1;
00453

```

```

00451 /*-----*/
00452 /* added by Michael Grabe */
00453 /*-----*/
00454
00455     thee->z_mem = z_mem;
00456     thee->L = L;
00457     thee->membraneDiel = membraneDiel;
00458     thee->V = V;
00459
00460 //    if (V != VNULL) thee->param2Flag = 1;
00461 //    else thee->param2Flag = 0;
00462
00463 /*-----*/
00464
00465     return 1;
00466 }
00467
00468 VPUBLIC void Vpbe_dtor(Vpbe **thee) {
00469     if ((*thee) != VNULL) {
00470         Vpbe_dtor2(*thee);
00471         Vmem_free(VNULL, 1, sizeof(Vpbe), (void **)thee);
00472         (*thee) = VNULL;
00473     }
00474 }
00475
00476 VPUBLIC void Vpbe_dtor2(Vpbe *thee) {
00477     Vclist_dtor(&(thee->clist));
00478     Vacc_dtor(&(thee->acc));
00479     Vmem_dtor(&(thee->vmem));
00480 }
00481
00482 VPUBLIC double Vpbe_getCoulombEnergy1(Vpbe *thee) {
00483
00484     int i, j, k, natoms;
00485
00486     double dist, *ipos, *jpos, icharge, jcharge;
00487     double energy = 0.0;
00488     double eps, T;
00489     Vatom *iatom, *jatom;
00490     Valist *alist;
00491
00492     VASSERT(thee != VNULL);
00493     alist = Vpbe_getValist(thee);
00494     VASSERT(alist != VNULL);
00495     natoms = Valist_getNumberAtoms(alist);
00496
00497     /* Do the sum */
00498     for (i=0; i<natoms; i++) {
00499         iatom = Valist_getAtom(alist,i);
00500         icharge = Vatom_getCharge(iatom);
00501         ipos = VatomGetPosition(iatom);
00502         for (j=i+1; j<natoms; j++) {
00503             jatom = Valist_getAtom(alist,j);
00504             jcharge = Vatom_getCharge(jatom);
00505             jpos = VatomGetPosition(jatom);
00506             dist = 0;
00507             for (k=0; k<3; k++) dist += ((ipos[k]-jpos[k])*(ipos[k]-jpos[k]));
00508             dist = VSQRT(dist);
00509             energy = energy + icharge*jcharge/dist;
00510         }
00511     }
00512
00513     /* Convert the result to J */
00514     T = Vpbe_getTemperature(thee);
00515     eps = Vpbe_getSoluteDiel(thee);
00516     energy = energy*Vunit_ec*Vunit_ec/(4*Vunit_pi*
00517     Vunit_eps0*eps*(1.0e-10));
00518
00519     /* Scale by Boltzmann energy */
00520     energy = energy/(Vunit_kb*T);
00521
00522     return energy;
00523 }
00524
00525 VPUBLIC unsigned long int Vpbe_memChk(Vpbe *thee) {
00526
00527     unsigned long int memUse = 0;
00528
00529     if (thee == VNULL) return 0;
00530
00531     memUse = memUse + sizeof(Vpbe);

```

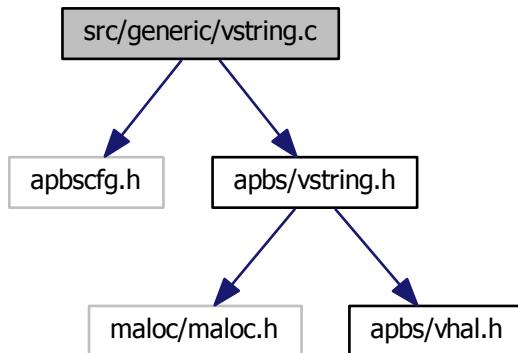
```

00531     memUse = memUse + (unsigned long int)Vacc_memChk(thee->acc);
00532
00533     return memUse;
00534 }
00535
00536 VPUBLIC int Vpbe_getIons(Vpbe *thee, int *nion, double ionConc
00537 [MAXION],
00538     double ionRadii[MAXION], double ionQ[MAXION]) {
00539     int i;
00540
00541     VASSERT(thee != VNULL);
00542
00543     *nion = thee->numIon;
00544     for (i=0; i<(*nion); i++) {
00545         ionConc[i] = thee->ionConc[i];
00546         ionRadii[i] = thee->ionRadii[i];
00547         ionQ[i] = thee->ionQ[i];
00548     }
00549
00550     return *nion;
00551 }
```

9.79 src/generic/vstring.c File Reference

Class Vstring methods.

```
#include "apbscfg.h"
#include "apbs/vstring.h"
Include dependency graph for vstring.c:
```



Functions

- VPUBLIC int [Vstring_strcasecmp](#) (const char *s1, const char *s2)
Case-insensitive string comparison (BSD standard)
- VPUBLIC int [Vstring_isdigit](#) (const char *tok)
A modified sscanf that examines the complete string.
- VEXTERNC char * [Vstring_wrappedtext](#) (const char *str, int right_margin, int left_padding)

9.79.1 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](#).

9.80 vstring.c

```

00001
00056 #include "apbscfg.h"
00057 #include "apbs/vstring.h"
00058
00059 /* /////////////////////////////////
00060 // Routine: Vstring_strcasecmp
00061 //
00062 // Copyright (c) 1988-1993 The Regents of the University of
00063 // California.
00064 // Copyright (c) 1995-1996 Sun Microsystems, Inc.
00065 VPUBLIC int Vstring_strcasecmp(const char *s1, const char *s2) {
00066
00068 #if !defined(HAVE_STRCASECMP)
00069     unsigned char charmap[] = {
00070         0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
00071         0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
00072         0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
00073         0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f,
00074         0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
00075         0x28, 0x29, 0x2a, 0x2b, 0x2c, 0x2d, 0x2e, 0x2f,
00076         0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
00077         0x38, 0x39, 0x3a, 0x3b, 0x3c, 0x3d, 0x3e, 0x3f,
00078         0x40, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67,
00079         0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f,
00080         0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77,
00081         0x78, 0x79, 0x7a, 0x5b, 0x5c, 0x5d, 0x5e, 0x5f,
00082         0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67,
00083         0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f,
00084         0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77,
00085         0x78, 0x79, 0x7a, 0x7b, 0x7c, 0x7d, 0x7e, 0x7f,
00086         0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
00087         0x88, 0x89, 0x8a, 0x8b, 0x8c, 0x8d, 0x8e, 0x8f,
00088         0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97,
00089         0x98, 0x99, 0x9a, 0x9b, 0x9c, 0x9d, 0x9e, 0x9f,
00090         0xa0, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7,
00091         0xa8, 0xa9, 0xaa, 0xab, 0xac, 0xad, 0xae, 0xaf,
00092         0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7,
00093         0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0xbd, 0xbe, 0xbf,
00094         0xc0, 0xe1, 0xe2, 0xe3, 0xe4, 0xc5, 0xe6, 0xe7,
00095         0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef,
00096         0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
00097         0xf8, 0xf9, 0xfa, 0xdb, 0xdc, 0xdd, 0xde, 0xdf,
00098         0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7,
00099         0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef,
00100         0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
00101         0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff,
00102     };
00103
00104     unsigned char u1, u2;
00105
00106     for ( ; s1++, s2++) {
00107     u1 = (unsigned char) *s1;
00108     u2 = (unsigned char) *s2;
00109     if ((u1 == '\0') || (charmap[u1] != charmap[u2])) {
00110         break;
00111     }
00112     }
00113     return charmap[u1] - charmap[u2];
00114
00115 #else
00116
00117     return strcasecmp(s1, s2);
00118
00119 #endif
00120
00121 }
00122
00123 /* /////////////////////////////////
00124 // Routine: Vstring_isdigit
00125 //
00126 // Improves upon sscanf to see if a token is an int or not
00127 //
00128 // Returns isdigit: 1 if a digit, 0 otherwise
00129 VPUBLIC int Vstring_isdigit(const char *tok) {
00130     int i, isdigit, ti;
00131     char checkchar[1];
00132     char name[VMAX_BUFSIZE];
00133     strcpy(name,tok);
00134

```

```

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
00155 char* Vstring_wrappedtext(const char* str, int right_margin, int left_padding)
00156 {
00157     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     }
00188
00189
00190
00191         if( i + span >= len )
00192         {
00193             hyphenate = 0;
00194             line_len = len - i;
00195         }
00196         else
00197         {
00198             j = span;
00199             do
00200             {
00201                 if( str[ i + j ] == ' ' )
00202                 {
00203                     hyphenate = 0;
00204                     line_len = j;
00205                     break;
00206                 }
00207                 else if( j == 0 )
00208                 {
00209                     hyphenate = 1;
00210                     line_len = span - 1;
00211                     break;
00212                 }
00213                 else
00214                 {
00215                     j--;
00216                 }
00217             } while( 1 );
00218         }
00219
00220         memset( wrap_str + k, ' ', left_padding * sizeof( char ) );
00221         k += left_padding;
00222

```

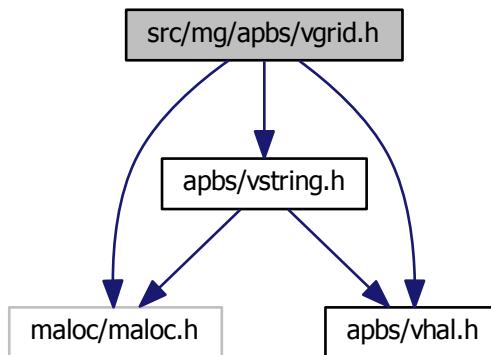
```

00223     memcpy( wrap_str + k, str + i, line_len * sizeof( char ) );
00224     k += line_len;
00225     i += line_len;
00226
00227     if( hyphenate )
00228         wrap_str[k++] = '-';
00229
00230     wrap_str[k++] = '\n';
00231
00232     wrap_str[k] = '\0';
00233 }
00234 } while( i < len );
00235
00236 return wrap_str;
00237 }
```

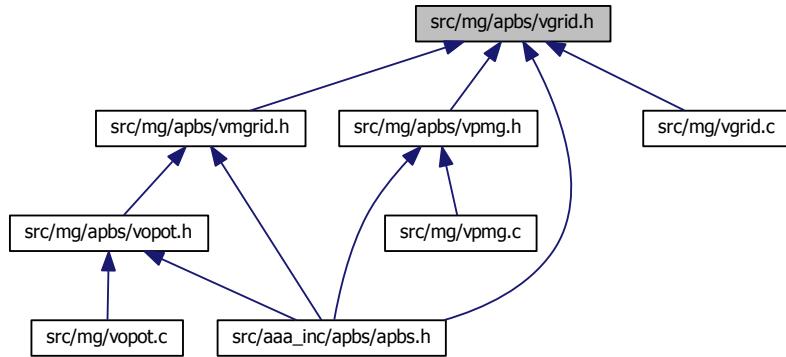
9.81 src/mg/apbs/vgrid.h File Reference

Potential oracle for Cartesian mesh data.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/vstring.h"
Include dependency graph for vgrid.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `sVgrid`

Electrostatic potential oracle for Cartesian mesh data.

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)`
FORTRAN stub object destructor.
- VEXTERNC int `Vgrid_curvature (Vgrid *thee, double pt[3], int cflag, double *curv)`
Get second derivative values at a point.
- VEXTERNC int `Vgrid_gradient (Vgrid *thee, double pt[3], double grad[3])`
Get first derivative values at a point.
- VEXTERNC int `Vgrid_readGZ (Vgrid *thee, const char *fname)`
Read in OpenDX data in GZIP format.
- VEXTERNC void `Vgrid_writeGZ (Vgrid *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, char *title, double *pvec)`
Write out OpenDX data in GZIP format.
- VEXTERNC void `Vgrid_writeUHBD (Vgrid *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, char *title, double *pvec)`
Write out the data in UHBD grid format.
- VEXTERNC void `Vgrid_writeDX (Vgrid *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, char *title, double *pvec)`
Write out the data in OpenDX grid format.
- VEXTERNC int `Vgrid_readDX (Vgrid *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname)`
Read in data in OpenDX grid format.
- VEXTERNC double `Vgrid_integrate (Vgrid *thee)`
Get the integral of the data.
- VEXTERNC double `Vgrid_normL1 (Vgrid *thee)`
Get the L_1 norm of the data. This returns the integral:

$$\|u\|_{L_1} = \int_{\Omega} |u(x)| dx$$

- VEXTERNC double `Vgrid_normL2 (Vgrid *thee)`

Get the L_2 norm of the data. This returns the integral:

$$\|u\|_{L_2} = \left(\int_{\Omega} |u(x)|^2 dx \right)^{1/2}$$

- VEXTERNC double `Vgrid_normLinf (Vgrid *thee)`

Get the L_∞ norm of the data. This returns the integral:

$$\|u\|_{L_\infty} = \sup_{x \in \Omega} |u(x)|$$

- VEXTERNC double `Vgrid_seminormH1 (Vgrid *thee)`

Get the H_1 semi-norm of the data. This returns the integral:

$$\|u\|_{H_1} = \left(\int_{\Omega} |\nabla u(x)|^2 dx \right)^{1/2}$$

- VEXTERNC double `Vgrid_normH1 (Vgrid *thee)`

Get the H_1 norm (or energy norm) of the data. This returns the integral:

$$\|u\|_{H_1} = \left(\int_{\Omega} |\nabla u(x)|^2 dx + \int_{\Omega} |u(x)|^2 dx \right)^{1/2}$$

9.81.1 Detailed Description

Potential oracle for Cartesian mesh data.

Author

Nathan Baker and Steve Bond

Version

Id:

[vgrid.h](#) 1667 2011-12-02 23:22:02Z pcells

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2012 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](#).

9.81.2 Function Documentation

9.81.2.1 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 808 of file [vgrid.c](#).

9.82 vgrid.h

```

00001
00062 #ifndef _VGRID_H_
00063 #define _VGRID_H_
00064
00065 #include "malloc/malloc.h"
00066 #include "apbs/vhal.h"
00067 #include "apbs/vstring.h"
00068
00069
00072 #define VGRID_DIGITS 6
00073
00079 struct sVgrid {
00080
00081     int nx;
00082     int ny;
00083     int nz;
00084     double hx;
00085     double hy;
00086     double hzed;
00087     double xmin;
00088     double ymin;
00089     double zmin;
00090     double xmax;
00091     double ymax;
00092     double zmax;
00093     double *data;
00094     int readdata;
00095     int ctordata;
00097     Vmem *mem;
00098 };
00099
00104 typedef struct sVgrid Vgrid;
00105
00106 #if !defined(VINLINE_VGRID)
00107
00115     VEXTERNC unsigned long int Vgrid_memChk(Vgrid *thee);
00116
00117 #else /* if defined(VINLINE_VGRID) */
00118
00126 # define Vgrid_memChk(thee) (Vmem_bytes((thee)->vmem))
00127

```

```

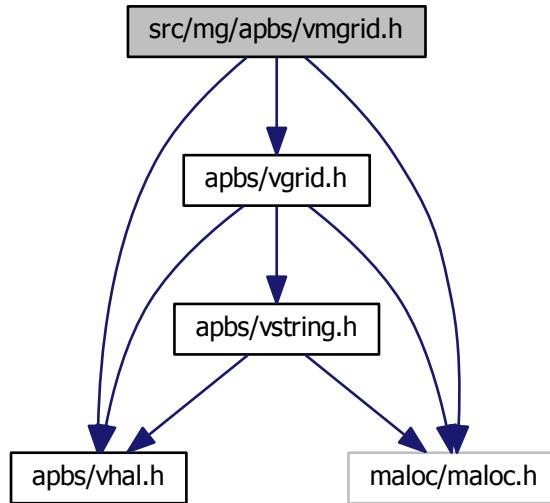
00128 #endif /* if !defined(VINLINE_VPMG) */
00129
00147 VEXTERNC Vgrid* Vgrid_ctor(int nx, int ny, int nz,
00148     double hx, double hy, double hzed,
00149     double xmin, double ymin, double zmin,
00150     double *data);
00151
00170 VEXTERNC int Vgrid_ctor2(Vgrid *thee, int nx, int ny, int
00171     nz,
00172     double hx, double hy, double hzed,
00173     double xmin, double ymin, double zmin,
00174     double *data);
00175
00183 VEXTERNC int Vgrid_value(Vgrid *thee, double x[3], double *
00184     value);
00185
00190 VEXTERNC void Vgrid_dtor(Vgrid **thee);
00191
00197 VEXTERNC void Vgrid_dtor2(Vgrid *thee);
00198
00212 VEXTERNC int Vgrid_curvature(Vgrid *thee, double pt[3], int
00213     cflag,
00214     double *curv);
00215
00223 VEXTERNC int Vgrid_gradient(Vgrid *thee, double pt[3],
00224     double grad[3] );
00225
00229 VEXTERNC int Vgrid_readGZ(
00230     Vgrid *thee,
00231     const char *fname
00232 );
00233
00237 VEXTERNC void Vgrid_writeGZ(
00238     Vgrid *thee,
00239     const char *iodev,
00240     const char *iofmt,
00241     const char *thost,
00242     const char *fname,
00243     char *title,
00244     double *pvec
00245 );
00246
00264 VEXTERNC void Vgrid_writeUHBD(Vgrid *thee, const char *
00265     iodev,
00266     const char *iofmt, const char *thost, const char *fname, char *title,
00267     double *pvec);
00268
00282 VEXTERNC void Vgrid_writeDX(Vgrid *thee, const char *iodev,
00283     const char *iofmt, const char *thost, const char *fname, char *title,
00284     double *pvec);
00285
00297 VEXTERNC int Vgrid_readDX(Vgrid *thee, const char *iodev,
00298     const char *iofmt,
00299     const char *thost, const char *fname);
00300
00306 VEXTERNC double Vgrid_integrate(Vgrid *thee);
00307
00316 VEXTERNC double Vgrid_normL1(Vgrid *thee);
00317
00326 VEXTERNC double Vgrid_normL2(Vgrid *thee);
00327
00336 VEXTERNC double Vgrid_normLinf(Vgrid *thee);
00337
00347 VEXTERNC double Vgrid_seminormH1(Vgrid *thee);
00348
00359 VEXTERNC double Vgrid_normH1(Vgrid *thee);
00360
00361 #endif

```

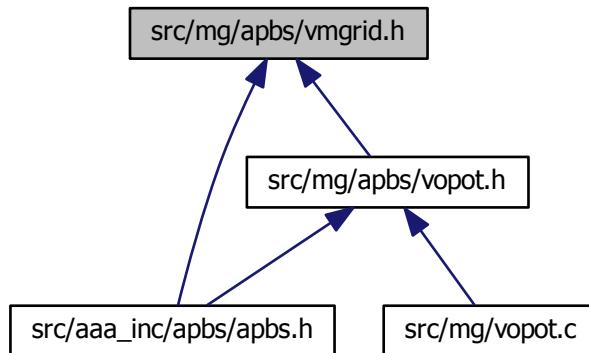
9.83 src/mg/apbs/vmgrid.h File Reference

Multiresolution oracle for Cartesian mesh data.

```
#include "maloc/malloc.h"
#include "apbs/vhal.h"
#include "apbs/vgrid.h"
Include dependency graph for vmgrid.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `sVmgrid`

Multiresolution oracle for Cartesian mesh data.

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 Vgmrid structure.

Functions

- `VEXTERNC Vmgrid * Vmgrid_ctor ()`

Construct Vmgrid object.

- `VEXTERNC int Vmgrid_ctor2 (Vmgrid *thee)`

Initialize Vmgrid object.

- `VEXTERNC int Vmgrid_value (Vmgrid *thee, double x[3], double *value)`

Get potential value (from mesh or approximation) at a point.

- `VEXTERNC void Vmgrid_dtor (Vmgrid **thee)`

Object destructor.

- `VEXTERNC void Vmgrid_dtor2 (Vmgrid *thee)`

FORTRAN stub object destructor.

- `VEXTERNC int Vmgrid_addGrid (Vmgrid *thee, Vgrid *grid)`

Add a grid to the hierarchy.

- `VEXTERNC int Vmgrid_curvature (Vmgrid *thee, double pt[3], int cflag, double *curv)`

Get second derivative values at a point.

- `VEXTERNC int Vmgrid_gradient (Vmgrid *thee, double pt[3], double grad[3])`

Get first derivative values at a point.

- `VEXTERNC Vgrid * Vmgrid_getGridByNum (Vmgrid *thee, int num)`

Get specific grid in hierarchy.

- `VEXTERNC Vgrid * Vmgrid_getGridByPoint (Vmgrid *thee, double pt[3])`

Get grid in hierarchy which contains specified point or VNULL.

9.83.1 Detailed Description

Multiresolution oracle for Cartesian mesh data.

Author

Nathan Baker

Version

Id:

[vmgrid.h](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2012 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](#).

9.84 vmgrid.h

```
00001  
00062 #ifndef _VMGRID_H_  
00063 #define _VMGRID_H_
```

```

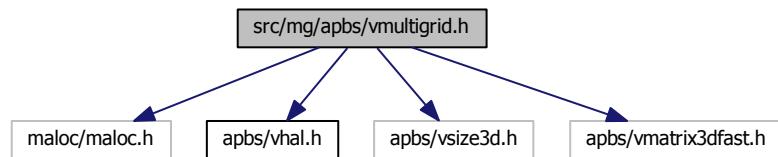
00064
00065 /* Generic headers */
00066 #include "maloc/maloc.h"
00067 #include "apbs/vhal.h"
00068
00069 /* Headers specific to this file */
00070 #include "apbs/vgrid.h"
00071
00072 #define VMGRIDMAX 20
00073
00074
00075 struct sVmgrid {
00076     int ngrids;
00077     Vgrid *grids[VMGRIDMAX];
00078 };
00079
00080 typedef struct sVmgrid Vmgrid;
00081
00082 VEXTERNC Vmgrid* Vmgrid_ctor();
00083
00084 VEXTERNC int Vmgrid_ctor2(Vmgrid *thee);
00085
00086 VEXTERNC int Vmgrid_value(Vmgrid *thee, double x[3], double *
00087     value);
00088
00089 VEXTERNC void Vmgrid_dtor(Vmgrid **thee);
00090
00091 VEXTERNC void Vmgrid_dtor2(Vmgrid *thee);
00092
00093 VEXTERNC int Vmgrid_addGrid(Vmgrid *thee, Vgrid *grid)
00094 ;
00095
00096 VEXTERNC int Vmgrid_curvature(Vmgrid *thee, double pt[3],
00097     int cflag,
00098     double *curv);
00099
00100 VEXTERNC int Vmgrid_gradient(Vmgrid *thee, double pt[3],
00101     double grad[3]);
00102
00103 VEXTERNC Vgrid* Vmgrid_getGridByNum(Vmgrid *thee,
00104     int num);
00105
00106 VEXTERNC Vgrid* Vmgrid_getGridByPoint(Vmgrid *
00107     thee, double pt[3]);
00108
00109 #endif
00110
00111

```

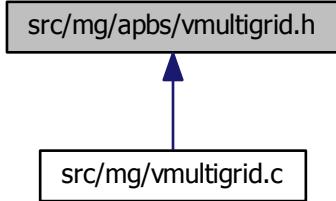
9.85 src/mg/apbs/vmultigrid.h File Reference

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/vsize3d.h"
#include "apbs/vmatrix3dfast.h"

Include dependency graph for vmultigrid.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [sVmultigrid](#)

Declaration of Vmultigrid class.

Typedefs

- typedef enum [eVmultipgrid_method](#) **Vmultigrid_method**
- typedef enum [eVmultipgrid_boundary_method](#) **Vmultigrid_boundary_method**
- typedef enum [eVmultipgrid_prolong_method](#) **Vmultigrid_prolong_method**
- typedef enum [eVmultipgrid_coarsen_method](#) **Vmultigrid_coarsen_method**
- typedef enum [eVmultipgrid_discrete_method](#) **Vmultigrid_discrete_method**
- typedef enum [eVmultipgrid_smooth_method](#) **Vmultigrid_smooth_method**
- typedef enum [eVmultipgrid_coarse_solver](#) **Vmultigrid_coarse_solver**
- typedef enum [eVmultipgrid_iterate_method](#) **Vmultigrid_iterate_method**
- typedef enum [eVmultipgrid_verbosity](#) **Vmultigrid_verbosity**
- typedef enum [eVmultipgrid_stop_criterion](#) **Vmultigrid_stop_criterion**
- typedef enum [eVmultipgrid_linear](#) **Vmultigrid_linear**
- typedef enum [eVmultipgrid_operator_analysis](#) **Vmultigrid_operator_analysis**
- typedef struct [sVmultigrid](#) **Vmultigrid**

Declaration of the Vmultigrid class as the Vmultigrid structure.

Enumerations

- enum `eVmultigrid_method` {

VMULTIGRID_CGMG, VMULTIGRID_NEW, VMULTIGRID_MG, VMULTIGRID(CG,
VMULTIGRID_SOR, VMULTIGRID_RBGS, VMULTIGRID_WJ, VMULTIGRID_RICH,
VMULTIGRID_CGMG_AQUA, VMULTIGRID_NEW_AQUA }

Defines the method to be used for the computation.
- enum `eVmultigrid_boundary_method` {

VMULTIGRID_BOUNDARY_ZERO, VMULTIGRID_BOUNDARY_SDH, VMULTIGRID_BOUNDARY_MDH, VM-
ULTIGRID_BOUNDARY_FOCUS,
VMULTIGRID_BOUNDARY_MEM, VMULTIGRID_BOUNDARY_MAP, VMULTIGRID_BOUNDARY_UNUSED

}

Defines the boundary condition method.
- enum `eVmultigrid_prolong_method` { VMULTIGRID_PROLONG_TRI_LINEAR, VMULTIGRID_PROLONG_OPE-
RATOR, VMULTIGRID_PROLONG_MODULO }

Controls the Prolongation method.
- enum `eVmultigrid_coarsen_method` { VMULTIGRID_COARSEN_STANDARD, VMULTIGRID_COARSEN_AVE-
RAGED, VMULTIGRID_COARSEN_GALERKIN }

Controls the coarsening method.
- enum `eVmultigrid_discrete_method` { VMULTIGRID_DISCRETE_VOLUME, VMULTIGRID_DISCRETE_ELEME-
NT }

Controls the discretization method.
- enum `eVmultigrid_smooth_method` {

VMULTIGRID_SMOOTH_JACOBIAN, VMULTIGRID_SMOOTH_GAUSS, VMULTIGRID_SMOOTH_SOR, VM-
ULTIGRID_SMOOTH_RICHARDSON,
VMULTIGRID_SMOOTH_CGHS }

Controls the smoothing method.
- enum `eVmultigrid_coarse_solver` { VMULTIGRID_SOLVERCG, VMULTIGRID_SOLVER_SYMMETRIC }

Controls the coarse grid solver.
- enum `eVmultigrid_iterate_method` { VMULTIGRID_ITERATE_VARIABLE, VMULTIGRID_ITERATE_NESTED }

Controls the multigrid iteration method.
- enum `eVmultigrid_verbosity` { VMULTIGRID_VERBOSE_NONE, VMULTIGRID_VERBOSE_SOME, VMULTIGRID_VERBOSE_MOST, VMULTIGRID_VERBOSE_ALL }

Controls run-time status message display.
- enum `eVmultigrid_stop_criterion` {

VMULTIGRID_STOP_RESIDUAL, VMULTIGRID_STOP_RELATIVE, VMULTIGRID_STOP_DIFF, VMULTIGRID_STOP_ERRC,
VMULTIGRID_STOP_ERRD, VMULTIGRID_STOP_AERRD }

Controls the stopping criterion.
- enum `eVmultigrid_linear` { VMULTIGRID_LINEAR_SIZE, VMULTIGRID_LINEAR_NORMAL, VMULTIGRID_LIN-
EAR_NON, VMULTIGRID_LINEAR_POLYNOMIAL }

Controls the use of non-linearity.
- enum `eVmultigrid_operator_analysis` { VMULTIGRID_ANALYSIS_NONE, VMULTIGRID_ANALYSIS_CONDITI-
ONAL, VMULTIGRID_ANALYSIS_SPECTRAL, VMULTIGRID_ANALYSIS_BOTH }

Controls method for analysis of the operator.

9.85.1 Detailed Description

Author

Tucker Beck

Version**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 [vmultigrid.h](#).

9.85.2 Enumeration Type Documentation

9.85.2.1 enum eVmultigrid_boundary_method

Defines the boundary condition method.

Enumerator:

VMULTIGRID_BOUNDARY_ZERO Zero Dirichlet.

VMULTIGRID_BOUNDARY_SDH Single-sphere Debye-Huckel Dirichlet.

VMULTIGRID_BOUNDARY_MDH Multiple-sphere Debye-Huckel Dirichlet.

VMULTIGRID_BOUNDARY_FOCUS Focusing Dirichlet.

VMULTIGRID_BOUNDARY_MEM Focusing membrane.

VMULTIGRID_BOUNDARY_MAP Skip first level focusing. Use external map.

VMULTIGRID_BOUNDARY_UNUSED Unused method (placeholder)

Definition at line 85 of file [vmultigrid.h](#).

9.85.2.2 enum eVmultigrid_coarse_solver

Controls the coarse grid solver.

Enumerator:

VMULTIGRID_SOLVER_CG Conjugate Gradients solver.

VMULTIGRID_SOLVER_SYMMETRIC Symmetric banded linpack solver.

Definition at line 145 of file [vmultigrid.h](#).

9.85.2.3 enum eVmultigrid_coarsen_method

Controls the coarsening method.

Enumerator:

VMULTIGRID_COARSEN_STANDARD Standard Discretization.

VMULTIGRID_COARSEN_AVERAGED Averaged Coef and Std Discretization.

VMULTIGRID_COARSEN_GALERKIN Algebraic Galerkin Coarsening.

Definition at line 111 of file [vmultigrid.h](#).

9.85.2.4 enum eVmultigrid_discrete_method

Controls the discretization method.

Enumerator:

VMULTIGRID_DISCRETE_VOLUME Finite Volume.

VMULTIGRID_DISCRETE_ELEMENT Finite Element Method.

Definition at line 122 of file [vmultigrid.h](#).

9.85.2.5 enum eVmultigrid_iterate_method

Controls the multigrid iteration method.

Enumerator:

VMULTIGRID_ITERATE_VARIABLE Variable v-cycle.

VMULTIGRID_ITERATE_NESTED Nested iteration.

Definition at line 155 of file [vmultigrid.h](#).

9.85.2.6 enum eVmultipgrid_linear

Controls the use of non-linearity.

Enumerator:

- VMULTIGRID_LINEAR_SIZE** Size-modified PBE.
- VMULTIGRID_LINEAR_NORMAL**
- VMULTIGRID_LINEAR_NON** Nonlinear PBE with capped sinh term.
- VMULTIGRID_LINEAR_POLYNOMIAL** Polynomial approximation to sinh.

Definition at line 192 of file [vmultigrid.h](#).

9.85.2.7 enum eVmultipgrid_method

Defines the method to be used for the computation.

Enumerator:

- VMULTIGRID_CGMG** Conjugate Gradient Multi-grid.
- VMULTIGRID_NEW** Newton's.
- VMULTIGRID_MG** Standard Multigrid.
- VMULTIGRID(CG** Conjugate Gradient.
- VMULTIGRID_SOR** Successive Overrelaxation.
- VMULTIGRID_RBGS** Red-black Gauss-Seidel.
- VMULTIGRID_WJ** Weighted Jacobian.
- VMULTIGRID_RICH** Richardson.
- VMULTIGRID_CGMG_AQUA** Conjugate Gradient Multigrid Aqua.
- VMULTIGRID_NEW_AQUA** Newton's Aqua.

Definition at line 67 of file [vmultigrid.h](#).

9.85.2.8 enum eVmultipgrid_operator_analysis

Controls method for analysis of the operator.

Enumerator:

- VMULTIGRID_ANALYSIS_NONE** No analysis.
- VMULTIGRID_ANALYSIS_CONDITIONAL** Conditional number analysis.
- VMULTIGRID_ANALYSIS_SPECTRAL** Spectral radius analysis.
- VMULTIGRID_ANALYSIS_BOTH** Use both previous analysis methods.

Definition at line 204 of file [vmultigrid.h](#).

9.85.2.9 enum eVmultigrid_prolong_method

Controls the Prolongation method.

Enumerator:

- VMULTIGRID_PROLONG_TRILINEAR** Trilinear prolongation.
- VMULTIGRID_PROLONG_OPERATOR** Operator based prolongation.
- VMULTIGRID_PROLONG_MODULO** Modulo operator based prolongation.

Definition at line 100 of file [vmultigrid.h](#).

9.85.2.10 enum eVmultigrid_smooth_method

Controls the smoothing method.

Enumerator:

- VMULTIGRID_SMOOTH_JACOBIAN** Weighted Jacobian method.
- VMULTIGRID_SMOOTH_GAUSS** Gauss-seidel method.
- VMULTIGRID_SMOOTH_SOR** Successive overrelaxtion.
- VMULTIGRID_SMOOTH_RICHARDSON** Richardson's method.
- VMULTIGRID_SMOOTH_CGHS**

Definition at line 132 of file [vmultigrid.h](#).

9.85.2.11 enum eVmultigrid_stop_criterion

Controls the stopping criterion.

Enumerator:

- VMULTIGRID_STOP_RESIDUAL** Residual.
- VMULTIGRID_STOP_RELATIVE** Relative Residual.
- VMULTIGRID_STOP_DIFF**
- VMULTIGRID_STOP_ERRC**
- VMULTIGRID_STOP_ERRD**
- VMULTIGRID_STOP_AERRD**

Definition at line 178 of file [vmultigrid.h](#).

9.85.2.12 enum eVmultigrid_verbosity

Controls run-time status message display.

Enumerator:

- VMULTIGRID_VERBOSE_NONE** Print no status messages.
- VMULTIGRID_VERBOSE_SOME** Print some status messages.
- VMULTIGRID_VERBOSE_MOST** Print most status messages.
- VMULTIGRID_VERBOSE_ALL** Print all status messages.

Definition at line 165 of file [vmultigrid.h](#).

9.86 vmultigrid.h

```

00001
00055 #ifndef _VMULTIGRID_H_
00056 #define _VMULTIGRID_H_
00057
00058 #include "maloc/maloc.h"
00059
00060 #include "apbs/vhal.h"
00061 #include "apbs/vsize3d.h"
00062 #include "apbs/vmatrix3dfast.h"
00063
00064
00065
00067 enum eVmultigrid_method {
00068     VMULTIGRID_CGMG,
00069     VMULTIGRID_NEW,
00070     VMULTIGRID_MG,
00071     VMULTIGRID(CG,
00072     VMULTIGRID_SOR,
00073     VMULTIGRID_RBGS,
00074     VMULTIGRID_WJ,
00075     VMULTIGRID_RICH,
00076     VMULTIGRID_CGMG_AQUA,
00077     VMULTIGRID_NEW_AQUA
00078 };
00079
00080 typedef enum eVmultigrid_method Vmultigrid_method;
00081
00082
00083
00085 enum eVmultigrid_boundary_method {
00086     VMULTIGRID_BOUNDARY_ZERO,
00087     VMULTIGRID_BOUNDARY_SDH,
00088     VMULTIGRID_BOUNDARY_MDH,
00089     VMULTIGRID_BOUNDARY_FOCUS,
00090     VMULTIGRID_BOUNDARY_MEM,
00091     VMULTIGRID_BOUNDARY_MAP,
00092     VMULTIGRID_BOUNDARY_UNUSED
00093 };
00094
00095 typedef enum eVmultigrid_boundary_method
    Vmultigrid_boundary_method;
00096
00097
00098
00100 enum eVmultigrid_prolong_method {
00101     VMULTIGRID_PROLONG_TRILINEAR,
00102     VMULTIGRID_PROLONG_OPERATOR,
00103     VMULTIGRID_PROLONG_MODULO,
00104 };
00105
00106 typedef enum eVmultigrid_prolong_method
    Vmultigrid_prolong_method;
00107
00108
00109
00111 enum eVmultigrid_coarsen_method {
00112     VMULTIGRID_COARSEN_STANDARD,
00113     VMULTIGRID_COARSEN_AVERAGED,
00114     VMULTIGRID_COARSEN_GALERKIN
00115 };
00116
00117 typedef enum eVmultigrid_coarsen_method
    Vmultigrid_coarsen_method;
00118
00119
00120
00122 enum eVmultigrid_discrete_method {
00123     VMULTIGRID_DISCRETE_VOLUME,
00124     VMULTIGRID_DISCRETE_ELEMENT
00125 };
00126
00127 typedef enum eVmultigrid_discrete_method
    Vmultigrid_discrete_method;
00128
00129
00130
00132 enum eVmultigrid_smooth_method {
00133     VMULTIGRID_SMOOTH_JACOBIAN,

```

```
00134     VMULTIGRID_SMOOTH_GAUSS,
00135     VMULTIGRID_SMOOTHSOR,
00136     VMULTIGRID_SMOOTH_RICHARDSON,
00137     VMULTIGRID_SMOOTH_CGHS
00138 };
00139
00140 typedef enum eVmultipgrid_smooth_method
00141     Vmultipgrid_smooth_method;
00142
00143
00144 enum eVmultipgrid_coarse_solver {
00145     VMULTIGRID_SOLVERCG,
00146     VMULTIGRID_SOLVER_SYMMETRIC
00147 };
00148
00149
00150 typedef enum eVmultipgrid_coarse_solver
00151     Vmultipgrid_coarse_solver;
00152
00153
00154 enum eVmultipgrid_iterate_method {
00155     VMULTIGRID_ITERATE_VARIABLE,
00156     VMULTIGRID_ITERATE_NESTED,
00157 };
00158
00159
00160 typedef enum eVmultipgrid_iterate_method
00161     Vmultipgrid_iterate_method;
00162
00163
00164 enum eVmultipgrid_verbosity {
00165     VMULTIGRID_VERBOSE_NONE,
00166     VMULTIGRID_VERBOSE_SOME,
00167     VMULTIGRID_VERBOSE_MOST,
00168     VMULTIGRID_VERBOSE_ALL,
00169 };
00170
00171
00172 typedef enum eVmultipgrid_verbosity Vmultipgrid_verbosity;
00173
00174
00175
00176
00177 enum eVmultipgrid_stop_criterion {
00178     VMULTIGRID_STOP_RESIDUAL,
00179     VMULTIGRID_STOP_RELATIVE,
00180     VMULTIGRID_STOP_DIFF,
00181     VMULTIGRID_STOP_ERRC,
00182     VMULTIGRID_STOP_ERRD,
00183     VMULTIGRID_STOP_AERRD,
00184 };
00185
00186
00187 typedef enum eVmultipgrid_stop_criterion
00188     Vmultipgrid_stop_criterion;
00189
00190
00191 enum eVmultipgrid_linear {
00192     VMULTIGRID_LINEAR_SIZE,
00193     VMULTIGRID_LINEAR_NORMAL,
00194     VMULTIGRID_LINEAR_NON,
00195     VMULTIGRID_LINEAR_POLYNOMIAL
00196 };
00197
00198
00199 typedef enum eVmultipgrid_linear Vmultipgrid_linear;
00200
00201
00202
00203 enum eVmultipgrid_operator_analysis {
00204     VMULTIGRID_ANALYSIS_NONE,
00205     VMULTIGRID_ANALYSIS_CONDITIONAL,
00206     VMULTIGRID_ANALYSIS_SPECTRAL,
00207     VMULTIGRID_ANALYSIS_BOTH
00208 };
00209
00210
00211 typedef enum eVmultipgrid_operator_analysis
00212     Vmultipgrid_operator_analysis;
00213
00214
00220 struct svmultipgrid {
```

```
00221 // Algorithmic control flags
00222 Vmultigrid_method method;
00223
00225 Vmultigrid_boundary_method boundary_method;
00226
00228 Vmultigrid_prolong_method prolong_method;
00229
00230 Vmultigrid_coarsen_method coarsen_method;
00231
00232 Vmultigrid_discrete_method discrete_method;
00233
00234 Vmultigrid_smooth_method smooth_method;
00235
00236 Vmultigrid_coarse_solver coarse_solver;
00237
00238 Vmultigrid_iterate_method iterate_method;
00239
00240 Vmultigrid_verbose verbosity;
00241
00242 Vmultigrid_stop_criterion stop_criterion;
00243
00244 Vmultigrid_linear linear;
00245
00246 Vmultigrid_operator_analysis operator_analysis;
00247
00248
00249 // Workspace boundary totals
00250
00251 int real_workspace_limit;
00252
00253 int integer_workspace_limit;
00254
00255 int real_workspace_size;
00256
00257 int integer_workspace_size;
00258
00259
00260 // Grid dimensions
00261
00262 int x_size;
00263
00264 int y_size;
00265
00266 int z_size;
00267
00268 int level_count;
00269
00270 int fine_count;
00271
00272 int coarse_count;
00273
00274
00275 // Iteration control
00276
00277 int max_iterations;
00278
00279
00280 // Computation Matrices
00281 Vm
00282
00283
00284 // Undocumented
00285 // @todo Finish investigation, refactor, and documentation
00286
00287 int ipcon;
00288
00289 int irite;
00290
00291 int nonlin;
00292
00293
00294 }
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341 };
00342
00343
00344
00345
00346
00347 typedef struct sVm multigrid Vmultigrid;
00348
```

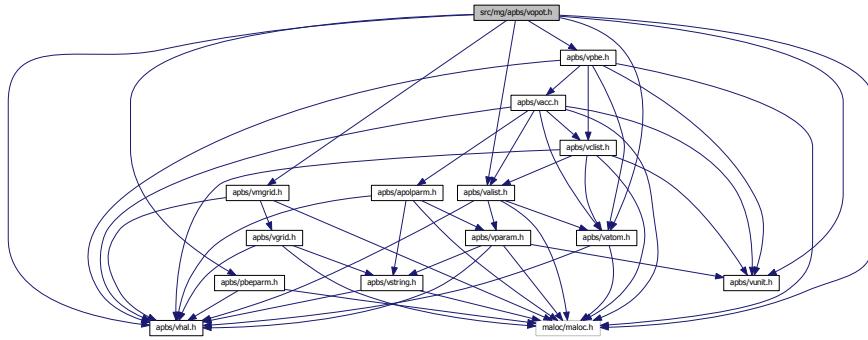
```
00349 #endif // _VMULTIGRID_H_
00350
```

9.87 src/mg/apbs/vopot.h File Reference

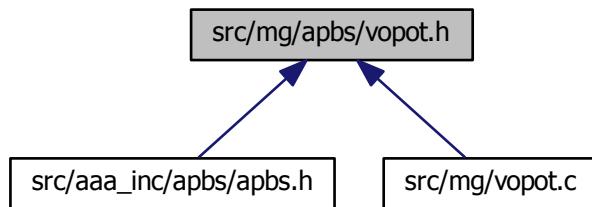
Potential oracle for Cartesian mesh data.

```
#include "malloc/malloc.h"
#include "apbs/vhal.h"
#include "apbs/vatom.h"
#include "apbs/valist.h"
#include "apbs/vmgrid.h"
#include "apbs/vunit.h"
#include "apbs/vpbe.h"
#include "apbs/pbeparm.h"
Include dependency graph for vopot.h:
```

Include dependency graph for vopot.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct `sVopot`

Electrostatic potential oracle for Cartesian mesh data.

Typedefs

- `typedef struct sVopot Vopot`

Declaration of the Vopot class as the Vopot structure.

Functions

- `VEXTERNC Vopot * Vopot_ctor (Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl)`
Construct Vopot object with values obtained from Vpmg_readDX (for example)
- `VEXTERNC int Vopot_ctor2 (Vopot *thee, Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl)`
Initialize Vopot object with values obtained from Vpmg_readDX (for example)
- `VEXTERNC int Vopot_pot (Vopot *thee, double x[3], double *pot)`
Get potential value (from mesh or approximation) at a point.
- `VEXTERNC void Vopot_dtor (Vopot **thee)`
Object destructor.
- `VEXTERNC void Vopot_dtor2 (Vopot *thee)`
FORTRAN stub object destructor.
- `VEXTERNC int Vopot_curvature (Vopot *thee, double pt[3], int cflag, double *curv)`
Get second derivative values at a point.
- `VEXTERNC int Vopot_gradient (Vopot *thee, double pt[3], double grad[3])`
Get first derivative values at a point.

9.87.1 Detailed Description

Potential oracle for Cartesian mesh data.

Author

Nathan Baker

Version

Id:

`vopot.h` 1667 2011-12-02 23:22:02Z pcellis

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 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](#).

9.88 vopot.h

```

00001
00062 #ifndef _VOPOT_H_
00063 #define _VOPOT_H_
00064
00065 /* Generic headers */
00066 #include "maloc/maloc.h"
00067 #include "apbs/vhal.h"
00068
00069 /* Specific headers */
00070 #include "apbs/vatom.h"
00071 #include "apbs/valist.h"
00072 #include "apbs/vmgrid.h"
00073 #include "apbs/vunit.h"
00074 #include "apbs/vpbe.h"
00075 #include "apbs/pbeparm.h"
00076
00082 struct sVopot {
00083
00084     Vmgrid *mggrid;
00086     Vpbe   *pbe;
00087     Vbcfl bcfl;
00089 };
00090
00095 typedef struct sVopot Vopot;
00096
00107 VEXTERNC Vopot* Vopot_ctor(Vmgrid *mggrid, Vpbe *
00108     pbe, Vbcfl bcfl);
00108
00120 VEXTERNC int Vopot_ctor2(Vopot *thee, Vmgrid *mggrid,
00121     Vpbe *pbe, Vbcfl bcfl);
00121
00130 VEXTERNC int Vopot_pot(Vopot *thee, double x[3], double *pot);
00131

```

```

00137 VEXTERNC void Vopot_dtor(Vopot **thee);
00138
00144 VEXTERNC void Vopot_dtor2(Vopot *thee);
00145
00159 VEXTERNC int Vopot_curvature(Vopot *thee, double pt[3], int
00160   cflag, double
00161   *curv);
00161
00170 VEXTERNC int Vopot_gradient(Vopot *thee, double pt[3],
00171   double grad[3] );
00171
00172
00173 #endif

```

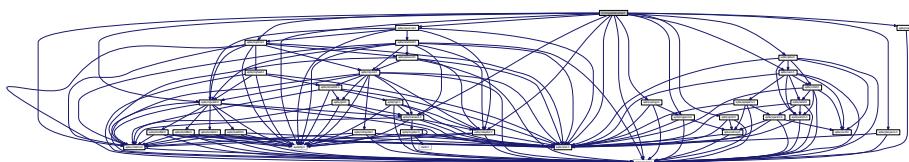
9.89 src/mg/apbs/vpmg.h File Reference

Contains declarations for class Vpmg.

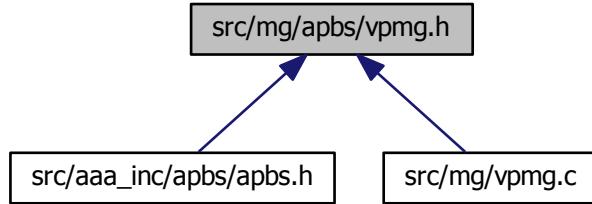
```

#include "apbscfg.h"
#include "malloc/malloc.h"
#include "apbs/vhal.h"
#include "apbs/vpmgp.h"
#include "apbs/vacc.h"
#include "apbs/vcap.h"
#include "apbs/vpbe.h"
#include "apbs/vgrid.h"
#include "apbs/mgparm.h"
#include "apbs/pbeparm.h"
#include "apbs/vmatrix.h"
#include "apbs/mgdrv.h"
#include "apbs/newdrv.h"
#include "apbs/mgsubd.h"
#include "apbs/mikpckd.h"
#include "apbs/matvecd.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)

FORTRAN stub constructor for the Vpmg class (uses previously-allocated memory)
- VEXTERNC void [Vpmg_dtor](#) (**Vpmg** **thee)

- Object destructor.*
- VEXTERNC void `Vpmg_dtor2 (Vpmg *thee)`

FORTRAN stub object destructor.
 - VEXTERNC int `Vpmg_fillco (Vpmg *thee, Vsurf_Meth surfMeth, double splineWin, Vchrg_Meth chargeMeth, int useDielXMap, Vgrid *dielXMap, int useDielYMap, Vgrid *dielYMap, int useDielZMap, Vgrid *dielZMap, int useKappaMap, Vgrid *kappaMap, int usePotMap, Vgrid *potMap, int useChargeMap, Vgrid *chargeMap)`

Fill the coefficient arrays prior to solving the equation.
 - VEXTERNC int `Vpmg_solve (Vpmg *thee)`

Solve the PBE using PMG.
 - VEXTERNC int `Vpmg_solveLaplace (Vpmg *thee)`

Solve Poisson's equation with a homogeneous Laplacian operator using the solvent dielectric constant. This solution is performed by a sine wave decomposition.
 - VEXTERNC double `Vpmg_energy (Vpmg *thee, int extFlag)`

Get the total electrostatic energy.
 - VEXTERNC double `Vpmg_qfEnergy (Vpmg *thee, int extFlag)`

Get the "fixed charge" contribution to the electrostatic energy.
 - VEXTERNC double `Vpmg_qfAtomEnergy (Vpmg *thee, Vatom *atom)`

Get the per-atom "fixed charge" contribution to the electrostatic energy.
 - VEXTERNC double `Vpmg_qmEnergy (Vpmg *thee, int extFlag)`

Get the "mobile charge" contribution to the electrostatic energy.
 - VEXTERNC double `Vpmg_dielEnergy (Vpmg *thee, int extFlag)`

Get the "polarization" contribution to the electrostatic energy.
 - VEXTERNC double `Vpmg_dielGradNorm (Vpmg *thee)`

Get the integral of the gradient of the dielectric function.
 - VEXTERNC int `Vpmg_force (Vpmg *thee, double *force, int atomID, Vsurf_Meth srfm, Vchrg_Meth chgm)`

Calculate the total force on the specified atom in units of k_B T/AA.
 - VEXTERNC int `Vpmg_qfForce (Vpmg *thee, double *force, int atomID, Vchrg_Meth chgm)`

Calculate the "charge-field" force on the specified atom in units of k_B T/AA.
 - VEXTERNC int `Vpmg_dbForce (Vpmg *thee, double *dbForce, int atomID, Vsurf_Meth srfm)`

Calculate the dielectric boundary forces on the specified atom in units of k_B T/AA.
 - VEXTERNC int `Vpmg_ibForce (Vpmg *thee, double *force, int atomID, Vsurf_Meth srfm)`

Calculate the osmotic pressure on the specified atom in units of k_B T/AA.
 - VEXTERNC void `Vpmg_setPart (Vpmg *thee, double lowerCorner[3], double upperCorner[3], int bflags[6])`

Set partition information which restricts the calculation of observables to a (rectangular) subset of the problem domain.
 - VEXTERNC void `Vpmg_unsetPart (Vpmg *thee)`

Remove partition restrictions.
 - VEXTERNC int `Vpmg_fillArray (Vpmg *thee, double *vec, Vdata_Type type, double parm, Vhal_PBEType pbeType, PBEParm *pbeparm)`

Fill the specified array with accessibility values.
 - VPUBLIC void `Vpmg_fieldSpline4 (Vpmg *thee, int atomID, double field[3])`

Computes the field at an atomic center using a stencil based on the first derivative of a 5th order B-spline.
 - VEXTERNC double `Vpmg_qfPermanentMultipoleEnergy (Vpmg *thee, int atomID)`

Computes the permanent multipole electrostatic hydration energy (the polarization component of the hydration energy currently computed in TINKER).
 - VEXTERNC void `Vpmg_qfPermanentMultipoleForce (Vpmg *thee, int atomID, double force[3], double torque[3])`

Computes the q-Phi Force for permanent multipoles based on 5th order B-splines.
 - VEXTERNC void `Vpmg_ibPermanentMultipoleForce (Vpmg *thee, int atomID, double force[3])`

Compute the ionic boundary force for permanent multipoles.

- VEXTERNC void `Vpmg_dbPermanentMultipoleForce` (`Vpmg *thee`, int atomID, double force[3])
Compute the dielectric boundary force for permanent multipoles.
- VEXTERNC void `Vpmg_qfDirectPolForce` (`Vpmg *thee`, `Vgrid *perm`, `Vgrid *induced`, int atomID, double force[3], double torque[3])
 $q\text{-}\Phi$ direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.
- VEXTERNC void `Vpmg_qfNLDirectPolForce` (`Vpmg *thee`, `Vgrid *perm`, `Vgrid *nllInduced`, int atomID, double force[3], double torque[3])
 $q\text{-}\Phi$ direct polarization force between permanent multipoles and non-local induced dipoles based on 5th Order B-Splines. Keep in mind that the "non-local" induced dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.
- VEXTERNC void `Vpmg_ibDirectPolForce` (`Vpmg *thee`, `Vgrid *perm`, `Vgrid *induced`, int atomID, double force[3])
Ionic boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.
- VEXTERNC void `Vpmg_ibNLDirectPolForce` (`Vpmg *thee`, `Vgrid *perm`, `Vgrid *nllInduced`, int atomID, double force[3])
Ionic boundary direct polarization force between permanent multipoles and non-local induced dipoles based on 5th order. Keep in mind that the "non-local" induced dipoles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.
- VEXTERNC void `Vpmg_dbDirectPolForce` (`Vpmg *thee`, `Vgrid *perm`, `Vgrid *induced`, int atomID, double force[3])
Dielectric boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.
- VEXTERNC void `Vpmg_dbNLDirectPolForce` (`Vpmg *thee`, `Vgrid *perm`, `Vgrid *nllInduced`, int atomID, double force[3])
Dielectric 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 *nllInduced`, int atomID, double force[3])
Mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.
- VEXTERNC void `Vpmg_ibMutualPolForce` (`Vpmg *thee`, `Vgrid *induced`, `Vgrid *nllInduced`, int atomID, double force[3])
Ionic boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.
- VEXTERNC void `Vpmg_dbMutualPolForce` (`Vpmg *thee`, `Vgrid *induced`, `Vgrid *nllInduced`, int atomID, double force[3])
Dielectric boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.
- VEXTERNC void `Vpmg_printColComp` (`Vpmg *thee`, char path[72], char title[72], char mxtype[3], int flag)
Print out a column-compressed sparse matrix in Harwell-Boeing format.
- VPRIATE 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.
- VPRIATE 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.
- VPRIATE void `bcolcomp3` (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *ac, double *cc, double *values, int *rowind, int *colptr, int *flag)

- **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)

Build 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 polarizable 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 srfm, **Vacc** *acc, double *gpos, double win, double inrad, **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-Hückel 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-Hückel 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 `fillcoInducedDipole` (`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)

9.89.1 Detailed Description

Contains declarations for class Vpmg.

Version

Id:

[vpmg.h](#) 1750 2012-07-18 18:34:27Z tuckerbeck

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](#).

9.89.2 Function Documentation

9.89.2.1 VPRIVATE void bcCalc (Vpmg * *thee*)

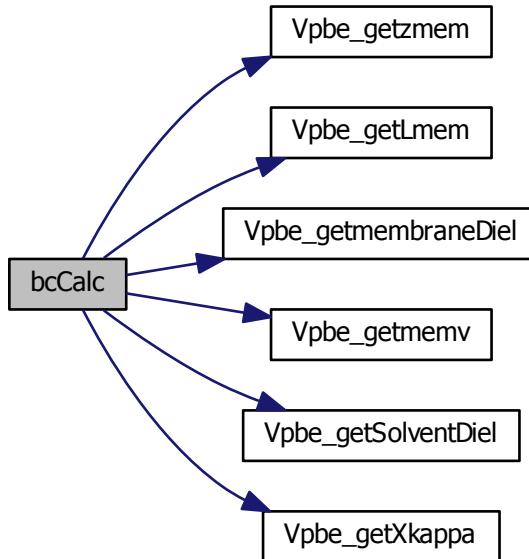
Fill boundary condition arrays.

Author

Nathan Baker

Definition at line 4370 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.89.2.2 VPRIVATE void bcf1 (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 $\text{pre1} * (\text{charge}/\text{d}) * (\exp(-\text{xkappa} * (\text{d}-\text{size})) / (1 + \text{xkappa} * \text{size}))$ to add the effect of the Debye-Hückel 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 z-direction |

Definition at line 2567 of file [vpmg.c](#).

9.89.2.3 VPRIVATE double bspline2 (double x)

Evaluate a cubic B-spline.

Author

Nathan Baker

Returns

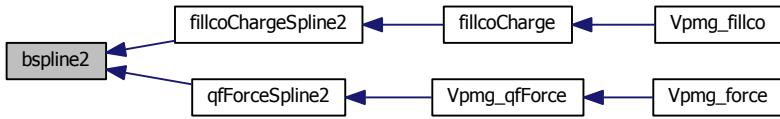
Cubic B-spline value

Parameters

| | |
|----------------|----------|
| <code>x</code> | Position |
|----------------|----------|

Definition at line 5484 of file [vpmg.c](#).

Here is the caller graph for this function:

**9.89.2.4 VPRIVATE double bspline4 (double x)**

Evaluate a 5th Order B-Spline (4th order polynomial)

Author

: Michael Schnieders

Returns

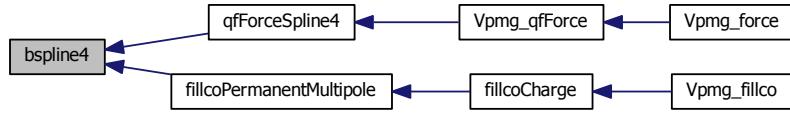
5th Order B-Spline

Parameters

| | |
|----------------|----------|
| <code>x</code> | Position |
|----------------|----------|

Definition at line 7124 of file [vpmg.c](#).

Here is the caller graph for this function:

**9.89.2.5 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

| | |
|----------------|----------|
| <code>x</code> | Position |
|----------------|----------|

Definition at line [7190](#) of file [vpmg.c](#).

Here is the caller graph for this function:



9.89.2.6 VPRIATE 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

| | |
|----------------|----------|
| <code>x</code> | Position |
|----------------|----------|

Definition at line [7217](#) of file [vpmg.c](#).

9.89.2.7 VPRIATE double dbspline2 (double x)

Evaluate a cubic B-spline derivative.

Author

Nathan Baker

Returns

Cubic B-spline derivative

Parameters

| | |
|----------------|----------|
| <code>x</code> | Position |
|----------------|----------|

Definition at line 5500 of file [vpmg.c](#).

Here is the caller graph for this function:

**9.89.2.8 VPRIVATE double dbspline4 (double x)**

Evaluate a 5th Order B-Spline derivative (4th order polynomial)

Author

: Michael Schnieders

Returns

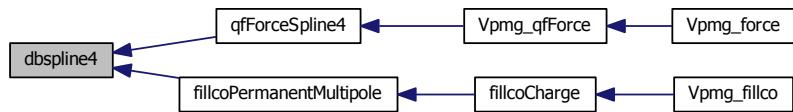
5th Order B-Spline derivative

Parameters

| | |
|----------------|----------|
| <code>x</code> | Position |
|----------------|----------|

Definition at line 7158 of file [vpmg.c](#).

Here is the caller graph for this function:



9.89.2.9 VPRIVATE Vrc.Codes fillcoCharge (Vpmg * thee)

Top-level driver to fill source term charge array.

Returns

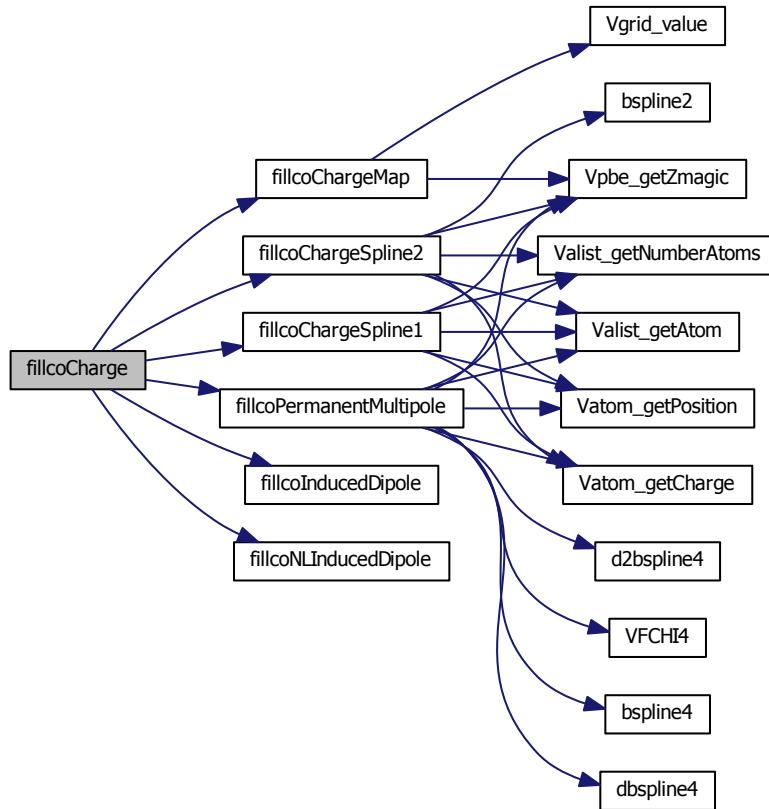
Success/failure status

Author

Nathan Baker

Definition at line 5275 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.89.2.10 VPRIVATE Vrc_Codes fillcoChargeMap (Vpmg * *thee*)

Fill source term charge array from a pre-calculated map.

Returns

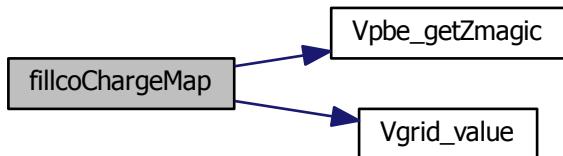
Success/failure status

Author

Nathan Baker

Definition at line 5331 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.89.2.11 VPRIVATE void fillcoChargeSpline1 (Vpmg * *thee*)

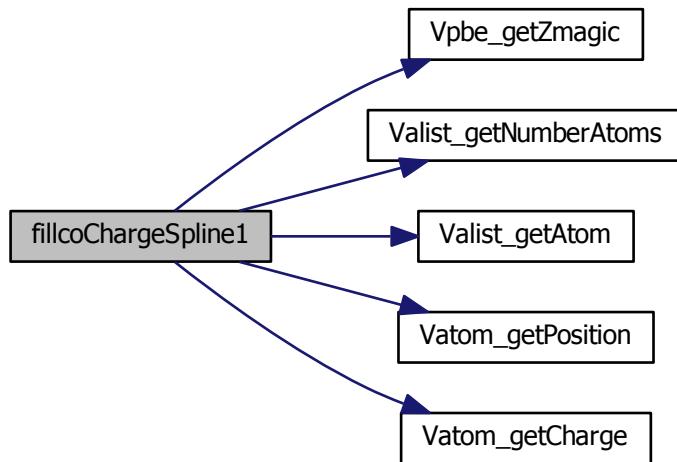
Fill source term charge array from linear interpolation.

Author

Nathan Baker

Definition at line 5379 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**9.89.2.12 VPRIVATE void fillcoChargeSpline2 (Vpmg * *thee*)**

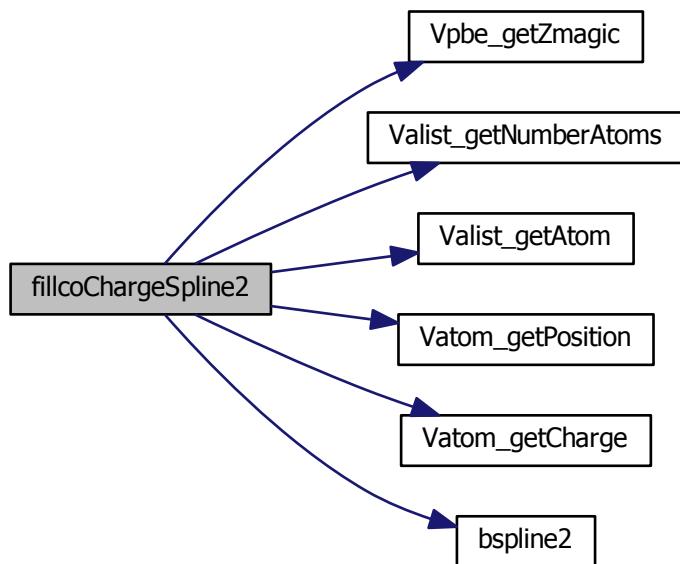
Fill source term charge array from cubic spline interpolation.

Author

Nathan Baker

Definition at line 5516 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.89.2.13 VPRIVATE void fillcoCoef (Vpmg * thee)

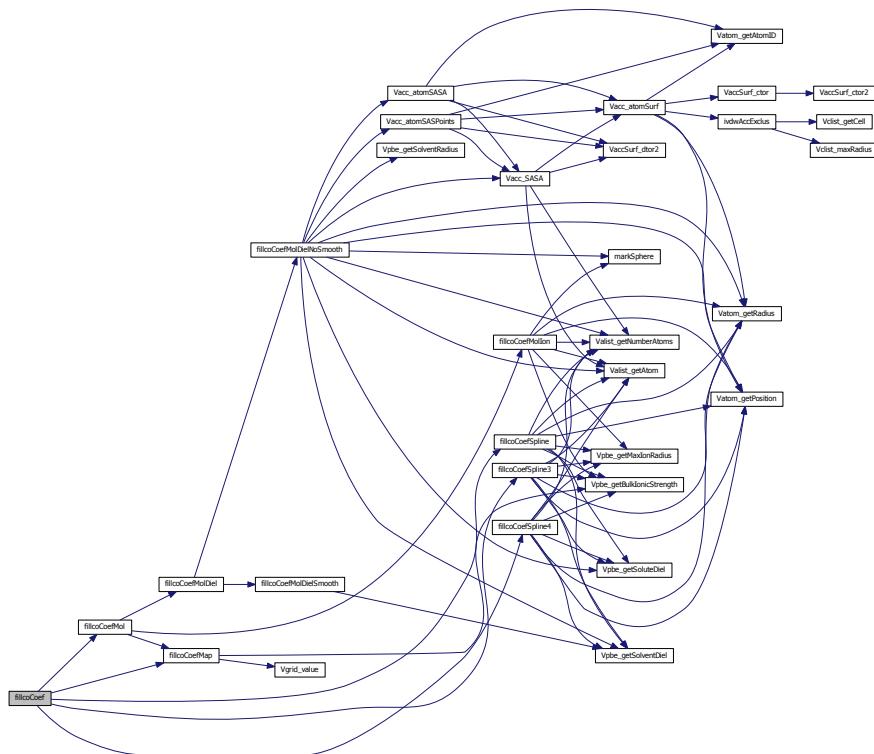
Top-level driver to fill all operator coefficient arrays.

Author

Nathan Baker

Definition at line 5235 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.89.2.14 VPRIVATE void fillcoCoefMap (Vpmg * *thee*)

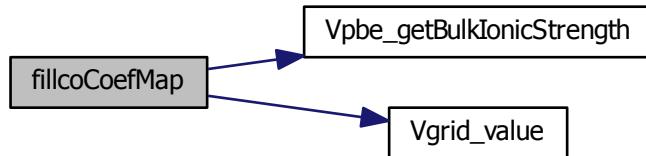
Fill operator coefficient arrays from pre-calculated maps.

Author

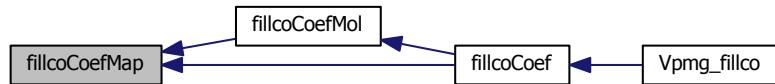
Nathan Baker

Definition at line [4477](#) of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**9.89.2.15 VPRIVATE void fillcoCoefMol (Vpmg * *thee*)**

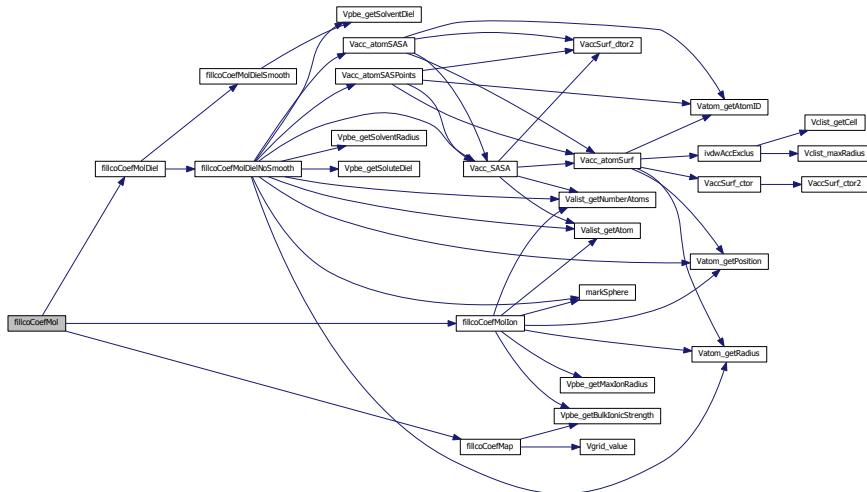
Fill operator coefficient arrays from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4600 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.89.2.16 VPRIVATE void fillcoCoefMolDiEl (Vpmg * thee)

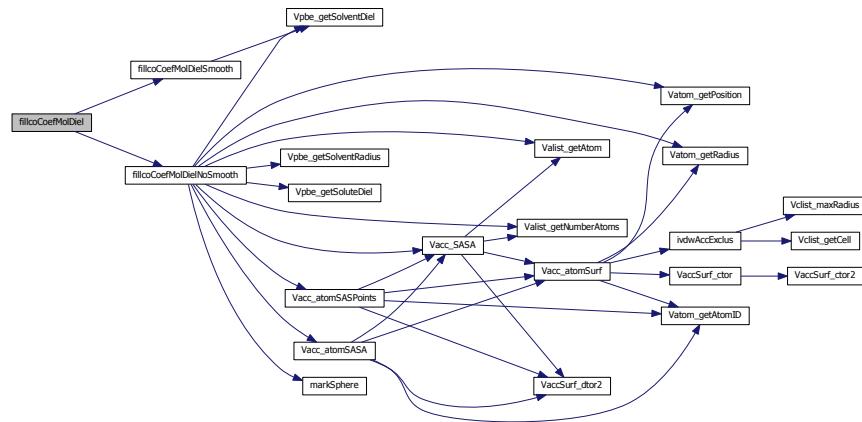
Fill differential operator coefficient arrays from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4714 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**9.89.2.17 VPRIVATE void fillcoCoefMolDielNoSmooth (Vpmg * thee)**

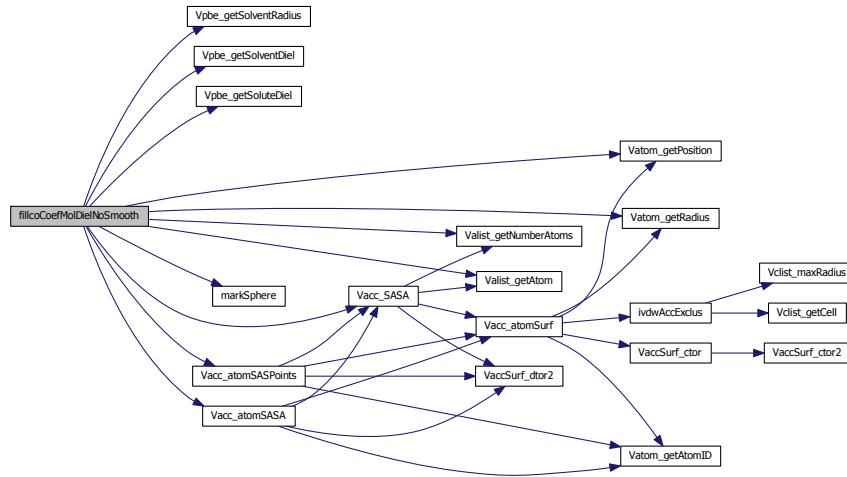
Fill differential operator coefficient arrays from a molecular surface calculation without smoothing.

Author

Nathan Baker

Definition at line 4725 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.89.2.18 VPRIVATE void fillCoefMolDielectricSmooth (**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 4879 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**9.89.2.19 VPRIVATE void fillcoCoefMollon (Vpmg * *thee*)**

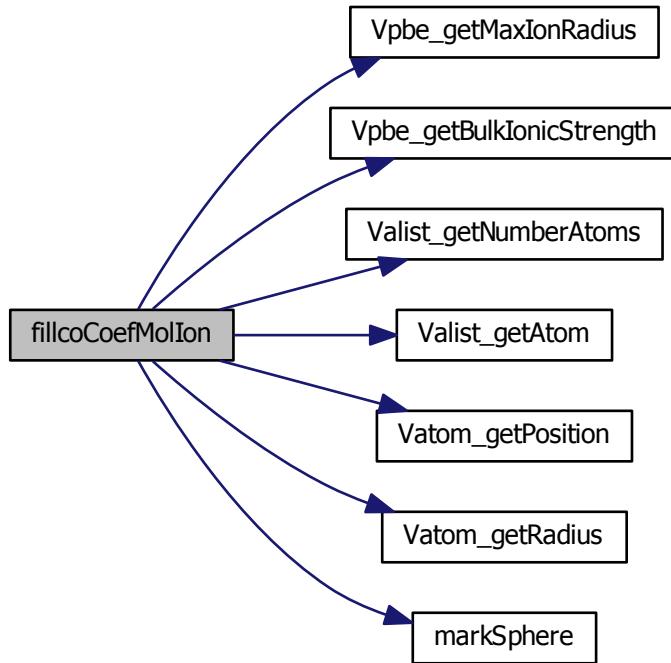
Fill ion (nonlinear) operator coefficient array from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4616 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.89.2.20 VPRIVATE void fillcoCoefSpline (Vpmg * *thee*)

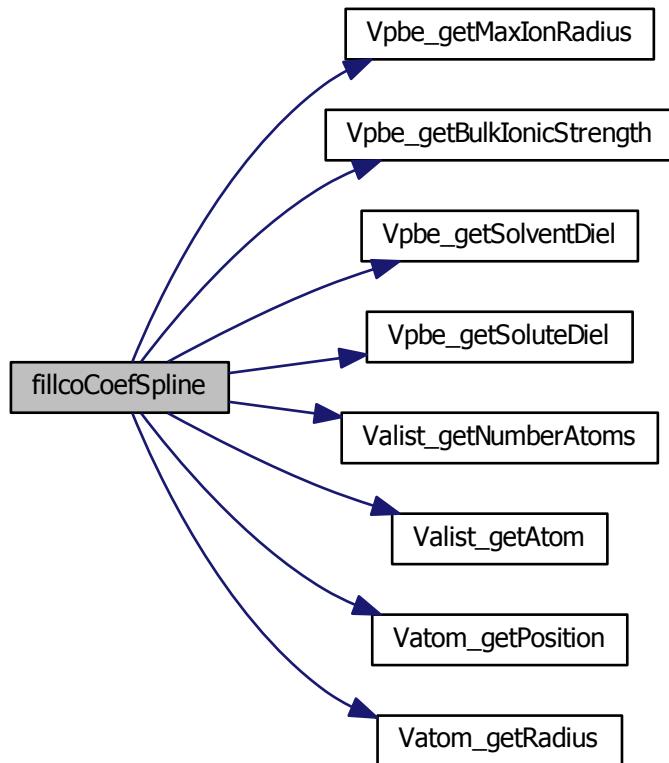
Fill operator coefficient arrays from a spline-based surface calculation.

Author

Nathan Baker

Definition at line 5010 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.89.2.21 VPRIVATE void fillcoCoefSpline3 (Vpmg * thee)

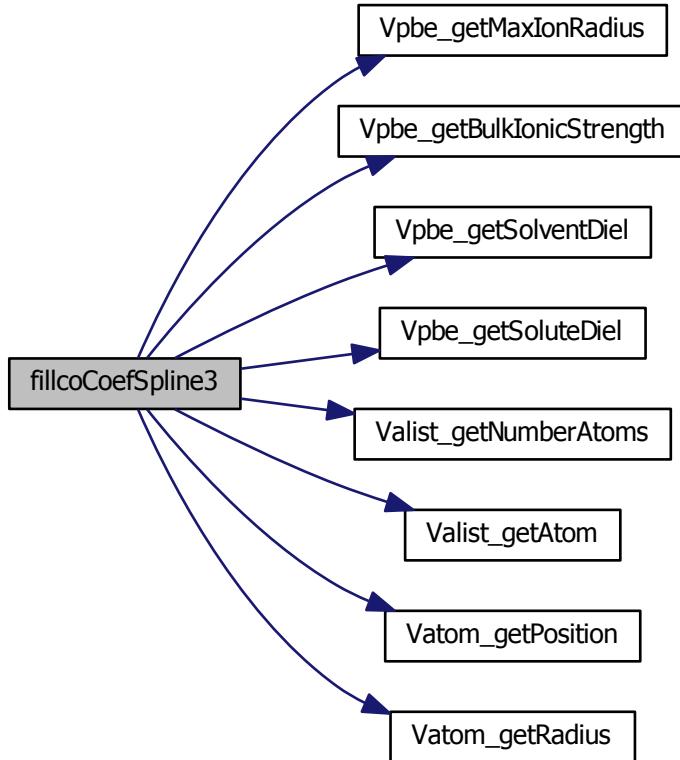
Fill operator coefficient arrays from a 5th order polynomial based surface calculation.

Author

Michael Schnieders

Definition at line 10418 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.89.2.22 VPRIVATE void fillcoCoefSpline4 (Vpmg * thee)

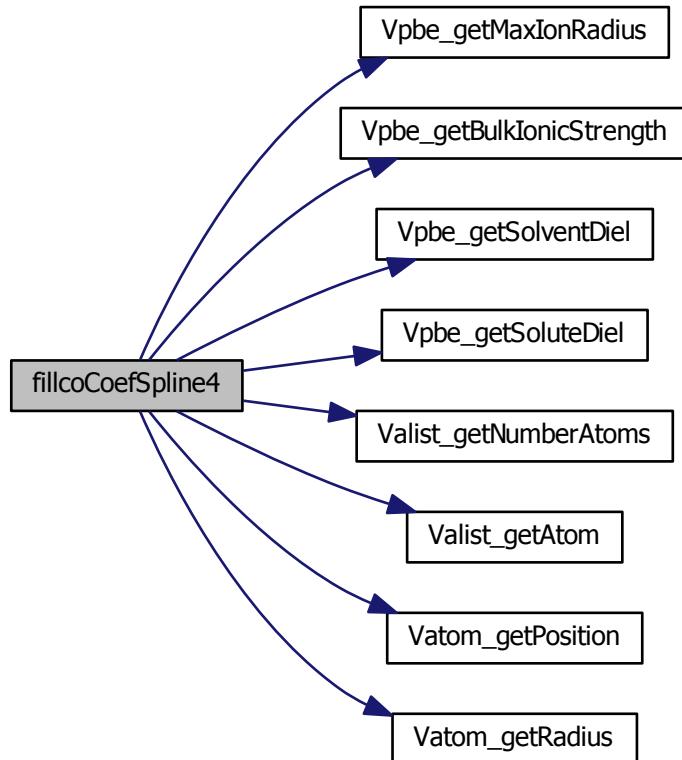
Fill operator coefficient arrays from a 7th order polynomial based surface calculation.

Author

Michael Schnieders

Definition at line 9927 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.89.2.23 VPRIVATE void fillcoInducedDipole (Vpmg * *thee*)

Fill source term charge array for use of induced dipoles.

Author

Michael Schnieders

Here is the caller graph for this function:

**9.89.2.24 VPRIVATE void fillcoNLInducedDipole (Vpmg * *thee*)**

Fill source term charge array for non-local induced dipoles.

Author

Michael Schnieders

Here is the caller graph for this function:

**9.89.2.25 VPRIVATE void fillcoPermanentMultipole (Vpmg * *thee*)**

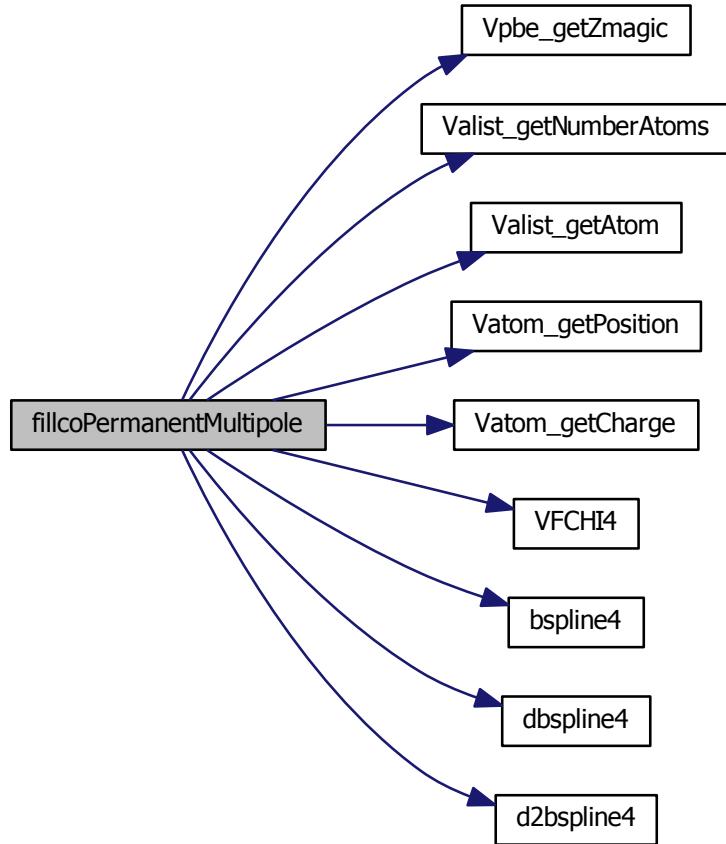
Fill source term charge array for the use of permanent multipoles.

Author

Michael Schnieders

Definition at line 7228 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.89.2.26 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.

Author

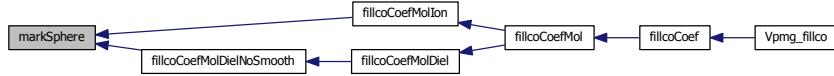
Nathan Baker

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>hzed</i> | Grid spacing |
| <i>xmin</i> | Grid lower corner |
| <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 6837 of file [vpmg.c](#).

Here is the caller graph for this function:



9.89.2.27 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.

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 3480 of file [vpmg.c](#).

9.89.2.28 VPRIVATE void qfForceSpline1 (Vpmg * *thee*, double * *force*, int *atomID*)

Charge-field force due to a linear spline charge function.

Author

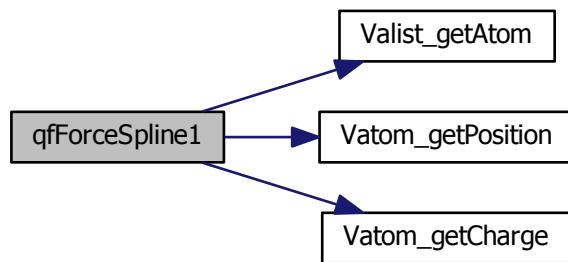
Nathan Baker

Parameters

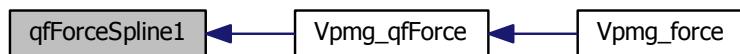
| | |
|---------------|-----------------------------|
| <i>atomID</i> | Set to force Valist atom ID |
|---------------|-----------------------------|

Definition at line 6299 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.89.2.29 VPRIVATE void qfForceSpline2 (Vpmg * *thee*, double * *force*, int *atomID*)

Charge-field force due to a cubic spline charge function.

Author

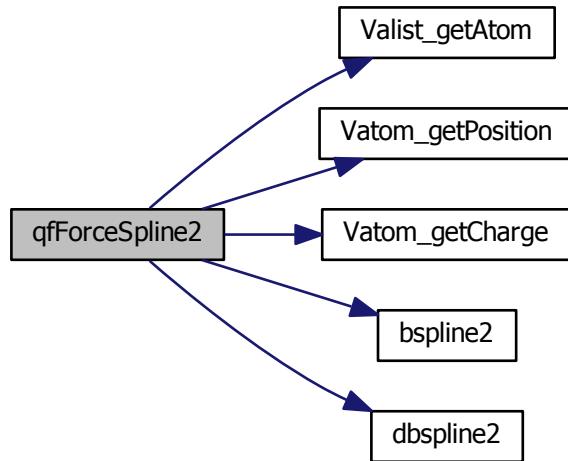
Nathan Baker

Parameters

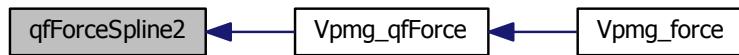
| | |
|---------------|-----------------------------|
| <i>atomID</i> | Set to force Valist atom ID |
|---------------|-----------------------------|

Definition at line 6436 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.89.2.30 VPRIVATE void qfForceSpline4 (Vpmg * *thee*, double * *force*, int *atomID*)

Charge-field force due to a quintic spline charge function.

Author

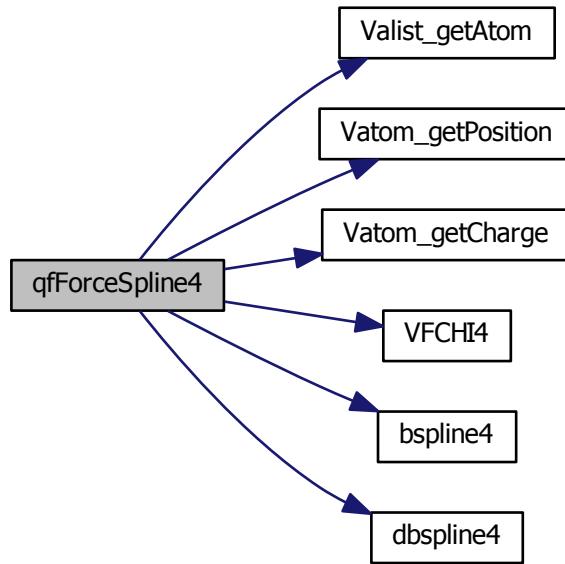
Michael Schnieders

Parameters

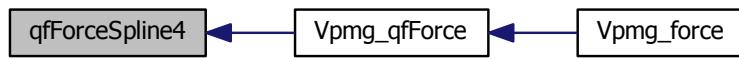
| | |
|---------------|-----------------------------|
| <i>atomID</i> | Set to force Valist atom ID |
|---------------|-----------------------------|

Definition at line 6549 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.89.2.31 VPRIVATE double VFCHI4 (int i, double f)

Return 2.5 plus difference of i - f.

Author

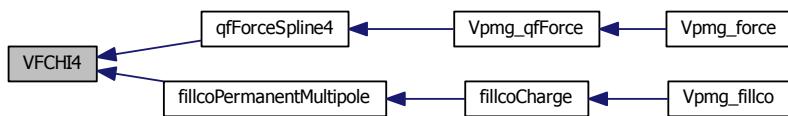
Michael Schnieders

Returns

$(2.5 + ((double)(i) - (f)))$

Definition at line 7120 of file [vpmg.c](#).

Here is the caller graph for this function:



9.89.2.32 VPRIVATE double Vpmg_polarizEnergy (Vpmg * thee, int extFlag)

Determines energy from polarizable charge and interaction with fixed charges according to Rocchia et al.

Author

Nathan Baker

Returns

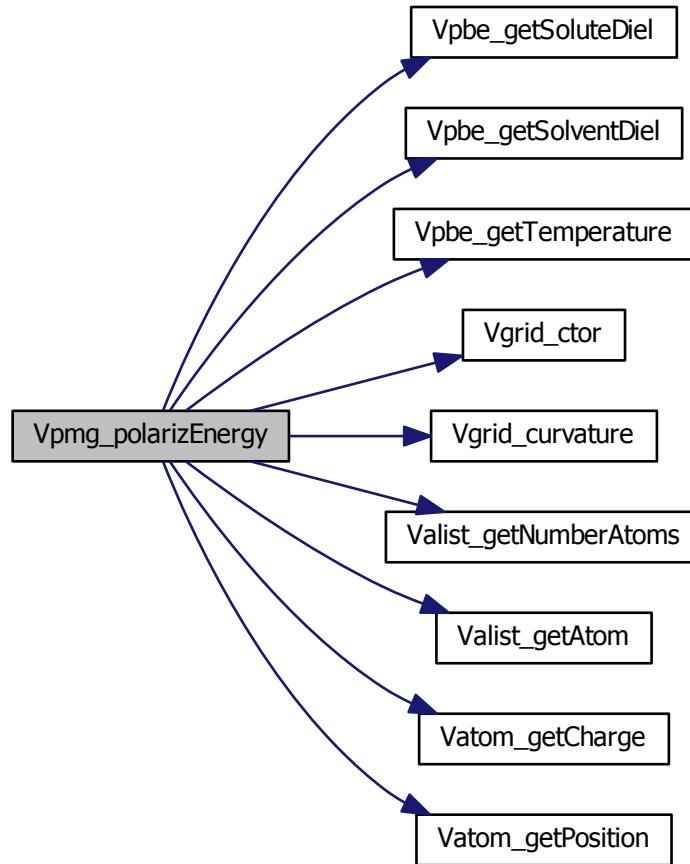
Energy in kT

Parameters

| | |
|----------------|---|
| <i>extFlag</i> | If 1, add external energy contributions to result |
|----------------|---|

Definition at line 1151 of file [vpmg.c](#).

Here is the call graph for this function:



9.89.2.33 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

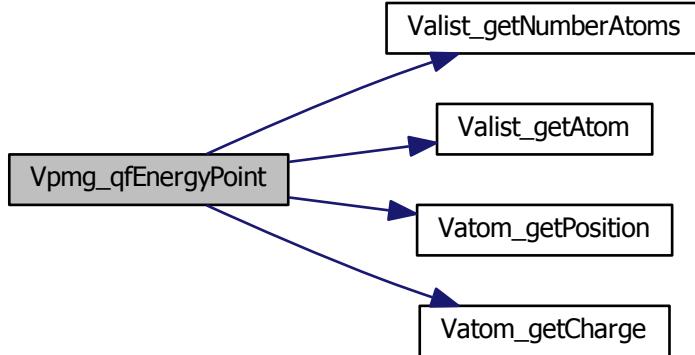
Energy in kT

Parameters

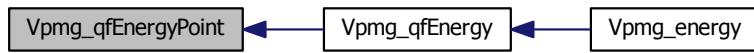
| | |
|----------------------|---|
| <code>extFlag</code> | If 1, add external energy contributions to result |
|----------------------|---|

Definition at line 1707 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.89.2.34 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

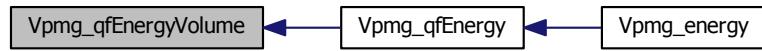
| | |
|----------------|---|
| <i>extFlag</i> | If 1, add external energy contributions to result |
|----------------|---|

Definition at line 1864 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.89.2.35 VPRIVATE double Vpmg_qmEnergySMPBE (Vpmg * thee, int extFlag)

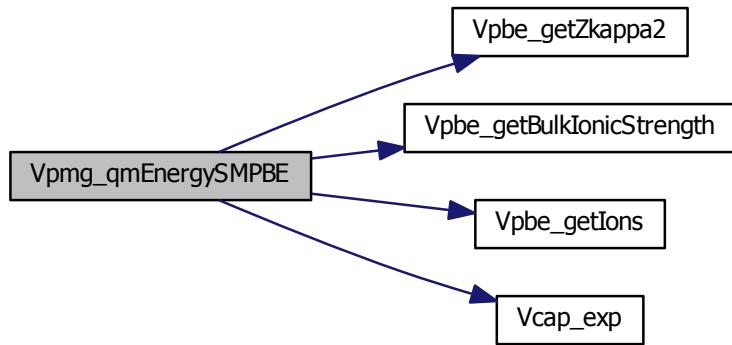
Vpmg_qmEnergy for SMPBE.

Author

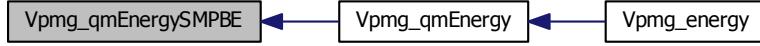
Vincent Chu

Definition at line 1493 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.89.2.36 **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

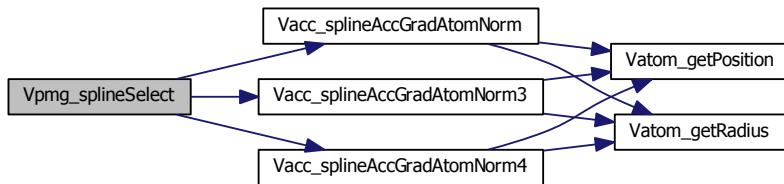
David Gohara

Parameters

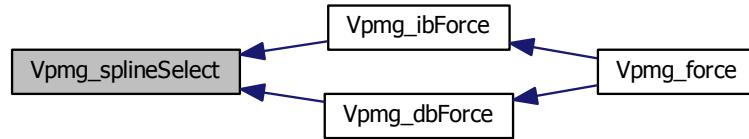
| | |
|---------------|---|
| <i>acc</i> | Surface method, currently VSM_SPLINE, VSM_SPLINE5, or VSM_SPLINE7 |
| <i>gpos</i> | Accessibility object |
| <i>win</i> | Position array -> array[3] |
| <i>infrad</i> | Spline window |
| <i>atom</i> | Inflation radius |
| <i>force</i> | Atom object Force array -> array[3] |

Definition at line 1896 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.89.2.37 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

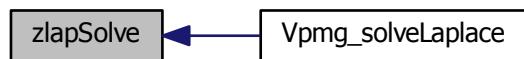
Vpmg_fillco must be called first

Parameters

| | |
|---------------|--------------------------------|
| <i>source</i> | Solution term vector |
| <i>work1</i> | Source term vector Work vector |

Definition at line 6886 of file [vpmg.c](#).

Here is the caller graph for this function:



9.90 vpmg.h

```

00001
00081 #ifndef _VPMG_H_
00082 #define _VPMG_H_
00083
  
```

```
00084 /* Generic headers */
00085 #include "apbscfg.h"
00086 #include "maloc/maloc.h"
00087
00088 #include "apbs/vhal.h"
00089 #include "apbs/vpmgp.h"
00090 #include "apbs/vacc.h"
00091 #include "apbs/vcap.h"
00092 #include "apbs/vpbe.h"
00093 #include "apbs/vgrid.h"
00094 #include "apbs/mgparm.h"
00095 #include "apbs/pbeparm.h"
00096 #include "apbs/vmatrix.h"
00097
00098 #include "apbs/mgdrv.h"
00099 #include "apbs/newdrv.h"
00100 #include "apbs/mgsb.h"
00101 #include "apbs/mikpckd.h"
00102 #include "apbs/matvecd.h"
00103
00104
00105
00109 #define VPMGMAXPART 2000
00110
00120 struct sVpmg {
00121
00122     Vmem *vmem;
00123     Vpmgp *pmgp;
00124     Vpbe *pbe;
00126 #ifdef BURY_FORTRAN
00127     Vpde *pde;
00128     Vmgdriver *mgdriver;
00129 #endif
00130
00131     double *epsx;
00132     double *epsy;
00133     double *epsz;
00134     double *kappa;
00135     double *pot;
00136     double *charge;
00138     int *iparm;
00139     double *rparm;
00140     int *iwork;
00141     double *rwork;
00142     double *alcf;
00144     double *a2cf;
00146     double *a3cf;
00148     double *ccf;
00149     double *fcf;
00150     double *tcf;
00151     double *u;
00152     double *xf;
00153     double *yf;
00154     double *zf;
00155     double *gxcf;
00156     double *gycf;
00157     double *gzcf;
00158     double *pvec;
00159     double extDiEnergy;
00161     double extQmEnergy;
00163     double extQfEnergy;
00165     double extNpEnergy;
00167     Vsurf_Meth surfMeth;
00168     double splineWin;
00169     Vchrg_Meth chargeMeth;
00170     Vchrg_Src chargeSrc;
00172     int filled;
00174     int useDie1XMap;
00176     Vgrid *die1XMap;
00177     int useDie1YMap;
00179     Vgrid *die1YMap;
00180     int useDie1ZMap;
00182     Vgrid *die1ZMap;
00183     int useKappaMap;
00185     Vgrid *kappaMap;
00186     int usePotMap;
00188     Vgrid *potMap;
00190     int useChargeMap;
00192     Vgrid *chargeMap;
00193 };
00194
```

```
00199 typedef struct sVpmg Vpmg;
00200
00201 /* //////////////////////////////// */
00204 #if !defined(VINLINE_VPMG)
00205
00212     VEXTERNC unsigned long int Vpmg_memChk(
00213         Vpmg *thee
00214     );
00215
00216 #else /* if defined(VINLINE_VPMG) */
00217
00218 # define Vpmg_memChk(thee) (Vmem_bytes((thee)->vmem))
00219
00220 #endif /* if !defined(VINLINE_VPMG) */
00221
00222 /* //////////////////////////////// */
00225
00230 VEXTERNC Vpmg* Vpmg_ctor(
00231     Vpmgp *parms,
00232     Vpbe *pbe,
00233     int focusFlag,
00234     Vpmg *pmgOLD,
00235     MGparm *mgparm,
00236     PBEparm_calcEnergy energyFlag
00237 );
00238
00246 VEXTERNC int Vpmg_ctor2(
00247     Vpmg *thee,
00248     Vpmgp *parms,
00249     Vpbe *pbe,
00250     int focusFlag,
00251     Vpmg *pmgOLD,
00253     MGparm *mgparm,
00255     PBEparm_calcEnergy energyFlag
00258 );
00259
00264 VEXTERNC void Vpmg_dtor(
00265     Vpmg **thee
00267 );
00268
00273 VEXTERNC void Vpmg_dtor2(
00274     Vpmg *thee
00275 );
00276
00285 VEXTERNC int Vpmg_fillco(
00286     Vpmg *thee,
00287     Vsurf_Meth surfMeth,
00288     double splineWin,
00290     Vchrg_Meth chargeMeth,
00291     int useDielXMap,
00292     Vgrid *dielXMap,
00293     int useDielyMap,
00294     Vgrid *dielYMap,
00295     int useDielZMap,
00296     Vgrid *dielZMap,
00297     int useKappaMap,
00298     Vgrid *kappaMap,
00299     int usePotMap,
00300     Vgrid *potMap,
00301     int useChargeMap,
00302     Vgrid *chargeMap
00303 );
00304
00310 VEXTERNC int Vpmg_solve(
00311     Vpmg *thee
00312 );
00313
00325 VEXTERNC int Vpmg_solveLaplace(
00326     Vpmg *thee
00327 );
00328
00338 VEXTERNC double Vpmg_energy(
00339     Vpmg *thee,
00340     int extFlag
00344 );
00345
00363 VEXTERNC double Vpmg_qfEnergy(
00364     Vpmg *thee,
00365     int extFlag
00369 );
00370
```

```

00390 VEXTERNC double Vpmg_qfAtomEnergy(
00391     Vpmg *thee,
00392     Vatom *atom
00393 );
00394
00419 VEXTERNC double Vpmg_qmEnergy(
00420     Vpmg *thee,
00421     int extFlag
00425 );
00426
00427
00446 VEXTERNC double Vpmg_dielEnergy(
00447     Vpmg *thee,
00448     int extFlag
00452 );
00453
00454
00471 VEXTERNC double Vpmg_dielGradNorm(
00472     Vpmg *thee
00473 );
00474
00486 VEXTERNC int Vpmg_force(
00487     Vpmg *thee,
00488     double *force,
00490     int atomID,
00491     Vsurf_Meth srfm,
00492     Vchrg_Meth chgm
00493 );
00494
00506 VEXTERNC int Vpmg_qfForce(
00507     Vpmg *thee,
00508     double *force,
00510     int atomID,
00511     Vchrg_Meth chgm
00512 );
00513
00525 VEXTERNC int Vpmg_dbForce(
00526     Vpmg *thee,
00527     double *dbForce,
00529     int atomID,
00530     Vsurf_Meth srfm
00531 );
00532
00544 VEXTERNC int Vpmg_ibForce(
00545     Vpmg *thee,
00546     double *force,
00548     int atomID,
00549     Vsurf_Meth srfm
00550 );
00551
00557 VEXTERNC void Vpmg_setPart(
00558     Vpmg *thee,
00559     double lowerCorner[3],
00560     double upperCorner[3],
00561     int bflags[6]
00565 );
00566
00571 VEXTERNC void Vpmg_unsetPart(
00572     Vpmg *thee
00573 );
00574
00580 VEXTERNC int Vpmg_fillArray(
00581     Vpmg *thee,
00582     double *vec,
00584     Vdata_Type type,
00585     double parm,
00586     Vhal_PBEType pbetype,
00587     PBEparm * pbeparm
00588 );
00589
00595 VPUBLIC void Vpmg_fieldspline4(
00596     Vpmg *thee,
00597     int atomID,
00598     double field[3]
00599 );
00600
00608 VEXTERNC double Vpmg_qfPermanentMultipoleEnergy(
00609     Vpmg *thee,
00610     int atomID
00611 );
00612

```

```
00618 VEXTERNC void Vpmg_qfPermanentMultipoleForce(
00619     Vpmg *thee,
00620     int atomID,
00621     double force[3],
00622     double torque[3]
00623 );
00624
00629 VEXTERNC void Vpmg_ibPermanentMultipoleForce(
00630     Vpmg *thee,
00631     int atomID,
00632     double force[3]
00633 );
00634
00639 VEXTERNC void Vpmg_dbPermanentMultipoleForce(
00640     Vpmg *thee,
00641     int atomID,
00642     double force[3]
00643 );
00644
00651 VEXTERNC void Vpmg_qfDirectPolForce(
00652     Vpmg *thee,
00653     Vgrid *perm,
00654     Vgrid *induced,
00655     int atomID,
00656     double force[3],
00657     double torque[3]
00658 );
00659
00668 VEXTERNC void Vpmg_qfNLDirectPolForce(
00669     Vpmg *thee,
00670     Vgrid *perm,
00671     Vgrid *nlInduced,
00672     int atomID,
00673     double force[3],
00674     double torque[3]
00675 );
00676
00684 VEXTERNC void Vpmg_ibDirectPolForce(
00685     Vpmg *thee,
00686     Vgrid *perm,
00687     Vgrid *induced,
00688     int atomID,
00689     double force[3]
00690 );
00691
00700 VEXTERNC void Vpmg_ibNLDirectPolForce(
00701     Vpmg *thee,
00702     Vgrid *perm,
00703     Vgrid *nlInduced,
00704     int atomID,
00705     double force[3]
00706 );
00707
00715 VEXTERNC void Vpmg_dbDirectPolForce(
00716     Vpmg *thee,
00717     Vgrid *perm,
00718     Vgrid *induced,
00719     int atomID,
00720     double force[3]
00721 );
00722
00731 VEXTERNC void Vpmg_dbNLDirectPolForce(
00732     Vpmg *thee,
00733     Vgrid *perm,
00734     Vgrid *nlInduced,
00735     int atomID,
00736     double force[3]
00737 );
00738
00745 VEXTERNC void Vpmg_qfMutualPolForce(
00746     Vpmg *thee,
00747     Vgrid *induced,
00748     Vgrid *nlInduced,
00749     int atomID,
00750     double force[3]
00751 );
00752
00760 VEXTERNC void Vpmg_ibMutualPolForce(
00761     Vpmg *thee,
00762     Vgrid *induced,
00763     Vgrid *nlInduced,
```

```

00764         int atomID,
00765         double force[3]
00766     );
00767
00775 VEXTERN void Vpmg_dbMutualPolForce(
00776     Vpmg *thee,
00777     Vgrid *induced,
00778     Vgrid *nlInduced,
00779     int atomID,
00780     double force[3]
00781 );
00782
00789 VEXTERN void Vpmg_printColComp(
00790     Vpmg *thee,
00791     char path[72],
00792     char title[72],
00793     char mxtype[3],
00794     int flag
00795 );
00796
00797
00798
00815 VPRIVATE void bcolcomp(
00816     int *iparm, ///< @todo Document
00817     double *rparm, ///< @todo Document
00818     int *iwork, ///< @todo Document
00819     double *rwork, ///< @todo Document
00820     double *values, ///< @todo Document
00821     int *rowind, ///< @todo Document
00822     int *colptr, ///< @todo Document
00823     int *flag
00824 );
00825
00826
00827
00838 VPRIVATE void bcolcomp2(
00839     int *iparm, ///< @todo Document
00840     double *rparm, ///< @todo Document
00841     int *nx, ///< @todo Document
00842     int *ny, ///< @todo Document
00843     int *nz, ///< @todo Document
00844     int *iz, ///< @todo Document
00845     int *ipc, ///< @todo Document
00846     double *rpc, ///< @todo Document
00847     double *ac, ///< @todo Document
00848     double *cc, ///< @todo Document
00849     double *values, ///< @todo Document
00850     int *rowind, ///< @todo Document
00851     int *colptr, ///< @todo Document
00852     int *flag
00853 );
00854
00855
00856
00867 VPRIVATE void bcolcomp3(
00868     int *nx, ///< @todo Document
00869     int *ny, ///< @todo Document
00870     int *nz, ///< @todo Document
00871     int *ipc, ///< @todo Document
00872     double *rpc, ///< @todo Document
00873     double *ac, ///< @todo Document
00874     double *cc, ///< @todo Document
00875     double *values, ///< @todo Document
00876     int *rowind, ///< @todo Document
00877     int *colptr, ///< @todo Document
00878     int *flag ///< @todo Document
00879 );
00880
00881
00882
00889 VPRIVATE void bcolcomp4(
00890     int *nx, ///< @todo Document
00891     int *ny, ///< @todo Document
00892     int *nz, ///< @todo Document
00893     int *ipc, ///< @todo Document
00894     double *rpc, ///< @todo Document
00895     double *oC, ///< @todo Document
00896     double *cc, ///< @todo Document
00897     double *oE, ///< @todo Document
00898     double *oN, ///< @todo Document
00899     double *uC, ///< @todo Document

```

```

00900     double *values, ///< @todo Document
00901     int    *rowind, ///< @todo Document
00902     int    *colptr, ///< @todo Document
00903     int    *flag   ///< @todo Document
00904   );
00905
00906
00907
00914 VPRIVATE void pcolcomp(
00915     int    *nrow,   ///< @todo Document
00916     int    *ncol,   ///< @todo Document
00917     int    *nnzero, ///< @todo Document
00918     double *values, ///< @todo Document
00919     int    *rowind, ///< @todo Document
00920     int    *colptr, ///< @todo Document
00921     char   *path,  ///< @todo Document
00922     char   *title, ///< @todo Document
00923     char   *mxtype ///< @todo Document
00924   );
00925
00926
00927
00928 /* /////////////////////////////////
00929 // Internal routines
00931
00937 VPRIVATE double bspline2(
00938     double x
00939   );
00940
00946 VPRIVATE double dbspline2(
00947     double x
00948   );
00949
00955 VPRIVATE double VFCHI4(
00956     int i,
00957     double f
00958   );
00959
00965 VPRIVATE double bspline4(
00966     double x
00967   );
00968
00974 VPRIVATE double dbspline4(
00975     double x
00976   );
00977
00983 VPRIVATE double d2bspline4(
00984     double x
00985   );
00986
00992 VPRIVATE double d3bspline4(
00993     double x
00994   );
00995
01002 VPRIVATE double Vpmg_polarizEnergy(
01003     Vpmg *thee,
01004     int extFlag
01006   );
01013 VPRIVATE double Vpmg_qfEnergyPoint(
01014     Vpmg *thee,
01015     int extFlag
01017   );
01018
01024 VPRIVATE double Vpmg_qfEnergyVolume(
01025     Vpmg *thee,
01026     int extFlag
01028   );
01029
01035 VPRIVATE void Vpmg_splineSelect(
01036     int srfm,
01038     Vacc *acc,
01039     double *gpos,
01040     double win,
01041     double infrad,
01042     Vatom *atom,
01043     double *force
01044   );
01045
01051 VPRIVATE void focusFillBound(
01052     Vpmg *thee,
01053     Vpmg *pmg

```

```
01054      );
01055
01062 VPRIVATE void bcf11(
01063     double size,
01064     double *apos,
01065     double charge,
01066     double xkappa,
01067     double prel,
01068     double *gxcf,
01069     double *gycf,
01070     double *gzcf,
01071     double *xf,
01072     double *yf,
01073     double *zf,
01074     int nx,
01075     int ny,
01076     int nz
01077 );
01078
01084 VPRIVATE void bcf12(
01085     double size,
01086     double *apos,
01087     double charge,
01088     double *dipole,
01089     double *quad,
01090     double xkappa,
01091     double eps_p,
01092     double eps_w,
01093     double T,
01094     double *gxcf,
01095     double *gycf,
01096     double *gzcf,
01097     double *xf,
01098     double *yf,
01099     double *zf,
01100     int nx,
01101     int ny,
01102     int nz
01103 );
01104
01113 VPRIVATE void multipolebc(
01114     double r,
01115     double kappa,
01116     double eps_p,
01117     double eps_w,
01118     double rad,
01119     double tsr[3]
01120 );
01121
01130 VPRIVATE double bcf11sp(
01131     double size,
01132     double *apos,
01133     double charge,
01134     double xkappa,
01135     double prel,
01136     double *pos
01137 );
01138
01143 VPRIVATE void bcCalc(
01144     Vpmg *thee
01145 );
01146
01151 VPRIVATE void fillcoCoef(
01152     Vpmg *thee
01153 );
01154
01159 VPRIVATE void fillcoCoefMap(
01160     Vpmg *thee
01161 );
01162
01168 VPRIVATE void fillcoCoefMol(
01169     Vpmg *thee
01170 );
01171
01177 VPRIVATE void fillcoCoefMolion(
01178     Vpmg *thee
01179 );
01180
01186 VPRIVATE void fillcoCoefMolDiel(
01187     Vpmg *thee
01188 );
```

```
01189
01195 VPRIVATE void fillcoCoefMolDielNoSmooth(
01196     Vpmg *thee
01197 );
01198
01212 VPRIVATE void fillcoCoefMolDielSmooth(
01213     Vpmg *thee
01214 );
01215
01221 VPRIVATE void fillcoCoefSpline(
01222     Vpmg *thee
01223 );
01224
01230 VPRIVATE void fillcoCoefSpline3(
01231     Vpmg *thee
01232 );
01233
01239 VPRIVATE void fillcoCoefSpline4(
01240     Vpmg *thee
01241 );
01242
01248 VPRIVATE Vrc_Codes fillcoCharge(
01249     Vpmg *thee
01250 );
01251
01257 VPRIVATE Vrc_Codes fillcoChargeMap(
01258     Vpmg *thee
01259 );
01260
01265 VPRIVATE void fillcoChargeSpline1(
01266     Vpmg *thee
01267 );
01268
01273 VPRIVATE void fillcoChargeSpline2(
01274     Vpmg *thee
01275 );
01276
01281 VPRIVATE void fillcoPermanentMultipole(
01282     Vpmg *thee
01283 );
01284
01289 VPRIVATE void fillcoInducedDipole(
01290     Vpmg *thee
01291 );
01292
01298 VPRIVATE void fillcoNLInducedDipole(
01299     Vpmg *thee
01300 );
01301
01308 VPRIVATE void extEnergy(
01309     Vpmg *thee,
01310     Vpmg *pmgOLD,
01311     PBEparm_calcEnergy extFlag,
01312     double partMin[3],
01313     double partMax[3],
01314     int bflags[6]
01315 );
01316
01321 VPRIVATE void qfForceSpline1(
01322     Vpmg *thee,
01323     double *force,
01324     int atomID
01325 );
01326
01331 VPRIVATE void qfForceSpline2(
01332     Vpmg *thee,
01333     double *force,
01334     int atomID
01335 );
01336
01341 VPRIVATE void qfForceSpline4(
01342     Vpmg *thee,
01343     double *force,
01344     int atomID
01345 );
01346
01347
01355 VPRIVATE void zlapSolve(
01356     Vpmg *thee,
01357     double **solution,
01358     double **source,
```

```

01359     double **work1
01360     );
01361
01368 VPRIVATE void markSphere(
01369     double rtot,
01370     double *tpos,
01371     int nx,
01372     int ny,
01373     int nz,
01374     double hx,
01375     double hy,
01376     double hzed,
01377     double xmin,
01378     double ymin,
01379     double zmin,
01380     double *array,
01381     double markVal
01382     );
01383
01388 VPRIVATE double Vpmg_qmEnergySMPBE(Vpmg *thee, int extFlag);
01389 VPRIVATE double Vpmg_qmEnergyNONLIN(Vpmg *thee, int extFlag);
01390
01391
01392
01393 // Additional macros and definitions. May not be needed
01394
01395 // Added by Vincent Chu 9/13/06 for SMPB
01396 #define VCUB(x)          ((x)*(x)*(x))
01397 #define VLOG(x)           (log(x))
01398
01399 #define IJK(i,j,k)    (((k)*(nx)*(ny))+((j)*(nx))+(i))
01400 #define IJKx(j,k,i)   (((i)*(ny)*(nz))+((k)*(ny))+(j))
01401 #define IJKy(i,k,j)   (((j)*(nx)*(nz))+((k)*(nx))+(i))
01402 #define IJKz(i,j,k)   (((k)*(nx)*(ny))+((j)*(nx))+(i))
01403 #define VFCHI(iint,iflt) (1.5+((double)(iint)-(iflt)))
01404
01405
01406 #endif /* ifndef _VPMG_H_ */
01407

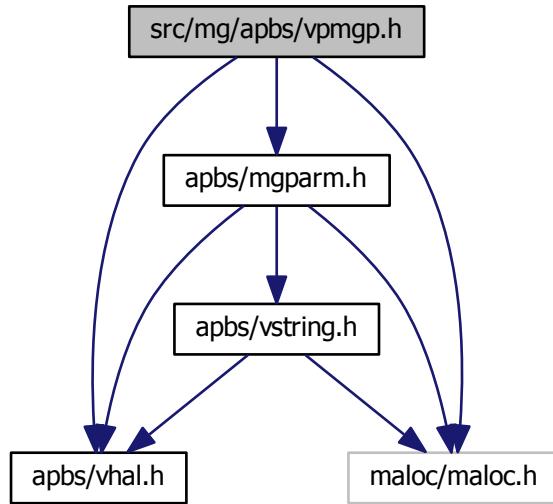
```

9.91 src/mg/apbs/vpmgp.h File Reference

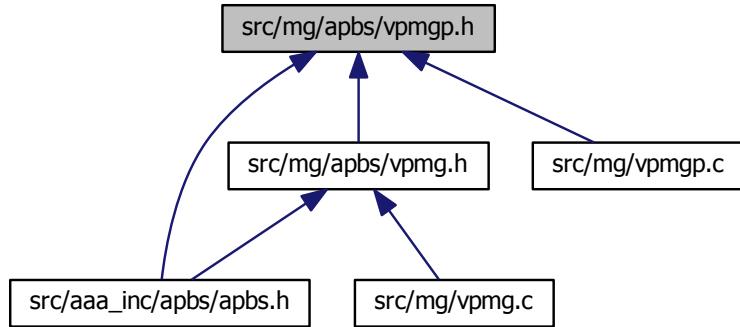
Contains declarations for class Vpmgp.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/mgparm.h"
```

Include dependency graph for vpmgp.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [sVpmgp](#)

Contains public data members for Vpmgp class/module.

Typedefs

- `typedef struct sVpmgp Vpmgp`

Declaration of the Vpmgp class as the `sVpmgp` structure.

Functions

- `VEXTERNC Vpmgp * Vpmgp_ctor (MGparm *mgparm)`
Construct PMG parameter object and initialize to default values.
- `VEXTERNC int Vpmgp_ctor2 (Vpmgp *thee, MGparm *mgparm)`
FORTRAN stub to construct PMG parameter object and initialize to default values.
- `VEXTERNC void Vpmgp_dtor (Vpmgp **thee)`
Object destructor.
- `VEXTERNC void Vpmgp_dtor2 (Vpmgp *thee)`
FORTRAN stub for object destructor.
- `VEXTERNC void Vpmgp_size (Vpmgp *thee)`
Determine array sizes and parameters for multigrid solver.
- `VEXTERNC void Vpmgp_makeCoarse (int numLevel, int nxOld, int nyOld, int nzOld, int *nxNew, int *nyNew, int *nzNew)`
Coarsen the grid by the desired number of levels and determine the resulting numbers of grid points.

9.91.1 Detailed Description

Contains declarations for class Vpmgp.

Version

Id:

[vpmgp.h](#) 1667 2011-12-02 23:22:02Z pcellis

Author

Nathan A. Baker

Note

Variables and many default values taken directly from PMG

Attention

```

*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 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.h](#).

9.92 vpmgp.h

```

00001
00065 #ifndef _VPMGP_H_
00066 #define _VPMGP_H_
00067
00068 #include "maloc/maloc.h"
00069 #include "apbs/vhal.h"
00070 #include "apbs/mgparm.h"
00071
00078 struct sVpmgp {
00079
00080     /* ***** USER-SPECIFIED PARAMETERS **** */
00081     int nx;
00082     int ny;
00083     int nz;
00084     int nlev;
00085     double hx;
00086     double hy;
00087     double hzed;
00088     int nonlin;
00089     /* ***** DERIVED PARAMETERS **** */
00090     int nxc;
00091     int nyc;
00092     int nzc;
00093     int nf;
00094     int nc;
00095     int narrc;
00096     int n_rpc;
00097     int n_iz;

```

```

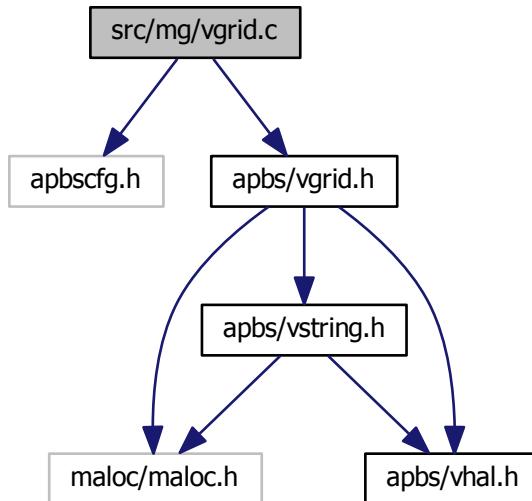
00102     int n_ipc;
00104     int nrwk;
00105     int niwk;
00106     int narr;
00107     int ipkey;
00115     /* ***** PARAMETERS WITH DEFAULT VALUES ***** */
00116     double xcent;
00117     double ycent;
00118     double zcent;
00119     double errtol;
00120     int itmax;
00121     int istop;
00128     int iinfo;
00133     Vbcfl bcfl;
00134     int key;
00137     int iperf;
00142     int meth;
00153     int mgkey;
00156     int nul;
00157     int nu2;
00158     int mgs moo;
00164     int mgprol;
00168     int mgcoar;
00172     int mgsolv;
00175     int mgdisc;
00178     double omegal;
00179     double omegan;
00180     int irite;
00181     int ipcon;
00187     double xlabel;
00188     double ylabel;
00189     double zlabel;
00190     double xmin;
00191     double ymin;
00192     double zmin;
00193     double xmax;
00194     double ymax;
00195     double zmax;
00196 };
00197
00202 typedef struct sVpmgp Vpmgp;
00203
00204 /* //////////////////////////////// */
00205 // Class Vpmgp: Inlineable methods (vpmgp.c)
00207
00208 #if !defined(VINLINE_VPMGP)
00209 #else /* if defined(VINLINE_VPMGP) */
00210 #endif /* if !defined(VINLINE_VPMGP) */
00211
00212 /* //////////////////////////////// */
00213 // Class Vpmgp: Non-Inlineable methods (vpmgp.c)
00215
00222 VEXTERNC Vpmgp* Vpmgp_ctor(MGparm *mgparm);
00223
00232 VEXTERNC int Vpmgp_ctor2(Vpmgp *thee, MGparm *mgparm);
00233
00239 VEXTERNC void Vpmgp_dtor(Vpmgp **thee);
00240
00246 VEXTERNC void Vpmgp_dtor2(Vpmgp *thee);
00247
00252 VEXTERNC void Vpmgp_size(
00253     Vpmgp *thee
00254 );
00255
00260 VEXTERNC void Vpmgp_makeCoarse(
00261     int numLevel,
00262     int nxOld,
00263     int nyOld,
00264     int nzOld,
00265     int *nxNew,
00266     int *nyNew,
00267     int *nzNew
00268 );
00269
00270
00271
00272 #endif /* ifndef _VPMGP_H_ */

```

9.93 src/mg/vgrid.c File Reference

Class Vgrid methods.

```
#include "apbscfg.h"
#include "apbs/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]

9.93.1 Detailed Description

Class Vgrid methods.

Author

Nathan Baker

Version

Id:

[vgrid.c](#) 1757 2012-07-18 20:36:42Z tuckerbeck

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](#).

9.93.2 Function Documentation

9.93.2.1 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 808 of file [vgrid.c](#).

9.94 vgrid.c

```

00001
00057 #include "apbscfg.h"
00058 #include "apbs/vgrid.h"
00059
00060 VEMBED(rcsid="$Id: vgrid.c 1757 2012-07-18 20:36:42Z tuckerbeck $")
00061
00062 #if !defined(VINLINE_VGRID)
00063     VPUBLIC unsigned long int Vgrid_memChk(Vgrid *thee) {
00064         return Vmem_bytes(thee->mem);
00065     }
00066 #endif
00067 #define IJK(i,j,k) (((k)*(nx)*(ny)) + ((j)*(nx)) + (i))
00068
00069 VPRIVATE char *MCwhiteChars = " =,;\t\n";
00070 VPRIVATE char *MCcommChars = "#%";
00071 VPRIVATE double Vcompare;
00072 VPRIVATE char Vprecision[26];
00073
00074 /* //////////////////////////////// */
00075 // Routine: Vgrid_ctor
00076 // Author: Nathan Baker
00078 VPUBLIC Vgrid* Vgrid_ctor(int nx,

```

```

00079             int ny,
00080             int nz,
00081             double hx,
00082             double hy,
00083             double hzed,
00084             double xmin,
00085             double ymin,
00086             double zmin,
00087             double *data
00088         ) {
00089
00090     Vgrid *thee = VNULL;
00091
00092     thee = (Vgrid*)Vmem_malloc(VNULL, 1, sizeof(Vgrid));
00093     VASSERT(thee != VNULL);
00094     VASSERT(Vgrid_ctor2(thee, nx, ny, nz, hx, hy, hzed,
00095                           xmin, ymin, zmin, data));
00096
00097     return thee;
00098 }
00099
00100 /* /////////////////////////////////
00101 // Routine:  Vgrid_ctor2
00102 // Author:   Nathan Baker
00103 VPUBLIC int Vgrid_ctor2(Vgrid *thee, int nx, int ny, int nz,
00104                           double hx, double hy, double hzed,
00105                           double xmin, double ymin, double zmin,
00106                           double *data) {
00107
00108     if (thee == VNULL) return 0;
00109     thee->nx = nx;
00110     thee->ny = ny;
00111     thee->nz = nz;
00112     thee->hx = hx;
00113     thee->hy = hy;
00114     thee->hzed = hzed;
00115     thee->xmin = xmin;
00116     thee->xmax = xmin + (nx-1)*hx;
00117     thee->ymin = ymin;
00118     thee->ymax = ymin + (ny-1)*hy;
00119     thee->zmin = zmin;
00120     thee->zmax = zmin + (nz-1)*hzed;
00121
00122     if (data == VNULL) {
00123         thee->ctordata = 0;
00124         thee->readdata = 0;
00125     } else {
00126         thee->ctordata = 1;
00127         thee->readdata = 0;
00128         thee->data = data;
00129     }
00130
00131     thee->mem = Vmem_ctor("APBS:VGRID");
00132
00133     Vcompare = pow(10, -1*(VGRID_DIGITS - 2));
00134     sprintf(Vprecision,"%12.%de %12.%de %%12.%de", VGRID_DIGITS,
00135             VGRID_DIGITS, VGRID_DIGITS);
00136
00137     return 1;
00138 }
00139
00140 /* /////////////////////////////////
00141 // Routine:  Vgrid_dtor
00142 // Author:   Nathan Baker
00143 VPUBLIC void Vgrid_dtor(Vgrid **thee) {
00144
00145     if ((*thee) != VNULL) {
00146         Vgrid_dtor2(*thee);
00147         Vmem_free(VNULL, 1, sizeof(Vgrid), (void **)thee);
00148         (*thee) = VNULL;
00149     }
00150 }
00151
00152
00153 /* /////////////////////////////////
00154 // Routine:  Vgrid_dtor2
00155 // Author:   Nathan Baker
00156 VPUBLIC void Vgrid_dtor2(Vgrid *thee) {
00157
00158     if (thee->readdata) {
00159         Vmem_free(thee->mem, (thee->nx*thee->ny*thee->nz), sizeof(double),
00160                     (void **)&(thee->data));
00161     }
00162 }
```

```

00163     Vmem_dtor(&(thee->mem));
00164
00165 }
00166
00167 /* /////////////////////////////////
00168 // Routine: Vgrid_value
00169 // Author: Nathan Baker
00170 VPUBLIC int Vgrid_value(Vgrid *thee, double pt[3], double *value) {
00171
00172     int nx, ny, nz, ihi, jhi, khi, ilo, jlo, klo;
00173     double hx, hy, hzed, xmin, ymin, zmin, ifloat, jfloat, kfloat;
00174     double xmax, ymax, zmax;
00175     double u, dx, dy, dz;
00176
00177     if (thee == VNULL) {
00178         Vnm_print(2, "Vgrid_value: Error -- got VNULL thee!\n");
00179         VASSERT(0);
00180     }
00181     if (!(thee->ctordata || thee->readdata)) {
00182         Vnm_print(2, "Vgrid_value: Error -- no data available!\n");
00183         VASSERT(0);
00184     }
00185 }
00186
00187     nx = thee->nx;
00188     ny = thee->ny;
00189     nz = thee->nz;
00190     hx = thee->hx;
00191     hy = thee->hy;
00192     hzed = thee->hzed;
00193     xmin = thee->xmin;
00194     ymin = thee->ymin;
00195     zmin = thee->zmin;
00196     xmax = thee->xmax;
00197     ymax = thee->ymax;
00198     zmax = thee->zmax;
00199
00200     u = 0;
00201
00202     ifloat = (pt[0] - xmin)/hx;
00203     jfloat = (pt[1] - ymin)/hy;
00204     kfloat = (pt[2] - zmin)/hzed;
00205
00206     ihi = (int)ceil(ifloat);
00207     jhi = (int)ceil(jfloat);
00208     khi = (int)ceil(kfloat);
00209     ilo = (int)floor(ifloat);
00210     jlo = (int)floor(jfloat);
00211     klo = (int)floor(kfloat);
00212     if (VABS(pt[0] - xmin) < Vcompare) ilo = 0;
00213     if (VABS(pt[1] - ymin) < Vcompare) jlo = 0;
00214     if (VABS(pt[2] - zmin) < Vcompare) klo = 0;
00215     if (VABS(pt[0] - xmax) < Vcompare) ihi = nx-1;
00216     if (VABS(pt[1] - ymax) < Vcompare) jhi = ny-1;
00217     if (VABS(pt[2] - zmax) < Vcompare) khi = nz-1;
00218
00219     /* See if we're on the mesh */
00220     if ((ihi<nx) && (jhi<ny) && (khi<nz) &&
00221         (ilo>=0) && (jlo>=0) && (klo>=0)) {
00222
00223         dx = ifloat - (double)(ilo);
00224         dy = jfloat - (double)(jlo);
00225         dz = kfloat - (double)(klo);
00226         u = dx      *dy      *dz      *(thee->data[IJK(ihi,jhi,khi)])
00227             + dx      *(1.0-dy)*dz      *(thee->data[IJK(ihi,jlo,khi)])
00228             + dx      *dy      *(1.0-dz)* (thee->data[IJK(ihi,jhi,klo)])
00229             + dx      *(1.0-dy)*(1.0-dz)* (thee->data[IJK(ihi,jlo,klo)])
00230             + (1.0-dx)*dy      *dz      *(thee->data[IJK(ilohi,jhi,khi)])
00231             + (1.0-dx)*(1.0-dy)*dz      *(thee->data[IJK(ilohi,jlo,khi)])
00232             + (1.0-dx)*dy      *(1.0-dz)* (thee->data[IJK(ilohi,jhi,klo)])
00233             + (1.0-dx)*(1.0-dy)*(1.0-dz)* (thee->data[IJK(ilohi,jlo,klo)]);
00234
00235         *value = u;
00236
00237     if (isnan(u)) {
00238         Vnm_print(2, "Vgrid_value: Got NaN!\n");
00239         Vnm_print(2, "Vgrid_value: (x, y, z) = (%4.3f, %4.3f, %4.3f)\n",
00240             pt[0], pt[1], pt[2]);
00241         Vnm_print(2, "Vgrid_value: (ihi, jhi, khi) = (%d, %d, %d)\n",
00242             ihi, jhi, khi);
00243         Vnm_print(2, "Vgrid_value: (ilo, jlo, klo) = (%d, %d, %d)\n",
00244             ilo, jlo, klo);

```

```

00245     Vnm_print(2, "Vgrid_value: (nx, ny, nz) = (%d, %d, %d)\n",
00246             nx, ny, nz);
00247     Vnm_print(2, "Vgrid_value: (dx, dy, dz) = (%4.3f, %4.3f, %4.3f)\n"
00248             dx, dy, dz);
00249     Vnm_print(2, "Vgrid_value: data[IJK(ihi,jhi,khi)] = %g\n",
00250             theee->data[IJK(ihi,jhi,khi)]);
00251     Vnm_print(2, "Vgrid_value: data[IJK(ihi,jlo,khi)] = %g\n",
00252             theee->data[IJK(ihi,jlo,khi)]);
00253     Vnm_print(2, "Vgrid_value: data[IJK(ihi,jhi,klo)] = %g\n",
00254             theee->data[IJK(ihi,jhi,klo)]);
00255     Vnm_print(2, "Vgrid_value: data[IJK(ihi,jlo,klo)] = %g\n",
00256             theee->data[IJK(ihi,jlo,klo)]);
00257     Vnm_print(2, "Vgrid_value: data[IJK(ilo,jhi,khi)] = %g\n",
00258             theee->data[IJK(ilo,jhi,khi)]);
00259     Vnm_print(2, "Vgrid_value: data[IJK(ilo,jlo,khi)] = %g\n",
00260             theee->data[IJK(ilo,jlo,khi)]);
00261     Vnm_print(2, "Vgrid_value: data[IJK(ilo,jhi,klo)] = %g\n",
00262             theee->data[IJK(ilo,jhi,klo)]);
00263     Vnm_print(2, "Vgrid_value: data[IJK(ilo,jlo,klo)] = %g\n",
00264             theee->data[IJK(ilo,jlo,klo)]);
00265     }
00266     return 1;
00267 }
00268 } else {
00269     *value = 0;
00270     return 0;
00271 }
00272 }
00273 }
00274
00275 return 0;
00276 }
00277 }
00278 /* /////////////////////////////////
00279 // Routine: Vgrid_curvature
00280 // Notes: cflag=0 ==> Reduced Maximal Curvature
00281 //          cflag=1 ==> Mean Curvature (Laplace)
00282 //          cflag=2 ==> Gauss Curvature
00283 //          cflag=3 ==> True Maximal Curvature
00284 // Authors: Stephen Bond and Nathan Baker
00285 VPUBLIC int Vgrid_curvature(Vgrid *thee, double pt[3], int cflag,
00286     double *value) {
00287
00288     double hx, hy, hzed, curv;
00289     double dxx, dyy, dzz;
00290     double uleft, umid, uright, testpt[3];
00291
00292     if (thee == VNULL) {
00293         Vnm_print(2, "Vgrid_curvature: Error -- got VNULL thee!\n");
00294         VASSERT(0);
00295     }
00296     if (!(thee->ctordata || thee->readdata)) {
00297         Vnm_print(2, "Vgrid_curvature: Error -- no data available!\n");
00298         VASSERT(0);
00299     }
00300
00301     hx = thee->hx;
00302     hy = thee->hy;
00303     hzed = thee->hzed;
00304
00305     curv = 0.0;
00306
00307     testpt[0] = pt[0];
00308     testpt[1] = pt[1];
00309     testpt[2] = pt[2];
00310
00311     /* Compute 2nd derivative in the x-direction */
00312     VJMPERR1(Vgrid_value( thee, testpt, &umid));
00313     testpt[0] = pt[0] - hx;
00314     VJMPERR1(Vgrid_value( thee, testpt, &uleft));
00315     testpt[0] = pt[0] + hx;
00316     VJMPERR1(Vgrid_value( thee, testpt, &uright));
00317     testpt[0] = pt[0];
00318
00319     dxx = (uright - 2*umid + uleft)/(hx*hx);
00320
00321     /* Compute 2nd derivative in the y-direction */
00322
00323     /* Compute 2nd derivative in the z-direction */
00324
00325 }
```

```

00326 VJMPERR1(Vgrid_value( thee, testpt, &umid));
00327 testpt[1] = pt[1] - hy;
00328 VJMPERR1(Vgrid_value( thee, testpt, &uleft));
00329 testpt[1] = pt[1] + hy;
00330 VJMPERR1(Vgrid_value( thee, testpt, &uright));
00331 testpt[1] = pt[1];
00332
00333 dyx = (uright - 2*umid + uleft)/(hy*hy);
00334
00335 /* Compute 2nd derivative in the z-direction */
00336 VJMPERR1(Vgrid_value( thee, testpt, &umid));
00337 testpt[2] = pt[2] - hzed;
00338 VJMPERR1(Vgrid_value( thee, testpt, &uleft));
00339 testpt[2] = pt[2] + hzed;
00340 VJMPERR1(Vgrid_value( thee, testpt, &uright));
00341
00342 dzx = (uright - 2*umid + uleft)/(hzed*hzed);
00343
00344 if ( cflag == 0 ) {
00345     curv = fabs(dxx);
00346     curv = ( curv > fabs(dyx) ) ? curv : fabs(dyx);
00347     curv = ( curv > fabs(dzx) ) ? curv : fabs(dzx);
00348 } else if ( cflag == 1 ) {
00349     curv = (dxx + dyx + dzx)/3.0;
00350 } else {
00351     Vnm_print(2, "Vgrid_curvature: support for cflag = %d not available!\n",
00352     ", cflag);
00353     VASSERT( 0 ); /* Feature Not Coded Yet! */
00354 }
00355
00356 *value = curv;
00357 return 1;
00358
00359 VERROR1:
00360     return 0;
00361
00362 }
00363
00364 /* /////////////////////////////////
00365 // Routine: Vgrid_gradient
00366 //
00367 // Authors: Nathan Baker and Stephen Bond
00368 VPUBLIC int Vgrid_gradient(Vgrid *thee, double pt[3], double grad[3]) {
00369
00370     double hx, hy, hzed;
00371     double uleft, umid, uright, testpt[3];
00372     int haveleft, haverright;
00373
00374     if (thee == VNULL) {
00375         Vnm_print(2, "Vgrid_gradient: Error -- got VNULL thee!\n");
00376         VASSERT(0);
00377     }
00378     if (!(thee->ctordata || thee->readdata)) {
00379         Vnm_print(2, "Vgrid_gradient: Error -- no data available!\n");
00380         VASSERT(0);
00381     }
00382 }
00383
00384 hx = thee->hx;
00385 hy = thee->hy;
00386 hzed = thee->hzed;
00387
00388 /* Compute derivative in the x-direction */
00389 testpt[0] = pt[0];
00390 testpt[1] = pt[1];
00391 testpt[2] = pt[2];
00392 VJMPERR1( Vgrid_value( thee, testpt, &umid));
00393 testpt[0] = pt[0] - hx;
00394 if (Vgrid_value( thee, testpt, &uleft)) haveleft = 1;
00395 else haveleft = 0;
00396 testpt[0] = pt[0] + hx;
00397 if (Vgrid_value( thee, testpt, &uright)) haverright = 1;
00398 else haverright = 0;
00399 if (haverright && haveleft) grad[0] = (uright - uleft)/(2*hx);
00400 else if (haverright) grad[0] = (uright - umid)/hx;
00401 else if (haveleft) grad[0] = (umid - uleft)/hx;
00402 else VJMPERR1(0);
00403
00404 /* Compute derivative in the y-direction */
00405 testpt[0] = pt[0];
00406 testpt[1] = pt[1];

```

```

00407     testpt[2] = pt[2];
00408     VJMPERR1(Vgrid_value(thee, testpt, &umid));
00409     testpt[1] = pt[1] - hy;
00410     if (Vgrid_value( thee, testpt, &uleft)) haveleft = 1;
00411     else haveleft = 0;
00412     testpt[1] = pt[1] + hy;
00413     if (Vgrid_value( thee, testpt, &uright)) haveright = 1;
00414     else haveright = 0;
00415     if (haveright && haveleft) grad[1] = (uright - uleft)/(2*hy);
00416     else if (haveright) grad[1] = (uright - umid)/hy;
00417     else if (haveleft) grad[1] = (umid - uleft)/hy;
00418     else VJMPERR1(0);
00419
00420     /* Compute derivative in the z-direction */
00421     testpt[0] = pt[0];
00422     testpt[1] = pt[1];
00423     testpt[2] = pt[2];
00424     VJMPERR1(Vgrid_value(thee, testpt, &umid));
00425     testpt[2] = pt[2] - hzed;
00426     if (Vgrid_value( thee, testpt, &uleft)) haveleft = 1;
00427     else haveleft = 0;
00428     testpt[2] = pt[2] + hzed;
00429     if (Vgrid_value( thee, testpt, &uright)) haveright = 1;
00430     else haveright = 0;
00431     if (haveright && haveleft) grad[2] = (uright - uleft)/(2*hzed);
00432     else if (haveright) grad[2] = (uright - umid)/hzed;
00433     else if (haveleft) grad[2] = (umid - uleft)/hzed;
00434     else VJMPERR1(0);
00435
00436     return 1;
00437
00438 VERROR1:
00439     return 0;
00440
00441 }
00442
00443 /* /////////////////////////////////
00444 // Routine: Vgrid_readGZ
00445 //
00446 // Author: David Gohara
00447 #ifdef HAVE_ZLIB
00448 #define off_t long
00449 #include "../../../contrib/zlib/zlib.h"
00450 #endif
00451 VPUBLIC int Vgrid_readGZ(Vgrid *thee, const char *fname) {
00452
00453 #ifdef HAVE_ZLIB
00454     int i, j, k;
00455     int len; // Temporary counter variable for loop conditionals
00456     int q, itmp, u, header, incr;
00457     double *temp;
00458     double dtmpl, dtmp2, dtmp3;
00459     gzFile infile;
00460     char line[VMAX_ARGLEN];
00461     header = 0;
00462
00463     /* Check to see if the existing data is null and, if not, clear it out */
00464     if (thee->data != VNULL) {
00465         Vnm_print(1, "%s: destroying existing data!\n", __func__);
00466         Vmem_free(thee->mem, thee->nx * thee->ny * thee->nz, sizeof(double),
00467                   (void **)&(thee->data));
00468     }
00469
00470     thee->readdata = 1;
00471     thee->ctordata = 0;
00472
00473     infile = gzopen(fname, "rb");
00474     if (infile == Z_NULL) {
00475         Vnm_print(2, "%s: Problem opening compressed file %s\n", __func__, fname);
00476         return VRC_FAILURE;
00477     }
00478
00479     thee->hx = 0.0;
00480     thee->hy = 0.0;
00481     thee->hzed = 0.0;
00482
00483     //read data here
00484     while (header < 7) {
00485         if(gzgets(infile, line, VMAX_ARGLEN) == Z_NULL) {
00486             return VRC_FAILURE;

```

```

00489 }
00490 // Skip comments and newlines
00491 if(strncmp(line, "#", 1) == 0) continue;
00492 if(line[0] == '\n') continue;
00493
00494 switch (header) {
00495 case 0:
00496     sscanf(line, "object 1 class gridpositions counts %d %d %d",
00497             &(thee->nx), &(thee->ny), &(thee->nz));
00498     break;
00499 case 1:
00500     sscanf(line, "origin %lf %lf %lf",
00501             &(thee->xmin), &(thee->ymin), &(thee->zmin));
00502     break;
00503 case 2:
00504 case 3:
00505 case 4:
00506     sscanf(line, "delta %lf %lf %lf", &dtmpl1, &dtmpl2, &dtmpl3);
00507     thee->hx += dtmpl1;
00508     thee->hy += dtmpl2;
00509     thee->hzed += dtmpl3;
00510     break;
00511 default:
00512     break;
00513 }
00514 }
00515 header++;
00516 }
00517 */
00518 /* Allocate space for the data */
00519 Vnm_print(0, "%s: allocating %d x %d x %d doubles for storage\n",
00520 __func__, thee->nx, thee->ny, thee->nz);
00521 len = thee->nx * thee->ny * thee->nz;
00522
00523 thee->data = VNULL;
00524 thee->data = Vmem_malloc(thee->mem, len, sizeof(double));
00525 if (thee->data == VNULL) {
00526     Vnm_print(2, "%s: Unable to allocate space for data!\n", __func__);
00527     return 0;
00528 }
00529 */
00530 /* Allocate a temporary buffer to store the compressed
00531 * data into (column major order). Add 2 to ensure the buffer is
00532 * big enough to take extra data on the final read loop.
00533 */
00534 temp = (double *)malloc(len * (2 * sizeof(double)));
00535
00536 for (i = 0; i < len; i += 3){
00537     memset(&line, 0, sizeof(line));
00538     gzgets(infile, line, VMAX_ARGLEN);
00539     sscanf(line, "%lf %lf %lf", &temp[i], &temp[i+1], &temp[i+2]);
00540 }
00541
00542 /* Now move the data to row major order */
00543 incr = 0;
00544 for (i=0; i<thee->nx; i++) {
00545     for (j=0; j<thee->ny; j++) {
00546         for (k=0; k<thee->nz; k++) {
00547             u = k*(thee->nx)*(thee->ny)+j*(thee->nx)+i;
00548             (thee->data)[u] = temp[incr++];
00549         }
00550     }
00551 }
00552 }
00553
00554 /* calculate grid maxima */
00555 thee->xmax = thee->xmin + (thee->nx-1)*thee->hx;
00556 thee->ymax = thee->ymin + (thee->ny-1)*thee->hy;
00557 thee->zmax = thee->zmin + (thee->nz-1)*thee->hzed;
00558
00559 /* Close off the socket */
00560 gzclose(infile);
00561 free(temp);
00562 #else
00563
00564 Vnm_print(0, "WARNING\n");
00565 Vnm_print(0, "Vgrid_readGZ: gzip read/write support is disabled in this build
00566 \n");
00567 Vnm_print(0, "Vgrid_readGZ: configure and compile without the --disable-zlib
flag.\n");
00568 Vnm_print(0, "WARNING\n");

```

```

00568 #endif
00569     return VRC_SUCCESS;
00570 }
00571
00576 VPUBLIC int Vgrid_readDX(Vgrid *thee,
00577                             const char *iodev,
00578                             const char *iofmt,
00579                             const char *thost,
00580                             const char *fname
00581                             ) {
00582
00583     int i,
00584         j,
00585         k,
00586         itmp,
00587         u;
00588     double dtmp;
00589     char tok[VMAX_BUFSIZE];
00590     Vio *sock;
00591
00592     /* Check to see if the existing data is null and, if not, clear it out */
00593     if (thee->data != VNULL) {
00594         Vnm_print(1, "Vgrid_readDX: destroying existing data!\n");
00595     Vmem_free(thee->mem, (thee->nx*thee->ny*thee->nz), sizeof(double),
00596             (void **)&(thee->data));
00597     thee->readdata = 1;
00598     thee->ctordata = 0;
00599
00600     /* Set up the virtual socket */
00601     sock = Vio_ctor(iodev, iofmt, thost, fname, "r");
00602     if (sock == VNULL) {
00603         Vnm_print(2, "Vgrid_readDX: Problem opening virtual socket %s\n",
00604                 fname);
00605         return 0;
00606     }
00607     if (Vio_accept(sock, 0) < 0) {
00608         Vnm_print(2, "Vgrid_readDX: Problem accepting virtual socket %s\n",
00609                 fname);
00610         return 0;
00611     }
00612
00613     Vio_setWhiteChars(sock, MCwhiteChars);
00614     Vio_setCommChars(sock, MCcommChars);
00615
00616     /* Read in the DX regular positions */
00617     /* Get "object" */
00618     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00619     VJMPERR1(!strcmp(tok, "object"));
00620     /* Get "1" */
00621     VJMPERR2(l == Vio_scanf(sock, "%d", &itmp));
00622     /* Get "class" */
00623     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00624     VJMPERR1(!strcmp(tok, "class"));
00625     /* Get "gridpositions" */
00626     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00627     VJMPERR1(!strcmp(tok, "gridpositions"));
00628     /* Get "counts" */
00629     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00630     VJMPERR1(!strcmp(tok, "counts"));
00631     /* Get nx */
00632     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00633     VJMPERR1(l == sscanf(tok, "%d", &(thee->nx)));
00634     /* Get ny */
00635     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00636     VJMPERR1(l == sscanf(tok, "%d", &(thee->ny)));
00637     /* Get nz */
00638     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00639     VJMPERR1(l == sscanf(tok, "%d", &(thee->nz)));
00640     Vnm_print(0, "Vgrid_readDX: Grid dimensions %d x %d x %d grid\n",
00641             thee->nx, thee->ny, thee->nz);
00642     /* Get "origin" */
00643     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00644     VJMPERR1(!strcmp(tok, "origin"));
00645     /* Get xmin */
00646     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00647     VJMPERR1(l == sscanf(tok, "%lf", &(thee->xmin)));
00648     /* Get ymin */
00649     VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00650     VJMPERR1(l == sscanf(tok, "%lf", &(thee->ymin)));
00651     /* Get zmin */
00652     VJMPERR2(l == Vio_scanf(sock, "%s", tok));

```

```

00653 VJMPERR1(l == sscanf(tok, "%lf", &(thee->zmin)));
00654 Vnm_print(0, "Vgrid_readDX: Grid origin = (%g, %g, %g)\n",
00655     thee->xmin, thee->ymin, thee->zmin);
00656 /* Get "delta" */
00657 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00658 VJMPERR1(!strcmp(tok, "delta"));
00659 /* Get hx */
00660 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00661 VJMPERR1(l == sscanf(tok, "%lf", &(thee->hx)));
00662 /* Get 0.0 */
00663 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00664 VJMPERR1(l == sscanf(tok, "%lf", &dtmp));
00665 VJMPERR1(dtmp == 0.0);
00666 /* Get 0.0 */
00667 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00668 VJMPERR1(l == sscanf(tok, "%lf", &dtmp));
00669 VJMPERR1(dtmp == 0.0);
00670 /* Get "delta" */
00671 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00672 VJMPERR1(!strcmp(tok, "delta"));
00673 /* Get 0.0 */
00674 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00675 VJMPERR1(l == sscanf(tok, "%lf", &dtmp));
00676 VJMPERR1(dtmp == 0.0);
00677 /* Get hy */
00678 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00679 VJMPERR1(l == sscanf(tok, "%lf", &(thee->hy)));
00680 /* Get 0.0 */
00681 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00682 VJMPERR1(l == sscanf(tok, "%lf", &dtmp));
00683 VJMPERR1(dtmp == 0.0);
00684 /* Get "delta" */
00685 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00686 VJMPERR1(!strcmp(tok, "delta"));
00687 /* Get 0.0 */
00688 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00689 VJMPERR1(l == sscanf(tok, "%lf", &dtmp));
00690 VJMPERR1(dtmp == 0.0);
00691 /* Get 0.0 */
00692 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00693 VJMPERR1(l == sscanf(tok, "%lf", &dtmp));
00694 VJMPERR1(dtmp == 0.0);
00695 /* Get hz */
00696 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00697 VJMPERR1(l == sscanf(tok, "%lf", &(thee->hzed)));
00698 Vnm_print(0, "Vgrid_readDX: Grid spacings = (%g, %g, %g)\n",
00699     thee->hx, thee->hy, thee->hzed);
00700 /* Get "object" */
00701 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00702 VJMPERR1(!strcmp(tok, "object"));
00703 /* Get "2" */
00704 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00705 /* Get "class" */
00706 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00707 VJMPERR1(!strcmp(tok, "class"));
00708 /* Get "gridconnections" */
00709 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00710 VJMPERR1(!strcmp(tok, "gridconnections"));
00711 /* Get "counts" */
00712 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00713 VJMPERR1(!strcmp(tok, "counts"));
00714 /* Get the dimensions again */
00715 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00716 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00717 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00718 /* Get "object" */
00719 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00720 VJMPERR1(!strcmp(tok, "object"));
00721 /* Get # */
00722 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00723 /* Get "class" */
00724 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00725 VJMPERR1(!strcmp(tok, "class"));
00726 /* Get "array" */
00727 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00728 VJMPERR1(!strcmp(tok, "array"));
00729 /* Get "type" */
00730 VJMPERR2(l == Vio_scanf(sock, "%s", tok));
00731 VJMPERR1(!strcmp(tok, "type"));
00732 /* Get "double" */
00733 VJMPERR2(l == Vio_scanf(sock, "%s", tok));

```

```

00734     VJMPERR1(!strcmp(tok, "double"));
00735     /* Get "rank" */
00736     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00737     VJMPERR1(!strcmp(tok, "rank"));
00738     /* Get # */
00739     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00740     /* Get "items" */
00741     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00742     VJMPERR1(!strcmp(tok, "items"));
00743     /* Get # */
00744     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00745     VJMPERR1(1 == sscanf(tok, "%d", &itmp));
00746     VJMPERR1(((thee->nx)*(thee->ny)*(thee->nz)) == itmp);
00747     /* Get "data" */
00748     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00749     VJMPERR1(!strcmp(tok, "data"));
00750     /* Get "follows" */
00751     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00752     VJMPERR1(!strcmp(tok, "follows"));

00753
00754     /* Allocate space for the data */
00755     Vnm_print(0, "Vgrid_readDX: allocating %d x %d x %d doubles for storage\n"
00756
00757         thee->nx, thee->ny, thee->nz);
00758         thee->data = VNULL;
00759         thee->data = (double*)Vmem_malloc(thee->mem, (thee->nx)*(thee->ny)
00760             *(thee->nz),
00761             sizeof(double));
00762         if (thee->data == VNULL) {
00763             Vnm_print(2, "Vgrid_readDX: Unable to allocate space for data!\n");
00764             return 0;
00765         }
00766         for (i=0; i<thee->nx; i++) {
00767             for (j=0; j<thee->ny; j++) {
00768                 for (k=0; k<thee->nz; k++) {
00769                     u = k*(thee->nx)*(thee->ny)+j*(thee->nx)+i;
00770                     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00771                     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00772                     (thee->data)[u] = dtmp;
00773                 }
00774             }
00775         }
00776         /* calculate grid maxima */
00777         thee->xmax = thee->xmin + (thee->nx-1)*thee->hx;
00778         thee->ymax = thee->ymin + (thee->ny-1)*thee->hy;
00779         thee->zmax = thee->zmin + (thee->nz-1)*thee->hzed;
00780
00781         /* Close off the socket */
00782         Vio_acceptFree(sock);
00783         Vio_dtor(&sock);
00784
00785         return 1;
00786
00787     VERROR1:
00788         Vio_dtor(&sock);
00789         Vnm_print(2, "Vgrid_readDX: Format problem with input file <%s>\n",
00790             fname);
00791         return 0;
00792
00793     VERROR2:
00794         Vio_dtor(&sock);
00795         Vnm_print(2, "Vgrid_readDX: I/O problem with input file <%s>\n",
00796             fname);
00797         return 0;
00798
00799
00800
00801 }
00802
00803 /* /////////////////////////////////
00804 // Routine: Vgrid_writeGZ
00805 //
00806 // Author: Nathan Baker
00807 VPUBLIC void Vgrid_writeGZ(Vgrid *thee, const char *iodev, const char *iofmt,
00808     const char *host, const char *fname, char *title, double *pvec) {
00809
00810 #ifdef HAVE_ZLIB
00811     double xmin, ymin, zmin, hx, hy, hzed;
00812
00813

```

```

00814 int nx, ny, nz;
00815 int icol, i, j, k, u, usepart, nxPART, nyPART, nzPART, gotit;
00816 double x, y, z, xminPART, yminPART, zminPART;
00817
00818 int txyz;
00819 double txmin, tymin, tzmin;
00820
00821 char header[8196];
00822 char footer[8196];
00823 char line[80];
00824 char newline[] = "\n";
00825 gzfile outfile;
00826 char precFormat[VMAX_BUFSIZE];
00827
00828 if (thee == VNULL) {
00829   Vnm_print(2, "Vgrid_writeGZ: Error -- got VNULL thee!\n");
00830   VASSERT(0);
00831 }
00832 if (!(thee->ctordata || thee->readdata)) {
00833   Vnm_print(2, "Vgrid_writeGZ: Error -- no data available!\n");
00834   VASSERT(0);
00835 }
00836
00837 hx = thee->hx;
00838 hy = thee->hy;
00839 hzed = thee->hzed;
00840 nx = thee->nx;
00841 ny = thee->ny;
00842 nz = thee->nz;
00843 xmin = thee->xmin;
00844 ymin = thee->ymin;
00845 zmin = thee->zmin;
00846
00847 if (pvec == VNULL) usepart = 0;
00848 else usepart = 1;
00849
00850 /* Set up the virtual socket */
00851 Vnm_print(0, "Vgrid_writeGZ: Opening file...\n");
00852 outfile = gzopen(fname, "wb");
00853
00854 if (usepart) {
00855   /* Get the lower corner and number of grid points for the local
00856    * partition */
00857   xminPART = VLARGE;
00858   yminPART = VLARGE;
00859   zminPART = VLARGE;
00860   nxPART = 0;
00861   nyPART = 0;
00862   nzPART = 0;
00863   /* First, search for the lower corner */
00864   for (k=0; k<nz; k++) {
00865     z = k*hzed + zmin;
00866     for (j=0; j<ny; j++) {
00867       y = j*hy + ymin;
00868       for (i=0; i<nx; i++) {
00869         x = i*hx + xmin;
00870         if (pvec[IJK(i,j,k)] > 0.0) {
00871           if (x < xminPART) xminPART = x;
00872           if (y < yminPART) yminPART = y;
00873           if (z < zminPART) zminPART = z;
00874         }
00875       }
00876     }
00877   }
00878   /* Now search for the number of grid points in the z direction */
00879   for (k=0; k<nz; k++) {
00880     gotit = 0;
00881     for (j=0; j<ny; j++) {
00882       for (i=0; i<nx; i++) {
00883         if (pvec[IJK(i,j,k)] > 0.0) {
00884           gotit = 1;
00885           break;
00886         }
00887       }
00888       if (gotit) break;
00889     }
00890     if (gotit) nzPART++;
00891   }
00892   /* Now search for the number of grid points in the y direction */
00893   for (j=0; j<ny; j++) {
00894     gotit = 0;

```

```

00895     for (k=0; k<nz; k++) {
00896         for (i=0; i<nx; i++) {
00897             if (pvec[IJK(i,j,k)] > 0.0) {
00898                 gotit = 1;
00899                 break;
00900             }
00901         }
00902         if (gotit) break;
00903     }
00904     if (gotit) nyPART++;
00905 }
00906 /* Now search for the number of grid points in the x direction */
00907 for (i=0; i<nx; i++) {
00908     gotit = 0;
00909     for (k=0; k<nz; k++) {
00910         for (j=0; j<ny; j++) {
00911             if (pvec[IJK(i,j,k)] > 0.0) {
00912                 gotit = 1;
00913                 break;
00914             }
00915         }
00916         if (gotit) break;
00917     }
00918     if (gotit) nxPART++;
00919 }
00920
00921 if ((nxPART != nx) || (nyPART != ny) || (nzPART != nz)) {
00922     Vnm_pprint(0, "Vgrid_writeGZ: printing only subset of domain\n");
00923 }
00924
00925 txyz = (nxPART*nyPART*nzPART);
00926 txmin = xminPART;
00927 tymin = yminPART;
00928 tzmin = zminPART;
00929
00930 }else {
00931
00932     txyz = (nx*ny*nz);
00933     txmin = xmin;
00934     tymin = ymin;
00935     tzmin = zmin;
00936
00937 }
00938
00939 /* Write off the title (if we're not XDR) */
00940 sprintf(header,
00941     "# Data from %s\n"
00942     "# \n"
00943     "# %s\n"
00944     "# \n"
00945     "object 1 class gridpositions counts %i %i %i\n"
00946     "origin %12.6e %12.6e %12.6e\n"
00947     "delta %12.6e 0.000000e+00 0.000000e+00\n" \
00948     "delta 0.000000e+00 %12.6e 0.000000e+00\n" \
00949     "delta 0.000000e+00 0.000000e+00 %12.6e\n"
00950     "object 2 class gridconnections counts %i %i %i\n"
00951     "object 3 class array type double rank 0 items %i data follows\n",
00952     PACKAGE_STRING,title,nx,ny,nz,txmin,tymin,tzmin,
00953     hx,hy,hzed,nx,ny,nz,txyz);
00954 gzwrite(outfile, header, strlen(header)*sizeof(char));
00955
00956 /* Now write the data */
00957 icol = 0;
00958 for (i=0; i<nx; i++) {
00959     for (j=0; j<ny; j++) {
00960         for (k=0; k<nz; k++) {
00961             u = k*(nx)*(ny)+j*(nx)+i;
00962             if (pvec[u] > 0.0) {
00963                 sprintf(line, "%12.6e ", thee->data[u]);
00964                 gzwrite(outfile, line, strlen(line)*sizeof(char));
00965                 icol++;
00966                 if (icol == 3) {
00967                     icol = 0;
00968                     gzwrite(outfile, newline, strlen(newline)*sizeof(char));
00969                 }
00970             }
00971         }
00972     }
00973 }
00974 if(icol < 3){
00975     char newline[] = "\n";

```

```

00976     gzwrite(outfile, newline, strlen(newline)*sizeof(char));
00977 }
00978
00979 /* Create the field */
00980 sprintf(footer, "attribute \"dep\" string \"positions\"\n" \
00981   "object \"regular positions regular connections\" class field\n" \
00982   "component \"positions\" value 1\n" \
00983   "component \"connections\" value 2\n" \
00984   "component \"data\" value 3\n");
00985 gzwrite(outfile, footer, strlen(footer)*sizeof(char));
00986
00987 gzclose(outfile);
00988 #else
00989
00990 Vnm_print(0, "WARNING\n");
00991 Vnm_print(0, "Vgrid_readGZ: gzip read/write support is disabled in this build
\\n");
00992 Vnm_print(0, "Vgrid_readGZ: configure and compile without the --disable-zlib
flag.\n");
00993 Vnm_print(0, "WARNING\n");
00994 #endif
00995 }
00996
00997 /* /////////////////////////////////
00998 // Routine: Vgrid_writeDX
00999 //
01000 // Author: Nathan Baker
01002 VPUBLIC void Vgrid_writeDX(Vgrid *thee, const char *iodev, const char *iofmt,
01003   const char *thost, const char *fname, char *title, double *pvec) {
01004
01005   double xmin, ymin, zmin, hx, hy, hzed;
01006   int nx, ny, nz;
01007   int icol, i, j, k, u, usepart, nxPART, nyPART, nzPART, gotit;
01008   double x, y, z, xminPART, yminPART, zminPART;
01009   Vio *sock;
01010   char precFormat[VMAX_BUFSIZE];
01011
01012   if (thee == VNULL) {
01013     Vnm_print(2, "Vgrid_writeDX: Error -- got VNULL thee!\n");
01014     VASSERT(0);
01015   }
01016   if (!thee->ctordata || thee->readdata) {
01017     Vnm_print(2, "Vgrid_writeDX: Error -- no data available!\n");
01018     VASSERT(0);
01019   }
01020
01021   hx = thee->hx;
01022   hy = thee->hy;
01023   hzed = thee->hzed;
01024   nx = thee->nx;
01025   ny = thee->ny;
01026   nz = thee->nz;
01027   xmin = thee->xmin;
01028   ymin = thee->ymin;
01029   zmin = thee->zmin;
01030
01031   if (pvec == VNULL) usepart = 0;
01032   else usepart = 1;
01033
01034   /* Set up the virtual socket */
01035   Vnm_print(0, "Vgrid_writeDX: Opening virtual socket...\n");
01036   sock = Vio_ctor(iodev, iofmt, thost, fname, "w");
01037   if (sock == VNULL) {
01038     Vnm_print(2, "Vgrid_writeDX: Problem opening virtual socket %s\n",
01039       fname);
01040     return;
01041   }
01042   if (Vio_connect(sock, 0) < 0) {
01043     Vnm_print(2, "Vgrid_writeDX: Problem connecting virtual socket %s\n",
01044       fname);
01045     return;
01046   }
01047
01048   Vio_setWhiteChars(sock, MCwhiteChars);
01049   Vio_setCommChars(sock, MCcommChars);
01050
01051   Vnm_print(0, "Vgrid_writeDX: Writing to virtual socket...\n");
01052
01053   if (usepart) {
01054     /* Get the lower corner and number of grid points for the local
01055      * partition */

```

```

01056     xminPART = VLARGE;
01057     yminPART = VLARGE;
01058     zminPART = VLARGE;
01059     nxPART = 0;
01060     nyPART = 0;
01061     nzPART = 0;
01062     /* First, search for the lower corner */
01063     for (k=0; k<nz; k++) {
01064         z = k*hzed + zmin;
01065         for (j=0; j<ny; j++) {
01066             y = j*hy + ymin;
01067             for (i=0; i<nx; i++) {
01068                 x = i*hx + xmin;
01069                 if (pvec[IJK(i,j,k)] > 0.0) {
01070                     if (x < xminPART) xminPART = x;
01071                     if (y < yminPART) yminPART = y;
01072                     if (z < zminPART) zminPART = z;
01073                 }
01074             }
01075         }
01076     }
01077     /* Now search for the number of grid points in the z direction */
01078     for (k=0; k<nz; k++) {
01079         gotit = 0;
01080         for (j=0; j<ny; j++) {
01081             for (i=0; i<nx; i++) {
01082                 if (pvec[IJK(i,j,k)] > 0.0) {
01083                     gotit = 1;
01084                     break;
01085                 }
01086             }
01087             if (gotit) break;
01088         }
01089         if (gotit) nzPART++;
01090     }
01091     /* Now search for the number of grid points in the y direction */
01092     for (j=0; j<ny; j++) {
01093         gotit = 0;
01094         for (k=0; k<nz; k++) {
01095             for (i=0; i<nx; i++) {
01096                 if (pvec[IJK(i,j,k)] > 0.0) {
01097                     gotit = 1;
01098                     break;
01099                 }
01100             }
01101             if (gotit) break;
01102         }
01103         if (gotit) nyPART++;
01104     }
01105     /* Now search for the number of grid points in the x direction */
01106     for (i=0; i<nx; i++) {
01107         gotit = 0;
01108         for (k=0; k<nz; k++) {
01109             for (j=0; j<ny; j++) {
01110                 if (pvec[IJK(i,j,k)] > 0.0) {
01111                     gotit = 1;
01112                     break;
01113                 }
01114             }
01115             if (gotit) break;
01116         }
01117         if (gotit) nxPART++;
01118     }
01119
01120     if ((nxPART != nx) || (nyPART != ny) || (nzPART != nz)) {
01121         Vnm_print(0, "Vgrid_writeDX: printing only subset of domain\n");
01122     }
01123
01124
01125     /* Write off the title (if we're not XDR) */
01126     if (Vstring_strcasecmp(iofmt, "XDR") == 0) {
01127         Vnm_print(0, "Vgrid_writeDX: Skipping comments for XDR format.\n");
01128     }
01129     else {
01130         Vnm_print(0, "Vgrid_writeDX: Writing comments for %s format.\n",
01131                   iofmt);
01132         Vio_printf(sock, "# Data from %s\n", PACKAGE_STRING);
01133         Vio_printf(sock, "#\n");
01134         Vio_printf(sock, "# %s\n", title);
01135         Vio_printf(sock, "#\n");
01136     }
01137

```

```

01136
01137     /* Write off the DX regular positions */
01138     Vio_printf(sock, "object 1 class gridpositions counts %d %d %d\n",
01139                 nxPART, nyPART, nzPART);
01140
01141     sprintf(precFormat, Vprecision, xminPART, yminPART, zminPART);
01142     Vio_printf(sock, "origin %s\n", precFormat);
01143     sprintf(precFormat, Vprecision, hx, 0.0, 0.0);
01144     Vio_printf(sock, "delta %s\n", precFormat);
01145     sprintf(precFormat, Vprecision, 0.0, hy, 0.0);
01146     Vio_printf(sock, "delta %s\n", precFormat);
01147     sprintf(precFormat, Vprecision, 0.0, 0.0, hzed);
01148     Vio_printf(sock, "delta %s\n", precFormat);
01149
01150     /* Write off the DX regular connections */
01151     Vio_printf(sock, "object 2 class gridconnections counts %d %d %d\n",
01152                 nxPART, nyPART, nzPART);
01153
01154     /* Write off the DX data */
01155     Vio_printf(sock, "object 3 class array type double rank 0 items %d \
01156 data follows\n", (nxPART*nyPART*nzPART));
01157     icol = 0;
01158     for (i=0; i<nx; i++) {
01159         for (j=0; j<ny; j++) {
01160             for (k=0; k<nz; k++) {
01161                 u = k*(nx)*(ny)+j*(nx)+i;
01162                 if (pvec[u] > 0.0) {
01163                     Vio_printf(sock, "%12.6e ", thee->data[u]);
01164                     icol++;
01165                     if (icol == 3) {
01166                         icol = 0;
01167                         Vio_printf(sock, "\n");
01168                     }
01169                 }
01170             }
01171         }
01172     }
01173
01174     if (icol != 0) Vio_printf(sock, "\n");
01175
01176     /* Create the field */
01177     Vio_printf(sock, "attribute \"dep\" string \"positions\"\n");
01178     Vio_printf(sock, "object \"regular positions regular connections\" \
01179 class field\n");
01180     Vio_printf(sock, "component \"positions\" value 1\n");
01181     Vio_printf(sock, "component \"connections\" value 2\n");
01182     Vio_printf(sock, "component \"data\" value 3\n");
01183
01184 } else {
01185     /* Write off the title (if we're not XDR) */
01186     if (Vstring_strcasecmp(iofmt, "XDR") == 0) {
01187         Vnm_print(0, "Vgrid_writeDX: Skipping comments for XDR format.\n");
01188     } else {
01189         Vnm_print(0, "Vgrid_writeDX: Writing comments for %s format.\n",
01190                   iofmt);
01191         Vio_printf(sock, "# Data from %s\n", PACKAGE_STRING);
01192         Vio_printf(sock, "#\n");
01193         Vio_printf(sock, "# %s\n", title);
01194         Vio_printf(sock, "#\n");
01195     }
01196
01197
01198     /* Write off the DX regular positions */
01199     Vio_printf(sock, "object 1 class gridpositions counts %d %d %d\n",
01200                 nx, ny, nz);
01201
01202     sprintf(precFormat, Vprecision, xmin, ymin, zmin);
01203     Vio_printf(sock, "origin %s\n", precFormat);
01204     sprintf(precFormat, Vprecision, hx, 0.0, 0.0);
01205     Vio_printf(sock, "delta %s\n", precFormat);
01206     sprintf(precFormat, Vprecision, 0.0, hy, 0.0);
01207     Vio_printf(sock, "delta %s\n", precFormat);
01208     sprintf(precFormat, Vprecision, 0.0, 0.0, hzed);
01209     Vio_printf(sock, "delta %s\n", precFormat);
01210
01211     /* Write off the DX regular connections */
01212     Vio_printf(sock, "object 2 class gridconnections counts %d %d %d\n",
01213                 nx, ny, nz);
01214
01215     /* Write off the DX data */

```

```

01216     Vio_printf(sock, "object 3 class array type double rank 0 items %d \
01217 data follows\n", (nx*ny*nz));
01218     icol = 0;
01219     for (i=0; i<nx; i++) {
01220         for (j=0; j<ny; j++) {
01221             for (k=0; k<nz; k++) {
01222                 u = k*(nx)*(ny)+j*(nx)+i;
01223                 Vio_printf(sock, "%12.6e ", thee->data[u]);
01224                 icol++;
01225                 if (icol == 3) {
01226                     icol = 0;
01227                     Vio_printf(sock, "\n");
01228                 }
01229             }
01230         }
01231     }
01232     if (icol != 0) Vio_printf(sock, "\n");
01233
01234     /* Create the field */
01235     Vio_printf(sock, "attribute \"dep\" string \"positions\"\n");
01236     Vio_printf(sock, "object \"regular positions regular connections\" \
01237 class field\n");
01238     Vio_printf(sock, "component \"positions\" value 1\n");
01239     Vio_printf(sock, "component \"connections\" value 2\n");
01240     Vio_printf(sock, "component \"data\" value 3\n");
01241 }
01242
01243 /* Close off the socket */
01244 Vio_connectFree(sock);
01245 Vio_dtor(&sock);
01246 }
01247
01248 /* /////////////////////////////////
01249 // Routine: Vgrid_writeUHBD
01250 // Author: Nathan Baker
01251 VPUBLIC void Vgrid_writeUHBD(Vgrid *thee, const char *iodev, const char *iofmt,
01252     const char *thost, const char *fname, char *title, double *pvec) {
01253
01254     int icol, i, j, k, u, nx, ny, nz, gotit;
01255     double xmin, ymin, zmin, hzed, hy, hx;
01256     Vio *sock;
01257
01258     if (thee == VNULL) {
01259         Vnm_print(2, "Vgrid_writeUHBD: Error -- got VNULL thee!\n");
01260         VASSERT(0);
01261     }
01262     if (!(thee->ctordata || thee->readdata)) {
01263         Vnm_print(2, "Vgrid_writeUHBD: Error -- no data available!\n");
01264         VASSERT(0);
01265     }
01266
01267     if ((thee->hx!=thee->hy) || (thee->hy!=thee->hzed)
01268         || (thee->hx!=thee->hzed)) {
01269         Vnm_print(2, "Vgrid_writeUHBD: can't write UHBD mesh with non-uniform \
01270 spacing\n");
01271     return;
01272 }
01273
01274     /* Set up the virtual socket */
01275     sock = Vio_ctor(iodev, iofmt, thost, fname, "w");
01276     if (sock == VNULL) {
01277         Vnm_print(2, "Vgrid_writeUHBD: Problem opening virtual socket %s\n",
01278                 fname);
01279         return;
01280     }
01281     if (Vio_connect(sock, 0) < 0) {
01282         Vnm_print(2, "Vgrid_writeUHBD: Problem connecting virtual socket %s\n",
01283                 fname);
01284         return;
01285     }
01286
01287     /* Get the lower corner and number of grid points for the local
01288      * partition */
01289     hx = thee->hx;
01290     hy = thee->hy;
01291     hzed = thee->hzed;
01292     nx = thee->nx;
01293     ny = thee->ny;
01294     nz = thee->nz;
01295     xmin = thee->xmin;
01296     ymin = thee->ymin;
01297

```

```

01298     zmin = thee->zmin;
01299
01300     /* Let interested folks know that partition information is ignored */
01301     if (pvec != VNULL) {
01302         gotit = 0;
01303         for (i=0; i<(nx*ny*nz); i++) {
01304             if (pvec[i] == 0) {
01305                 gotit = 1;
01306                 break;
01307             }
01308         }
01309         if (gotit) {
01310             Vnm_print(2, "Vgrid_writeUHBD: IGNORING PARTITION INFORMATION!\n")
01311 ;
01312             Vnm_print(2, "Vgrid_writeUHBD: This means I/O from parallel runs \
01313 will have significant overlap.\n");
01314         }
01315     }
01316     /* Write out the header */
01317     Vio_printf(sock, "%72s\n", title);
01318     Vio_printf(sock, "%12.5e%12.5e%7d%7d%7d%7d%7d\n", 1.0, 0.0, -1, 0,
01319                 nz, 1, nz);
01320     Vio_printf(sock, "%7d%7d%7d%12.5e%12.5e%12.5e%12.5e\n", nx, ny, nz,
01321                 hx, (xmin-hx), (ymin-hx));
01322     Vio_printf(sock, "%12.5e%12.5e%12.5e%12.5e\n", 0.0, 0.0, 0.0, 0.0);
01323     Vio_printf(sock, "%12.5e%12.5e%7d%7d\n", 0.0, 0.0, 0, 0);
01324
01325     /* Write out the entries */
01326     icol = 0;
01327     for (k=0; k<nz; k++) {
01328         Vio_printf(sock, "\n%7d%7d%7d\n", k+1, thee->nx, thee->ny);
01329         icol = 0;
01330         for (j=0; j<ny; j++) {
01331             for (i=0; i<nx; i++) {
01332                 u = k* (nx)* (ny)+j* (nx)+i;
01333                 icol++;
01334                 Vio_printf(sock, " %12.5e", thee->data[u]);
01335                 if (icol == 6) {
01336                     icol = 0;
01337                     Vio_printf(sock, "\n");
01338                 }
01339             }
01340         }
01341     }
01342     if (icol != 0) Vio_printf(sock, "\n");
01343
01344     /* Close off the socket */
01345     Vio_connectFree(sock);
01346     Vio_dtor(&sock);
01347 }
01348
01349 VPUBLIC double Vgrid_integrate(Vgrid *thee) {
01350
01351     int i, j, k, nx, ny, nz;
01352     double sum, w;
01353
01354     if (thee == VNULL) {
01355         Vnm_print(2, "Vgrid_integrate: Got VNULL thee!\n");
01356         VASSERT(0);
01357     }
01358
01359     nx = thee->nx;
01360     ny = thee->ny;
01361     nz = thee->nz;
01362
01363     sum = 0.0;
01364
01365     for (k=0; k<nz; k++) {
01366         w = 1.0;
01367         if ((k==0) || (k==(nz-1))) w = w * 0.5;
01368         for (j=0; j<ny; j++) {
01369             w = 1.0;
01370             if ((j==0) || (j==(ny-1))) w = w * 0.5;
01371             for (i=0; i<nx; i++) {
01372                 w = 1.0;
01373                 if ((i==0) || (i==(nx-1))) w = w * 0.5;
01374                 sum = sum + w* (thee->data[IJK(i,j,k)]);
01375             }
01376         }
01377     }

```

```

01378     sum = sum*(thee->hx)*(thee->hy)*(thee->hzed);
01379
01380     return sum;
01381 }
01382
01383 }
01384
01385
01386 VPUBLIC double Vgrid_normL1(Vgrid *thee) {
01387
01388     int i, j, k, nx, ny, nz;
01389     double sum;
01390
01391     if (thee == VNULL) {
01392         Vnm_print(2, "Vgrid_normL1: Got VNULL thee!\n");
01393         VASSERT(0);
01394     }
01395
01396     nx = thee->nx;
01397     ny = thee->ny;
01398     nz = thee->nz;
01399
01400     sum = 0.0;
01401     for (k=0; k<nz; k++) {
01402         for (j=0; j<ny; j++) {
01403             for (i=0; i<nx; i++) {
01404                 sum = sum + VABS(thee->data[IJK(i,j,k)]);
01405             }
01406         }
01407     }
01408
01409     sum = sum*(thee->hx)*(thee->hy)*(thee->hzed);
01410
01411     return sum;
01412
01413 }
01414
01415 VPUBLIC double Vgrid_normL2(Vgrid *thee) {
01416
01417     int i, j, k, nx, ny, nz;
01418     double sum;
01419
01420     if (thee == VNULL) {
01421         Vnm_print(2, "Vgrid_normL2: Got VNULL thee!\n");
01422         VASSERT(0);
01423     }
01424
01425     nx = thee->nx;
01426     ny = thee->ny;
01427     nz = thee->nz;
01428
01429     sum = 0.0;
01430     for (k=0; k<nz; k++) {
01431         for (j=0; j<ny; j++) {
01432             for (i=0; i<nx; i++) {
01433                 sum = sum + VSQR(thee->data[IJK(i,j,k)]);
01434             }
01435         }
01436     }
01437
01438     sum = sum*(thee->hx)*(thee->hy)*(thee->hzed);
01439
01440     return VSQRT(sum);
01441
01442 }
01443
01444 VPUBLIC double Vgrid_seminormH1(Vgrid *thee) {
01445
01446     int i, j, k, d, nx, ny, nz;
01447     double pt[3], grad[3], sum, hx, hy, hzed, xmin, ymin, zmin
01448 ;
01449
01450     if (thee == VNULL) {
01451         Vnm_print(2, "Vgrid_seminormH1: Got VNULL thee!\n");
01452         VASSERT(0);
01453     }
01454
01455     nx = thee->nx;
01456     ny = thee->ny;
01457     nz = thee->nz;
01458     hx = thee->hx;

```

```

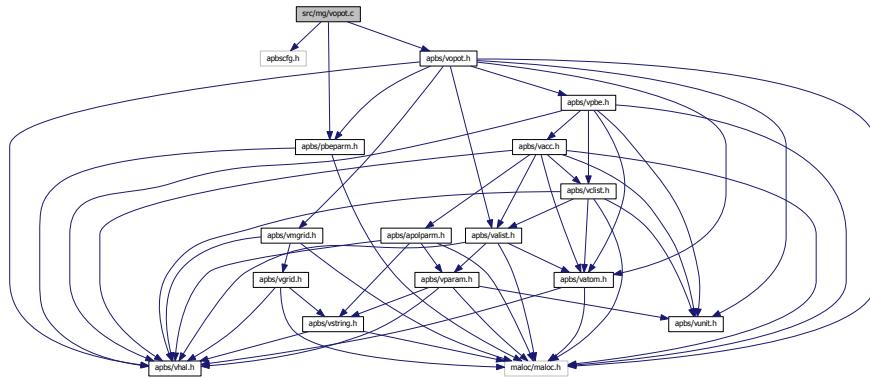
01458     hy = thee->hy;
01459     hzed = thee->hzed;
01460     xmin = thee->xmin;
01461     ymin = thee->ymin;
01462     zmin = thee->zmin;
01463
01464     sum = 0.0;
01465     for (k=0; k<nz; k++) {
01466         pt[2] = k*hzed + zmin;
01467         for (j=0; j<ny; j++) {
01468             pt[1] = j*hy + ymin;
01469             for (i=0; i<nx; i++) {
01470                 pt[0] = i*hx + xmin;
01471                 VASSERT(Vgrid_gradient(thee, pt, grad));
01472                 for (d=0; d<3; d++) sum = sum + VSQR(grad[d]);
01473             }
01474         }
01475     }
01476
01477     sum = sum*(hx)*(hy)*(hzed);
01478
01479     if (VABS(sum) < VSMALL) sum = 0.0;
01480     else sum = VSQRT(sum);
01481
01482     return sum;
01483
01484 }
01485
01486 VPUBLIC double Vgrid_normH1(Vgrid *thee) {
01487
01488     double sum = 0.0;
01489
01490     if (thee == VNULL) {
01491         Vnm_print(2, "Vgrid_normH1: Got VNULL thee!\n");
01492         VASSERT(0);
01493     }
01494
01495     sum = VSQR(Vgrid_seminormH1(thee)) + VSQR(Vgrid_normL2
01496     (thee));
01497
01498     return VSQRT(sum);
01499 }
01500
01501 VPUBLIC double Vgrid_normLinf(Vgrid *thee) {
01502
01503     int i, j, k, nx, ny, nz, gotval;
01504     double sum, val;
01505
01506     if (thee == VNULL) {
01507         Vnm_print(2, "Vgrid_normLinf: Got VNULL thee!\n");
01508         VASSERT(0);
01509     }
01510
01511     nx = thee->nx;
01512     ny = thee->ny;
01513     nz = thee->nz;
01514
01515     sum = 0.0;
01516     gotval = 0;
01517     for (k=0; k<nz; k++) {
01518         for (j=0; j<ny; j++) {
01519             for (i=0; i<nx; i++) {
01520                 val = VABS(thee->data[IJK(i,j,k)]);
01521                 if (!gotval) {
01522                     gotval = 1;
01523                     sum = val;
01524                 }
01525                 if (val > sum) sum = val;
01526             }
01527         }
01528     }
01529
01530     return sum;
01531
01532 }
01533

```

9.95 src/mg/vopot.c File Reference

Class Vopot methods.

```
#include "apbscfg.h"
#include "apbs/vopot.h"
#include "apbs/pbeparm.h"
Include dependency graph for vopot.c:
```



Macros

- `#define IJK(i, j, k) (((k)*(nx)*(ny)) + ((j)*(nx)) + (i))`

Functions

- VPUBLIC `Vopot * Vopot_ctor (Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl)`
Construct Vopot object with values obtained from Vpmg_readDX (for example)
- VPUBLIC int `Vopot_ctor2 (Vopot *thee, Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl)`
Initialize Vopot object with values obtained from Vpmg_readDX (for example)
- VPUBLIC void `Vopot_dtor (Vopot **thee)`
Object destructor.
- VPUBLIC void `Vopot_dtor2 (Vopot *thee)`
FORTRAN stub object destructor.
- VPUBLIC int `Vopot_pot (Vopot *thee, double pt[3], double *value)`
Get potential value (from mesh or approximation) at a point.
- VPUBLIC int `Vopot_curvature (Vopot *thee, double pt[3], int cflag, double *value)`
Get second derivative values at a point.
- VPUBLIC int `Vopot_gradient (Vopot *thee, double pt[3], double grad[3])`
Get first derivative values at a point.

9.95.1 Detailed Description

Class Vopot methods.

Author

Nathan Baker

Version**Id:**

[vopot.c](#) 1667 2011-12-02 23:22:02Z pcellis

Attention

```
*  
* APBS -- Adaptive Poisson-Boltzmann Solver  
*  
* Nathan A. Baker (nathan.baker@pnnl.gov)  
* Pacific Northwest National Laboratory  
*  
* Additional contributing authors listed in the code documentation.  
*  
* Copyright (c) 2010-2012 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](#).

9.96 vopot.c

```

00001
00057 #include "apbscfg.h"
00058 #include "apbs/vopot.h"
00059 #include "apbs/pbeparm.h"
00060
00061 VEMBED(rcsid="$Id: vopot.c 1667 2011-12-02 23:22:02Z pcellis $")
00062
00063 /* //////////////////////////////// */
00064 // Routine: Vopot_ctor
00065 // Author: Nathan Baker
00067 VPUBLIC Vopot* Vopot_ctor(Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl) {
00068
00069     Vopot *thee = VNULL;
00070
00071     thee = Vmem_malloc(VNULL, 1, sizeof(Vopot));
00072     VASSERT(thee != VNULL);
00073     VASSERT(Vopot_ctor2(thee, mgrid, pbe, bcfl));
00074
00075     return thee;
00076 }
00077
00078 /* //////////////////////////////// */
00079 // Routine: Vopot_ctor2
00080 // Author: Nathan Baker
00082 VPUBLIC int Vopot_ctor2(Vopot *thee, Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl) {
00083
00084     if (thee == VNULL) return 0;
00085     thee->bcfl = bcfl;
00086     thee->mgrid = mgrid;
00087     thee->pbe = pbe;
00088
00089     return 1;
00090 }
00091
00092 /* //////////////////////////////// */
00093 // Routine: Vopot_dtor
00094 // Author: Nathan Baker
00096 VPUBLIC void Vopot_dtor(Vopot **thee) {
00097
00098     if ((*thee) != VNULL) {
00099         Vopot_dtor2(*thee);
00100         Vmem_free(VNULL, 1, sizeof(Vopot), (void **)thee);
00101         (*thee) = VNULL;
00102     }
00103 }
00104
00105 /* //////////////////////////////// */
00106 // Routine: Vopot_dtor2
00107 // Author: Nathan Baker
00109 VPUBLIC void Vopot_dtor2(Vopot *thee) { return; }
00110
00111 /* //////////////////////////////// */
00112 // Routine: Vopot_pot
00113 // Author: Nathan Baker
00115 #define IJK(i,j,k) (((k)*(nx)*(ny))+((j)*(nx))+(i))
00116 VPUBLIC int Vopot_pot(Vopot *thee, double pt[3], double *value) {
00117
00118     Vatom *atom;
00119     int i, iatom;
00120     double u, T, charge, eps_w, xkappa, dist, size, val, *position;
00121     Valist *alist;
00122
00123     VASSERT(thee != VNULL);
00124
00125     eps_w = Vpbe_getSolventDiel(thee->pbe);
00126     xkappa = (1.0e10)*Vpbe_getXkappa(thee->pbe);
00127     T = Vpbe_getTemperature(thee->pbe);
00128     alist = Vpbe_getValist(thee->pbe);
00129
00130     u = 0;
00131
00132     /* See if we're on the mesh */
00133     if (Vmgrid_value(thee->mgrid, pt, &u)) {
00134
00135         *value = u;
00136
00137     } else {
00138

```

```

00139     switch (thee->bcfl) {
00140
00141     case BCFL_ZERO:
00142         u = 0;
00143         break;
00144
00145     case BCFL_SDH:
00146         size = (1.0e-10)*Vpbe_getSoluteRadius(thee
00147             ->pbe);
00148         position = Vpbe_getSoluteCenter(thee->pbe);
00149         charge = Vunit_ec*Vpbe_getSoluteCharge
00150             (thee->pbe);
00151         dist = 0;
00152         for (i=0; i<3; i++)
00153             dist += VSQR(position[i] - pt[i]);
00154         dist = (1.0e-10)*VSQRT(dist);
00155         val = (charge)/(4*VPI*Vunit_eps0*eps_w*dist);
00156         if (xkappa != 0.0)
00157             val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00158         val = val*Vunit_ec/(Vunit_kb*T);
00159         u = val;
00160         break;
00161
00162     case BCFL_MDH:
00163         u = 0;
00164         for (iatom=0; iatom<Valist_getNumberAtoms(
00165             alist); iatom++) {
00166             atom = Valist_getAtom(alist, iatom);
00167             position = Vatom_getPosition(atom);
00168             charge = Vunit_ec*Vatom_getCharge(
00169                 atom);
00170             size = (1e-10)*Vatom_getRadius(atom);
00171             dist = 0;
00172             for (i=0; i<3; i++)
00173                 dist += VSQR(position[i] - pt[i]);
00174             dist = (1.0e-10)*VSQRT(dist);
00175             val = (charge)/(4*VPI*Vunit_eps0*eps_w*dist);
00176             if (xkappa != 0.0)
00177                 val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00178             val = val*Vunit_ec/(Vunit_kb*T);
00179             u = u + val;
00180         }
00181         break;
00182
00183     case BCFL_UNUSED:
00184         Vnm_print(2, "Vopot_pot: Invalid bcfl flag (%d)!\n",
00185             thee->bcfl);
00186         return 0;
00187
00188     case BCFL_FOCUS:
00189         Vnm_print(2, "Vopot_pot: Invalid bcfl flag (%d)!",
00190             thee->bcfl);
00191         return 0;
00192
00193     default:
00194         Vnm_print(2, "Vopot_pot: Bogus thee->bcfl flag (%d)!\n",
00195             thee->bcfl);
00196         return 0;
00197         break;
00198     }
00199
00200     *value = u;
00201
00202 }
00203
00204 /* /////////////////////////////////
00205 // Routine: Vopot_curvature
00206 //
00207 // Notes: cflag=0 ==> Reduced Maximal Curvature
00208 //          cflag=1 ==> Mean Curvature (Laplace)
00209 //          cflag=2 ==> Gauss Curvature
00210 //          cflag=3 ==> True Maximal Curvature
00211 // If we are off the grid, we can still evaluate the Laplacian; assuming, we
00212 // are away from the molecular surface, it is simply equal to the DH factor.
00213 //
00214 // Authors: Nathan Baker
00215 VPUBLIC int Vopot_curvature(Vopot *thee, double pt[3], int cflag,

```

```

00217     double *value) {
00218
00219     Vatom *atom;
00220     int i, iatom;
00221     double u, T, charge, eps_w, xkappa, dist, size, val, *position, zkappa2;
00222     Valist *alist;
00223
00224     VASSERT(thee != VNULL);
00225
00226     eps_w = Vpbe_getSolventDiel(thee->pbe);
00227     xkappa = (1.0e10)*Vpbe_getXkappa(thee->pbe);
00228     zkappa2 = Vpbe_getZkappa2(thee->pbe);
00229     T = Vpbe_getTemperature(thee->pbe);
00230     alist = Vpbe_getValist(thee->pbe);
00231
00232     u = 0;
00233
00234     if (Vmgrid_curvature(thee->mgrid, pt, cflag, value)) return 1;
00235     else if (cflag != 1) {
00236         Vnm_print(2, "Vopot_curvature: Off mesh!\n");
00237         return 1;
00238     } else {
00239
00240         switch (thee->bcfl) {
00241
00242             case BCFL_ZERO:
00243                 u = 0;
00244                 break;
00245
00246             case BCFL_SDH:
00247                 size = (1.0e-10)*Vpbe_getSoluteRadius(thee->pbe);
00248                 position = Vpbe_getSoluteCenter(thee->pbe);
00249                 charge = Vunit_ec*Vpbe_getSoluteCharge(thee->pbe);
00250                 dist = 0;
00251                 for (i=0; i<3; i++)
00252                     dist += VSQR(position[i] - pt[i]);
00253                 dist = (1.0e-10)*VSQRT(dist);
00254                 if (xkappa != 0.0)
00255                     u = zkappa2*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00256                 break;
00257
00258             case BCFL_MDH:
00259                 u = 0;
00260                 for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
00261                     atom = Valist_getAtom(alist, iatom);
00262                     position = Vatom_getPosition(atom);
00263                     charge = Vunit_ec*Vatom_getCharge(atom);
00264                     size = (1e-10)*Vatom_getRadius(atom);
00265                     dist = 0;
00266                     for (i=0; i<3; i++)
00267                         dist += VSQR(position[i] - pt[i]);
00268                     dist = (1.0e-10)*VSQRT(dist);
00269                     if (xkappa != 0.0)
00270                         val = zkappa2*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00271                     u = u + val;
00272                 }
00273                 break;
00274
00275             case BCFL_UNUSED:
00276                 Vnm_print(2, "Vopot_pot: Invlid bcfl (%d)!\n", thee->bcfl);
00277                 return 0;
00278
00279             case BCFL_FOCUS:
00280                 Vnm_print(2, "Vopot_pot: Invlid bcfl (%d)!\n", thee->bcfl);
00281                 return 0;
00282
00283             default:
00284                 Vnm_print(2, "Vopot_pot: Bogus thee->bcfl flag (%d)!\n",
00285                           thee->bcfl);
00286                 return 0;
00287                 break;
00288             }
00289
00290             *value = u;
00291         }
00292
00293         return 1;
00294     }
00295 }
00296
00297 /* //////////////////////////////// */

```

```

00298 // Routine: Vopot_gradient
00299 //
00300 // Authors: Nathan Baker
00302 VPUBLIC int Vopot_gradient(Vopot *thee, double pt[3], double grad[3]) {
00303
00304     Vatom *atom;
00305     int iatom;
00306     double T, charge, eps_w, xkappa, size, val, *position;
00307     double dx, dy, dz, dist;
00308     Valist *alist;
00309
00310     VASSERT(thee != VNULL);
00311
00312     eps_w = Vpbe_getSolventDiel(thee->pbe);
00313     xkappa = (1.0e10)*Vpbe_getXkappa(thee->pbe);
00314     T = Vpbe_getTemperature(thee->pbe);
00315     alist = Vpbe_getValist(thee->pbe);
00316
00317
00318     if (!Vmgrid_gradient(thee->mgrid, pt, grad)) {
00319
00320         switch (thee->bcfl) {
00321
00322             case BCFL_ZERO:
00323                 grad[0] = 0.0;
00324                 grad[1] = 0.0;
00325                 grad[2] = 0.0;
00326                 break;
00327
00328             case BCFL_SDH:
00329                 grad[0] = 0.0;
00330                 grad[1] = 0.0;
00331                 grad[2] = 0.0;
00332                 size = (1.0e-10)*Vpbe_getSoluteRadius(thee->pbe);
00333                 position = Vpbe_getSoluteCenter(thee->pbe);
00334                 charge = Vunit_ec*Vpbe_getSoluteCharge(thee->pbe);
00335                 dx = position[0] - pt[0];
00336                 dy = position[1] - pt[1];
00337                 dz = position[2] - pt[2];
00338                 dist = VSQR(dx) + VSQR(dy) + VSQR(dz);
00339                 dist = (1.0e-10)*VSQRT(dist);
00340                 val = (charge)/(4*VPI*Vunit_eps0*eps_w);
00341                 if (xkappa != 0.0)
00342                     val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00343                 val = val*Vunit_ec/(Vunit_kb*T);
00344                 grad[0] = val*dx/dist*(-1.0/dist/dist + xkappa/dist);
00345                 grad[1] = val*dy/dist*(-1.0/dist/dist + xkappa/dist);
00346                 grad[2] = val*dz/dist*(-1.0/dist/dist + xkappa/dist);
00347                 break;
00348
00349             case BCFL_MDH:
00350                 grad[0] = 0.0;
00351                 grad[1] = 0.0;
00352                 grad[2] = 0.0;
00353                 for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
00354                     atom = Valist_getAtom(alist, iatom);
00355                     position = Vatom_getPosition(atom);
00356                     charge = Vunit_ec*Vatom_getCharge(atom);
00357                     size = (1e-10)*Vatom_getRadius(atom);
00358                     dx = position[0] - pt[0];
00359                     dy = position[1] - pt[1];
00360                     dz = position[2] - pt[2];
00361                     dist = VSQR(dx) + VSQR(dy) + VSQR(dz);
00362                     dist = (1.0e-10)*VSQRT(dist);
00363                     val = (charge)/(4*VPI*Vunit_eps0*eps_w);
00364                     if (xkappa != 0.0)
00365                         val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00366                     val = val*Vunit_ec/(Vunit_kb*T);
00367                     grad[0] += (val*dx/dist*(-1.0/dist/dist + xkappa/dist));
00368                     grad[1] += (val*dy/dist*(-1.0/dist/dist + xkappa/dist));
00369                     grad[2] += (val*dz/dist*(-1.0/dist/dist + xkappa/dist));
00370                 }
00371                 break;
00372
00373             case BCFL_UNUSED:
00374                 Vnm_print(2, "Vopot: Invalid bcfl (%d)!\n", thee->bcfl);
00375                 return 0;
00376
00377             case BCFL_FOCUS:
00378                 Vnm_print(2, "Vopot: Invalid bcfl (%d)!\n", thee->bcfl);
00379                 return 0;

```

```

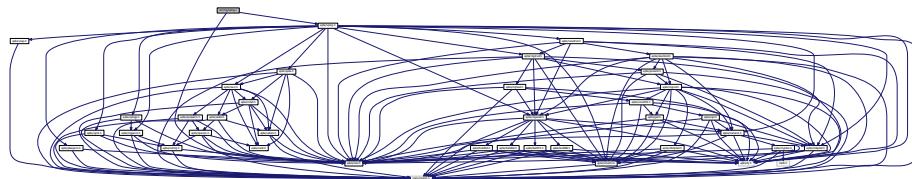
00380
00381     default:
00382         Vnm_print(2, "Vopot_pot: Bogus thee->bcfl flag (%d)!\n",
00383             thee->bcfl);
00384         return 0;
00385         break;
00386     }
00387
00388     return 1;
00389 }
00390
00391 return 1;
00392
00393 }
00394

```

9.97 src/mg/vpmg.c File Reference

Class Vpmg methods.

```
#include "apbs/vpmg.h"
#include "apbs/vhal.h"
Include dependency graph for vpmg.c:
```



Functions

- VPUBLIC unsigned long int [Vpmg_memChk](#) ([Vpmg](#) *thee)
Return the memory used by this structure (and its contents) in bytes.
- VPUBLIC void [Vpmg_printColComp](#) ([Vpmg](#) *thee, char path[72], char title[72], char mxtype[3], int flag)
Print out a column-compressed sparse matrix in Harwell-Boeing format.
- VPUBLIC [Vpmg](#) * [Vpmg_ctor](#) ([Vpmgp](#) *pmgp, [Vpbe](#) *pbe, int focusFlag, [Vpmg](#) *pmgOLD, [MGparm](#) *mgparm, [PBEparm_calcEnergy](#) energyFlag)
Constructor for the Vpmg class (allocates new memory)
- VPUBLIC int [Vpmg_ctor2](#) ([Vpmg](#) *thee, [Vpmgp](#) *pmgp, [Vpbe](#) *pbe, int focusFlag, [Vpmg](#) *pmgOLD, [MGparm](#) *mgparm, [PBEparm_calcEnergy](#) energyFlag)
FORTRAN stub constructor for the Vpmg class (uses previously-allocated memory)
- VPUBLIC int [Vpmg_solve](#) ([Vpmg](#) *thee)
Solve the PBE using PMG.
- VPUBLIC void [Vpmg_dtor](#) ([Vpmg](#) **thee)
Object destructor.
- VPUBLIC void [Vpmg_dtor2](#) ([Vpmg](#) *thee)
FORTRAN stub object destructor.
- VPUBLIC void [Vpmg_setPart](#) ([Vpmg](#) *thee, double lowerCorner[3], double upperCorner[3], int bflags[6])
Set partition information which restricts the calculation of observables to a (rectangular) subset of the problem domain.
- VPUBLIC void [Vpmg_unsetPart](#) ([Vpmg](#) *thee)
Remove partition restrictions.

- VPUBLIC int **Vpmg_fillArray** (**Vpmg** *thee, double *vec, **Vdata_Type** type, double parm, **Vhal_PBEType** pbetype, **PBEmprm** *pbeparm)

Fill the specified array with accessibility values.
- VPRIVATE double **Vpmg_polarizEnergy** (**Vpmg** *thee, int extFlag)

Determines energy from polarizable 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 srfm, **Vacc** *acc, double *gpos, double win, double inrad, **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-Hückel potential due to a single charge.*
- VPRIVATE void **bcCalcOrig** (**Vpmg** *thee)
- VPRIVATE int **gridPointisValid** (int i, int j, int k, int nx, int ny, int nz)
- VPRIVATE void **packAtoms** (double *ax, double *ay, double *az, double *charge, double *size, **Vpmg** *thee)
- VPRIVATE void **packUnpack** (int nx, int ny, int nz, int ngrid, double *gx, double *gy, double *gz, double *value, **Vpmg** *thee, int pack)
- VPRIVATE void **bcflnew** (**Vpmg** *thee)
- VPRIVATE void **multipolebc** (double r, double kappa, double eps_p, double eps_w, double rad, double tsr[3])

This routine serves bcfl2. It returns (in tsr) the contraction independent portion of the Debye-Hückel 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 **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 `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 srfm, Vchrg_Meth chgm`)

Calculate the total force on the specified atom in units of k_B T/AA.
- VPUBLIC int `Vpmg_ibForce` (`Vpmg *thee, double *force, int atomID, Vsurf_Meth srfm`)

Calculate the osmotic pressure on the specified atom in units of k_B T/AA.
- VPUBLIC int `Vpmg_dbForce` (`Vpmg *thee, double *dbForce, int atomID, Vsurf_Meth srfm`)

Calculate the dielectric boundary forces on the specified atom in units of k_B T/AA.
- VPUBLIC int `Vpmg_qfForce` (`Vpmg *thee, double *force, int atomID, Vchrg_Meth chgm`)

Calculate the "charge-field" force on the specified atom in units of k_B T/AA.
- VPRIVATE void `qfForceSpline1` (`Vpmg *thee, double *force, int atomID`)

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`)

Mark fraction of a volume element.
- 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 a sphere in a volume element.

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 [dbspline4](#) (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 [filcoCoefSpline3](#) ([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.

9.97.1 Detailed Description

Class [Vpmg](#) methods.

Author

Nathan Baker

Version

Id:

vpmg.c 1750 2012-07-18 18:34:27Z tuckerbeck

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](#).

9.97.2 Function Documentation

9.97.2.1 VPRIVATE void bcCalc (*Vpmg * thee*)

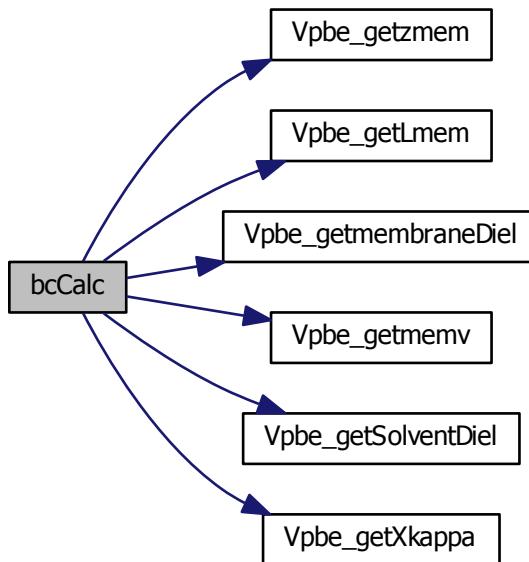
Fill boundary condition arrays.

Author

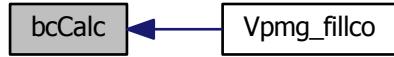
Nathan Baker

Definition at line 4370 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.2 VPRIVATE void bcf1 (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 $\text{pre1} * (\text{charge}/\text{d}) * (\exp(-\text{xkappa} * (\text{d}-\text{size})) / (1 + \text{xkappa} * \text{size}))$ to add the effect of the Debye-Hückel 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 z-direction |

Definition at line 2567 of file [vpmg.c](#).

9.97.2.3 VPRIVATE double bspline2 (double *x*)

Evaluate a cubic B-spline.

Author

Nathan Baker

Returns

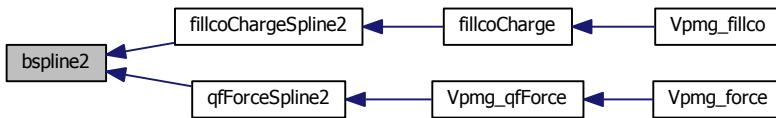
Cubic B-spline value

Parameters

| | |
|---|----------|
| x | Position |
|---|----------|

Definition at line 5484 of file [vpmg.c](#).

Here is the caller graph for this function:



9.97.2.4 VPRIVATE double bspline4 (double x)

Evaluate a 5th Order B-Spline (4th order polynomial)

Author

: Michael Schnieders

Returns

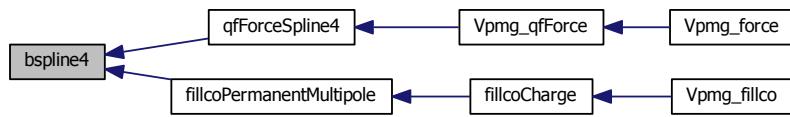
5th Order B-Spline

Parameters

| | |
|---|----------|
| x | Position |
|---|----------|

Definition at line 7124 of file [vpmg.c](#).

Here is the caller graph for this function:



9.97.2.5 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

| | |
|----------------|----------|
| <code>x</code> | Position |
|----------------|----------|

Definition at line [7190](#) of file [vpmg.c](#).

Here is the caller graph for this function:

**9.97.2.6 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

| | |
|----------------|----------|
| <code>x</code> | Position |
|----------------|----------|

Definition at line [7217](#) of file [vpmg.c](#).

9.97.2.7 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 5500 of file [vpmg.c](#).

Here is the caller graph for this function:

**9.97.2.8 VPUBLIC double dbspline4 (double x)**

Evaluate a 5th Order B-Spline derivative (4th order polynomial)

Author

: Michael Schnieders

Returns

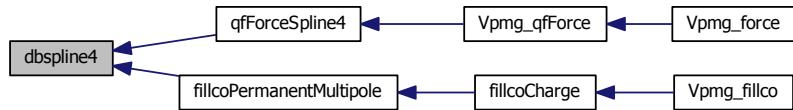
5th Order B-Spline derivative

Parameters

| | | |
|--|-----|----------|
| | x | Position |
|--|-----|----------|

Definition at line 7158 of file [vpmg.c](#).

Here is the caller graph for this function:



9.97.2.9 VPRIVATE Vrc.Codes fillcoCharge (Vpmg * thee)

Top-level driver to fill source term charge array.

Returns

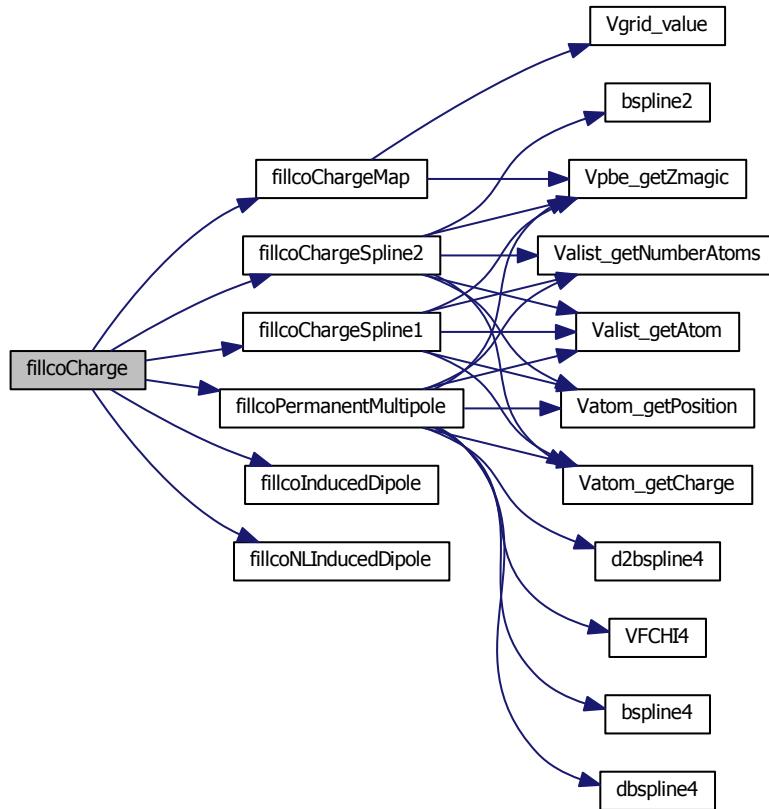
Success/failure status

Author

Nathan Baker

Definition at line 5275 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.10 VPRIVATE Vrc_Codes fillcoChargeMap (Vpmg * thee)

Fill source term charge array from a pre-calculated map.

Returns

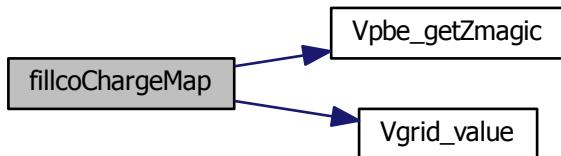
Success/failure status

Author

Nathan Baker

Definition at line 5331 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.11 VPRIVATE void fillcoChargeSpline1 (Vpmg * thee)

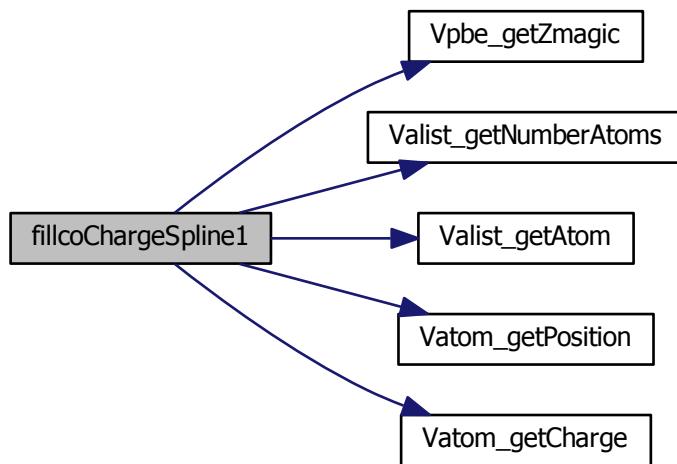
Fill source term charge array from linear interpolation.

Author

Nathan Baker

Definition at line 5379 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**9.97.2.12 VPRIVATE void fillcoChargeSpline2 (Vpmg * thee)**

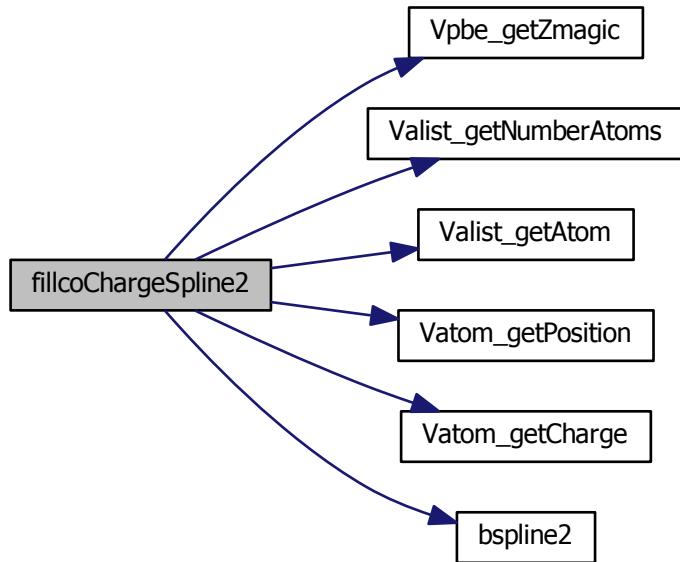
Fill source term charge array from cubic spline interpolation.

Author

Nathan Baker

Definition at line 5516 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**9.97.2.13 VPRIVATE void fillcoCoef (Vpmg * *thee*)**

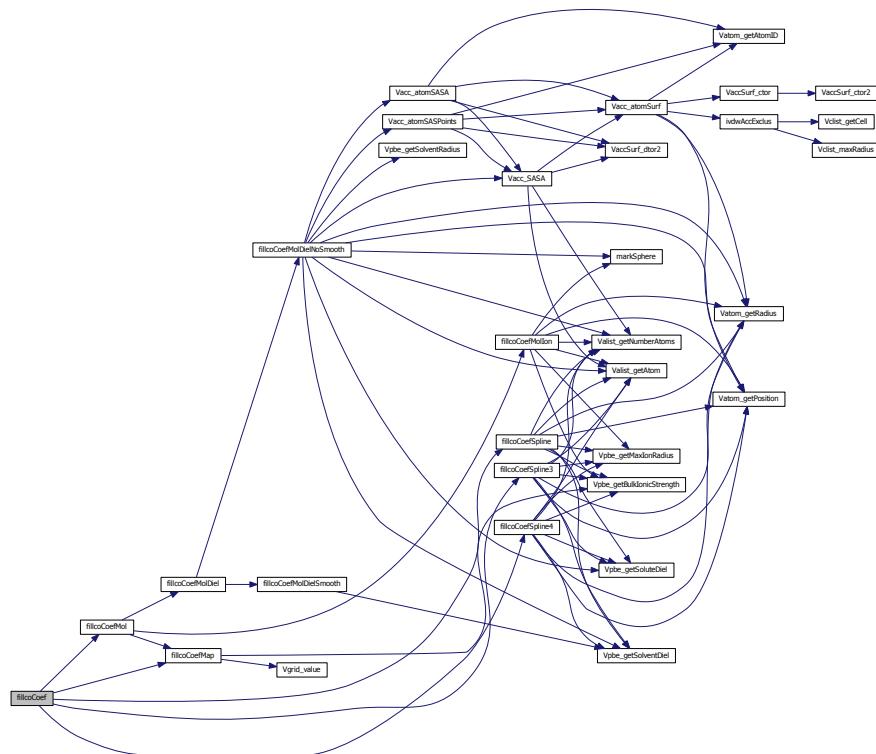
Top-level driver to fill all operator coefficient arrays.

Author

Nathan Baker

Definition at line 5235 of file vpmg.c.

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.14 VPRIVATE void fillCoefMap (Vpmg * *thee*)

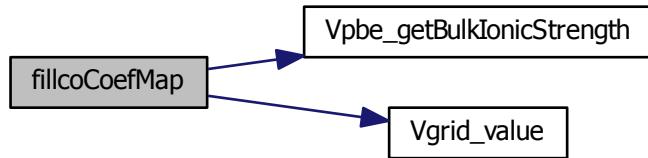
Fill operator coefficient arrays from pre-calculated maps.

Author

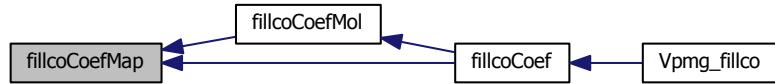
Nathan Baker

Definition at line [4477](#) of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.15 VPRIVATE void fillcoCoefMol (Vpmg * *thee*)

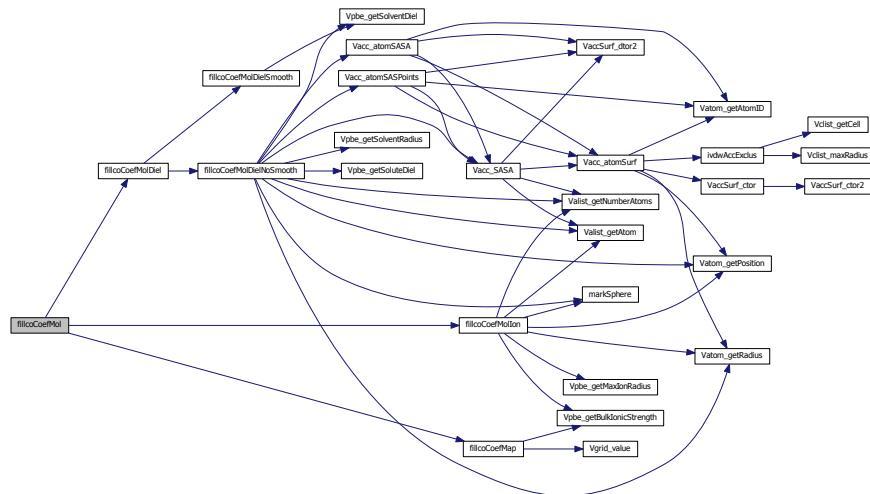
Fill operator coefficient arrays from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4600 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.16 VPRIVATE void fillcoCoefMolDiel (Vpmg * thee)

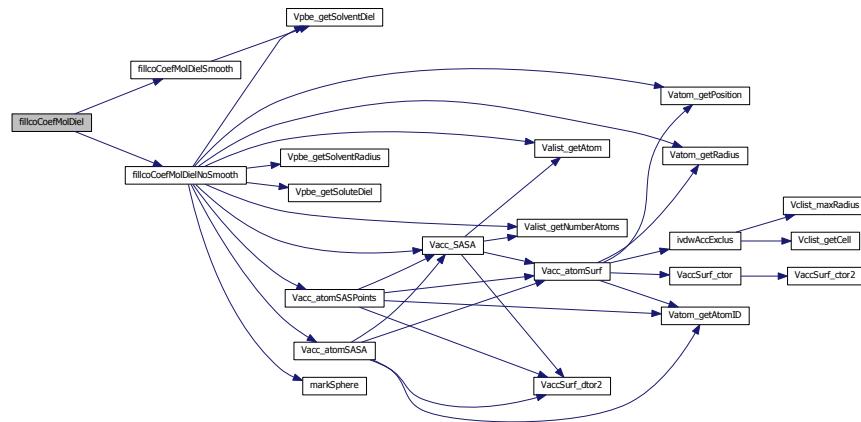
Fill differential operator coefficient arrays from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4714 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.17 VPRIVATE void fillcoCoefMolDielNoSmooth (*Vpmg* * *thee*)

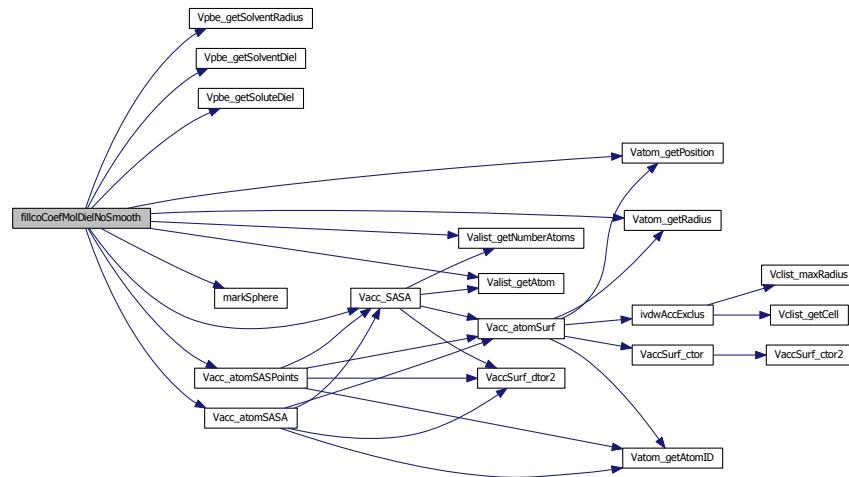
Fill differential operator coefficient arrays from a molecular surface calculation without smoothing.

Author

Nathan Baker

Definition at line 4725 of file vpmg.c.

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.18 VPRIVATE void fillcoCoefMoDielSmooth (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 [4879](#) of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**9.97.2.19 VPRIVATE void fillcoCoefMollon (Vpmg * *thee*)**

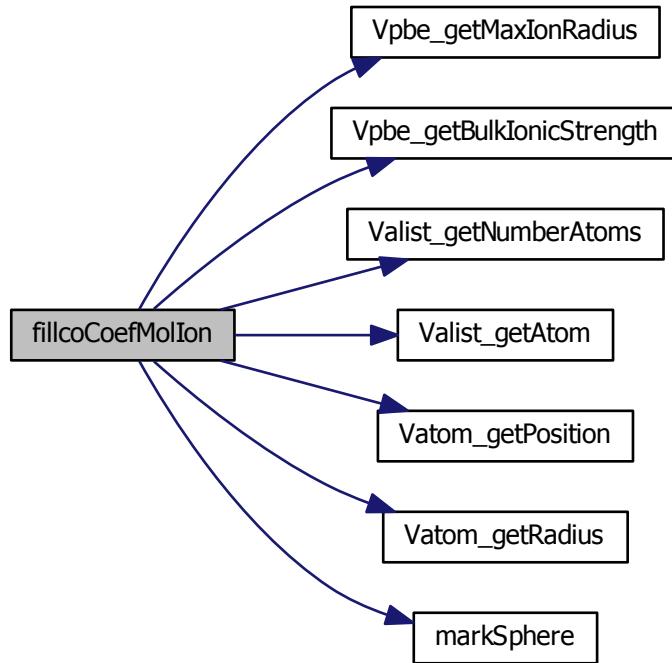
Fill ion (nonlinear) operator coefficient array from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4616 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.20 VPRIVATE void fillcoCoefSpline (Vpmg * *thee*)

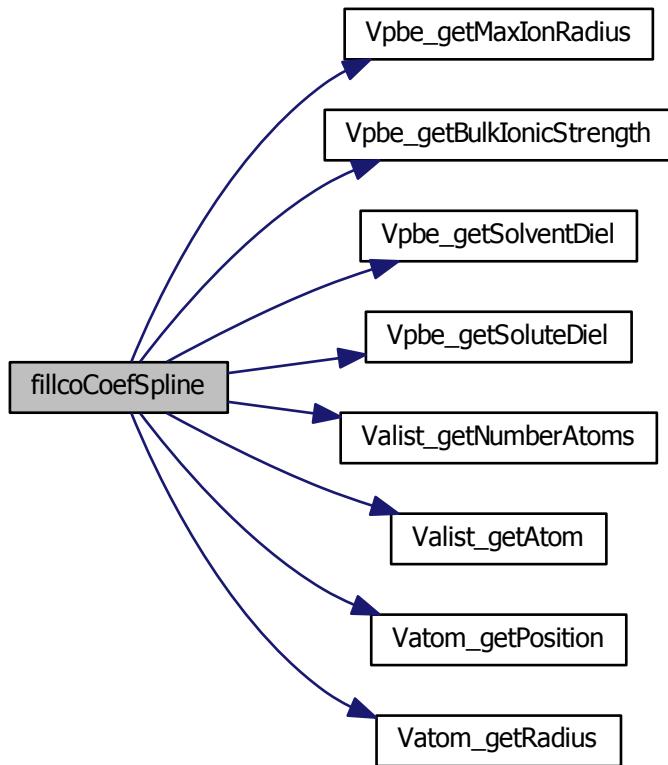
Fill operator coefficient arrays from a spline-based surface calculation.

Author

Nathan Baker

Definition at line 5010 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.21 VPRIVATE void fillcoCoefSpline3 (Vpmg * thee)

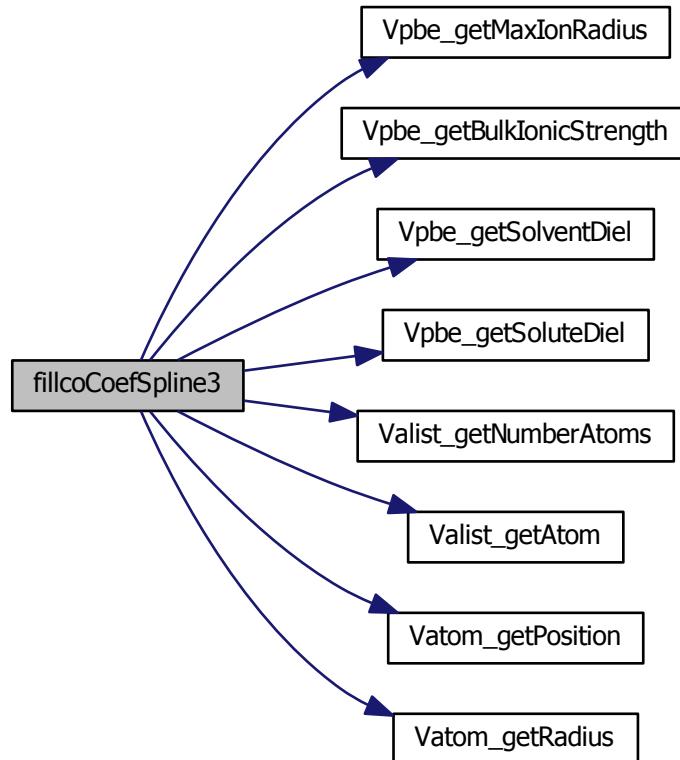
Fill operator coefficient arrays from a 5th order polynomial based surface calculation.

Author

Michael Schnieders

Definition at line 10418 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.22 VPRIVATE void fillcoCoefSpline4 (Vpmg * thee)

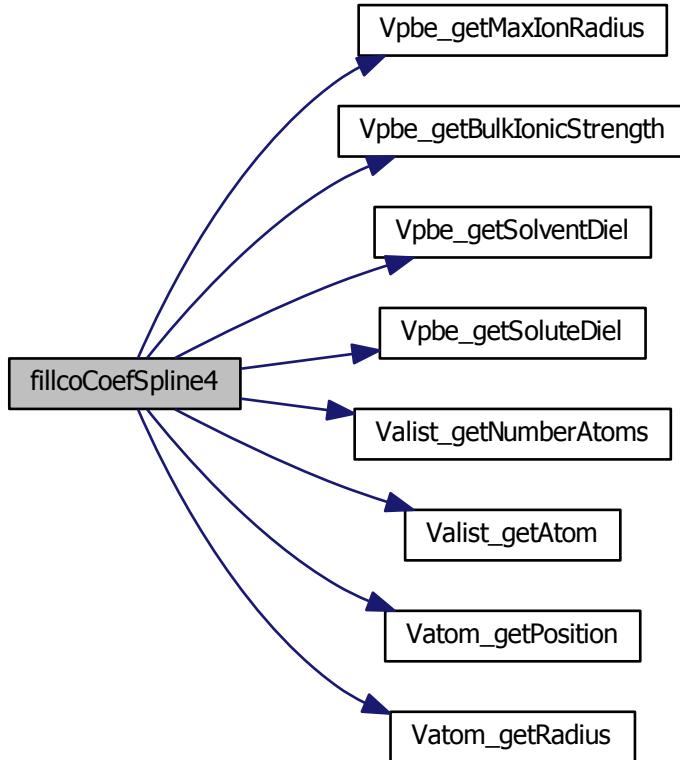
Fill operator coefficient arrays from a 7th order polynomial based surface calculation.

Author

Michael Schnieders

Definition at line 9927 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.23 VPUBLIC void fillcoPermanentMultipole (Vpmg * thee)

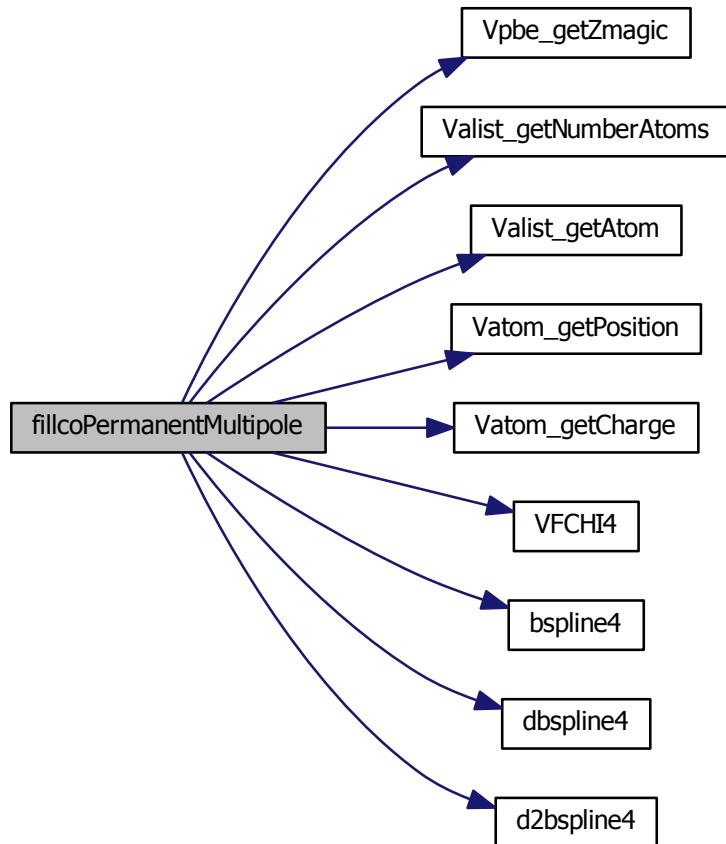
Fill source term charge array for the use of permanent multipoles.

Author

Michael Schnieders

Definition at line 7228 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.24 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.

Author

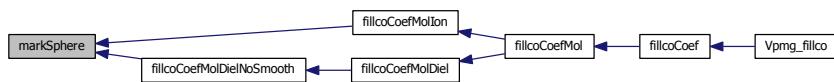
Nathan Baker

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 6837 of file [vpmg.c](#).

Here is the caller graph for this function:



9.97.2.25 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.

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 3480 of file [vpmg.c](#).

9.97.2.26 VPPRIVATE void qfForceSpline1 (Vpmg * *thee*, double * *force*, int *atomID*)

Charge-field force due to a linear spline charge function.

Author

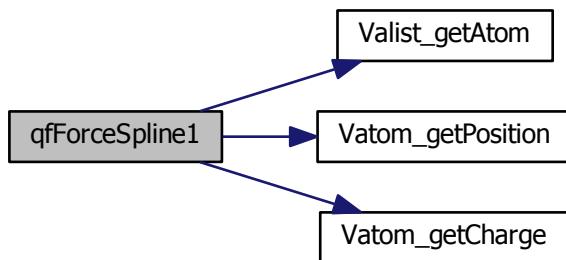
Nathan Baker

Parameters

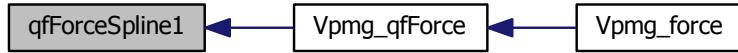
| | |
|---------------|-----------------------------|
| <i>atomID</i> | Set to force Valist atom ID |
|---------------|-----------------------------|

Definition at line 6299 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.27 VPRIVATE void qfForceSpline2 (Vpmg * *thee*, double * *force*, int *atomID*)

Charge-field force due to a cubic spline charge function.

Author

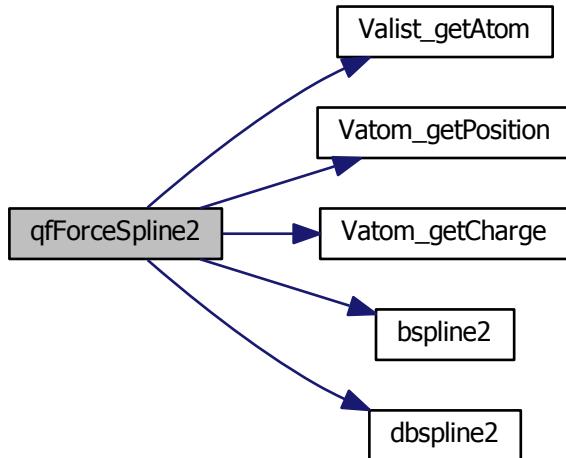
Nathan Baker

Parameters

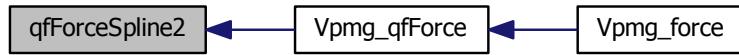
| | |
|---------------|-----------------------------|
| <i>atomID</i> | Set to force Valist atom ID |
|---------------|-----------------------------|

Definition at line 6436 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.28 VPRIVATE void qfForceSpline4 (Vpmg * *thee*, double * *force*, int *atomID*)

Charge-field force due to a quintic spline charge function.

Author

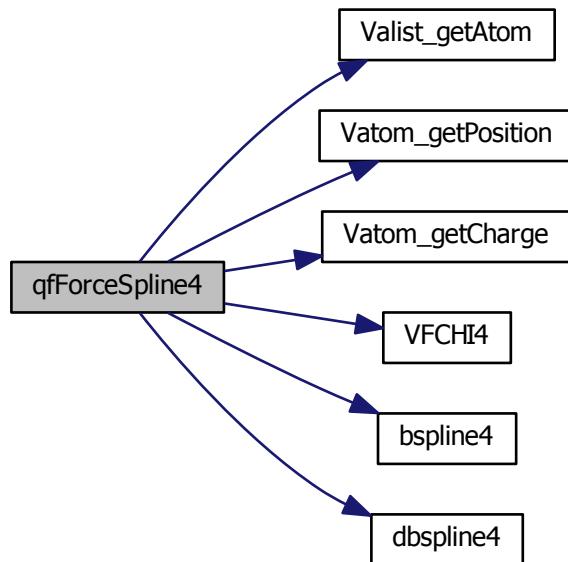
Michael Schnieders

Parameters

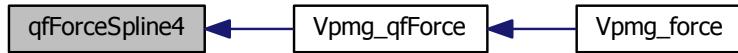
| | |
|---------------|-----------------------------|
| <i>atomID</i> | Set to force Valist atom ID |
|---------------|-----------------------------|

Definition at line 6549 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.29 VPRIVATE double VFCHI4 (int i, double f)

Return 2.5 plus difference of i - f.

Author

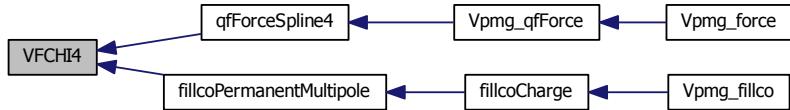
Michael Schnieders

Returns

$(2.5 + ((double)(i) - (f)))$

Definition at line 7120 of file [vpmg.c](#).

Here is the caller graph for this function:



9.97.2.30 VPRIVATE double Vpmg_polarizEnergy (Vpmg * thee, int extFlag)

Determines energy from polarizable charge and interaction with fixed charges according to Rocchia et al.

Author

Nathan Baker

Returns

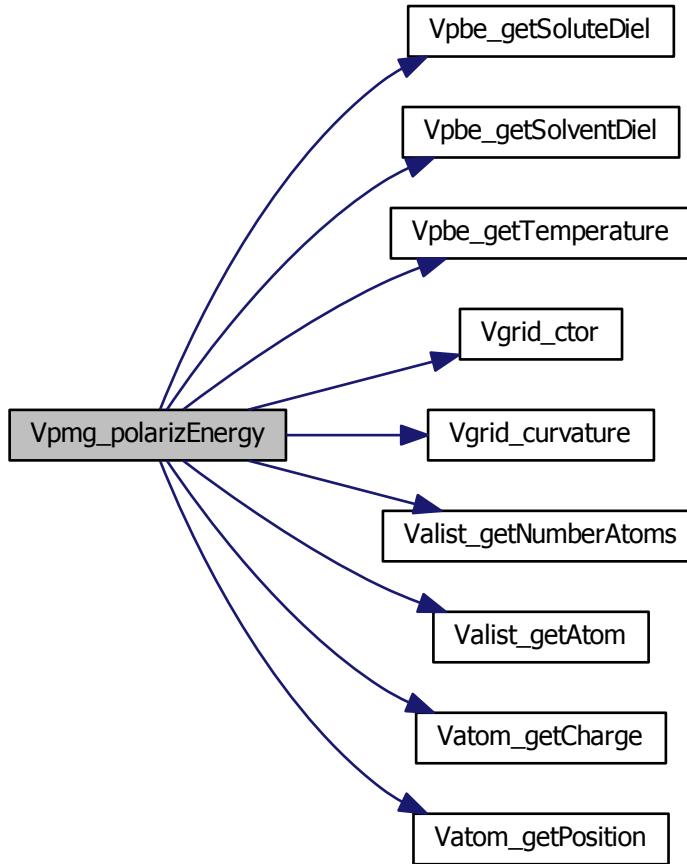
Energy in kT

Parameters

| | |
|----------------------|---|
| <code>extFlag</code> | If 1, add external energy contributions to result |
|----------------------|---|

Definition at line 1151 of file [vpmg.c](#).

Here is the call graph for this function:



9.97.2.31 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

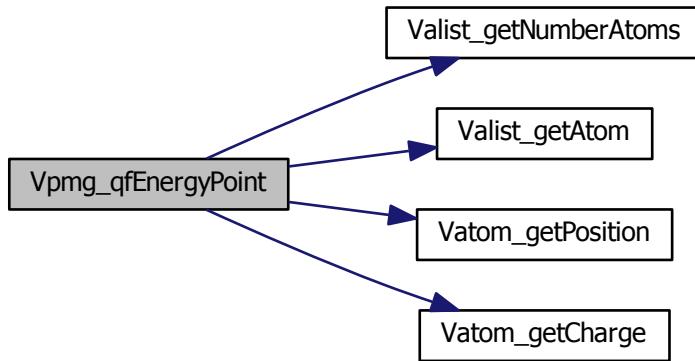
Energy in kT

Parameters

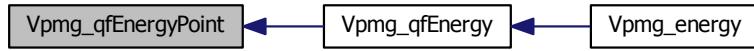
| | |
|----------------|---|
| <i>extFlag</i> | If 1, add external energy contributions to result |
|----------------|---|

Definition at line 1707 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.32 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

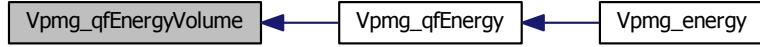
| | |
|----------------------|---|
| <code>extFlag</code> | If 1, add external energy contributions to result |
|----------------------|---|

Definition at line 1864 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.33 VPUBLIC double Vpmg_qmEnergySMPBE (`Vpmg *thee`, int `extFlag`)

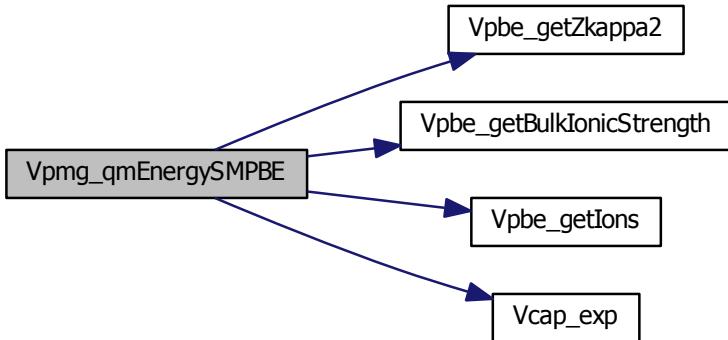
`Vpmg_qmEnergy` for SMPBE.

Author

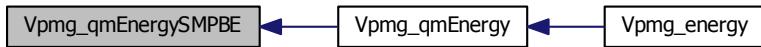
Vincent Chu

Definition at line 1493 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.34 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

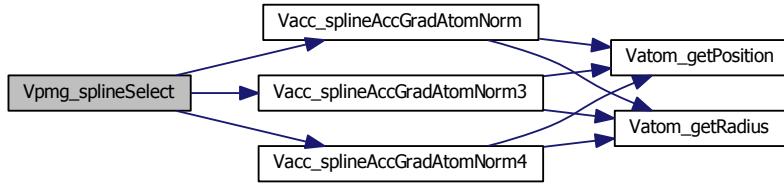
David Gohara

Parameters

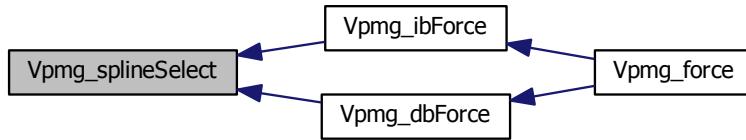
| | |
|---------------------|---|
| <code>acc</code> | Surface method, currently VSM_SPLINE, VSM_SPLINE5, or VSM_SPLINE7 |
| <code>gpos</code> | Accessibility object |
| <code>win</code> | Position array -> array[3] |
| <code>infrad</code> | Spline window |
| <code>atom</code> | Inflation radius |
| <code>force</code> | Atom object Force array -> array[3] |

Definition at line 1896 of file [vpmg.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.97.2.35 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 6886 of file [vpmg.c](#).

Here is the caller graph for this function:



9.98 vpmg.c

```

00001
00073 #include "apbs/vpmg.h"
00074 #include "apbs/vhal.h"
00075
00076 VEMBED(rcsid="$Id: vpmg.c 1750 2012-07-18 18:34:27Z tuckerbeck $")
00077
00078 #if !defined(VINLINE_VPMG)
00079
00080 VPUBLIC unsigned long int Vpmg_memChk(Vpmg *thee) {
00081     if (thee == VNULL) return 0;
00082     return Vmem_bytes(thee->vmem);
00083 }
00084
00085 #endif /* if !defined(VINLINE_VPMG) */
00086
00087
00088 VPUBLIC void Vpmg_printColComp(Vpmg *thee, char path[72],
00089     char title[72],
00090     char mxtype[3], int flag) {
00091     int nn, nxm2, nym2, nzm2, ncol, nrow, nonz;
00092     double *nzval;
00093     int *colptr, *rowind;
00094
00095     /* Calculate the total number of unknowns */
00096     nxm2 = thee->pmpg->nx - 2;
00097     nym2 = thee->pmpg->ny - 2;
00098     nzm2 = thee->pmpg->nz - 2;
00099     nn = nxm2*nym2*nzm2;
00100     ncol = nn;
00101     nrow = nn;
00102
00103     /* Calculate the number of non-zero matrix entries:
00104      * nn      nonzeros on diagonal
00105      * nn-1    nonzeros on first off-diagonal
00106      * nn-nx   nonzeros on second off-diagonal
00107      * nn-nx*ny nonzeros on third off-diagonal
00108      *
00109      * 7*nn-2*nx*ny-2*nx-2 TOTAL non-zeros
00110      */
00111     nonz = 7*nn - 2*nxm2*nym2 - 2*nxm2 - 2;
00112     nzval = (double*)Vmem_malloc(thee->vmem, nonz, sizeof(double));
00113     rowind = (int*)Vmem_malloc(thee->vmem, nonz, sizeof(int));
00114     colptr = (int*)Vmem_malloc(thee->vmem, (ncol+1), sizeof(int));
00115
00116 #ifndef VAPBSQUIET
00117     Vnm_print(1, "Vpmg_printColComp: Allocated space for %d nonzeros\n",
00118             nonz);
00119 #endif
00120
00121     bcolcomp(thee->iparm, thee->rparm, thee->iwork, thee
00122             ->rwork,
00123             nzval, rowind, colptr, &flag);
00124
00125 #if 0
00126     for (i=0; i<nn; i++) {

```

```

00127         Vnm_print(1, "nnz(%d) = %g\n", i, nzval[i]);
00128     }
00129 #endif
00130
00131     /* I do not understand why I need to pass nzval in this way, but it
00132      * works... */
00133     pcolcomp(&nrow, &ncol, &nonz, &(nzval[0]), rowind, colptr, path,
00134      title,
00135      mxtype);
00136     Vmem_free(thee->vmem, (ncol+1), sizeof(int), (void **) &colptr);
00137     Vmem_free(thee->vmem, nonz, sizeof(int), (void **) &rowind);
00138     Vmem_free(thee->vmem, nonz, sizeof(double), (void **) &nzval);
00139
00140 }
00141
00142 VPUBLIC Vpmg* Vpmg_ctor(Vpmgp *pmgp, Vpbe *pbe, int
00143   focusFlag,
00144   Vpmg *pmgOLD, MGparm *mgparm, PBEparm_calcEnergy
00145   energyFlag) {
00146
00147     Vpmg *thee = VNULL;
00148
00149     thee = (Vpmg*) Vmem_malloc(VNULL, 1, sizeof(Vpmg));
00150     VASSERT(thee != VNULL);
00151     VASSERT( Vpmg_ctor2(thee, pmgp, pbe, focusFlag, pmgOLD, mgparm,
00152       energyFlag) );
00153     return thee;
00154 }
00155
00156 VPUBLIC int Vpmg_ctor2(Vpmg *thee, Vpmgp *pmgp, Vpbe *
00157   pbe, int focusFlag,
00158   Vpmg *pmgOLD, MGparm *mgparm, PBEparm_calcEnergy
00159   energyFlag) {
00160
00161     int i, j, nion;
00162     double ionConc[MAXION], ionQ[MAXION], ionRadii[MAXION],
00163     zkappa2, zks2;
00164     double ionstr, partMin[3], partMax[3];
00165
00166     /* Get the parameters */
00167     VASSERT(pmgp != VNULL);
00168     VASSERT(pbe != VNULL);
00169     thee->pmgp = pmgp;
00170     thee->pbe = pbe;
00171
00172     /* Set up the memory */
00173     thee->vmem = Vmem_ctor("APBS:VPMG");
00174
00175     /* Initialize ion concentrations and valencies in PMG routines */
00176     zkappa2 = Vpbe_getZkappa2(thee->pbe);
00177     ionstr = Vpbe_getBulkIonicStrength(thee->pbe);
00178     if (ionstr > 0.0) zks2 = 0.5/ionstr;
00179     else zks2 = 0.0;
00180     Vpbe_getIons(thee->pbe, &nion, ionConc, ionRadii, ionQ);
00181
00182     /* TEMPORARY USEAQUA */
00183     /* Calculate storage requirements */
00184     if (mgparm->useAqua == 0){
00185       Vpmgp_size(thee->pmgp);
00186     } else{
00187       VABORT_MSG0("Aqua is currently disabled");
00188     }
00189
00190     /* We need some additional storage if: nonlinear & newton OR cgmgs */
00191     /* SMPBE Added - nonlin = 2 added since it mimics NPBE */
00192     if ( ((thee->pmgp->nonlin == NONLIN_NPBE) || (thee->pmgp->
00193       nonlin == NONLIN_SMPBE))
00194       && (thee->pmgp->meth == VSOL_Newton) ) || (thee->pmgp->meth
00195       == VSOL_CGMG)
00196     {
00197       thee->pmgp->nwk += (2*(thee->pmgp->nf));
00198     }
00199
00200     if (thee->pmgp->iinfo > 1) {
00201       Vnm_print(2, "Vpmg_ctor2: PMG chose nx = %d, ny = %d, nz = %d\n",
00202                 thee->pmgp->nx, thee->pmgp->ny, thee->
00203                 pmgp->nz);

```

```

00200      Vnm_print(2, "Vpmg_ctor2: PMG chose nlev = %d\n",
00201                  thee->pmgp->nlev);
00202      Vnm_print(2, "Vpmg_ctor2: PMG chose nxc = %d, nyc = %d, nzc = %d\n
00203      ",
00204                  thee->pmgp->nxc, thee->pmgp->nyc,
00205                  thee->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 );
00211      Vnm_print(2, "Vpmg_ctor2: PMG chose n_rpc = %d, n_iz = %d, n_ipc =
00212      %d\n",
00213                  thee->pmgp->n_rpc, thee->pmgp->n_iz
00214 , thee->pmgp->n_ipc);
00215      Vnm_print(2, "Vpmg_ctor2: PMG chose nrwk = %d, niwk = %d\n",
00216                  thee->pmgp->nrwk, thee->pmgp->niwk)
00217 ;
00218     }
00219
00220 /* Allocate boundary storage */
00221 thee->gxcf = (double *)Vmemp_malloc(thee->vmem,
00222                                     10*(thee->pmgp->ny)*(thee->pmgp->nz), sizeof(double));
00223 thee->gycf = (double *)Vmemp_malloc(thee->vmem,
00224                                     10*(thee->pmgp->nx)*(thee->pmgp->nz), sizeof(double));
00225 thee->gzcf = (double *)Vmemp_malloc(thee->vmem,
00226                                     10*(thee->pmgp->nx)*(thee->pmgp->ny), sizeof(double));
00227
00228 /* Warn users if they are using BCFL_MAP that
00229    we do not include external energies */
00230 if (thee->pmgp->bcfl == BCFL_MAP)
00231 Vnm_print(2, "Vpmg_ctor2: \nWarning: External energies are not used in
00232 BCFL_MAP calculations!\n");
00233
00234 if (focusFlag) {
00235 /* Overwrite any default or user-specified boundary condition
00236 * arguments; we are now committed to a calculation via focusing */
00237 if (thee->pmgp->bcfl != BCFL_FOCUS) {
00238     Vnm_print(2,
00239                 "Vpmg_ctor2: reset boundary condition flag to BCFL_FOCUS!\n");
00240     thee->pmgp->bcfl = BCFL_FOCUS;
00241 }
00242
00243 /* Fill boundaries */
00244 Vnm_print(0, "Vpmg_ctor2: Filling boundary with old solution!\n");
00245 focusFillBound(thee, pmgOLD);
00246
00247 /* Calculate energetic contributions from region outside focusing
00248 * domain */
00249 if (energyFlag != PCE_NO) {
00250
00251     if (mgparm->type == MCT_PARALLEL) {
00252
00253         for (j=0; j<3; j++) {
00254             partMin[j] = mgparm->partDisjCenter[j]
00255             - 0.5*mgparm->partDisjLength[j];
00256             partMax[j] = mgparm->partDisjCenter[j]
00257             + 0.5*mgparm->partDisjLength[j];
00258         }
00259
00260         extEnergy(thee, pmgOLD, energyFlag, partMin, partMax,
00261                   mgparm->partDisjOwnSide);
00262     }
00263 } else {
00264
00265 /* Ignore external energy contributions */
00266 thee->extQmEnergy = 0;
00267 thee->extDiEnergy = 0;
00268 thee->extQfEnergy = 0;
00269 }
00270
00271 /* Allocate partition vector storage */
00272 thee->pvec = (double *)Vmemp_malloc(thee->vmem,
00273                                     (thee->pmgp->nx)*(thee->pmgp->ny)*(thee->pmgp->nz

```

```

), sizeof(double));
00274 /* Allocate remaining storage */
00275 thee->iparm = (int *)Vmem_malloc(thee->vmem, 100, sizeof(int));
00276 thee->rparm = (double *)Vmem_malloc(thee->vmem, 100, sizeof(double));
00277 thee->iwork = (int *)Vmem_malloc(thee->vmem, thee->pmgp->nwk
00278
00279     sizeof(int));
00280 thee->rwork = (double *)Vmem_malloc(thee->vmem, thee->pmgp->nrwk
00281
00282     sizeof(double));
00283 thee->charge = (double *)Vmem_malloc(thee->vmem, thee->pmgp->
narr,
00284     sizeof(double));
00285 thee->kappa = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr
00286
00287     sizeof(double));
00288 thee->pot = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00289     sizeof(double));
00290 thee->epsx = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr
00291
00292     sizeof(double));
00293 thee->epsy = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr
00294
00295     sizeof(double));
00296 thee->epsz = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr
00297
00298     sizeof(double));
00299 thee->a1cf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr
00300
00301     sizeof(double));
00302 thee->f1cf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00303     sizeof(double));
00304 thee->t1cf = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00305     sizeof(double));
00306 thee->u = (double *)Vmem_malloc(thee->vmem, thee->pmgp->narr,
00307     sizeof(double));
00308 thee->xf = (double *)Vmem_malloc(thee->vmem, 5*(thee->pmgp->nx),
00309     sizeof(double));
00310 thee->yf = (double *)Vmem_malloc(thee->vmem, 5*(thee->pmgp->ny),
00311     sizeof(double));
00312 thee->zf = (double *)Vmem_malloc(thee->vmem, 5*(thee->pmgp->nz),
00313     sizeof(double));
00314
00315
00316
00317 /* Packs parameters into the iparm and rparm arrays */
00318 Vpackmg(thee->iparm, thee->rparm, &(thee->pmgp->nrwk
00319 ), &(thee->pmgp->nwk),
00320     &(thee->pmgp->nx), &(thee->pmgp->ny), &(thee->pmgp->nz)
00321
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->mgsmo),
00325     &(thee->pmgp->mgprol), &(thee->pmgp->mgcoar), &(thee
->pmgp->mgolv),
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 /* Currently for SMPBE type calculations we do not want to apply a scale
00331 factor to the ionConc */
00332 switch(pmgp->ipkey){
00333
00334 case IPKEY_SMPBE:
00341

```

```

00342             Vmypdefinitmpbe(&nion, ionQ, ionConc,
00343             &pbe->smvolume, &pbe->smsize);
00344             break;
00345
00346
00347     case IPKEY_NPBE:
00348
00349         /* Else adjust the ionConc by scaling factor zks2 */
00350         for (i=0; i<nion; i++)
00351             ionConc[i] = zks2 * ionConc[i];
00352
00353             Vmypdefinitnpbe(&nion, ionQ, ionConc);
00354             break;
00355
00356
00357
00358     case IPKEY_LPBE:
00359
00360         /* Else adjust the ionConc by scaling factor zks2 */
00361         for (i=0; i<nion; i++)
00362             ionConc[i] = zks2 * ionConc[i];
00363
00364     Vmypdefinitlpbe(&nion, ionQ, ionConc);
00365     break;
00366
00367
00368
00369     default:
00370         Vnm_print(2, "PMG: Warning: PBE structure not initialized!\n");
00371         /* Else adjust the ionConc by scaling factor zks2 */
00372         for (i=0; i<nion; i++)
00373             ionConc[i] = zks2 * ionConc[i];
00374         break;
00375     }
00376
00377     /* Set the default chargeSrc for 5th order splines */
00378     thee->chargeSrc = mgparm->chgs;
00379
00380     /* Turn off restriction of observable calculations to a specific
00381     * partition */
00382     Vpmg_unsetPart(thee);
00383
00384     /* The coefficient arrays have not been filled */
00385     thee->filled = 0;
00386
00387
00388     /*
00389     * TODO: Move the dtor out of here. The current ctor is done in routines.c,
00390     * This was originally moved out to kill a memory leak. The dtor has
00391     * has been removed from initMG and placed back here to keep memory
00392     * usage low. killMG has been modified accordingly.
00393     */
00394     Vpmg_dtor(&pmgOLD);
00395
00396     return 1;
00397 }
00398
00399 VPUBLIC int Vpmg_solve(Vpmg *thee) {
00400
00401     int i,
00402         nx,
00403         ny,
00404         nz,
00405         n;
00406     double zkappa2;
00407
00408     nx = thee->pmgp->nx;
00409     ny = thee->pmgp->ny;
00410     nz = thee->pmgp->nz;
00411     n = nx*ny*nz;
00412
00413     if (! (thee->filled)) {
00414         Vnm_print(2, "Vpmg_solve: Need to call Vpmg_fillco() !\n");
00415         return 0;
00416     }
00417
00418     /* Fill the "true solution" array */
00419     for (i=0; i<n; i++) {
00420         thee->tcf[i] = 0.0;
00421     }

```

```

00422
00423     /* Fill the RHS array */
00424     for (i=0; i<n; i++) {
00425         thee->fcf[i] = thee->charge[i];
00426     }
00427
00428     /* Fill the operator coefficient array. */
00429     for (i=0; i<n; i++) {
00430         thee->a1cf[i] = thee->epsx[i];
00431         thee->a2cf[i] = thee->epsy[i];
00432         thee->a3cf[i] = thee->epsz[i];
00433     }
00434
00435     /* Fill the nonlinear coefficient array by multiplying the kappa
00436      * accessibility array (containing values between 0 and 1) by zkappa2. */
00437     zkappa2 = Vpbe_getZkappa2(thee->pbe);
00438     if (zkappa2 > VPMGSMALL) {
00439         for (i=0; i<n; i++) {
00440             thee->ccf[i] = zkappa2*thee->kappa[i];
00441         }
00442     } else {
00443         for (i=0; i<n; i++) {
00444             thee->ccf[i] = 0.0;
00445         }
00446     }
00447
00448     switch(thee->pmgp->meth) {
00449         /* CGMG (linear) */
00450         case VSOL_CGMG:
00451
00452             if (thee->pmgp->iinfo > 1)
00453                 Vnm_print(2, "Driving with CGMGDRIV\n");
00454
00455             VABORT_MSG0("CGMGDRIV is not currently supported");
00456             break;
00457
00458         /* Newton (nonlinear) */
00459         case VSOL_Newton:
00460
00461             if (thee->pmgp->iinfo > 1)
00462                 Vnm_print(2, "Driving with NEWDRIV\n");
00463
00464             Vnewdriv
00465             (thee->iparm, thee->rparm, thee->iwork,
00466             thee->rwork,
00467             thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf
00468             , thee->gycf,
00469             thee->gzcf, thee->a1cf, thee->a2cf, thee->
00470             a3cf,
00471             thee->ccf, thee->tcf);
00472             break;
00473
00474         /* MG (linear/nonlinear) */
00475         case VSOL_MG:
00476
00477             if (thee->pmgp->iinfo > 1)
00478                 Vnm_print(2, "Driving with MGDRIV\n");
00479
00480             Vmgdriv(thee->iparm, thee->rparm, thee->iwork
00481             , thee->rwork,
00482             thee->zf, thee->gxcf, thee->gycf,
00483             thee->gzcf, thee->a1cf, thee->
00484             a2cf, thee->a3cf, thee->ccf,
00485             thee->tcf);
00486             break;
00487
00488         /* CGHS (linear/nonlinear) */
00489         case VSOL(CG:
00490
00491             if (thee->pmgp->iinfo > 1)
00492                 Vnm_print(2, "Driving with NCGHSDRIV\n");
00493
00494             VABORT_MSG0("NCGHSDRIV is not currently supported");
00495             break;
00496
00497         /* SOR (linear/nonlinear) */
00498         case VSOL_SOR:
00499
00500             if (thee->pmgp->iinfo > 1)
00501                 Vnm_print(2, "Driving with NSORDRIV\n");

```

```

00497     VABORT_MSG0("NSORDRIV is not currently supported");
00498     break;
00499
00500     /* GSRB (linear/nonlinear) */
00501     case VSOL_RBGS:
00502
00503         if (thee->pmgp->iinfo > 1)
00504             Vnm_print(2, "Driving with NGSRBDRIV\n");
00505
00506         VABORT_MSG0("NGSRBDRIV is not currently supported");
00507         break;
00508
00509     /* WJAC (linear/nonlinear) */
00510     case VSOL_WJ:
00511
00512         if (thee->pmgp->iinfo > 1)
00513             Vnm_print(2, "Driving with NWJACDRIV\n");
00514
00515         VABORT_MSG0("NWJACDRIV is not currently supported");
00516         break;
00517
00518     /* RICH (linear/nonlinear) */
00519     case VSOL_Richardson:
00520
00521         if (thee->pmgp->iinfo > 1)
00522             Vnm_print(2, "Driving with NRICHDRIV\n");
00523
00524         VABORT_MSG0("NRICHDRIV is not currently supported");
00525         break;
00526
00527
00528     /* CGMG (linear) TEMPORARY USEAQUA */
00529     case VSOL_CGMGAqua:
00530
00531         if (thee->pmgp->iinfo > 1)
00532             Vnm_print(2, "Driving with CGMGDRIVAQUA\n");
00533
00534         VABORT_MSG0("CGMGDRIVAQUA is not currently supported");
00535         break;
00536
00537     /* Newton (nonlinear) TEMPORARY USEAQUA */
00538     case VSOL_NewtonAqua:
00539
00540         if (thee->pmgp->iinfo > 1)
00541             Vnm_print(2, "Driving with NEWDRIVAQUA\n");
00542
00543         VABORT_MSG0("NEWDRIVAQUA is not currently supported");
00544         break;
00545
00546     /* Error handling */
00547     default:
00548         Vnm_print(2, "Vpmg_solve: invalid solver method key (%d)\n",
00549                   thee->pmgp->key);
00550         return 0;
00551         break;
00552     }
00553
00554     return 1;
00555
00556 }
00557
00558
00559 VPUBLIC void Vpmg_dtor(Vpmg **thee) {
00560
00561     if ((*thee) != VNULL) {
00562         Vpmg_dtor2(*thee);
00563         Vmem_free(VNULL, 1, sizeof(Vpmg), (void **)thee);
00564         (*thee) = VNULL;
00565     }
00566
00567 }
00568
00569 VPUBLIC void Vpmg_dtor2(Vpmg *thee) {
00570
00571     /* Clean up the storage */
00572
00573     Vmem_free(thee->vmem, 100, sizeof(int), (void **)&(thee->iparm));
00574     Vmem_free(thee->vmem, 100, sizeof(double), (void **)&(thee->rparm)
00575 );
00576     Vmem_free(thee->vmem, thee->pmgp->nwk, sizeof(int),
00577               (void **)&(thee->iwork));

```

```

00577     Vmem_free(thee->vmem, thee->pmgp->nwk, sizeof(double),
00578             (void **) &(thee->rwork));
00579     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00580             (void **) &(thee->charge));
00581     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00582             (void **) &(thee->kappa));
00583     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00584             (void **) &(thee->pot));
00585     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00586             (void **) &(thee->epsx));
00587     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00588             (void **) &(thee->epsy));
00589     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00590             (void **) &(thee->epsz));
00591     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00592             (void **) &(thee->alcf));
00593     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00594             (void **) &(thee->a2cf));
00595     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00596             (void **) &(thee->a3cf));
00597     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00598             (void **) &(thee->ccf));
00599     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00600             (void **) &(thee->fcf));
00601     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00602             (void **) &(thee->tcf));
00603     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00604             (void **) &(thee->u));
00605     Vmem_free(thee->vmem, 5*(thee->pmgp->nx), sizeof(double),
00606             (void **) &(thee->xf));
00607     Vmem_free(thee->vmem, 5*(thee->pmgp->ny), sizeof(double),
00608             (void **) &(thee->yf));
00609     Vmem_free(thee->vmem, 5*(thee->pmgp->nz), sizeof(double),
00610             (void **) &(thee->zf));
00611     Vmem_free(thee->vmem, 10*(thee->pmgp->ny)*(thee->pmgp->nz),
00612             sizeof(double),
00613             (void **) &(thee->gxcf));
00614     Vmem_free(thee->vmem, 10*(thee->pmgp->nx)*(thee->pmgp->nz),
00615             sizeof(double),
00616             (void **) &(thee->gycf));
00617     Vmem_free(thee->vmem, 10*(thee->pmgp->nx)*(thee->pmgp->ny),
00618             sizeof(double),
00619             (void **) &(thee->gzcf));
00620     Vmem_dtor(&(thee->vmem));
00621 }
00622
00623 VPUBLIC void Vpmg_setPart(Vpmg *thee, double lowerCorner[3],
00624     double upperCorner[3], int bflags[6]) {
00625
00626     Valist *alist;
00627     Vatom *atom;
00628     int i, j, k, nx, ny, nz;
00629     double xmin, ymin, zmin, x, y, z, hx, hy, hzed, yok,
00630     yok, zok;
00631     double x0, x1, y0, y1, z0, z1;
00632     nx = thee->pmgp->nx;
00633     ny = thee->pmgp->ny;
00634     nz = thee->pmgp->nz;
00635     hx = thee->pmgp->hx;
00636     hy = thee->pmgp->hy;
00637     hzed = thee->pmgp->hzed;
00638     xmin = thee->pmgp->xcent - 0.5*hx*(nx-1);
00639     ymin = thee->pmgp->ycent - 0.5*hy*(ny-1);
00640     zmin = thee->pmgp->zcent - 0.5*hzed*(nz-1);
00641
00642     yok = 0;
00643     yok = 0;
00644     zok = 0;
00645
00646     /* We need have called Vpmg_fillco first */
00647
00648     alist = thee->pbe->alist;
00649
00650     Vnm_print(0, "Vpmg_setPart: lower corner = (%g, %g, %g)\n",
00651             lowerCorner[0], lowerCorner[1], lowerCorner[2]);
00652     Vnm_print(0, "Vpmg_setPart: upper corner = (%g, %g, %g)\n",

```

```

00653     upperCorner[0], upperCorner[1], upperCorner[2]);
00654 Vnm_print(0, "Vpmg_setPart: actual minima = (%g, %g, %g)\n",
00655     xmin, ymin, zmin);
00656 Vnm_print(0, "Vpmg_setPart: actual maxima = (%g, %g, %g)\n",
00657     xmin+hx*(nx-1), ymin+hy*(ny-1), zmin+hzed*(nz-1));
00658 Vnm_print(0, "Vpmg_setPart: bflag[FRONT] = %d\n",
00659     bflags[VAPBS_FRONT]);
00660 Vnm_print(0, "Vpmg_setPart: bflag[BACK] = %d\n",
00661     bflags[VAPBS_BACK]);
00662 Vnm_print(0, "Vpmg_setPart: bflag[LEFT] = %d\n",
00663     bflags[VAPBS_LEFT]);
00664 Vnm_print(0, "Vpmg_setPart: bflag[RIGHT] = %d\n",
00665     bflags[VAPBS_RIGHT]);
00666 Vnm_print(0, "Vpmg_setPart: bflag[UP] = %d\n",
00667     bflags[VAPBS_UP]);
00668 Vnm_print(0, "Vpmg_setPart: bflag[DOWN] = %d\n",
00669     bflags[VAPBS_DOWN]);
00670
00671 /* Identify atoms as inside, outside, or on the border
00672   If on the border, use the bflags to determine if there
00673   is an adjacent processor - if so, this atom should be equally
00674   shared. */
00675
00676 for (i=0; i<Valist_getNumberAtoms(alist); i++) {
00677     atom = Valist_getAtom(alist, i);
00678
00679     if ((atom->position[0] < upperCorner[0]) &&
00680         (atom->position[0] > lowerCorner[0])) xok = 1;
00681     else {
00682         if ((VABS(atom->position[0] - lowerCorner[0]) < VPMGSMALL
00683             ) &&
00684             (bflags[VAPBS_LEFT] == 0)) xok = 1;
00685         else if ((VABS(atom->position[0] - lowerCorner[0]) <
00686                   VPMGSMALL) &&
00687             (bflags[VAPBS_LEFT] == 1)) xok = 0.5;
00688         else if ((VABS(atom->position[0] - upperCorner[0]) <
00689                   VPMGSMALL) &&
00690             (bflags[VAPBS_RIGHT] == 0)) xok = 1;
00691         else if ((VABS(atom->position[0] - upperCorner[0]) <
00692                   VPMGSMALL) &&
00693             (bflags[VAPBS_RIGHT] == 1)) xok = 0.5;
00694         else xok = 0;
00695     }
00696     if ((atom->position[1] < upperCorner[1]) &&
00697         (atom->position[1] > lowerCorner[1])) yok = 1;
00698     else {
00699         if ((VABS(atom->position[1] - lowerCorner[1]) < VPMGSMALL
00700             ) &&
00701             (bflags[VAPBS_BACK] == 0)) yok = 1;
00702         else if ((VABS(atom->position[1] - lowerCorner[1]) <
00703                   VPMGSMALL) &&
00704             (bflags[VAPBS_BACK] == 1)) yok = 0.5;
00705         else if ((VABS(atom->position[1] - upperCorner[1]) <
00706                   VPMGSMALL) &&
00707             (bflags[VAPBS_FRONT] == 0)) yok = 1;
00708         else if ((VABS(atom->position[1] - upperCorner[1]) <
00709                   VPMGSMALL) &&
00710             (bflags[VAPBS_FRONT] == 1)) yok = 0.5;
00711         else yok = 0;
00712     }
00713     if ((atom->position[2] < upperCorner[2]) &&
00714         (atom->position[2] > lowerCorner[2])) zok = 1;
00715     else {
00716         if ((VABS(atom->position[2] - lowerCorner[2]) < VPMGSMALL
00717             ) &&
00718             (bflags[VAPBS_DOWN] == 0)) zok = 1;
00719         else if ((VABS(atom->position[2] - lowerCorner[2]) <
00720                   VPMGSMALL) &&
00721             (bflags[VAPBS_DOWN] == 1)) zok = 0.5;
00722         else zok = 0;
00723     }
00724     atom->partID = xok*yok*zok;
00725
00726 /* Vnm_print(1, "DEBUG (%s, %d): atom->position[0] - upperCorner[0] = %g\n",
00727

```

```

00722     __FILE__, __LINE__, atom->position[0] - upperCorner[0]);
00723 Vnm_print(1, "DEBUG (%s, %d): atom->position[0] - lowerCorner[0] = %g\n",
00724     __FILE__, __LINE__, atom->position[0] - lowerCorner[0]);
00725 Vnm_print(1, "DEBUG (%s, %d): atom->position[1] - upperCorner[1] = %g\n",
00726     __FILE__, __LINE__, atom->position[1] - upperCorner[1]);
00727 Vnm_print(1, "DEBUG (%s, %d): atom->position[1] - lowerCorner[1] = %g\n",
00728     __FILE__, __LINE__, atom->position[1] - lowerCorner[1]);
00729 Vnm_print(1, "DEBUG (%s, %d): atom->position[2] - upperCorner[2] = %g\n",
00730     __FILE__, __LINE__, atom->position[2] - upperCorner[2]);
00731 Vnm_print(1, "DEBUG (%s, %d): atom->position[2] - lowerCorner[0] = %g\n",
00732     __FILE__, __LINE__, atom->position[2] - lowerCorner[2]);
00733 Vnm_print(1, "DEBUG (%s, %d): xok = %g, yok = %g, zok = %g\n",
00734     __FILE__, __LINE__, xok, yok, zok);
00735     */
00736
00737 }
00738
00739 /* Load up pvec -
00740   For all points within h{axis}/2 of a border - use a gradient
00741   to determine the pvec weight.
00742   Points on the boundary depend on the presence of an adjacent
00743   processor. */
00744
00745 for (i=0; i<(nx*ny*nz); i++) thee->pvec[i] = 0.0;
00746
00747 for (i=0; i<nx; i++) {
00748     xok = 0.0;
00749     x = i*hx + xmin;
00750     if ((x < (upperCorner[0]-hx/2)) &&
00751         (x > (lowerCorner[0]+hx/2)))
00752         xok = 1.0;
00753     else if ((VABS(x - lowerCorner[0]) < VPMGSMALL) &&
00754             (bflags[VAPBS_LEFT] == 0)) xok = 1.0;
00755     else if ((VABS(x - lowerCorner[0]) < VPMGSMALL) &&
00756             (bflags[VAPBS_LEFT] == 1)) xok = 0.5;
00757     else if ((VABS(x - upperCorner[0]) < VPMGSMALL) &&
00758             (bflags[VAPBS_RIGHT] == 0)) xok = 1.0;
00759     else if ((VABS(x - upperCorner[0]) < VPMGSMALL) &&
00760             (bflags[VAPBS_RIGHT] == 1)) xok = 0.5;
00761     else if ((x > (upperCorner[0] + hx/2)) || (x < (lowerCorner[0] - hx/2)))
00762         xok = 0.0;
00763     else if ((x < (upperCorner[0] + hx/2)) || (x > (lowerCorner[0] - hx/2)))
00764     ) {
00765         x0 = VMAX2(x - hx/2, lowerCorner[0]);
00766         x1 = VMIN2(x + hx/2, upperCorner[0]);
00767         xok = VABS(x1-x0)/hx;
00768
00769         if (xok < 0.0) {
00770             if (VABS(xok) < VPMGSMALL) xok = 0.0;
00771             else {
00772                 Vnm_print(2, "Vpmg_setPart: fell off x-interval (%1.12E)!\n",
00773                         xok);
00774                 VASSERT(0);
00775             }
00776         if (xok > 1.0) {
00777             if (VABS(xok - 1.0) < VPMGSMALL) xok = 1.0;
00778             else {
00779                 Vnm_print(2, "Vpmg_setPart: fell off x-interval (%1.12E)!\n",
00780                         xok);
00781                 VASSERT(0);
00782             }
00783         } else xok = 0.0;
00784     }
00785
00786     for (j=0; j<ny; j++) {
00787         yok = 0.0;
00788         y = j*hy + ymin;
00789         if ((y < (upperCorner[1]-hy/2)) && (y > (lowerCorner[1]+hy/2))) yok
00790 = 1.0;
00791         else if ((VABS(y - lowerCorner[1]) < VPMGSMALL) &&
00792             (bflags[VAPBS_BACK] == 0)) yok = 1.0;
00793         else if ((VABS(y - lowerCorner[1]) < VPMGSMALL) &&
00794             (bflags[VAPBS_BACK] == 1)) yok = 0.5;
00795         else if ((VABS(y - upperCorner[1]) < VPMGSMALL) &&
00796             (bflags[VAPBS_FRONT] == 0)) yok = 1.0;
00797         else if ((VABS(y - upperCorner[1]) < VPMGSMALL) &&
00798             (bflags[VAPBS_FRONT] == 1)) yok = 0.5;

```

```

00798         else if ((y > (upperCorner[1] + hy/2)) || (y < (lowerCorner[1] - hy
00799             /2))) yok=0.0;
00800         else if ((y < (upperCorner[1] + hy/2)) || (y > (lowerCorner[1] - hy
00801             /2))) {
00802             y0 = VMAX2(y - hy/2, lowerCorner[1]);
00803             y1 = VMIN2(y + hy/2, upperCorner[1]);
00804             yok = VABS(y1-y0)/hy;
00805             if (yok < 0.0) {
00806                 if (VABS(yok) < VPMGSMALL) yok = 0.0;
00807                 else {
00808                     Vnm_print(2, "Vpmg_setPart: fell off y-interval
00809                         (%1.12E)\n",
00810                         yok);
00811                     VASSERT(0);
00812                 }
00813                 if (yok > 1.0) {
00814                     if (VABS(yok - 1.0) < VPMGSMALL) yok = 1.0;
00815                     else {
00816                         Vnm_print(2, "Vpmg_setPart: fell off y-interval
00817                             (%1.12E)\n",
00818                             yok);
00819                         VASSERT(0);
00820                     }
00821                 }
00822             else yok=0.0;
00823             for (k=0; k<nz; k++) {
00824                 zok = 0.0;
00825                 z = k*hzed + zmin;
00826                 if ((z < (upperCorner[2]-hzed/2)) && (z > (lowerCorner[2]+hzed/
00827                     2))) zok = 1.0;
00828                 else if ((VABS(z - lowerCorner[2]) < VPMGSMALL) &&
00829                     (bflags[VAPBS_DOWN] == 0)) zok = 1.0;
00830                 else if ((VABS(z - lowerCorner[2]) < VPMGSMALL) &&
00831                     (bflags[VAPBS_DOWN] == 1)) zok = 0.5;
00832                 else if ((VABS(z - upperCorner[2]) < VPMGSMALL) &&
00833                     (bflags[VAPBS_UP] == 0)) zok = 1.0;
00834                 else if ((VABS(z - upperCorner[2]) < VPMGSMALL) &&
00835                     (bflags[VAPBS_UP] == 1)) zok = 0.5;
00836                 else if ((z > (upperCorner[2] + hzed/2)) || (z < (lowerCorner[2
00837                     ] - hzed/2))) zok=0.0;
00838                 else if ((z < (upperCorner[2] + hzed/2)) || (z > (lowerCorner[2
00839                     ] - hzed/2))) {
00840                     z0 = VMAX2(z - hzed/2, lowerCorner[2]);
00841                     z1 = VMIN2(z + hzed/2, upperCorner[2]);
00842                     zok = VABS(z1-z0)/hzed;
00843                     if (zok < 0.0) {
00844                         if (VABS(zok) < VPMGSMALL) zok = 0.0;
00845                         else {
00846                             Vnm_print(2, "Vpmg_setPart: fell off z-interval
00847                                 (%1.12E)\n",
00848                                 zok);
00849                             VASSERT(0);
00850                         }
00851                         if (zok > 1.0) {
00852                             if (VABS(zok - 1.0) < VPMGSMALL) zok = 1.0;
00853                             else {
00854                                 Vnm_print(2, "Vpmg_setPart: fell off z-interval
00855                                     (%1.12E)\n",
00856                                     zok);
00857                             }
00858                         }
00859                     else zok = 0.0;
00860                     if (VABS(xok*yok*zok) < VPMGSMALL) thee->pvec[IJK(
00861                         i,j,k)] = 0.0;
00862                     else thee->pvec[IJK(i,j,k)] = xok*yok*zok;
00863                 }
00864             }
00865         }
00866     }
00867
00868 VPUBLIC void Vpmg_unsetPart (Vpmg *thee) {

```

```

00869
00870     int i, nx, ny, nz;
00871     Vatom *atom;
00872     Valist *alist;
00873
00874     VASSERT(thee != VNULL);
00875
00876     nx = thee->pmgp->nx;
00877     ny = thee->pmgp->ny;
00878     nz = thee->pmgp->nz;
00879     alist = thee->pbe->alist;
00880
00881     for (i=0; i<(nx*ny*nz); i++) thee->pvec[i] = 1;
00882     for (i=0; i<Valist_getNumberAtoms(alist); i++) {
00883         atom = Valist_getAtom(alist, i);
00884         atom->partID = 1;
00885     }
00886 }
00887
00888 VPUBLIC int Vpmg_fillArray(Vpmg *thee, double *vec,
00889     Vdata_Type type,
00890     double parm, Vhal_PBEType pbetype, PBEparm *pbeparm) {
00891
00892     Vacc *acc = VNULL;
00893     Vpbe *pbe = VNULL;
00894     Vgrid *grid = VNULL;
00895     Vatom *atoms = VNULL;
00896     Valist *alist = VNULL;
00897     double position[3], hx, hy, hzed, xmin, ymin, zmin;
00898     double grad[3], eps, epsp, epss, zmagic;
00899     int i, j, k, l, nx, ny, nz, ichop;
00900
00901     pbe = thee->pbe;
00902     acc = Vpbe_getVacc(pbe);
00903     nx = thee->pmgp->nx;
00904     ny = thee->pmgp->ny;
00905     nz = thee->pmgp->nz;
00906     hx = thee->pmgp->hx;
00907     hy = thee->pmgp->hy;
00908     hzed = thee->pmgp->hzed;
00909     xmin = thee->pmgp->xmin;
00910     ymin = thee->pmgp->ymin;
00911     zmin = thee->pmgp->zmin;
00912     epsp = Vpbe_getSoluteDiel(pbe);
00913     epss = Vpbe_getSolventDiel(pbe);
00914     zmagic = Vpbe_getZmagic(pbe);
00915
00916     if (!thee->filled) {
00917         Vnm_print(2, "Vpmg_fillArray: need to call Vpmg_fillco first!\n");
00918         return 0;
00919     }
00920
00921     switch (type) {
00922         case VDT_CHARGE:
00923
00924             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->charge[i]/zmagic;
00925             break;
00926
00927         case VDT_DIELX:
00928
00929             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->epsx[i];
00930             break;
00931
00932         case VDT_DIELY:
00933
00934             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->epsy[i];
00935             break;
00936
00937         case VDT_DIELZ:
00938
00939             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->epsz[i];
00940             break;
00941
00942         case VDT_KAPPA:
00943
00944             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->kappa[i];
00945             break;
00946
00947         case VDT_POT:
00948

```

```

00949         for (i=0; i<nx*ny*nz; i++) vec[i] = thee->u[i];
00950         break;
00951
00952     case VDT_ATOMPOT:
00953         alist = thee->pbe->alist;
00954         atoms = alist[pbeparm->molid-1].atoms;
00955         grid = Vgrid_ctor(nx, ny, nz, hx, hy,
00956                           hzed, xmin, ymin, zmin, thee->u);
00957         for (i=0; i<alist[pbeparm->molid-1].number; i++) {
00958             position[0] = atoms[i].position[0];
00959             position[1] = atoms[i].position[1];
00960             position[2] = atoms[i].position[2];
00961
00962             Vgrid_value(grid, position, &vec[i]);
00963         }
00964         Vgrid_dtor(&grid);
00965         break;
00966
00967     case VDT_SMOL:
00968
00969         for (k=0; k<nz; k++) {
00970             for (j=0; j<ny; j++) {
00971                 for (i=0; i<nx; i++) {
00972
00973                     position[0] = i*hx + xmin;
00974                     position[1] = j*hy + ymin;
00975                     position[2] = k*hzed + zmin;
00976
00977                     vec[IJK(i,j,k)] = (Vacc_molAcc(acc, position,
00978                                         parm));
00979                 }
00980             }
00981         }
00982         break;
00983
00984     case VDT_SSPL:
00985
00986         for (k=0; k<nz; k++) {
00987             for (j=0; j<ny; j++) {
00988                 for (i=0; i<nx; i++) {
00989
00990                     position[0] = i*hx + xmin;
00991                     position[1] = j*hy + ymin;
00992                     position[2] = k*hzed + zmin;
00993
00994                     vec[IJK(i,j,k)] = Vacc_splineAcc(acc,
00995                                         position, parm, 0);
00996                 }
00997             }
00998         }
00999         break;
01000
01001     case VDT_VDW:
01002
01003         for (k=0; k<nz; k++) {
01004             for (j=0; j<ny; j++) {
01005                 for (i=0; i<nx; i++) {
01006
01007                     position[0] = i*hx + xmin;
01008                     position[1] = j*hy + ymin;
01009                     position[2] = k*hzed + zmin;
01010
01011                     vec[IJK(i,j,k)] = Vacc_vdwAcc(acc, position);
01012                 }
01013             }
01014         }
01015         break;
01016
01017     case VDT_IVDW:
01018
01019         for (k=0; k<nz; k++) {
01020             for (j=0; j<ny; j++) {
01021                 for (i=0; i<nx; i++) {
01022
01023                     position[0] = i*hx + xmin;
01024                     position[1] = j*hy + ymin;
01025                     position[2] = k*hzed + zmin;
01026
01027                     vec[IJK(i,j,k)] = Vacc_ivdwAcc(acc, position
01028                                         , parm);
01029             }
01030         }

```

```

01027             }
01028         }
01029         break;
01030     case VDT_LAP:
01031         grid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin,
01032                           zmin,
01033                           theee->u);
01034         for (k=0; k<nz; k++) {
01035             for (j=0; j<ny; j++) {
01036                 for (i=0; i<nx; i++) {
01037                     if ((k==0) || (k==(nz-1)) ||
01038                         (j==0) || (j==(ny-1)) ||
01039                         (i==0) || (i==(nx-1))) {
01040                         vec[IJK(i,j,k)] = 0;
01041                     } else {
01042                         position[0] = i*hx + xmin;
01043                         position[1] = j*hy + ymin;
01044                         position[2] = k*hzed + zmin;
01045                         VASSERT(Vgrid_curvature(grid,
01046                           position, 1,
01047                           &(vec[IJK(i,j,k)])));
01048                     }
01049                 }
01050             }
01051         }
01052     }
01053 }
01054 Vgrid_dtor(&grid);
01055 break;
01056 case VDT_EDENS:
01057
01058     grid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin,
01059                           zmin,
01060                           theee->u);
01061     for (k=0; k<nz; k++) {
01062         for (j=0; j<ny; j++) {
01063             for (i=0; i<nx; i++) {
01064                 position[0] = i*hx + xmin;
01065                 position[1] = j*hy + ymin;
01066                 position[2] = k*hzed + zmin;
01067                 VASSERT(Vgrid_gradient(grid, position,
01068                           grad));
01069                 eps = epss + (epss-epsp)*Vacc_molAcc(acc,
01070                           position,
01071                           pbe->solventRadius);
01072                 vec[IJK(i,j,k)] = 0.0;
01073                 for (l=0; l<3; l++)
01074                     vec[IJK(i,j,k)] += eps*VSQR(grad[l]);
01075             }
01076         }
01077     }
01078 Vgrid_dtor(&grid);
01079 break;
01080 case VDT_NDENS:
01081
01082     for (k=0; k<nz; k++) {
01083         for (j=0; j<ny; j++) {
01084             for (i=0; i<nx; i++) {
01085                 position[0] = i*hx + xmin;
01086                 position[1] = j*hy + ymin;
01087                 position[2] = k*hzed + zmin;
01088                 vec[IJK(i,j,k)] = 0.0;
01089                 if ( VABS(Vacc_ivdwAcc(acc,
01090                               position, pbe->maxIonRadius) - 1.0)
01091                     < VSMALL) {
01092                     for (l=0; l<pbe->numIon; l++) {
01093                         if (pbetype == PBE_NPBE || pbetype ==
01094                             PBE_SMPBE /* SMPBE Added */ ) {
01095                             vec[IJK(i,j,k)] += (pbe->ionConc[l]
01096                                     * Vcap_exp(-pbe->ionQ[l]*
01097                                     theee->u[IJK(i,j,k)],
01098                                     &ichop));
01099                         } else if (pbetype == PBE_LPBE){
01100                             vec[IJK(i,j,k)] += (pbe->ionConc[l]
01101

```

```

01100     j,k));
01101             }
01102         }
01103     }
01104   }
01105 }
01106 break;
01107
01108
01109 case VDT_QDENS:
01110
01111   for (k=0; k<nz; k++) {
01112     for (j=0; j<ny; j++) {
01113       for (i=0; i<nx; i++) {
01114
01115         position[0] = i*hx + xmin;
01116         position[1] = j*hy + ymin;
01117         position[2] = k*hzed + zmin;
01118         vec[IJK(i,j,k)] = 0.0;
01119         if ( VABS(Vacc_ivdwAcc(acc,
01120           position, pbe->maxIonRadius) - 1.0)
01121 < VSMALL) {
01122           for (l=0; l<pbe->numIon; l++) {
01123             if (pbetype == PBE_NPBE || pbetype ==
01124 PBE_SMPBE /* SMPBE Added */ ) {
01125               vec[IJK(i,j,k)] += (pbe->ionConc[l]
01126                         * pbe->ionQ[l]
01127                         * Vcap_exp(-pbe->ionQ[l]*
01128                           thee->u[IJK(i,j,k)],
01129                           &ichop));
01130             } else if (pbetype == PBE_LPBE) {
01131               vec[IJK(i,j,k)] += (pbe->ionConc[l]
01132                         * pbe->ionQ[l]
01133                         * (1 - pbe->ionQ[l]*thee->u[IJK(i,
01134                           j,k)]);
01135           }
01136         }
01137       break;
01138     default:
01139       Vnm_print(2, "main: Bogus data type (%d)!\n", type);
01140       return 0;
01141       break;
01142     }
01143   }
01144 }
01145 }
01146 return 1;
01147
01148 }
01149 }
01150
01151 VPRIVATE double Vpmg_polarizEnergy(Vpmg *thee,
01152                               int extFlag
01153                               ) {
01154
01155   int i,
01156     j,
01157     k,
01158     ijk,
01159     nx,
01160     ny,
01161     nz,
01162     iatom;
01163   double xmin,
01164     ymin,
01165     zmin,
01166     //x, // gcc: not used
01167     //y,
01168     //z,
01169     hx,
01170     hy,
01171     hzed,
01172     epsp,
01173     lap,
01174     pt[3],
01175     T,

```

```

01176      pre,
01177      polq,
01178      dist2,
01179      dist,
01180      energy,
01181      q,
01182      *charge,
01183      *pos,
01184      eps_w;
01185  Vgrid *potgrid;
01186  Vpbe *pbe;
01187  Valist *alist;
01188  Vatom *atom;
01189
01190  xmin = theee->pmgp->xmin;
01191  ymin = theee->pmgp->ymin;
01192  zmin = theee->pmgp->ymin;
01193  hx = theee->pmgp->hx;
01194  hy = theee->pmgp->hy;
01195  hzed = theee->pmgp->hzed;
01196  nx = theee->pmgp->nx;
01197  ny = theee->pmgp->ny;
01198  nz = theee->pmgp->nz;
01199  pbe = theee->pbe;
01200  epsp = Vpbe_getSoluteDiel(pbe);
01201  eps_w = Vpbe_getSolventDiel(pbe);
01202  alist = pbe->alist;
01203  charge = theee->charge;
01204
01205  /* Calculate the prefactor for Coulombic calculations */
01206  T = Vpbe_getTemperature(pbe);
01207  pre = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*eps_w
  *Vunit_kb*T);
01208  pre = pre*(1.0e10);
01209
01210  /* Set up Vgrid object with solution */
01211  potgrid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin,
theee->u);
01212
01213  /* Calculate polarization charge */
01214  energy = 0.0;
01215  for (i=1; i<(nx-1); i++) {
01216      pt[0] = xmin + hx*i;
01217      for (j=1; j<(ny-1); j++) {
01218          pt[1] = ymin + hy*j;
01219          for (k=1; k<(nz-1); k++) {
01220              pt[2] = zmin + hzed*k;
01221
01222              /* Calculate polarization charge */
01223              VASSERT(Vgrid_curvature(potgrid, pt, 1, &lap));
01224              ijk = IJK(i,j,k);
01225              polq = charge[ijk] + epsp*lap*3.0;
01226
01227              /* Calculate interaction energy with atoms */
01228              if (VABS(polq) > VSMALL) {
01229                  for (iatom=0; iatom<Valist_getNumberAtoms
(alist); iatom++) {
01230                      atom = Valist_getAtom(alist, iatom);
01231                      q = Vatom_getCharge(atom);
01232                      pos = Vatom_getPosition(atom);
01233                      dist2 = VSQR(pos[0]-pt[0]) + VSQR(pos[1]-pt[1]) \
01234                         + VSQR(pos[2]-pt[2]);
01235                      dist = VSQRT(dist2);
01236
01237                      if (dist < VSMALL) {
01238                          Vnm_print(2, "Vpmg_polarizEnergy: atom on grid
point; ignoring!\n");
01239                      } else {
01240                          energy = energy + polq*q/dist;
01241                      }
01242                  }
01243              }
01244          }
01245      }
01246  }
01247
01248  return pre*energy;
01249 }
01250
01251 VPUBLIC double Vpmg_energy (Vpmg *theee,
01252                               int extFlag

```

```

01253                     ) {
01254
01255     double totEnergy = 0.0,
01256         dielEnergy = 0.0,
01257         qmEnergy = 0.0,
01258         qfEnergy = 0.0;
01259
01260     VASSEERT(thee != VNULL);
01261
01262     if ((thee->pmgp->nonlin) && (Vpbe_getBulkIonicStrength
01263     (thee->pbe) > 0.)) {
01264         Vnm_print(0, "Vpmg_energy: calculating full PBE energy\n");
01265         qmEnergy = Vpmg_qmEnergy(thee, extFlag);
01266         Vnm_print(0, "Vpmg_energy: qmEnergy = %1.12E kT\n", qmEnergy);
01267         qfEnergy = Vpmg_qfEnergy(thee, extFlag);
01268         Vnm_print(0, "Vpmg_energy: qfEnergy = %1.12E kT\n", qfEnergy);
01269         dielEnergy = Vpmg_dielEnergy(thee, extFlag);
01270         Vnm_print(0, "Vpmg_energy: dielEnergy = %1.12E kT\n", dielEnergy);
01271         totEnergy = qfEnergy - dielEnergy - qmEnergy;
01272     } else {
01273         Vnm_print(0, "Vpmg_energy: calculating only q-phi energy\n");
01274         qfEnergy = Vpmg_qfEnergy(thee, extFlag);
01275         Vnm_print(0, "Vpmg_energy: qfEnergy = %1.12E kT\n", qfEnergy);
01276         totEnergy = 0.5*qfEnergy;
01277     }
01278
01279     return totEnergy;
01280 }
01281
01282 VPUBLIC double Vpmg_dielEnergy(Vpmg *thee,
01283                                     int extFlag
01284                                     ) {
01285
01286     double hx,
01287            hy,
01288            hzed,
01289            energy,
01290            nrgx,
01291            nrgy,
01292            nrgz,
01293            pvecx,
01294            pvecy,
01295            pvecz;
01296     int i,
01297            j,
01298            k,
01299            nx,
01300            ny,
01301            nz;
01302
01303     VASSEERT(thee != VNULL);
01304
01305     /* Get the mesh information */
01306     nx = thee->pmgp->nx;
01307     ny = thee->pmgp->ny;
01308     nz = thee->pmgp->nz;
01309     hx = thee->pmgp->hx;
01310     hy = thee->pmgp->hy;
01311     hzed = thee->pmgp->hzed;
01312
01313     energy = 0.0;
01314
01315     if (!thee->filled) {
01316         Vnm_print(2, "Vpmg_dielEnergy: Need to call Vpmg_fillco!\n");
01317         VASSEERT(0);
01318     }
01319
01320     for (k=0; k<(nz-1); k++) {
01321         for (j=0; j<(ny-1); j++) {
01322             for (i=0; i<(nx-1); i++) {
01323                 pvecx = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i+1,
01324 j,k)]);
01325                 pvecy = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j+
01326 1,k)]);
01327                 pvecz = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j,
01328 k+1)]);
01329                 nrgx = thee->epsx[IJK(i,j,k)]*pvecx
01330                         * VSQR((thee->u[IJK(i,j,k)]-thee->u[IJK(i+1,j,k)]) / hx);
01331                 nrgy = thee->epsy[IJK(i,j,k)]*pvecy
01332                         * VSQR((thee->u[IJK(i,j,k)]-thee->u[IJK(i,j+1,k)]) / hy);
01333             }
01334         }
01335     }

```

```

01330             nrgz = thee->epsz[IJK(i,j,k)]*pvecz
01331             * VSQR((thee->u[IJK(i,j,k)]-thee->u[IJK(i,j,k+1)])/hzed);
01332             energy += (nrgx + nrgy + nrgz);
01333         }
01334     }
01335 }
01336
01337 energy = 0.5*energy*hx*hy*hzed;
01338 energy = energy/Vpbe_getZmagic(thee->pbe);
01339
01340 if (extFlag == 1) energy += (thee->extDiEnergy);
01341
01342 return energy;
01343 }
01344
01345 VPUBLIC double Vpmg_dielGradNorm(Vpmg *thee) {
01346
01347     double hx, hy, hzed, energy, nrgx, nrgy, nrgz, pvecx, pvecy, pvecz;
01348     int i, j, k, nx, ny, nz;
01349
01350     VASSERT(thee != VNULL);
01351
01352     /* Get the mesh information */
01353     nx = thee->pmgp->nx;
01354     ny = thee->pmgp->ny;
01355     nz = thee->pmgp->nz;
01356     hx = thee->pmgp->hx;
01357     hy = thee->pmgp->hy;
01358     hzed = thee->pmgp->hzed;
01359
01360     energy = 0.0;
01361
01362     if (!thee->filled) {
01363         Vnm_print(2, "Vpmg_dielGradNorm: Need to call Vpmg_fillco!\n");
01364         VASSERT(0);
01365     }
01366
01367     for (k=1; k<nz; k++) {
01368         for (j=1; j<ny; j++) {
01369             for (i=1; i<nx; i++) {
01370                 pvecx = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i-1,
j,k)]);
01371                 pvecy = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j-1,k)]);
01372                 pvecz = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j,k-1)]);
01373                 nrgx = pvecx
01374                 * VSQR((thee->epsx[IJK(i,j,k)]-thee->epsx[IJK(i-1,j,k)]
01375                         ))/hx);
01376                 nrgy = pvecy
01377                 * VSQR((thee->epsy[IJK(i,j,k)]-thee->epsy[IJK(i,j-1,k)]
01378                         ))/hy);
01379                 nrgz = pvecz
01380                 * VSQR((thee->epsz[IJK(i,j,k)]-thee->epsz[IJK(i,j,k-1)]
01381                         ))/hzed);
01382             energy += VSQRT(nrgx + nrgy + nrgz);
01383         }
01384     energy = energy*hx*hy*hzed;
01385
01386     return energy;
01387 }
01388
01389 VPUBLIC double Vpmg_qmEnergy(Vpmg *thee,
01390                                 int extFlag
01391                               ) {
01392
01393     double energy;
01394
01395     if (thee->pbe->ipkey == IPKEY_SMPBE){
01396         energy = Vpmg_qmEnergySMPBE(thee,extFlag);
01397     }else{
01398         energy = Vpmg_qmEnergyNONLIN(thee,extFlag);
01399     }
01400
01401     return energy;
01402 }
01403
01404 VPRIIVATE double Vpmg_qmEnergyNONLIN(Vpmg *thee,

```

```

01405                               int extFlag
01406                               )
01407 {
01408     double hx,
01409     hy,
01410     hzed,
01411     energy,
01412     ionConc[MAXION],
01413     ionRadii[MAXION],
01414     ionQ[MAXION],
01415     zkappa2,
01416     ionstr,
01417     zks2;
01418     int i, /* Loop variable */
01419     j,
01420     nx,
01421     ny,
01422     nz,
01423     nion,
01424     ichop,
01425     nchop,
01426     len; /* Stores number of iterations for loops to avoid multiple
01427     recalculations */
01428
01429     VASSERT(thee != VNULL);
01430
01431     /* Get the mesh information */
01432     nx = thee->pmgp->nx;
01433     ny = thee->pmgp->ny;
01434     nz = thee->pmgp->nz;
01435     hx = thee->pmgp->hx;
01436     hy = thee->pmgp->hy;
01437     hzed = thee->pmgp->hzed;
01438     zkappa2 = Vpbe_getZkappa2(thee->pbe);
01439     ionstr = Vpbe_getBulkIonicStrength(thee->pbe);
01440
01441     /* Bail if we're at zero ionic strength */
01442     if (zkappa2 < VSMALL) {
01443
01444 #ifndef VAPBSQUIET
01445         Vnm_print(0, "Vpmg_qmEnergy: Zero energy for zero ionic strength!\n");
01446 #endif
01447
01448     return 0.0;
01449 }
01450     zks2 = 0.5*zkappa2/ionstr;
01451
01452     if (!thee->filled) {
01453         Vnm_print(2, "Vpmg_qmEnergy: Need to call Vpmg_fillco()!\n");
01454         VASSERT(0);
01455     }
01456
01457     energy = 0.0;
01458     nchop = 0;
01459     Vpbe_getIons(thee->pbe, &nion, ionConc, ionRadii, ionQ);
01460     if (thee->pmgp->nonlin) {
01461         Vnm_print(0, "Vpmg_qmEnergy: Calculating nonlinear energy\n");
01462         for (i=0, len=nx*ny*nz; i<len; i++) {
01463             if (thee->pvec[i]*thee->kappa[i] > VSMALL) {
01464                 for (j=0; j<nion; j++) {
01465                     energy += (thee->pvec[i]*thee->kappa[i]*zks2
01466                                * ionConc[j]
01467                                * (Vcap_exp(-ionQ[j]*thee->u[i], &ichop)-1.0));
01468                     nchop += ichop;
01469                 }
01470             }
01471             if (nchop > 0){
01472                 Vnm_print(2, "Vpmg_qmEnergy: Chopped EXP %d times!\n",nchop);
01473                 Vnm_print(2, "\nERROR! Detected large potential values in energy
01474 evaluation! \nERROR! This calculation failed -- please report to the APBS developers!\n\n");
01475             }
01476         } else {
01477             /* Zkappa2 OK here b/c LPBE approx */
01478             Vnm_print(0, "Vpmg_qmEnergy: Calculating linear energy\n");
01479             for (i=0, len=nx*ny*nz; i<len; i++) {
01480                 if (thee->pvec[i]*thee->kappa[i] > VSMALL)
01481                     energy += (thee->pvec[i]*zkappa2*thee->kappa[i]*VSQR(
thee->u[i]));

```

```

01482         }
01483         energy = 0.5*energy;
01484     }
01485     energy = energy*hx*hy*hzed;
01486     energy = energy/Vpbe_getZmagic(thee->pbe);
01487
01488     if (extFlag == 1) energy += thee->extQmEnergy;
01489
01490     return energy;
01491 }
01492
01493 VPUBLIC double Vpmg_qmEnergySMPBE (Vpmg *thee,
01494                                         int extFlag
01495                                         ) {
01496
01497     double hx,
01498             hy,
01499             hzed,
01500             energy,
01501             ionConc[MAXION],
01502             ionRadii[MAXION],
01503             ionQ[MAXION],
01504             zkappa2,
01505             ionstr,
01506             zks2;
01507     int i,
01508         //j, // gcc: not used
01509         nx,
01510         ny,
01511         nz,
01512         nion,
01513         //ichop, // gcc: not used
01514         nchop,
01515         len; /* Loop variable */
01516
01517 /* SMPB Modification (vchu, 09/21/06)*/
01518 /* variable declarations for SMPB energy terms */
01519     double a,
01520             k,
01521             z1,
01522             z2,
01523             z3,
01524             cb1,
01525             cb2,
01526             cb3,
01527             a1,
01528             a2,
01529             a3,
01530             c1,
01531             c2,
01532             c3,
01533             currEnergy,
01534             fracOccA,
01535             fracOccB,
01536             fracOccC,
01537             phi,
01538             gspark,
01539             denom;
01540
01541     // Na; /**< @todo remove if no conflicts are caused - This constant
01542     // is already defined in vpde.h. no need to redefine. */
01543     int ichop1,
01544         ichop2,
01545         ichop3;
01546
01547     VASSERT(thee != VNULL);
01548
01549     /* Get the mesh information */
01550     nx = thee->pmgp->nx;
01551     ny = thee->pmgp->ny;
01552     nz = thee->pmgp->nz;
01553     hx = thee->pmgp->hx;
01554     hy = thee->pmgp->hy;
01555     hzed = thee->pmgp->hzed;
01556     zkappa2 = Vpbe_getZkappa2(thee->pbe);
01557     ionstr = Vpbe_getBulkIonicStrength(thee->pbe);
01558
01559     /* Bail if we're at zero ionic strength */
01560     if (zkappa2 < VSMALL) {
01561         Vnm_print(0, "Vpmg_qmEnergySMPBE: Zero energy for zero ionic strength!");

```

```

01562 \n");
01563 #endif
01564     return 0.0;
01565 }
01566 zks2 = 0.5*zkappa2/ionstr;
01567
01568 if (!thee->filled) {
01569     Vnm_print(2, "Vpmg_qmEnergySMPBE: Need to call Vpmg_fillco()!\n");
01570     VASSERT(0);
01571 }
01572
01573 energy = 0.0;
01574 nchop = 0;
01575 Vpbe_getIons(thee->pbe, &ion, ionConc, ionRadii, ionQ);
01576
01577 /* SMPB Modification (vchu, 09/21/06) */
01578 /* Extensive modification to the first part of the if statement
01579 where that handles the thee->pmgp->nonlin part. Basically, I've
01580 deleted all of the original code and written my own code that computes
01581 the electrostatic free energy in the SMPB framework. Definitely really hacky
01582 at this stage of the game, but gets the job done. The second part of the
01583 if statement (the part that handles linear poisson-boltzmann) has been
01584 deleted
01585 because there will be no linearized SMPB energy.. */
01586
01587 z1 = ionQ[0];
01588 z2 = ionQ[1];
01589 z3 = ionQ[2];
01590 cb1 = ionConc[0];
01591 cb2 = ionConc[1];
01592 cb3 = ionConc[2];
01593 a = thee->pbe->smvolume;
01594 k = thee->pbe->smsize;
01595
01596 // This constant is defined in vpde.h Do not need to redefine
01597 //Na = 6.022045000e-04; /* Converts from Molar to N/A^3 */
01598
01599 fracOccA = Na*cb1*VCUB(a);
01600 fracOccB = Na*cb2*VCUB(a);
01601 fracOccC = Na*cb3*VCUB(a);
01602
01603 phi = (fracOccA/k) + fracOccB + fracOccC;
01604
01605 if (thee->pmgp->nonlin) {
01606     Vnm_print(0, "Vpmg_qmEnergySMPBE: Calculating nonlinear energy using
SMPB functional!\n");
01607     for (i=0, len=nx*ny*nz; i<len; i++) {
01608         if (((k-1) > VSMALL) && (thee->pvec[i]*thee->kappa[i] > VSMALL)) {
01609             a1 = Vcap_exp(-1.0*z1*thee->u[i], &ichop1);
01610             a2 = Vcap_exp(-1.0*z2*thee->u[i], &ichop2);
01611             a3 = Vcap_exp(-1.0*z3*thee->u[i], &ichop3);
01612
01613             nchop += ichop1 + ichop2 + ichop3;
01614
01615             gpark = (1 - phi + (fracOccA/k)*a1);
01616             denom = VPOW(gpark, k) + VPOW(1-fracOccB-fracOccC, k-1)*(fracOccB*a2+
01617             fracOccC*a3);
01618
01619             if (cb1 > VSMALL) {
01620                 c1 = Na*cb1*VPOW(gpark, k-1)*a1/denom;
01621                 if(c1 != c1) c1 = 0.;
01622             } else c1 = 0.;
01623
01624             if (cb2 > VSMALL) {
01625                 c2 = Na*cb2*VPOW(1-fracOccB-fracOccC,k-1)*a2/denom;
01626                 if(c2 != c2) c2 = 0. ;
01627             } else c2 = 0. ;
01628
01629             if (cb3 > VSMALL) {
01630                 c3 = Na*cb3*VPOW(1-fracOccB-fracOccC,k-1)*a3/denom;
01631                 if(c3 != c3) c3 = 0. ;
01632             } else c3 = 0. ;
01633
01634             currEnergy = k*VLOG((1-(c1*VCUB(a)/k)-c2*VCUB(a)-c3*VCUB(a))/(1-phi))
01635             -(k-1)*VLOG((1-c2*VCUB(a)-c3*VCUB(a))/(1-phi+(fracOccA/k)));
01636
01637             energy += thee->pvec[i]*thee->kappa[i]*currEnergy;
01638
01639         } else if (thee->pvec[i]*thee->kappa[i] > VSMALL) {

```

```

01640
01641     a1 = Vcap_exp(-1.0*z1*thee->u[i], &ichop1);
01642     a2 = Vcap_exp(-1.0*z2*thee->u[i], &ichop2);
01643     a3 = Vcap_exp(-1.0*z3*thee->u[i], &ichop3);
01644
01645     nchop += ichop1 + ichop2 + ichop3;
01646
01647     gpark = (1 - phi + (fracOccA)*a1);
01648     denom = gpark + (fracOccB*a2+fracOccC*a3);
01649
01650     if (cb1 > VSMALL) {
01651         c1 = Na*cb1*a1/denom;
01652         if(c1 != c1) c1 = 0.;
01653     } else c1 = 0.;
01654
01655     if (cb2 > VSMALL) {
01656         c2 = Na*cb2*a2/denom;
01657         if(c2 != c2) c2 = 0.;
01658     } else c2 = 0.;
01659
01660     if (cb3 > VSMALL) {
01661         c3 = Na*cb3*a3/denom;
01662         if(c3 != c3) c3 = 0.;
01663     } else c3 = 0.;
01664
01665     currEnergy = VLOG((1-c1*VCUB(a)-c2*VCUB(a)-c3*VCUB(a))/(1-frcOccA-frcOccB
-fracOccC));
01666
01667     energy += thee->pvec[i]*thee->kappa[i]*currEnergy;
01668 }
01669 }
01670
01671     energy = -energy/VCUB(a);
01672
01673     if (nchop > 0) Vnm_print(2, "Vpmg_qmEnergySMPBE: Chopped EXP %d times!
\n",
01674         nchop);
01675
01676 } else {
01677     /* Zkappa2 OK here b/c LPBE approx */
01678     Vnm_print(0, "Vpmg_qmEnergySMPBE: ERROR: NO LINEAR ENERGY!! Returning
0!\n");
01679
01680     energy = 0.0;
01681
01682 }
01683     energy = energy*hx*hy*hzed;
01684
01685     if (extFlag == 1) energy += thee->extQmEnergy;
01686
01687     return energy;
01688 }
01689
01690 VPUBLIC double Vpmg_qfEnergy(Vpmg *thee,
01691                                     int extFlag
01692                                     )
01693
01694     double energy = 0.0;
01695
01696     VASSERT(thee != VNULL);
01697
01698     if ((thee->useChargeMap) || (thee->chargeMeth ==
01699         VCM_BSP12)) {
01700         energy = Vpmg_qfEnergyVolume(thee, extFlag);
01701     } else {
01702         energy = Vpmg_qfEnergyPoint(thee, extFlag);
01703     }
01704
01705     return energy;
01706
01707 VPRIVATE double Vpmg_qfEnergyPoint(Vpmg *thee,
01708                                     int extFlag
01709                                     )
01710
01711     int iatom, nx, ny, nz, ihi, ilo, jhi, jlo, khi, klo;
01712     double xmax, ymax, zmax, xmin, ymin, zmin, hx, hy
01713 , hzed, ifloat, jfloat;
01714     double charge, kfloat, dx, dy, dz, energy, uval, *position;
01715     double *u;
01716     double *pvec;

```

```

01716     Valist *alist;
01717     Vatom *atom;
01718     Vpbe *pbe;
01719
01720     pbe = thee->pbe;
01721     alist = pbe->alist;
01722     VASSERT(alist != VNULL);
01723
01724     /* Get the mesh information */
01725     nx = thee->pmgp->nx;
01726     ny = thee->pmgp->ny;
01727     nz = thee->pmgp->nz;
01728     hx = thee->pmgp->hx;
01729     hy = thee->pmgp->hy;
01730     hzed = thee->pmgp->hzed;
01731     xmax = thee->pmgp->xmax;
01732     ymax = thee->pmgp->ymax;
01733     zmax = thee->pmgp->zmax;
01734     xmin = thee->pmgp->xmin;
01735     ymin = thee->pmgp->ymin;
01736     zmin = thee->pmgp->zmin;
01737
01738     u = thee->u;
01739     pvec = thee->pvec;
01740
01741     energy = 0.0;
01742
01743     for (iatom=0; iatom<Valist_getNumberAtoms(alist);
01744         iatom++) {
01745         /* Get atomic information */
01746         atom = Valist_getAtom(alist, iatom);
01747
01748         position = VatomGetPosition(atom);
01749         charge = Vatom_getCharge(atom);
01750
01751         /* Figure out which vertices we're next to */
01752         ifloat = (position[0] - xmin)/hx;
01753         jfloat = (position[1] - ymin)/hy;
01754         kfloat = (position[2] - zmin)/hzed;
01755         ihi = (int)ceil(ifloat);
01756         ilo = (int)floor(ifloat);
01757         jhi = (int)ceil(jfloat);
01758         jlo = (int)floor(jfloat);
01759         khi = (int)ceil(kfloat);
01760         klo = (int)floor(kfloat);
01761
01762         if (atom->partID > 0) {
01763
01764             if ((ihi<nx) && (jhi<ny) && (khi<nz) &&
01765                 (ilo>=0) && (jlo>=0) && (klo>=0)) {
01766
01767                 /* Now get trilinear interpolation constants */
01768                 dx = ifloat - (double)(ilo);
01769                 dy = jfloat - (double)(jlo);
01770                 dz = kfloat - (double)(klo);
01771                 uval =
01772                     dx*dy*dz*u[IJK(ihi,jhi,khi)]
01773                     + dx*(1.0-dy)*dz*u[IJK(ihi,jlo,khi)]
01774                     + dx*dy*(1.0-dz)*u[IJK(ihi,jhi,klo)]
01775                     + dx*(1.0-dy)*(1.0-dz)*u[IJK(ihi,jlo,khi)]
01776                     + (1.0-dx)*dy*dz*u[IJK(ilo,jhi,khi)]
01777                     + (1.0-dx)*(1.0-dy)*dz*u[IJK(ilo,jlo,khi)]
01778                     + (1.0-dx)*dy*(1.0-dz)*u[IJK(ilo,jhi,klo)]
01779                     + (1.0-dx)*(1.0-dy)*(1.0-dz)*u[IJK(ilo,jlo,klo)];
01780                 energy += (uval*charge*atom->partID);
01781             } else if (thee->pmgp->bcl1 != BCFL_FOCUS) {
01782                 Vnm_print(2, "Vpmg_qfEnergy: Atom #d at (%4.3f, %4.3f, \
01783 %4.3f) is off the mesh (ignoring)!\n",
01784                 iatom, position[0], position[1], position[2]);
01785             }
01786         }
01787     }
01788
01789     if (extFlag) energy += thee->extQfEnergy;
01790
01791     return energy;
01792 }
01793
01794 VPUBLIC double Vpmg_qfAtomEnergy(Vpmg *thee, Vatom *
01795     atom) {

```

```

01795
01796     int nx, ny, nz, ihi, ilo, jhi, jlo, khi, klo;
01797     double xmax, xmin, ymax, ymin, zmax, zmin, hx, hy
01798 , hzed, ifloat, jfloat;
01799     double charge, kfloat, dx, dy, dz, energy, uval, *position;
01800     double *u;
01800
01801
01802     /* Get the mesh information */
01803     nx = theee->pmgp->nx;
01804     ny = theee->pmgp->ny;
01805     nz = theee->pmgp->nz;
01806     hx = theee->pmgp->hx;
01807     hy = theee->pmgp->hy;
01808     hzed = theee->pmgp->hzed;
01809     xmax = theee->xf[nx-1];
01810     ymax = theee->yf[ny-1];
01811     zmax = theee->zf[nz-1];
01812     xmin = theee->xf[0];
01813     ymin = theee->yf[0];
01814     zmin = theee->zf[0];
01815
01816     u = theee->u;
01817
01818     energy = 0.0;
01819
01820
01821     position = Vatom_getPosition(atom);
01822     charge = Vatom_getCharge(atom);
01823
01824     /* Figure out which vertices we're next to */
01825     ifloat = (position[0] - xmin)/hx;
01826     jfloat = (position[1] - ymin)/hy;
01827     kfloat = (position[2] - zmin)/hzed;
01828     ihi = (int)ceil(ifloat);
01829     ilo = (int)floor(ifloat);
01830     jhi = (int)ceil(jfloat);
01831     jlo = (int)floor(jfloat);
01832     khi = (int)ceil(kfloat);
01833     klo = (int)floor(kfloat);
01834
01835     if (atom->partID > 0) {
01836
01837         if ((ihi<nx) && (jhi<ny) && (khi<nz) &&
01838             (ilo>=0) && (jlo>=0) && (klo>=0)) {
01839
01840             /* Now get trilinear interpolation constants */
01841             dx = ifloat - (double)(ilo);
01842             dy = jfloat - (double)(jlo);
01843             dz = kfloat - (double)(klo);
01844             uval =
01845                 dx*dy*dz*u[IJK(ihi,jhi,khi)]
01846                 + dx*(1.0-dy)*dz*u[IJK(ihi,jlo,khi)]
01847                 + dx*dy*(1.0-dz)*u[IJK(ihi,jhi,klo)]
01848                 + dx*(1.0-dy)*(1.0-dz)*u[IJK(ihi,jlo,klo)]
01849                 + (1.0-dx)*dy*dz*u[IJK(ilo,jhi,khi)]
01850                 + (1.0-dx)*(1.0-dy)*dz*u[IJK(ilo,jlo,khi)]
01851                 + (1.0-dx)*dy*(1.0-dz)*u[IJK(ilo,jhi,klo)]
01852                 + (1.0-dx)*(1.0-dy)*(1.0-dz)*u[IJK(ilo,jlo,klo)];
01853             energy += (uval*charge*atom->partID);
01854         } else if (theee->pmgp->bclf != BCFL_FOCUS) {
01855             Vnm_print(2, "Vpmg_qfAtomEnergy: Atom at (%4.3f, %4.3f, \
01856 %4.3f) is off the mesh (ignoring)!\n",
01857                     position[0], position[1], position[2]);
01858         }
01859     }
01860
01861     return energy;
01862 }
01863
01864 VPRIVATE double Vpmg_qfEnergyVolume(Vpmg *theee, int
extFlag) {
01865
01866     double hx, hy, hzed, energy;
01867     int i, nx, ny, nz;
01868
01869     VASSERT(theee != VNULL);
01870
01871     /* Get the mesh information */
01872     nx = theee->pmgp->nx;
01873     ny = theee->pmgp->ny;

```

```

01874     nz = thee->pmgp->nz;
01875     hx = thee->pmgp->hx;
01876     hy = thee->pmgp->hy;
01877     hzed = thee->pmgp->hzed;
01878
01879     if (!thee->filled) {
01880         Vnm_Print(2, "Vpmg_qfEnergyVolume: need to call Vpmg_fillco!\n");
01881         VASSERT(0);
01882     }
01883
01884     energy = 0.0;
01885     Vnm_Print(0, "Vpmg_qfEnergyVolume: Calculating energy\n");
01886     for (i=0; i<(nx*ny*nz); i++) {
01887         energy += (thee->pvec[i]*thee->u[i]*thee->charge[i]);
01888     }
01889     energy = energy*hx*hy*hzed/Vpbe_getZmagic(thee->pbe);
01890
01891     if (extFlag == 1) energy += thee->extQfEnergy;
01892
01893     return energy;
01894 }
01895
01896 VPRIPRIVATE void Vpmg_splineSelect(int srfm,Vacc *acc,double
01897     *gpos,double win,
01898             double infrad,Vatom *atom,double *force){
01899     switch (srfm) {
01900     case VSM_SPLINE :
01901         Vacc_splineAccGradAtomNorm(acc, gpos, win, infrad,
01902             atom, force);
01903         break;
01904     case VSM_SPLINE3 :
01905         Vacc_splineAccGradAtomNorm3(acc, gpos, win,
01906             infrad, atom, force);
01907         break;
01908     case VSM_SPLINE4 :
01909         Vacc_splineAccGradAtomNorm4(acc, gpos, win,
01910             infrad, atom, force);
01911         break;
01912     default:
01913         Vnm_Print(2, "Vpmg_dbnbForce: Unknown surface method.\n");
01914         return;
01915     }
01916
01917 VPRIPRIVATE void focusFillBound(Vpmg *thee,
01918                                     Vpmg *pmgOLD
01919                                     )
01920
01921     Vpbe *pbe;
01922     double hxOLD,
01923             hyOLD,
01924             hzOLD,
01925             xminOLD,
01926             yminOLD,
01927             zminOLD,
01928             xmaxOLD,
01929             ymaxOLD,
01930             zmaxOLD,
01931             hxNEW,
01932             hyNEW,
01933             hzNEW,
01934             xminNEW,
01935             yminNEW,
01936             zminNEW,
01937             xmaxNEW,
01938             ymaxNEW,
01939             zmaxNEW,
01940             x,
01941             y,
01942             z,
01943             dx,
01944             dy,
01945             dz,
01946             ifloat,
01947             jfloat,
01948             kfloat,
01949             uval,
01950             eps_w,

```

```

01951      T,
01952      prel,
01953      xkappa,
01954      size,
01955      *apos,
01956      charge,
01957      //pos[3], // gcc: not used
01958      uvalMin,
01959      uvalMax,
01960      *data;
01961      int nxOLD,
01962      nyOLD,
01963      nzOLD,
01964      nxNEW,
01965      nyNEW,
01966      nzNEW,
01967      i,
01968      j,
01969      k,
01970      ihi,
01971      ilo,
01972      jhi,
01973      jlo,
01974      khi,
01975      klo,
01976      nx,
01977      ny,
01978      nz;
01979
01980     /* Calculate new problem dimensions */
01981     hxNEW = thee->pmgp->hx;
01982     hyNEW = thee->pmgp->hy;
01983     hzNEW = thee->pmgp->hz;
01984     nx = thee->pmgp->nx;
01985     ny = thee->pmgp->ny;
01986     nz = thee->pmgp->nz;
01987     nxNEW = thee->pmgp->nx;
01988     nyNEW = thee->pmgp->ny;
01989     nzNEW = thee->pmgp->nz;
01990     xminNEW = thee->pmgp->xcent - ((double)(nxNEW-1)*hxNEW)/2.0;
01991     xmaxNEW = thee->pmgp->xcent + ((double)(nxNEW-1)*hxNEW)/2.0;
01992     yminNEW = thee->pmgp->ycent - ((double)(nyNEW-1)*hyNEW)/2.0;
01993     ymaxNEW = thee->pmgp->ycent + ((double)(nyNEW-1)*hyNEW)/2.0;
01994     zminNEW = thee->pmgp->zcent - ((double)(nzNEW-1)*hzNEW)/2.0;
01995     zmaxNEW = thee->pmgp->zcent + ((double)(nzNEW-1)*hzNEW)/2.0;
01996
01997     if(pmgOLD != VNULL) {
01998     /* Relevant old problem parameters */
01999     hxOLD = pmgOLD->pmgp->hx;
02000     hyOLD = pmgOLD->pmgp->hy;
02001     hzOLD = pmgOLD->pmgp->hz;
02002     nxOLD = pmgOLD->pmgp->nx;
02003     nyOLD = pmgOLD->pmgp->ny;
02004     nzOLD = pmgOLD->pmgp->nz;
02005     xminOLD = pmgOLD->pmgp->xcent - ((double)(nxOLD-1)*hxOLD)/2.0;
02006     xmaxOLD = pmgOLD->pmgp->xcent + ((double)(nxOLD-1)*hxOLD)/2.0;
02007     yminOLD = pmgOLD->pmgp->ycent - ((double)(nyOLD-1)*hyOLD)/2.0;
02008     ymaxOLD = pmgOLD->pmgp->ycent + ((double)(nyOLD-1)*hyOLD)/2.0;
02009     zminOLD = pmgOLD->pmgp->zcent - ((double)(nzOLD-1)*hzOLD)/2.0;
02010     zmaxOLD = pmgOLD->pmgp->zcent + ((double)(nzOLD-1)*hzOLD)/2.0;
02011
02012     data = pmgOLD->u;
02013   } else{
02014     /* Relevant old problem parameters */
02015     hxOLD = thee->potMap->hx;
02016     hyOLD = thee->potMap->hy;
02017     hzOLD = thee->potMap->hz;
02018     nxOLD = thee->potMap->nx;
02019     nyOLD = thee->potMap->ny;
02020     nzOLD = thee->potMap->nz;
02021     xminOLD = thee->potMap->xmin;
02022     xmaxOLD = thee->potMap->xmax;
02023     yminOLD = thee->potMap->ymin;
02024     ymaxOLD = thee->potMap->ymax;
02025     zminOLD = thee->potMap->zmin;
02026     zmaxOLD = thee->potMap->zmax;
02027
02028     data = thee->potMap->data;
02029   }
02030   /* BOUNDARY CONDITION SETUP FOR POINTS OFF OLD MESH:
02031    * For each "atom" (only one for bcfl=1), we use the following formula to

```

```

02032     * calculate the boundary conditions:
02033     *   g(x) = \frac{q_e_c}{4\pi\epsilon_0 k_b T} \frac{\exp(-xkappa*(d-a))}{1+xkappa*a}
02034     *   * 1/d
02035     * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
02036     * We only need to evaluate some of these prefactors once:
02037     *   prel = \frac{e_c}{4\pi\epsilon_0 k_b T}
02038     * which gives the potential as
02039     *   g(x) = prel * q/d * \frac{\exp(-xkappa*(d-a))}{1+xkappa*a}
02040     */
02041     pbe = theee->pbe;
02042     eps_w = Vpbe_getSolventDiel(pbe);           /*
02043     Dimensionless */
02044     T = Vpbe_getTemperature(pbe);                /* K
02045     */
02046     prel = (Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb
02047     *T);
02048     /* Finally, if we convert keep xkappa in A^{-1} and scale prel by
02049     * m/A, then we will only need to deal with distances and sizes in
02050     * Angstroms rather than meters. */
02051     xkappa = Vpbe_getXkappa(pbe);                /* A^{-1}
02052     */
02053     prel = prel*(1.0e10);
02054     size = Vpbe_getSoluteRadius(pbe);
02055     apos = Vpbe_getSoluteCenter(pbe);
02056     charge = Vunit_ec*Vpbe_getSoluteCharge(pbe);
02057
02058     /* Check for rounding error */
02059     if (VABS(xminOLD-xminNEW) < VSMALL) xminNEW = xminOLD;
02060     if (VABS(xmaxOLD-xmaxNEW) < VSMALL) xmaxNEW = xmaxOLD;
02061     if (VABS(yminOLD-yminNEW) < VSMALL) yminNEW = yminOLD;
02062     if (VABS(ymaxOLD-ymaxNEW) < VSMALL) ymaxNEW = ymaxOLD;
02063     if (VABS(zminOLD-zminNEW) < VSMALL) zminNEW = zminOLD;
02064     if (VABS(zmaxOLD-zmaxNEW) < VSMALL) zmaxNEW = zmaxOLD;
02065
02066     /* Sanity check: make sure we're within the old mesh */
02067     Vnm_print(0, "VPMG::focusFillBound -- New mesh mins = %g, %g, %g\n",
02068               xminNEW, yminNEW, zminNEW);
02069     Vnm_print(0, "VPMG::focusFillBound -- New mesh maxs = %g, %g, %g\n",
02070               xmaxNEW, ymaxNEW, zmaxNEW);
02071     Vnm_print(0, "VPMG::focusFillBound -- Old mesh mins = %g, %g, %g\n",
02072               xminOLD, yminOLD, zminOLD);
02073     Vnm_print(0, "VPMG::focusFillBound -- Old mesh maxs = %g, %g, %g\n",
02074               xmaxOLD, ymaxOLD, zmaxOLD);
02075
02076     /* The following is obsolete; we'll substitute analytical boundary
02077     * condition values when the new mesh falls outside the old */
02078     if ((xmaxNEW>xmaxOLD) || (ymaxNEW>ymaxOLD) || (zmaxNEW>zmaxOLD) ||
02079         (xminOLD>xminNEW) || (yminOLD>yminNEW) || (zminOLD>zminNEW)) {
02080
02081         Vnm_print(2, "Vpmg::focusFillBound -- new mesh not contained in old!\n");
02082
02083         Vnm_print(2, "Vpmg::focusFillBound -- old mesh min = (%g, %g, %g)\n",
02084                   xminOLD, yminOLD, zminOLD);
02085         Vnm_print(2, "Vpmg::focusFillBound -- old mesh max = (%g, %g, %g)\n",
02086                   xmaxOLD, ymaxOLD, zmaxOLD);
02087         Vnm_print(2, "Vpmg::focusFillBound -- new mesh min = (%g, %g, %g)\n",
02088                   xminNEW, yminNEW, zminNEW);
02089         Vnm_print(2, "Vpmg::focusFillBound -- new mesh max = (%g, %g, %g)\n",
02090                   xmaxNEW, ymaxNEW, zmaxNEW);
02091         fflush(stderr);
02092         VASSERT(0);
02093     }
02094
02095     uvalMin = VPMGSMAL;
02096     uvalMax = -VPMGSMAL;
02097
02098     /* Fill the "i" boundaries (dirichlet) */
02099     for (k=0; k<nzNEW; k++) {
02100         for (j=0; j<nyNEW; j++) {
02101             /* Low X face */
02102             x = xminNEW;
02103             y = yminNEW + j*hyNEW;
02104             z = zminNEW + k*hzNEW;
02105             if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&
02106                 (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
02107                 ifloat = (x - xminOLD)/hxOLD;

```

```

02106         jfloat = (y - yminOLD)/hyOLD;
02107         kfloat = (z - zminOLD)/hzOLD;
02108         ihi = (int)ceil(ifloat);
02109         if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02110         ilo = (int)floor(ifloat);
02111         if (ilo < 0) ilo = 0;
02112         jhi = (int)ceil(jfloat);
02113         if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02114         jlo = (int)floor(jfloat);
02115         if (jlo < 0) jlo = 0;
02116         khi = (int)ceil(kfloat);
02117         if (khi > (nzOLD-1)) khi = nzOLD-1;
02118         klo = (int)floor(kfloat);
02119         if (klo < 0) klo = 0;
02120         dx = ifloat - (double)(ilo);
02121         dy = jfloat - (double)(jlo);
02122         dz = kfloat - (double)(klo);
02123         nx = nxOLD; ny = nyOLD; nz = nzOLD;
02124         uval = dx*dy*dz*(data[IJK(ihi,jhi,khi)]);
02125         + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02126         + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02127         + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02128         + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02129         + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02130         + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02131         + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02132         nx = nxNEW; ny = nyNEW; nz = nzNEW;
02133     } else {
02134     Vnm_print(2, "focusFillBound (%s, %d): Off old mesh at %g, %g \
02135     %g!\n", __FILE__, __LINE__, x, y, z);
02136     Vnm_print(2, "focusFillBound (%s, %d): old mesh lower corner at \
02137     %g %g.%\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02138     Vnm_print(2, "focusFillBound (%s, %d): old mesh upper corner
at \
02139     %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02140     VASSERT(0);
02141     }
02142     nx = nxNEW; ny = nyNEW; nz = nzNEW;
02143     thee->gxcf[IJKx(j,k,0)] = uval;
02144     if(uval < uvalMin) uvalMin = uval;
02145     if(uval > uvalMax) uvalMax = uval;
02146
02147     /* High X face */
02148     x = xmaxNEW;
02149     if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (
02150     zminOLD-VSMALL)) &&
02151     (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (
02152     zmaxOLD+VSMALL))) {
02153         ifloat = (x - xminOLD)/hxOLD;
02154         jfloat = (y - yminOLD)/hyOLD;
02155         kfloat = (z - zminOLD)/hzOLD;
02156         ihi = (int)ceil(ifloat);
02157         if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02158         ilo = (int)floor(ifloat);
02159         if (ilo < 0) ilo = 0;
02160         jhi = (int)ceil(jfloat);
02161         if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02162         jlo = (int)floor(jfloat);
02163         if (jlo < 0) jlo = 0;
02164         khi = (int)ceil(kfloat);
02165         if (khi > (nzOLD-1)) khi = nzOLD-1;
02166         klo = (int)floor(kfloat);
02167         if (klo < 0) klo = 0;
02168         dx = ifloat - (double)(ilo);
02169         dy = jfloat - (double)(jlo);
02170         dz = kfloat - (double)(klo);
02171         nx = nxOLD; ny = nyOLD; nz = nzOLD;
02172         uval = dx*dy*dz*(data[IJK(ihi,jhi,khi)]);
02173         + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02174         + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02175         + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02176         + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02177         + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jhi,klo)]);
02178         nx = nxNEW; ny = nyNEW; nz = nzNEW;
02179     } else {
02180     Vnm_print(2, "focusFillBound (%s, %d): Off old mesh at %g, %g \
02181     %g!\n", __FILE__, __LINE__, x, y, z);
02182     Vnm_print(2, "focusFillBound (%s, %d): old mesh lower corner at \
02183     %g %g.%\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);

```

```

02184             Vnm_print(2, "focusFillBound (%s, %d): old mesh upper corner
02185         at \
02186             %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02187             VASSERT(0);
02188             }
02189             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02190             thee->gxfc[IJKx(j,k,1)] = uval;
02191             if(uval < uvalMin) uvalMin = uval;
02192             if(uval > uvalMax) uvalMax = uval;
02193             /* Zero Neumann conditions */
02194             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02195             thee->gxfc[IJKx(j,k,2)] = 0.0;
02196             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02197             thee->gxfc[IJKx(j,k,3)] = 0.0;
02198         }
02199     }
02200
02201 /* Fill the "j" boundaries (dirichlet) */
02202 for (k=0; k<nzNEW; k++) {
02203     for (i=0; i<nxNEW; i++) {
02204         /* Low Y face */
02205         x = xminNEW + i*hxNEW;
02206         y = yminNEW;
02207         z = zminNEW + k*hzNEW;
02208         if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (
02209             zminOLD-VSMALL)) &&
02210             (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (
02211             zmaxOLD+VSMALL))) {
02212             ifloat = (x - xminOLD)/hxOLD;
02213             jfloat = (y - yminOLD)/hyOLD;
02214             kfloat = (z - zminOLD)/hzOLD;
02215             ihi = (int)ceil(ifloat);
02216             if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02217             ilo = (int)floor(ifloat);
02218             if (ilo < 0) ilo = 0;
02219             jhi = (int)ceil(jfloat);
02220             if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02221             jlo = (int)floor(jfloat);
02222             if (jlo < 0) jlo = 0;
02223             khi = (int)ceil(kfloat);
02224             if (khi > (nzOLD-1)) khi = nzOLD-1;
02225             klo = (int)floor(kfloat);
02226             if (klo < 0) klo = 0;
02227             dx = ifloat - (double)(ilo);
02228             dy = jfloat - (double)(jlo);
02229             dz = kfloat - (double)(klo);
02230             nx = nxOLD; ny = nyOLD; nz = nzOLD;
02231             uval = dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02232             + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02233             + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02234             + (1.0-dx)*(1.0-dy)*(data[IJK(ihi,jlo,khi)])
02235             + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ihi,jlo,klo)])
02236             + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)]);
02237             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02238         } else {
02239             Vnm_print(2, "focusFillBound (%s, %d): Off old mesh at %g, %g \
02240             %g!\n", __FILE__, __LINE__, x, y, z);
02241             Vnm_print(2, "focusFillBound (%s, %d): old mesh lower corner at \
02242             %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02243             Vnm_print(2, "focusFillBound (%s, %d): old mesh upper corner
02244         at \
02245             %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02246             VASSERT(0);
02247             }
02248             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02249             thee->gycf[IJKy(i,k,0)] = uval;
02250             if(uval < uvalMin) uvalMin = uval;
02251             if(uval > uvalMax) uvalMax = uval;
02252             /* High Y face */
02253             y = ymaxNEW;
02254             if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (
02255                 zminOLD-VSMALL)) &&
02256                 (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (
02257                 zmaxOLD+VSMALL))) {
02258             ifloat = (x - xminOLD)/hxOLD;
02259             jfloat = (y - yminOLD)/hyOLD;
02260             kfloat = (z - zminOLD)/hzOLD;

```

```

02259             ihi = (int)ceil(ifloat);
02260             if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02261             ilo = (int)floor(ifloat);
02262             if (ilo < 0) ilo = 0;
02263             jhi = (int)ceil(jfloat);
02264             if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02265             jlo = (int)floor(jfloat);
02266             if (jlo < 0) jlo = 0;
02267             khi = (int)ceil(kfloat);
02268             if (khi > (nzOLD-1)) khi = nzOLD-1;
02269             klo = (int)floor(kfloat);
02270             if (klo < 0) klo = 0;
02271             dx = ifloat - (double)(ilo);
02272             dy = jfloat - (double)(jlo);
02273             dz = kfloat - (double)(klo);
02274             nx = nxOLD; ny = nyOLD; nz = nzOLD;
02275             uval = dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02276             + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02277             + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02278             + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02279             + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02280             + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02281             + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02282             + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02283             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02284         } else {
02285             Vnm_print(2, "focusFillBound (%s, %d): Off old mesh at %g, %g \
02286             %g!\n", __FILE__, __LINE__, x, y, z);
02287             Vnm_print(2, "focusFillBound (%s, %d): old mesh lower corner at \
02288             %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02289             Vnm_print(2, "focusFillBound (%s, %d): old mesh upper corner
at \
02290             %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02291             VASSERT(0);
02292         }
02293         nx = nxNEW; ny = nyNEW; nz = nzNEW;
02294         thee->gycf[IJKy(i,k,1)] = uval;
02295         if(uval < uvalMin) uvalMin = uval;
02296         if(uval > uvalMax) uvalMax = uval;
02297
02298         /* Zero Neumann conditions */
02299         nx = nxNEW; ny = nyNEW; nz = nzNEW;
0300         thee->gycf[IJKy(i,k,2)] = 0.0;
0301         nx = nxNEW; ny = nyNEW; nz = nzNEW;
0302         thee->gycf[IJKy(i,k,3)] = 0.0;
0303     }
0304 }
0305
0306 /* Fill the "k" boundaries (dirichlet) */
0307 for (j=0; j<nyNEW; j++) {
0308     for (i=0; i<nxNEW; i++) {
0309         /* Low Z face */
0310         x = xminNEW + i*hxNEW;
0311         y = yminNEW + j*hyNEW;
0312         z = zminNEW;
0313         if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&
0314             (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
0315             ifloat = (x - xminOLD)/hxOLD;
0316             jfloat = (y - yminOLD)/hyOLD;
0317             kfloat = (z - zminOLD)/hzOLD;
0318             ihi = (int)ceil(ifloat);
0319             if (ihi > (nxOLD-1)) ihi = nxOLD-1;
0320             ilo = (int)floor(ifloat);
0321             if (ilo < 0) ilo = 0;
0322             jhi = (int)ceil(jfloat);
0323             if (jhi > (nyOLD-1)) jhi = nyOLD-1;
0324             jlo = (int)floor(jfloat);
0325             if (jlo < 0) jlo = 0;
0326             khi = (int)ceil(kfloat);
0327             if (khi > (nzOLD-1)) khi = nzOLD-1;
0328             klo = (int)floor(kfloat);
0329             if (klo < 0) klo = 0;
0330             dx = ifloat - (double)(ilo);
0331             dy = jfloat - (double)(jlo);
0332             dz = kfloat - (double)(klo);
0333             nx = nxOLD; ny = nyOLD; nz = nzOLD;
0334             uval = dx*dy*dz*(data[IJK(ihi,jhi,khi)])
0335             + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
0336             + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])

```

```

02337     + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02338     + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02339     + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02340     + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02341     + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02342         nx = nxNEW; ny = nyNEW; nz = nzNEW;
02343     } else {
02344     Vnm_print(2, "focusFillBound (%s, %d): Off old mesh at %g, %g \
02345         %g!\n", __FILE__, __LINE__, x, y, z);
02346     Vnm_print(2, "focusFillBound (%s, %d): old mesh lower corner at \
02347         %g %g.%\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02348     Vnm_print(2, "focusFillBound (%s, %d): old mesh upper corner
at \
02349         %g %g.%\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02350     VASSERT(0);
02351     }
02352     nx = nxNEW; ny = nyNEW; nz = nzNEW;
02353     thee->gzcf[IJKz(i,j,0)] = uval;
02354     if(uval < uvalMin) uvalMin = uval;
02355     if(uval > uvalMax) uvalMax = uval;
02356
02357     /* High Z face */
02358     z = zmaxNEW;
02359     if((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&
02360         (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
02361         ifloat = (x - xminOLD)/hxOLD;
02362         jfloat = (y - yminOLD)/hyOLD;
02363         kfloat = (z - zminOLD)/hzOLD;
02364         ihi = (int)ceil(ifloat);
02365         if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02366         ilo = (int)floor(ifloat);
02367         if (ilo < 0) ilo = 0;
02368         jhi = (int)ceil(jfloat);
02369         if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02370         jlo = (int)floor(jfloat);
02371         if (jlo < 0) jlo = 0;
02372         khi = (int)ceil(kfloat);
02373         if (khi > (nzOLD-1)) khi = nzOLD-1;
02374         klo = (int)floor(kfloat);
02375         if (klo < 0) klo = 0;
02376         dx = ifloat - (double)(ilo);
02377         dy = jfloat - (double)(jlo);
02378         dz = kfloat - (double)(klo);
02379         nx = nxOLD; ny = nyOLD; nz = nzOLD;
02380         uval = dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02381     + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02382     + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02383     + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02384     + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02385     + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02386     + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02387     + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02388         nx = nxNEW; ny = nyNEW; nz = nzNEW;
02389     } else {
02390     Vnm_print(2, "focusFillBound (%s, %d): Off old mesh at %g, %g \
02391         %g!\n", __FILE__, __LINE__, x, y, z);
02392     Vnm_print(2, "focusFillBound (%s, %d): old mesh lower corner at \
02393         %g %g.%\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02394     Vnm_print(2, "focusFillBound (%s, %d): old mesh upper corner
at \
02395         %g %g.%\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02396     VASSERT(0);
02397     }
02398     nx = nxNEW; ny = nyNEW; nz = nzNEW;
02399     thee->gzcf[IJKz(i,j,1)] = uval;
02400     if(uval < uvalMin) uvalMin = uval;
02401     if(uval > uvalMax) uvalMax = uval;
02402
02403     /* Zero Neumann conditions */
02404     nx = nxNEW; ny = nyNEW; nz = nzNEW;
02405     thee->gzcf[IJKz(i,j,2)] = 0.0;
02406     nx = nxNEW; ny = nyNEW; nz = nzNEW;
02407     thee->gzcf[IJKz(i,j,3)] = 0.0;
02408     }
02409 }
02410
02411 if((uvalMin < SINH_MIN) || (uvalMax > SINH_MAX)){
02412     Vnm_print(2, "\nfocusFillBound: WARNING! Unusually large potential values\n"
\\

```

```

02413      "          detected on the focusing boundary! \n" \
02414      "          Convergence not guaranteed for NPBE/NRPBE
02415  }
02416
02417 }
02418
02419 VPRIVATE void extEnergy(Vpmg *thee, Vpmg *pmgOLD, PBEparm_calcEnergy
02420   extFlag,
02421   double partMin[3], double partMax[3], int bflags[6]) {
02422
02423   Vatom *atom;
02424   double hxNEW, hyNEW, hzNEW;
02425   double lowerCorner[3], upperCorner[3];
02426   int nxNEW, nyNEW, nzNEW;
02427   int nxOLD, nyOLD, nzOLD;
02428   int i,j,k;
02429   double xmin, xmax, ymin, ymax, zmin, zmax;
02430   double hxOLD, hyOLD, hzOLD;
02431   double xval, yval, zval;
02432   double x,y,z;
02433   int nx, ny, nz;
02434
02435 /* Set the new external energy contribution to zero. Any external
02436 * contributions from higher levels will be included in the appropriate
02437 * energy function call. */
02438 thee->extQmEnergy = 0;
02439 thee->extQfEnergy = 0;
02440 thee->extDiEnergy = 0;
02441
02442 /* New problem dimensions */
02443 hxNEW = thee->pmgp->hx;
02444 hyNEW = thee->pmgp->hy;
02445 hzNEW = thee->pmgp->hzed;
02446 nxNEW = thee->pmgp->nx;
02447 nyNEW = thee->pmgp->ny;
02448 nzNEW = thee->pmgp->nz;
02449 lowerCorner[0] = thee->pmgp->xcent - ((double)(nxNEW-1)*hxNEW)/2.0
02450 ;
02451 upperCorner[0] = thee->pmgp->xcent + ((double)(nxNEW-1)*hxNEW)/2.0
02452 ;
02453 lowerCorner[1] = thee->pmgp->ycent - ((double)(nyNEW-1)*hyNEW)/2.0
02454 ;
02455 upperCorner[1] = thee->pmgp->ycent + ((double)(nyNEW-1)*hyNEW)/2.0
02456 ;
02457 lowerCorner[2] = thee->pmgp->zcent - ((double)(nzNEW-1)*hzNEW)/2.0
02458 ;
02459 upperCorner[2] = thee->pmgp->zcent + ((double)(nzNEW-1)*hzNEW)/2.0
02460 ;
02461
02462 Vnm_print(0, "VPMG::extEnergy:  energy flag = %d\n", extFlag);
02463
02464 /* Old problem dimensions */
02465 nxOLD = pmgOLD->pmgp->nx;
02466 nyOLD = pmgOLD->pmgp->ny;
02467 nzOLD = pmgOLD->pmgp->nz;
02468
02469 /* Create a partition based on the new problem dimensions */
02470 /* Vnm_print(1, "DEBUG (%s, %d): extEnergy calling Vpmg_setPart for old
02471 PMG.\n",
02472 __FILE__, __LINE__); */
02473 Vpmg_setPart(pmgOLD, lowerCorner, upperCorner, bflags);
02474
02475
02476 Vnm_print(0,"VPMG::extEnergy:  Finding extEnergy dimensions...\n");
02477 Vnm_print(0,"VPMG::extEnergy  Disj part lower corner = (%g, %g, %g)\n",
02478 partMin[0], partMin[1], partMin[2]);
02479 Vnm_print(0,"VPMG::extEnergy  Disj part upper corner = (%g, %g, %g)\n",
02480 partMax[0], partMax[1], partMax[2]);
02481
02482 /* Find the old dimensions */
02483
02484 hxOLD = pmgOLD->pmgp->hx;
02485 hyOLD = pmgOLD->pmgp->hy;
02486 hzOLD = pmgOLD->pmgp->hzed;
02487 xmin = pmgOLD->pmgp->xcent - 0.5*hxOLD*(nxOLD-1);
02488 ymin = pmgOLD->pmgp->ycent - 0.5*hyOLD*(nyOLD-1);
02489 zmin = pmgOLD->pmgp->zcent - 0.5*hzOLD*(nzOLD-1);
02490 xmax = xmin+hxOLD*(nxOLD-1);
02491 ymax = ymin+hyOLD*(nyOLD-1);
02492 zmax = zmin+hzOLD*(nzOLD-1);

```

```

02485
02486     Vnm_Print(0,"VPMG::extEnergy      Old lower corner = (%g, %g, %g)\n",
02487             xmin, ymin, zmin);
02488     Vnm_Print(0,"VPMG::extEnergy      Old upper corner = (%g, %g, %g)\n",
02489             xmax, ymax, zmax);
02490
02491     /* Flip the partition, but do not include any points that will
02492     be included by another processor */
02493
02494     nx = nxOLD;
02495     ny = nyOLD;
02496     nz = nzOLD;
02497
02498     for(i=0; i<nx; i++) {
02499         xval = 1;
02500         x = i*hxOLD + xmin;
02501         if (x < partMin[0] && bflags[VAPBS_LEFT] == 1) xval = 0;
02502         else if (x > partMax[0] && bflags[VAPBS_RIGHT] == 1) xval =
0;
02503
02504         for(j=0; j<ny; j++) {
02505             yval = 1;
02506             y = j*hyOLD + ymin;
02507             if (y < partMin[1] && bflags[VAPBS_BACK] == 1) yval = 0;
02508             else if (y > partMax[1] && bflags[VAPBS_FRONT] == 1)
02509                 yval = 0;
02510
02511             for(k=0; k<nz; k++) {
02512                 zval = 1;
02513                 z = k*hzOLD + zmin;
02514                 if (z < partMin[2] && bflags[VAPBS_DOWN] == 1) zval =
0;
02515                 else if (z > partMax[2] && bflags[VAPBS_UP] == 1) zval
02516                 = 0;
02517
02518                 if (pmgOLD->pvec[IJK(i,j,k)] > VSMALL) pmgOLD->pvec[IJK
02519                 (i,j,k)] = 1.0;
02520                 pmgOLD->pvec[IJK(i,j,k)] = (1 - (pmgOLD->pvec[IJK(i,j,k
02521                 )])) * (xval*yval*zval);
02522             }
02523         }
02524     }
02525
02526     for (i=0; i<Valist_getNumberAtoms(thee->pbe->alist
02527     ); i++) {
02528         xval=1;
02529         yval=1;
02530         zval=1;
02531         atom = Valist_getAtom(thee->pbe->alist, i);
02532         x = atom->position[0];
02533         y = atom->position[1];
02534         z = atom->position[2];
02535         if (x < partMin[0] && bflags[VAPBS_LEFT] == 1) xval = 0;
02536         else if (x > partMax[0] && bflags[VAPBS_RIGHT] == 1) xval =
0;
02537         if (y < partMin[1] && bflags[VAPBS_BACK] == 1) yval = 0;
02538         else if (y > partMax[1] && bflags[VAPBS_FRONT] == 1) yval =
0;
02539         if (z < partMin[2] && bflags[VAPBS_DOWN] == 1) zval = 0;
02540         else if (z > partMax[2] && bflags[VAPBS_UP] == 1) zval = 0;
02541         if (atom->partID > VSMALL) atom->partID = 1.0;
02542         atom->partID = (1 - atom->partID) * (xval*yval*zval);
02543     }
02544
02545     /* Now calculate the energy on inverted subset of the domain */
02546     thee->extQmEnergy = Vpmg_qmEnergy(pmgOLD, 1);
02547     Vnm_Print(0, "VPMG::extEnergy: extQmEnergy = %g kT\n", thee->extQmEnergy
02548 );
02549     thee->extQfEnergy = Vpmg_qfEnergy(pmgOLD, 1);
02550     Vnm_Print(0, "VPMG::extEnergy: extQfEnergy = %g kT\n", thee->extQfEnergy
02551 );
02552     thee->extDiEnergy = Vpmg_dielEnergy(pmgOLD, 1);
02553     Vnm_Print(0, "VPMG::extEnergy: extDiEnergy = %g kT\n", thee->extDiEnergy
02554 );
02555
02556     Vpmg_unsetPart(pmgOLD);
02557
02558 }
02559
02560 VPRIPRIVATE double bcfl1sp(double size, double *apos, double charge,
02561     double xkappa, double prel, double *pos) {
02562
02563     double dist, val;

```

```

02554
02555     dist = VSQRT(VSQR(pos[0]-apos[0]) + VSQR(pos[1]-apos[1])
02556     + VSQR(pos[2]-apos[2]));
02557     if (xkappa > VSMALL) {
02558         val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02559     / (1+xkappa*size);
02560     } else {
02561         val = pre1*(charge/dist);
02562     }
02563
02564     return val;
02565 }
02566
02567 VPRIVATE void bcf11(double size, double *apos, double charge,
02568     double xkappa, double pre1, double *gxcf, double *gycf, double *gzcf,
02569     double *xf, double *yf, double *zf, int nx, int ny, int nz) {
02570
02571     int i, j, k;
02572     double dist, val;
02573     double gpos[3];
02574
02575     /* the "i" boundaries (dirichlet) */
02576     for (k=0; k<nz; k++) {
02577         gpos[2] = zf[k];
02578         for (j=0; j<ny; j++) {
02579             gpos[1] = yf[j];
02580             gpos[0] = xf[0];
02581             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02582             + VSQR(gpos[2]-apos[2]));
02583             if (xkappa > VSMALL) {
02584                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02585             / (1+xkappa*size);
02586             } else {
02587                 val = pre1*(charge/dist);
02588             }
02589             gxcf[IJKx(j,k,0)] += val;
02590             gpos[0] = xf[nx-1];
02591             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02592             + VSQR(gpos[2]-apos[2]));
02593             if (xkappa > VSMALL) {
02594                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02595             / (1+xkappa*size);
02596             } else {
02597                 val = pre1*(charge/dist);
02598             }
02599             gxcf[IJKx(j,k,1)] += val;
02600         }
02601     }
02602
02603     /* the "j" boundaries (dirichlet) */
02604     for (k=0; k<nz; k++) {
02605         gpos[2] = zf[k];
02606         for (i=0; i<nx; i++) {
02607             gpos[0] = xf[i];
02608             gpos[1] = yf[0];
02609             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02610             + VSQR(gpos[2]-apos[2]));
02611             if (xkappa > VSMALL) {
02612                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02613             / (1+xkappa*size);
02614             } else {
02615                 val = pre1*(charge/dist);
02616             }
02617             gycf[IJKy(i,k,0)] += val;
02618             gpos[1] = yf[ny-1];
02619             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02620             + VSQR(gpos[2]-apos[2]));
02621             if (xkappa > VSMALL) {
02622                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02623             / (1+xkappa*size);
02624             } else {
02625                 val = pre1*(charge/dist);
02626             }
02627             gycf[IJKy(i,k,1)] += val;
02628         }
02629     }
02630
02631     /* the "k" boundaries (dirichlet) */
02632     for (j=0; j<ny; j++) {
02633         gpos[1] = yf[j];
02634         for (i=0; i<nx; i++) {

```

```

02635         gpos[0] = xf[i];
02636         gpos[2] = zf[0];
02637         dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02638     + VSQR(gpos[2]-apos[2]));
02639         if (xkappa > VSMALL) {
02640             val = pre1*(charge/dist)*VEXP (-xkappa*(dist-size))
02641     / (1+xkappa*size);
02642         } else {
02643             val = pre1*(charge/dist);
02644         }
02645         gzcf[IJKz(i,j,0)] += val;
02646         gpos[2] = zf[nz-1];
02647         dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02648     + VSQR(gpos[2]-apos[2]));
02649         if (xkappa > VSMALL) {
02650             val = pre1*(charge/dist)*VEXP (-xkappa*(dist-size))
02651     / (1+xkappa*size);
02652         } else {
02653             val = pre1*(charge/dist);
02654         }
02655         gzcf[IJKz(i,j,1)] += val;
02656     }
02657 }
02658 }
02659
02660 VPRIvATE void bcf12(double size, double *apos,
02661                     double charge, double *dipole, double *quad,
02662                     double xkappa, double eps_p, double eps_w, double T,
02663                     double *gxcf, double *gycf, double *gzcf,
02664                     double *xf, double *yf, double *zf,
02665                     int nx, int ny, int nz) {
02666
02667     int i, j, k;
02668     double val;
02669     double gpos[3], tensor[3];
02670     double ux, uy, uz, xr, yr, zr;
02671     double qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
02672     double dist, pre;
02673
02674     VASSERT(dipole != VNULL);
02675     ux = dipole[0];
02676     uy = dipole[1];
02677     uz = dipole[2];
02678     if (quad != VNULL) {
02679 /* The factor of 1/3 results from using a
02680 traceless quadrupole definition. See, for example,
02681 "The Theory of Intermolecular Forces" by A.J. Stone,
02682 Chapter 3. */
02683     qxx = quad[0] / 3.0;
02684     qxy = quad[1] / 3.0;
02685     qxz = quad[2] / 3.0;
02686     qyx = quad[3] / 3.0;
02687     qyy = quad[4] / 3.0;
02688     qyz = quad[5] / 3.0;
02689     qzx = quad[6] / 3.0;
02690     qzy = quad[7] / 3.0;
02691     qzz = quad[8] / 3.0;
02692     } else {
02693     qxx = 0.0;
02694     qxy = 0.0;
02695     qxz = 0.0;
02696     qyx = 0.0;
02697     qyy = 0.0;
02698     qyz = 0.0;
02699     qzx = 0.0;
02700     qzy = 0.0;
02701     qzz = 0.0;
02702     }
02703
02704     pre = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*
02705     Vunit_kb*T);
02706     pre = pre*(1.0e10);
02707
02708     /* the "i" boundaries (dirichlet) */
02709     for (k=0; k<nz; k++) {
02710         gpos[2] = zf[k];
02711         for (j=0; j<ny; j++) {
02712             gpos[1] = yf[j];
02713             gpos[0] = xf[0];
02714             xr = gpos[0] - apos[0];
02715             yr = gpos[1] - apos[1];

```

```

02715     zr = gpos[2] - apos[2];
02716     dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02717     multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02718     val = precharge*tensor[0];
02719     val -= pre*ux*xr*tensor[1];
02720     val -= pre*uy*yr*tensor[1];
02721     val -= pre*uz*zr*tensor[1];
02722     val += pre*qxx*xr*xr*tensor[2];
02723     val += pre*qyy*yr*yr*tensor[2];
02724     val += pre*qzz*zr*zr*tensor[2];
02725     val += pre*2.0*qxy*xr*yr*tensor[2];
02726     val += pre*2.0*qxz*xr*zr*tensor[2];
02727     val += pre*2.0*qyz*yrszr*tensor[2];
02728     gxcf[IJKx(j,k,0)] += val;
02729
02730     gpos[0] = xf[nx-1];
02731     xr = gpos[0] - apos[0];
02732     dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02733     multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02734     val = pre*charge*tensor[0];
02735     val -= pre*ux*xr*tensor[1];
02736     val -= pre*uy*yr*tensor[1];
02737     val -= pre*uz*zr*tensor[1];
02738     val += pre*qxx*xr*xr*tensor[2];
02739     val += pre*qyy*yr*yr*tensor[2];
02740     val += pre*qzz*zr*zr*tensor[2];
02741     val += pre*2.0*qxy*xr*yr*tensor[2];
02742     val += pre*2.0*qxz*xr*zr*tensor[2];
02743     val += pre*2.0*qyz*yrszr*tensor[2];
02744     gxcf[IJKx(j,k,1)] += val;
02745   }
02746 }
02747
02748 /* the "j" boundaries (dirichlet) */
02749 for (k=0; k<nz; k++) {
02750   gpos[2] = zf[k];
02751   for (i=0; i<nx; i++) {
02752     gpos[0] = xf[i];
02753     gpos[1] = yf[0];
02754     xr = gpos[0] - apos[0];
02755     yr = gpos[1] - apos[1];
02756     zr = gpos[2] - apos[2];
02757     dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02758     multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02759     val = pre*charge*tensor[0];
02760     val -= pre*ux*xr*tensor[1];
02761     val -= pre*uy*yr*tensor[1];
02762     val -= pre*uz*zr*tensor[1];
02763     val += pre*qxx*xr*xr*tensor[2];
02764     val += pre*qyy*yr*yr*tensor[2];
02765     val += pre*qzz*zr*zr*tensor[2];
02766     val += pre*2.0*qxy*xr*yr*tensor[2];
02767     val += pre*2.0*qxz*xr*zr*tensor[2];
02768     val += pre*2.0*qyz*yrszr*tensor[2];
02769     gycf[IJKy(i,k,0)] += val;
02770
02771     gpos[1] = yf[ny-1];
02772     yr = gpos[1] - apos[1];
02773     dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02774     multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02775     val = precharge*tensor[0];
02776     val -= pre*ux*xr*tensor[1];
02777     val -= pre*uy*yr*tensor[1];
02778     val -= pre*uz*zr*tensor[1];
02779     val += pre*qxx*xr*xr*tensor[2];
02780     val += pre*qyy*yr*yr*tensor[2];
02781     val += pre*qzz*zr*zr*tensor[2];
02782     val += pre*2.0*qxy*xr*yr*tensor[2];
02783     val += pre*2.0*qxz*xr*zr*tensor[2];
02784     val += pre*2.0*qyz*yrszr*tensor[2];
02785     gycf[IJKy(i,k,1)] += val;
02786   }
02787 }
02788
02789 /* the "k" boundaries (dirichlet) */
02790 for (j=0; j<ny; j++) {
02791   gpos[1] = yf[j];
02792   for (i=0; i<nx; i++) {
02793     gpos[0] = xf[i];
02794     gpos[2] = zf[0];
02795     xr = gpos[0] - apos[0];

```

```

02796     yr = gpos[1] - apos[1];
02797     zr = gpos[2] - apos[2];
02798     dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02799     multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02800     val = pre*charge*tensor[0];
02801     val -= preux*xr*tensor[1];
02802     val -= preuy*yr*tensor[1];
02803     val -= preuz*zr*tensor[1];
02804     val += pre*qxx*xr*xr*tensor[2];
02805     val += pre*qyy*yr*yr*tensor[2];
02806     val += pre*qzz*zr*zr*tensor[2];
02807     val += pre*2.0*qxy*xr*yr*tensor[2];
02808     val += pre*2.0*qxz*xr*zr*tensor[2];
02809     val += pre*2.0*qyz*yr*zr*tensor[2];
02810     gzcf[IJKz(i,j,0)] += val;
02811
02812     gpos[2] = zf[nz-1];
02813     zr = gpos[2] - apos[2];
02814     dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02815     multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02816     val = pre*charge*tensor[0];
02817     val -= preux*xr*tensor[1];
02818     val -= preuy*yr*tensor[1];
02819     val -= preuz*zr*tensor[1];
02820     val += pre*qxx*xr*xr*tensor[2];
02821     val += pre*qyy*yr*yr*tensor[2];
02822     val += pre*qzz*zr*zr*tensor[2];
02823     val += pre*2.0*qxy*xr*yr*tensor[2];
02824     val += pre*2.0*qxz*xr*zr*tensor[2];
02825     val += pre*2.0*qyz*yr*zr*tensor[2];
02826     gzcf[IJKz(i,j,1)] += val;
02827 }
02828 }
02829 }
02830
02831 VPRIIVATE void bcCalcOrig(Vpmg *thee) {
02832
02833     int nx, ny, nz;
02834     double size, *position, charge, xkappa, eps_w, T, prel;
02835     double *dipole, *quadrupole, debye, eps_p;
02836     double xr, yr, zr, qave, *apos;
02837     double sdhcharge, sdhdipole[3], traced[9], sdhquadrupole[9];
02838     int i, j, k, iatom;
02839     Vpbe *pbe;
02840     Vatom *atom;
02841     Valist *alist;
02842
02843     pbe = thee->pbe;
02844     alist = thee->pbe->alist;
02845     nx = thee->pmgp->nx;
02846     ny = thee->pmgp->ny;
02847     nz = thee->pmgp->nz;
02848
02849     /* Zero out the boundaries */
02850     /* the "x" boundaries (dirichlet) */
02851     for (k=0; k<nz; k++) {
02852         for (j=0; j<ny; j++) {
02853             thee->gxcf[IJKx(j,k,0)] = 0.0;
02854             thee->gxcf[IJKx(j,k,1)] = 0.0;
02855             thee->gxcf[IJKx(j,k,2)] = 0.0;
02856             thee->gxcf[IJKx(j,k,3)] = 0.0;
02857         }
02858     }
02859
02860     /* the "y" boundaries (dirichlet) */
02861     for (k=0; k<nz; k++) {
02862         for (i=0; i<nx; i++) {
02863             thee->gycf[IJKy(i,k,0)] = 0.0;
02864             thee->gycf[IJKy(i,k,1)] = 0.0;
02865             thee->gycf[IJKy(i,k,2)] = 0.0;
02866             thee->gycf[IJKy(i,k,3)] = 0.0;
02867         }
02868     }
02869
02870     /* the "z" boundaries (dirichlet) */
02871     for (j=0; j<ny; j++) {
02872         for (i=0; i<nx; i++) {
02873             thee->gzcf[IJKz(i,j,0)] = 0.0;
02874             thee->gzcf[IJKz(i,j,1)] = 0.0;
02875             thee->gzcf[IJKz(i,j,2)] = 0.0;
02876             thee->gzcf[IJKz(i,j,3)] = 0.0;

```

```

02877         }
02878     }
02879
02880     /* For each "atom" (only one for bcfl=1), we use the following formula to
02881 * calculate the boundary conditions:
02882 *   g(x) = \frac{q e_c}{4\pi\epsilon_0 k_b T} \frac{\exp(-x\kappa_a(d - a))}{1+x\kappa_a a}
02883 *   where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
02884 *   where q = 1/d
02885 *   where \kappa_a = Vpbe_getXkappa(pbe)
02886 *   We only need to evaluate some of these prefactors once:
02887 *   prel = \frac{q e_c}{4\pi\epsilon_0 k_b T} \frac{1}{1+x\kappa_a a}
02888 *   which gives the potential as
02889 *   g(x) = prel * q/d * \frac{\exp(-x\kappa_a(d - a))}{1+x\kappa_a a}
02890 */
02891     eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
02892     eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
02893     T = Vpbe_getTemperature(pbe);                /* K */
02894     prel = (Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb
02895 *T);
02896
02897     /* Finally, if we convert keep xkappa in A^{-1} and scale prel by
02898 * m/A, then we will only need to deal with distances and sizes in
02899 * Angstroms rather than meters. */                  /* A^{-1} */
02900     xkappa = Vpbe_getXkappa(pbe);               /* A^{-1} */
02901 */
02902     prel = prel*(1.0e10);
02903
02904     switch (thee->pmgp->bcfl) {
02905         /* If we have zero boundary conditions, we're done */
02906         case BCFL_ZERO:
02907             return;
02908
02909         /* For single DH sphere BC's, we only have one "atom" to deal with;
02910         * get its information and */
02911         case BCFL_SDH:
02912             size = Vpbe_getSoluteRadius(pbe);
02913             position = Vpbe_getSoluteCenter(pbe);
02914
02915             /* For AMOEBA SDH boundary conditions, we need to find the
02916             * total monopole, dipole and traceless quadrupole moments
02917             * of either the permanent multipoles, induced dipoles or
02918             * non-local induced dipoles.
02919 */
02920             sdhcharge = 0.0;
02921             for (i=0; i<3; i++) sdhdipole[i] = 0.0;
02922             for (i=0; i<9; i++) sdhquadrupole[i] = 0.0;
02923
02924             for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom
02925            ++) {
02926                 atom = Valist_getAtom(alist, iatom);
02927                 apos = Vatom_getPosition(atom);
02928                 xr = apos[0] - position[0];
02929                 yr = apos[1] - position[1];
02930                 zr = apos[2] - position[2];
02931                 switch (thee->chargeSrc) {
02932                     case VCM_CHARGE:
02933                         charge = Vatom_getCharge(atom);
02934                         sdhcharge += charge;
02935                         sdhdipole[0] += xr * charge;
02936                         sdhdipole[1] += yr * charge;
02937                         sdhdipole[2] += zr * charge;
02938                         traced[0] = xr*xr*charge;
02939                         traced[1] = xr*yr*charge;
02940                         traced[2] = xr*zr*charge;
02941                         traced[3] = yr*xr*charge;
02942                         traced[4] = yr*yr*charge;
02943                         traced[5] = yr*zr*charge;
02944                         traced[6] = zr*xr*charge;
02945                         traced[7] = zr*yr*charge;
02946                         traced[8] = zr*zr*charge;
02947                         qave = (traced[0] + traced[4] + traced[8]) / 3.0;
02948                         sdhquadrupole[0] += 1.5*(traced[0] - qave);
02949                         sdhquadrupole[1] += 1.5*(traced[1]);
02950                         sdhquadrupole[2] += 1.5*(traced[2]);
02951                         sdhquadrupole[3] += 1.5*(traced[3]);
02952                         sdhquadrupole[4] += 1.5*(traced[4] - qave);

```

```

02952     sdhquadrupole[5] += 1.5*(traced[5]);
02953     sdhquadrupole[6] += 1.5*(traced[6]);
02954     sdhquadrupole[7] += 1.5*(traced[7]);
02955     sdhquadrupole[8] += 1.5*(traced[8] - qave);
02956 #if defined(WITH_TINKER)
02957     case VCM_PERMANENT:
02958         charge = Vatom_getCharge(atom);
02959         dipole = Vatom_getDipole(atom);
02960         quadrupole = Vatom_getQuadrupole(atom);
02961         sdhcharge += charge;
02962         sdhdipole[0] += xr * charge;
02963         sdhdipole[1] += yr * charge;
02964         sdhdipole[2] += zr * charge;
02965         traced[0] = xr*xr*charge;
02966         traced[1] = xr*yr*charge;
02967         traced[2] = xr*zr*charge;
02968         traced[3] = yr*xr*charge;
02969         traced[4] = yr*yr*charge;
02970         traced[5] = yr*zr*charge;
02971         traced[6] = zr*xr*charge;
02972         traced[7] = zr*yr*charge;
02973         traced[8] = zr*zr*charge;
02974         sdhdipole[0] += dipole[0];
02975         sdhdipole[1] += dipole[1];
02976         sdhdipole[2] += dipole[2];
02977         traced[0] += 2.0*xr*dipole[0];
02978         traced[1] += xr*dipole[1] + yr*dipole[0];
02979         traced[2] += xr*dipole[2] + zr*dipole[0];
02980         traced[3] += yr*dipole[0] + xr*dipole[1];
02981         traced[4] += 2.0*yr*dipole[1];
02982         traced[5] += yr*dipole[2] + zr*dipole[1];
02983         traced[6] += zr*dipole[0] + xr*dipole[2];
02984         traced[7] += zr*dipole[1] + yr*dipole[2];
02985         traced[8] += 2.0*zr*dipole[2];
02986         qave = (traced[0] + traced[4] + traced[8]) / 3.0;
02987         sdhquadrupole[0] += 1.5*(traced[0] - qave);
02988         sdhquadrupole[1] += 1.5*(traced[1]);
02989         sdhquadrupole[2] += 1.5*(traced[2]);
02990         sdhquadrupole[3] += 1.5*(traced[3]);
02991         sdhquadrupole[4] += 1.5*(traced[4] - qave);
02992         sdhquadrupole[5] += 1.5*(traced[5]);
02993         sdhquadrupole[6] += 1.5*(traced[6]);
02994         sdhquadrupole[7] += 1.5*(traced[7]);
02995         sdhquadrupole[8] += 1.5*(traced[8] - qave);
02996         sdhquadrupole[0] += quadrupole[0];
02997         sdhquadrupole[1] += quadrupole[1];
02998         sdhquadrupole[2] += quadrupole[2];
02999         sdhquadrupole[3] += quadrupole[3];
03000         sdhquadrupole[4] += quadrupole[4];
03001         sdhquadrupole[5] += quadrupole[5];
03002         sdhquadrupole[6] += quadrupole[6];
03003         sdhquadrupole[7] += quadrupole[7];
03004         sdhquadrupole[8] += quadrupole[8];
03005     case VCM_INDUCED:
03006         dipole = Vatom_getInducedDipole(atom);
03007         sdhdipole[0] += dipole[0];
03008         sdhdipole[1] += dipole[1];
03009         sdhdipole[2] += dipole[2];
03010         traced[0] = 2.0*xr*dipole[0];
03011         traced[1] = xr*dipole[1] + yr*dipole[0];
03012         traced[2] = xr*dipole[2] + zr*dipole[0];
03013         traced[3] = yr*dipole[0] + xr*dipole[1];
03014         traced[4] = 2.0*yr*dipole[1];
03015         traced[5] = yr*dipole[2] + zr*dipole[1];
03016         traced[6] = zr*dipole[0] + xr*dipole[2];
03017         traced[7] = zr*dipole[1] + yr*dipole[2];
03018         traced[8] = 2.0*zr*dipole[2];
03019         qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03020         sdhquadrupole[0] += 1.5*(traced[0] - qave);
03021         sdhquadrupole[1] += 1.5*(traced[1]);
03022         sdhquadrupole[2] += 1.5*(traced[2]);
03023         sdhquadrupole[3] += 1.5*(traced[3]);
03024         sdhquadrupole[4] += 1.5*(traced[4] - qave);
03025         sdhquadrupole[5] += 1.5*(traced[5]);
03026         sdhquadrupole[6] += 1.5*(traced[6]);
03027         sdhquadrupole[7] += 1.5*(traced[7]);
03028         sdhquadrupole[8] += 1.5*(traced[8] - qave);
03029     case VCM_NLINDUCED:
03030         dipole = Vatom_getNLInducedDipole(atom);
03031         sdhdipole[0] += dipole[0];
03032         sdhdipole[1] += dipole[1];

```

```

03033     sdhdipole[2] += dipole[2];
03034     traced[0] = 2.0*xr*dipole[0];
03035     traced[1] = xr*dipole[1] + yr*dipole[0];
03036     traced[2] = xr*dipole[2] + zr*dipole[0];
03037     traced[3] = yr*dipole[0] + xr*dipole[1];
03038     traced[4] = 2.0*yr*dipole[1];
03039     traced[5] = yr*dipole[2] + zr*dipole[1];
03040     traced[6] = zr*dipole[0] + xr*dipole[2];
03041     traced[7] = zr*dipole[1] + yr*dipole[2];
03042     traced[8] = 2.0*zr*dipole[2];
03043     qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03044     sdhquadrupole[0] += 1.5*(traced[0] - qave);
03045     sdhquadrupole[1] += 1.5*(traced[1]);
03046     sdhquadrupole[2] += 1.5*(traced[2]);
03047     sdhquadrupole[3] += 1.5*(traced[3]);
03048     sdhquadrupole[4] += 1.5*(traced[4] - qave);
03049     sdhquadrupole[5] += 1.5*(traced[5]);
03050     sdhquadrupole[6] += 1.5*(traced[6]);
03051     sdhquadrupole[7] += 1.5*(traced[7]);
03052     sdhquadrupole[8] += 1.5*(traced[8] - qave);
03053 #endif /* if defined(WITH_TINKER) */
03054 }
03055 }
03056 /* SDH dipole and traceless quadrupole values
03057 were checked against similar routines in TINKER
03058 for large proteins.
03059
03060 debye=4.8033324;
03061 printf("%6.3f, %6.3f, %6.3f\n", sdhdipole[0]*debye,
03062     sdhdipole[1]*debye, sdhdipole[2]*debye);
03063 printf("%6.3f\n", sdhquadrupole[0]*debye);
03064 printf("%6.3f %6.3f\n", sdhquadrupole[3]*debye,
03065     sdhquadrupole[4]*debye);
03066 printf("%6.3f %6.3f %6.3f\n", sdhquadrupole[6]*debye,
03067     sdhquadrupole[7]*debye, sdhquadrupole[8]*debye);
03068 */
03069
03070 bcf12(size, position, sdhcharge, sdhdipole, sdhquadrupole,
03071     xkappa, eps_p, eps_w, T, thee->gxcf, thee->gycf,
03072     thee->gzcf, thee->xf, thee->yf, thee->zf, nx, ny, nz);
03073     break;
03074
03075     case BCFL_MDH:
03076     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom
03077    ++) {
03077     atom = Valist_getAtom(alist, iatom);
03078     position = Vatom_getPosition(atom);
03079     charge = Vunit_ec*Vatom_getCharge(atom);
03080     dipole = VNULL;
03081     quadrupole = VNULL;
03082     size = Vatom_getRadius(atom);
03083     switch (thee->chargeSrc)
03084     {
03085     case VCM_CHARGE:
03086     ;
03087 #if defined(WITH_TINKER)
03088     case VCM_PERMANENT:
03089     dipole = Vatom_getDipole(atom);
03090     quadrupole = Vatom_getQuadrupole(atom);
03091
03092     case VCM_INDUCED:
03093     dipole = Vatom_getInducedDipole(atom);
03094
03095     case VCM_NLINDUCED:
03096     dipole = Vatom_getNLIInducedDipole(atom);
03097 #endif
03098     }
03099     bcf11(size, position, charge, xkappa, prel,
03100     thee->gxcf, thee->gycf, thee->gzcf,
03101     thee->xf, thee->yf, thee->zf, nx, ny, nz);
03102     }
03103     break;
03104
03105     case BCFL_UNUSED:
03106     Vnm_print(2, "bcCalc: Invalid bcfl (%d)!\n", thee->pmgp->bcfl
03107     );
03108     VASSERT(0);
03109
03110     case BCFL_FOCUS:
03111     Vnm_print(2, "VPMG::bcCalc -- not appropriate for focusing!\n");
03111     VASSERT(0);

```

```

03112
03113     default:
03114         Vnm_print(2, "VPMG::bcCalc -- invalid boundary condition \
03115 flag (%d)!\n", theee->pmgp->bclf);
03116         VASSEERT(0);
03117     }
03118 }
03119
03120 /*
03121 Used by bcflnew
03122 */
03123 VPRIIVATE int gridPointIsValid(int i, int j, int k, int nx, int ny, int nz){
03124
03125     int isValid = 0;
03126
03127     if((k==0) || (k==nz-1)){
03128         isValid = 1;
03129     }else if((j==0) || (j==ny-1)){
03130         isValid = 1;
03131     }else if((i==0) || (i==nx-1)){
03132         isValid = 1;
03133     }
03134
03135     return isValid;
03136 }
03137
03138 /*
03139 Used by bcflnew
03140 */
03141 #ifdef DEBUG_MAC OSX_OCL
03142 #include "mach_chud.h"
03143 VPRIIVATE void packAtomsOpenCL(float *ax, float *ay, float *az,
03144     float *charge, float *size, Vpmg *thee){
03145
03146     int i;
03147     int natoms;
03148
03149     Vatom *atom = VNULL;
03150     Valist *alist = VNULL;
03151
03152     alist = theee->pbe->alist;
03153     natoms = Valist_getNumberAtoms(alist);
03154
03155     for(i=0;i<natoms;i++){
03156         atom = &(alist->atoms[i]);
03157         charge[i] = Vunit_ec*atom->charge;
03158         ax[i] = atom->position[0];
03159         ay[i] = atom->position[1];
03160         az[i] = atom->position[2];
03161         size[i] = atom->radius;
03162     }
03163 }
03164
03165 /*
03166 Used by bcflnew
03167 */
03168 VPRIIVATE void packUnpackOpenCL(int nx, int ny, int nz, int ngrid,
03169     float *gx, float *gy, float *gz, float *value,
03170     Vpmg *thee, int pack){
03171
03172     int i,j,k,igrid;
03173     int x0,x1,y0,y1,z0,z1;
03174
03175     float gpos[3];
03176     double *xf, *yf, *zf;
03177     double *gxfc, *gycf, *gzcf;
03178
03179     xf = theee->xf;
03180     yf = theee->yf;
03181     zf = theee->zf;
03182
03183     gxfc = theee->gxfc;
03184     gycf = theee->gycf;
03185     gzcf = theee->gzcf;
03186
03187     igrid = 0;
03188     for(k=0;k<nz;k++) {
03189         gpos[2] = zf[k];
03190         for(j=0;j<ny;j++) {
03191             gpos[1] = yf[j];
03192             for(i=0;i<nx;i++) {

```

```

03193     gpos[0] = xf[i];
03194     if(gridPointIsValid(i, j, k, nx, ny, nz)){
03195         if(pack != 0){
03196             gx[igrid] = gpos[0];
03197             gy[igrid] = gpos[1];
03198             gz[igrid] = gpos[2];
03199
03200             value[igrid] = 0.0;
03201         }else{
03202             x0 = IJKx(j,k,0);
03203             x1 = IJKx(j,k,1);
03204             y0 = IJKy(i,k,0);
03205             y1 = IJKy(i,k,1);
03206             z0 = IJKz(i,j,0);
03207             z1 = IJKz(i,j,1);
03208
03209             if(i==0){
03210                 gxcf[x0] += value[igrid];
03211             }
03212             if(i==nx-1){
03213                 gxcf[x1] += value[igrid];
03214             }
03215             if(j==0){
03216                 gycf[y0] += value[igrid];
03217             }
03218             if(j==ny-1){
03219                 gycf[y1] += value[igrid];
03220             }
03221             if(k==0){
03222                 gzcf[z0] += value[igrid];
03223             }
03224             if(k==nz-1){
03225                 gzcf[z1] += value[igrid];
03226             }
03227         }
03228
03229         igrid++;
03230     } //end is valid point
03231 } //end i
03232 } //end j
03233 } //end k
03234
03235 }
03236
03237 /*
03238 bcflnew is an optimized replacement for bcfl1. bcfl1 is still used when TINKER
03239 support is compiled in.
03240 bcflnew uses: packUnpack, packAtoms, gridPointIsValid
03241 */
03242 VPRIIVATE void bcflnewOpenCL(Vpmg *thee) {
03243
03244     int i,j,k, iatom, igrid;
03245     int x0, x1, y0, y1, z0, z1;
03246
03247     int nx, ny, nz;
03248     int natoms, ngrid, ngadj;
03249
03250     float dist, prel, eps_w, eps_p, T, xkappa;
03251
03252     float *ax, *ay, *az;
03253     float *charge, *size, *val;
03254
03255     float *gx, *gy, *gz;
03256
03257     Vpbe *pbe = thee->pbe;
03258
03259     nx = thee->pmgp->nx;
03260     ny = thee->pmgp->ny;
03261     nz = thee->pmgp->nz;
03262
03263     eps_w = Vpbe_getSolventDiel(pbe);           /*
03264     Dimensionless */
03265     eps_p = Vpbe_getSoluteDiel(pbe);           /*
03266     Dimensionless */
03267     T = Vpbe_getTemperature(pbe);                /* K
03268     */
03269     prel = ((Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb
03270     *T))*(.1e10);
03271     xkappa = Vpbe_getXkappa(pbe);
03272
03273     natoms = Valist_getNumberAtoms(thee->pbe->alist);

```

```

03270 ngrid = 2*((nx*ny) + (ny*nz) + (nx*nz));
03271 ngadj = ngrid + (512 - (ngrid & 511));
03272
03273 ax = (float*)malloc(natoms * sizeof(float));
03274 ay = (float*)malloc(natoms * sizeof(float));
03275 az = (float*)malloc(natoms * sizeof(float));
03276
03277 charge = (float*)malloc(natoms * sizeof(float));
03278 size = (float*)malloc(natoms * sizeof(float));
03279
03280 gx = (float*)malloc(ngrid * sizeof(float));
03281 gy = (float*)malloc(ngrid * sizeof(float));
03282 gz = (float*)malloc(ngrid * sizeof(float));
03283
03284 val = (float*)malloc(ngrid * sizeof(float));
03285
03286 packAtomsOpenCL(ax,ay,az,charge,size,thee);
03287 packUnpackOpenCL(nx,ny,nz,ngrid,gx,gy,gz,val,thee,1);
03288
03289 runMDHCL(ngrid,natoms,ngadj,ax,ay,az,gx,gy,gz,charge,size,xkappa,prel,val);
03290
03291 packUnpackOpenCL(nx,ny,nz,ngrid,gx,gy,gz,val,thee,0);
03292
03293 free(ax);
03294 free(ay);
03295 free(az);
03296 free(charge);
03297 free(size);
03298
03299 free(gx);
03300 free(gy);
03301 free(gz);
03302 free(val);
03303 }
03304 #endif
03305
03306 VPRIIVATE void packAtoms(double *ax, double *ay, double *az,
03307     double *charge, double *size, Vpmg *thee){
03308
03309     int i;
03310     int natoms;
03311
03312     Vatom *atom = VNULL;
03313     Valist *alist = VNULL;
03314
03315     alist = thee->pbe->alist;
03316     natoms = Valist_getNumberAtoms(alist);
03317
03318     for(i=0;i<natoms;i++){
03319         atom = &(alist->atoms[i]);
03320         charge[i] = Vunit_ec*atom->charge;
03321         ax[i] = atom->position[0];
03322         ay[i] = atom->position[1];
03323         az[i] = atom->position[2];
03324         size[i] = atom->radius;
03325     }
03326 }
03327
03328 /*
03329 Used by bcflnew
03330 */
03331 VPRIIVATE void packUnpack(int nx, int ny, int nz, int ngrid,
03332     double *gx, double *gy, double *gz, double *value,
03333     Vpmg *thee, int pack){
03334
03335     int i,j,k,igrid;
03336     int x0,x1,y0,y1,z0,z1;
03337
03338     double gpos[3];
03339     double *xf, *yf, *zf;
03340     double *gxfc, *gycf, *gzcf;
03341
03342     xf = thee->xf;
03343     yf = thee->yf;
03344     zf = thee->zf;
03345
03346     gxfc = thee->gxfc;
03347     gycf = thee->gycf;
03348     gzcf = thee->gzcf;
03349
03350     igrid = 0;

```

```

03351   for(k=0; k<nz; k++) {
03352     gpos[2] = zf[k];
03353     for(j=0; j<ny; j++) {
03354       gpos[1] = yf[j];
03355       for(i=0; i<nx; i++) {
03356         gpos[0] = xf[i];
03357         if(gridPointIsValid(i, j, k, nx, ny, nz)) {
03358           if(pack != 0) {
03359             gx[igrid] = gpos[0];
03360             gy[igrid] = gpos[1];
03361             gz[igrid] = gpos[2];
03362             value[igrid] = 0.0;
03363           } else {
03364             x0 = IJKx(j,k,0);
03365             x1 = IJKx(j,k,1);
03366             y0 = IJKy(i,k,0);
03367             y1 = IJKy(i,k,1);
03368             z0 = IJKz(i,j,0);
03369             z1 = IJKz(i,j,1);
03370             if(i==0) {
03371               gxcf[x0] += value[igrid];
03372             }
03373             if(i==nx-1) {
03374               gxcf[x1] += value[igrid];
03375             }
03376             if(j==0) {
03377               gycf[y0] += value[igrid];
03378             }
03379             if(j==ny-1) {
03380               gycf[y1] += value[igrid];
03381             }
03382             if(k==0) {
03383               gzcf[z0] += value[igrid];
03384             }
03385             if(k==nz-1) {
03386               gzcf[z1] += value[igrid];
03387             }
03388           }
03389         }
03390       }
03391       igrid++;
03392     } //end is valid point
03393   } //end i
03394 } //end j
03395 } //end k
03396 }
03397
03398 }
03399
03400 VPRIIVATE void bcflnew(Vpmg *thee) {
03401
03402   int i,j,k, iatom, igrid;
03403   int x0, x1, y0, y1, z0, z1;
03404
03405   int nx, ny, nz;
03406   int natoms, ngrid;
03407
03408   double dist, prel, eps_w, eps_p, T, xkappa;
03409
03410   double *ax, *ay, *az;
03411   double *charge, *size, *val;
03412
03413   double *gx, *gy, *gz;
03414
03415   Vpbe *pbe = thee->pbe;
03416
03417   nx = thee->pmgp->nx;
03418   ny = thee->pmgp->ny;
03419   nz = thee->pmgp->nz;
03420
03421   eps_w = Vpbe_getSolventDiel(pbe); /* Dimensionless */
03422   eps_p = Vpbe_getSoluteDiel(pbe); /* Dimensionless */
03423   T = Vpbe_getTemperature(pbe); /* K */
03424   prel = ((Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb
03425   *T))*(.1.0e10);
03426   xkappa = Vpbe_getXkappa(pbe);
03427   natoms = Valist_getNumberAtoms(thee->pbe->alist);

```

```

03428 ngrid = 2*((nx*ny) + (ny*nz) + (nx*nz));
03429
03430 ax = (double*)malloc(natoms * sizeof(double));
03431 ay = (double*)malloc(natoms * sizeof(double));
03432 az = (double*)malloc(natoms * sizeof(double));
03433
03434 charge = (double*)malloc(natoms * sizeof(double));
03435 size = (double*)malloc(natoms * sizeof(double));
03436
03437 gx = (double*)malloc(ngrid * sizeof(double));
03438 gy = (double*)malloc(ngrid * sizeof(double));
03439 gz = (double*)malloc(ngrid * sizeof(double));
03440
03441 val = (double*)malloc(ngrid * sizeof(double));
03442
03443 packAtoms(ax,ay,az,charge,size,thee);
03444 packUnpack(nx,ny,nz,ngrid,gx,gy,gz,val,thee,1);
03445
03446 if(xkappa > VSMALL){
03447 #pragma omp parallel for default(shared) private(igrid,iatom,dist)
03448   for(igrid=0;igrid<ngrid;igrid++){
03449     for(iatom=0; iatom<natoms; iatom++){
03450       dist = VSQRT(VSQR(gx[igrid]-ax[iatom]) + VSQR(gy[igrid]-ay[iatom])
03451           + VSQR(gz[igrid]-az[iatom]));
03452       val[igrid] += prel*(charge[iatom]/dist)*VEXP(-xkappa*(dist-size[iatom]))
03453         / (1+xkappa*size[iatom]);
03454     }
03455   }
03456 }else{
03457 #pragma omp parallel for default(shared) private(igrid,iatom,dist)
03458   for(igrid=0;igrid<ngrid;igrid++){
03459     for(iatom=0; iatom<natoms; iatom++){
03460       dist = VSQRT(VSQR(gx[igrid]-ax[iatom]) + VSQR(gy[igrid]-ay[iatom])
03461           + VSQR(gz[igrid]-az[iatom]));
03462       val[igrid] += prel*(charge[iatom]/dist);
03463     }
03464   }
03465 }
03466 packUnpack(nx,ny,nz,ngrid,gx,gy,gz,val,thee,0);
03467
03468 free(ax);
03469 free/ay);
03470 free(az);
03471 free(charge);
03472 free(size);
03473
03474 free(gx);
03475 free(gy);
03476 free(gz);
03477 free(val);
03478 }
03479
03480 VPRIvate void multipolebc(double r, double kappa, double eps_p,
03481                               double eps_w, double rad, double tsr[3]) {
03482   double r2,r3,r5;
03483   double eps_r;
03484   double ka,ka2,ka3;
03485   double kr,kr2,kr3;
03486
03487 /*
03488 Below an attempt is made to explain the potential outside of a
03489 multipole located at the center of spherical cavity of dielectric
03490 eps_p, with dielectric eps_w outside (and possibly kappa > 0).
03491
03492
03493 First, eps_p = 1.0
03494 eps_w = 1.0
03495 kappa = 0.0
03496
03497 The general form for the potential of a traceless multipole tensor
03498 of rank n in vacuum is:
03499
03500 V(r) = (-1)^n * u . n . Del^n (1/r)
03501
03502 where
03503 u is a multipole of order n (3^n components)
03504 u . n . Del^n (1/r) is the contraction of u with the nth
03505 derivative of 1/r
03506
03507 for example, if n = 1, the dipole potential is
03508 V_vac(r) = (-1)*[ux*x + uy*y + uz*z]/r^3

```

```

03509
03510 This function returns the parts of V(r) for multipoles of
03511 order 0, 1 and 2 that are independent of the contraction.
03512
03513 For the vacuum example, this would be 1/r, -1/r^3 and 3/r^5
03514 respectively.
03515
03516 *** Note that this requires that the quadrupole is
03517 traceless. If not, the diagonal quadrupole potential changes
03518 from
03519 qaa * 3*a^2/r^5
03520 to
03521 qaa * (3*a^2/r^5 - 1/r^3a )
03522 where we sum over the trace; a = x, y and z.
03523
03524 (In other words, the -1/r^3 term cancels for a traceless quadrupole.
03525 qxx + qyy + qzz = 0
03526 such that
03527 -(qxx + qyy + qzz)/r^3 = 0
03528
03529 For quadrupole with trace:
03530 qxx + qyy + qzz != 0
03531 such that
03532 -(qxx + qyy + qzz)/r^3 != 0
03533 )
03534
03535 =====
03536
03537 eps_p != 1 or eps_w != 1
03538 kappa = 0.0
03539
03540 If the multipole is placed at the center of a sphere with
03541 dielectric eps_p in a solvent of dielectric eps_w, the potential
03542 outside the sphere is the solution to the Laplace equation:
03543
03544 V(r) = 1/eps_w * (2*n+1)*eps_r/(n+(n+1)*eps_r)
03545 * (-1)^n * u . n . Del^n (1/r)
03546 where
03547 eps_r = eps_w / eps_p
03548 is the ratio of solvent to solute dielectric
03549
03550 =====
03551
03552 kappa > 0
03553
03554 Finally, if the region outside the sphere is treated by the linearized
03555 PB equation with Debye-Huckel parameter kappa, the solution is:
03556
03557 V(r) = kappa/eps_w * alpha_n(kappa*a) * K_n(kappa*r) * r^(n+1)/a^n
03558 * (-1)^n * u . n . Del^n (1/r)
03559 where
03560 alpha_n(x) is [(2n + 1) / x] / [(n*K_n(x)/eps_r) - x*K_n'(x)]
03561 K_n(x) are modified spherical Bessel functions of the third kind.
03562 K_n'(x) is the derivative of K_n(x)
03563 */
03564
03565 eps_r = eps_w/eps_p;
03566 r2 = r*r;
03567 r3 = r2*r;
03568 r5 = r3*r2;
03569 tsr[0] = (1.0/eps_w)/r;
03570 tsr[1] = (1.0/eps_w)*(-1.0)/r3;
03571 tsr[2] = (1.0/eps_w)*(3.0)/r5;
03572 if (kappa < VSMALL) {
03573     tsr[1] = (3.0*eps_r)/(1.0 + 2.0*eps_r)*tsr[1];
03574     tsr[2] = (5.0*eps_r)/(2.0 + 3.0*eps_r)*tsr[2];
03575 } else {
03576     ka = kappa*rad;
03577     ka2 = ka*ka;
03578     ka3 = ka2*ka;
03579     kr = kappa*r;
03580     kr2 = kr*kr;
03581     kr3 = kr2*kr;
03582     tsr[0] = exp(ka-kr) / (1.0 + ka) * tsr[0];
03583     tsr[1] = 3.0*eps_r*exp(ka-kr)*(1.0 + kr) * tsr[1];
03584     tsr[1] = tsr[1] / (1.0 + ka + eps_r*(2.0 + 2.0*ka + ka2));
03585     tsr[2] = 5.0*eps_r*exp(ka-kr)*(3.0 + 3.0*kr + kr2) * tsr[2];
03586     tsr[2] = tsr[2]/(6.0+6.0*ka+2.0*ka2+eps_r*(9.0+9.0*ka+4.0*ka2+ka3));
03587 }
03588 }
03589

```

```

03590 VPUBLIC void bcfl_sdh(Vpmg *thee){
03591     int i,j,k,iatom;
03592     int nx, ny, nz;
03593
03594     double size, *position, charge, xkappa, eps_w, eps_p, T, pre, dist;
03595     double sdhcharge, sdhdipole[3], traced[9], sdhquadrupole[9];
03596     double *dipole, *quadrupole;
03597
03598     double val, *apos, gpos[3], tensor[3], qave;
03599     double ux, uy, uz, xr, yr, zr;
03600     double qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
03601
03602     double *xf, *yf, *zf;
03603     double *gxfc, *gycf, *gzcf;
03604
03605     Vpbe *pbe;
03606     Vatom *atom;
03607     Valist *alist;
03608
03609     pbe = thee->pbe;
03610     alist = thee->pbe->alist;
03611     nx = thee->pmgp->nx;
03612     ny = thee->pmgp->ny;
03613     nz = thee->pmgp->nz;
03614
03615     xf = thee->xf;
03616     yf = thee->yf;
03617     zf = thee->zf;
03618
03619     gxfc = thee->gxfc;
03620     gycf = thee->gycf;
03621     gzcf = thee->gzcf;
03622
03623
03624     /* For each "atom" (only one for bcfl=1), we use the following formula to
03625     * calculate the boundary conditions:
03626     *      g(x) = \frac{q e_c}{4\pi\epsilon_0\epsilon_w k_b T} \frac{\exp(-xkappa*(d - a))}{1+xkappa*a}
03627     *      * 1/d
03628     * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
03629     * We only need to evaluate some of these prefactors once:
03630     *      prel = \frac{4\pi\epsilon_0\epsilon_w k_b T}{q}
03631     * which gives the potential as
03632     *      g(x) = prel * q/d * \frac{\exp(-xkappa*(d - a))}{1+xkappa*a}
03633     */
03634
03635     eps_w = Vpbe_getSolventDiel(pbe);           /*
03636     Dimensionless */
03636     eps_p = Vpbe_getSoluteDiel(pbe);           /*
03637     Dimensionless */
03637     T = Vpbe_getTemperature(pbe);                /* K
03638     */
03639
03640     pre = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*
03641     Vunit_kb*T);
03640     pre = pre*(1.0e10);
03641
03642     /* Finally, if we convert keep xkappa in A^{-1} and scale prel by
03643     * m/A, then we will only need to deal with distances and sizes in
03644     * Angstroms rather than meters. */
03645     xkappa = Vpbe_getXkappa(pbe);                /* A^{-1}
03646
03647     /* Solute size and position */
03648     size = Vpbe_getSoluteRadius(pbe);
03649     position = Vpbe_getSoluteCenter(pbe);
03650
03651     sdhcharge = 0.0;
03652     for (i=0; i<3; i++) sdhdipole[i] = 0.0;
03653     for (i=0; i<9; i++) sdhquadrupole[i] = 0.0;
03654
03655     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++)
03656     {
03657         atom = Valist_getAtom(alist, iatom);
03658         apos = VatomGetPosition(atom);
03659         xr = apos[0] - position[0];
03660         yr = apos[1] - position[1];
03661         zr = apos[2] - position[2];
03662         switch (thee->chargeSrc) {
03663             case VCM_CHARGE:
03664                 charge = Vatom_getCharge(atom);
03664                 sdhcharge += charge;

```

```

03665     sdhdipole[0] += xr * charge;
03666     sdhdipole[1] += yr * charge;
03667     sdhdipole[2] += zr * charge;
03668     traced[0] = xx*xr*charge;
03669     traced[1] = xr*yr*charge;
03670     traced[2] = xx*zr*charge;
03671     traced[3] = yr*xr*charge;
03672     traced[4] = yr*yr*charge;
03673     traced[5] = yr*zr*charge;
03674     traced[6] = zx*xr*charge;
03675     traced[7] = zr*yr*charge;
03676     traced[8] = zx*zr*charge;
03677     qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03678     sdhquadrupole[0] += 1.5*(traced[0] - qave);
03679     sdhquadrupole[1] += 1.5*(traced[1]);
03680     sdhquadrupole[2] += 1.5*(traced[2]);
03681     sdhquadrupole[3] += 1.5*(traced[3]);
03682     sdhquadrupole[4] += 1.5*(traced[4] - qave);
03683     sdhquadrupole[5] += 1.5*(traced[5]);
03684     sdhquadrupole[6] += 1.5*(traced[6]);
03685     sdhquadrupole[7] += 1.5*(traced[7]);
03686     sdhquadrupole[8] += 1.5*(traced[8] - qave);
03687 #if defined(WITH_TINKER)
03688 case VCM_PERMANENT:
03689     charge = Vatom_getCharge(atom);
03690     dipole = Vatom_getDipole(atom);
03691     quadrupole = Vatom_getQuadrupole(atom);
03692     sdhcharge += charge;
03693     sdhdipole[0] += xr * charge;
03694     sdhdipole[1] += yr * charge;
03695     sdhdipole[2] += zr * charge;
03696     traced[0] = xr*xr*charge;
03697     traced[1] = xr*yr*charge;
03698     traced[2] = xr*zr*charge;
03699     traced[3] = yr*xr*charge;
03700     traced[4] = yr*yr*charge;
03701     traced[5] = yr*zr*charge;
03702     traced[6] = zx*xr*charge;
03703     traced[7] = zr*yr*charge;
03704     traced[8] = zr*zr*charge;
03705     sdhdipole[0] += dipole[0];
03706     sdhdipole[1] += dipole[1];
03707     sdhdipole[2] += dipole[2];
03708     traced[0] += 2.0*xx*dipole[0];
03709     traced[1] += xr*dipole[1] + yr*dipole[0];
03710     traced[2] += xr*dipole[2] + zr*dipole[0];
03711     traced[3] += yr*dipole[0] + xr*dipole[1];
03712     traced[4] += 2.0*yr*dipole[1];
03713     traced[5] += yr*dipole[2] + zr*dipole[1];
03714     traced[6] += zr*dipole[0] + xr*dipole[2];
03715     traced[7] += zr*dipole[1] + yr*dipole[2];
03716     traced[8] += 2.0*zr*dipole[2];
03717     qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03718     sdhquadrupole[0] += 1.5*(traced[0] - qave);
03719     sdhquadrupole[1] += 1.5*(traced[1]);
03720     sdhquadrupole[2] += 1.5*(traced[2]);
03721     sdhquadrupole[3] += 1.5*(traced[3]);
03722     sdhquadrupole[4] += 1.5*(traced[4] - qave);
03723     sdhquadrupole[5] += 1.5*(traced[5]);
03724     sdhquadrupole[6] += 1.5*(traced[6]);
03725     sdhquadrupole[7] += 1.5*(traced[7]);
03726     sdhquadrupole[8] += 1.5*(traced[8] - qave);
03727     sdhquadrupole[0] += quadrupole[0];
03728     sdhquadrupole[1] += quadrupole[1];
03729     sdhquadrupole[2] += quadrupole[2];
03730     sdhquadrupole[3] += quadrupole[3];
03731     sdhquadrupole[4] += quadrupole[4];
03732     sdhquadrupole[5] += quadrupole[5];
03733     sdhquadrupole[6] += quadrupole[6];
03734     sdhquadrupole[7] += quadrupole[7];
03735     sdhquadrupole[8] += quadrupole[8];
03736 case VCM_INDUCED:
03737     dipole = Vatom_getInducedDipole(atom);
03738     sdhdipole[0] += dipole[0];
03739     sdhdipole[1] += dipole[1];
03740     sdhdipole[2] += dipole[2];
03741     traced[0] = 2.0*xr*dipole[0];
03742     traced[1] = xr*dipole[1] + yr*dipole[0];
03743     traced[2] = xr*dipole[2] + zr*dipole[0];
03744     traced[3] = yr*dipole[0] + xr*dipole[1];
03745     traced[4] = 2.0*yr*dipole[1];

```

```

03746     traced[5] = yr*dipole[2] + zr*dipole[1];
03747     traced[6] = zx*dipole[0] + xr*dipole[2];
03748     traced[7] = zx*dipole[1] + yr*dipole[2];
03749     traced[8] = 2.0*zr*dipole[2];
03750     qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03751     sdhquadrupole[0] += 1.5*(traced[0] - qave);
03752     sdhquadrupole[1] += 1.5*(traced[1]);
03753     sdhquadrupole[2] += 1.5*(traced[2]);
03754     sdhquadrupole[3] += 1.5*(traced[3]);
03755     sdhquadrupole[4] += 1.5*(traced[4] - qave);
03756     sdhquadrupole[5] += 1.5*(traced[5]);
03757     sdhquadrupole[6] += 1.5*(traced[6]);
03758     sdhquadrupole[7] += 1.5*(traced[7]);
03759     sdhquadrupole[8] += 1.5*(traced[8] - qave);
03760 case VCM_NLINDUCED:
03761     dipole = Vatom_getNLInducedDipole(atom);
03762     sdhdipole[0] += dipole[0];
03763     sdhdipole[1] += dipole[1];
03764     sdhdipole[2] += dipole[2];
03765     traced[0] = 2.0*xr*dipole[0];
03766     traced[1] = xr*dipole[1] + yr*dipole[0];
03767     traced[2] = xr*dipole[2] + zr*dipole[0];
03768     traced[3] = yr*dipole[0] + xr*dipole[1];
03769     traced[4] = 2.0*yxr*dipole[1];
03770     traced[5] = yr*dipole[2] + zr*dipole[1];
03771     traced[6] = zx*dipole[0] + xr*dipole[2];
03772     traced[7] = zx*dipole[1] + yr*dipole[2];
03773     traced[8] = 2.0*zr*dipole[2];
03774     qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03775     sdhquadrupole[0] += 1.5*(traced[0] - qave);
03776     sdhquadrupole[1] += 1.5*(traced[1]);
03777     sdhquadrupole[2] += 1.5*(traced[2]);
03778     sdhquadrupole[3] += 1.5*(traced[3]);
03779     sdhquadrupole[4] += 1.5*(traced[4] - qave);
03780     sdhquadrupole[5] += 1.5*(traced[5]);
03781     sdhquadrupole[6] += 1.5*(traced[6]);
03782     sdhquadrupole[7] += 1.5*(traced[7]);
03783     sdhquadrupole[8] += 1.5*(traced[8] - qave);
03784 /*endif /* if defined(WITH_TINKER) */
03785 }
03786 }
03787
03788 ux = sdhdipole[0];
03789 uy = sdhdipole[1];
03790 uz = sdhdipole[2];
03791
03792 /* The factor of 1/3 results from using a
03793 traceless quadrupole definition. See, for example,
03794 "The Theory of Intermolecular Forces" by A.J. Stone,
03795 Chapter 3. */
03796 qxz = sdhquadrupole[0] / 3.0;
03797 qxy = sdhquadrupole[1] / 3.0;
03798 qxz = sdhquadrupole[2] / 3.0;
03799 qyx = sdhquadrupole[3] / 3.0;
03800 qyy = sdhquadrupole[4] / 3.0;
03801 qyz = sdhquadrupole[5] / 3.0;
03802 qzx = sdhquadrupole[6] / 3.0;
03803 qzy = sdhquadrupole[7] / 3.0;
03804 qzz = sdhquadrupole[8] / 3.0;
03805
03806 for(k=0;k<nz;k++) {
03807     gpos[2] = zf[k];
03808     for(j=0;j<ny;j++) {
03809         gpos[1] = yf[j];
03810         for(i=0;i<nx;i++) {
03811             gpos[0] = xf[i];
03812             if(gridPointIsValid(i, j, k, nx, ny, nz)){
03813                 xr = gpos[0] - position[0];
03814                 yr = gpos[1] - position[1];
03815                 zr = gpos[2] - position[2];
03816
03817                 dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
03818                 multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
03819
03820                 val = pre*sdhcharge*tensor[0];
03821                 val -= pre*ux*xr*tensor[1];
03822                 val -= pre*uy*yr*tensor[1];
03823                 val -= pre*uz*zr*tensor[1];
03824                 val += pre*qxx*xr*xr*tensor[2];
03825                 val += pre*qyy*yr*yr*tensor[2];
03826                 val += pre*qzz*zr*zr*tensor[2];

```

```

03827     val += pre*2.0*qxy*xr*yr*tensor[2];
03828     val += pre*2.0*qxz*xr*zr*tensor[2];
03829     val += pre*2.0*qyz*yr*zr*tensor[2];
03830
03831     if(i==0){
03832         gxcf[IJKx(j,k,0)] = val;
03833     }
03834     if(i==nx-1){
03835         gxcf[IJKx(j,k,1)] = val;
03836     }
03837     if(j==0){
03838         gycf[IJKy(i,k,0)] = val;
03839     }
03840     if(j==ny-1){
03841         gycf[IJKy(i,k,1)] = val;
03842     }
03843     if(k==0){
03844         gzcf[IJKz(i,j,0)] = val;
03845     }
03846     if(k==nz-1){
03847         gzcf[IJKz(i,j,1)] = val;
03848     }
03849     } /* End grid point is valid */
03850     } /* End i loop */
03851 } /* End j loop */
03852 } /* End k loop */
03853
03854 }
03855
03856 VPRIIVATE void bcfl_mdh(Vpmg *thee){
03857
03858     int i,j,k,iatom;
03859     int nx, ny, nz;
03860
03861     double val, *apos, gpos[3];
03862     double *dipole, *quadrupole;
03863     double size, charge, xkappa, eps_w, eps_p, T, prel, dist;
03864
03865     double *xf, *yf, *zf;
03866     double *gxcf, *gycf, *gzcf;
03867
03868     Vpbe *pbe;
03869     Vatom *atom;
03870     Valist *alist;
03871
03872     pbe = thee->pbe;
03873     alist = thee->pbe->alist;
03874     nx = thee->pmgp->nx;
03875     ny = thee->pmgp->ny;
03876     nz = thee->pmgp->nz;
03877
03878     xf = thee->xf;
03879     yf = thee->yf;
03880     zf = thee->zf;
03881
03882     gxcf = thee->gxcf;
03883     gycf = thee->gycf;
03884     gzcf = thee->gzcf;
03885
03886     /* For each "atom" (only one for bcfl=1), we use the following formula to
03887      * calculate the boundary conditions:
03888      *   g(x) = \frac{q e_c}{4\pi\epsilon_0\epsilon_w k_b T} \frac{\exp(-xkappa*(d - a))}{1+xkappa*a}
03889      *   * 1/d
03890      * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
03891      * We only need to evaluate some of these prefactors once:
03892      *   prel = \frac{q e_c}{4\pi\epsilon_0\epsilon_w k_b T}
03893      * which gives the potential as
03894      *   g(x) = prel * q/d * \frac{\exp(-xkappa*(d - a))}{1+xkappa*a}
03895      */
03896
03897     eps_w = Vpbe_getSolventDiel(pbe);           /*
03898     Dimensionless */
03899     eps_p = Vpbe_getSoluteDiel(pbe);           /*
03900     Dimensionless */
03901     T = Vpbe_getTemperature(pbe);                /* K
03902     */
03903     prel = (Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb
03904     *T);
03905
03906     /* Finally, if we convert keep xkappa in A^{-1} and scale prel by
03907      * m/A, then we will only need to deal with distances and sizes in

```

```

03904 * Angstroms rather than meters.                                     */
03905 xkappa = Vpbe_getXkappa(pbe);                                     /* A^{-1}
03906 */
03907
03908 /* Finally, if we convert keep xkappa in A^{-1} and scale prel by
03909 * m/A, then we will only need to deal with distances and sizes in
03910 * Angstroms rather than meters.                                         */
03911 xkappa = Vpbe_getXkappa(pbe);                                     /* A^{-1}
03912 */
03913 for(k=0;k<nz;k++) {
03914     gpos[2] = zf[k];
03915     for(j=0;j<ny;j++) {
03916         gpos[1] = yf[j];
03917         for(i=0;i<nx;i++) {
03918             gpos[0] = xf[i];
03919             if(gridPointIsValid(i, j, k, nx, ny, nz)){
03920                 val = 0.0;
03921
03922                 for(iatom=0; iatom<Valist_getNumberAtoms(alist);
03923                     iatom++) {
03924                     atom = Valist_getAtom(alist, iatom);
03925                     apos = Vatom_getPosition(atom);
03926                     charge = Vunit_ec*Vatom_getCharge(atom);
03927                     size = Vatom_getRadius(atom);
03928
03929                     dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
03930                         + VSQR(gpos[2]-apos[2]));
03931                     if(xkappa > VSMALL) {
03932                         val += prel*(charge/dist)*VEXP(-xkappa*(dist-size))
03933                         / (1+xkappa*size);
03934                     } else {
03935                         val += prel*(charge/dist);
03936                     }
03937
03938                 }
03939
03940                 if(i==0){
03941                     gxcf[IJKx(j,k,0)] = val;
03942                 }
03943                 if(i==nx-1){
03944                     gxcf[IJKx(j,k,1)] = val;
03945                 }
03946                 if(j==0){
03947                     gycf[IJKy(i,k,0)] = val;
03948                 }
03949                 if(j==ny-1){
03950                     gycf[IJKy(i,k,1)] = val;
03951                 }
03952                 if(k==0){
03953                     gzcf[IJKz(i,j,0)] = val;
03954                 }
03955                 if(k==nz-1){
03956                     gzcf[IJKz(i,j,1)] = val;
03957                 }
03958             } /* End grid point is valid */
03959         } /* End i loop */
03960     } /* End j loop */
03961 } /* End k loop */
03962
03963 }
03964
03965 /* //////////////////////////////// */
03966 // Routine: bcfl_mem
03967 //
03968 // Purpose: Increment all the boundary points by the
03969 //           analytic expression for a membrane system in
03970 //           the presence of a membrane potential. This
03971 //           Boundary flag should only be used for systems
03972 //           that explicitly have membranes in the dielectric
03973 //           and solvent maps.
03974 //
03975 //           There should be several input variables add to this
03976 //           function such as membrane potential, membrane thickness
03977 //           and height.
03978 //
03979 // Args:      apos is a 3-vector
03980 //
03981 // Author: Michael Grabe

```

```

03983 VPRIVATE void bcfl_mem(double zmem, double L, double eps_m, double eps_w,
03984     double V, double xkappa, double *gxcf, double *gycf, double *gzcf,
03985     double *xf, double *yf, double *zf, int nx, int ny, int nz) {
03986
03988     /* some definitions */ */
03989     /* L = total length of the membrane */ */
03990     /* xkappa = inverse Debye length */ */
03991     /* zmem = z value of membrane bottom (Cytoplasm) */ */
03992     /* V = electrical potential inside the cell */ */
03994     int i, j, k;
03995     double dist, val, z_low, z_high, z_shift;
03996     double A, B, C, D, edge_L, l;
03997     double G, z_0, z_rel;
03998     double gpos[3];
03999
04000     printf("Here is the value of kappa: %f\n", xkappa);
04001     printf("Here is the value of L: %f\n", L);
04002     printf("Here is the value of zmem: %f\n", zmem);
04003     printf("Here is the value of mdie: %f\n", eps_m);
04004     printf("Here is the value of memv: %f\n", V);
04005
04006     /* no salt symmetric BC's at +/- infinity */
04007     // B=V/(edge_L - l*(1-eps_w/eps_m));
04008     // A=V + B*edge_L;
04009     // D=eps_w/eps_m*B;
04010     z_low = zmem;      /* this defines the bottom of the membrane */
04011     z_high = zmem + L; /* this is the top of the membrane */
04012
04013     /*****
04014     /* proper boundary conditions for V = 0 extracellular */
04015     /* and psi=-V cytoplasm. */
04016     /* Implicit in this formulation is that the membrane */
04017     /* center be at z = 0 */
04018     /*****/
04019
04020     l=L/2;           /* half of the membrane length */
04021     z_0 = z_low + l; /* center of the membrane */
04022     G=l*eps_w/eps_m*xkappa;
04023     A=-V/2*(1/(G+1))*exp(xkappa*l);
04024     B=V/2;
04025     C=-V/2*eps_w/eps_m*xkappa*(1/(G+1));
04026     D=-A;
04027     /* The analytic expression for the boundary conditions */
04028     /* had the cytoplasmic surface of the membrane set to zero. */
04029     /* This requires an off-set of the BC equations. */
04030
04031     /* the "i" boundaries (dirichlet) */
04032     for (k=0; k<nz; k++) {
04033         gpos[2] = zf[k];
04034         z_rel = gpos[2] - z_0;    /* relative position for BCs */
04035
04036         for (j=0; j<ny; j++) {
04037
04038             if (gpos[2] <= z_low) {                                /* cytoplasmic */
04039
04040                 val = A*exp(xkappa*z_rel) + V;
04041                 gxcf[IJKx(j,k,0)] += val; /* assign low side BC */
04042                 gxcf[IJKx(j,k,1)] += val; /* assign high side BC */
04043
04044             }
04045
04046             else if (gpos[2] > z_low && gpos[2] <= z_high) { /* in membrane */
04047
04048                 val = B + C*z_rel;
04049                 gxcf[IJKx(j,k,0)] += val; /* assign low side BC */
04050                 gxcf[IJKx(j,k,1)] += val; /* assign high side BC */
04051
04052             }
04053
04054             else if (gpos[2] > z_high) {                            /* extracellular */
04055
04056                 val = D*exp(-xkappa*z_rel);
04057                 gxcf[IJKx(j,k,0)] += val; /* assign low side BC */
04058                 gxcf[IJKx(j,k,1)] += val; /* assign high side BC */
04059
04060             }
04061
04062         }
04063     }
04064
04065     /* the "j" boundaries (dirichlet) */

```

```

04066     for (k=0; k<nz; k++) {
04067         gpos[2] = zf[k];
04068         z_rel = gpos[2] - z_0;
04069         for (i=0; i<nx; i++) {
04070
04071             if (gpos[2] <= z_low) { /* cytoplasmic */
04072
04073                 val = A*exp(xkappa*z_rel) + V;
04074                 gycf[IJKy(i,k,0)] += val; /* assign low side BC */
04075                 gycf[IJKy(i,k,1)] += val; /* assign high side BC */
04076 //printf("%f \n",val);
04077
04078         }
04079
04080         else if (gpos[2] > z_low && gpos[2] <= z_high) { /* in membrane */
04081
04082             val = B + C*z_rel;
04083             gycf[IJKy(i,k,0)] += val; /* assign low side BC */
04084             gycf[IJKy(i,k,1)] += val; /* assign high side BC */
04085 //printf("%f \n",val);
04086
04087         }
04088         else if (gpos[2] > z_high) { /* extracellular */
04089
04090             val = D*exp(-xkappa*z_rel);
04091             gycf[IJKy(i,k,0)] += val; /* assign low side BC */
04092             gycf[IJKy(i,k,1)] += val; /* assign high side BC */
04093 //printf("%f \n",val);
04094
04095     }
04096
04097     }
04098 }
04099
04100 /* the "k" boundaries (dirichlet) */
04101 for (j=0; j<ny; j++) {
04102     for (i=0; i<nx; i++) {
04103
04104     /* first assign the bottom boundary */
04105
04106     gpos[2] = zf[0];
04107     z_rel = gpos[2] - z_0;
04108
04109     if (gpos[2] <= z_low) { /* cytoplasmic */
04110
04111         val = A*exp(xkappa*z_rel) + V;
04112         gycf[IJKz(i,j,0)] += val; /* assign low side BC */
04113 //printf("%f \n",val);
04114
04115     }
04116
04117     else if (gpos[2] > z_low && gpos[2] <= z_high) { /* in membrane */
04118
04119         val = B + C*z_rel;
04120         gycf[IJKz(i,j,0)] += val; /* assign low side BC */
04121
04122     }
04123
04124     else if (gpos[2] > z_high) { /* extracellular */
04125
04126         val = D*exp(-xkappa*z_rel);
04127         gycf[IJKz(i,j,0)] += val; /* assign low side BC */
04128
04129     }
04130
04131     /* now assign the top boundary */
04132
04133     gpos[2] = zf[nz-1];
04134     z_rel = gpos[2] - z_0;
04135
04136     if (gpos[2] <= z_low) { /* cytoplasmic */
04137
04138         val = A*exp(xkappa*z_rel) + V;
04139         gycf[IJKz(i,j,1)] += val; /* assign high side BC */
04140
04141     }
04142
04143     else if (gpos[2] > z_low && gpos[2] <= z_high) { /* in membrane */
04144
04145         val = B + C*z_rel;
04146         gycf[IJKz(i,j,1)] += val; /* assign high side BC */

```

```

04147
04148     }
04149
04150     else if (gpos[2] > z_high) {           /* extracellular */
04151
04152         val = D*exp(-xkappa*z_rel);
04153         gzcfc[IJKz(i,j,1)] += val;      /* assign high side BC */
04154         //printf("%f \n",val);
04155     }
04156 }
04157
04158     }
04159 }
04160 }
04161
04162 VPRIVATE void bcfl_map(Vpmg *thee){
04163
04164     Vpbe *pbe;
04165     double position[3], pot, hx, hy, hzed;
04166     int i, j, k, nx, ny, nz, rc;
04167
04168
04169     VASSERT(thee != VNULL);
04170
04171     /* Mesh info */
04172     nx = thee->pmgp->nx;
04173     ny = thee->pmgp->ny;
04174     nz = thee->pmgp->nz;
04175     hx = thee->pmgp->hx;
04176     hy = thee->pmgp->hy;
04177     hzed = thee->pmgp->hzed;
04178
04179     /* Reset the potential array */
04180     for (i=0; i<(nx*ny*nz); i++) thee->pot[i] = 0.0;
04181
04182     /* Fill in the source term (atomic potentials) */
04183     Vnm_print(0, "Vpmg_fillco: filling in source term.\n");
04184     for (k=0; k<nz; k++) {
04185         for (j=0; j<ny; j++) {
04186             for (i=0; i<nx; i++) {
04187                 position[0] = thee->xf[i];
04188                 position[1] = thee->yf[j];
04189                 position[2] = thee->zf[k];
04190                 rc = Vgrid_value(thee->potMap, position, &pot);
04191                 if (!rc) {
04192                     Vnm_print(2, "fillcoChargeMap: Error -- fell off of potential map at (%g,
04193 %g)! \n",
04194                         position[0], position[1], position[2]);
04195                     VASSERT(0);
04196                 }
04197                 thee->pot[IJK(i,j,k)] = pot;
04198             }
04199         }
04200     }
04201 }
04202
04203 #if defined(WITH_TINKER)
04204 VPRIVATE void bcfl_mdh_tinker(Vpmg *thee){
04205
04206     int i,j,k,iatom;
04207     int nx, ny, nz;
04208
04209     double val, *apos, gpos[3], tensor[9];
04210     double *dipole, *quadrupole;
04211     double size, charge, xkappa, eps_w, eps_p, T, prel, dist;
04212
04213     double ux,uy,uz,xr,yr,zr;
04214     double qxx,qxy,qxz,qyx,qyy,qyz,qzx,qzy,qzz;
04215
04216     double *xf, *yf, *zf;
04217     double *gxfc, *gycf, *gzcf;
04218
04219     Vpbe *pbe;
04220     Vatom *atom;
04221     Valist *alist;
04222
04223     pbe = thee->pbe;
04224     alist = thee->pbe->alist;
04225     nx = thee->pmgp->nx;
04226     ny = thee->pmgp->ny;

```

```

04227     nz = thee->pmgp->nz;
04228
04229     xf = thee->xf;
04230     yf = thee->yf;
04231     zf = thee->zf;
04232
04233     gxcf = thee->gxcf;
04234     gycf = thee->gycf;
04235     gzcdf = thee->gzcdf;
04236
04237     /* For each "atom" (only one for bcfl=1), we use the following formula to
04238     * calculate the boundary conditions:
04239     *   g(x) = \frac{q e_c}{4\pi} \frac{\exp(-xkappa*(d - a))}{(1+xkappa*a)} \frac{1/d}{}
04240     * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
04241     * We only need to evaluate some of these prefactors once:
04242     *   prel = \frac{4\pi e_c}{V^3} \frac{\exp(-xkappa*(d - a))}{(1+xkappa*a)} \frac{1}{d}
04243     * which gives the potential as
04244     *   g(x) = prel * q/d * \frac{\exp(-xkappa*(d - a))}{(1+xkappa*a)}
04245     */
04246
04247     eps_w = Vpbe_getSolventDiel(pbe);           /*
04248     Dimensionless */
04249     eps_p = Vpbe_getSoluteDiel(pbe);           /*
04250     Dimensionless */
04251     T = Vpbe_getTemperature(pbe);                /*
04252     */
04253     prel = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*
04254         Vunit_kb*T);
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
04262     /* Finally, if we convert keep xkappa in A^{-1} and scale prel by
04263     * m/A, then we will only need to deal with distances and sizes in
04264     * Angstroms rather than meters. */
04265     xkappa = Vpbe_getXkappa(pbe);                /* A^{-1}
04266 */
04267
04268     for(k=0;k<nz;k++) {
04269         gpos[2] = zf[k];
04270         for(j=0;j<ny;j++) {
04271             gpos[1] = yf[j];
04272             for(i=0;i<nx;i++) {
04273                 gpos[0] = xf[i];
04274                 if(gridPointIsValid(i, j, k, nx, ny, nz)){
04275                     val = 0.0;
04276
04277                     for (iatom=0; iatom<Valist_getNumberAtoms(alist);
04278                         iatom++) {
04279                         atom = Valist_getAtom(alist, iatom);
04280                         apos = Vatom_getPosition(atom);
04281                         size = Vatom_getRadius(atom);
04282
04283                         charge = 0.0;
04284
04285                         if (thee->chargeSrc == VCM_PERMANENT) {
04286                             charge = Vatom_getCharge(atom);
04287                             dipole = Vatom_getDipole(atom);
04288                             quadrupole = Vatom_getQuadrupole(atom);
04289                         } else if (thee->chargeSrc == VCM_INDUCED) {
04290                             dipole = Vatom_getInducedDipole(atom);
04291                         } else {
04292                             dipole = Vatom_getNLInducedDipole(atom);
04293                         }
04294
04295                         ux = dipole[0];
04296                         uy = dipole[1];
04297                         uz = dipole[2];
04298
04299                         if (quadrupole != VNULL) {
04300                             /* The factor of 1/3 results from using a
04301                                traceless quadrupole definition. See, for example,

```

```

04301      "The Theory of Intermolecular Forces" by A.J. Stone,
04302      Chapter 3. */
04303      qxx = quadrupole[0] / 3.0;
04304      qxy = quadrupole[1] / 3.0;
04305      qxz = quadrupole[2] / 3.0;
04306      qyx = quadrupole[3] / 3.0;
04307      qyy = quadrupole[4] / 3.0;
04308      qyz = quadrupole[5] / 3.0;
04309      qzx = quadrupole[6] / 3.0;
04310      qzy = quadrupole[7] / 3.0;
04311      qzz = quadrupole[8] / 3.0;
04312  } else {
04313      qxx = 0.0;
04314      qxy = 0.0;
04315      qxz = 0.0;
04316      qyx = 0.0;
04317      qyy = 0.0;
04318      qyz = 0.0;
04319      qzx = 0.0;
04320      qzy = 0.0;
04321      qzz = 0.0;
04322  }
04323
04324  xr = gpos[0] - apos[0];
04325  yr = gpos[1] - apos[1];
04326  zr = gpos[2] - apos[2];
04327
04328  dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
04329  multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
04330
04331  val += prel*charge*tensor[0];
04332  val -= prel*ux*xr*tensor[1];
04333  val -= prel*uy*yr*tensor[1];
04334  val -= prel*uz*zr*tensor[1];
04335  val += prel*qxx*xr*xr*tensor[2];
04336  val += prel*qyy*yr*yr*tensor[2];
04337  val += prel*qzz*zr*zr*tensor[2];
04338  val += prel*2.0*qxy*xr*yr*tensor[2];
04339  val += prel*2.0*qxz*xr*zr*tensor[2];
04340  val += prel*2.0*qyz*yr*zr*tensor[2];
04341
04342  }
04343
04344  if(i==0){
04345    gxcf[IJKx(j,k,0)] = val;
04346  }
04347  if(i==nx-1){
04348    gxcf[IJKx(j,k,1)] = val;
04349  }
04350  if(j==0){
04351    gycf[IJKy(i,k,0)] = val;
04352  }
04353  if(j==ny-1){
04354    gycf[IJKy(i,k,1)] = val;
04355  }
04356  if(k==0){
04357    gzcf[IJKz(i,j,0)] = val;
04358  }
04359  if(k==nz-1){
04360    gzcf[IJKz(i,j,1)] = val;
04361  }
04362  /* End grid point is valid */
04363  /* End i loop */
04364  /* End j loop */
04365  /* End k loop */
04366
04367 }
04368 #endif
04369
04370 VPRIIVATE void bcCalc(Vpmg *thee){
04371
04372  int i, j, k;
04373  int nx, ny, nz;
04374
04375  double zmem, eps_m, Lmem, memv, eps_w, xkappa;
04376
04377  nx = thee->pmgp->nx;
04378  ny = thee->pmgp->ny;
04379  nz = thee->pmgp->nz;
04380
04381  /* Zero out the boundaries */

```

```

04382     /* the "i" boundaries (dirichlet) */
04383     for (k=0; k<nz; k++) {
04384         for (j=0; j<ny; j++) {
04385             thee->gxcf[IJKx(j,k,0)] = 0.0;
04386             thee->gxcf[IJKx(j,k,1)] = 0.0;
04387             thee->gxcf[IJKx(j,k,2)] = 0.0;
04388             thee->gxcf[IJKx(j,k,3)] = 0.0;
04389         }
04390     }
04391
04392     /* the "j" boundaries (dirichlet) */
04393     for (k=0; k<nz; k++) {
04394         for (i=0; i<nx; i++) {
04395             thee->gycf[IJKy(i,k,0)] = 0.0;
04396             thee->gycf[IJKy(i,k,1)] = 0.0;
04397             thee->gycf[IJKy(i,k,2)] = 0.0;
04398             thee->gycf[IJKy(i,k,3)] = 0.0;
04399         }
04400     }
04401
04402     /* the "k" boundaries (dirichlet) */
04403     for (j=0; j<ny; j++) {
04404         for (i=0; i<nx; i++) {
04405             thee->gzcf[IJKz(i,j,0)] = 0.0;
04406             thee->gzcf[IJKz(i,j,1)] = 0.0;
04407             thee->gzcf[IJKz(i,j,2)] = 0.0;
04408             thee->gzcf[IJKz(i,j,3)] = 0.0;
04409         }
04410     }
04411
04412     switch (thee->pmgp->bcfl) {
04413         /* If we have zero boundary conditions, we're done */
04414         case BCFL_ZERO:
04415             return;
04416         case BCFL_SDH:
04417             bcfl_sdh(thee);
04418             break;
04419         case BCFL_MDH:
04420 #if defined(WITH_TINKER)
04421             bcfl_mdh_tinker(thee);
04422 #else
04423
04424 #ifdef DEBUG_MAC OSX_OCL
04425 #include "mach_chud.h"
04426     uint64_t mbeg = mach_absolute_time();
04427
04428     /*
04429      * If OpenCL is available we use it, otherwise fall back to
04430      * normal route (CPU multithreaded w/ OpenMP)
04431      */
04432     if (kOpenCLAvailable == 1) bcflnewOpenCL(thee);
04433     else bcflnew(thee);
04434
04435     mets_(&mbeg, "MDH");
04436 #else
04437     /* bcfl_mdh(thee); */
04438     bcflnew(thee);
04439 #endif /* DEBUG_MAC OSX_OCL */
04440
04441 #endif /* WITH_TINKER */
04442     break;
04443     case BCFL_MEM:
04444
04445     zmem = Vpbe_getzmem(thee->pbe);
04446     Lmem = Vpbe_getLmem(thee->pbe);
04447     eps_m = Vpbe_getmembraneDiel(thee->pbe);
04448     memv = Vpbe_getmemv(thee->pbe);
04449
04450     eps_w = Vpbe_getSolventDiel(thee->pbe);
04451     xkappa = Vpbe_getXkappa(thee->pbe);
04452
04453     bcfl_mem(zmem, Lmem, eps_m, eps_w, memv, xkappa,
04454         thee->gxcf, thee->gycf, thee->gzcf,
04455         thee->xf, thee->yf, thee->zf, nx, ny, nz);
04456     break;
04457     case BCFL_UNUSED:
04458         Vnm_print(2, "bcCalc: Invalid bcfl (%d)!\n", thee->pmgp->bcfl
04459     );
04460         VASSERT(0);
04461         break;
04462         case BCFL_FOCUS:

```

```

04462             Vnm_print(2, "VPMG::bcCalc -- not appropriate for focusing!\n");
04463             VASSERT(0);
04464         break;
04465     case BCFL_MAP:
04466         bcfl_map(thee);
04467         focusFillBound(thee,VNULL);
04468         break;
04469     default:
04470         Vnm_print(2, "VPMG::bcCalc -- invalid boundary condition \
04471         flag (%d)!\n", thee->pmgp->bcfl);
04472         VASSERT(0);
04473     break;
04474 }
04475 }
04476
04477 VPRIvate void fillcoCoefMap(Vpmg *thee) {
04478
04479     Vpbe *pbe;
04480     double ionstr, position[3], tkappa, eps, pot, hx, hy, hzed;
04481     int i, j, k, nx, ny, nz;
04482     double kappamax;
04483     VASSERT(thee != VNULL);
04484
04485     /* Get PBE info */
04486     pbe = thee->pbe;
04487     ionstr = Vpbe_getBulkIonicStrength(pbe);
04488
04489     /* Mesh info */
04490     nx = thee->pmgp->nx;
04491     ny = thee->pmgp->ny;
04492     nz = thee->pmgp->nz;
04493     hx = thee->pmgp->hx;
04494     hy = thee->pmgp->hy;
04495     hzed = thee->pmgp->hzed;
04496
04497     if ((!thee->useDielXMap) || (!thee->useDielYMap)
04498     || (!thee->useDielZMap) || ((!thee->useKappaMap) &&
04499     ionstr>VPMGSMALL))) {
04500         Vnm_print(2, "fillcoCoefMap: You need to use all coefficient maps!\n")
04501 ;
04502     VASSERT(0);
04503 }
04504
04505     /* Scale the kappa map to values between 0 and 1
04506     Thus get the maximum value in the map - this
04507     is theoretically unnecessary, but a good check.*/
04508     kappamax = -1.00;
04509     for (k=0; k<nz; k++) {
04510         for (j=0; j<ny; j++) {
04511             for (i=0; i<nx; i++) {
04512                 if (ionstr > VPMGSMALL) {
04513                     position[0] = thee->xf[i];
04514                     position[1] = thee->yf[j];
04515                     position[2] = thee->zf[k];
04516                     if (!Vgrid_value(thee->kappaMap,
04517                         position, &tkappa)) {
04518                         Vnm_print(2, "Vpmg_fillco: Off kappaMap at:\n");
04519                         Vnm_print(2, "Vpmg_fillco: (x,y,z) = (%g,%g %g)\n",
04520                             position[0], position[1], position[2]);
04521                         VASSERT(0);
04522                     }
04523                     if (tkappa > kappamax) {
04524                         kappamax = tkappa;
04525                     }
04526                     if (tkappa < 0.0){
04527                         Vnm_print(2, "Vpmg_fillcoCoefMap: Kappa map less than 0
04528                         \n");
04529                         Vnm_print(2, "Vpmg_fillcoCoefMap: at (x,y,z) = (%g,%g
04530                         %g)\n",
04531                             position[0], position[1], position[2]);
04532                         VASSERT(0);
04533                     }
04534                 }
04535             }
04536         if (kappamax > 1.0){
04537             Vnm_print(2, "Vpmg_fillcoCoefMap: Maximum Kappa value\n");

```

```

04538     Vnm_print(2, "%g is greater than 1 - will scale appropriately!\n",
04539                 kappaMax);
04540 }
04541 else {
04542     kappaMax = 1.0;
04543 }
04544
04545 for (k=0; k<nz; k++) {
04546     for (j=0; j<ny; j++) {
04547         for (i=0; i<nx; i++) {
04548
04549             if (ionstr > VPMGSIMAL) {
04550                 position[0] = thee->xf[i];
04551                 position[1] = thee->yf[j];
04552                 position[2] = thee->zf[k];
04553                 if (!Vgrid_value(thee->kappaMap,
04554                     position, &tkappa)) {
04555                     Vnm_print(2, "Vpmg_fillco: Off kappaMap at:\n");
04556                     Vnm_print(2, "Vpmg_fillco: (x,y,z) = (%g,%g %g)\n",
04557                             position[0], position[1], position[2]);
04558                     VASSERT(0);
04559                 }
04560                 if (tkappa < VPMGSIMAL) tkappa = 0.0;
04561                 thee->kappa[IJK(i,j,k)] = (tkappa / kappaMax);
04562             }
04563
04564             position[0] = thee->xf[i] + 0.5*hx;
04565             position[1] = thee->yf[j];
04566             position[2] = thee->zf[k];
04567             if (!Vgrid_value(thee->dielXMap, position, &
04568                 eps)) {
04569                 Vnm_print(2, "Vpmg_fillco: Off dielXMap at:\n");
04570                 Vnm_print(2, "Vpmg_fillco: (x,y,z) = (%g,%g %g)\n",
04571                         position[0], position[1], position[2]);
04572                 VASSERT(0);
04573             }
04574             thee->epsx[IJK(i,j,k)] = eps;
04575
04576             position[0] = thee->xf[i];
04577             position[1] = thee->yf[j] + 0.5*hy;
04578             position[2] = thee->zf[k];
04579             if (!Vgrid_value(thee->dielYMap, position,
04580                 &eps)) {
04581                 Vnm_print(2, "Vpmg_fillco: Off dielYMap at:\n");
04582                 Vnm_print(2, "Vpmg_fillco: (x,y,z) = (%g,%g %g)\n",
04583                         position[0], position[1], position[2]);
04584                 VASSERT(0);
04585             }
04586             thee->epsy[IJK(i,j,k)] = eps;
04587
04588             position[0] = thee->xf[i];
04589             position[1] = thee->yf[j];
04590             position[2] = thee->zf[k] + 0.5*hzed;
04591             if (!Vgrid_value(thee->dielZMap, position,
04592                 &eps)) {
04593                 Vnm_print(2, "Vpmg_fillco: Off dielZMap at:\n");
04594                 Vnm_print(2, "Vpmg_fillco: (x,y,z) = (%g,%g %g)\n",
04595                         position[0], position[1], position[2]);
04596                 VASSERT(0);
04597             }
04598         }
04599     }
04600 VPRIVATE void fillcoCoefMol(Vpmg *thee) {
04601
04602     if (thee->useDielXMap || thee->useDielYMap || thee->
04603         useDielZMap ||
04604         thee->useKappaMap) {
04605
04606         fillcoCoefMap(thee);
04607     } else {
04608
04609         fillcoCoefMolDiel(thee);
04610         fillcoCoefMolIon(thee);
04611
04612     }
04613

```

```

04614 }
04615
04616 VPRIvATE void fillcoCoefMolIon(Vpmg *thee) {
04617
04618     Vacc *acc;
04619     Valist *alist;
04620     Vpbe *pbe;
04621     Vatom *atom;
04622     double xmin, xmax, ymin, ymax, zmin, zmax, ionmask,
04623     ionstr;
04624     double xlabel, ylabel, zlabel, irad;
04625     double hx, hy, hzed, *apos, arad;
04626     int i, nx, ny, nz, iatom;
04627     Vsurf_Meth surfMeth;
04628
04629     VASSERT(thee != VNULL);
04630     surfMeth = thee->surfMeth;
04631
04632     /* Get PBE info */
04633     pbe = thee->pbe;
04634     acc = pbe->acc;
04635     alist = pbe->alist;
04636     irad = Vpbe_getMaxIonRadius(pbe);
04637     ionstr = Vpbe_getBulkIonicStrength(pbe);
04638
04639     /* Mesh info */
04640     nx = thee->pmgp->nx;
04641     ny = thee->pmgp->ny;
04642     nz = thee->pmgp->nz;
04643     hx = thee->pmgp->hx;
04644     hy = thee->pmgp->hy;
04645     hzed = thee->pmgp->hzed;
04646
04647     /* Define the total domain size */
04648     xlabel = thee->pmgp->xlabel;
04649     ylabel = thee->pmgp->ylabel;
04650     zlabel = thee->pmgp->zlabel;
04651
04652     /* Define the min/max dimensions */
04653     xmin = thee->pmgp->xcent - (xlabel/2.0);
04654     ymin = thee->pmgp->ycent - (ylabel/2.0);
04655     zmin = thee->pmgp->zcent - (zlabel/2.0);
04656     xmax = thee->pmgp->xcent + (xlabel/2.0);
04657     ymax = thee->pmgp->ycent + (ylabel/2.0);
04658     zmax = thee->pmgp->zcent + (zlabel/2.0);
04659
04660     /* This is a floating point parameter related to the non-zero nature of the
04661     * bulk ionic strength. If the ionic strength is greater than zero; this
04662     * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
04663     * Otherwise, this parameter is set to 0.0 */
04664     if (ionstr > VPMGSMALL) ionmask = 1.0;
04665     else ionmask = 0.0;
04666
04667     /* Reset the kappa array, marking everything accessible */
04668     for (i=0; i<(nx*ny*nz); i++) thee->kappa[i] = ionmask;
04669
04670     if (ionstr < VPMGSMALL) return;
04671
04672     /* Loop through the atoms and set kappa = 0.0 (inaccessible) if a point
04673     * is inside the ion-inflated van der Waals radii */
04674     for (iatom=0; iatom<Valist_getNumberAtoms(alist);
04675         iatom++) {
04676
04677         atom = Valist_getAtom(alist, iatom);
04678         apos = Vatom_getPosition(atom);
04679         arad = Vatom_getRadius(atom);
04680
04681         if (arad > VSMALL) {
04682             /* Make sure we're on the grid */
04683             if ((apos[0]<(xmin-irad-arad)) || (apos[0]>(xmax+irad+arad)) || \
04684                 (apos[1]<(ymin-irad-arad)) || (apos[1]>(ymax+irad+arad)) || \
04685                 (apos[2]<(zmin-irad-arad)) || (apos[2]>(zmax+irad+arad))) {
04686                 if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
04687                     (thee->pmgp->bcfl != BCFL_MAP)) {
04688                     Vnm_print(2,
04689                     "Vpmg_fillco: Atom %d at (%4.3f, %4.3f, %4.3f) is off the mesh
04690                     (ignoring):\n",
04691                     iatom, apos[0], apos[1], apos[2]);
04692                     Vnm_print(2, "Vpmg_fillco: xmin = %g, xmax = %g\n",
04693                     xmin, xmax);
04694
04695
04696
04697
04698
04699
04700
04701
04702
04703
04704
04705
04706
04707
04708
04709
04710
04711
04712
04713
04714
04715
04716
04717
04718
04719
04720
04721
04722
04723
04724
04725
04726
04727
04728
04729
04730
04731
04732
04733
04734
04735
04736
04737
04738
04739
04740
04741
04742
04743
04744
04745
04746
04747
04748
04749
04750
04751
04752
04753
04754
04755
04756
04757
04758
04759
04760
04761
04762
04763
04764
04765
04766
04767
04768
04769
04770
04771
04772
04773
04774
04775
04776
04777
04778
04779
04780
04781
04782
04783
04784
04785
04786
04787
04788
04789
04790
04791
04792
04793
04794
04795
04796
04797
04798
04799
04800
04801
04802
04803
04804
04805
04806
04807
04808
04809
04810
04811
04812
04813
04814
04815
04816
04817
04818
04819
04820
04821
04822
04823
04824
04825
04826
04827
04828
04829
04830
04831
04832
04833
04834
04835
04836
04837
04838
04839
04840
04841
04842
04843
04844
04845
04846
04847
04848
04849
04850
04851
04852
04853
04854
04855
04856
04857
04858
04859
04860
04861
04862
04863
04864
04865
04866
04867
04868
04869
04870
04871
04872
04873
04874
04875
04876
04877
04878
04879
04880
04881
04882
04883
04884
04885
04886
04887
04888
04889
04890
04891
04892
04893
04894
04895
04896
04897
04898
04899
04900
04901
04902
04903
04904
04905
04906
04907
04908
04909
04910
04911
04912
04913
04914
04915
04916
04917
04918
04919
04920
04921
04922
04923
04924
04925
04926
04927
04928
04929
04930
04931
04932
04933
04934
04935
04936
04937
04938
04939
04940
04941
04942
04943
04944
04945
04946
04947
04948
04949
04950
04951
04952
04953
04954
04955
04956
04957
04958
04959
04960
04961
04962
04963
04964
04965
04966
04967
04968
04969
04970
04971
04972
04973
04974
04975
04976
04977
04978
04979
04980
04981
04982
04983
04984
04985
04986
04987
04988
04989
04990
04991
04992
04993
04994
04995
04996
04997
04998
04999
05000
05001
05002
05003
05004
05005
05006
05007
05008
05009
05010
05011
05012
05013
05014
05015
05016
05017
05018
05019
05020
05021
05022
05023
05024
05025
05026
05027
05028
05029
05030
05031
05032
05033
05034
05035
05036
05037
05038
05039
05040
05041
05042
05043
05044
05045
05046
05047
05048
05049
05050
05051
05052
05053
05054
05055
05056
05057
05058
05059
05060
05061
05062
05063
05064
05065
05066
05067
05068
05069
05070
05071
05072
05073
05074
05075
05076
05077
05078
05079
05080
05081
05082
05083
05084
05085
05086
05087
05088
05089
05090
05091
05092
05093
05094
05095
05096
05097
05098
05099
05100
05101
05102
05103
05104
05105
05106
05107
05108
05109
05110
05111
05112
05113
05114
05115
05116
05117
05118
05119
05120
05121
05122
05123
05124
05125
05126
05127
05128
05129
05130
05131
05132
05133
05134
05135
05136
05137
05138
05139
05140
05141
05142
05143
05144
05145
05146
05147
05148
05149
05150
05151
05152
05153
05154
05155
05156
05157
05158
05159
05160
05161
05162
05163
05164
05165
05166
05167
05168
05169
05170
05171
05172
05173
05174
05175
05176
05177
05178
05179
05180
05181
05182
05183
05184
05185
05186
05187
05188
05189
05190
05191
05192
05193
05194
05195
05196
05197
05198
05199
05200
05201
05202
05203
05204
05205
05206
05207
05208
05209
05210
05211
05212
05213
05214
05215
05216
05217
05218
05219
05220
05221
05222
05223
05224
05225
05226
05227
05228
05229
05230
05231
05232
05233
05234
05235
05236
05237
05238
05239
05240
05241
05242
05243
05244
05245
05246
05247
05248
05249
05250
05251
05252
05253
05254
05255
05256
05257
05258
05259
05260
05261
05262
05263
05264
05265
05266
05267
05268
05269
05270
05271
05272
05273
05274
05275
05276
05277
05278
05279
05280
05281
05282
05283
05284
05285
05286
05287
05288
05289
05290
05291
05292
05293
05294
05295
05296
05297
05298
05299
05300
05301
05302
05303
05304
05305
05306
05307
05308
05309
05310
05311
05312
05313
05314
05315
05316
05317
05318
05319
05320
05321
05322
05323
05324
05325
05326
05327
05328
05329
05330
05331
05332
05333
05334
05335
05336
05337
05338
05339
05340
05341
05342
05343
05344
05345
05346
05347
05348
05349
05350
05351
05352
05353
05354
05355
05356
05357
05358
05359
05360
05361
05362
05363
05364
05365
05366
05367
05368
05369
05370
05371
05372
05373
05374
05375
05376
05377
05378
05379
05380
05381
05382
05383
05384
05385
05386
05387
05388
05389
05390
05391
05392
05393
05394
05395
05396
05397
05398
05399
05400
05401
05402
05403
05404
05405
05406
05407
05408
05409
05410
05411
05412
05413
05414
05415
05416
05417
05418
05419
05420
05421
05422
05423
05424
05425
05426
05427
05428
05429
05430
05431
05432
05433
05434
05435
05436
05437
05438
05439
05440
05441
05442
05443
05444
05445
05446
05447
05448
05449
05450
05451
05452
05453
05454
05455
05456
05457
05458
05459
05460
05461
05462
05463
05464
05465
05466
05467
05468
05469
05470
05471
05472
05473
05474
05475
05476
05477
05478
05479
05480
05481
05482
05483
05484
05485
05486
05487
05488
05489
05490
05491
05492
05493
05494
05495
05496
05497
05498
05499
05500
05501
05502
05503
05504
05505
05506
05507
05508
05509
05510
05511
05512
05513
05514
05515
05516
05517
05518
05519
05520
05521
05522
05523
05524
05525
05526
05527
05528
05529
05530
05531
05532
05533
05534
05535
05536
05537
05538
05539
05540
05541
05542
05543
05544
05545
05546
05547
05548
05549
05550
05551
05552
05553
05554
05555
05556
05557
05558
05559
05560
05561
05562
05563
05564
05565
05566
05567
05568
05569
05570
05571
05572
05573
05574
05575
05576
05577
05578
05579
05580
05581
05582
05583
05584
05585
05586
05587
05588
05589
05590
05591
05592
05593
05594
05595
05596
05597
05598
05599
05600
05601
05602
05603
05604
05605
05606
05607
05608
05609
05610
05611
05612
05613
05614
05615
05616
05617
05618
05619
05620
05621
05622
05623
05624
05625
05626
05627
05628
05629
05630
05631
05632
05633
05634
05635
05636
05637
05638
05639
05640
05641
05642
05643
05644
05645
05646
05647
05648
05649
05650
05651
05652
05653
05654
05655
05656
05657
05658
05659
05660
05661
05662
05663
05664
05665
05666
05667
05668
05669
05670
05671
05672
05673
05674
05675
05676
05677
05678
05679
05680
05681
05682
05683
05684
05685
05686
05687
05688
05689
05690
05691
05692
05693
05694
05695
05696
05697
05698
05699
05700
05701
05702
05703
05704
05705
05706
05707
05708
05709
05710
05711
05712
05713
05714
05715
05716
05717
05718
05719
05720
05721
05722
05723
05724
05725
05726
05727
05728
05729
05730
05731
05732
05733
05734
05735
05736
05737
05738
05739
05740
05741
05742
05743
05744
05745
05746
05747
05748
05749
05750
05751
05752
05753
05754
05755
05756
05757
05758
05759
05760
05761
05762
05763
05764
05765
05766
05767
05768
05769
05770
05771
05772
05773
05774
05775
05776
05777
05778
05779
05780
05781
05782
05783
05784
05785
05786
05787
05788
05789
05790
05791
05792
05793
05794
05795
05796
05797
05798
05799
05800
05801
05802
05803
05804
05805
05806
05807
05808
05809
05810
05811
05812
05813
05814
05815
05816
05817
05818
05819
05820
05821
05822
05823
05824
05825
05826
05827
05828
05829
05830
05831
05832
05833
05834
05835
05836
05837
05838
05839
05840
05841
05842
05843
05844
05845
05846
05847
05848
05849
05850
05851
05852
05853
05854
05855
05856
05857
05858
05859
05860
05861
05862
05863
05864
05865
05866
05867
05868
05869
05870
05871
05872
05873
05874
05875
05876
05877
05878
05879
05880
05881
05882
05883
05884
05885
05886
05887
05888
05889
05890
05891
05892
05893
05894
05895
05896
05897
05898
05899
05900
05901
05902
05903
05904
05905
05906
05907
05908
05909
05910
05911
05912
05913
05914
05915
05916
05917
05918
05919
05920
05921
05922
05923
05924
05925
05926
05927
05928
05929
05930
05931
05932
05933
05934
05935
05936
05937
05938
05939
05940
05941
05942
05943
05944
05945
05946
05947
05948
05949
05950
05951
05952
05953
05954
05955
05956
05957
05958
05959
05960
05961
05962
05963
05964
05965
05966
05967
05968
05969
05970
05971
05972
05973
05974
05975
05976
05977
05978
05979
05980
05981
05982
05983
05984
05985
05986
05987
05988
05989
05990
05991
05992
05993
05994
05995
05996
05997
05998
05999
06000
06001
06002
06003
06004
06005
06006
06007
06008
06009
06010
06011
06012
06013
06014
06015
06016
06017
06018
06019
06020
06021
06022
06023
06024
06025
06026
06027
06028
06029
06030
06031
06032
06033
06034
06035
06036
06037
06038
06039
06040
06041
06042
06043
06044
06045
06046
06047
06048
06049
06050
06051
06052
06053
06054
06055
06056
06057
06058
06059
06060
06061
06062
06063
06064
06065
06066
06067
06068
06069
06070
06071
06072
06073
06074
06075
06076
06077
06078
06079
06080
06081
06082
06083
06084
06085
06086
06087
06088
06089
06090
06091
06092
06093
06094
06095
06096
06097
06098
06099
06100
06101
06102
06103
06104
06105
06106
06107
06108
06109
06110
06111
06112
06113
06114
06115
06116
06117
06118
06119
06120
06121
06122
06123
06124
06125
06126
06127
06128
06129
06130
06131
06132
06133
06134
06135
06136
06137
06138
06139
06140
06141
06142
06143
06144
06145
06146
06147
06148
06149
06150
06151
06152
06153
06154
06155
06156
06157
06158
06159
06160
06161
06162
06163
06164
06165
06166
06167
06168
06169
06170
06171
06172
06173
06174
06175
06176
06177
06178
06179
06180
06181
06182
06183
06184
06185
06186
06187
06188
06189
06190
06191
06192
06193
06194
06195
06196
06197
06198
06199
06200
06201
06202
06203
06204
06205
06206
06207
06208
06209
06210
06211
06212
06213
06214
06215
06216
06217
06218
06219
06220
06221
06222
06223
06224
06225
06226
06227
06228
06229
06230
06231
06232
06233
06234
06235
06236
06237
06238
06239
06240
06241
06242
06243
06244
06245
06246
06247
06248
06249
06250
06251
06252
06253
06254
06255
06256
06257
06258
06259
06260
06261
06262
06263
06264
06265
06266
06267
06268
06269
06270
06271
06272
06273
06274
06275
06276
06277
06278
06279
06280
06281
06282
06283
06284
06285
06286
06287
06288
06289
06290
06291
06292
06293
06294
06295
06296
06297
06298
06299
06300
06301
06302
06303
06304
06305
06306
06307
06308
06309
06310
06311
06312
06313
06314
06315
06316
06317
06318
06319
06320
06321
06322
06323
06324
06325
06326
06327
06328
06329
06330
06331
06332
06333
06334
06335
06336
06337
06338
06339
```

```

04692             Vnm_print(2, "Vpmg_fillco: ymin = %g, ymax = %g\n",
04693                         ymin, ymax);
04694             Vnm_print(2, "Vpmg_fillco: zmin = %g, zmax = %g\n",
04695                         zmin, zmax);
04696         }
04697         fflush(stderr);
04698
04699     } /* else { /* if we're on the mesh */
04700
04701         /* Mark ions */
04702         markSphere((irad+arad), apos,
04703                     nx, ny, nz,
04704                     hx, hy, hzed,
04705                     xmin, ymin, zmin,
04706                     thee->kappa, 0.0);
04707
04708     } /* endif (on the mesh) */
04709 }
04710 } /* endfor (over all atoms) */
04711
04712 }
04713
04714 VPRIIVATE void fillcoCoefMolDiel(Vpmg *thee) {
04715
04716     /* Always call NoSmooth to fill the epsilon arrays */
04717     fillcoCoefMolDielNoSmooth(thee);
04718
04719     /* Call the smoothing algorithm as needed */
04720     if (thee->surfMeth == VSM_MOLSMOOTH) {
04721         fillcoCoefMolDielSmooth(thee);
04722     }
04723 }
04724
04725 VPRIIVATE void fillcoCoefMolDielNoSmooth(Vpmg *thee
04726     ) {
04727
04728     Vacc *acc;
04729     VaccSurf *asurf;
04730     Valist *alist;
04731     Vpbe *pbe;
04732     Vatom *atom;
04733     double xmin, xmax, ymin, ymax, zmin, zmax;
04734     double xlabel, ylabel, zlabel, position[3];
04735     double srad, epsw, epsp, deps, area;
04736     double hx, hy, hzed, *apos, arad;
04737     int i, nx, ny, nz, ntot, iatom, ipt;
04738
04739     /* Get PBE info */
04740     pbe = thee->pbe;
04741     acc = pbe->acc;
04742     alist = pbe->alist;
04743     srad = Vpbe_getSolventRadius(pbe);
04744     epsw = Vpbe_getSolventDiel(pbe);
04745     epsp = Vpbe_getSoluteDiel(pbe);
04746
04747     /* Mesh info */
04748     nx = thee->pmgp->nx;
04749     ny = thee->pmgp->ny;
04750     nz = thee->pmgp->nz;
04751     hx = thee->pmgp->hx;
04752     hy = thee->pmgp->hy;
04753     hzed = thee->pmgp->hzed;
04754
04755     /* Define the total domain size */
04756     xlabel = thee->pmgp->xlen;
04757     ylabel = thee->pmgp->ylen;
04758     zlabel = thee->pmgp->zlen;
04759
04760     /* Define the min/max dimensions */
04761     xmin = thee->pmgp->xcent - (xlabel/2.0);
04762     ymin = thee->pmgp->ycent - (ylabel/2.0);
04763     zmin = thee->pmgp->zcent - (zlabel/2.0);
04764     xmax = thee->pmgp->xcent + (xlabel/2.0);
04765     ymax = thee->pmgp->ycent + (ylabel/2.0);
04766     zmax = thee->pmgp->zcent + (zlabel/2.0);
04767
04768     /* Reset the arrays */
04769     ntot = nx*ny*nz;
04770     for (i=0; i<ntot; i++) {
04771         thee->epsx[i] = epsw;
04772         thee->epsy[i] = epsw;

```

```

04772     thee->epsz[i] = epsw;
04773 }
04774
04775 /* Loop through the atoms and set a{123}cf = 0.0 (inaccessible)
04776 * if a point is inside the solvent-inflated van der Waals radii */
04777 #pragma omp parallel for default(shared) private(iatom,atom,apos,arad)
04778     for (iatom=0; iatom<Valist_getNumberAtoms(alist);
04779         iatom++) {
04780         atom = Valist_getAtom(alist, iatom);
04781         apos = Vatom_getPosition(atom);
04782         arad = Vatom_getRadius(atom);
04783
04784         /* Make sure we're on the grid */
04785         if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
04786             (apos[1]<=ymin) || (apos[1]>=ymax) || \
04787             (apos[2]<=zmin) || (apos[2]>=zmax)) {
04788             if ((thee->pmpg->bpcf != BCFL_FOCUS) &&
04789                 (thee->pmpg->bpcf != BCFL_MAP)) {
04790                 Vnm_print(2, "Vpmg_fillco: Atom #%d at (%4.3f, %4.3f,\n",
04791 %4.3f) is off the mesh (ignoring):\n",
04792                 iatom, apos[0], apos[1], apos[2]);
04793                 Vnm_print(2, "Vpmg_fillco: xmin = %g, xmax = %g\n",
04794                 xmin, xmax);
04795                 Vnm_print(2, "Vpmg_fillco: ymin = %g, ymax = %g\n",
04796                 ymin, ymax);
04797                 Vnm_print(2, "Vpmg_fillco: zmin = %g, zmax = %g\n",
04798                 zmin, zmax);
04799             }
04800             fflush(stderr);
04801
04802         } else { /* if we're on the mesh */
04803
04804             if (arad > VSMALL) {
04805                 /* Mark x-shifted dielectric */
04806                 markSphere((arad+srad), apos,
04807                             nx, ny, nz,
04808                             hx, hy, hzed,
04809                             (xmin+0.5*hx), ymin, zmin,
04810                             thee->epsx, epsp);
04811
04812                 /* Mark y-shifted dielectric */
04813                 markSphere((arad+srad), apos,
04814                             nx, ny, nz,
04815                             hx, hy, hzed,
04816                             xmin, (ymin+0.5*hy), zmin,
04817                             thee->epsy, epsp);
04818
04819                 /* Mark z-shifted dielectric */
04820                 markSphere((arad+srad), apos,
04821                             nx, ny, nz,
04822                             hx, hy, hzed,
04823                             xmin, ymin, (zmin+0.5*hzed),
04824                             thee->epsz, epsp);
04825             }
04826
04827         } /* endif (on the mesh) */
04828     } /* endfor (over all atoms) */
04829
04830     area = Vacc_SASA(acc, srad);
04831
04832     /* We only need to do the next step for non-zero solvent radii */
04833     if (srad > VSMALL) {
04834
04835         /* Now loop over the solvent accessible surface points */
04836
04837 #pragma omp parallel for default(shared)
04838     private(iatom,atom,area,asurf,ipt,position)
04839     for (iatom=0; iatom<Valist_getNumberAtoms(alist);
04840         iatom++) {
04841         atom = Valist_getAtom(alist, iatom);
04842         area = Vacc_atomSASA(acc, srad, atom);
04843         if (area > 0.0 ) {
04844             asurf = Vacc_atomSASPoints(acc, srad, atom);
04845
04846             /* Use each point on the SAS to reset the solvent accessibility */
04847             /* TODO: Make sure we're not still wasting time here. */
04848             for (ipt=0; ipt<(asurf->npts); ipt++) {
04849
04850                 position[0] = asurf->xpts[ipt];
04851                 position[1] = asurf->ypts[ipt];

```

```

04850     position[2] = asurf->zpts[ipt];
04851
04852     /* Mark x-shifted dielectric */
04853     markSphere(srads, position,
04854         nx, ny, nz,
04855         hx, hy, hzed,
04856         (xmin+0.5*hx), ymin, zmin,
04857         thee->epsx, epsw);
04858
04859     /* Mark y-shifted dielectric */
04860     markSphere(srads, position,
04861         nx, ny, nz,
04862         hx, hy, hzed,
04863         xmin, (ymin+0.5*hy), zmin,
04864         thee->epsy, epsw);
04865
04866     /* Mark z-shifted dielectric */
04867     markSphere(srads, position,
04868         nx, ny, nz,
04869         hx, hy, hzed,
04870         xmin, ymin, (zmin+0.5*hzed),
04871         thee->epsz, epsw);
04872
04873 }
04874 }
04875 }
04876 }
04877 }

04878 VPRIVATE void fillcoCoefMolDielsSmooth(Vpmg *thee) {
04879
04880     /* This function smoothes using a 9 point method based on
04881     Brucolieri, et al. J Comput Chem 18 268-276 (1997). The nine points
04882     used are the shifted grid point and the 8 points that are 1/sqrt(2)
04883     grid spacings away. The harmonic mean of the 9 points is then used to
04884     find the overall dielectric value for the point in question. The use of
04885     this function assumes that the non-smoothed values were placed in the
04886     dielectric arrays by the fillcoCoefMolDielNoSmooth function.*/
04887
04888     Vpbe *pbe;
04889     double frac, epsw;
04890     int i, j, k, nx, ny, nz, numpts;
04891
04892     /* Mesh info */
04893     nx = thee->pmgp->nx;
04894     ny = thee->pmgp->ny;
04895     nz = thee->pmgp->nz;
04896
04897     pbe = thee->pbe;
04898     epsw = Vpbe_getSolventDiel(pbe);
04899
04900     /* Copy the existing diel arrays to work arrays */
04901     for (i=0; i<(nx*ny*nz); i++) {
04902         thee->a1cf[i] = thee->epsx[i];
04903         thee->a2cf[i] = thee->epsy[i];
04904         thee->a3cf[i] = thee->epsz[i];
04905         thee->epsx[i] = epsw;
04906         thee->epsy[i] = epsw;
04907         thee->epsz[i] = epsw;
04908     }
04909
04910     /* Smooth the dielectric values */
04911     for (i=0; i<nx; i++) {
04912         for (j=0; j<ny; j++) {
04913             for (k=0; k<nz; k++) {
04914
04915                 /* Get the 8 points that are 1/sqrt(2) grid spacings away */
04916
04917                 /* Points for the X-shifted array */
04918                 frac = 1.0/thee->a1cf[IJK(i,j,k)];
04919                 frac += 1.0/thee->a2cf[IJK(i,j,k)];
04920                 frac += 1.0/thee->a3cf[IJK(i,j,k)];
04921                 numpts = 3;
04922
04923                 if (j > 0) {
04924                     frac += 1.0/thee->a2cf[IJK(i,j-1,k)];
04925                     numpts += 1;
04926                 }
04927                 if (k > 0) {
04928                     frac += 1.0/thee->a3cf[IJK(i,j,k-1)];
04929                     numpts += 1;
04930

```

```

04931 }
04932     if (i < (nx-1)) {
04933         frac += 1.0/thee->a2cf[IJK(i+1,j,k)];
04934         frac += 1.0/thee->a3cf[IJK(i+1,j,k)];
04935         numpts += 2;
04936         if (j > 0) {
04937             frac += 1.0/thee->a2cf[IJK(i+1,j-1,k)];
04938             numpts += 1;
04939         }
04940         if (k > 0) {
04941             frac += 1.0/thee->a3cf[IJK(i+1,j,k-1)];
04942             numpts += 1;
04943         }
04944     }
04945     thee->epsx[IJK(i,j,k)] = numpts/frac;
04946
04947 /* Points for the Y-shifted array */
04948 frac = 1.0/thee->a2cf[IJK(i,j,k)];
04949 frac += 1.0/thee->a1cf[IJK(i,j,k)];
04950 frac += 1.0/thee->a3cf[IJK(i,j,k)];
04951 numpts = 3;
04952
04953 if (i > 0) {
04954     frac += 1.0/thee->a1cf[IJK(i-1,j,k)];
04955     numpts += 1;
04956 }
04957 if (k > 0) {
04958     frac += 1.0/thee->a3cf[IJK(i,j,k-1)];
04959     numpts += 1;
04960 }
04961 if (j < (ny-1)){
04962     frac += 1.0/thee->a1cf[IJK(i,j+1,k)];
04963     frac += 1.0/thee->a3cf[IJK(i,j+1,k)];
04964     numpts += 2;
04965     if (i > 0) {
04966         frac += 1.0/thee->a1cf[IJK(i-1,j+1,k)];
04967         numpts += 1;
04968     }
04969     if (k > 0) {
04970         frac += 1.0/thee->a3cf[IJK(i,j+1,k-1)];
04971         numpts += 1;
04972     }
04973 }
04974 thee->epsy[IJK(i,j,k)] = numpts/frac;
04975
04976 /* Points for the Z-shifted array */
04977 frac = 1.0/thee->a3cf[IJK(i,j,k)];
04978 frac += 1.0/thee->a1cf[IJK(i,j,k)];
04979 frac += 1.0/thee->a2cf[IJK(i,j,k)];
04980 numpts = 3;
04981
04982 if (i > 0) {
04983     frac += 1.0/thee->a1cf[IJK(i-1,j,k)];
04984     numpts += 1;
04985 }
04986 if (j > 0) {
04987     frac += 1.0/thee->a2cf[IJK(i,j-1,k)];
04988     numpts += 1;
04989 }
04990 if (k < (nz-1)){
04991     frac += 1.0/thee->a1cf[IJK(i,j,k+1)];
04992     frac += 1.0/thee->a2cf[IJK(i,j,k+1)];
04993     numpts += 2;
04994     if (i > 0) {
04995         frac += 1.0/thee->a1cf[IJK(i-1,j,k+1)];
04996         numpts += 1;
04997     }
04998     if (j > 0) {
04999         frac += 1.0/thee->a2cf[IJK(i,j-1,k+1)];
05000         numpts += 1;
05001     }
05002 }
05003 thee->epsz[IJK(i,j,k)] = numpts/frac;
05004 }
05005 }
05006 }
05007 }
05008
05009
05010 VPRIVATE void fillcoCoefSpline(Vpmg *thee) {
05011

```



```

05090             xmin, xmax);
05091             Vnm_print(2, "Vpmg_fillco:    ymin = %g, ymax = %g\n",
05092                         ymin, ymax);
05093             Vnm_print(2, "Vpmg_fillco:    zmin = %g, zmax = %g\n",
05094                         zmin, zmax);
05095         }
05096         fflush(stderr);
05097     } else if (arad > VPMGSMALL) { /* if we're on the mesh */
05098
05099         /* Convert the atom position to grid reference frame */
05100         position[0] = apos[0] - xmin;
05101         position[1] = apos[1] - ymin;
05102         position[2] = apos[2] - zmin;
05103
05104         /* MARK ION ACCESSIBILITY AND DIELECTRIC VALUES FOR LATER
05105         * ASSIGNMENT (Steps #1-3) */
05106         itot = irad + arad + splineWin;
05107         itot2 = VSQR(itot);
05108         ictot = VMAX2(0, (irad + arad - splineWin));
05109         ictot2 = VSQR(ictot);
05110         stot = arad + splineWin;
05111         stot2 = VSQR(stot);
05112         sctot = VMAX2(0, (arad - splineWin));
05113         sctot2 = VSQR(sctot);
05114
05115         /* We'll search over grid points which are in the greater of
05116         * these two radii */
05117         rtot = VMAX2(itot, stot);
05118         rtot2 = VMAX2(itot2, stot2);
05119         dx = rtot + 0.5*hx;
05120         dy = rtot + 0.5*hy;
05121         dz = rtot + 0.5*hzed;
05122         imin = VMAX2(0, (int)floor((position[0] - dx)/hx));
05123         imax = VMIN2(nx-1, (int)ceil((position[0] + dx)/hx));
05124         jmin = VMAX2(0, (int)floor((position[1] - dy)/hy));
05125         jmax = VMIN2(ny-1, (int)ceil((position[1] + dy)/hy));
05126         kmin = VMAX2(0, (int)floor((position[2] - dz)/hzed));
05127         kmax = VMIN2(nz-1, (int)ceil((position[2] + dz)/hzed));
05128
05129         for (i=imin; i<=imax; i++) {
05130             dx2 = VSQR(position[0] - hx*i);
05131             for (j=jmin; j<=jmax; j++) {
05132                 dy2 = VSQR(position[1] - hy*j);
05133                 for (k=kmin; k<=kmax; k++) {
05134                     dz2 = VSQR(position[2] - k*hzed);
05135
05136                     /* ASSIGN CCF */
05137                     if (thee->kappa[IJK(i,j,k)] > VPMGSMALL)
05138                     {
05139                         dist2 = dz2 + dy2 + dx2;
05140                         if (dist2 >= itot2) {
05141                             ;
05142                         if (dist2 <= ictot2) {
05143                             thee->kappa[IJK(i,j,k)] = 0.0;
05144                         }
05145                         if ((dist2 < itot2) && (dist2 > ictot2)) {
05146                             dist = VSQRT(dist2);
05147                             sm = dist - (arad + irad) + splineWin;
05148                             sm2 = VSQR(sm);
05149                             value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
05150                             thee->kappa[IJK(i,j,k)] *= value;
05151                         }
05152                     }
05153
05154                     /* ASSIGN A1CF */
05155                     if (thee->epsx[IJK(i,j,k)] > VPMGSMALL) {
05156                         dist2 = dz2+dy2+VSQR(position[0]-(i+0.5)*hx);
05157                         if (dist2 >= stot2) {
05158                             thee->epsx[IJK(i,j,k)] *= 1.0;
05159                         }
05160                         if (dist2 <= sctot2) {
05161                             thee->epsx[IJK(i,j,k)] = 0.0;
05162                         }
05163                         if ((dist2 > sctot2) && (dist2 < stot2)) {
05164                             dist = VSQRT(dist2);
05165                             sm = dist - arad + splineWin;
05166                             sm2 = VSQR(sm);
05167                             value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
05168                             thee->epsx[IJK(i,j,k)] *= value;
05169                     }
05170
05171
05172
05173
05174
05175
05176
05177
05178
05179
05180
05181
05182
05183
05184
05185
05186
05187
05188
05189
05190
05191
05192
05193
05194
05195
05196
05197
05198
05199
05200
05201
05202
05203
05204
05205
05206
05207
05208
05209
05210
05211
05212
05213
05214
05215
05216
05217
05218
05219
05220
05221
05222
05223
05224
05225
05226
05227
05228
05229
05230
05231
05232
05233
05234
05235
05236
05237
05238
05239
05240
05241
05242
05243
05244
05245
05246
05247
05248
05249
05250
05251
05252
05253
05254
05255
05256
05257
05258
05259
05260
05261
05262
05263
05264
05265
05266
05267
05268
05269
05270
05271
05272
05273
05274
05275
05276
05277
05278
05279
05280
05281
05282
05283
05284
05285
05286
05287
05288
05289
05290
05291
05292
05293
05294
05295
05296
05297
05298
05299
05300
05301
05302
05303
05304
05305
05306
05307
05308
05309
05310
05311
05312
05313
05314
05315
05316
05317
05318
05319
05320
05321
05322
05323
05324
05325
05326
05327
05328
05329
05330
05331
05332
05333
05334
05335
05336
05337
05338
05339
05340
05341
05342
05343
05344
05345
05346
05347
05348
05349
05350
05351
05352
05353
05354
05355
05356
05357
05358
05359
05360
05361
05362
05363
05364
05365
05366
05367
05368
05369
05370
05371
05372
05373
05374
05375
05376
05377
05378
05379
05380
05381
05382
05383
05384
05385
05386
05387
05388
05389
05390
05391
05392
05393
05394
05395
05396
05397
05398
05399
05400
05401
05402
05403
05404
05405
05406
05407
05408
05409
05410
05411
05412
05413
05414
05415
05416
05417
05418
05419
05420
05421
05422
05423
05424
05425
05426
05427
05428
05429
05430
05431
05432
05433
05434
05435
05436
05437
05438
05439
05440
05441
05442
05443
05444
05445
05446
05447
05448
05449
05450
05451
05452
05453
05454
05455
05456
05457
05458
05459
05460
05461
05462
05463
05464
05465
05466
05467
05468
05469
05470
05471
05472
05473
05474
05475
05476
05477
05478
05479
05480
05481
05482
05483
05484
05485
05486
05487
05488
05489
05490
05491
05492
05493
05494
05495
05496
05497
05498
05499
05500
05501
05502
05503
05504
05505
05506
05507
05508
05509
05510
05511
05512
05513
05514
05515
05516
05517
05518
05519
05520
05521
05522
05523
05524
05525
05526
05527
05528
05529
05530
05531
05532
05533
05534
05535
05536
05537
05538
05539
05540
05541
05542
05543
05544
05545
05546
05547
05548
05549
05550
05551
05552
05553
05554
05555
05556
05557
05558
05559
05560
05561
05562
05563
05564
05565
05566
05567
05568
05569
05570
05571
05572
05573
05574
05575
05576
05577
05578
05579
05580
05581
05582
05583
05584
05585
05586
05587
05588
05589
05590
05591
05592
05593
05594
05595
05596
05597
05598
05599
05600
05601
05602
05603
05604
05605
05606
05607
05608
05609
05610
05611
05612
05613
05614
05615
05616
05617
05618
05619
05620
05621
05622
05623
05624
05625
05626
05627
05628
05629
05630
05631
05632
05633
05634
05635
05636
05637
05638
05639
05640
05641
05642
05643
05644
05645
05646
05647
05648
05649
05650
05651
05652
05653
05654
05655
05656
05657
05658
05659
05660
05661
05662
05663
05664
05665
05666
05667
05668
05669
05670
05671
05672
05673
05674
05675
05676
05677
05678
05679
05680
05681
05682
05683
05684
05685
05686
05687
05688
05689
05690
05691
05692
05693
05694
05695
05696
05697
05698
05699
05700
05701
05702
05703
05704
05705
05706
05707
05708
05709
05710
05711
05712
05713
05714
05715
05716
05717
05718
05719
05720
05721
05722
05723
05724
05725
05726
05727
05728
05729
05730
05731
05732
05733
05734
05735
05736
05737
05738
05739
05740
05741
05742
05743
05744
05745
05746
05747
05748
05749
05750
05751
05752
05753
05754
05755
05756
05757
05758
05759
05760
05761
05762
05763
05764
05765
05766
05767
05768
05769
05770
05771
05772
05773
05774
05775
05776
05777
05778
05779
05780
05781
05782
05783
05784
05785
05786
05787
05788
05789
05790
05791
05792
05793
05794
05795
05796
05797
05798
05799
05800
05801
05802
05803
05804
05805
05806
05807
05808
05809
05810
05811
05812
05813
05814
05815
05816
05817
05818
05819
05820
05821
05822
05823
05824
05825
05826
05827
05828
05829
05830
05831
05832
05833
05834
05835
05836
05837
05838
05839
05840
05841
05842
05843
05844
05845
05846
05847
05848
05849
05850
05851
05852
05853
05854
05855
05856
05857
05858
05859
05860
05861
05862
05863
05864
05865
05866
05867
05868
05869
05870
05871
05872
05873
05874
05875
05876
05877
05878
05879
05880
05881
05882
05883
05884
05885
05886
05887
05888
05889
05890
05891
05892
05893
05894
05895
05896
05897
05898
05899
05900
05901
05902
05903
05904
05905
05906
05907
05908
05909
05910
05911
05912
05913
05914
05915
05916
05917
05918
05919
05920
05921
05922
05923
05924
05925
05926
05927
05928
05929
05930
05931
05932
05933
05934
05935
05936
05937
05938
05939
05940
05941
05942
05943
05944
05945
05946
05947
05948
05949
05950
05951
05952
05953
05954
05955
05956
05957
05958
05959
05960
05961
05962
05963
05964
05965
05966
05967
05968
05969
05970
05971
05972
05973
05974
05975
05976
05977
05978
05979
05980
05981
05982
05983
05984
05985
05986
05987
05988
05989
05990
05991
05992
05993
05994
05995
05996
05997
05998
05999
05999
06000
06001
06002
06003
06004
06005
06006
06007
06008
06009
060010
060011
060012
060013
060014
060015
060016
060017
060018
060019
060020
060021
060022
060023
060024
060025
060026
060027
060028
060029
060030
060031
060032
060033
060034
060035
060036
060037
060038
060039
060040
060041
060042
060043
060044
060045
060046
060047
060048
060049
060050
060051
060052
060053
060054
060055
060056
060057
060058
060059
060060
060061
060062
060063
060064
060065
060066
060067
060068
060069
060070
060071
060072
060073
060074
060075
060076
060077
060078
060079
060080
060081
060082
060083
060084
060085
060086
060087
060088
060089
060090
060091
060092
060093
060094
060095
060096
060097
060098
060099
060099
060100
060101
060102
060103
060104
060105
060106
060107
060108
060109
060110
060111
060112
060113
060114
060115
060116
060117
060118
060119
060120
060121
060122
060123
060124
060125
060126
060127
060128
060129
060130
060131
060132
060133
060134
060135
060136
060137
060138
060139
060140
060141
060142
060143
060144
060145
060146
060147
060148
060149
060150
060151
060152
060153
060154
060155
060156
060157
060158
060159
060160
060161
060162
060163
060164
060165
060166
060167
060168
060169
060170
060171
060172
060173
060174
060175
060176
060177
060178
060179
060180
060181
060182
060183
060184
060185
060186
060187
060188
060189
060190
060191
060192
060193
060194
060195
060196
060197
060198
060199
060199
060200
060201
060202
060203
060204
060205
060206
060207
060208
060209
060210
060211
060212
060213
060214
060215
060216
060217
060218
060219
060219
060220
060221
060222
060223
060224
060225
060226
060227
060228
060229
060229
060230
060231
060232
060233
060234
060235
060236
060237
060238
060239
060239
060240
060241
060242
060243
060244
060245
060246
060247
060248
060249
060249
060250
060251
060252
060253
060254
060255
060256
060257
060258
060259
060259
060260
060261
060262
060263
060264
060265
060266
060267
060268
060269
060269
060270
060271
060272
060273
060274
060275
060276
060277
060278
060278
060279
060280
060281
060282
060283
060284
060285
060286
060287
060288
060289
060289
060290
060291
060292
060293
060294
060295
060296
060297
060297
060298
060299
060299
060300
060301
060302
060303
060304
060305
060306
060307
060308
060309
060309
060310
060311
060312
060313
060314
060315
060316
060317
060318
060319
060319
060320
060321
060322
060323
060324
060325
060326
060327
060328
060329
060329
060330
060331
060332
060333
060334
060335
060336
060337
060338
060339
060339
060340
060341
060342
060343
060344
060345
060346
060347
060348
060349
060349
060350
060351
060352
060353
060354
060355
060356
060357
060358
060359
060359
060360
060361
060362
060363
060364
060365
060366
060367
060368
060369
060369
060370
060371
060372
060373
060374
060375
060376
060377
060378
060379
060379
060380
060381
060382
060383
060384
060385
060386
060387
060388
060389
060389
060390
060391
060392
060393
060394
060395
060396
060397
060398
060399
060399
060400
060401
060402
060403
060404
060405
060406
060407
060408
060409
060409
060410
060411
060412
060413
060414
060415
060416
060417
060418
060419
060419
060420
060421
060422
060423
060424
060425
060426
060427
060428
060429
060429
060430
060431
060432
060433
060434
060435
060436
060437
060438
060439
060439
060440
060441
060442
060443
060444
060445
060446
060447
060448
060449
060449
060450
060451
060452
060453
060454
060455
060456
060457
060458
060459
060459
060460
060461
060462
060463
060464
060465
060466
060467
060468
060469
060469
060470
060471
060472
060473
060474
060475
060476
060477
060478
060479
060479
060480
060481
060482
060483
060484
060485
060486
060487
060488
060489
060489
060490
060491
060492
060493
060494
060495
060496
060497
060498
060499
060499
060500
060501
060502
060503
060504
060505
060506
060507
060508
060509
060509
060510
060511
060512
060513
060514
060515
060516
060517
060518
060519
060519
060520
060521
060522
060523
060524
060525
060526
060527
060528
060529
060529
060530
060531
060532
060533
060534
060535
060536
060537
060538
060539
060539
060540
060541
060542
060543
060544
060545
060546
060547
060548
060549
060549
060550
060551
060552
060553
060554
060555
060556
060557
060558
060559
060559
060560
060561
060562
060563
060564
060565
060566
060567
060568
060569
060569
060570
060571
060572
060573
060574
060575
060576
060577
060578
060579
060579
060580
060581
060582
060583
060584
060585
060586
060587
060588
060589
060589
060590
060591
060592
060593
060594
060595
060596
060597
060598
060599
060599
060600
060601
060602
060603
060604
060605
060606
060607
060608
060609
060609
060610
060611
060612
060613
060614
060615
060616
060617
060618
060619
060619
060620
060621
060622
060623
060624
060625
060626
060627
060628
060629
060629
060630
060631
060632
060633
060634
060635
060636
060637
060638
060639
060639
060640
060641
060642
060643
060644
060645
060646
```

```

05170
05171
05172     }
05173     /* ASSIGN A2CF */
05174     if (thee->epsy[IJK(i,j,k)] > VPMGSMALL) {
05175         dist2 = dz2+dx2+VSQR(position[1]-(j+0.5)*hy);
05176         if (dist2 >= stot2) {
05177             thee->epsy[IJK(i,j,k)] *= 1.0;
05178         }
05179         if (dist2 <= stot2) {
05180             thee->epsy[IJK(i,j,k)] = 0.0;
05181         }
05182         if ((dist2 > stot2) && (dist2 < stot2)) {
05183             dist = VSQRT(dist2);
05184             sm = dist - arad + splineWin;
05185             sm2 = VSQR(sm);
05186             value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
05187             thee->epsy[IJK(i,j,k)] *= value;
05188         }
05189     }
05190     /* ASSIGN A3CF */
05191     if (thee->epsz[IJK(i,j,k)] > VPMGSMALL) {
05192         dist2 = dy2+dx2+VSQR(position[2]-(k+0.5)*hzed);
05193         if (dist2 >= stot2) {
05194             thee->epsz[IJK(i,j,k)] *= 1.0;
05195         }
05196         if (dist2 <= stot2) {
05197             thee->epsz[IJK(i,j,k)] = 0.0;
05198         }
05199         if ((dist2 > stot2) && (dist2 < stot2)) {
05200             dist = VSQRT(dist2);
05201             sm = dist - arad + splineWin;
05202             sm2 = VSQR(sm);
05203             value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
05204             thee->epsz[IJK(i,j,k)] *= value;
05205         }
05206     }
05207
05208     } /* k loop */
05209     } /* j loop */
05210     } /* i loop */
05211 } /* endif (on the mesh) */
05212 } /* endfor (over all atoms) */
05213
05214 Vnm_print(0, "Vpmg_fillco: filling coefficient arrays\n");
05215 /* Interpret markings and fill the coefficient arrays */
05216 for (k=0; k<nz; k++) {
05217     for (j=0; j<ny; j++) {
05218         for (i=0; i<nx; i++) {
05219
05220             thee->kappa[IJK(i,j,k)] = ionmask*thee->kappa[IJK(i,j
05221 ,k)];
05222             thee->epsp[IJK(i,j,k)] = (epsw-epsp)*thee->epsp[IJK(i,j
05223 ,k)]
05224                 + epsp;
05225             thee->epsy[IJK(i,j,k)] = (epsw-epsp)*thee->epsy[IJK(i,j
05226 ,k)]
05227                 + epsp;
05228             thee->epsz[IJK(i,j,k)] = (epsw-epsp)*thee->epsz[IJK(i,j
05229 ,k)]
05230                 + epsp;
05231
05232     } /* i loop */
05233 } /* j loop */
05234 } /* k loop */
05235
05236 VPRIIVATE void fillcoCoef(Vpmg *thee) {
05237     VASSERT(thee != VNNULL);
05238
05239     if (thee->useDielXMap || thee->useDielYMap ||
05240     thee->useDielZMap || thee->useKappaMap) {
05241         fillcoCoefMap(thee);
05242         return;
05243     }
05244
05245     switch(thee->surfMeth) {
05246         case VSM_MOL:

```

```

05247             Vnm_print(0, "fillcoCoef: Calling fillcoCoefMol...\n");
05248             fillcoCoefMol(thee);
05249             break;
05250         case VSM_MOLSMOOTH:
05251             Vnm_print(0, "fillcoCoef: Calling fillcoCoefMol...\n");
05252             fillcoCoefMol(thee);
05253             break;
05254         case VSM_SPLINE:
05255             Vnm_print(0, "fillcoCoef: Calling fillcoCoefSpline...\n");
05256             fillcoCoefSpline(thee);
05257             break;
05258     case VSM_SPLINE3:
05259         Vnm_print(0, "fillcoCoef: Calling fillcoCoefSpline3...\n");
05260         fillcoCoefSpline3(thee);
05261         break;
05262     case VSM_SPLINE4:
05263         Vnm_print(0, "fillcoCoef: Calling fillcoCoefSpline4...\n");
05264         fillcoCoefSpline4(thee);
05265         break;
05266     default:
05267         Vnm_print(2, "fillcoCoef: Invalid surfMeth (%d)!\n",
05268                 thee->surfMeth);
05269         VASSERT(0);
05270         break;
05271     }
05272 }
05273
05274
05275 VPRIVATE Vrc_Codes fillcoCharge(Vpmg *thee) {
05276
05277     Vrc_Codes rc;
05278
05279     VASSERT(thee != VNULL);
05280
05281     if (thee->useChargeMap) {
05282         return fillcoChargeMap(thee);
05283     }
05284
05285     switch(thee->chargeMeth) {
05286         case VCM_TRIL:
05287             Vnm_print(0, "fillcoCharge: Calling fillcoChargeSpline1...\n");
05288             fillcoChargeSpline1(thee);
05289             break;
05290         case VCM_BSPL2:
05291             Vnm_print(0, "fillcoCharge: Calling fillcoChargeSpline2...\n");
05292             fillcoChargeSpline2(thee);
05293             break;
05294         case VCM_BSPL4:
05295             switch (thee->chargeSrc) {
05296                 case VCM_CHARGE:
05297                     Vnm_print(0, "fillcoCharge: Calling
05298                         fillcoPermanentMultipole...\n");
05299                     fillcoPermanentMultipole(thee);
05300                     break;
05301 #if defined(WITH_TINKER)
05302                     case VCM_PERMANENT:
05303                         Vnm_print(0, "fillcoCharge: Calling
05304                             fillcoPermanentMultipole...\n");
05305                         fillcoPermanentMultipole(thee);
05306                         break;
05307                     case VCM_INDUCED:
05308                         Vnm_print(0, "fillcoCharge: Calling
05309                             fillcoInducedDipole...\n");
05310                         fillcoInducedDipole(thee);
05311                         break;
05312 #endif /* if defined(WITH_TINKER) */
05313                     default:
05314                         Vnm_print(2, "fillcoCharge: Invalid chargeSource (%d)!\n",
05315                         thee->chargeSrc);
05316                         return VRC_FAILURE;
05317                         break;
05318                     }
05319                 break;
05320             default:
05321                 Vnm_print(2, "fillcoCharge: Invalid chargeMeth (%d)!\n",
05322                         thee->chargeMeth);

```

```

05324         return VRC_FAILURE;
05325     break;
05326 }
05327
05328 return VRC_SUCCESS;
05329 }
05330
05331 VPRIVATE Vrc_Codes fillcoChargeMap(Vpmg *thee) {
05332
05333     Vpbe *pbe;
05334     double position[3], charge, zmagic, hx, hy, hzed;
05335     int i, j, k, nx, ny, nz, rc;
05336
05337
05338     VASSERT(thee != VNULL);
05339
05340     /* Get PBE info */
05341     pbe = thee->pbe;
05342     zmagic = Vpbe_getZmagic(pbe);
05343
05344     /* Mesh info */
05345     nx = thee->pmgp->nx;
05346     ny = thee->pmgp->ny;
05347     nz = thee->pmgp->nz;
05348     hx = thee->pmgp->hx;
05349     hy = thee->pmgp->hy;
05350     hzed = thee->pmgp->hzed;
05351
05352     /* Reset the charge array */
05353     for (i=0; i<(nx*ny*nz); i++) thee->charge[i] = 0.0;
05354
05355     /* Fill in the source term (atomic charges) */
05356     Vnm_print(0, "Vpmg_fillco: filling in source term.\n");
05357     for (k=0; k<nz; k++) {
05358         for (j=0; j<ny; j++) {
05359             for (i=0; i<nx; i++) {
05360                 position[0] = thee->xf[i];
05361                 position[1] = thee->yf[j];
05362                 position[2] = thee->zf[k];
05363                 rc = Vgrid_value(thee->chargeMap, position, &charge);
05364                 if (!rc) {
05365                     Vnm_print(2, "fillcoChargeMap: Error -- fell off of charge map at (%g,
05366 %g, %g)\n",
05367                         position[0], position[1], position[2]);
05368                     return VRC_FAILURE;
05369                 }
05370                 /* Scale the charge to internal units */
05371                 charge = charge*zmagic;
05372                 thee->charge[IJK(i,j,k)] = charge;
05373             }
05374         }
05375     }
05376     return VRC_SUCCESS;
05377 }
05378
05379 VPRIVATE void fillcoChargeSpline1(Vpmg *thee) {
05380
05381     Valist *alist;
05382     Vpbe *pbe;
05383     Vatom *atom;
05384     double xmin, xmax, ymin, ymax, zmin, zmax;
05385     double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;
05386     double charge, dx, dy, dz, zmagic, hx, hy, hzed, *apos;
05387     int i, nx, ny, nz, atom, ihi, ilo, jhi, jlo, khi, klo;
05388
05389
05390     VASSERT(thee != VNULL);
05391
05392     /* Get PBE info */
05393     pbe = thee->pbe;
05394     alist = pbe->alist;
05395     zmagic = Vpbe_getZmagic(pbe);
05396
05397     /* Mesh info */
05398     nx = thee->pmgp->nx;
05399     ny = thee->pmgp->ny;
05400     nz = thee->pmgp->nz;
05401     hx = thee->pmgp->hx;
05402     hy = thee->pmgp->hy;
05403     hzed = thee->pmgp->hzed;

```

```

05404
05405     /* Define the total domain size */
05406     xlen = thee->pmgp->xlen;
05407     ylen = thee->pmgp->ylen;
05408     zlen = thee->pmgp->zlen;
05409
05410     /* Define the min/max dimensions */
05411     xmin = thee->pmgp->xcent - (xlen/2.0);
05412     ymin = thee->pmgp->ycent - (ylen/2.0);
05413     zmin = thee->pmgp->zcent - (zlen/2.0);
05414     xmax = thee->pmgp->xcent + (xlen/2.0);
05415     ymax = thee->pmgp->ycent + (ylen/2.0);
05416     zmax = thee->pmgp->zcent + (zlen/2.0);
05417
05418     /* Reset the charge array */
05419     for (i=0; i<(nx*ny*nz); i++) thee->charge[i] = 0.0;
05420
05421     /* Fill in the source term (atomic charges) */
05422     Vnm_print(0, "Vpmg_fillco: filling in source term.\n");
05423     for (iatom=0; iatom<Valist_getNumberAtoms(alist);
05424         iatom++) {
05425         atom = Valist_getAtom(alist, iatom);
05426         apos = Vatom_getPosition(atom);
05427         charge = Vatom_getCharge(atom);
05428
05429         /* Make sure we're on the grid */
05430         if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
05431             (apos[1]<=ymin) || (apos[1]>=ymax) || \
05432             (apos[2]<=zmin) || (apos[2]>=zmax)) {
05433             if ((thee->pmgp->bclf != BCFL_FOCUS) &&
05434                 (thee->pmgp->bclf != BCFL_MAP)) {
05435                 Vnm_print(2, "Vpmg_fillco: Atom #d at (%4.3f, %4.3f, \
05436 %4.3f) is off the mesh (ignoring):\n",
05437                     iatom, apos[0], apos[1], apos[2]);
05438                 Vnm_print(2, "Vpmg_fillco: xmin = %g, xmax = %g\n",
05439                     xmin, xmax);
05440                 Vnm_print(2, "Vpmg_fillco: ymin = %g, ymax = %g\n",
05441                     ymin, ymax);
05442                 Vnm_print(2, "Vpmg_fillco: zmin = %g, zmax = %g\n",
05443                     zmin, zmax);
05444             }
05445             fflush(stderr);
05446         } else {
05447             /* Convert the atom position to grid reference frame */
05448             position[0] = apos[0] - xmin;
05449             position[1] = apos[1] - ymin;
05450             position[2] = apos[2] - zmin;
05451
05452             /* Scale the charge to be a delta function */
05453             charge = charge*zmagic/(hx*hy*hzed);
05454
05455             /* Figure out which vertices we're next to */
05456             ifloat = position[0]/hx;
05457             jfloat = position[1]/hy;
05458             kfloat = position[2]/hzed;
05459
05460             ihi = (int)ceil(ifloat);
05461             ilo = (int)floor(ifloat);
05462             jhi = (int)ceil(jfloat);
05463             jlo = (int)floor(jfloat);
05464             khi = (int)ceil(kfloat);
05465             klo = (int)floor(kfloat);
05466
05467             /* Now assign fractions of the charge to the nearby verts */
05468             dx = ifloat - (double)(ilo);
05469             dy = jfloat - (double)(jlo);
05470             dz = kfloat - (double)(klo);
05471             thee->charge[IJK(ihi, jhi, khi)] += (dx*dy*dz*charge);
05472             thee->charge[IJK(ihi, jlo, khi)] += (dx*(1.0-dy)*dz*charge);
05473             thee->charge[IJK(ihi, jhi, klo)] += (dx*dy*(1.0-dz)*charge);
05474             thee->charge[IJK(ihi, jlo, klo)] += (dx*(1.0-dy)*(1.0-dz)*
05475             charge);
05476             thee->charge[IJK(ilo, jhi, khi)] += ((1.0-dx)*dy*dz *charge);
05477             thee->charge[IJK(ilo, jlo, khi)] += ((1.0-dx)*(1.0-dy)*dz *
05478             charge);
05479             thee->charge[IJK(ilo, jhi, klo)] += ((1.0-dx)*dy*(1.0-dz)*
05480             charge);

```

```

05480      } /* endif (on the mesh) */
05481  } /* endfor (each atom) */
05482 }
05483
05484 VPRIIVATE double bspine2(double x) {
05485
05486     double m2m, m2, m3;
05487
05488     if ((x >= 0.0) && (x <= 2.0)) m2m = 1.0 - VABS(x - 1.0);
05489     else m2m = 0.0;
05490     if ((x >= 1.0) && (x <= 3.0)) m2 = 1.0 - VABS(x - 2.0);
05491     else m2 = 0.0;
05492
05493     if ((x >= 0.0) && (x <= 3.0)) m3 = 0.5*x*m2m + 0.5*(3.0-x)*m2;
05494     else m3 = 0.0;
05495
05496     return m3;
05497 }
05498 }
05499
05500 VPRIIVATE double dbspline2(double x) {
05501
05502     double m2m, m2, dm3;
05503
05504     if ((x >= 0.0) && (x <= 2.0)) m2m = 1.0 - VABS(x - 1.0);
05505     else m2m = 0.0;
05506     if ((x >= 1.0) && (x <= 3.0)) m2 = 1.0 - VABS(x - 2.0);
05507     else m2 = 0.0;
05508
05509     dm3 = m2m - m2;
05510
05511     return dm3;
05512 }
05513 }
05514
05515
05516 VPRIIVATE void fillcoChargeSpline2(Vpmg *thee) {
05517
05518     Valist *alist;
05519     Vpbe *pbe;
05520     Vatom *atom;
05521     double xmin, xmax, ymin, ymax, zmin, zmax, zmagic;
05522     double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;
05523     double charge, hx, hy, hzed, *apos, mx, my, mz;
05524     int i, ii, jj, kk, nx, ny, nz, iatom;
05525     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
05526
05527
05528     VASSERT(thee != VNULL);
05529
05530     /* Get PBE info */
05531     pbe = thee->pbe;
05532     alist = pbe->alist;
05533     zmagic = Vpbe_getZmagic(pbe);
05534
05535     /* Mesh info */
05536     nx = thee->pmgp->nx;
05537     ny = thee->pmgp->ny;
05538     nz = thee->pmgp->nz;
05539     hx = thee->pmgp->hx;
05540     hy = thee->pmgp->hy;
05541     hzed = thee->pmgp->hzed;
05542
05543     /* Define the total domain size */
05544     xlen = thee->pmgp->xlen;
05545     ylen = thee->pmgp->ylen;
05546     zlen = thee->pmgp->zlen;
05547
05548     /* Define the min/max dimensions */
05549     xmin = thee->pmgp->xcent - (xlen/2.0);
05550     ymin = thee->pmgp->ycent - (ylen/2.0);
05551     zmin = thee->pmgp->zcent - (zlen/2.0);
05552     xmax = thee->pmgp->xcent + (xlen/2.0);
05553     ymax = thee->pmgp->ycent + (ylen/2.0);
05554     zmax = thee->pmgp->zcent + (zlen/2.0);
05555
05556     /* Reset the charge array */
05557     for (i=0; i<(nx*ny*nz); i++) thee->charge[i] = 0.0;
05558
05559     /* Fill in the source term (atomic charges) */
05560     Vnm_print(0, "Vpmg_fillco: filling in source term.\n");

```

```

05561     for (iatom=0; iatom<Valist_getNumberAtoms(alist);
05562         iatom++) {
05563         atom = Valist_getAtom(alist, iatom);
05564         apos = Vatom_getPosition(atom);
05565         charge = Vatom_getCharge(atom);
05566
05567         /* Make sure we're on the grid */
05568         if ((apos[0]<=(xmin-hx)) || (apos[0]>=(xmax+hx)) || \
05569             (apos[1]<=(ymin-hy)) || (apos[1]>=(ymax+hy)) || \
05570             (apos[2]<=(zmin-hzed)) || (apos[2]>=(zmax+hzed))) {
05571             if ((thee->pmpg->bclf != BCFL_FOCUS) &&
05572                 (thee->pmpg->bclf != BCFL_MAP)) {
05573                 Vnm_print(2, "Vpmg_fillco: Atom #d at (%4.3f, %4.3f, \
05574 %4.3f) is off the mesh (for cubic splines!!) (ignoring this atom):\n",
05575                 iatom, apos[0], apos[1], apos[2]);
05576                 Vnm_print(2, "Vpmg_fillco: xmin = %g, xmax = %g\n",
05577                 xmin, xmax);
05578                 Vnm_print(2, "Vpmg_fillco: ymin = %g, ymax = %g\n",
05579                 ymin, ymax);
05580                 Vnm_print(2, "Vpmg_fillco: zmin = %g, zmax = %g\n",
05581                 zmin, zmax);
05582             }
05583             fflush(stderr);
05584         } else {
05585
05586             /* Convert the atom position to grid reference frame */
05587             position[0] = apos[0] - xmin;
05588             position[1] = apos[1] - ymin;
05589             position[2] = apos[2] - zmin;
05590
05591             /* Scale the charge to be a delta function */
05592             charge = charge*zmagic/(hx*hy*hzed);
05593
05594             /* Figure out which vertices we're next to */
05595             ifloat = position[0]/hx;
05596             jfloat = position[1]/hy;
05597             kfloat = position[2]/hzed;
05598
05599             ip1    = (int)ceil(ifloat);
05600             ip2    = ip1 + 1;
05601             im1    = (int)floor(ifloat);
05602             im2    = im1 - 1;
05603             jp1    = (int)ceil(jfloat);
05604             jp2    = jp1 + 1;
05605             jm1    = (int)floor(jfloat);
05606             jm2    = jm1 - 1;
05607             kp1    = (int)ceil(kfloat);
05608             kp2    = kp1 + 1;
05609             km1    = (int)floor(kfloat);
05610             km2    = km1 - 1;
05611
05612             /* This step shouldn't be necessary, but it saves nasty debugging
05613              * later on if something goes wrong */
05614             ip2 = VMIN2(ip2,nx-1);
05615             ip1 = VMIN2(ip1,nx-1);
05616             im1 = VMAX2(im1,0);
05617             im2 = VMAX2(im2,0);
05618             jp2 = VMIN2(jp2,ny-1);
05619             jp1 = VMIN2(jp1,ny-1);
05620             jm1 = VMAX2(jm1,0);
05621             jm2 = VMAX2(jm2,0);
05622             kp2 = VMIN2(kp2,nz-1);
05623             kp1 = VMIN2(kp1,nz-1);
05624             km1 = VMAX2(km1,0);
05625             km2 = VMAX2(km2,0);
05626
05627             /* Now assign fractions of the charge to the nearby verts */
05628             for (ii=im2; ii<ip2; ii++) {
05629                 mx = bspline2(VFCHI(ii,ifloat));
05630                 for (jj=jm2; jj<jp2; jj++) {
05631                     my = bspline2(VFCHI(jj,jfloat));
05632                     for (kk=km2; kk<kp2; kk++) {
05633                         mz = bspline2(VFCHI(kk,kfloat));
05634                         thee->charge[IJK(ii,jj,kk)] += (charge*mx*my*mz);
05635                     }
05636                 }
05637             }
05638
05639         } /* endif (on the mesh) */
05640     } /* endfor (each atom) */

```

```

05641 }
05642
05643 VPUBLIC int Vpmg_fillco(Vpmg *thee,
05644     Vsurf_Meth surfMeth,
05645     double splineWin,
05646     Vchrg_Meth chargeMeth,
05647     int useDielXMap,
05648     Vgrid *dielXMap,
05649     int useDielYMap,
05650     Vgrid *dielYMap,
05651     int useDielZMap,
05652     Vgrid *dielZMap,
05653     int useKappaMap,
05654     Vgrid *kappaMap,
05655     int usePotMap,
05656     Vgrid *potMap,
05657     int useChargeMap,
05658     Vgrid *chargeMap
05659 ) {
05660
05661     Vpbe *pbe;
05662     double xmin,
05663         xmax,
05664         ymin,
05665         ymax,
05666         zmin,
05667         zmax,
05668         xlen,
05669         ylen,
05670         zlen,
05671         hx,
05672         hy,
05673         hzed,
05674         epsw,
05675         epsp,
05676         ionstr;
05677     int i,
05678         nx,
05679         ny,
05680         nz,
05681         islap;
05682     Vrc_Codes rc;
05683
05684     if (thee == VNULL) {
05685         Vnm_print(2, "Vpmg_fillco: got NULL thee!\n");
05686         return 0;
05687     }
05688
05689     thee->surfMeth = surfMeth;
05690     thee->splineWin = splineWin;
05691     thee->chargeMeth = chargeMeth;
05692     thee->useDielXMap = useDielXMap;
05693     if (thee->useDielXMap) thee->dielXMap = dielXMap;
05694     thee->useDielYMap = useDielYMap;
05695     if (thee->useDielYMap) thee->dielYMap = dielYMap;
05696     thee->useDielZMap = useDielZMap;
05697     if (thee->useDielZMap) thee->dielZMap = dielZMap;
05698     thee->useKappaMap = useKappaMap;
05699     if (thee->useKappaMap) thee->kappaMap = kappaMap;
05700     thee->usePotMap = usePotMap;
05701     if (thee->usePotMap) thee->potMap = potMap;
05702     thee->useChargeMap = useChargeMap;
05703     if (thee->useChargeMap) thee->chargeMap = chargeMap;
05704
05705     /* Get PBE info */
05706     pbe = thee->pbe;
05707     ionstr = Vpbe_getBulkIonicStrength(pbe);
05708     epsw = Vpbe_getSolventDiel(pbe);
05709     epsp = Vpbe_getSoluteDiel(pbe);
05710
05711     /* Mesh info */
05712     nx = thee->pmgp->nx;
05713     ny = thee->pmgp->ny;
05714     nz = thee->pmgp->nz;
05715     hx = thee->pmgp->hx;
05716     hy = thee->pmgp->hy;
05717     hzed = thee->pmgp->hzed;
05718
05719     /* Define the total domain size */
05720     xlen = thee->pmgp->xlen;
05721     ylen = thee->pmgp->ylen;

```

```

05722     zlen = thee->pmgp->zlen;
05723
05724     /* Define the min/max dimensions */
05725     xmin = thee->pmgp->xcent - (xlen/2.0);
05726     thee->pmgp->xmin = xmin;
05727     ymin = thee->pmgp->ycent - (ylen/2.0);
05728     thee->pmgp->ymin = ymin;
05729     zmin = thee->pmgp->zcent - (zlen/2.0);
05730     thee->pmgp->zmin = zmin;
05731     xmax = thee->pmgp->xcent + (xlen/2.0);
05732     thee->pmgp->xmax = xmax;
05733     ymax = thee->pmgp->ycent + (ylen/2.0);
05734     thee->pmgp->ymax = ymax;
05735     zmax = thee->pmgp->zcent + (zlen/2.0);
05736     thee->pmgp->zmax = zmax;
05737     thee->rparm[2] = xmin;
05738     thee->rparm[3] = xmax;
05739     thee->rparm[4] = ymin;
05740     thee->rparm[5] = ymax;
05741     thee->rparm[6] = zmin;
05742     thee->rparm[7] = zmax;
05743
05744     /* This is a flag that gets set if the operator is a simple Laplacian;
05745      * i.e., in the case of a homogenous dielectric and zero ionic strength
05746      * The operator cannot be a simple Laplacian if maps are read in. */
05747     if(thee->useDie1XMap || thee->useDie1YMap || thee->
05748       useDie1ZMap ||
05749       thee->useKappaMap || thee->usePotMap){
05750     } else if ( (ionstr < VPMGSMALL) && (VABS(epsp-epsw) < VPMGSMALL
05751     ) ){
05751       islap = 1;
05752     }else{
05753       islap = 0;
05754     }
05755
05756     /* Fill the mesh point coordinate arrays */
05757     for (i=0; i<nx; i++) thee->xf[i] = xmin + i*hx;
05758     for (i=0; i<ny; i++) thee->yi[i] = ymin + i*hy;
05759     for (i=0; i<nz; i++) thee->zf[i] = zmin + i*hzed;
05760
05761     /* Reset the tcf array */
05762     for (i=0; i<(nx*ny*nz); i++) thee->tcf[i] = 0.0;
05763
05764     /* Fill in the source term (atomic charges) */
05765     Vnm_print(0, "Vpmg_fillco: filling in source term.\n");
05766     rc = fillcoCharge(thee);
05767     switch(rc) {
05768     case VRC_SUCCESS:
05769       break;
05770     case VRC_WARNING:
05771       Vnm_print(2, "Vpmg_fillco: non-fatal errors while filling charge map!\n");
05772       break;
05773     case VRC_FAILURE:
05774       Vnm_print(2, "Vpmg_fillco: fatal errors while filling charge map!\n");
05775       return 0;
05776       break;
05777   }
05778
05779     /* THE FOLLOWING NEEDS TO BE DONE IF WE'RE NOT USING A SIMPLE LAPLACIAN
05780      * OPERATOR */
05781     if (!islip) {
05782       Vnm_print(0, "Vpmg_fillco: marking ion and solvent accessibility.\n");
05783       fillcoCoef(thee);
05784       Vnm_print(0, "Vpmg_fillco: done filling coefficient arrays\n");
05785
05786     } else { /* else (!islip) ==> It's a Laplacian operator! */
05787
05788       for (i=0; i<(nx*ny*nz); i++) {
05789         thee->kappa[i] = 0.0;
05790         thee->epsx[i] = epsp;
05791         thee->epsy[i] = epsp;
05792         thee->epsz[i] = epsp;
05793       }
05794
05795     } /* endif (!islip) */
05796
05797     /* Fill the boundary arrays (except when focusing, bcfl = 4) */
05798     if (thee->pmgp->bcfl != BCFL_FOCUS) {
05799       Vnm_print(0, "Vpmg_fillco: filling boundary arrays\n");
05800       bcCalc(thee);

```

```

05801     Vnm_print(0, "Vpmg_fillco: done filling boundary arrays\n");
05802 }
05803
05804     thee->filled = 1;
05805
05806     return 1;
05807 }
05808
05809
05810 VPUBLIC int Vpmg_force(Vpmg *thee, double *force, int atomID,
05811   Vsurf_Meth srfm, Vchrg_Meth chgm) {
05812
05813     int rc = 1;
05814     double qFF[3];           /* Charge-field force */
05815     double dbF[3];          /* Dielectric boundary force */
05816     double ibF[3];          /* Ion boundary force */
05817     double npF[3];          /* Non-polar boundary force */
05818
05819     VASSERT(thee != VNULL);
05820
05821     rc = rc && Vpmg_dbForce(thee, qFF, atomID, srfm);
05822     rc = rc && Vpmg_ibForce(thee, dbF, atomID, srfm);
05823     rc = rc && Vpmg_qfForce(thee, ibF, atomID, chgm);
05824
05825     force[0] = qFF[0] + dbF[0] + ibF[0];
05826     force[1] = qFF[1] + dbF[1] + ibF[1];
05827     force[2] = qFF[2] + dbF[2] + ibF[2];
05828
05829     return rc;
05830
05831 }
05832
05833 VPUBLIC int Vpmg_ibForce(Vpmg *thee, double *force, int atomID,
05834   Vsurf_Meth srfm) {
05835
05836     Valist *alist;
05837     Vacc *acc;
05838     Vpbe *pbe;
05839     Vatom *atom;
05840
05841     double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
05842     double xlabel, ylabel, zlabel, xmin, ymin, zmin, xmax
05843 , ymax, zmax, rrot2;
05844     double rotx, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
05845     double izmagic;
05846     int i, j, k, nx, ny, nz, imin,imax, jmin, jmax, kmin, kmax;
05847
05848 /* For nonlinear forces */
05849     int ichop, nchop, nion, m;
05850     double ionConc[MAXION], ionRadii[MAXION], ionQ[MAXION],
05851     ionstr;
05852
05853     VASSERT(thee != VNULL);
05854
05855     acc = thee->pbe->acc;
05856     atom = Valist_getAtom(thee->pbe->alist, atomID);
05857     apos = Vatom_getPosition(atom);
05858     arad = Vatom_getRadius(atom);
05859
05860     /* Reset force */
05861     force[0] = 0.0;
05862     force[1] = 0.0;
05863     force[2] = 0.0;
05864
05865     /* Check surface definition */
05866     if ((srfm != VSM SPLINE) && (srfm!=VSM SPLINE3) && (
05867       srfm!=VSM SPLINE4)) {
05868       Vnm_print(2, "Vpmg_ibForce: Forces *must* be calculated with \
05869 spline-based surfaces!\n");
05870       Vnm_print(2, "Vpmg_ibForce: Skipping ionic boundary force \
05871 calculation!\n");
05872       return 0;
05873     }
05874
05875     /* If we aren't in the current position, then we're done */
05876     if (atom->partID == 0) return 1;
05877
05878     /* Get PBE info */
05879     pbe = thee->pbe;
05880     acc = pbe->acc;
05881     alist = pbe->alist;

```

```

05879     irad = Vpbe_getMaxIonRadius(pbe);
05880     zkappa2 = Vpbe_getZkappa2(pbe);
05881     izmagic = 1.0/Vpbe_getZmagic(pbe);
05882
05883     ionstr = Vpbe_getBulkIonicStrength(pbe);
05884     Vpbe_getIons(pbe, &ion, ionConc, ionRadii, ionQ);
05885
05886     /* Mesh info */
05887     nx = thee->pmgp->nx;
05888     ny = thee->pmgp->ny;
05889     nz = thee->pmgp->nz;
05890     hx = thee->pmgp->hx;
05891     hy = thee->pmgp->hy;
05892     hzed = thee->pmgp->hzed;
05893     xlen = thee->pmgp->xlen;
05894     ylen = thee->pmgp->ylen;
05895     zlen = thee->pmgp->zlen;
05896     xmin = thee->pmgp->xmin;
05897     ymin = thee->pmgp->ymin;
05898     zmin = thee->pmgp->zmin;
05899     xmax = thee->pmgp->xmax;
05900     ymax = thee->pmgp->ymax;
05901     zmax = thee->pmgp->zmax;
05902
05903     /* Sanity check: there is no force if there is zero ionic strength */
05904     if (zkappa2 < VPMGSMALL) {
05905 #ifndef VAPBSQUIET
05906         Vnm_print(2, "Vpmg_ibForce: No force for zero ionic strength!\n");
05907 #endif
05908         return 1;
05909     }
05910
05911     /* Make sure we're on the grid */
05912     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
05913         (apos[1]<=ymin) || (apos[1]>=ymax) || \
05914         (apos[2]<=zmin) || (apos[2]>=zmax)) {
05915     if ((thee->pmgp->bclf != BCFL_FOCUS) &&
05916         (thee->pmgp->bclf != BCFL_MAP)) {
05917         Vnm_print(2, "Vpmg_ibForce: Atom #%d at (%4.3f, %4.3f, %4.3f) is
05918 off the mesh (ignoring):\n",
05919             atom, apos[0], apos[1], apos[2]);
05920         Vnm_print(2, "Vpmg_ibForce: xmin = %g, xmax = %g\n",
05921             xmin, xmax);
05922         Vnm_print(2, "Vpmg_ibForce: ymin = %g, ymax = %g\n",
05923             ymin, ymax);
05924         Vnm_print(2, "Vpmg_ibForce: zmin = %g, zmax = %g\n",
05925             zmin, zmax);
05926     }
05927     fflush(stderr);
05928 } else {
05929     /* Convert the atom position to grid reference frame */
05930     position[0] = apos[0] - xmin;
05931     position[1] = apos[1] - ymin;
05932     position[2] = apos[2] - zmin;
05933
05934     /* Integrate over points within this atom's (inflated) radius */
05935     rtot = (irad + arad + thee->splineWin);
05936     rtot2 = VSQR(rtot);
05937     dx = rtot + 0.5*hx;
05938     imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
05939     imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
05940     for (i=imin; i<imax; i++) {
05941         dx2 = VSQR(position[0] - hx*i);
05942         if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
05943         else dy = 0.5*hy;
05944         jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
05945         jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
05946         for (j=jmin; j<jmax; j++) {
05947             dy2 = VSQR(position[1] - hy*j);
05948             if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
05949             else dz = 0.5*hzed;
05950             kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
05951             kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
05952             for (k=kmin; k<=kmax; k++) {
05953                 dz2 = VSQR(k*hzed - position[2]);
05954                 /* See if grid point is inside ivdw radius and set kappa
05955                  * accordingly (do spline assignment here) */
05956                 if ((dz2 + dy2 + dx2) <= rtot2) {
05957                     gpos[0] = i*hx + xmin;
05958                     gpos[1] = j*hy + ymin;

```

```

05959                     gpos[2] = k*hzed + zmin;
05960
05961     /* Select the correct function based on the surface definition
05962     * (now including the 7th order polynomial) */
05963     Vpmg_splineSelect(srfm,acc, gpos,thee->splineWin
05964 , irad, atom, tgrad);
05965
05966     if (thee->pmgp->nonlin) {
05967         /* Nonlinear forces */
05968         fmag = 0.0;
05969         nchop = 0;
05970         for (m=0; m<nion; m++) {
05971             fmag += (thee->kappa[IJK(i,j,k)])*ionConc[
05972             m]* (Vcap_exp(-ionQ[m]*thee->u[IJK(i,j,k)], &ichop)-1.0)/ionstr;
05973             nchop += ichop;
05974         }
05975         /* if (nchop > 0) Vnm_print(2, "Vpmg_ibForce: Chopped EXP %d
05976         times!\n", nchop); */
05977
05978         force[0] += (zkappa2*fmag*tgrad[0]);
05979         force[1] += (zkappa2*fmag*tgrad[1]);
05980         force[2] += (zkappa2*fmag*tgrad[2]);
05981     } else {
05982         /* Use of bulk factor (zkappa2) OK here because
05983         * LPBE force approximation */
05984         /* NAB -- did we forget a kappa factor here??? */
05985         fmag = VSQR(thee->u[IJK(i,j,k)])*(thee->kappa
05986 [IJK(i,j,k)]);
05987         force[0] += (zkappa2*fmag*tgrad[0]);
05988         force[1] += (zkappa2*fmag*tgrad[1]);
05989         force[2] += (zkappa2*fmag*tgrad[2]);
05990     }
05991     force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
05992     force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
05993     force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
05994
05995     return 1;
05996 }
05997
05998 VPUBLIC int Vpmg_dbForce(Vpmg *thee, double *dbForce, int
05999 atomID,
06000     Vsurf_Meth srfm) {
06001
06002     Vacc *acc;
06003     Vpbe *pbe;
06004     Vatom *atom;
06005
06006     double *apos, position[3], arad, srad, hx, hy, hzed, izmagic, deps,
06007    depsi;
06008     double xlabel, ylabel, zlabel, xmin, ymin, zmin, xmax
06009     , ymax, zmax, rtot2, epsp;
06010     double rtot, dx, gpos[3], dbFmag, epsw, kT;
06011     double *u, Hxijk, Hyijk, Hzijk, Hximljk, Hyjm1k, Hzijkml;
06012     double dHxijk[3], dHyijk[3], dHzijk[3], dHximljk[3], dHyjm1k[3];
06013     double dHzijkml[3];
06014     int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
06015
06016     VASSERT(thee != VNULL);
06017     if (!thee->filled) {
06018         Vnm_print(2, "Vpmg_dbForce: Need to callVpmg_fillco!\n");
06019         return 0;
06020     }
06021
06022     acc = thee->pbe->acc;
06023     atom = Valist_getAtom(thee->pbe->alist, atomID);
06024     apos = Vatom_getPosition(atom);
06025     arad = Vatom_getRadius(atom);
06026     srad = Vpbe_getSolventRadius(thee->pbe);
06027
06028     /* Reset force */
06029     dbForce[0] = 0.0;
06030     dbForce[1] = 0.0;
06031     dbForce[2] = 0.0;
06032
06033     /* Check surface definition */
06034     if ((srfm != VSM_SPLINE) && (srfm!=VSM_SPLINE3) &&
06035     (srfm!=VSM_SPLINE4)) {

```

```

06032     Vnm_print(2, "Vpmg_dbForce: Forces *must* be calculated with \
06033 spline-based surfaces!\n");
06034     Vnm_print(2, "Vpmg_dbForce: Skipping dielectric/apolar boundary \
06035 force calculation!\n");
06036     return 0;
06037 }
06038
06039
06040 /* If we aren't in the current position, then we're done */
06041 if (atom->partID == 0) return 1;
06042
06043 /* Get PBE info */
06044 pbe = thee->pbe;
06045 acc = pbe->acc;
06046 epsp = Vpbe_getSoluteDiel(pbe);
06047 epsw = Vpbe_getSolventDiel(pbe);
06048 kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*
    Vunit_kb;
06049 izmagic = 1.0/Vpbe_getZmagic(pbe);
06050
06051 /* Mesh info */
06052 nx = thee->pmgp->nx;
06053 ny = thee->pmgp->ny;
06054 nz = thee->pmgp->nz;
06055 hx = thee->pmgp->hx;
06056 hy = thee->pmgp->hy;
06057 hzed = thee->pmgp->hzed;
06058 xlen = thee->pmgp->xlen;
06059 ylen = thee->pmgp->ylen;
06060 zlen = thee->pmgp->zlen;
06061 xmin = thee->pmgp->xmin;
06062 ymin = thee->pmgp->ymin;
06063 zmin = thee->pmgp->zmin;
06064 xmax = thee->pmgp->xmax;
06065 ymax = thee->pmgp->ymax;
06066 zmax = thee->pmgp->zmax;
06067 u = thee->u;
06068
06069 /* Sanity check: there is no force if there is zero ionic strength */
06070 if (VABS(epsp-epsw) < VPMGSMALL) {
06071 Vnm_print(0, "Vpmg_dbForce: No force for uniform dielectric!\n");
06072 return 1;
06073 }
06074 deps = (epsw - epsp);
06075 depsi = 1.0/deps;
06076 rtot = (arad + thee->splineWin + srad);
06077
06078 /* Make sure we're on the grid */
06079 /* Grid checking modified by Matteo Rotter */
06080 if ((apos[0]<=xmin + rtot) || (apos[0]>=xmax - rtot) || \
06081 (apos[1]<=ymin + rtot) || (apos[1]>=ymax - rtot) || \
06082 (apos[2]<=zmin + rtot) || (apos[2]>=zmax - rtot)) {
06083 if ((thee->pmgp->bclf != BCFL_FOCUS) &&
06084 (thee->pmgp->bclf != BCFL_MAP)) {
06085     Vnm_print(2, "Vpmg_dbForce: Atom %d at (%4.3f, %4.3f, %4.3f) is
off the mesh (ignoring):\n",
06086     atomID, apos[0], apos[1], apos[2]);
06087     Vnm_print(2, "Vpmg_dbForce: xmin = %g, xmax = %g\n",
06088     xmin, xmax);
06089     Vnm_print(2, "Vpmg_dbForce: ymin = %g, ymax = %g\n",
06090     ymin, ymax);
06091     Vnm_print(2, "Vpmg_dbForce: zmin = %g, zmax = %g\n",
06092     zmin, zmax);
06093     }
06094     fflush(stderr);
06095 } else {
06096
06097     /* Convert the atom position to grid reference frame */
06098     position[0] = apos[0] - xmin;
06099     position[1] = apos[1] - ymin;
06100     position[2] = apos[2] - zmin;
06101
06102     /* Integrate over points within this atom's (inflated) radius */
06103     rtot2 = VSQR(rtot);
06104     dx = rtot/hx;
06105     imin = (int)floor((position[0]-rtot)/hx);
06106     if (imin < 1) {
06107         Vnm_print(2, "Vpmg_dbForce: Atom %d off grid!\n", atomID);
06108         return 0;
06109     }
06110     imax = (int)ceil((position[0]+rtot)/hx);

```

```

06111     if (imax > (nx-2)) {
06112         Vnm_print(2, "Vpmg_dbForce: Atom %d off grid!\n", atomID);
06113         return 0;
06114     }
06115     jmin = (int)floor((position[1]-rtot)/hy);
06116     if (jmin < 1) {
06117         Vnm_print(2, "Vpmg_dbForce: Atom %d off grid!\n", atomID);
06118         return 0;
06119     }
06120     jmax = (int)ceil((position[1]+rtot)/hy);
06121     if (jmax > (ny-2)) {
06122         Vnm_print(2, "Vpmg_dbForce: Atom %d off grid!\n", atomID);
06123         return 0;
06124     }
06125     kmin = (int)floor((position[2]-rtot)/hzed);
06126     if (kmin < 1) {
06127         Vnm_print(2, "Vpmg_dbForce: Atom %d off grid!\n", atomID);
06128         return 0;
06129     }
06130     kmax = (int)ceil((position[2]+rtot)/hzed);
06131     if (kmax > (nz-2)) {
06132         Vnm_print(2, "Vpmg_dbForce: Atom %d off grid!\n", atomID);
06133         return 0;
06134     }
06135     for (i=iimin; i<=imax; i++) {
06136         for (j=jmin; j<=jmax; j++) {
06137             for (k=kmin; k<=kmax; k++) {
06138                 /* i,j,k */
06139                 gpos[0] = (i+0.5)*hx + xmin;
06140                 gpos[1] = j*hy + ymin;
06141                 gpos[2] = k*hzed + zmin;
06142                 Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
06143
06144             /* Select the correct function based on the surface definition
06145             * (now including the 7th order polynomial) */
06146             Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin
06147             , 0.,atom, dHxijk);
06148             /*
06149             switch (srfm) {
06150                 case VSM_SPLINE :
06151                     Vacc_splineAccGradAtomNorm(acc, gpos, thee->splineWin, 0.,
06152                         atom, dHxijk);
06153                     break;
06154                 case VSM_SPLINE4 :
06155                     Vacc_splineAccGradAtomNorm4(acc, gpos, thee->splineWin, 0.,
06156                         atom, dHxijk);
06157                     break;
06158                 default:
06159                     Vnm_print(2, "Vpmg_dnbForce: Unknown surface method.\n");
06160                     return;
06161             }
06162             /*
06163                 for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
06164                 gpos[0] = i*hx + xmin;
06165                 gpos[1] = (j+0.5)*hy + ymin;
06166                 gpos[2] = k*hzed + zmin;
06167                 Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
06168
06169             /* Select the correct function based on the surface definition
06170             * (now including the 7th order polynomial) */
06171             Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin
06172             , 0.,atom, dHyijk);
06173             /*
06174                 for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
06175                 gpos[0] = i*hx + xmin;
06176                 gpos[1] = (j+0.5)*hy + ymin;
06177                 gpos[2] = (k+0.5)*hzed + zmin;
06178                 Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
06179
06180             /* Select the correct function based on the surface definition
06181             * (now including the 7th order polynomial) */
06182             Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin
06183             , 0.,atom, dHzijk);
06184             /*
06185                 for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
06186                 /* i-1,j,k */
06187                 gpos[0] = (i-0.5)*hx + xmin;
06188                 gpos[1] = j*hy + ymin;
06189                 gpos[2] = k*hzed + zmin;
06190                 Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
06191
06192
06193
06194
06195
06196
06197
06198
06199
06200
06201
06202
06203
06204
06205
06206
06207
06208
06209
06210
06211
06212
06213
06214
06215
06216
06217
06218
06219
06220
06221
06222
06223
06224
06225
06226
06227
06228
06229
06230
06231
06232
06233
06234
06235
06236
06237
06238
06239
06240
06241
06242
06243
06244
06245
06246
06247
06248
06249
06250
06251
06252
06253
06254
06255
06256
06257
06258
06259
06260
06261
06262
06263
06264
06265
06266
06267
06268
06269
06270
06271
06272
06273
06274
06275
06276
06277
06278
06279
06280
06281
06282
06283
06284
06285
06286
06287
06288
06289
06290
06291
06292
06293
06294
06295
06296
06297
06298
06299
06300
06301
06302
06303
06304
06305
06306
06307
06308
06309
06310
06311
06312
06313
06314
06315
06316
06317
06318
06319
06320
06321
06322
06323
06324
06325
06326
06327
06328
06329
06330
06331
06332
06333
06334
06335
06336
06337
06338
06339
06340
06341
06342
06343
06344
06345
06346
06347
06348
06349
06350
06351
06352
06353
06354
06355
06356
06357
06358
06359
06360
06361
06362
06363
06364
06365
06366
06367
06368
06369
06370
06371
06372
06373
06374
06375
06376
06377
06378
06379
06380
06381
06382
06383
06384
06385
06386
06387
06388
06389
06390
06391
06392
06393
06394
06395
06396
06397
06398
06399
06400
06401
06402
06403
06404
06405
06406
06407
06408
06409
06410
06411
06412
06413
06414
06415
06416
06417
06418
06419
06420
06421
06422
06423
06424
06425
06426
06427
06428
06429
06430
06431
06432
06433
06434
06435
06436
06437
06438
06439
06440
06441
06442
06443
06444
06445
06446
06447
06448
06449
06450
06451
06452
06453
06454
06455
06456
06457
06458
06459
06460
06461
06462
06463
06464
06465
06466
06467
06468
06469
06470
06471
06472
06473
06474
06475
06476
06477
06478
06479
06480
06481
06482
06483
06484
06485
06486
06487
06488
06489
06490
06491
06492
06493
06494
06495
06496
06497
06498
06499
06500
06501
06502
06503
06504
06505
06506
06507
06508
06509
06510
06511
06512
06513
06514
06515
06516
06517
06518
06519
06520
06521
06522
06523
06524
06525
06526
06527
06528
06529
06530
06531
06532
06533
06534
06535
06536
06537
06538
06539
06540
06541
06542
06543
06544
06545
06546
06547
06548
06549
06550
06551
06552
06553
06554
06555
06556
06557
06558
06559
06560
06561
06562
06563
06564
06565
06566
06567
06568
06569
06570
06571
06572
06573
06574
06575
06576
06577
06578
06579
06580
06581
06582
06583
06584
06585
06586
06587
06588
06589
06590
06591
06592
06593
06594
06595
06596
06597
06598
06599
06600
06601
06602
06603
06604
06605
06606
06607
06608
06609
06610
06611
06612
06613
06614
06615
06616
06617
06618
06619
06620
06621
06622
06623
06624
06625
06626
06627
06628
06629
06630
06631
06632
06633
06634
06635
06636
06637
06638
06639
06640
06641
06642
06643
06644
06645
06646
06647
06648
06649
06650
06651
06652
06653
06654
06655
06656
06657
06658
06659
06660
06661
06662
06663
06664
06665
06666
06667
06668
06669
06670
06671
06672
06673
06674
06675
06676
06677
06678
06679
06680
06681
06682
06683
06684
06685
06686
06687
06688
06689
06690
06691
06692
06693
06694
06695
06696
06697
06698
06699
06700
06701
06702
06703
06704
06705
06706
06707
06708
06709
06710
06711
06712
06713
06714
06715
06716
06717
06718
06719
06720
06721
06722
06723
06724
06725
06726
06727
06728
06729
06730
06731
06732
06733
06734
06735
06736
06737
06738
06739
06740
06741
06742
06743
06744
06745
06746
06747
06748
06749
06750
06751
06752
06753
06754
06755
06756
06757
06758
06759
06760
06761
06762
06763
06764
06765
06766
06767
06768
06769
06770
06771
06772
06773
06774
06775
06776
06777
06778
06779
06780
06781
06782
06783
06784
06785
06786
06787
06788
06789
06790
06791
06792
06793
06794
06795
06796
06797
06798
06799
06800
06801
06802
06803
06804
06805
06806
06807
06808
06809
06810
06811
06812
06813
06814
06815
06816
06817
06818
06819
06820
06821
06822
06823
06824
06825
06826
06827
06828
06829
06830
06831
06832
06833
06834
06835
06836
06837
06838
06839
06840
06841
06842
06843
06844
06845
06846
06847
06848
06849
06850
06851
06852
06853
06854
06855
06856
06857
06858
06859
06860
06861
06862
06863
06864
06865
06866
06867
06868
06869
06870
06871
06872
06873
06874
06875
06876
06877
06878
06879
06880
06881
06882
06883
06884
06885
06886
06887
06888
06889
06890
06891
06892
06893
06894
06895
06896
06897
06898
06899
06900
06901
06902
06903
06904
06905
06906
06907
06908
06909
06910
06911
06912
06913
06914
06915
06916
06917
06918
06919
06920
06921
06922
06923
06924
06925
06926
06927
06928
06929
06930
06931
06932
06933
06934
06935
06936
06937
06938
06939
06940
06941
06942
06943
06944
06945
06946
06947
06948
06949
06950
06951
06952
06953
06954
06955
06956
06957
06958
06959
06960
06961
06962
06963
06964
06965
06966
06967
06968
06969
06970
06971
06972
06973
06974
06975
06976
06977
06978
06979
06980
06981
06982
06983
06984
06985
06986
06987
06988
06989
06990
06991
06992
06993
06994
06995
06996
06997
06998
06999
06999
07000
07001
07002
07003
07004
07005
07006
07007
07008
07009
070010
070011
070012
070013
070014
070015
070016
070017
070018
070019
070020
070021
070022
070023
070024
070025
070026
070027
070028
070029
070030
070031
070032
070033
070034
070035
070036
070037
070038
070039
070040
070041
070042
070043
070044
070045
070046
070047
070048
070049
070050
070051
070052
070053
070054
070055
070056
070057
070058
070059
070060
070061
070062
070063
070064
070065
070066
070067
070068
070069
070070
070071
070072
070073
070074
070075
070076
070077
070078
070079
070080
070081
070082
070083
070084
070085
070086
070087
070088
070089
070090
070091
070092
070093
070094
070095
070096
070097
070098
070099
070099
070100
070101
070102
070103
070104
070105
070106
070107
070108
070109
070110
070111
070112
070113
070114
070115
070116
070117
070118
070119
070120
070121
070122
070123
070124
070125
070126
070127
070128
070129
070130
070131
070132
070133
070134
070135
070136
070137
070138
070139
070140
070141
070142
070143
070144
070145
070146
070147
070148
070149
070150
070151
070152
070153
070154
070155
070156
070157
070158
070159
070160
070161
070162
070163
070164
070165
070166
070167
070168
070169
070170
070171
070172
070173
070174
070175
070176
070177
070178
070179
070180
070181
070182
070183
070184
070185
070186
070187
070188
070189
070190
070191
070192
070193
070194
070195
070196
070197
070198
070199
070199
070200
070201
070202
070203
070204
070205
070206
070207
070208
070209
070210
070211
070212
070213
070214
070215
070216
070217
070218
070219
070220
070221
070222
070223
070224
070225
070226
070227
070228
070229
070229
070230
070231
070232
070233
070234
070235
070236
070237
070238
070239
070239
070240
070241
070242
070243
070244
070245
070246
070247
070248
070249
070249
070250
070251
070252
070253
070254
070255
070256
070257
070258
070259
070259
070260
070261
070262
070263
070264
070265
070266
070267
070268
070269
070269
070270
070271
070272
070273
070274
070275
070276
070277
070278
070279
070279
070280
070281
070282
070283
070284
070285
070286
070287
070288
070289
070289
070290
070291
070292
070293
070294
070295
070296
070297
070298
070299
070299
070300
070301
070302
070303
070304
070305
070306
070307
070308
070309
070309
070310
070311
070312
070313
070314
070315
070316
070317
070318
070319
070319
070320
070321
070322
070323
070324
070325
070326
070327
070328
070329
070329
070330
070331
070332
070333
070334
070335
070336
070337
070338
070339
070339
070340
070341
070342
070343
070344
070345
070346
070347
070348
070349
070349
070350
070351
070352
070353
070354
070355
070356
070357
070358
070359
070359
070360
070361
070362
070363
070364
070365
070366
070367
070368
070369
070369
070370
070371
070372
070373
070374
070375
070376
070377
070378
070379
070379
070380
070381
070382
070383
070384
070385
070386
070387
070388
070389
070389
070390
070391
070392
070393
070394
070395
070396
070397
070398
070399
070399
070400
070401
070402
070403
070404
070405
070406
070407
070408
070409
070409
070410
070411
070412
070413
070414
070415
070416
070417
070418
070419
070419
070420
070421
070422
070423
070424
070425
070426
070427
070428
070429
070429
070430
070431
070432
070433
070434
070435
070436
070437
070438
070439
070439
070440
070441
070442
070443
070444
070445
070446
070447
070448
070449
070449
070450
070451
070452
070453
070454
070455
070456
070457
070458
070459
070459
070460
070461
070462
070463
070464
070465
070466
070467
070468
070469
070469
070470
070471
070472
070473
070474
070475
070476
070477
070478
070479
070479
070480
070481
070482
070483
070484
070485
070486
070487
070488
070489
070489
070490
070491
070492
070493
070494
070495
070496
070497
070498
070499
070499
070500
070501
070502
070503
070504
070505
070506
070507
070508
070509
070509
070510
070511
070512
070513
070514
070515
070516
070517
070518
070519
070519
070520
070521
070522
070523
070524
070525
070526
070527
070528
070529
070529
070530
070531
070532
070533
070534
070535
070536
070537
070538
070539
070539
070540
070541
070542
070543
070544
070545
070546
070547
070548
070549
070549
070550
070551
070552
070553
070554
070555
070556
070557
070558
070559
070559
070560
070561
070562
070563
070564
070565
070566
070567
070568
070569
070569
070570
070571
070572
070573
070574
070575
070576
070577
070578
070579
070579
070580
070581
070582
070583
070584
070585
070586
070587
070588
070589
070589
070590
070591
070592
070593
070594
070595
070596
070597
070598
070599
070599
070600
070601
070602
070603
070604
070605
070606
070607
070608
070609
070609
070610
070611
070612
070613
070614
070615
070616
070617
070618
070619
070619
070620
070621
070622
070623
070624
070625
070626
070627
070628
070629
070629
070630
070631
070632
070633
070634
070635
070636
070637
070638
070639
070639
070640
070641
070642
070643
070644
070645
070646
070647
070648
070649
070649
070650
070651
070652
070653
070654
070655
070656
070657
070658
070659
070659
070660
070661
070662
070663
070664
070665
070666
070667
070668
070
```

```

06189     /* Select the correct function based on the surface definition
06190      * (now including the 7th order polynomial) */
06191      Vpmg_splineSelect(srfm,acc,gpos,thee->splineWin
06192      ,0.,atom,dHxim1jk);
06193      for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
06194      /* i,j-1,k */
06195      gpos[0] = i*hx + xmin;
06196      gpos[1] = (j-0.5)*hy + ymin;
06197      gpos[2] = k*hzed + zmin;
06198      Hyijm1k = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
06199
06200      /* Select the correct function based on the surface definition
06201      * (now including the 7th order polynomial) */
06202      Vpmg_splineSelect(srfm,acc,gpos,thee->splineWin
06203      ,0.,atom,dHyijm1k);
06204      for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;
06205      /* i,j,k-1 */
06206      gpos[0] = i*hx + xmin;
06207      gpos[1] = j*hy + ymin;
06208      gpos[2] = (k-0.5)*hzed + zmin;
06209      Hzijkml = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
06210
06211      /* Select the correct function based on the surface definition
06212      * (now including the 7th order polynomial) */
06213      Vpmg_splineSelect(srfm,acc,gpos,thee->splineWin
06214      ,0.,atom,dHzijkml);
06215      for (l=0; l<3; l++) dHzijkml[l] *= Hzijkml;
06216      /* *** CALCULATE DIELECTRIC BOUNDARY FORCES *** */
06217      dbFmag = u[IJK(i,j,k)];
06218      tgrad[0] =
06219      (dHxijk[0] *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
06220      + dHxim1jk[0]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
06221      + (dHyijk[0] *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
06222      + dHyijm1k[0]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
06223      + (dHzijk[0] *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
06224      + dHzijkml[0]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
06225      tgrad[1] =
06226      (dHxijk[1] *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
06227      + dHxim1jk[1]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
06228      + (dHyijk[1] *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
06229      + dHyijm1k[1]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
06230      + (dHzijk[1] *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
06231      + dHzijkml[1]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
06232      tgrad[2] =
06233      (dHxijk[2] *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
06234      + dHxim1jk[2]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
06235      + (dHyijk[2] *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
06236      + dHyijm1k[2]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
06237      + (dHzijk[2] *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
06238      + dHzijkml[2]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
06239      dbForce[0] += (dbFmag*tgrad[0]);
06240      dbForce[1] += (dbFmag*tgrad[1]);
06241      dbForce[2] += (dbFmag*tgrad[2]);
06242
06243      } /* k loop */
06244      } /* j loop */
06245      } /* i loop */
06246
06247      dbForce[0] = -dbForce[0]*hx*hy*hzed*deps*0.5*izmagic;
06248      dbForce[1] = -dbForce[1]*hx*hy*hzed*deps*0.5*izmagic;
06249      dbForce[2] = -dbForce[2]*hx*hy*hzed*deps*0.5*izmagic;
06250  }
06251
06252  return 1;
06253 }
06254
06255 VPUBLIC int Vpmg_qfForce(Vpmg *thee, double *force, int atomID,
06256   Vchrg_Meth chgm) {
06257
06258  double tforce[3];
06259
06260  /* Reset force */
06261  force[0] = 0.0;
06262  force[1] = 0.0;
06263  force[2] = 0.0;
06264
06265  /* Check surface definition */
06266  if (chgm != VCM_BSPL2) {

```

```

06267     Vnm_print(2, "Vpmg_qfForce: It is recommended that forces be \
06268 calculated with the\n");
06269     Vnm_print(2, "Vpmg_qfForce: cubic spline charge discretization \
06270 scheme\n");
06271 }
06272
06273     switch (chgm) {
06274         case VCM_TRIL:
06275             qfForceSpline1(thee, tforce, atomID);
06276             break;
06277         case VCM_BSPL2:
06278             qfForceSpline2(thee, tforce, atomID);
06279             break;
06280     case VCM_BSPL4:
06281         qfForceSpline4(thee, tforce, atomID);
06282         break;
06283     default:
06284         Vnm_print(2, "Vpmg_qfForce: Undefined charge discretization \
06285 method (%d)\n", chgm);
06286         Vnm_print(2, "Vpmg_qfForce: Forces not calculated!\n");
06287         return 0;
06288     }
06289
06290 /* Assign forces */
06291 force[0] = tforce[0];
06292 force[1] = tforce[1];
06293 force[2] = tforce[2];
06294
06295 return 1;
06296 }
06297
06298
06299 VPRIIVATE void qfForceSpline1(Vpmg *thee, double *force, int
06300 atomID) {
06301     Vatom *atom;
06302
06303     double *apos, position[3], hx, hy, hzed;
06304     double xmin, ymin, zmin, xmax, ymax, zmax;
06305     double dx, dy, dz;
06306     double *u, charge, ifloat, jfloat, kfloat;
06307     int nx, ny, nz, ihi, ilo, jhi, jlo, khi, klo;
06308
06309     VASSERT(thee != VNULL);
06310
06311     atom = Valist_getAtom(thee->pbe->alist, atomID);
06312     apos = VatomGetPosition(atom);
06313     charge = Vatom_getCharge(atom);
06314
06315 /* Reset force */
06316     force[0] = 0.0;
06317     force[1] = 0.0;
06318     force[2] = 0.0;
06319
06320 /* If we aren't in the current position, then we're done */
06321 if (atom->partID == 0) return;
06322
06323 /* Mesh info */
06324     nx = thee->pmgp->nx;
06325     ny = thee->pmgp->ny;
06326     nz = thee->pmgp->nz;
06327     hx = thee->pmgp->hx;
06328     hy = thee->pmgp->hy;
06329     hzed = thee->pmgp->hzed;
06330     xmin = thee->pmgp->xmin;
06331     ymin = thee->pmgp->ymin;
06332     zmin = thee->pmgp->zmin;
06333     xmax = thee->pmgp->xmax;
06334     ymax = thee->pmgp->ymax;
06335     zmax = thee->pmgp->zmax;
06336     u = thee->u;
06337
06338 /* Make sure we're on the grid */
06339 if ((apos[0]<=xmin) || (apos[0]>=xmax) || (apos[1]<=ymin) || \
06340     (apos[1]>=ymax) || (apos[2]<=zmin) || (apos[2]>=zmax)) {
06341     if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
06342         (thee->pmgp->bcfl != BCFL_MAP)) {
06343         Vnm_print(2, "Vpmg_qfForce: Atom #%d at (%4.3f, %4.3f, %4.3f) is
06344 off the mesh (ignoring):\n", atomID, apos[0], apos[1], apos[2]);
06345         Vnm_print(2, "Vpmg_qfForce: xmin = %g, xmax = %g\n", xmin, xmax)
06346     };
}

```

```

06345      Vnm_print(2, "Vpmg_qfForce:    ymin = %g, ymax = %g\n", ymin, ymax)
06346      Vnm_print(2, "Vpmg_qfForce:    zmin = %g, zmax = %g\n", zmin, zmax)
06347  }
06348  fflush(stderr);
06349 } else {
06350
06351 /* Convert the atom position to grid coordinates */
06352 position[0] = apos[0] - xmin;
06353 position[1] = apos[1] - ymin;
06354 position[2] = apos[2] - zmin;
06355 ifloat = position[0]/hx;
06356 jfloat = position[1]/hy;
06357 kfloat = position[2]/hzed;
06358 ihi = (int)ceil(ifloat);
06359 ilo = (int)floor(ifloat);
06360 jhi = (int)ceil(jfloat);
06361 jlo = (int)floor(jfloat);
06362 khi = (int)ceil(kfloat);
06363 klo = (int)floor(kfloat);
06364 VASSERT((ihi < nx) && (ihi >=0));
06365 VASSERT((ilo < nx) && (ilo >=0));
06366 VASSERT((jhi < ny) && (jhi >=0));
06367 VASSERT((jlo < ny) && (jlo >=0));
06368 VASSERT((khi < nz) && (khi >=0));
06369 VASSERT((klo < nz) && (klo >=0));
06370 dx = ifloat - (double)(ilo);
06371 dy = jfloat - (double)(jlo);
06372 dz = kfloat - (double)(klo);
06373
06374 #if 0
06375     Vnm_print(1, "Vpmg_qfForce: (DEBUG) u ~ %g\n",
06376                 dx *dy *dz *u[IJK(ihi,jhi,khi)]
06377                 +dx *dy *(1-dz)*u[IJK(ihi,jhi,klo)]
06378                 +dx *(1-dy)*dz *u[IJK(ihi,jlo,khi)]
06379                 +dx *(1-dy)*(1-dz)*u[IJK(ihi,jlo,klo)]
06380                 +(1-dx)*dy *dz *u[IJK(ilo,jhi,khi)]
06381                 +(1-dx)*dy *(1-dz)*u[IJK(ilo,jhi,klo)]
06382                 +(1-dx)*(1-dy)*dz *u[IJK(ilo,jlo,khi)]
06383                 +(1-dx)*(1-dy)*(1-dz)*u[IJK(ilo,jlo,klo)]);
06384 #endif
06385
06386
06387
06388     if ((dx > VPMGSMALL) && (VABS(1.0-dx) > VPMGSMALL)) {
06389         force[0] =
06390             -charge*(dy *dz *u[IJK(ihi,jhi,khi)]
06391                     + dy *(1-dz)*u[IJK(ihi,jhi,klo)]
06392                     + (1-dy)*dz *u[IJK(ihi,jlo,khi)]
06393                     + (1-dy)*(1-dz)*u[IJK(ihi,jlo,klo)]
06394                     - dy *dz *u[IJK(ilo,jhi,khi)]
06395                     - dy *(1-dz)*u[IJK(ilo,jhi,klo)]
06396                     - (1-dy)*dz *u[IJK(ilo,jlo,khi)]
06397                     - (1-dy)*(1-dz)*u[IJK(ilo,jlo,klo)])/hx;
06398     } else {
06399         force[0] = 0;
06400         Vnm_print(0,
06401             "Vpmg_qfForce: Atom %d on x gridline; zero x-force\n", atomID);
06402     }
06403     if ((dy > VPMGSMALL) && (VABS(1.0-dy) > VPMGSMALL)) {
06404         force[1] =
06405             -charge*(dx *dz *u[IJK(ihi,jhi,khi)]
06406                     + dx *(1-dz)*u[IJK(ihi,jhi,klo)]
06407                     - dx *dz *u[IJK(ihi,jlo,khi)]
06408                     - dx *(1-dz)*u[IJK(ihi,jlo,klo)]
06409                     + (1-dx)*dz *u[IJK(ilo,jhi,khi)]
06410                     + (1-dx)*(1-dz)*u[IJK(ilo,jhi,klo)]
06411                     - (1-dx)*dz *u[IJK(ilo,jlo,khi)]
06412                     - (1-dx)*(1-dz)*u[IJK(ilo,jlo,klo)])/hy;
06413     } else {
06414         force[1] = 0;
06415         Vnm_print(0,
06416             "Vpmg_qfForce: Atom %d on y gridline; zero y-force\n", atomID);
06417     }
06418     if ((dz > VPMGSMALL) && (VABS(1.0-dz) > VPMGSMALL)) {
06419         force[2] =
06420             -charge*(dy *dx *u[IJK(ihi,jhi,khi)]
06421                     - dy *dx *u[IJK(ihi,jhi,klo)]
06422                     + (1-dy)*dx *u[IJK(ihi,jlo,khi)]
06423                     - (1-dy)*dx *u[IJK(ihi,jlo,klo)])

```

```

06424             + dy     * (1-dx)*u[IJK(ilo,jhi,khi)]
06425             - dy     * (1-dx)*u[IJK(ilo,jhi,klo)]
06426             + (1-dy)* (1-dx)*u[IJK(ilo,jlo,khi)]
06427             - (1-dy)* (1-dx)*u[IJK(ilo,jlo,klo)]) /hzed;
06428     } else {
06429         force[2] = 0;
06430         Vnm_print(0,
06431             "Vpmg_qfForce: Atom %d on z gridline; zero z-force\n", atomID);
06432     }
06433 }
06434 }
06435
06436 VPRIIVATE void qfForceSpline2(Vpmg *thee, double *force, int
06437     atomID) {
06438
06439     Vatom *atom;
06440
06441     double *apos, position[3], hx, hy, hzed;
06442     double xlen, ylen, zlen, xmin, ymin, zmin, xmax
06443     , ymax, zmax;
06444     double mx, my, mz, dmx, dmy, dmz;
06445     double *u, charge, ifloat, jfloat, kffloat;
06446     int nx, ny, nz, im2, im1, ipl, ip2, jm2, jm1, jp1, jp2, km2, km1;
06447     int kpl, kp2, ii, jj, kk;
06448
06449     VASSERT(thee != VNULL);
06450
06451     atom = Valist_getAtom(thee->pbe->alist, atomID);
06452     apos = Vatom_getPosition(atom);
06453     charge = Vatom_getCharge(atom);
06454
06455     /* Reset force */
06456     force[0] = 0.0;
06457     force[1] = 0.0;
06458     force[2] = 0.0;
06459
06460     /* If we aren't in the current position, then we're done */
06461     if (atom->partID == 0) return;
06462
06463     /* Mesh info */
06464     nx = thee->pmgp->nx;
06465     ny = thee->pmgp->ny;
06466     nz = thee->pmgp->nz;
06467     hx = thee->pmgp->hx;
06468     hy = thee->pmgp->hy;
06469     hzed = thee->pmgp->hzed;
06470     xlen = thee->pmgp->xlen;
06471     ylen = thee->pmgp->ylen;
06472     zlen = thee->pmgp->zlen;
06473     xmin = thee->pmgp->xmin;
06474     ymin = thee->pmgp->ymin;
06475     zmin = thee->pmgp->zmin;
06476     xmax = thee->pmgp->xmax;
06477     ymax = thee->pmgp->ymax;
06478     zmax = thee->pmgp->zmax;
06479     u = thee->u;
06480
06481     /* Make sure we're on the grid */
06482     if ((apos[0]<=(xmin+hx)) || (apos[0]>=(xmax-hx)) \
06483         || (apos[1]<=(ymin+hy)) || (apos[1]>=(ymax-hy)) \
06484         || (apos[2]<=(zmin+hzed)) || (apos[2]>=(zmax-hzed))) {
06485         if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
06486             (thee->pmgp->bcfl != BCFL_MAP)) {
06487             Vnm_print(2, "qfForceSpline2: Atom #%d off the mesh \
06488             (ignoring)\n", atomID);
06489             fflush(stderr);
06490     } else {
06491
06492         /* Convert the atom position to grid coordinates */
06493         position[0] = apos[0] - xmin;
06494         position[1] = apos[1] - ymin;
06495         position[2] = apos[2] - zmin;
06496         ifloat = position[0]/hx;
06497         jfloat = position[1]/hy;
06498         kffloat = position[2]/hzed;
06499         ipl = (int)ceil(ifloat);
06500         ip2 = ipl + 1;
06501         im1 = (int)floor(ifloat);
06502         im2 = im1 - 1;

```

```

06503     jp1 = (int)ceil(jffloat);
06504     jp2 = jp1 + 1;
06505     jml = (int)floor(jffloat);
06506     jm2 = jml - 1;
06507     kp1 = (int)ceil(kffloat);
06508     kp2 = kp1 + 1;
06509     kml = (int)floor(kffloat);
06510     km2 = kml - 1;
06511
06512     /* This step shouldn't be necessary, but it saves nasty debugging
06513      * later on if something goes wrong */
06514     ip2 = VMIN2(ip1,nx-1);
06515     ip1 = VMIN2(ip1,nx-1);
06516     im1 = VMAX2(im1,0);
06517     im2 = VMAX2(im2,0);
06518     jp2 = VMIN2(jp2,ny-1);
06519     jp1 = VMIN2(jp1,ny-1);
06520     jm1 = VMAX2(jm1,0);
06521     jm2 = VMAX2(jm2,0);
06522     kp2 = VMIN2(kp2,nz-1);
06523     kp1 = VMIN2(kp1,nz-1);
06524     kml = VMAX2(kml,0);
06525     km2 = VMAX2(km2,0);
06526
06527
06528     for (ii=im2; ii<=ip2; ii++) {
06529         mx = bspline2(VFCHI(ii,ifloat));
06530         dmx = dbspline2(VFCHI(ii,ifloat));
06531         for (jj=jm2; jj<=jp2; jj++) {
06532             my = bspline2(VFCHI(jj,jffloat));
06533             dmy = dbspline2(VFCHI(jj,jffloat));
06534             for (kk=km2; kk<=kp2; kk++) {
06535                 mz = bspline2(VFCHI(kk,kffloat));
06536                 dmz = dbspline2(VFCHI(kk,kffloat));
06537
06538                 force[0] += (charge*dmx*my*mz*u[IJK(ii,jj,kk)])/hx;
06539                 force[1] += (charge*mx*dmy*mz*u[IJK(ii,jj,kk)])/hy;
06540                 force[2] += (charge*mx*my*dmz*u[IJK(ii,jj,kk)])/hzd;
06541
06542             }
06543         }
06544     }
06545
06546 }
06547 }
06548
06549 VPRIVATE void qfForceSpline4(Vpmg *thee, double *force, int
atomID) {
06550
06551     Vatom *atom;
06552     double f, c, *u, *apos, position[3];
06553
06554     /* Grid variables */
06555     int nx,ny,nz;
06556     double xlen, ylen, zlen, xmin, ymin, zmin, xmax
, ymax, zmax;
06557     double hx, hy, hzed, ifloat, jffloat, kffloat;
06558
06559     /* B-spline weights */
06560     double mx, my, mz, dmx, dmy, dmz;
06561     double mi, mj, mk;
06562
06563     /* Loop indeces */
06564     int i, j, k, ii, jj, kk;
06565     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, kml, kp1, kp2;
06566
06567     /* field */
06568     double e[3];
06569
06570     VASSERT(thee != VNULL);
06571     VASSERT(thee->filled);
06572
06573     atom = Valist_getAtom(thee->pbe->alist, atomID);
06574     apos = Vatom_getPosition(atom);
06575     c = Vatom_getCharge(atom);
06576
06577     for (i=0;i<3;i++) {
06578         e[i] = 0.0;
06579     }
06580
06581     /* Mesh info */

```

```

06582     nx = thee->pmgp->nx;
06583     ny = thee->pmgp->ny;
06584     nz = thee->pmgp->nz;
06585     hx = thee->pmgp->hx;
06586     hy = thee->pmgp->hy;
06587     hzed = thee->pmgp->hzed;
06588     xlen = thee->pmgp->xlen;
06589     ylen = thee->pmgp->ylen;
06590     zlen = thee->pmgp->zlen;
06591     xmin = thee->pmgp->xmin;
06592     ymin = thee->pmgp->ymin;
06593     zmin = thee->pmgp->zmin;
06594     xmax = thee->pmgp->xmax;
06595     ymax = thee->pmgp->ymax;
06596     zmax = thee->pmgp->zmax;
06597     u = thee->u;
06598
06599     /* Make sure we're on the grid */
06600     if ((apos[0]<=(xmin+2*hx)) || (apos[0]>=(xmax-2*hx)) \
06601     || (apos[1]<=(ymin+2*hy)) || (apos[1]>=(ymax-2*hy)) \
06602     || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
06603         Vnm_print(2, "qfForceSpline4: Atom off the mesh \
06604 (ignoring) %6.3f %6.3f %6.3f\n", apos[0], apos[1], apos[2]);
06605         fflush(stderr);
06606     } else {
06607
06608         /* Convert the atom position to grid coordinates */
06609         position[0] = apos[0] - xmin;
06610         position[1] = apos[1] - ymin;
06611         position[2] = apos[2] - zmin;
06612         ifloat = position[0]/hx;
06613         jfloat = position[1]/hy;
06614         kfloat = position[2]/hzed;
06615         ip1 = (int)ceil(ifloat);
06616         ip2 = ip1 + 2;
06617         im1 = (int)floor(ifloat);
06618         im2 = im1 - 2;
06619         jp1 = (int)ceil(jfloat);
06620         jp2 = jp1 + 2;
06621         jm1 = (int)floor(jfloat);
06622         jm2 = jm1 - 2;
06623         kp1 = (int)ceil(kfloat);
06624         kp2 = kp1 + 2;
06625         km1 = (int)floor(kfloat);
06626         km2 = km1 - 2;
06627
06628         /* This step shouldn't be necessary, but it saves nasty debugging
06629 * later on if something goes wrong */
06630         ip2 = VMIN2(ip2,nx-1);
06631         ip1 = VMIN2(ip1,nx-1);
06632         im1 = VMAX2(im1,0);
06633         im2 = VMAX2(im2,0);
06634         jp2 = VMIN2(jp2,ny-1);
06635         jp1 = VMIN2(jp1,ny-1);
06636         jm1 = VMAX2(jm1,0);
06637         jm2 = VMAX2(jm2,0);
06638         kp2 = VMIN2(kp2,nz-1);
06639         kp1 = VMIN2(kp1,nz-1);
06640         km1 = VMAX2(km1,0);
06641         km2 = VMAX2(km2,0);
06642
06643         for (ii=im2; ii<=ip2; ii++) {
06644             mi = VFCHI4(ii,ifloat);
06645             mx = bspline4(mi);
06646             dmx = dbspline4(mi);
06647             for (jj=jm2; jj<=jp2; jj++) {
06648                 mj = VFCHI4(jj,jfloat);
06649                 my = bspline4(mj);
06650                 dmy = dbspline4(mj);
06651                 for (kk=km2; kk<=kp2; kk++) {
06652                     mk = VFCHI4(kk,kfloat);
06653                     mz = bspline4(mk);
06654                     dmz = dbspline4(mk);
06655                     f = u[IJK(ii,jj,kk)];
06656                     /* Field */
06657                     e[0] += f*dmx*my*mz/hx;
06658                     e[1] += f*mx*dmy*mz/hy;
06659                     e[2] += f*mx*my*dmz/hzed;
06660                 }
06661             }
06662         }

```

```

06663      }
06664
06665  /* Monopole Force */
06666  force[0] = e[0]*c;
06667  force[1] = e[1]*c;
06668  force[2] = e[2]*c;
06669
06670 }
06671
06672 VPRIIVATE void markFrac(
06673     double rtot, double *tpos,
06674     int nx, int ny, int nz,
06675     double hx, double hy, double hzed,
06676     double xmin, double ymin, double zmin,
06677     double *xarray, double *yarray, double *zarray) {
06678
06679     int i, j, k, imin, imax, jmin, jmax, kmin, kmax;
06680     double dx, dx2, dy, dy2, dz, dz2, a000, a001, a100, a101, r2;
06681     double x, xp, xm, y, yp, ym, zp, z, zm, xspan, yspan, zspan;
06682     double rtot2, pos[3];
06683
06684  /* Convert to grid reference frame */
06685  pos[0] = tpos[0] - xmin;
06686  pos[1] = tpos[1] - ymin;
06687  pos[2] = tpos[2] - zmin;
06688
06689  rtot2 = VSQR(rtot);
06690
06691  xspan = rtot + 2*hx;
06692  imin = VMAX2(0, (int)ceil((pos[0] - xspan)/hx));
06693  imax = VMIN2(nx-1, (int)floor((pos[0] + xspan)/hx));
06694  for (i=imin; i<=imax; i++) {
06695      x = hx*i;
06696      dx2 = VSQR(pos[0] - x);
06697      if (rtot2 > dx2) {
06698          yspan = VSQRT(rtot2 - dx2) + 2*hy;
06699      } else {
06700          yspan = 2*hy;
06701      }
06702      jmin = VMAX2(0,(int)ceil((pos[1] - yspan)/hy));
06703      jmax = VMIN2(ny-1,(int)floor((pos[1] + yspan)/hy));
06704      for (j=jmin; j<=jmax; j++) {
06705          y = hy*j;
06706          dy2 = VSQR(pos[1] - y);
06707          if (rtot2 > (dx2+dy2)) {
06708              zspan = VSQRT(rtot2-dx2-dy2) + 2*hzed;
06709          } else {
06710              zspan = 2*hzed;
06711          }
06712          kmin = VMAX2(0,(int)ceil((pos[2] - zspan)/hzed));
06713          kmax = VMIN2(nz-1,(int)floor((pos[2] + zspan)/hzed));
06714          for (k=kmin; k<=kmax; k++) {
06715              z = hzed*k;
06716              dz2 = VSQR(pos[2] - z);
06717
06718              r2 = dx2 + dy2 + dz2;
06719
06720  /* We need to determine the inclusion value a000 at (i,j,k) */
06721  if (r2 < rtot2) a000 = 1.0;
06722  else a000 = 0.0;
06723
06724  /* We need to evaluate the values of x which intersect the
06725   * sphere and determine if these are in the interval
06726   * [(i,j,k), (i+1,j,k)] */
06727  if (r2 < (rtot2 - hx*hx)) a100 = 1.0;
06728  else if (r2 > (rtot2 + hx*hx)) a100 = 0.0;
06729  else if (rtot2 > (dy2 + dz2)) {
06730      dx = VSQRT(rtot2 - dy2 - dz2);
06731      xm = pos[0] - dx;
06732      xp = pos[0] + dx;
06733      if ((xm < x+hx) && (xm > x)) {
06734          a100 = (xm - x)/hx;
06735      } else if ((xp < x+hx) && (xp > x)) {
06736          a100 = (xp - x)/hx;
06737      }
06738  } else a100 = 0.0;
06739
06740  /* We need to evaluate the values of y which intersect the
06741   * sphere and determine if these are in the interval
06742   * [(i,j,k), (i,j+1,k)] */
06743  if (r2 < (rtot2 - hy*hy)) a010 = 1.0;

```

```

06744         else if (r2 > (rtot2 + hy*hy)) a010 = 0.0;
06745         else if (rtot2 > (dx2 + dz2)) {
06746             dy = VSQRT(rtot2 - dx2 - dz2);
06747             ym = pos[1] - dy;
06748             yp = pos[1] + dy;
06749             if ((ym < y+hy) && (ym > y)) {
06750                 a010 = (ym - y)/hy;
06751             } else if ((yp < y+hy) && (yp > y)) {
06752                 a010 = (yp - y)/hy;
06753             }
06754         } else a010 = 0.0;
06755
06756     /* We need to evaluate the values of y which intersect the
06757     * sphere and determine if these are in the interval
06758     * [(i,j,k), (i,j,k+1)] */
06759     if (r2 < (rtot2 - hzed*hzed)) a001 = 1.0;
06760     else if (r2 > (rtot2 + hzed*hzed)) a001 = 0.0;
06761     else if (rtot2 > (dx2 + dy2)) {
06762         dz = VSQRT(rtot2 - dx2 - dy2);
06763         zm = pos[2] - dz;
06764         zp = pos[2] + dz;
06765         if ((zm < z+hzed) && (zm > z)) {
06766             a001 = (zm - z)/hzed;
06767         } else if ((zp < z+hzed) && (zp > z)) {
06768             a001 = (zp - z)/hzed;
06769         }
06770     } else a001 = 0.0;
06771
06772     if (a100 < xarray[IJK(i,j,k)]) xarray[IJK(i,j,k)] = a100;
06773     if (a010 < yarray[IJK(i,j,k)]) yarray[IJK(i,j,k)] = a010;
06774     if (a001 < zarray[IJK(i,j,k)]) zarray[IJK(i,j,k)] = a001;
06775
06776     } /* k loop */
06777     } /* j loop */
06778 } /* i loop */
06779 }
0680
0681 /*
0682
0683 NOTE: This is the original version of the markSphere function. It's in here
0684 for reference and in case a reversion to the original code is needed.
0685 D. Gohara (2/14/08)
0686 */
0687 /*
0688 VPRIVATE void markSphere(
0689     double rtot, double *tpos,
0690     int nx, int ny, int nz,
0691     double hx, double hy, double hzed,
0692     double xmin, double ymin, double zmin,
0693     double *array, double markVal) {
0694
0695     int i, j, k, imin, imax, jmin, jmax, kmin, kmax;
0696     double dx, dx2, dy, dy2, dz, dz2;
0697     double rtot2, pos[3];
0698
0699     // Convert to grid reference frame
0700     pos[0] = tpos[0] - xmin;
0701     pos[1] = tpos[1] - ymin;
0702     pos[2] = tpos[2] - zmin;
0703
0704     rtot2 = VSQR(rtot);
0705
0706     dx = rtot + 0.5*hx;
0707     imin = VMAX2(0, (int)ceil((pos[0] - dx)/hx));
0708     imax = VMIN2(nx-1, (int)floor((pos[0] + dx)/hx));
0709     for (i=imin; i<=imax; i++) {
0710         dx2 = VSQR(pos[0] - hx*i);
0711         if (rtot2 > dx2) {
0712             dy = VSQRT(rtot2 - dx2) + 0.5*hy;
0713         } else {
0714             dy = 0.5*hy;
0715         }
0716         jmin = VMAX2(0, (int)ceil((pos[1] - dy)/hy));
0717         jmax = VMIN2(ny-1, (int)floor((pos[1] + dy)/hy));
0718         for (j=jmin; j<=jmax; j++) {
0719             dy2 = VSQR(pos[1] - hy*j);
0720             if (rtot2 > (dx2+dy2)) {
0721                 dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
0722             } else {
0723                 dz = 0.5*hzed;
0724             }

```

```

06825     kmin = VMAX2(0,(int)ceil((pos[2] - dz)/hzed));
06826     kmax = VMIN2(nz-1,(int)floor((pos[2] + dz)/hzed));
06827     for (k=kmin; k<=kmax; k++) {
06828         dz2 = VSQR(k*hzed - pos[2]);
06829         if ((dz2 + dy2 + dx2) <= rtot2) {
06830             array[IJK(i,j,k)] = markVal;
06831         }
06832     } // k loop
06833 } // j loop
06834 } // i loop
06835 }
06836 */
06837 VPRIIVATE void markSphere(double rtot, double *tpos,
06838     int nx, int ny, int nz,
06839     double hx, double hy, double hz,
06840     double xmin, double ymin, double zmin,
06841     double *array, double markVal) {
06842
06843     int i, j, k;
06844     double fi,fj,fk;
06845     int imin, imax;
06846     int jmin, jmax;
06847     int kmin, kmax;
06848     double dx2, dy2, dz2;
06849     double xrange, yrange, zrange;
06850     double rtot2, posx, posy, posz;
06851
06852     /* Convert to grid reference frame */
06853     posx = tpos[0] - xmin;
06854     posy = tpos[1] - ymin;
06855     posz = tpos[2] - zmin;
06856
06857     rtot2 = VSQR(rtot);
06858
06859     xrange = rtot + 0.5 * hx;
06860     yrange = rtot + 0.5 * hy;
06861     zrange = rtot + 0.5 * hz;
06862
06863     imin = VMAX2(0, (int)ceil((posx - xrange)/hx));
06864     jmin = VMAX2(0, (int)ceil((posy - yrange)/hy));
06865     kmin = VMAX2(0, (int)ceil((posz - zrange)/hz));
06866
06867     imax = VMIN2(nx-1, (int)floor((posx + xrange)/hx));
06868     jmax = VMIN2(ny-1, (int)floor((posy + yrange)/hy));
06869     kmax = VMIN2(nz-1, (int)floor((posz + zrange)/hz));
06870
06871     for (i=imin,fi=imin; i<=imax; i++, fi+=1.) {
06872         dx2 = VSQR(posx - hx*fi);
06873         for (j=jmin,fj=jmin; j<=jmax; j++, fj+=1.) {
06874             dy2 = VSQR(posy - hy*fj);
06875             if((dx2 + dy2) > rtot2) continue;
06876             for (k=kmin, fk=kmin; k<=kmax; k++, fk+=1.) {
06877                 dz2 = VSQR(posz - hz*fk);
06878                 if ((dz2 + dy2 + dx2) <= rtot2) {
06879                     array[IJK(i,j,k)] = markVal;
06880                 }
06881             }
06882         }
06883     }
06884 }
06885
06886 VPRIIVATE void zlapSolve(
06887     Vpmg *thee,
06888     double **solution,
06889     double **source,
06890     double **work1
06891     ) {
06892
06893     /* NOTE: this is an incredibly inefficient algorithm. The next
06894     * improvement is to focus on only non-zero entries in the source term.
06895     * The best improvement is to use a fast sine transform */
06896
06897     int n, nx, ny, nz, i, j, k, kx, ky, kz;
06898     double hx, hy, hzed, wx, wy, wz, xlen, ylen, zlen;
06899     double phix, phixp1, phixm1, phiy, phiyml, phiypl, phiz, phizml, phizp1;
06900     double norm, coef, proj, eigx, eigy, eigz;
06901     double ihx2, ihy2, ihzed2;
06902     double *u, *f, *phi;
06903
06904     /* Snarf grid parameters */
06905     nx = thee->pmgp->nx;

```

```

06906     ny = theee->pmgp->ny;
06907     nz = theee->pmgp->nz;
06908     n = nx*ny*nz;
06909     hx = theee->pmgp->hx;
06910     ihx2 = 1.0/hx/hx;
06911     hy = theee->pmgp->hy;
06912     ihy2 = 1.0/hy/hy;
06913     hzed = theee->pmgp->hzed;
06914     ihzed2 = 1.0/hzed/hzed;
06915     xlen = theee->pmgp->xlen;
06916     ylen = theee->pmgp->ylen;
06917     zlen = theee->pmgp->zlen;
06918
06919     /* Set solution and source array pointers */
06920     u = *solution;
06921     f = *source;
06922     phi = *work1;
06923
06924     /* Zero out the solution vector */
06925     for (i=0; i<n; i++) theee->u[i] = 0.0;
06926
06927     /* Iterate through the wavenumbers */
06928     for (kx=1; kx<(nx-1); kx++) {
06929
06930         wx = (VPI*(double)kx)/((double)nx - 1.0);
06931         eigx = 2.0*ihx2*(1.0 - cos(wx));
06932
06933         for (ky=1; ky<(ny-1); ky++) {
06934
06935             wy = (VPI*(double)ky)/((double)ny - 1.0);
06936             eigy = 2.0*ihy2*(1.0 - cos(wy));
06937
06938             for (kz=1; kz<(nz-1); kz++) {
06939
06940                 wz = (VPI*(double)kz)/((double)nz - 1.0);
06941                 eigz = 2.0*ihzed2*(1.0 - cos(wz));
06942
06943                 /* Calculate the basis function.
06944                  * We could calculate each basis function as
06945                  *   phix(i) = sin(wx*i)
06946                  *   phiy(j) = sin(wy*j)
06947                  *   phiz(k) = sin(wz*k)
06948                  * However, this is likely to be very expensive.
06949                  * Therefore, we can use the fact that
06950                  *   phix(i+1) = (2.0-hx*hx*eigx)*phix(i) - phix(i-1)
06951                  * */
06952                 for (i=1; i<(nx-1); i++) {
06953                     if (i == 1) {
06954                         phix = sin(wx*(double)i);
06955                         phixml = 0.0;
06956                     } else {
06957                         phixp1 = (2.0-hx*hx*eigx)*phix - phixml;
06958                         phixml = phix;
06959                         phix = phixp1;
06960                     }
06961                     /* phix = sin(wx*(double)i); */
06962                     for (j=1; j<(ny-1); j++) {
06963                         if (j == 1) {
06964                             phiy = sin(wy*(double)j);
06965                             phiyml = 0.0;
06966                         } else {
06967                             phiypl = (2.0-hy*hy*eigy)*phiy - phiyml;
06968                             phiyml = phiy;
06969                             phiy = phiypl;
06970                         }
06971                         /* phiy = sin(wy*(double)j); */
06972                         for (k=1; k<(nz-1); k++) {
06973                             if (k == 1) {
06974                                 phiz = sin(wz*(double)k);
06975                                 phizml = 0.0;
06976                             } else {
06977                                 phizp1 = (2.0-hzed*hzed*eigz)*phiz - phizml;
06978                                 phizml = phiz;
06979                                 phiz = phizp1;
06980                             }
06981                             /* phiz = sin(wz*(double)k); */
06982                             phi[IJK(i,j,k)] = phix*phiy*phiz;
06983
06984                         }
06985                     }
06986                 }
06987             }
06988         }
06989     }

```

```

06987             }
06988
06989     /* Calculate the projection of the source function on this
06990     * basis function */
06991     proj = 0.0;
06992     for (i=1; i<(nx-1); i++) {
06993         for (j=1; j<(ny-1); j++) {
06994             for (k=1; k<(nz-1); k++) {
06995
06996                 proj += f[IJK(i,j,k)]*phi[IJK(i,j,k)];
06997
06998             } /* k loop */
06999         } /* j loop */
07000     } /* i loop */
07001
07002     /* Assemble the coefficient to weight the contribution of this
07003     * basis function to the solution */
07004     /* The first contribution is the projection */
07005     coef = proj;
07006     /* The second contribution is the eigenvalue */
07007     coef = coef/(eigx + eigy + eigz);
07008     /* The third contribution is the normalization factor */
07009     coef = (8.0*xlen/ylen/zlen)*coef;
07010     /* The fourth contribution is from scaling the diagonal */
07011     /* coef = hx*hy*hzed*coef; */
07012
07013     /* Evaluate the basis function at each grid point */
07014     for (i=1; i<(nx-1); i++) {
07015         for (j=1; j<(ny-1); j++) {
07016             for (k=1; k<(nz-1); k++) {
07017
07018                 u[IJK(i,j,k)] += coef*phi[IJK(i,j,k)];
07019
07020             } /* k loop */
07021         } /* j loop */
07022     } /* i loop */
07023
07024     } /* kz loop */
07025 } /* ky loop */
07026 } /* kx loop */
07027
07028 }
07029
07030 VPUBLIC int Vpmg_solveLaplace(Vpmg *thee) {
07031
07032     int i, j, k, ijk, nx, ny, nz, n, dilo, dihi, djlo, djhi, dklo, dkhi;
07033     double hx, hy, hzed, epsw, iepsw, scal, scalx, scaly, scalz;
07034
07035     nx = thee->pmgp->nx;
07036     ny = thee->pmgp->ny;
07037     nz = thee->pmgp->nz;
07038     n = nx*ny*nz;
07039     hx = thee->pmgp->hx;
07040     hy = thee->pmgp->hy;
07041     hzed = thee->pmgp->hzed;
07042     epsw = Vpbe_getSolventDielectric(thee->pbe);
07043     iepsw = 1.0/epsw;
07044     scal = hx*hy*hzed;
07045     scalx = hx*hy/hzed;
07046     scaly = hx*hzed/hy;
07047     scalz = hx*hy/hzed;
07048
07049     if (! (thee->filled)) {
07050         Vnm_print(2, "Vpmg_solve: Need to call Vpmg_fillco()!\n");
07051         return 0;
07052     }
07053
07054     /* Load boundary conditions into the RHS array */
07055     for (i=1; i<(nx-1); i++) {
07056
07057         if (i == 1) dilo = 1;
07058         else dilo = 0;
07059         if (i == nx-2) dihi = 1;
07060         else dihi = 0;
07061
07062         for (j=1; j<(ny-1); j++) {
07063
07064             if (j == 1) djlo = 1;
07065             else djlo = 0;
07066             if (j == ny-2) djhi = 1;
07067             else djhi = 0;

```

```

07068
07069     for (k=1; k<(nz-1); k++) {
07070
07071         if (k == 1) dklo = 1;
07072         else dklo = 0;
07073         if (k == nz-2) dkhi = 1;
07074         else dkhi = 0;
07075
07076         thee->fcf[IJK(i,j,k)] = \
07077             iepsw*scalx*thee->charge[IJK(i,j,k)] \
07078             + dilo*scalx*thee->gxfc[IJKx(j,k,0)] \
07079             + dihi*scalx*thee->gxfc[IJKx(j,k,1)] \
07080             + djlo*scaly*thee->gycf[IJKy(i,k,0)] \
07081             + djhi*scaly*thee->gycf[IJKy(i,k,1)] \
07082             + dklo*scalz*thee->gzcf[IJKz(i,j,0)] \
07083             + dkhi*scalz*thee->gzcf[IJKz(i,j,1)] ;
07084
07085     }
07086 }
07087 }
07088 }
07089
07090 /* Solve */
07091 zlapsolve( thee, &(thee->u), &(thee->fcf), &(thee->tcf) );
07092
07093 /* Add boundary conditions to solution */
07094 /* i faces */
07095 for (j=0; j<ny; j++) {
07096     for (k=0; k<nz; k++) {
07097         thee->u[IJK(0,j,k)] = thee->gxfc[IJKx(j,k,0)];
07098         thee->u[IJK(nx-1,j,k)] = thee->gycf[IJKx(j,k,1)];
07099     }
07100 }
07101 /* j faces */
07102 for (i=0; i<nx; i++) {
07103     for (k=0; k<nz; k++) {
07104         thee->u[IJK(i,0,k)] = thee->gycf[IJKy(i,k,0)];
07105         thee->u[IJK(i,ny-1,k)] = thee->gycf[IJKy(i,k,1)];
07106     }
07107 }
07108 /* k faces */
07109 for (i=0; i<nx; i++) {
07110     for (j=0; j<ny; j++) {
07111         thee->u[IJK(i,j,0)] = thee->gzcf[IJKz(i,j,0)];
07112         thee->u[IJK(i,j,nz-1)] = thee->gzcf[IJKz(i,j,1)];
07113     }
07114 }
07115
07116     return 1;
07117
07118 }
07119
07120 VPRIPRIVATE double VFCHI4(int i, double f) {
07121     return (2.5+((double)(i)-(f)));
07122 }
07123
07124 VPRIPRIVATE double bspline4(double x) {
07125
07126     double m, m2;
07127     static double one6 = 1.0/6.0;
07128     static double one8 = 1.0/8.0;
07129     static double one24 = 1.0/24.0;
07130     static double thirteen24 = 13.0/24.0;
07131     static double fourtyseven24 = 47.0/24.0;
07132     static double seventeen24 = 17.0/24.0;
07133
07134     if ((x > 0.0) && (x <= 1.0)){
07135         m = x*x;
07136         return one24*m*m;
07137     } else if ((x > 1.0) && (x <= 2.0)){
07138         m = x - 1.0;
07139         m2 = m*m;
07140         return -one8 + one6*x + m2*(0.25 + one6*m - one6*m2);
07141     } else if ((x > 2.0) && (x <= 3.0)){
07142         m = x - 2.0;
07143         m2 = m*m;
07144         return -thirteen24 + 0.5*x + m2*(-0.25 - 0.5*m + 0.25*m2);
07145     } else if ((x > 3.0) && (x <= 4.0)){
07146         m = x - 3.0;
07147         m2 = m*m;
07148         return fourtyseven24 - 0.5*x + m2*(-0.25 + 0.5*m - one6*m2);
07149     } else if ((x > 4.0) && (x <= 5.0)){

```

```

07150     m = x - 4.0;
07151     m2 = m*m;
07152     return seventeen24 - one6*x + m2*(0.25 - one6*m + one24*m2);
07153 } else {
07154     return 0.0;
07155 }
07156 }
07157
07158 VPUBLIC double dbspline4(double x) {
07159
07160     double m, m2;
07161     static double one6 = 1.0/6.0;
07162     static double one3 = 1.0/3.0;
07163     static double two3 = 2.0/3.0;
07164     static double thirteen6 = 13.0/6.0;
07165
07166     if ((x > 0.0) && (x <= 1.0)){
07167         m2 = x*x;
07168         return one6*x*m2;
07169     } else if ((x > 1.0) && (x <= 2.0)){
07170         m = x - 1.0;
07171         m2 = m*m;
07172         return -one3 + 0.5*x + m2*(0.5 - two3*m);
07173     } else if ((x > 2.0) && (x <= 3.0)){
07174         m = x - 2.0;
07175         m2 = m*m;
07176         return 1.5 - 0.5*x + m2*(-1.5 + m);
07177     } else if ((x > 3.0) && (x <= 4.0)){
07178         m = x - 3.0;
07179         m2 = m*m;
07180         return 1.0 - 0.5*x + m2*(1.5 - two3*m);
07181     } else if ((x > 4.0) && (x <= 5.0)){
07182         m = x - 4.0;
07183         m2 = m*m;
07184         return -thirteen6 + 0.5*x + m2*(-0.5 + one6*m);
07185     } else {
07186         return 0.0;
07187     }
07188 }
07189
07190 VPUBLIC double d2bspline4(double x) {
07191
07192     double m, m2;
07193
07194     if ((x > 0.0) && (x <= 1.0)){
07195         return 0.5*x*x;
07196     } else if ((x > 1.0) && (x <= 2.0)){
07197         m = x - 1.0;
07198         m2 = m*m;
07199         return -0.5 + x - 2.0*m2;
07200     } else if ((x > 2.0) && (x <= 3.0)){
07201         m = x - 2.0;
07202         m2 = m*m;
07203         return 5.5 - 3.0*x + 3.0*m2;
07204     } else if ((x > 3.0) && (x <= 4.0)){
07205         m = x - 3.0;
07206         m2 = m*m;
07207         return -9.5 + 3.0*x - 2.0*m2;
07208     } else if ((x > 4.0) && (x <= 5.0)){
07209         m = x - 4.0;
07210         m2 = m*m;
07211         return 4.5 - x + 0.5*m2;
07212     } else {
07213         return 0.0;
07214     }
07215 }
07216
07217 VPUBLIC double d3bspline4(double x) {
07218
07219     if      ((x > 0.0) && (x <= 1.0)) return x;
07220     else if ((x > 1.0) && (x <= 2.0)) return 5.0 - 4.0 * x;
07221     else if ((x > 2.0) && (x <= 3.0)) return -15.0 + 6.0 * x;
07222     else if ((x > 3.0) && (x <= 4.0)) return 15.0 - 4.0 * x;
07223     else if ((x > 4.0) && (x <= 5.0)) return x - 5.0;
07224     else
07225
07226 }
07227
07228 VPUBLIC void fillcoPermanentMultipole(Vpmg *thee) {
07229
07230     Valist *alist;

```

```

07231     Vpbe *pbe;
07232     Vatom *atom;
07233     /* Coverions */
07234     double zmagic, f;
07235     /* Grid */
07236     double xmin, xmax, ymin, ymax, zmin, zmax;
07237     double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;
07238     double hx, hy, hzed, *apos;
07239     /* Multipole */
07240     double charge, *dipole, *quad;
07241     double c, ux, uy, uz, qx, qy, qz, qxx, qyy, qzx, qzy, qzz, qave;
07242     /* B-spline weights */
07243     double mx, my, mz, dmxx, dmy, dmz, d2mx, d2my, d2mz;
07244     double m1, m2, m3;
07245     /* Loop variables */
07246     int i, ii, jj, kk, nx, ny, nz, iatom;
07247     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
07248
07249     /* sanity check */
07250     double mir, mjr, mkr, mr2;
07251     double debye, mc, mux, muy, muz, mqxx, mqyy, mqzx, mqzy, mqzz;
07252
07253     VASSERT(thee != VNULL);
07254
07255     /* Get PBE info */
07256     pbe = thee->pbe;
07257     alist = pbe->alist;
07258     zmagic = Vpbe_getZmagic(pbe);
07259
07260     /* Mesh info */
07261     nx = thee->pmgp->nx;
07262     ny = thee->pmgp->ny;
07263     nz = thee->pmgp->nz;
07264     hx = thee->pmgp->hx;
07265     hy = thee->pmgp->hy;
07266     hzed = thee->pmgp->hzed;
07267
07268     /* Conversion */
07269     f = zmagic/(hx*hy*hzed);
07270
07271     /* Define the total domain size */
07272     xlen = thee->pmgp->xlen;
07273     ylen = thee->pmgp->ylen;
07274     zlen = thee->pmgp->zlen;
07275
07276     /* Define the min/max dimensions */
07277     xmin = thee->pmgp->xcent - (xlen/2.0);
07278     ymin = thee->pmgp->ycent - (ylen/2.0);
07279     zmin = thee->pmgp->zcent - (zlen/2.0);
07280     xmax = thee->pmgp->xcent + (xlen/2.0);
07281     ymax = thee->pmgp->ycent + (ylen/2.0);
07282     zmax = thee->pmgp->zcent + (zlen/2.0);
07283
07284     /* Fill in the source term (permanent atomic multipoles) */
07285     Vnm_print(0, "fillcoPermanentMultipole: filling in source term.\n");
07286     for (iatom=0; iatom<Valist_getNumberAtoms(alist);
07287         iatom++) {
07288         atom = Valist_getAtom(alist, iatom);
07289         apos = Vatom_getPosition(atom);
07290
07291         c = Vatom_getCharge(atom)*f;
07292
07293 #if defined(WITH_TINKER)
07294         dipole = Vatom_getDipole(atom);
07295         ux = dipole[0]/hx*f;
07296         uy = dipole[1]/hy*f;
07297         uz = dipole[2]/hzed*f;
07298         quad = Vatom_getQuadrupole(atom);
07299         qxx = (1.0/3.0)*quad[0]/(hx*hx)*f;
07300         qyx = (2.0/3.0)*quad[3]/(hx*hy)*f;
07301         qyy = (1.0/3.0)*quad[4]/(hy*hy)*f;
07302         qzx = (2.0/3.0)*quad[6]/(hzed*hx)*f;
07303         qzy = (2.0/3.0)*quad[7]/(hzed*hy)*f;
07304         qzz = (1.0/3.0)*quad[8]/(hzed*hzed)*f;
07305 #else
07306         ux = 0.0;
07307         uy = 0.0;
07308         uz = 0.0;
07309         qxx = 0.0;
07310         qyx = 0.0;

```

```

07311     qyy = 0.0;
07312     qzx = 0.0;
07313     qzy = 0.0;
07314     qzz = 0.0;
07315 #endif /* if defined(WITH_TINKER) */
07316
07317     /* check
07318     mc = 0.0;
07319     mux = 0.0;
07320     muy = 0.0;
07321     muz = 0.0;
07322     mqxx = 0.0;
07323     mqyx = 0.0;
07324     mqyy = 0.0;
07325     mqzx = 0.0;
07326     mqzy = 0.0;
07327     mqzz = 0.0; */
07328
07329     /* Make sure we're on the grid */
07330     if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx)) || \
07331         (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy)) || \
07332         (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
07333         Vnm_print(2, "fillcoPermanentMultipole: Atom #%d at (%4.3f, %4.3f,
07334             %4.3f) is off the mesh (ignoring this atom):\n", iatom, apos[0], apos[1], apos[2]
07335     );
07336         Vnm_print(2, "fillcoPermanentMultipole: xmin = %g, xmax = %g\n",
07337         xmin, xmax);
07338         Vnm_print(2, "fillcoPermanentMultipole: ymin = %g, ymax = %g\n",
07339         ymin, ymax);
07340         Vnm_print(2, "fillcoPermanentMultipole: zmin = %g, zmax = %g\n",
07341         zmin, zmax);
07342         fflush(stderr);
07343     } else {
07344
07345         /* Convert the atom position to grid reference frame */
07346         position[0] = apos[0] - xmin;
07347         position[1] = apos[1] - ymin;
07348         position[2] = apos[2] - zmin;
07349
07350         /* Figure out which vertices we're next to */
07351         ifloat = position[0]/hx;
07352         jfloat = position[1]/hy;
07353         kfloat = position[2]/hzed;
07354
07355         ip1 = (int)ceil(ifloat);
07356         ip2 = ip1 + 2;
07357         im1 = (int)floor(ifloat);
07358         im2 = im1 - 2;
07359         jp1 = (int)ceil(jfloat);
07360         jp2 = jp1 + 2;
07361         jm1 = (int)floor(jfloat);
07362         jm2 = jm1 - 2;
07363         kp1 = (int)ceil(kfloat);
07364         kp2 = kp1 + 2;
07365         km1 = (int)floor(kfloat);
07366         km2 = km1 - 2;
07367
07368         /* This step shouldn't be necessary, but it saves nasty debugging
07369          * later on if something goes wrong */
07370         ip2 = VMIN2(ip2,nx-1);
07371         ip1 = VMIN2(ip1,nx-1);
07372         im1 = VMAX2(im1,0);
07373         im2 = VMAX2(im2,0);
07374         jp2 = VMIN2(jp2,ny-1);
07375         jp1 = VMIN2(jp1,ny-1);
07376         jm1 = VMAX2(jm1,0);
07377         jm2 = VMAX2(jm2,0);
07378         kp2 = VMIN2(kp2,nz-1);
07379         kp1 = VMIN2(kp1,nz-1);
07380         km1 = VMAX2(km1,0);
07381         km2 = VMAX2(km2,0);
07382
07383         /* Now assign fractions of the charge to the nearby verts */
07384         for (ii=im2; ii<=ip2; ii++) {
07385             mi = VFCHI4(ii,ifloat);
07386             mx = bspline4(mi);
07387             dnx = db spline4(mi);
07388             d2mx = d2bspline4(mi);
07389             for (jj=jm2; jj<=jp2; jj++) {
07390                 mj = VFCHI4(jj,jfloat);
07391                 my = bspline4(mj);
07392             }
07393         }
07394     }
07395 }
```

```

07387             dmy = dbspline4(mj);
07388             d2my = d2bspline4(mj);
07389             for (kk=km2; kk<=kp2; kk++) {
07390                 mk = VFCHI4(kk,kfloat);
07391                 mz = bspline4(mk);
07392                 dmz = dbspline4(mk);
07393                 d2mz = d2bspline4(mk);
07394                 charge = mx*my*mz*c -
07395                     dmz*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz +
07396                     d2mx*my*mz*qxx +
07397                     dmz*dmy*mz*qyx + mx*d2my*mz*qyy +
07398                     dmz*my*dmz*qzx + mx*dmy*dmz*qzy + mx*my*d2mz*qzz;
07399                 thee->charge[IJK(ii,jj,kk)] += charge;
07400
07401             /* sanity check - recalculate traceless multipoles
07402                 from the grid charge distribution for this
07403                 site.
07404
07405                 mir = (mi - 2.5) * hx;
07406                 mjr = (mj - 2.5) * hy;
07407                 mkr = (mk - 2.5) * hzed;
07408                 mr2 = mir*mir+mjr*mjr+mkr*mkr;
07409                 mc += charge;
07410                 mux += mir * charge;
07411                 muy += mjr * charge;
07412                 muz += mkr * charge;
07413                 mqxx += (1.5*mir*mir - 0.5*mr2) * charge;
07414                 mqyx += 1.5*mjr*mjr * charge;
07415                 mqyy += (1.5*mjr*mjr - 0.5*mr2) * charge;
07416                 mqzx += 1.5*mkr*mkr * charge;
07417                 mqzy += 1.5*mkr*mjr * charge;
07418                 mqzz += (1.5*mkr*mkr - 0.5*mr2) * charge;
07419                 */
07420             }
07421         }
07422     }
07423 } /* endif (on the mesh) */
07424
07425 /* print out the Grid vs. Ideal Point Multipole. */
07426
07427 /*
07428 debye = 4.8033324;
07429 mc = mc/f;
07430 mux = mux/f*debye;
07431 muy = muy/f*debye;
07432 muz = muz/f*debye;
07433 mqxx = mqxx/f*debye;
07434 mqyy = mqyy/f*debye;
07435 mqzz = mqzz/f*debye;
07436 mqyx = mqyx/f*debye;
07437 mqzx = mqzx/f*debye;
07438 mqzy = mqzy/f*debye;
07439
07440     printf(" Grid v. Actual Permanent Multipole for Site %i\n",iatom);
07441     printf(" G: %10.6f\n",mc);
07442     printf(" A: %10.6f\n",c/f);
07443     printf(" G: %10.6f %10.6f %10.6f\n",mux,muy,muz);
07444     printf(" A: %10.6f %10.6f %10.6f\n",
07445         (ux * hx / f) * debye,
07446         (uy * hy / f) * debye,
07447         (uz * hzed / f) * debye);
07448     printf(" G: %10.6f\n",mqxx);
07449     printf(" A: %10.6f\n",quad[0]*debye);
07450     printf(" G: %10.6f %10.6f\n",mqyx,mqyy);
07451     printf(" A: %10.6f %10.6f\n",quad[3]*debye,quad[4]*debye);
07452     printf(" G: %10.6f %10.6f %10.6f\n",mqzx,mqzy,mqzz);
07453     printf(" A: %10.6f %10.6f %10.6f\n",
07454         quad[6]*debye,quad[7]*debye,quad[8]*debye); */
07455
07456 } /* endfor (each atom) */
07457 }
07458
07459 #if defined(WITH_TINKER)
07460
07461 VPUBLIC void fillcoInducedDipole(Vpmg *thee) {
07462
07463     Valist *alist;
07464     Vpbe *pbe;
07465     Vatom *atom;
07466     /* Conversions */
07467     double zmagic, f;

```

```

07468  /* Grid */
07469  double xmin, xmax, ymin, ymax, zmin, zmax;
07470  double xlen, ylen, zlen, ifloat, jfloat, kfloat;
07471  double hx, hy, hzed, *apos, position[3];
07472  /* B-spline weights */
07473  double mx, my, mz, dmx, dmy, dmz;
07474  /* Dipole */
07475  double charge, *dipole, ux, uy, uz;
07476  double mi, mj, mk;
07477  /* Loop indeces */
07478  int i, ii, jj, kk, nx, ny, nz, iatom;
07479  int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, kml, kpl, kp2;
07480
07481  double debye;
07482  double mux, muy, muz;
07483  double mir, mjr, mkr;
07484
07485  VASSERT(thee != VNULL);
07486
07487  /* Get PBE info */
07488  pbe = thee->pbe;
07489  alist = pbe->alist;
07490  zmagic = Vpbe_getZmagic(pbe);
07491
07492  /* Mesh info */
07493  nx = thee->pmgp->nx;
07494  ny = thee->pmgp->ny;
07495  nz = thee->pmgp->nz;
07496  hx = thee->pmgp->hx;
07497  hy = thee->pmgp->hy;
07498  hzed = thee->pmgp->hzed;
07499
07500  /* Conversion */
07501  f = zmagic/(hx*hy*hzed);
07502
07503  /* Define the total domain size */
07504  xlen = thee->pmgp->xlen;
07505  ylen = thee->pmgp->ylen;
07506  zlen = thee->pmgp->zlen;
07507
07508  /* Define the min/max dimensions */
07509  xmin = thee->pmgp->xcent - (xlen/2.0);
07510  ymin = thee->pmgp->ycent - (ylen/2.0);
07511  zmin = thee->pmgp->zcent - (zlen/2.0);
07512  xmax = thee->pmgp->xcent + (xlen/2.0);
07513  ymax = thee->pmgp->ycent + (ylen/2.0);
07514  zmax = thee->pmgp->zcent + (zlen/2.0);
07515
07516  /* Fill in the source term (induced dipoles) */
07517  Vnm_print(0, "fillcoInducedDipole: filling in the source term.\n");
07518  for (iatom=0; iatom<Valist_getNumberAtoms(alist);
07519    iatom++) {
07520    atom = Valist_getAtom(alist, iatom);
07521    apos = Vatom_getPosition(atom);
07522
07523    dipole = Vatom_getInducedDipole(atom);
07524    ux = dipole[0]/hx*f;
07525    uy = dipole[1]/hy*f;
07526    uz = dipole[2]/hzed*f;
07527
07528    mux = 0.0;
07529    muy = 0.0;
07530    muz = 0.0;
07531
07532    /* Make sure we're on the grid */
07533    if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx)) || \
07534      (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy)) || \
07535      (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
07536      Vnm_print(2, "fillcoInducedDipole: Atom #d at (%4.3f, %4.3f,
07537      %4.3f) is off the mesh (ignoring this atom):\n", iatom, apos[0], apos[1], apos[2]);
07538      Vnm_print(2, "fillcoInducedDipole: xmin = %g, xmax = %g\n", xmin,
07539      xmax);
07540      Vnm_print(2, "fillcoInducedDipole: ymin = %g, ymax = %g\n", ymin,
07541      ymax);
07542      Vnm_print(2, "fillcoInducedDipole: zmin = %g, zmax = %g\n", zmin,
07543      zmax);
07544      fflush(stderr);
07545    } else {
07546
07547      /* Convert the atom position to grid reference frame */

```

```

07544     position[0] = apos[0] - xmin;
07545     position[1] = apos[1] - ymin;
07546     position[2] = apos[2] - zmin;
07547
07548     /* Figure out which vertices we're next to */
07549     ifloat = position[0]/hx;
07550     jfloat = position[1]/hy;
07551     kfloat = position[2]/hzed;
07552
07553     ip1 = (int)ceil(ifloat);
07554     ip2 = ip1 + 2;
07555     im1 = (int)floor(ifloat);
07556     im2 = im1 - 2;
07557     jp1 = (int)ceil(jfloat);
07558     jp2 = jp1 + 2;
07559     jm1 = (int)floor(jfloat);
07560     jm2 = jm1 - 2;
07561     kp1 = (int)ceil(kfloat);
07562     kp2 = kp1 + 2;
07563     km1 = (int)floor(kfloat);
07564     km2 = km1 - 2;
07565
07566     /* This step shouldn't be necessary, but it saves nasty debugging
07567      * later on if something goes wrong */
07568     ip2 = VMIN2(ip2,nx-1);
07569     ip1 = VMIN2(ip1,nx-1);
07570     im1 = VMAX2(im1,0);
07571     im2 = VMAX2(im2,0);
07572     jp2 = VMIN2(jp2,ny-1);
07573     jp1 = VMIN2(jp1,ny-1);
07574     jm1 = VMAX2(jm1,0);
07575     jm2 = VMAX2(jm2,0);
07576     kp2 = VMIN2(kp2,nz-1);
07577     kp1 = VMIN2(kp1,nz-1);
07578     km1 = VMAX2(km1,0);
07579     km2 = VMAX2(km2,0);
07580
07581     /* Now assign fractions of the dipole to the nearby verts */
07582     for (ii=im2; ii<ip2; ii++) {
07583         mi = VFCHI4(ii,ifloat);
07584         mx = bspline4(mi);
07585         dmx = dbspline4(mi);
07586         for (jj=jm2; jj<=jp2; jj++) {
07587             mj = VFCHI4(jj,jfloat);
07588             my = bspline4(mj);
07589             dmy = dbspline4(mj);
07590             for (kk=km2; kk<=kp2; kk++) {
07591                 mk = VFCHI4(kk,kfloat);
07592                 mz = bspline4(mk);
07593                 dmz = dbspline4(mk);
07594                 charge = -dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz;
07595                 thee->charge[IJK(ii,jj,kk)] += charge;
07596
07597                 /*
07598                  mir = (mi - 2.5) * hx;
07599                  mjr = (mj - 2.5) * hy;
07600                  mkr = (mk - 2.5) * hzed;
07601                  mux += mir * charge;
07602                  muy += mjr * charge;
07603                  muz += mkr * charge;
07604                  */
07605             }
07606         }
07607     }
07608 } /* endif (on the mesh) */
07609
07610 /* check
07611 debye = 4.8033324;
07612 mux = mux/f*debye;
07613 muy = muy/f*debye;
07614 muz = muz/f*debye;
07615
07616 printf(" Grid v. Actual Induced Dipole for Site %i\n",iatom);
07617 printf(" G: %10.6f %10.6f %10.6f\n",mux,muy,muz);
07618 printf(" A: %10.6f %10.6f %10.6f\n",
07619             (ux * hx / f) * debye,
07620             (uy * hy / f) * debye,
07621             (uz * hzed / f) * debye);
07622 */
07623
07624 } /* endfor (each atom) */

```

```

07625 }
07626
07627 VPUBLIC void fillcoNLInducedDipole(Vpmg *thee) {
07628
07629     Valist *alist;
07630     Vpbe *pbe;
07631     Vatom *atom;
07632     /* Conversions */
07633     double zmagic, f;
07634     /* Grid */
07635     double xmin, xmax, ymin, ymax, zmin, zmax;
07636     double xlabel, ylabel, zlabel, ifloat, jfloat, kfloat;
07637     double hx, hy, hzed, *apos, position[3];
07638     /* B-spline weights */
07639     double mx, my, mz, dmx, dmy, dmz;
07640     /* Dipole */
07641     double charge, *dipole, ux, uy, uz;
07642     double mi, mj, mk;
07643     /* Loop indeces */
07644     int i, ii, jj, kk, nx, ny, nz, iatom;
07645     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
07646
07647     /* sanity check */
07648     double debye;
07649     double mux, muy, muz;
07650     double mir, mjr, mkr;
07651     */
07652
07653     VASSERT(thee != VNULL);
07654
07655     /* Get PBE info */
07656     pbe = thee->pbe;
07657     alist = pbe->alist;
07658     zmagic = Vpbe_getZmagic(pbe);
07659
07660     /* Mesh info */
07661     nx = thee->pmgp->nx;
07662     ny = thee->pmgp->ny;
07663     nz = thee->pmgp->nz;
07664     hx = thee->pmgp->hx;
07665     hy = thee->pmgp->hy;
07666     hzed = thee->pmgp->hzed;
07667
07668     /* Conversion */
07669     f = zmagic/(hx*hy*hzed);
07670
07671     /* Define the total domain size */
07672     xlabel = thee->pmgp->xlabel;
07673     ylabel = thee->pmgp->ylabel;
07674     zlabel = thee->pmgp->zlabel;
07675
07676     /* Define the min/max dimensions */
07677     xmin = thee->pmgp->xcent - (xlabel/2.0);
07678     ymin = thee->pmgp->ycent - (ylabel/2.0);
07679     zmin = thee->pmgp->zcent - (zlabel/2.0);
07680     xmax = thee->pmgp->xcent + (xlabel/2.0);
07681     ymax = thee->pmgp->ycent + (ylabel/2.0);
07682     zmax = thee->pmgp->zcent + (zlabel/2.0);
07683
07684     /* Fill in the source term (non-local induced dipoles) */
07685     Vnm_Print(0, "fillcoNLInducedDipole: filling in source term.\n");
07686     for (iatom=0; iatom<Valist_getNumberAtoms(alist);
07687         iatom++) {
07688         atom = Valist_getAtom(alist, iatom);
07689         apos = Vatom_getPosition(atom);
07690
07691         dipole = Vatom_getNLInducedDipole(atom);
07692         ux = dipole[0]/hx*f;
07693         uy = dipole[1]/hy*f;
07694         uz = dipole[2]/hzed*f;
07695
07696         /*
07697         mux = 0.0;
07698         muy = 0.0;
07699         muz = 0.0;
07700         */
07701
07702         /* Make sure we're on the grid */
07703         if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx)) || \
07704             (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy)) || \

```

```

07705             (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
07706                 Vnm_print(2, "fillcoNLInducedDipole: Atom #%d at (%4.3f,
07707 %4.3f,%4.3f) is off the mesh (ignoring this atom):\n", iatom, apos[0], apos[1], apos[2]);
07708                 Vnm_print(2, "fillcoNLInducedDipole: xmin = %g, xmax = %g\n", xmin,
07709                 xmax);
07710                 Vnm_print(2, "fillcoNLInducedDipole: ymin = %g, ymax = %g\n", ymin,
07711                 ymax);
07712                 Vnm_print(2, "fillcoNLInducedDipole: zmin = %g, zmax = %g\n", zmin,
07713                 zmax);
07714                 fflush(stderr);
07715             } else {
07716
07717                 /* Convert the atom position to grid reference frame */
07718                 position[0] = apos[0] - xmin;
07719                 position[1] = apos[1] - ymin;
07720                 position[2] = apos[2] - zmin;
07721
07722                 /* Figure out which vertices we're next to */
07723                 ifloat = position[0]/hx;
07724                 jfloat = position[1]/hy;
07725                 kfloat = position[2]/hzed;
07726
07727                 ip1 = (int)ceil(ifloat);
07728                 ip2 = ip1 + 2;
07729                 im1 = (int)floor(ifloat);
07730                 im2 = im1 - 2;
07731                 jp1 = (int)ceil(jfloat);
07732                 jp2 = jp1 + 2;
07733                 jm1 = (int)floor(jfloat);
07734                 jm2 = jm1 - 2;
07735                 kp1 = (int)ceil(kfloat);
07736                 kp2 = kp1 + 2;
07737                 km1 = (int)floor(kfloat);
07738                 km2 = km1 - 2;
07739
07740                 /* This step shouldn't be necessary, but it saves nasty debugging
07741                  * later on if something goes wrong */
07742                 ip2 = VMIN2(ip2,nx-1);
07743                 ip1 = VMIN2(ip1,nx-1);
07744                 im1 = VMAX2(im1,0);
07745                 im2 = VMAX2(im2,0);
07746                 jp2 = VMIN2(jp2,ny-1);
07747                 jp1 = VMIN2(jp1,ny-1);
07748                 jm1 = VMAX2(jm1,0);
07749                 jm2 = VMAX2(jm2,0);
07750                 kp2 = VMIN2(kp2,nz-1);
07751                 kp1 = VMIN2(kp1,nz-1);
07752                 km1 = VMAX2(km1,0);
07753                 km2 = VMAX2(km2,0);
07754
07755                 /* Now assign fractions of the non local induced dipole
07756                  * to the nearby verts */
07757                 for (ii=im2; ii<=ip2; ii++) {
07758                     mi = VFCHI4(ii,ifloat);
07759                     mx = bspline4(mi);
07760                     dmx = dbspline4(mi);
07761                     for (jj=jm2; jj<=jp2; jj++) {
07762                         mj = VFCHI4(jj,jfloat);
07763                         my = bspline4(mj);
07764                         dmy = dbspline4(mj);
07765                         for (kk=km2; kk<=kp2; kk++) {
07766                             mk = VFCHI4(kk,kfloat);
07767                             mz = bspline4(mk);
07768                             dmz = dbspline4(mk);
07769                             charge = -dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz;
07770                             thee->charge[IJK(ii,jj,kk)] += charge;
07771
07772                             /*
07773                             mir = (mi - 2.5) * hx;
07774                             mjr = (mj - 2.5) * hy;
07775                             mkr = (mk - 2.5) * hzed;
07776                             mux += mir * charge;
07777                             muy += mjr * charge;
07778                             muz += mkr * charge;
07779                             */
07780                         }
07781                 }
07782             }
07783         }
07784     }
07785 } /* endif (on the mesh) */
07786
07787
07788
07789
07790
07791
07792
07793
07794
07795
07796
07797
07798
07799
07800
07801
07802
07803
07804
07805
07806
07807
07808
07809
07810
07811
07812
07813
07814
07815
07816
07817
07818
07819
07820
07821
07822
07823
07824
07825
07826
07827
07828
07829
07830
07831
07832
07833
07834
07835
07836
07837
07838
07839
07840
07841
07842
07843
07844
07845
07846
07847
07848
07849
07850
07851
07852
07853
07854
07855
07856
07857
07858
07859
07860
07861
07862
07863
07864
07865
07866
07867
07868
07869
07870
07871
07872
07873
07874
07875
07876
07877
07878
07879
07880
07881
07882
07883
07884
07885
07886
07887
07888
07889
07890
07891
07892
07893
07894
07895
07896
07897
07898
07899
07900
07901
07902
07903
07904
07905
07906
07907
07908
07909
07910
07911
07912
07913
07914
07915
07916
07917
07918
07919
07920
07921
07922
07923
07924
07925
07926
07927
07928
07929
07930
07931
07932
07933
07934
07935
07936
07937
07938
07939
07940
07941
07942
07943
07944
07945
07946
07947
07948
07949
07950
07951
07952
07953
07954
07955
07956
07957
07958
07959
07960
07961
07962
07963
07964
07965
07966
07967
07968
07969
07970
07971
07972
07973
07974
07975
07976
07977
07978
07979
07980
07981
07982
07983
07984
07985
07986
07987
07988
07989
07990
07991
07992
07993
07994
07995
07996
07997
07998
07999
07999
08000
08001
08002
08003
08004
08005
08006
08007
08008
08009
080010
080011
080012
080013
080014
080015
080016
080017
080018
080019
080020
080021
080022
080023
080024
080025
080026
080027
080028
080029
080030
080031
080032
080033
080034
080035
080036
080037
080038
080039
080040
080041
080042
080043
080044
080045
080046
080047
080048
080049
080050
080051
080052
080053
080054
080055
080056
080057
080058
080059
080060
080061
080062
080063
080064
080065
080066
080067
080068
080069
080070
080071
080072
080073
080074
080075
080076
080077
080078
080079
080080
080081
080082
080083
080084
080085
080086
080087
080088
080089
080090
080091
080092
080093
080094
080095
080096
080097
080098
080099
0800100
0800101
0800102
0800103
0800104
0800105
0800106
0800107
0800108
0800109
0800110
0800111
0800112
0800113
0800114
0800115
0800116
0800117
0800118
0800119
0800120
0800121
0800122
0800123
0800124
0800125
0800126
0800127
0800128
0800129
0800130
0800131
0800132
0800133
0800134
0800135
0800136
0800137
0800138
0800139
0800140
0800141
0800142
0800143
0800144
0800145
0800146
0800147
0800148
0800149
0800150
0800151
0800152
0800153
0800154
0800155
0800156
0800157
0800158
0800159
0800160
0800161
0800162
0800163
0800164
0800165
0800166
0800167
0800168
0800169
0800170
0800171
0800172
0800173
0800174
0800175
0800176
0800177
0800178
0800179
0800180
0800181
0800182
0800183
0800184
0800185
0800186
0800187
0800188
0800189
0800190
0800191
0800192
0800193
0800194
0800195
0800196
0800197
0800198
0800199
0800200
0800201
0800202
0800203
0800204
0800205
0800206
0800207
0800208
0800209
0800210
0800211
0800212
0800213
0800214
0800215
0800216
0800217
0800218
0800219
0800220
0800221
0800222
0800223
0800224
0800225
0800226
0800227
0800228
0800229
0800230
0800231
0800232
0800233
0800234
0800235
0800236
0800237
0800238
0800239
0800240
0800241
0800242
0800243
0800244
0800245
0800246
0800247
0800248
0800249
0800250
0800251
0800252
0800253
0800254
0800255
0800256
0800257
0800258
0800259
0800260
0800261
0800262
0800263
0800264
0800265
0800266
0800267
0800268
0800269
0800270
0800271
0800272
0800273
0800274
0800275
0800276
0800277
0800278
0800279
0800280
0800281
0800282
0800283
0800284
0800285
0800286
0800287
0800288
0800289
0800290
0800291
0800292
0800293
0800294
0800295
0800296
0800297
0800298
0800299
0800300
0800301
0800302
0800303
0800304
0800305
0800306
0800307
0800308
0800309
0800310
0800311
0800312
0800313
0800314
0800315
0800316
0800317
0800318
0800319
0800320
0800321
0800322
0800323
0800324
0800325
0800326
0800327
0800328
0800329
0800330
0800331
0800332
0800333
0800334
0800335
0800336
0800337
0800338
0800339
0800340
0800341
0800342
0800343
0800344
0800345
0800346
0800347
0800348
0800349
0800350
0800351
0800352
0800353
0800354
0800355
0800356
0800357
0800358
0800359
0800360
0800361
0800362
0800363
0800364
0800365
0800366
0800367
0800368
0800369
0800370
0800371
0800372
0800373
0800374
0800375
0800376
0800377
0800378
0800379
0800380
0800381
0800382
0800383
0800384
0800385
0800386
0800387
0800388
0800389
0800390
0800391
0800392
0800393
0800394
0800395
0800396
0800397
0800398
0800399
0800400
0800401
0800402
0800403
0800404
0800405
0800406
0800407
0800408
0800409
0800410
0800411
0800412
0800413
0800414
0800415
0800416
0800417
0800418
0800419
0800420
0800421
0800422
0800423
0800424
0800425
0800426
0800427
0800428
0800429
0800430
0800431
0800432
0800433
0800434
0800435
0800436
0800437
0800438
0800439
0800440
0800441
0800442
0800443
0800444
0800445
0800446
0800447
0800448
0800449
0800450
0800451
0800452
0800453
0800454
0800455
0800456
0800457
0800458
0800459
0800460
0800461
0800462
0800463
0800464
0800465
0800466
0800467
0800468
0800469
0800470
0800471
0800472
0800473
0800474
0800475
0800476
0800477
0800478
0800479
0800480
0800481
0800482
0800483
0800484
0800485
0800486
0800487
0800488
0800489
0800490
0800491
0800492
0800493
0800494
0800495
0800496
0800497
0800498
0800499
0800500
0800501
0800502
0800503
0800504
0800505
0800506
0800507
0800508
0800509
0800510
0800511
0800512
0800513
0800514
0800515
0800516
0800517
0800518
0800519
0800520
0800521
0800522
0800523
0800524
0800525
0800526
0800527
0800528
0800529
0800530
0800531
0800532
0800533
0800534
0800535
0800536
0800537
0800538
0800539
0800540
0800541
0800542
0800543
0800544
0800545
0800546
0800547
0800548
0800549
0800550
0800551
0800552
0800553
0800554
0800555
0800556
0800557
0800558
0800559
0800560
0800561
0800562
0800563
0800564
0800565
0800566
0800567
0800568
0800569
0800570
0800571
0800572
0800573
0800574
0800575
0800576
0800577
0800578
0800579
0800580
0800581
0800582
0800583
0800584
0800585
0800586
0800587
0800588
0800589
0800590
0800591
0800592
0800593
0800594
0800595
0800596
0800597
0800598
0800599
0800600
0800601
0800602
0800603
0800604
0800605
0800606
0800607
0800608
0800609
0800610
0800611
0800612
0800613
0800614
0800615
0800616
0800617
0800618
0800619
0800620
0800621
0800622
0800623
0800624
0800625
0800626
0800627
0800628
0800629
0800630
0800631
0800632
0800633
0800634
0800635
0800636
0800637
0800638
0800639
0800640
0800641
0800642
0800643
0800644
0800645
0800646
0800647
0800648
0800649
0800650
0800651
0800652
0800653
0800654
0800655
0800656
0800657
0800658
0800659
0800660
0800661
0800662
0800663
0800664
0800665
0800666
0800667
0800668
0800669
0800670
0800671
0800672
0800673
0800674
0800675
0800676
0800677
0800678
0800679
0800680
0800681
0800682
0800683
0800684
0800685
0800686
0800687
0800688
0800689
0800690
0800691
0800692
0800693
0800694
0800695
0800696
0800697
0800698
0800699
0800700
0800701
0800702
0800703
0800704
0800705
0800706
0800707
0800708
0800709
0800710
0800711
0800712
0800713
0800714
0800715
0800716
0800717
0800718
0800719
0800720
0800721
0800722
0800723
0800724
0800725
0800726
0800727
0800728
0800729
0800730
0800731
0800732
0800733
0800734
0800735
0800736
0800737
0800738
0800739
0800740
0800741
0800742
0800743
0800744
0800745
0800746
0800747
0800748
0800749
0800750
0800751
0800752
0800753
0800754
0800755
0800756
0800757
0800758
0800759
0800760
0800761
0800762
0800763
0800764
0800765
0800766
0800767
0800768
0800769
0800770
0800771
0800772
0800773
0800774
0800775
0800776
0800777
0800778
0800779
0800780
0800781
0800782
0800783
0800784
0800785
0800786
0800787
0800788
0800789
0800790
0800791
0800792
0800793
0800794
0800795
0800796
0800797
0800798
0800799
0800800
0800801
0800802
0800803
0800804
0800805
0800806
0800807
0800808
0800809
0800810
0800811
0800812
0800813
0800814
0800815
0800816
0800817
0800818
0800819
0800820
0800821
0800822
0800823
0800824
0800825
0800826
0800827
0800828
0800829
0800830
0800831
0800832
0800833
0800834
0800835
0800836
0800837
0800838
0800839
0800840
0800841
0800842
0800843
0800844
0800845
0800846
0800847
0800848
0800849
0800850
0800851
0800852
0800853
0800854
0800855
0800856
0800857
0800858
0800859
0800860
0800861
0800862
0800863
0800864
0800865
0800866
0800867
0800868
0800869
0800870
0800871
0800872
0800873
0800874
0800875
0800876
0800877
0800878
0800879
0800880
0800881
0800882
0800883
0800884
0800885
0800886
0800887
0800888
0800889
0800890
0800891
0800892
0800893
0800894
0800895
0800896
0800897
0800898
0800899
0800900
0800901
0800902
0800903
0800904
0800905
0800906
0800907
0800908
0800909
0800910
0800911
0800912
0800913
0800914
0800915
0800916
0800917
0800918
0800919
0800920
0800921
0800922
0800923
0800924
0800925
0800926
0800927
0800928
0800929
0800930
0800931
0800932
0800933
0800934
0800935
0800936
0800937
0800938
0800939
0800940
0800941
0800942
0800943
0800944
0800945
0800946
0800947
0800948
0800949
0800950
0800951
0800952
0800953
0800954
0800955
0800956
0800957
0800958
0800959
0800960
0800961
0800962
0800963
0800964
0800965
0800966
0800967
0800968
0800969
0800970
0800971
0800972
0800973
0800974
0800975
0800976
0800977
0800978
0800979
0800980
0800981
0800982
0800983
0800984
0800985
0800986
0800987
0800988
0800989
0800990
0800991
0800992
0800993
0800994
0800995
0800996
0800997
0800998
0800999
0800100
0800101
0800102
0800103
0800104
0800105
0800106
0800107
0800108
0800109
0800110
0800111
0800112
0800113
0800114
0800115
0800116
0800117
0800118
0800119
0800120
0800121
0800122
0800123
0800124
0800125
0800126
0800127
0800128
0800129
0800130
0800131
0800132
0800133
0800134
0800135
0800136
0800137
0800138
0800139
0800140
0800141
0800142
0800143
0800144
0800145
0800146
0800147
0800148
0800149
0800150
0800151
0800152
0800153
0800154
0800155
0800156
0800157
0800158
0800159
0800160
0800161
0800162
0800163
0800164
0800165
0800166
0800167
0800168
0800169
0800170
0800171
0800172
0800173
0800174
0800175
0800176
0800177
0800178
0800179
0800180
0800181
0800182
0800183
0800184
0800185
0800186
0800187
0800188
0800189
0800190
0
```

```

07782     debye = 4.8033324;
07783     mux = mux/f*debye;
07784     muy = muy/f*debye;
07785     muz = muz/f*debye;
07786
07787     printf(" Grid v. Actual Non-Local Induced Dipole for Site %i\n",iatom);
07788     printf(" G: %10.6f %10.6f %10.6f\n",mux,muy,muz);
07789     printf(" A: %10.6f %10.6f %10.6f\n\n",
07790         (ux * hx / f) * debye,
07791         (uy * hy / f) * debye,
07792         (uz * hzed /f) * debye); */
07793
07794 } /* endfor (each atom) */
07795 }
07796
07797 VPUBLIC double Vpmg_qfPermanentMultipoleEnergy(
07798     Vpmg *thee, int atomID) {
07799
07800     double *u;
07801     Vatom *atom;
07802     /* Grid variables */
07803     int nx, ny, nz;
07804     double xmax, xmin, ymax, ymin, zmax, zmin;
07805     double hx, hy, hzed, ifloat, jfloat, kfloat;
07806     double mi, mj, mk;
07807     double *position;
07808     double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz;
07809     /* Loop indeces */
07810     int ip1,ip2,im1,im2,jp1,jp2,jm1,jm2,kp1,kp2,km1,km2;
07811     int i,j,ii,jj,kk;
07812     /* Potential, field, field gradient and multipole components */
07813     double pot, rfe[3], rfde[3][3], energy;
07814     double f, charge, *dipole, *quad;
07815     double qxx, qyx, qyy, qzx, qzy, qzz;
07816
07817
07818     VASSERT(thee != VNULL);
07819     VASSERT(thee->filled);
07820
07821     /* Get the mesh information */
07822     nx = thee->pmgp->nx;
07823     ny = thee->pmgp->ny;
07824     nz = thee->pmgp->nz;
07825     hx = thee->pmgp->hx;
07826     hy = thee->pmgp->hy;
07827     hzed = thee->pmgp->hzed;
07828     xmax = thee->xf[nx-1];
07829     ymax = thee->yf[ny-1];
07830     zmax = thee->zf[nz-1];
07831     xmin = thee->xf[0];
07832     ymin = thee->yf[0];
07833     zmin = thee->zf[0];
07834
07835     u = thee->u;
07836
07837     atom = Valist_getAtom(thee->pbe->alist, atomID);
07838
07839     /* Currently all atoms must be in the same partition. */
07840
07841     VASSERT(atom->partID != 0);
07842
07843     /* Convert the atom position to grid coordinates */
07844
07845     position = VatomGetPosition(atom);
07846     ifloat = (position[0] - xmin)/hx;
07847     jfloat = (position[1] - ymin)/hy;
07848     kfloat = (position[2] - zmin)/hzed;
07849     ip1 = (int)ceil(ifloat);
07850     ip2 = ip1 + 2;
07851     im1 = (int)floor(ipfloat);
07852     im2 = im1 - 2;
07853     jp1 = (int)ceil(jfloat);
07854     jp2 = jp1 + 2;
07855     jm1 = (int)floor(jfloat);
07856     jm2 = jm1 - 2;
07857     kp1 = (int)ceil(kfloat);
07858     kp2 = kp1 + 2;
07859     km1 = (int)floor(kfloat);
07860     km2 = km1 - 2;
07861

```

```

07862 /* This step shouldn't be necessary, but it saves nasty debugging
07863 * later on if something goes wrong */
07864 ip2 = VMIN2(ip2,nx-1);
07865 ip1 = VMIN2(ip1,ny-1);
07866 im1 = VMAX2(im1,0);
07867 im2 = VMAX2(im2,0);
07868 jp2 = VMIN2(jp2,ny-1);
07869 jp1 = VMIN2(jp1,ny-1);
07870 jm1 = VMAX2(jm1,0);
07871 jm2 = VMAX2(jm2,0);
07872 kp2 = VMIN2(kp2,nz-1);
07873 kp1 = VMIN2(kp1,nz-1);
07874 km1 = VMAX2(km1,0);
07875 km2 = VMAX2(km2,0);
07876
07877 /* Initialize observables to zero */
07878 energy = 0.0;
07879 pot = 0.0;
07880 for (i=0;i<3;i++) {
07881     rfe[i] = 0.0;
07882     for (j=0;j<3;j++) {
07883         rfde[i][j] = 0.0;
07884     }
07885 }
07886
07887 for (ii=im2; ii<=ip2; ii++) {
07888     mi = VFCHI4(ii,ifloat);
07889     mx = bspline4(mi);
07890     dmx = dbspline4(mi);
07891     d2mx = d2bspline4(mi);
07892     for (jj=jm2; jj<=jp2; jj++) {
07893         mj = VFCHI4(jj,jffloat);
07894         my = bspline4(mj);
07895         dmy = dbspline4(mj);
07896         d2my = d2bspline4(mj);
07897         for (kk=km2; kk<=kp2; kk++) {
07898             mk = VFCHI4(kk,kfloat);
07899             mz = bspline4(mk);
07900             dmz = dbspline4(mk);
07901             d2mz = d2bspline4(mk);
07902             f = u[IJK(ii,jj,kk)];
07903             /* potential */
07904             pot += f*mx*my*mz;
07905             /* field */
07906             rfe[0] += f*dmx*my*mz/hx;
07907             rfe[1] += f*mx*dmy*mz/hy;
07908             rfe[2] += f*mx*my*dmz/hzed;
07909             /* field gradient */
07910             rfde[0][0] += f*d2mx*my*mz/(hx*hx);
07911             rfde[1][0] += f*dmx*dmy*mz/(hy*hx);
07912             rfde[1][1] += f*mx*d2my*mz/(hy*hy);
07913             rfde[2][0] += f*dmx*my*dmz/(hx*hzed);
07914             rfde[2][1] += f*mx*dmy*dmz/(hy*hzed);
07915             rfde[2][2] += f*mx*my*d2mz/(hzed*hzed);
07916         }
07917     }
07918 }
07919
07920 charge = Vatom_getCharge(atom);
07921 dipole = Vatom_getDipole(atom);
07922 quad = Vatom_getQuadrupole(atom);
07923 qxx = quad[0]/3.0;
07924 qyx = quad[3]/3.0;
07925 qyy = quad[4]/3.0;
07926 qzx = quad[6]/3.0;
07927 qzy = quad[7]/3.0;
07928 qzz = quad[8]/3.0;
07929
07930 energy = pot * charge
07931     - rfe[0] * dipole[0]
07932     - rfe[1] * dipole[1]
07933     - rfe[2] * dipole[2]
07934     + rfde[0][0]*qxx
07935     + 2.0*rfde[1][0]*qyx + rfde[1][1]*qyy
07936     + 2.0*rfde[2][0]*qzx + 2.0*rfde[2][1]*qzy + rfde[2][2]*qzz;
07937
07938 return energy;
07939 }
07940
07941 VPUBLIC void Vpmg_fieldSpline4(Vpmg *thee, int atomID,
07942     double field[3]) {

```

```

07942
07943     Vatom *atom;
07944     double *u, f;
07945     /* Grid variables */
07946     int nx, ny, nz;
07947     double xmax, xmin, ymax, ymin, zmax, zmin;
07948     double hx, hy, hzed, ifloat, jfloat, kfloat;
07949     double *apos, position[3];
07950     /* B-Spline weights */
07951     double mx, my, mz, dmx, dmy, dmz;
07952     double mi, mj, mk;
07953     /* Loop indeces */
07954     int ip1, ip2, im1, im2, jp1, jp2, jm1, jm2, kp1, kp2, km1, km2;
07955     int i, j, ii, jj, kk;
07956
07957
07958     VASSERT (thee != VNULL);
07959
07960     /* Get the mesh information */
07961     nx = thee->pmgp->nx;
07962     ny = thee->pmgp->ny;
07963     nz = thee->pmgp->nz;
07964     hx = thee->pmgp->hx;
07965     hy = thee->pmgp->hy;
07966     hzed = thee->pmgp->hzed;
07967     xmax = thee->xf[nx-1];
07968     ymax = thee->yf[ny-1];
07969     zmax = thee->zf[nz-1];
07970     xmin = thee->xf[0];
07971     ymin = thee->yf[0];
07972     zmin = thee->zf[0];
07973
07974     u = thee->u;
07975
07976     atom = Valist_getAtom(thee->pbe->alist, atomID);
07977
07978     /* Currently all atoms must be in the same partition. */
07979
07980     VASSERT (atom->partID != 0);
07981
07982     /* Convert the atom position to grid coordinates */
07983
07984     apos = Vatom_getPosition(atom);
07985     position[0] = apos[0] - xmin;
07986     position[1] = apos[1] - ymin;
07987     position[2] = apos[2] - zmin;
07988     ifloat = position[0]/hx;
07989     jfloat = position[1]/hy;
07990     kfloat = position[2]/hzed;
07991     ip1 = (int)ceil(ifloat);
07992     ip2 = ip1 + 2;
07993     im1 = (int)floor(ifloat);
07994     im2 = im1 - 2;
07995     jp1 = (int)ceil(jfloat);
07996     jp2 = jp1 + 2;
07997     jm1 = (int)floor(jfloat);
07998     jm2 = jm1 - 2;
07999     kp1 = (int)ceil(kfloat);
08000     kp2 = kp1 + 2;
08001     km1 = (int)floor(kfloat);
08002     km2 = km1 - 2;
08003
08004     /* This step shouldn't be necessary, but it saves nasty debugging
08005      * later on if something goes wrong */
08006     ip2 = VMIN2(ip2,nx-1);
08007     ip1 = VMIN2(ip1,nx-1);
08008     im1 = VMAX2(im1,0);
08009     im2 = VMAX2(im2,0);
08010     jp2 = VMIN2(jp2,ny-1);
08011     jp1 = VMIN2(jp1,ny-1);
08012     jm1 = VMAX2(jm1,0);
08013     jm2 = VMAX2(jm2,0);
08014     kp2 = VMIN2(kp2,nz-1);
08015     kp1 = VMIN2(kp1,nz-1);
08016     km1 = VMAX2(km1,0);
08017     km2 = VMAX2(km2,0);
08018
08019     for (i=0;i<3;i++){
08020         field[i] = 0.0;
08021     }
08022

```

```

08023     for (ii=im2; ii<=ip2; ii++) {
08024         mi = VFCHI4(ii,ifloat);
08025         mx = bspline4(mi);
08026         dmx = db spline4(mi);
08027         for (jj=jm2; jj<=jp2; jj++) {
08028             mj = VFCHI4(jj,jf float);
08029             my = bspline4(mj);
08030             dmy = db spline4(mj);
08031             for (kk=km2; kk<=kp2; kk++) {
08032                 mk = VFCHI4(kk,kfloat);
08033                 mz = bspline4(mk);
08034                 dmz = db spline4(mk);
08035                 f = u[IJK(ii,jj,kk)];
08036
08037                 field[0] += f*dmx*my*mz/hx;
08038                 field[1] += f*mx*dmy*mz/hy;
08039                 field[2] += f*mx*my*dmz/hzed;
08040             }
08041         }
08042     }
08043 }
08044
08045 VPUBLIC void Vpmg_qfPermanentMultipoleForce(Vpmg
08046     *thee, int atomID,
08047                               double force[3], double torque[3]) {
08048
08049     Vatom *atom;
08050     double f, *u, *apos, position[3];
08051
08052     /* Grid variables */
08053     int nx,ny,nz;
08054     double xlen, ylen, zlen, xmin, ymin, zmin, xmax
08055     , ymax, zmax;
08056     double hx, hy, hzed, ifloat, jfloat, kfloat;
08057
08058     /* B-spline weights */
08059     double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz, d3mx, d3my, d3mz;
08060     double mi, mj, mk;
08061
08062     /* Loop indeces */
08063     int i, j, k, ii, jj, kk;
08064     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
08065
08066     /* Potential, field, field gradient and 2nd field gradient */
08067     double pot, e[3], de[3][3], d2e[3][3][3];
08068
08069     /* Permanent multipole components */
08070     double *dipole, *quad;
08071     double c, ux, uy, uz, qxq, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
08072
08073     VASSERT(thee != VNULL);
08074     VASSERT(thee->filled);
08075
08076     atom = Valist_getAtom(thee->pbe->alist, atomID);
08077
08078     /* Currently all atoms must be in the same partition. */
08079
08080     VASSERT(atom->partID != 0);
08081
08082     apos = Vatom_getPosition(atom);
08083
08084     c = Vatom_getCharge(atom);
08085     dipole = Vatom_getDipole(atom);
08086     ux = dipole[0];
08087     uy = dipole[1];
08088     uz = dipole[2];
08089     quad = Vatom_getQuadrupole(atom);
08090     qxq = quad[0]/3.0;
08091     qxy = quad[1]/3.0;
08092     qxz = quad[2]/3.0;
08093     qyx = quad[3]/3.0;
08094     qyy = quad[4]/3.0;
08095     qyz = quad[5]/3.0;
08096     qzx = quad[6]/3.0;
08097     qzy = quad[7]/3.0;
08098     qzz = quad[8]/3.0;
08099
08100     /* Initialize observables */
08101     pot = 0.0;
08102     for (i=0;i<3;i++){
08103         e[i] = 0.0;

```

```

08102     for (j=0; j<3; j++) {
08103         de[i][j] = 0.0;
08104         for (k=0; k<3; k++) {
08105             d2e[i][j][k] = 0.0;
08106         }
08107     }
08108 }
08109
08110 /* Mesh info */
08111 nx = thee->pmgp->nx;
08112 ny = thee->pmgp->ny;
08113 nz = thee->pmgp->nz;
08114 hx = thee->pmgp->hx;
08115 hy = thee->pmgp->hy;
08116 hzed = thee->pmgp->hzed;
08117 xlen = thee->pmgp->xlen;
08118 ylen = thee->pmgp->ylen;
08119 zlen = thee->pmgp->zlen;
08120 xmin = thee->pmgp->xmin;
08121 ymin = thee->pmgp->ymin;
08122 zmin = thee->pmgp->zmin;
08123 xmax = thee->pmgp->xmax;
08124 ymax = thee->pmgp->ymax;
08125 zmax = thee->pmgp->zmax;
08126 u = thee->u;
08127
08128 /* Make sure we're on the grid */
08129 if ((apos[0]<=(xmin+2*hx)) || (apos[0]>=(xmax-2*hx)) \
08130 || (apos[1]<=(ymin+2*hy)) || (apos[1]>=(ymax-2*hy)) \
08131 || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
08132     Vnm_print(2, "qfPermanentMultipoleForce: Atom off the mesh (ignoring)
%6.3f %6.3f\n", apos[0], apos[1], apos[2]);
08133     fflush(stderr);
08134 } else {
08135
08136     /* Convert the atom position to grid coordinates */
08137     position[0] = apos[0] - xmin;
08138     position[1] = apos[1] - ymin;
08139     position[2] = apos[2] - zmin;
08140     ifloat = position[0]/hx;
08141     jfloat = position[1]/hy;
08142     kfloat = position[2]/hzed;
08143     ipl = (int)ceil(ifloat);
08144     ip2 = ipl + 2;
08145     im1 = (int)floor(ifloat);
08146     im2 = im1 - 2;
08147     jp1 = (int)ceil(jfloat);
08148     jp2 = jp1 + 2;
08149     jml = (int)floor(jfloat);
08150     jm2 = jml - 2;
08151     kp1 = (int)ceil(kfloat);
08152     kp2 = kp1 + 2;
08153     km1 = (int)floor(kfloat);
08154     km2 = km1 - 2;
08155
08156     /* This step shouldn't be necessary, but it saves nasty debugging
08157      * later on if something goes wrong */
08158     ip2 = VMIN2(ip2,nx-1);
08159     ipl = VMIN2(ip1,nx-1);
08160     im1 = VMAX2(im1,0);
08161     im2 = VMAX2(im2,0);
08162     jp2 = VMIN2(jp2,ny-1);
08163     jp1 = VMIN2(jp1,ny-1);
08164     jml = VMAX2(jml,0);
08165     jm2 = VMAX2(jm2,0);
08166     kp2 = VMIN2(kp2,nz-1);
08167     kp1 = VMIN2(kp1,nz-1);
08168     km1 = VMAX2(km1,0);
08169     km2 = VMAX2(km2,0);
08170
08171     for (ii=im2; ii<=ip2; ii++) {
08172         mi = VFCHI4(ii,ifloat);
08173         mx = bspline4(mi);
08174         dmx = dbspline4(mi);
08175         d2mx = d2bspline4(mi);
08176         d3mx = d3bspline4(mi);
08177         for (jj=jm2; jj<=jp2; jj++) {
08178             mj = VFCHI4(jj,jfloat);
08179             my = bspline4(mj);
08180             dmy = dbspline4(mj);
08181             d2my = d2bspline4(mj);

```

```

08182     d3my = d3bspline4(mj);
08183     for (kk=km2; kk<=kp2; kk++) {
08184         mk = VFCHI4(kk,kfloat);
08185         mz = bspline4(mk);
08186         dmz = dbspline4(mk);
08187         d2mz = d2bspline4(mk);
08188         d3mz = d3bspline4(mk);
08189         f = u[IJK(ij,jj,kk)];
08190         /* Potential */
08191         pot += f*mx*my*mz;
08192         /* Field */
08193         e[0] += f*dmx*my*mz/hx;
08194         e[1] += f*mx*dmy*mz/hy;
08195         e[2] += f*mx*my*dmz/hzed;
08196         /* Field gradient */
08197         de[0][0] += f*d2mx*my*mz / (hx*hx);
08198         de[1][0] += f*dmx*dmy*mz / (hy*hy);
08199         de[1][1] += f*mx*d2my*mz / (hy*hy);
08200         de[2][0] += f*dmx*my*dmz / (hx*hzed);
08201         de[2][1] += f*mx*dmy*dmz / (hy*hzed);
08202         de[2][2] += f*mx*my*d2mz / (hzed*hzed);
08203         /* 2nd Field Gradient
08204             VxVxVa */
08205         d2e[0][0][0] += f*d3mx*my*mz / (hx*hx*hx);
08206         d2e[0][0][1] += f*d2mx*dmy*mz / (hx*hy*hx);
08207         d2e[0][0][2] += f*d2mx*my*dmz / (hx*hx*hzed);
08208         /* VyVxVa */
08209         d2e[1][0][0] += f*d2mx*dmy*mz / (hx*hx*hy);
08210         d2e[1][0][1] += f*dmx*d2my*mz / (hx*hy*hy);
08211         d2e[1][0][2] += f*dmx*dmy*dmz / (hx*hy*hzed);
08212         /* VyVyVa */
08213         d2e[1][1][0] += f*dmx*d2my*mz / (hy*hy*hy);
08214         d2e[1][1][1] += f*mx*d3my*mz / (hy*hy*hy);
08215         d2e[1][1][2] += f*mx*d2my*dmz / (hy*hy*hzed);
08216         /* VzVxVa */
08217         d2e[2][0][0] += f*d2mx*my*dmz / (hx*hx*hzed);
08218         d2e[2][0][1] += f*dmx*dmy*dmz / (hx*hy*hzed);
08219         d2e[2][0][2] += f*dmx*my*d2mz / (hx*hzed*hzed);
08220         /* VzVyVa */
08221         d2e[2][1][0] += f*dmx*dmy*dmz / (hx*hy*hzed);
08222         d2e[2][1][1] += f*mx*d2my*dmz / (hy*hy*hzed);
08223         d2e[2][1][2] += f*mx*dmy*d2mz / (hy*hzed*hzed);
08224         /* VzVzVa */
08225         d2e[2][2][0] += f*dmx*my*d2mz / (hx*hzed*hzed);
08226         d2e[2][2][1] += f*mx*dmy*d2mz / (hy*hzed*hzed);
08227         d2e[2][2][2] += f*mx*my*d3mz / (hzed*hzed*hzed);
08228     }
08229 }
08230 }
08231 }
08232
08233 /* Monopole Force */
08234 force[0] = e[0]*c;
08235 force[1] = e[1]*c;
08236 force[2] = e[2]*c;
08237
08238 /* Dipole Force */
08239 force[0] -= de[0][0]*ux+de[1][0]*uy+de[2][0]*uz;
08240 force[1] -= de[1][0]*ux+de[1][1]*uy+de[2][1]*uz;
08241 force[2] -= de[2][0]*ux+de[2][1]*uy+de[2][2]*uz;
08242
08243 /* Quadrupole Force */
08244 force[0] += d2e[0][0]*qxx
08245     + d2e[1][0]*qyx*2.0+d2e[1][1]*qyy
08246     + d2e[2][0]*qzx*2.0+d2e[2][1]*qzy*2.0+d2e[2][2]*qzz;
08247 force[1] += d2e[0][0]*qxx
08248     + d2e[1][0]*qyx*2.0+d2e[1][1]*qyy
08249     + d2e[2][0]*qzx*2.0+d2e[2][1]*qzy*2.0+d2e[2][2]*qzz;
08250 force[2] += d2e[0][0]*qxx
08251     + d2e[1][0]*qyx*2.0+d2e[1][1]*qyy
08252     + d2e[2][0]*qzx*2.0+d2e[2][1]*qzy*2.0+d2e[2][2]*qzz;
08253
08254 /* Dipole Torque */
08255 torque[0] = uy * e[2] - uz * e[1];
08256 torque[1] = uz * e[0] - ux * e[2];
08257 torque[2] = ux * e[1] - uy * e[0];
08258 /* Quadrupole Torque */
08259 de[0][1] = de[1][0];
08260 de[0][2] = de[2][0];
08261 de[1][2] = de[2][1];
08262 torque[0] -= 2.0*(qyx*de[0][2] + qyy*de[1][2] + qyz*de[2][2])

```

```

08263           - qzx*de[0][1] - qzy*de[1][1] - qzz*de[2][1]);
08264     torque[1] -= 2.0*(qzx*de[0][0] + qzy*de[1][0] + qzz*de[2][0]
08265           - qxx*de[0][2] - qxy*de[1][2] - qxz*de[2][2]);
08266     torque[2] -= 2.0*(qxx*de[0][1] + qxy*de[1][1] + qxz*de[2][1]
08267           - qyx*de[0][0] - qyy*de[1][0] - qyz*de[2][0]);
08268
08269
08270 /* printf(" qPhi Force %f %f %f\n", force[0], force[1], force[2]);
08271   printf(" qPhi Torque %f %f %f\n", torque[0], torque[1], torque[2]); */
08272 }
08273
08274 VPUBLIC void Vpmg_ibPermanentMultipoleForce(Vpmg
08275   *thee, int atomID,
08276                           double force[3]) {
08277
08278   Valist *alist;
08279   Vacc *acc;
08280   Vpbe *pbe;
08281   Vatom *atom;
08282   Vsurf_Meth srfm;
08283
08284   /* Grid variables */
08285   double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
08286   double xlen, ylen, zlen, xmin, ymin, zmin, xmax
08287 , ymax, zmax, rtot2;
08288   double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
08289   double izmagic;
08290   int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
08291
08292   VASSERT(thee != VNULL);
08293
08294   /* Nonlinear PBE is not implemented for AMOEBA */
08295   VASSERT(!thee->pmgp->nonlin);
08296
08297   acc = thee->pbe->acc;
08298   srfm = thee->surfMeth;
08299   atom = Valist_getAtom(thee->pbe->alist, atomID);
08300
08301   /* Currently all atoms must be in the same partition. */
08302
08303   VASSERT(atom->partID != 0);
08304   apos = VatomGetPosition(atom);
08305   arad = Vatom_getRadius(atom);
08306
08307   /* Reset force */
08308   force[0] = 0.0;
08309   force[1] = 0.0;
08310   force[2] = 0.0;
08311
08312   /* Get PBE info */
08313   pbe = thee->pbe;
08314   acc = pbe->acc;
08315   alist = pbe->alist;
08316   irad = Vpbe_getMaxIonRadius(pbe);
08317   zkappa2 = Vpbe_getZkappa2(pbe);
08318   izmagic = 1.0/Vpbe_getZmagic(pbe);
08319
08320   /* Should be a check for this further up. */
08321   VASSERT (zkappa2 > VPMGSALL);
08322
08323   /* Mesh info */
08324   nx = thee->pmgp->nx;
08325   ny = thee->pmgp->ny;
08326   nz = thee->pmgp->nz;
08327   hx = thee->pmgp->hx;
08328   hy = thee->pmgp->hy;
08329   hzed = thee->pmgp->hzed;
08330   xlen = thee->pmgp->xlen;
08331   ylen = thee->pmgp->ylen;
08332   zlen = thee->pmgp->zlen;
08333   xmin = thee->pmgp->xmin;
08334   ymin = thee->pmgp->ymin;
08335   zmin = thee->pmgp->zmin;
08336   xmax = thee->pmgp->xmax;
08337   ymax = thee->pmgp->ymax;
08338   zmax = thee->pmgp->zmax;
08339
08340   /* Make sure we're on the grid */
08341   if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
08342       (apos[1]<=ymin) || (apos[1]>=ymax) || \
08343       (apos[2]<=zmin) || (apos[2]>=zmax)) {

```

```

08342     Vnm_print(2, "ibPermanentMultipoleForce: Atom %d at (%4.3f, %4.3f,
08343     %4.3f) is off the mesh (ignoring):\n", atomID, apos[0], apos[1], apos[2]);
08344     Vnm_print(2, "ibPermanentMultipoleForce: xmin = %g, xmax = %g\n", xmin
08345     , xmax);
08346     Vnm_print(2, "ibPermanentMultipoleForce: ymin = %g, ymax = %g\n", ymin
08347     , ymax);
08348     Vnm_print(2, "ibPermanentMultipoleForce: zmin = %g, zmax = %g\n", zmin
08349     , zmax);
08350     fflush(stderr);
08351 } else {
08352
08353     /* Convert the atom position to grid reference frame */
08354     position[0] = apos[0] - xmin;
08355     position[1] = apos[1] - ymin;
08356     position[2] = apos[2] - zmin;
08357
08358     /* Integrate over points within this atom's (inflated) radius */
08359     rtot = (irad + arad + theee->splineWin);
08360     rtot2 = VSQR(rtot);
08361     dx = rtot + 0.5*hx;
08362     imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
08363     imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
08364     for (i=imin; i<=imax; i++) {
08365         dx2 = VSQR(position[0] - hx*i);
08366         if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
08367         else dy = 0.5*hy;
08368         jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
08369         jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
08370         for (j=jmin; j<=jmax; j++) {
08371             dy2 = VSQR(position[1] - hy*j);
08372             if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
08373             else dz = 0.5*hzed;
08374             kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
08375             kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
08376             for (k=kmin; k<=kmax; k++) {
08377                 dz2 = VSQR(k+hzed - position[2]);
08378                 /* See if grid point is inside ivdw radius and set ccf
08379                 * accordingly (do spline assignment here) */
08380                 if ((dz2 + dy2 + dx2) <= rtot2) {
08381                     gpos[0] = i*hx + xmin;
08382                     gpos[1] = j*hy + ymin;
08383                     gpos[2] = k*hzed + zmin;
08384                     Vpmg_splineSelect(srfm, acc, gpos,
08385                         theee->splineWin, irad, atom, tgrad);
08386                     fmag = VSQR(theee->u[IJK(i,j,k)])*theee->kappa[IJK(
08387                         i,j,k)];
08388                     force[0] += (zkappa2*fmag*tgrad[0]);
08389                     force[1] += (zkappa2*fmag*tgrad[1]);
08390                     force[2] += (zkappa2*fmag*tgrad[2]);
08391                 }
08392             }
08393         }
08394     }
08395 }
08396
08397 VPUBLIC void Vpmg_dbPermanentMultipoleForce(Vpmg
08398     *thee, int atomID,
08399     double force[3]) {
08400     Vacc *acc;
08401     Vpbe *pbe;
08402     Vatom *atom;
08403     Vsurf_Meth srfm;
08404
08405     double *apos, position[3], arad, hx, hy, hzed, izmagic, deps, depsi
08406     ;
08407     double xlen, ylen, zlen, xmin, ymin, zmin, xmax
08408     , ymax, zmax, rtot2, epsp;
08409     double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
08410     double *u, Hxijk, Hyijk, Hzijk, Hximljk, Hyijmlk, Hzijkml;
08411     double dHxijk[3], dHyijk[3], dHzijk[3], dHximljk[3], dHyijmlk[3];
08412     double dHzijkml[3];
08413     int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
08414
08415     VASSERT(thee != VNNULL);

```

```

08414
08415     acc = thee->pbe->acc;
08416     srfm = thee->surfMeth;
08417     atom = Valist_getAtom(thee->pbe->alist, atomID);
08418
08419     /* Currently all atoms must be in the same partition. */
08420
08421     VASSERT(atom->partID != 0);
08422     arad = Vatom_getRadius(atom);
08423     apos = VatomGetPosition(atom);
08424
08425     /* Reset force */
08426     force[0] = 0.0;
08427     force[1] = 0.0;
08428     force[2] = 0.0;
08429
08430     /* Get PBE info */
08431     pbe = thee->pbe;
08432     acc = pbe->acc;
08433     epsp = Vpbe_getSoluteDielectric(pbe);
08434     epsw = Vpbe_getSolventDielectric(pbe);
08435     kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*
08436     Vunit_kb;
08437     izmagic = 1.0/Vpbe_getZmagic(pbe);
08438
08439     deps = (epsw - epsp);
08440     depsi = 1.0/deps;
08441
08442     VASSERT(VABS(deps) > VPMGSMALL);
08443
08444     /* Mesh info */
08445     nx = thee->pmgp->nx;
08446     ny = thee->pmgp->ny;
08447     nz = thee->pmgp->nz;
08448     hx = thee->pmgp->hx;
08449     hy = thee->pmgp->hy;
08450     hzed = thee->pmgp->hzed;
08451     xlen = thee->pmgp->xlen;
08452     ylen = thee->pmgp->ylen;
08453     zlen = thee->pmgp->zlen;
08454     xmin = thee->pmgp->xmin;
08455     ymin = thee->pmgp->ymin;
08456     zmin = thee->pmgp->zmin;
08457     xmax = thee->pmgp->xmax;
08458     ymax = thee->pmgp->ymax;
08459     zmax = thee->pmgp->zmax;
08460     u = thee->u;
08461
08462     /* Make sure we're on the grid */
08463     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
08464         (apos[1]<=ymin) || (apos[1]>=ymax) || \
08465         (apos[2]<=zmin) || (apos[2]>=zmax)) {
08466         Vnm_print(2, "dbPermanentMultipoleForce: Atom at (%4.3f, %4.3f, %4.3f)\n"
08467             "is off the mesh (ignoring):\n", apos[0], apos[1], apos[2]);
08468         Vnm_print(2, "dbPermanentMultipoleForce: xmin = %g, xmax = %g\n", xmin
08469             , xmax);
08470         Vnm_print(2, "dbPermanentMultipoleForce: ymin = %g, ymax = %g\n", ymin
08471             , ymax);
08472         Vnm_print(2, "dbPermanentMultipoleForce: zmin = %g, zmax = %g\n", zmin
08473             , zmax);
08474         fflush(stderr);
08475     } else {
08476
08477         /* Convert the atom position to grid reference frame */
08478         position[0] = apos[0] - xmin;
08479         position[1] = apos[1] - ymin;
08480         position[2] = apos[2] - zmin;
08481
08482         /* Integrate over points within this atom's (inflated) radius */
08483         rtot = (arad + thee->splineWin);
08484         rtot2 = VSQR(rtot);
08485         dx = rtot/hx;
08486         imin = (int)floor((position[0]-rtot)/hx);
08487         if (imin < 1) {
08488             Vnm_print(2, "dbPermanentMultipoleForce: Atom off grid!\n");
08489             return;
08490         }
08491         imax = (int)ceil((position[0]+rtot)/hx);
08492         if (imax > (nx-2)) {
08493             Vnm_print(2, "dbPermanentMultipoleForce: Atom off grid!\n");

```

```

08490         return;
08491     }
08492     jmin = (int)floor((position[1]-rtot)/hy);
08493     if (jmin < 1) {
08494         Vnm_print(2, "dbPermanentMultipoleForce: Atom off grid!\n");
08495         return;
08496     }
08497     jmax = (int)ceil((position[1]+rtot)/hy);
08498     if (jmax > (ny-2)) {
08499         Vnm_print(2, "dbPermanentMultipoleForce: Atom off grid!\n");
08500         return;
08501     }
08502     kmin = (int)floor((position[2]-rtot)/hzed);
08503     if (kmin < 1) {
08504         Vnm_print(2, "dbPermanentMultipoleForce: Atom off grid!\n");
08505         return;
08506     }
08507     kmax = (int)ceil((position[2]+rtot)/hzed);
08508     if (kmax > (nz-2)) {
08509         Vnm_print(2, "dbPermanentMultipoleForce: Atom off grid!\n");
08510         return;
08511     }
08512     for (i=imin; i<imax; i++) {
08513         for (j=jmin; j<=jmax; j++) {
08514             for (k=kmin; k<=kmax; k++) {
08515                 /* i,j,k */
08516                 gpos[0] = (i+0.5)*hx + xmin;
08517                 gpos[1] = j*hy + ymin;
08518                 gpos[2] = k*hzed + zmin;
08519                 Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
08520                 Vpmg_splineSelect(srfm, acc, gpos, thee->
08521                     splineWin, 0.,
08522                     atom, dHxijk);
08523                 for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
08524                 gpos[0] = i*hx + xmin;
08525                 gpos[1] = j*hy + ymin;
08526                 gpos[2] = k*hzed + zmin;
08527                 Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
08528                 Vpmg_splineSelect(srfm, acc, gpos, thee->
08529                     splineWin, 0.,
08530                     atom, dHyijk);
08531                 for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
08532                 gpos[0] = i*hx + xmin;
08533                 gpos[1] = j*hy + ymin;
08534                 gpos[2] = k*hzed + zmin;
08535                 Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
08536                 Vpmg_splineSelect(srfm, acc, gpos, thee->
08537                     splineWin, 0.,
08538                     atom, dHzijk);
08539                 for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
08540                 /* i-1,j,k */
08541                 gpos[0] = (i-0.5)*hx + xmin;
08542                 gpos[1] = j*hy + ymin;
08543                 gpos[2] = k*hzed + zmin;
08544                 Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
08545                 Vpmg_splineSelect(srfm, acc, gpos, thee->
08546                     splineWin, 0.,
08547                     atom, dHxim1jk);
08548                 for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
08549                 /* i,j-1,k */
08550                 gpos[0] = i*hx + xmin;
08551                 gpos[1] = (j-0.5)*hy + ymin;
08552                 gpos[2] = k*hzed + zmin;
08553                 Hyijmlk = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
08554                 Vpmg_splineSelect(srfm, acc, gpos, thee->
08555                     splineWin, 0.,
08556                     atom, dHyijmlk);
08557                 for (l=0; l<3; l++) dHyijmlk[l] *= Hyijmlk;
08558                 /* i,j,k-1 */
08559                 gpos[0] = i*hx + xmin;
08560                 gpos[1] = j*hy + ymin;
08561                 gpos[2] = (k-0.5)*hzed + zmin;
08562                 Hzijkml = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
08563                 Vpmg_splineSelect(srfm, acc, gpos, thee->
08564                     splineWin, 0.,
08565                     atom, dHzijkml);
08566                 for (l=0; l<3; l++) dHzijkml[l] *= Hzijkml;
08567                 dbFmag = u[IJK(i,j,k)];
08568                 tgrad[0] =
08569                     (dHxijk[0] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
08570                     + dHxim1jk[0]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)

```

```

08565     + (dHyijk[0] * (u[IJK(i, j+1, k)]-u[IJK(i, j, k)])
08566     + (dHyijm1k[0]* (u[IJK(i, j+1, k)]-u[IJK(i, j, k))]) /VSQR(hy)
08567     + (dHzijk[0] * (u[IJK(i, j, k+1)]-u[IJK(i, j, k))])
08568     + (dHzijkml[0]* (u[IJK(i, j, k-1)]-u[IJK(i, j, k))]) /VSQR(hzed);
08569 tgrad[1] =
08570     (dHxijk[1] * (u[IJK(i+1, j, k)]-u[IJK(i, j, k)])
08571     + (dHxim1jk[1]* (u[IJK(i-1, j, k)]-u[IJK(i, j, k))]) /VSQR(hx)
08572     + (dHyijk[1] * (u[IJK(i, j+1, k)]-u[IJK(i, j, k))])
08573     + (dHyijm1k[1]* (u[IJK(i, j-1, k)]-u[IJK(i, j, k))]) /VSQR(hy)
08574     + (dHzijk[1] * (u[IJK(i, j, k+1)]-u[IJK(i, j, k))])
08575     + (dHzijkml[1]* (u[IJK(i, j, k-1)]-u[IJK(i, j, k))]) /VSQR(hzed);
08576 tgrad[2] =
08577     (dHxijk[2] * (u[IJK(i+1, j, k)]-u[IJK(i, j, k)])
08578     + (dHxim1jk[2]* (u[IJK(i-1, j, k)]-u[IJK(i, j, k))]) /VSQR(hx)
08579     + (dHyijk[2] * (u[IJK(i, j+1, k)]-u[IJK(i, j, k))])
08580     + (dHyijm1k[2]* (u[IJK(i, j-1, k)]-u[IJK(i, j, k))]) /VSQR(hy)
08581     + (dHzijk[2] * (u[IJK(i, j, k+1)]-u[IJK(i, j, k))])
08582     + (dHzijkml[2]* (u[IJK(i, j, k-1)]-u[IJK(i, j, k))]) /VSQR(hzed);
08583     force[0] += (dbFmag*tgrad[0]);
08584     force[1] += (dbFmag*tgrad[1]);
08585     force[2] += (dbFmag*tgrad[2]);
08586     } /* k loop */
08587     } /* j loop */
08588 } /* i loop */
08589 force[0] = -force[0]*hx*hy*hzed*deps*0.5*izmagic;
08590 force[1] = -force[1]*hx*hy*hzed*deps*0.5*izmagic;
08591 force[2] = -force[2]*hx*hy*hzed*deps*0.5*izmagic;
08592 }
08593 }
08594
08595 VPUBLIC void Vpmg_qfDirectPolForce(Vpmg *thee, Vgrid
08596     * perm, Vgrid *induced,
08597     int atomID, double force[3], double torque[3]
08598 ) {
08599     Vatom *atom;
08600     Vpbe *pbe;
08601     double f, fp, *u, *up, *apos, position[3];
08602     /* Grid variables */
08603     int nx,ny,nz;
08604     double xlabel, ylabel, zlabel, xmin, ymin, zmin, xmax
08605     , ymax, zmax;
08606     double hx, hy, hzed, ifloat, jfloat, kfloat;
08607     /* B-spline weights */
08608     double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz, d3mx, d3my, d3mz;
08609     double mi, mj, mk;
08610
08611     /* Loop indeces */
08612     int i, j, k, ii, jj, kk;
08613     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
08614
08615     /* Permanent potential, field, field gradient and 2nd field gradient */
08616     double pot, e[3], de[3][3], d2e[3][3][3];
08617     /* Induced dipole field */
08618     double dep[3][3];
08619
08620     /* Permanent multipole components */
08621     double *dipole, *quad;
08622     double c, ux, uy, uz, qx, qyy, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
08623     double uix, uiy, uiz;
08624
08625     VASSERT(thee != VNULL);
08626     VASSERT(induced != VNULL); /* the potential due to permanent multipoles.*/
08627     VASSERT(induced != VNULL); /* the potential due to local induced dipoles.*/
08628     VASSERT(thee->pbe != VNULL);
08629     VASSERT(thee->pbe->alist != VNULL);
08630
08631     atom = Valist_getAtom(thee->pbe->alist, atomID);
08632     VASSERT(atom->partID != 0); /* all atoms must be in the same
08633     partition.*/
08634     apos = VatomGetPosition(atom);
08635
08636     c = Vatom_getCharge(atom);
08637     dipole = Vatom_getDipole(atom);
08638     ux = dipole[0];
08639     uy = dipole[1];
08640     uz = dipole[2];
08641     quad = Vatom_getQuadrupole(atom);
08642     qx = quad[0]/3.0;

```

```

08642     qxy = quad[1]/3.0;
08643     qxz = quad[2]/3.0;
08644     qyx = quad[3]/3.0;
08645     qyy = quad[4]/3.0;
08646     qyz = quad[5]/3.0;
08647     qzx = quad[6]/3.0;
08648     qzy = quad[7]/3.0;
08649     qzz = quad[8]/3.0;
08650
08651     dipole = Vatom_getInducedDipole(atom);
08652     uix = dipole[0];
08653     uiy = dipole[1];
08654     uiz = dipole[2];
08655
08656     /* Reset Field Gradients */
08657     pot = 0.0;
08658     for (i=0;i<3;i++) {
08659         e[i] = 0.0;
08660         for (j=0;j<3;j++) {
08661             de[i][j] = 0.0;
08662             dep[i][j] = 0.0;
08663             for (k=0;k<3;k++) {
08664                 d2e[i][j][k] = 0.0;
08665             }
08666         }
08667     }
08668
08669     /* Mesh info */
08670     nx = theee->pmgp->nx;
08671     ny = theee->pmgp->ny;
08672     nz = theee->pmgp->nz;
08673     hx = theee->pmgp->hx;
08674     hy = theee->pmgp->hy;
08675     hzed = theee->pmgp->hzed;
08676     xlen = theee->pmgp->xlen;
08677     ylen = theee->pmgp->ylen;
08678     zlen = theee->pmgp->zlen;
08679     xmin = theee->pmgp->xmin;
08680     ymin = theee->pmgp->ymin;
08681     zmin = theee->pmgp->zmin;
08682     xmax = theee->pmgp->xmax;
08683     ymax = theee->pmgp->ymax;
08684     zmax = theee->pmgp->zmax;
08685     u = induced->data;
08686     up = perm->data;
08687
08688     /* Make sure we're on the grid */
08689     if ((apos[0]<=(xmin+2*hx)) || (apos[0]>=(xmax-2*hx)) \
08690         || (apos[1]<=(ymin+2*hy)) || (apos[1]>=(ymax-2*hy)) \
08691         || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
08692         Vnm_print(2, "qfDirectPolForce: Atom off the mesh (ignoring) %6.3f
%6.3f %6.3f\n", apos[0], apos[1], apos[2]);
08693         fflush(stderr);
08694     } else {
08695
08696         /* Convert the atom position to grid coordinates */
08697         position[0] = apos[0] - xmin;
08698         position[1] = apos[1] - ymin;
08699         position[2] = apos[2] - zmin;
08700         ifloat = position[0]/hx;
08701         jfloat = position[1]/hy;
08702         kfloat = position[2]/hzed;
08703         ip1 = (int)ceil(ifloat);
08704         ip2 = ip1 + 2;
08705         im1 = (int)floor(ifloat);
08706         im2 = im1 - 2;
08707         jp1 = (int)ceil(jfloat);
08708         jp2 = jp1 + 2;
08709         jm1 = (int)floor(jfloat);
08710         jm2 = jm1 - 2;
08711         kp1 = (int)ceil(kfloat);
08712         kp2 = kp1 + 2;
08713         km1 = (int)floor(kfloat);
08714         km2 = km1 - 2;
08715
08716         /* This step shouldn't be necessary, but it saves nasty debugging
08717          * later on if something goes wrong */
08718         ip2 = VMIN2(ip2,nx-1);
08719         ip1 = VMIN2(ip1,nx-1);
08720         im1 = VMAX2(im1,0);
08721

```

```

08722     im2 = VMAX2(im2,0);
08723     jp2 = VMIN2(jp2,ny-1);
08724     jp1 = VMIN2(jp1,ny-1);
08725     jm1 = VMAX2(jm1,0);
08726     jm2 = VMAX2(jm2,0);
08727     kp2 = VMIN2(kp2,nz-1);
08728     kp1 = VMIN2(kp1,nz-1);
08729     km1 = VMAX2(km1,0);
08730     km2 = VMAX2(km2,0);
08731
08732     for (ii=im2; ii<=ip2; ii++) {
08733         mi = VFCHI4(ii,ifloat);
08734         mx = bspline4(mi);
08735         dmx = dbspline4(mi);
08736         d2mx = d2bspline4(mi);
08737         d3mx = d3bspline4(mi);
08738         for (jj=jm2; jj<=jp2; jj++) {
08739             mj = VFCHI4(jj,jfloat);
08740             my = bspline4(mj);
08741             dmy = dbspline4(mj);
08742             d2my = d2bspline4(mj);
08743             d3my = d3bspline4(mj);
08744             for (kk=km2; kk<=kp2; kk++) {
08745                 mk = VFCHI4(kk,kfloat);
08746                 mz = bspline4(mk);
08747                 dmz = dbspline4(mk);
08748                 d2mz = d2bspline4(mk);
08749                 d3mz = d3bspline4(mk);
08750                 f = u[IJK(ii,jj,kk)];
08751                 fp = up[IJK(ii,jj,kk)];
08752                 /* The potential */
08753                 pot += f*mx*my*mz;
08754                 /* The field */
08755                 e[0] += f*dmx*my*mz/hx;
08756                 e[1] += f*mx*dmy*mz/hy;
08757                 e[2] += f*mx*my*dmz/hzed;
08758                 /* The gradient of the field */
08759                 de[0][0] += f*d2mx*my*mz/(hx*hx);
08760                 de[1][0] += f*dmx*dmy*mz/(hy*hy);
08761                 de[1][1] += f*mx*d2my*mz/(hy*hy);
08762                 de[2][0] += f*dmx*my*dmz/(hx*hzed);
08763                 de[2][1] += f*mx*dmy*dmz/(hy*hzed);
08764                 de[2][2] += f*mx*my*d2mz/(hzed*hzed);
08765                 /* The gradient of the (permanent) field */
08766                 dep[0][0] += fp*d2mx*my*mz/(hx*hx);
08767                 dep[1][0] += fp*dmx*dmy*mz/(hy*hy);
08768                 dep[1][1] += fp*mx*d2my*mz/(hy*hy);
08769                 dep[2][0] += fp*dmx*my*dmz/(hx*hzed);
08770                 dep[2][1] += fp*mx*dmy*dmz/(hy*hzed);
08771                 dep[2][2] += fp*mx*my*d2mz/(hzed*hzed);
08772                 /* The 2nd gradient of the field
08773                 VxVxVa */
08774                 d2e[0][0][0] += f*d3mx*my*mz/(hx*hx*hx);
08775                 d2e[0][0][1] += f*d2mx*dmy*mz/(hx*hy*hx);
08776                 d2e[0][0][2] += f*d2mx*my*dmz/(hx*hx*hzed);
08777                 /* VyVxVa */
08778                 d2e[1][0][0] += f*d2mx*dmy*mz/(hx*hx*hy);
08779                 d2e[1][0][1] += f*dmx*d2my*mz/(hx*hy*hy);
08780                 d2e[1][0][2] += f*dmx*dmy*dmz/(hx*hy*hzed);
08781                 /* VyVyVa */
08782                 d2e[1][1][0] += f*dmx*d2my*mz/(hx*hy*hy);
08783                 d2e[1][1][1] += f*mx*d3my*mz/(hy*hy*hy);
08784                 d2e[1][1][2] += f*mx*d2my*dmz/(hy*hy*hzed);
08785                 /* VzVxVa */
08786                 d2e[2][0][0] += f*d2mx*my*dmz/(hx*hx*hzed);
08787                 d2e[2][0][1] += f*dmx*dmy*dmz/(hx*hy*hzed);
08788                 d2e[2][0][2] += f*dmx*my*d2mz/(hx*hzed*hzed);
08789                 /* VzVyVa */
08790                 d2e[2][1][0] += f*dmx*dmy*dmz/(hx*hy*hzed);
08791                 d2e[2][1][1] += f*mx*d2my*dmz/(hy*hy*hzed);
08792                 d2e[2][1][2] += f*mx*dmy*d2mz/(hy*hzed*hzed);
08793                 /* VzVzVa */
08794                 d2e[2][2][0] += f*dmx*my*d2mz/(hx*hzed*hzed);
08795                 d2e[2][2][1] += f*mx*dmy*d2mz/(hy*hzed*hzed);
08796                 d2e[2][2][2] += f*mx*my*d3mz/(hzed*hzed*hzed);
08797             }
08798         }
08799     }
08800 }
08801
08802 /* force on permanent multipole due to induced reaction field */

```

```

08803
08804     /* Monopole Force */
08805     force[0] = e[0]*c;
08806     force[1] = e[1]*c;
08807     force[2] = e[2]*c;
08808
08809     /* Dipole Force */
08810     force[0] -= de[0][0]*ux+de[1][0]*uy+de[2][0]*uz;
08811     force[1] -= de[1][0]*ux+de[1][1]*uy+de[2][1]*uz;
08812     force[2] -= de[2][0]*ux+de[2][1]*uy+de[2][2]*uz;
08813
08814     /* Quadrupole Force */
08815     force[0] += d2e[0][0][0]*qxx
08816         + d2e[1][0][0]*qyx*2.0+d2e[1][1][0]*qyy
08817         + d2e[2][0][0]*qzx*2.0+d2e[2][1][0]*qzy*2.0+d2e[2][2][0]*qzz;
08818     force[1] += d2e[0][0][1]*qxx
08819         + d2e[1][0][1]*qyx*2.0+d2e[1][1][1]*qyy
08820         + d2e[2][0][1]*qzx*2.0+d2e[2][1][1]*qzy*2.0+d2e[2][2][1]*qzz;
08821     force[2] += d2e[0][0][2]*qxx
08822         + d2e[1][0][2]*qyx*2.0+d2e[1][1][2]*qyy
08823         + d2e[2][0][2]*qzx*2.0+d2e[2][1][2]*qzy*2.0+d2e[2][2][2]*qzz;
08824
08825     /* torque on permanent mulitpole due to induced reaction field */
08826
08827     /* Dipole Torque */
08828     torque[0] = uy * e[2] - uz * e[1];
08829     torque[1] = uz * e[0] - ux * e[2];
08830     torque[2] = ux * e[1] - uy * e[0];
08831
08832     /* Quadrupole Torque */
08833     /* Tx = -2.0*(Sum_a (Qya*dEaz) + Sum_b (Qzb*dEby))
08834     Ty = -2.0*(Sum_a (Qza*dEax) + Sum_b (Qxb*dEbz))
08835     Tz = -2.0*(Sum_a (Qxa*dEay) + Sum_b (Qyb*dEbx)) */
08836     de[0][1] = de[1][0];
08837     de[0][2] = de[2][0];
08838     de[1][2] = de[2][1];
08839     torque[0] -= 2.0*(qyx*de[0][2] + qyy*de[1][2] + qyz*de[2][2]
08840         - qzx*de[0][1] - qzy*de[1][1] - qzz*de[2][1]);
08841     torque[1] -= 2.0*(qzx*de[0][0] + qzy*de[1][0] + qzz*de[2][0]
08842         - qxx*de[0][2] - qxy*de[1][2] - qxz*de[2][2]);
08843     torque[2] -= 2.0*(qxx*de[0][1] + qxy*de[1][1] + qxz*de[2][1]
08844         - qyx*de[0][0] - qyy*de[1][0] - qyz*de[2][0]);
08845
08846     /* force on induced dipole due to permanent reaction field */
08847
08848     force[0] -= dep[0][0]*uix+dep[1][0]*uiy+dep[2][0]*uiz;
08849     force[1] -= dep[1][0]*uix+dep[1][1]*uiy+dep[2][1]*uiz;
08850     force[2] -= dep[2][0]*uix+dep[2][1]*uiy+dep[2][2]*uiz;
08851
08852     force[0] = 0.5 * force[0];
08853     force[1] = 0.5 * force[1];
08854     force[2] = 0.5 * force[2];
08855     torque[0] = 0.5 * torque[0];
08856     torque[1] = 0.5 * torque[1];
08857     torque[2] = 0.5 * torque[2];
08858
08859     /* printf(" qPhi Force %f %f %f\n", force[0], force[1], force[2]);
08860     printf(" qPhi Torque %f %f %f\n", torque[0], torque[1], torque[2]); */
08861 }
08862
08863 VPUBLIC void Vpmg_qfNLDirectPolForce(Vpmg *thee,
08864     Vgrid *perm, Vgrid *nlInduced,
08865             int atomID, double force[3], double torque
08866 [3]) {
08867     Vatom *atom;
08868     double *apos, *dipole, *quad, position[3], hx, hy, hzed;
08869     double xlen, ylen, zlen, xmin, ymin, zmin, xmax
08870     , ymax, zmax;
08871     double pot, e[3], de[3][3], dep[3][3], d2e[3][3][3];
08872     double mx, my, mz, dmx, dmy, dmz, mi, mj, mk;
08873     double d2mx, d2my, d2mz, d3mx, d3my, d3mz;
08874     double *u, *up, charge, ifloat, jfloat, kffloat;
08875     double f, fp, c, ux, uy, uz, qxx, qxy, qzx, qyy, qyz, qzx, qzy, qzz;
08876     double uix, uiy, uiz;
08877     int i, j, k, nx, ny, nz, im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2,
08878     km1;
08879     int kp1, kp2, ii, jj, kk;
08880
08881     VASSERT(thee != VNULL);
08882     VASSERT(perm != VNULL);      /* potential due to permanent multipoles. */

```

```

08880     VASSERT(nlInduced != VNULL); /* potential due to non-local induced dipoles
08881     */
08882     VASSERT(!thee->pmgp->nonlin); /* Nonlinear PBE is not implemented
08883     for AMOEBA */
08884     atom = Valist_getAtom(thee->pbe->alist, atomID);
08885     VASSERT(atom->partID != 0); /* Currently all atoms must be in the
08886     same partition. */
08887     apos = Vatom_getPosition(atom);
08888     c = Vatom_getCharge(atom);
08889     dipole = Vatom_getDipole(atom);
08890     ux = dipole[0];
08891     uy = dipole[1];
08892     uz = dipole[2];
08893     quad = Vatom_getQuadrupole(atom);
08894     qxx = quad[0]/3.0;
08895     qxy = quad[1]/3.0;
08896     qxz = quad[2]/3.0;
08897     qyx = quad[3]/3.0;
08898     qyy = quad[4]/3.0;
08899     qyz = quad[5]/3.0;
08900     qzx = quad[6]/3.0;
08901     qzy = quad[7]/3.0;
08902     qzz = quad[8]/3.0;
08903     dipole = Vatom_getNLInducedDipole(atom);
08904     uix = dipole[0];
08905     uiy = dipole[1];
08906     uiz = dipole[2];
08907
08908     /* Reset Field Gradients */
08909     pot = 0.0;
08910     for (i=0;i<3;i++){
08911         e[i] = 0.0;
08912         for (j=0;j<3;j++) {
08913             de[i][j] = 0.0;
08914             dep[i][j] = 0.0;
08915             for (k=0;k<3;k++) {
08916                 d2e[i][j][k] = 0.0;
08917             }
08918         }
08919     }
08920
08921     /* Mesh info */
08922     nx = thee->pmgp->nx;
08923     ny = thee->pmgp->ny;
08924     nz = thee->pmgp->nz;
08925     hx = thee->pmgp->hx;
08926     hy = thee->pmgp->hy;
08927     hzed = thee->pmgp->hzed;
08928     xlen = thee->pmgp->xlen;
08929     ylen = thee->pmgp->ylen;
08930     zlen = thee->pmgp->zlen;
08931     xmin = thee->pmgp->xmin;
08932     ymin = thee->pmgp->ymin;
08933     zmin = thee->pmgp->zmin;
08934     xmax = thee->pmgp->xmax;
08935     ymax = thee->pmgp->ymax;
08936     zmax = thee->pmgp->zmax;
08937     u = nlInduced->data;
08938     up = perm->data;
08939
08940
08941     /* Make sure we're on the grid */
08942     if ((apos[0]<=(xmin+2*hx)) || (apos[0]>=(xmax-2*hx)) \
08943         || (apos[1]<=(ymin+2*hy)) || (apos[1]>=(ymax-2*hy)) \
08944         || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
08945         Vnm_Print(2, "qfNLDirectMultipoleForce: Atom off the mesh (ignoring)
%6.3f %6.3f %6.3f\n", apos[0], apos[1], apos[2]);
08946     } else {
08947
08948         /* Convert the atom position to grid coordinates */
08949         position[0] = apos[0] - xmin;
08950         position[1] = apos[1] - ymin;
08951         position[2] = apos[2] - zmin;
08952         ifloat = position[0]/hx;
08953         jfloat = position[1]/hy;
08954         kfloat = position[2]/hzed;
08955         ip1 = (int)ceil(ifloat);
08956         ip2 = ip1 + 2;

```

```

08957     im1 = (int)floor(ifloat);
08958     im2 = im1 - 2;
08959     jp1 = (int)ceil(jfloat);
08960     jp2 = jp1 + 2;
08961     jm1 = (int)floor(jfloat);
08962     jm2 = jm1 - 2;
08963     kp1 = (int)ceil(kfloat);
08964     kp2 = kp1 + 2;
08965     km1 = (int)floor(kfloat);
08966     km2 = km1 - 2;
08967
08968     /* This step shouldn't be necessary, but it saves nasty debugging
08969      * later on if something goes wrong */
08970     ip2 = VMIN2(ip2,nx-1);
08971     ip1 = VMIN2(ip1,nx-1);
08972     im1 = VMAX2(im1,0);
08973     im2 = VMAX2(im2,0);
08974     jp2 = VMIN2(jp2,ny-1);
08975     jp1 = VMIN2(jp1,ny-1);
08976     jm1 = VMAX2(jm1,0);
08977     jm2 = VMAX2(jm2,0);
08978     kp2 = VMIN2(kp2,nz-1);
08979     kp1 = VMIN2(kp1,nz-1);
08980     km1 = VMAX2(km1,0);
08981     km2 = VMAX2(km2,0);
08982
08983     for (ii=im2; ii<=ip2; ii++) {
08984         mi = VFCHI4(ii,ifloat);
08985         mx = bspline4(mi);
08986         dmx = dbspline4(mi);
08987         d2mx = d2bspline4(mi);
08988         d3mx = d3bspline4(mi);
08989         for (jj=jm2; jj<=jp2; jj++) {
08990             mj = VFCHI4(jj,jfloat);
08991             my = bspline4(mj);
08992             dmy = dbspline4(mj);
08993             d2my = d2bspline4(mj);
08994             d3my = d3bspline4(mj);
08995             for (kk=km2; kk<=kp2; kk++) {
08996                 mk = VFCHI4(kk,kfloat);
08997                 mz = bspline4(mk);
08998                 dmz = dbspline4(mk);
08999                 d2mz = d2bspline4(mk);
09000                 d3mz = d3bspline4(mk);
09001                 f = u[IJK(ii,jj,kk)];
09002                 fp = up[IJK(ii,jj,kk)];
09003                 /* The potential */
09004                 pot += f*mx*my*mz;
09005                 /* The field */
09006                 e[0] += f*dmx*my*mz/hx;
09007                 e[1] += f*mx*dmy*mz/hy;
09008                 e[2] += f*mx*my*dmz/hzed;
09009                 /* The gradient of the field */
09010                 de[0][0] += f*d2mx*my*mz/(hx*hx);
09011                 de[1][0] += f*dmx*dmy*mz/(hy*hx);
09012                 de[1][1] += f*mx*d2my*mz/(hy*hy);
09013                 de[2][0] += f*dmx*my*dmz/(hx*hzed);
09014                 de[2][1] += f*mx*dmy*dmz/(hy*hzed);
09015                 de[2][2] += f*mx*my*d2mz/(hzed*hzed);
09016                 /* The gradient of the (permanent) field */
09017                 dep[0][0] += fp*d2mx*my*mz/(hx*hx);
09018                 dep[1][0] += fp*dmx*dmy*mz/(hy*hx);
09019                 dep[1][1] += fp*mx*d2my*mz/(hy*hy);
09020                 dep[2][0] += fp*dmx*my*dmz/(hx*hzed);
09021                 dep[2][1] += fp*mx*dmy*dmz/(hy*hzed);
09022                 dep[2][2] += fp*mx*my*d2mz/(hzed*hzed);
09023                 /* The 2nd gradient of the field */
09024                 /* VxVxVa */
09025                 d2e[0][0][0] += f*d3mx*my*mz/(hx*hx*hx);
09026                 d2e[0][0][1] += f*d2mx*dmy*mz/(hx*hy*hx);
09027                 d2e[0][0][2] += f*d2mx*my*dmz/(hx*hx*hzed);
09028                 /* VyVxVa */
09029                 d2e[1][0][0] += f*d2mx*dmy*mz/(hx*hx*hy);
09030                 d2e[1][0][1] += f*dmx*d2my*mz/(hx*hy*hy);
09031                 d2e[1][0][2] += f*dmx*dmy*dmz/(hx*hy*hzed);
09032                 /* VyVyVa */
09033                 d2e[1][1][0] += f*dmx*d2my*mz/(hx*hy*hy);
09034                 d2e[1][1][1] += f*mx*d3my*mz/(hy*hy*hy);
09035                 d2e[1][1][2] += f*mx*d2my*dmz/(hy*hy*hzed);
09036                 /* VzVxVa */
09037                 d2e[2][0][0] += f*d2mx*my*dmz/(hx*hx*hzed);

```

```

09038      d2e[2][0][1] += f*dmx*dmy*dmz / (hx*hy*hzed);
09039      d2e[2][0][2] += f*dmx*my*d2mz / (hx*hzed*hzed);
09040      /* VzVyVa */
09041      d2e[2][1][0] += f*dmx*dmy*dmz / (hx*hy*hzed);
09042      d2e[2][1][1] += f*mx*d2my*dmz / (hy*hy*hzed);
09043      d2e[2][1][2] += f*mx*dmy*d2mz / (hy*hzed*hzed);
09044      /* VzVzVa */
09045      d2e[2][2][0] += f*dmx*my*d2mz / (hx*hzed*hzed);
09046      d2e[2][2][1] += f*mx*dmy*d2mz / (hy*hzed*hzed);
09047      d2e[2][2][2] += f*mx*my*d3mz / (hzed*hzed*hzed);
09048
09049 }
09050 }
09051 }
09052
09053 /* force on permanent multipole due to non-local induced reaction field */
09054
09055 /* Monopole Force */
09056 force[0] = e[0]*c;
09057 force[1] = e[1]*c;
09058 force[2] = e[2]*c;
09059
09060 /* Dipole Force */
09061 force[0] -= de[0][0]*ux+de[1][0]*uy+de[2][0]*uz;
09062 force[1] -= de[1][0]*ux+de[1][1]*uy+de[2][1]*uz;
09063 force[2] -= de[2][0]*ux+de[2][1]*uy+de[2][2]*uz;
09064
09065 /* Quadrupole Force */
09066 force[0] += d2e[0][0][0]*qxx
09067     + d2e[1][0][0]*qyx*2.0+d2e[1][1][0]*qyy
09068     + d2e[2][0][0]*qzx*2.0+d2e[2][1][0]*qzy*2.0+d2e[2][2][0]*qzz;
09069 force[1] += d2e[0][0][1]*qxx
09070     + d2e[1][0][1]*qyx*2.0+d2e[1][1][1]*qyy
09071     + d2e[2][0][1]*qzx*2.0+d2e[2][1][1]*qzy*2.0+d2e[2][2][1]*qzz;
09072 force[2] += d2e[0][0][2]*qxx
09073     + d2e[1][0][2]*qyx*2.0+d2e[1][1][2]*qyy
09074     + d2e[2][0][2]*qzx*2.0+d2e[2][1][2]*qzy*2.0+d2e[2][2][2]*qzz;
09075
09076 /* torque on permanent multipole due to non-local induced reaction field */
09077
09078 /* Dipole Torque */
09079 torque[0] = uy * e[2] - uz * e[1];
09080 torque[1] = uz * e[0] - ux * e[2];
09081 torque[2] = ux * e[1] - uy * e[0];
09082
09083 /* Quadrupole Torque */
09084 /* Tx = -2.0*(Sum_a (Qya*dEaz) + Sum_b (Qzb*dEby))
09085   Ty = -2.0*(Sum_a (Qza*dEax) + Sum_b (Qxb*dEbz))
09086   Tz = -2.0*(Sum_a (Qxa*dEay) + Sum_b (Qyb*dEbxy)) */
09087 de[0][1] = de[1][0];
09088 de[0][2] = de[2][0];
09089 de[1][2] = de[2][1];
09090 torque[0] -= 2.0*(qyx*de[0][2] + qyy*de[1][2] + qyz*de[2][2]
09091     - qzx*de[0][1] - qzy*de[1][1] - qzz*de[2][1]);
09092 torque[1] -= 2.0*(qzx*de[0][0] + qzy*de[1][0] + qzz*de[2][0]
09093     - qxx*de[0][2] - qxy*de[1][2] - qxz*de[2][2]);
09094 torque[2] -= 2.0*(qxx*de[0][1] + qxy*de[1][1] + qxz*de[2][1]
09095     - qyx*de[0][0] - qyy*de[1][0] - qyz*de[2][0]);
09096
09097 /* force on non-local induced dipole due to permanent reaction field */
09098
09099 force[0] -= dep[0][0]*uix+dep[1][0]*uiy+dep[2][0]*uiz;
09100 force[1] -= dep[1][0]*uix+dep[1][1]*uiy+dep[2][1]*uiz;
09101 force[2] -= dep[2][0]*uix+dep[2][1]*uiy+dep[2][2]*uiz;
09102
09103 force[0] = 0.5 * force[0];
09104 force[1] = 0.5 * force[1];
09105 force[2] = 0.5 * force[2];
09106 torque[0] = 0.5 * torque[0];
09107 torque[1] = 0.5 * torque[1];
09108 torque[2] = 0.5 * torque[2];
09109
09110 /* printf(" qPhi Force %f %f %f\n", force[0], force[1], force[2]);
09111   printf(" qPhi Torque %f %f %f\n", torque[0], torque[1], torque[2]); */
09112 }
09113
09114 VPUBLIC void Vpmg_ibDirectPolForce(Vpmg *thee, Vgrid
09115   *perm, Vgrid *induced,
09116           int atomID, double force[3]) {
09117   Vatom *atom;

```

```

09118     Valist *alist;
09119     Vacc *acc;
09120     Vpbe *pbe;
09121     Vsurf_Meth srfm;
09122
09123     double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
09124     double xlabel, ylabel, zlabel, xmin, ymin, zmin, xmax
09125 , ymax, zmax, rtot2;
09126     double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
09127     double izmagic;
09128     int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
09129
09130     VASSERT(thee != VNULL);
09131     VASSERT(perm != VNULL); /* potential due to permanent multipoles.*/
09132     VASSERT(induced != VNULL); /* potential due to induced dipoles. */
09133     VASSERT (!thee->pmpg->nonlin); /* Nonlinear PBE is not
09134     implemented for AMOEBA */
09135
09136     acc = thee->pbe->acc;
09137     srfm = thee->surfMeth;
09138     atom = Valist_getAtom(thee->pbe->alist, atomID);
09139     VASSERT(atom->partID != 0); /* Currently all atoms must be in the
09140     same partition. */
09141     apos = Vatom_getPosition(atom);
09142     arad = Vatom_getRadius(atom);
09143
09144     /* Reset force */
09145     force[0] = 0.0;
09146     force[1] = 0.0;
09147     force[2] = 0.0;
09148
09149     /* Get PBE info */
09150     pbe = thee->pbe;
09151     acc = pbe->acc;
09152     alist = pbe->alist;
09153     irad = Vpbe_getMaxIonRadius(pbe);
09154     zkappa2 = Vpbe_getZkappa2(pbe);
09155     izmagic = 1.0/Vpbe_getZmagic(pbe);
09156
09157     /* Mesh info */
09158     nx = induced->nx;
09159     ny = induced->ny;
09160     nz = induced->nz;
09161     hx = induced->hx;
09162     hy = induced->hy;
09163     hzed = induced->hzed;
09164     xmin = induced->xmin;
09165     ymin = induced->ymin;
09166     zmin = induced->zmin;
09167     xmax = induced->xmax;
09168     ymax = induced->ymax;
09169     zmax = induced->zmax;
09170     xlabel = xmax-xmin;
09171     ylabel = ymax-ymin;
09172     zlabel = zmax-zmin;
09173
09174     /* Make sure we're on the grid */
09175     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
09176         (apos[1]<=ymin) || (apos[1]>=ymax) || \
09177         (apos[2]<=zmin) || (apos[2]>=zmax)) {
09178         Vnm_print(2, "Vpmg_ibForce: Atom at (%4.3f, %4.3f, %4.3f) is off the
09179         mesh (ignoring):\n",
09180             apos[0], apos[1], apos[2]);
09181         Vnm_print(2, "Vpmg_ibForce: xmin = %g, xmax = %g\n", xmin, xmax);
09182         Vnm_print(2, "Vpmg_ibForce: ymin = %g, ymax = %g\n", ymin, ymax);
09183         Vnm_print(2, "Vpmg_ibForce: zmin = %g, zmax = %g\n", zmin, zmax);
09184         fflush(stderr);
09185     } else {
09186
09187         /* Convert the atom position to grid reference frame */
09188         position[0] = apos[0] - xmin;
09189         position[1] = apos[1] - ymin;
09190         position[2] = apos[2] - zmin;
09191
09192         /* Integrate over points within this atom's (inflated) radius */
09193         rtot = (irad + arad + thee->splineWin);
09194         rtot2 = VSQR(rtot);
09195         dx = rtot + 0.5*hx;

```

```

09194     imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
09195     imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
09196     for (i=imin; i<=imax; i++) {
09197         dx2 = VSQR(position[0] - hx*i);
09198         if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
09199         else dy = 0.5*hy;
09200         jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
09201         jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
09202         for (j=jmin; j<=jmax; j++) {
09203             dy2 = VSQR(position[1] - hy*j);
09204             if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
09205             else dz = 0.5*hzed;
09206             kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
09207             kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
09208             for (k=kmin; k<=kmax; k++) {
09209                 dz2 = VSQR(k*hzed - position[2]);
09210                 /* See if grid point is inside ivdw radius and set ccf
09211                  * accordingly (do spline assignment here) */
09212                 if ((dz2 + dy2 + dx2) <= rtot2) {
09213                     gpos[0] = i*hx + xmin;
09214                     gpos[1] = j*hy + ymin;
09215                     gpos[2] = k*hzed + zmin;
09216                     Vpmg_splineSelect(srftm, acc, gpos,
09217                         theee->splineWin, irad,
09218                             atom, tgrad);
09219                     fmag = induced->data[IJK(i,j,k)];
09220                     fmag *= perm->data[IJK(i,j,k)];
09221                     fmag *= theee->kappa[IJK(i,j,k)];
09222                     force[0] += (zkappa2*fmag*tgrad[0]);
09223                     force[1] += (zkappa2*fmag*tgrad[1]);
09224                     force[2] += (zkappa2*fmag*tgrad[2]);
09225                 }
09226             } /* k loop */
09227         } /* j loop */
09228     } /* i loop */
09229 }
09230 force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
09231 force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
09232 force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
09233 }
09234 }
09235 VPUBLIC void Vpmg_ibNLDirectPolForce(Vpmg *thee,
09236     Vgrid *perm, Vgrid *nlInduced,
09237             int atomID, double force[3]) {
09238     Vpmg_ibDirectPolForce(thee, perm, nlInduced, atomID,
09239     force);
09240 }
09241 VPUBLIC void Vpmg_dbDirectPolForce(Vpmg *thee, Vgrid
09242     *perm, Vgrid *induced,
09243             int atomID, double force[3]) {
09244     Vatom *atom;
09245     Vacc *acc;
09246     Vpbe *pbe;
09247     Vsurf_Meth srftm;
09248
09249     double *apos, position[3], arad, hx, hy, hzed, izmagic, deps, depsi
09250 ;
09251     double xlabel, ylabel, zlabel, xlabel_min, ylabel_min, zlabel_min, xlabel_max
09252 , ylabel_max, zlabel_max, rtot2, epsp;
09253     double rtot, dx, gpos[3], dbFmag, epsw, kT;
09254     double *u, *up, Hxijk, Hyijk, Hzijk, Hximljk, Hyijmlk, Hzijkml;
09255     double dHxijk[3], dHyijk[3], dHzijk[3], dHximljk[3], dHyijmlk[3];
09256     double dHzijkml[3];
09257     int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
09258     VASSERT(thee != VNULL);
09259     VASSERT(perm != VNULL); /* permanent multipole PMG solution. */
09260     VASSERT(induced != VNULL); /* potential due to induced dipoles. */
09261
09262     acc = thee->pbe->acc;
09263     atom = Valist_getAtom(thee->pbe->alist, atomID);
09264     VASSERT (atom->partID != 0); /* Currently all atoms must be in the
09265 same partition. */
09266     apos = Vatom_getPosition(atom);
09267     arad = Vatom_getRadius(atom);
09268
09269     /* Reset force */

```

```

09268     force[0] = 0.0;
09269     force[1] = 0.0;
09270     force[2] = 0.0;
09271
09272     /* Get PBE info */
09273     pbe = theee->pbe;
09274     acc = pbe->acc;
09275     srfm = theee->surfMeth;
09276     epsp = Vpbe_getSoluteDiel(pbe);
09277     epsw = Vpbe_getSolventDiel(pbe);
09278     kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*
09279     Vunit_kb;
09280     izmagic = 1.0/Vpbe_getZmagic(pbe);
09281
09282     deps = (epsw - epsp);
09283     depsi = 1.0/deps;
09284     VASSERT(VABS(deps) > VPMGSMALL);
09285
09286     /* Mesh info */
09287     nx = theee->pmgp->nx;
09288     ny = theee->pmgp->ny;
09289     nz = theee->pmgp->nz;
09290     hx = theee->pmgp->hx;
09291     hy = theee->pmgp->hy;
09292     hzed = theee->pmgp->hzed;
09293     xlen = theee->pmgp->xlen;
09294     ylen = theee->pmgp->ylen;
09295     zlen = theee->pmgp->zlen;
09296     xmin = theee->pmgp->xmin;
09297     ymin = theee->pmgp->ymin;
09298     zmin = theee->pmgp->zmin;
09299     xmax = theee->pmgp->xmax;
09300     ymax = theee->pmgp->ymax;
09301     zmax = theee->pmgp->zmax;
09302     /* If the permanent and induced potentials are flipped the
09303      results are exactly the same. */
09304     u = induced->data;
09305     up = perm->data;
09306
09307     /* Make sure we're on the grid */
09308     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
09309         (apos[1]<=ymin) || (apos[1]>=ymax) || \
09310         (apos[2]<=zmin) || (apos[2]>=zmax)) {
09311         Vnm_print(2, "Vpmg_dbDirectPolForce: Atom at (%4.3f, %4.3f, %4.3f) is
09312 off the mesh (ignoring):\n", apos[0], apos[1], apos[2]);
09313         Vnm_print(2, "Vpmg_dbDirectPolForce: xmin = %g, xmax = %g\n", xmin,
09314         xmax);
09315         Vnm_print(2, "Vpmg_dbDirectPolForce: ymin = %g, ymax = %g\n", ymin,
09316         ymax);
09317         Vnm_print(2, "Vpmg_dbDirectPolForce: zmin = %g, zmax = %g\n", zmin,
09318         zmax);
09319         fflush(stderr);
09320     } else {
09321
09322         /* Convert the atom position to grid reference frame */
09323         position[0] = apos[0] - xmin;
09324         position[1] = apos[1] - ymin;
09325         position[2] = apos[2] - zmin;
09326
09327         /* Integrate over points within this atom's (inflated) radius */
09328         rtot = (arad + theee->splineWin);
09329         rtot2 = VSQR(rtot);
09330         dx = rtot/hx;
09331         imin = (int)floor((position[0]-rtot)/hx);
09332         if (imin < 1) {
09333             Vnm_print(2, "Vpmg_dbDirectPolForce: Atom %d off grid!\n", atomID)
09334 ;
09335             return;
09336         }
09337         imax = (int)ceil((position[0]+rtot)/hx);
09338         if (imax > (nx-2)) {
09339             Vnm_print(2, "Vpmg_dbDirectPolForce: Atom %d off grid!\n", atomID)
09340 ;
09341             return;
09342         }
09343         jmin = (int)floor((position[1]-rtot)/hy);
09344         if (jmin < 1) {
09345             Vnm_print(2, "Vpmg_dbDirectPolForce: Atom %d off grid!\n", atomID)
09346 ;
09347             return;
09348         }

```

```

09341     jmax = (int)ceil((position[1]+rtot)/hy);
09342     if (jmax > (ny-2)) {
09343         Vnm_print(2, "Vpmg_dbDirectPolForce: Atom %d off grid!\n", atomID)
09344         ;
09345         return;
09346     kmin = (int)floor((position[2]-rtot)/hzed);
09347     if (kmin < 1) {
09348         Vnm_print(2, "Vpmg_dbDirectPolForce: Atom %d off grid!\n", atomID)
09349         ;
09350         return;
09351     kmax = (int)ceil((position[2]+rtot)/hzed);
09352     if (kmax > (nz-2)) {
09353         Vnm_print(2, "Vpmg_dbDirectPolForce: Atom %d off grid!\n", atomID)
09354         ;
09355         return;
09356     for (i=iimin; i<=imax; i++) {
09357         for (j=jimin; j<=jmax; j++) {
09358             for (k=kmin; k<=kmax; k++) {
09359                 /* i,j,k */
09360                 gpos[0] = (i+0.5)*hx + xmin;
09361                 gpos[1] = j*hy + ymin;
09362                 gpos[2] = k*hzed + zmin;
09363                 Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
09364                 Vpmg_splineSelect(srfm, acc, gpos, thee->
09365                     splineWin, 0.,
09366                     atom, dHxijk);
09367                 for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
09368                 gpos[0] = i*hx + xmin;
09369                 gpos[1] = j*hy + ymin;
09370                 gpos[2] = k*hzed + zmin;
09371                 Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
09372                 Vpmg_splineSelect(srfm, acc, gpos, thee->
09373                     splineWin, 0.,
09374                     atom, dHyijk);
09375                 for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
09376                 gpos[0] = i*hx + xmin;
09377                 gpos[1] = j*hy + ymin;
09378                 gpos[2] = (k+0.5)*hzed + zmin;
09379                 Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
09380                 Vpmg_splineSelect(srfm, acc, gpos, thee->
09381                     splineWin, 0.,
09382                     atom, dHzijk);
09383                 for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
09384                 /* i-1,j,k */
09385                 gpos[0] = (i-0.5)*hx + xmin;
09386                 gpos[1] = j*hy + ymin;
09387                 gpos[2] = k*hzed + zmin;
09388                 Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
09389                 Vpmg_splineSelect(srfm, acc, gpos, thee->
09390                     splineWin, 0.,
09391                     atom, dHxim1jk);
09392                 for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
09393                 /* i,j-1,k */
09394                 gpos[0] = i*hx + xmin;
09395                 gpos[1] = (j-0.5)*hy + ymin;
09396                 gpos[2] = k*hzed + zmin;
09397                 Hyijmlk = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
09398                 Vpmg_splineSelect(srfm, acc, gpos, thee->
09399                     splineWin, 0.,
09400                     atom, dHyijmlk);
09401                 for (l=0; l<3; l++) dHyijmlk[l] *= Hyijmlk;
09402                 /* i,j,k-1 */
09403                 gpos[0] = i*hx + xmin;
09404                 gpos[1] = j*hy + ymin;
09405                 gpos[2] = (k-0.5)*hzed + zmin;
09406                 Hzijkml = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
09407                 Vpmg_splineSelect(srfm, acc, gpos, thee->
09408                     splineWin, 0.,
09409                     atom, dHzijkml);
09410                 for (l=0; l<3; l++) dHzijkml[l] *= Hzijkml;
09411                 dbFmag = up[IJK(i,j,k)];
09412                 tgrad[0] =
09413                     (dHxijk[0] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09414                     + dHxim1jk[0]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09415                     + (dHyijk[0] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)]))
09416                     + (dHyijmlk[0]* (u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09417                     + (dHzijk[0] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)]))

```

```

09413     + dHzijkml[0]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09414 tgrad[1] =
09415     (dHxijk[1]* (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09416     + dHxim1jk[1]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09417     + (dHyijk[1]* (u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09418     + dHyjm1k[1]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09419     + (dHzijk[1]* (u[IJK(i,j,k+1)]-u[IJK(i,j,k)]))/VSQR(hy)
09420     + dHzijkml[1]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09421 tgrad[2] =
09422     (dHxijk[2]* (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09423     + dHxim1jk[2]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09424     + (dHyijk[2]* (u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09425     + dHyjm1k[2]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09426     + (dHzijk[2]* (u[IJK(i,j,k+1)]-u[IJK(i,j,k)]))/VSQR(hy)
09427     + dHzijkml[2]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09428 force[0] += (dbFmag*tgrad[0]);
09429 force[1] += (dbFmag*tgrad[1]);
09430 force[2] += (dbFmag*tgrad[2]);
09431
09432     } /* k loop */
09433 } /* j loop */
09434 } /* i loop */
09435
09436 force[0] = -force[0]*hx*hy*hzed*deps*0.5*izmagic;
09437 force[1] = -force[1]*hx*hy*hzed*deps*0.5*izmagic;
09438 force[2] = -force[2]*hx*hy*hzed*deps*0.5*izmagic;
09439
09440 }
09441 }
09442
09443 VPUBLIC void Vpmg_dbNLDirectPolForce(Vpmg *thee,
09444     Vgrid *perm, Vgrid *nlInduced,
09445         int atomID, double force[3]) {
09446     Vpmg_dbDirectPolForce(thee, perm, nlInduced, atomID,
09447     force);
09448 }
09449
09450 VPUBLIC void Vpmg_qfMutualPolForce(Vpmg *thee, Vgrid
09451     *induced,
09452         Vgrid *nlinduced, int atomID, double force[3]
09453     ]) {
09454
09455     Vatom *atom;
09456     double *apos, *dipole, position[3], hx, hy, hzed;
09457     double *u, *unl;
09458     double xlabel, ylabel, zlabel, xmin, ymin, zmin, xmax
09459     , ymax, zmax;
09460     double de[3][3], denl[3][3];
09461     double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz, mi, mj, mk;
09462     double ifloat, jfloat, kfloat;
09463     double f, fnl, uix, uiy, uiz, uixnl, uiynl, uiznl;
09464     int i, j, k, nx, ny, nz, im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2,
09465     kml;
09466     int kp1, kp2, ii, jj, kk;
09467
09468     VASSERT(thee != VNULL); /* PMG object with PBE info. */
09469     VASSERT(induced != VNULL); /* potential due to induced dipoles. */
09470     VASSERT(nlinduced != VNULL); /* potential due to non-local induced dipoles.
09471 */
09472     atom = Valist_getAtom(thee->pbe->alist, atomID);
09473     VASSERT(atom->partID != 0); /* all atoms must be in the same
09474     partition. */
09475     apos = Vatom_getPosition(atom);
09476     dipole = Vatom_getInducedDipole(atom);
09477     uix = dipole[0];
09478     uiy = dipole[1];
09479     uiz = dipole[2];
09480     dipole = Vatom_getNLInducedDipole(atom);
09481     uixnl = dipole[0];
09482     uiynl = dipole[1];
09483     uiznl = dipole[2];
09484     u = induced->data;
09485     unl = nlinduced->data;
09486
09487     for (i=0;i<3;i++) {
09488         for (j=0;j<3;j++) {
09489             de[i][j] = 0.0;
09490             denl[i][j] = 0.0;
09491         }
09492     }
09493 }
```

```

09486 /* Mesh info */
09487 nx = induced->nx;
09488 ny = induced->ny;
09489 nz = induced->nz;
09490 hx = induced->hx;
09491 hy = induced->hy;
09492 hzed = induced->hzed;
09493 xmin = induced->xmin;
09494 ymin = induced->ymin;
09495 zmin = induced->zmin;
09496 xmax = induced->xmax;
09497 ymax = induced->ymax;
09498 zmax = induced->zmax;
09499 xlabel = xmax-xmin;
09500 ylabel = ymax-ymin;
09501 zlabel = zmax-zmin;
09502
09503 /* If we aren't in the current position, then we're done */
09504 if (atom->partID == 0) return;
09505
09506 /* Make sure we're on the grid */
09507 if ((apos[0]<=(xmin+2*hx)) || (apos[0]>=(xmax-2*hx)) \
09508 || (apos[1]<=(ymin+2*hy)) || (apos[1]>=(ymax-2*hy)) \
09509 || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
09510   Vnm_print(2, "qfMutualPolForce: Atom off the mesh (ignoring) %6.3f
%6.3f %6.3f\n", apos[0], apos[1], apos[2]);
09511   fflush(stderr);
09512 } else {
09513
09514   /* Convert the atom position to grid coordinates */
09515   position[0] = apos[0] - xmin;
09516   position[1] = apos[1] - ymin;
09517   position[2] = apos[2] - zmin;
09518   ifloat = position[0]/hx;
09519   jfloat = position[1]/hy;
09520   kfloat = position[2]/hzed;
09521   ip1 = (int)ceil(ifloat);
09522   ip2 = ip1 + 2;
09523   im1 = (int)floor(ifloat);
09524   im2 = im1 - 2;
09525   jp1 = (int)ceil(jfloat);
09526   jp2 = jp1 + 2;
09527   jm1 = (int)floor(jfloat);
09528   jm2 = jm1 - 2;
09529   kp1 = (int)ceil(kfloat);
09530   kp2 = kp1 + 2;
09531   km1 = (int)floor(kfloat);
09532   km2 = km1 - 2;
09533
09534 /* This step shouldn't be necessary, but it saves nasty debugging
09535 * later on if something goes wrong */
09536 ip2 = VMIN2(ip2,nx-1);
09537 ip1 = VMIN2(ip1,nx-1);
09538 im1 = VMAX2(im1,0);
09539 im2 = VMAX2(im2,0);
09540 jp2 = VMIN2(jp2,ny-1);
09541 jp1 = VMIN2(jp1,ny-1);
09542 jm1 = VMAX2(jm1,0);
09543 jm2 = VMAX2(jm2,0);
09544 kp2 = VMIN2(kp2,nz-1);
09545 kp1 = VMIN2(kp1,nz-1);
09546 km1 = VMAX2(km1,0);
09547 km2 = VMAX2(km2,0);
09548
09549 for (ii=im2; ii<=ip2; ii++) {
09550   mi = VFCHI4(ii,ifloat);
09551   mx = bspline4(mi);
09552   dmx = dbspline4(mi);
09553   d2mx = d2bspline4(mi);
09554   for (jj=jm2; jj<=jp2; jj++) {
09555     mj = VFCHI4(jj,jfloat);
09556     my = bspline4(mj);
09557     dmy = dbspline4(mj);
09558     d2my = d2bspline4(mj);
09559     for (kk=km2; kk<=kp2; kk++) {
09560       mk = VFCHI4(kk,kfloat);
09561       mz = bspline4(mk);
09562       dmz = dbspline4(mk);
09563       d2mz = d2bspline4(mk);
09564       f = u[IJK(ii,jj,kk)];
09565       fnl = unl[IJK(ii,jj,kk)];

```

```

09566
09567      /* The gradient of the reaction field
09568      due to induced dipoles */
09569      de[0][0] += f*d2mx*my*mz/(hx*hx);
09570      de[1][0] += f*dmx*dmy*mz/(hy*hx);
09571      de[1][1] += f*mx*d2my*mz/(hy*hy);
09572      de[2][0] += f*dmx*my*dmz/(hx*hzed);
09573      de[2][1] += f*mx*dmy*dmz/(hy*hzed);
09574      de[2][2] += f*mx*my*d2mz/(hzed*hzed);
09575
09576      /* The gradient of the reaction field
09577      due to non-local induced dipoles */
09578      denl[0][0] += fnl*d2mx*my*mz/(hx*hx);
09579      denl[1][0] += fnl*dmx*dmy*mz/(hy*hx);
09580      denl[1][1] += fnl*mx*d2my*mz/(hy*hy);
09581      denl[2][0] += fnl*dmx*my*dmz/(hx*hzed);
09582      denl[2][1] += fnl*mx*dmy*dmz/(hy*hzed);
09583      denl[2][2] += fnl*mx*my*d2mz/(hzed*hzed);
09584
09585    }
09586  }
09587 }
09588
09589 /* mutual polarization force */
09590 force[0] = -(de[0][0]*uixnl + de[1][0]*uiynl + de[2][0]*uiznl);
09591 force[1] = -(de[1][0]*uixnl + de[1][1]*uiynl + de[2][1]*uiznl);
09592 force[2] = -(de[2][0]*uixnl + de[2][1]*uiynl + de[2][2]*uiznl);
09593 force[0] -= denl[0][0]*uix + denl[1][0]*uiy + denl[2][0]*uiz;
09594 force[1] -= denl[1][0]*uix + denl[1][1]*uiy + denl[2][1]*uiz;
09595 force[2] -= denl[2][0]*uix + denl[2][1]*uiy + denl[2][2]*uiz;
09596
09597 force[0] = 0.5 * force[0];
09598 force[1] = 0.5 * force[1];
09599 force[2] = 0.5 * force[2];
09600
09601 }
09602
09603 VPUBLIC void Vpmg_ibMutualPolForce(Vpmg *thee, Vgrid
09604   *induced, Vgrid *nlinduced,
09605                           int atomID, double force[3]) {
09606
09607   Vatom *atom;
09608   Valist *alist;
09609   Vacc *acc;
09610   Vpbe *pbe;
09611   Vsurf_Meth srfm;
09612
09613   double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
09614   double xlen, ylen, zlen, xmin, ymin, zmin, xmax
09615   , ymax, zmax, rtot2;
09616   double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
09617   double izmagic;
09618   int i, j, k, nx, ny, nz, imin,imax, jmin, jmax, kmin, kmax;
09619   VASSERT(thee != VNULL); /* We need a PMG object with PBE info. */
09620   VASSERT(induced != VNULL); /* We need the potential due to induced
09621   dipoles. */
09622   VASSERT(nlinduced != VNULL); /* We need the potential due to non-local
09623   induced dipoles. */
09624   VASSERT (!thee->pmpg->nonlin); /* Nonlinear PBE is not
09625   implemented for AMOEBA */
09626
09627   atom = Valist_getAtom(thee->pbe->alist, atomID);
09628   VASSERT (atom->partID != 0); /* Currently all atoms must be in the
09629   same partition. */
09630
09631   acc = thee->pbe->acc;
09632   srfm = thee->surfMeth;
09633   apos = Vatom_getPosition(atom);
09634   arad = Vatom_getRadius(atom);
09635
09636   /* Reset force */
09637   force[0] = 0.0;
09638   force[1] = 0.0;
09639   force[2] = 0.0;
09640
09641   /* If we aren't in the current position, then we're done */
09642   if (atom->partID == 0) return;
09643
09644   /* Get PBE info */
09645   pbe = thee->pbe;

```

```

09641     acc = pbe->acc;
09642     alist = pbe->alist;
09643     irad = Vpbe_getMaxIonRadius(pbe);
09644     zkappa2 = Vpbe_getZkappa2(pbe);
09645     izmagic = 1.0/Vpbe_getZmagic(pbe);
09646
09647     VASSERT (zkappa2 > VPMGSMALL); /* Should be a check for this
09648     further up.*/
09649
09650     /* Mesh info */
09651     nx = induced->nx;
09652     ny = induced->ny;
09653     nz = induced->nz;
09654     hx = induced->hx;
09655     hy = induced->hy;
09656     hzed = induced->hzed;
09657     xmin = induced->xmin;
09658     ymin = induced->ymin;
09659     zmin = induced->zmin;
09660     xmax = induced->xmax;
09661     ymax = induced->ymax;
09662     zmax = induced->zmax;
09663     xlabel = xmax-xmin;
09664     ylabel = ymax-ymin;
09665     zlabel = zmax-zmin;
09666
09667     /* Make sure we're on the grid */
09668     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
09669         (apos[1]<=ymin) || (apos[1]>=ymax) || \
09670         (apos[2]<=zmin) || (apos[2]>=zmax)) {
09671         Vnm_print(2, "Vpmg_ibMutualPolForce: Atom at (%4.3f, %4.3f, %4.3f) is
09672 off the mesh (ignoring):\n", apos[0], apos[1], apos[2]);
09673         Vnm_print(2, "Vpmg_ibMutualPolForce: xmin = %g, xmax = %g\n", xmin,
09674         xmax);
09675         Vnm_print(2, "Vpmg_ibMutualPolForce: ymin = %g, ymax = %g\n", ymin,
09676         ymax);
09677         Vnm_print(2, "Vpmg_ibMutualPolForce: zmin = %g, zmax = %g\n", zmin,
09678         zmax);
09679         fflush(stderr);
09680     } else {
09681
09682         /* Convert the atom position to grid reference frame */
09683         position[0] = apos[0] - xmin;
09684         position[1] = apos[1] - ymin;
09685         position[2] = apos[2] - zmin;
09686
09687         /* Integrate over points within this atom's (inflated) radius */
09688         rtot = (irad + arad + thee->splineWin);
09689         rtot2 = VSQR(rtot);
09690         dx = rtot + 0.5*hx;
09691         imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
09692         imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
09693         for (i=imin; i<imax; i++) {
09694             dx2 = VSQR(position[0] - hx*i);
09695             if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
09696             else dy = 0.5*hy;
09697             jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
09698             jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
09699             for (j=jmin; j<jmax; j++) {
09700                 dy2 = VSQR(position[1] - hy*j);
09701                 if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
09702                 else dz = 0.5*hzed;
09703                 kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
09704                 kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
09705                 for (k=kmin; k<=kmax; k++) {
09706                     dz2 = VSQR(k*hzed - position[2]);
09707                     /* See if grid point is inside ivdw radius and set ccf
09708                      * accordingly (do spline assignment here) */
09709                     if ((dz2 + dy2 + dx2) <= rtot2) {
09710                         gpos[0] = i*hx + xmin;
09711                         gpos[1] = j*hy + ymin;
09712                         gpos[2] = k*hzed + zmin;
09713                         Vpmg_splineSelect(srfm, acc, gpos,
09714                             thee->splineWin, irad,
09715                             atom, tgrad);
09716                         fmag = induced->data[IJK(i,j,k)];
09717                         fmag *= nlinduced->data[IJK(i,j,k)];
09718                         fmag *= thee->kappa[IJK(i,j,k)];
09719                         force[0] += (zkappa2*fmag*tgrad[0]);
09720                         force[1] += (zkappa2*fmag*tgrad[1]);
09721                         force[2] += (zkappa2*fmag*tgrad[2]);

```

```

09716             }
09717         } /* k loop */
09718     } /* j loop */
09719 } /* i loop */
09720 }
09721
09722     force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
09723     force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
09724     force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
09725 }
09726
09727 VPUBLIC void Vpmg_dbMutualPolForce(Vpmg *thee, Vgrid
09728     *induced,
09729             Vgrid *nlinduced, int atomID,
09730             double force[3]) {
09731
09732     Vatom *atom;
09733     Vacc *acc;
09734     Vpbe *pbe;
09735     Vsurf_Meth srfm;
09736
09737     double *apos, position[3], arad, hx, hy, hzed, izmagic, deps, depsi
09738     ;
09739     double xlen, ylen, zlen, xmin, ymin, zmin, xmax
09740     , ymax, zmax, rtot2, epsp;
09741     double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
09742     double *u, *unl, Hxijk, Hyijk, Hxim1jk, Hyjm1k, Hzijkml;
09743     double dHxijk[3], dHyijk[3], dHzijk[3], dHxim1jk[3], dHyjm1k[3];
09744     double dHzijkml[3];
09745     int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
09746
09747     VASSERT(thee != VNULL); /* PMG object with PBE info. */
09748     VASSERT(induced != VNULL); /* potential due to induced dipoles.*/
09749     VASSERT(nlinduced != VNULL); /* potential due to non-local induced dipoles.
09750 */
09751     acc = thee->pbe->acc;
09752     srfm = thee->surfMeth;
09753     atom = Valist_getAtom(thee->pbe->alist, atomID);
09754     VASSERT(atom->partID != 0); /* all atoms must be in the same
09755     partition.*/
09756     apos = Vatom_getPosition(atom);
09757     arad = Vatom_getRadius(atom);
09758
09759     /* Reset force */
09760     force[0] = 0.0;
09761     force[1] = 0.0;
09762     force[2] = 0.0;
09763
09764     /* Get PBE info */
09765     pbe = thee->pbe;
09766     acc = pbe->acc;
09767     epsp = Vpbe_getSoluteDiel(pbe);
09768     epsw = Vpbe_getSolventDiel(pbe);
09769     kT = Vpbe_getTemperature(pbe) * (1e-3) * Vunit_Na *
09770     Vunit_kb;
09771     izmagic = 1.0 / Vpbe_getZmagic(pbe);
09772
09773     deps = (epsw - epsp);
09774     depsi = 1.0 / deps;
09775     VASSERT(VABS(deps) > VPMGSMLL);
09776
09777     /* Mesh info */
09778     nx = thee->pmgp->nx;
09779     ny = thee->pmgp->ny;
09780     nz = thee->pmgp->nz;
09781     hx = thee->pmgp->hx;
09782     hy = thee->pmgp->hy;
09783     hzed = thee->pmgp->hzed;
09784     xlen = thee->pmgp->xlen;
09785     ylen = thee->pmgp->ylen;
09786     zlen = thee->pmgp->zlen;
09787     xmin = thee->pmgp->xmin;
09788     ymin = thee->pmgp->ymin;
09789     zmin = thee->pmgp->zmin;
09790     xmax = thee->pmgp->xmax;
09791     ymax = thee->pmgp->ymax;
09792     zmax = thee->pmgp->zmax;
09793     u = induced->data;
09794     unl = nlinduced->data;
09795

```

```

09791     /* Make sure we're on the grid */
09792     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
09793         (apos[1]<=ymin) || (apos[1]>=ymax) || \
09794         (apos[2]<=zmin) || (apos[2]>=zmax)) {
09795         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]);
09796         Vnm_print(2, "Vpmg_dbMutualPolForce: xmin = %g, xmax = %g\n", xmin,
 xmax);
09797         Vnm_print(2, "Vpmg_dbMutualPolForce: ymin = %g, ymax = %g\n", ymin,
 ymax);
09798         Vnm_print(2, "Vpmg_dbMutualPolForce: zmin = %g, zmax = %g\n", zmin,
 zmax);
09799         fflush(stderr);
09800     } else {
09801
09802         /* Convert the atom position to grid reference frame */
09803         position[0] = apos[0] - xmin;
09804         position[1] = apos[1] - ymin;
09805         position[2] = apos[2] - zmin;
09806
09807         /* Integrate over points within this atom's (inflated) radius */
09808         rtot = (arad + thee->splineWin);
09809         rtot2 = VSQR(rtot);
09810         dx = rtot/hx;
09811         imin = (int)floor((position[0]-rtot)/hx);
09812         if (imin < 1) {
09813             Vnm_print(2, "Vpmg_dbMutualPolForce: Atom %d off grid!\n", atomID)
09814         ;
09815             return;
09816         }
09817         imax = (int)ceil((position[0]+rtot)/hx);
09818         if (imax > (nx-2)) {
09819             Vnm_print(2, "Vpmg_dbMutualPolForce: Atom %d off grid!\n", atomID)
09820         ;
09821             return;
09822         }
09823         jmin = (int)floor((position[1]-rtot)/hy);
09824         if (jmin < 1) {
09825             Vnm_print(2, "Vpmg_dbMutualPolForce: Atom %d off grid!\n", atomID)
09826         ;
09827             return;
09828         }
09829         jmax = (int)ceil((position[1]+rtot)/hy);
09830         if (jmax > (ny-2)) {
09831             Vnm_print(2, "Vpmg_dbMutualPolForce: Atom %d off grid!\n", atomID)
09832         ;
09833             return;
09834         }
09835         kmin = (int)floor((position[2]-rtot)/hzed);
09836         if (kmin < 1) {
09837             Vnm_print(2, "Vpmg_dbMutualPolForce: Atom %d off grid!\n", atomID)
09838         ;
09839             return;
09840         }
09841         for (i=imin; i<=imax; i++) {
09842             for (j=jmin; j<=jmax; j++) {
09843                 for (k=kmin; k<=kmax; k++) {
09844                     /* i,j,k */
09845                     gpos[0] = (i+0.5)*hx + xmin;
09846                     gpos[1] = j*hy + ymin;
09847                     gpos[2] = k*hzed + zmin;
09848                     Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
09849                     Vpmg_splineSelect(srfm, acc, gpos, thee->
09850                                     splineWin, 0.,
09851                                     atom, dHxijk);
09852                     for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
09853                     gpos[0] = i*hx + xmin;
09854                     gpos[1] = (j+0.5)*hy + ymin;
09855                     gpos[2] = k*hzed + zmin;
09856                     Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
09857                     Vpmg_splineSelect(srfm, acc, gpos, thee->
09858                                     splineWin, 0.,
09859                                     atom, dHyijk);
09860                     for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
09861                     gpos[0] = i*hx + xmin;

```

```

09860     gpos[1] = j*hy + ymin;
09861     gpos[2] = (k+0.5)*hzed + zmin;
09862     Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
09863     Vpmg_splineSelect(srfm, acc, gpos, thee->
09864         splineWin, 0.,
09865             atom, dHzijk);
09866             for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
09867             /* i-1, j, k */
09868             gpos[0] = (i-0.5)*hx + xmin;
09869             gpos[1] = j*hy + ymin;
09870             gpos[2] = k*hzed + zmin;
09871             Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
09872             Vpmg_splineSelect(srfm, acc, gpos, thee->
09873                 splineWin, 0.,
09874                     atom, dHxim1jk);
09875                     for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
09876                     /* i, j-1, k */
09877                     gpos[0] = i*hx + xmin;
09878                     gpos[1] = (j-0.5)*hy + ymin;
09879                     gpos[2] = k*hzed + zmin;
09880                     Hyijm1k = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
09881                     Vpmg_splineSelect(srfm, acc, gpos, thee->
09882                         splineWin, 0.,
09883                             atom, dHyijm1k);
09884                             for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;
09885                             /* i, j, k-1 */
09886                             gpos[0] = i*hx + xmin;
09887                             gpos[1] = j*hy + ymin;
09888                             gpos[2] = (k-0.5)*hzed + zmin;
09889                             Hzijkml = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
09890                             Vpmg_splineSelect(srfm, acc, gpos, thee->
09891                                 splineWin, 0.,
09892                                     atom, dHzijkml);
09893                                     for (l=0; l<3; l++) dHzijkml[l] *= Hzijkml;
09894                                     dbFmag = unl[IJK(i,j,k)];
09895                                     tgrad[0] =
09896                                         (dHxijk[0] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09897                                         + dHxim1jk[0]* (u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09898                                         + (dHyijk[0] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)]))
09899                                         + dHyijm1k[0]* (u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09900                                         + (dHzijk[0] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)]))
09901                                         + dHzijkml[0]* (u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09902                                     tgrad[1] =
09903                                         (dHxijk[1] * (u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09904                                         + dHxim1jk[1]* (u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09905                                         + (dHyijk[1] * (u[IJK(i,j+1,k)]-u[IJK(i,j,k)]))
09906                                         + dHyijm1k[1]* (u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09907                                         + (dHzijk[1] * (u[IJK(i,j,k+1)]-u[IJK(i,j,k)]))
09908                                         + dHzijkml[1]* (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                                     } /* k loop */
09917                                     } /* j loop */
09918                                     } /* i loop */
09919                                     force[0] = -force[0]*hx*hy*hzed*deps*0.5*izmagic;
09920                                     force[1] = -force[1]*hx*hy*hzed*deps*0.5*izmagic;
09921                                     force[2] = -force[2]*hx*hy*hzed*deps*0.5*izmagic;
09922                                     }
09923     }
09924
09925 #endif /* if defined(WITH_TINKER) */
09926
09927 VPPRIVATE void fillcoCoefSpline4(Vpmg *thee) {
09928
09929     Valist *alist;
09930     Vpbe *pbe;
09931     Vatom *atom;
09932     double xmin, xmax, ymin, ymax, zmin, zmax, ionmask,
09933     ionstr, dist2;
09934     double xlabel, ylabel, zlabel, position[3], itot, stot, ictot, ictot2
09935     , sctot;
09936     double irad, dx, dy, dz, epsw, epsp, w2i;

```

```

09935     double hx, hy, hzed, *apos, arad, sctot2;
09936     double dx2, dy2, dz2, stot2, itot2, rtot, rtot2, splineWin;
09937     double dist, value, denom, sm, sm2, sm3, sm4, sm5, sm6, sm7;
09938     double e, e2, e3, e4, e5, e6, e7;
09939     double b, b2, b3, b4, b5, b6, b7;
09940     double c0, c1, c2, c3, c4, c5, c6, c7;
09941     double ic0, ic1, ic2, ic3, ic4, ic5, ic6, ic7;
09942     int i, j, k, nx, ny, nz, iatom;
09943     int imin, imax, jmin, jmax, kmin, kmax;
09944
09945     VASSERT(thee != VNULL);
09946     splineWin = thee->splineWin;
09947
09948     /* Get PBE info */
09949     pbe = thee->pbe;
09950     alist = pbe->alist;
09951     irad = Vpbe_getMaxIonRadius(pbe);
09952     ionstr = Vpbe_getBulkIonicStrength(pbe);
09953     epsw = Vpbe_getSolventDiel(pbe);
09954     epsp = Vpbe_getSoluteDiel(pbe);
09955
09956     /* Mesh info */
09957     nx = thee->pmgp->nx;
09958     ny = thee->pmgp->ny;
09959     nz = thee->pmgp->nz;
09960     hx = thee->pmgp->hx;
09961     hy = thee->pmgp->hy;
09962     hzed = thee->pmgp->hzed;
09963
09964     /* Define the total domain size */
09965     xlen = thee->pmgp->xlen;
09966     ylen = thee->pmgp->ylen;
09967     zlen = thee->pmgp->zlen;
09968
09969     /* Define the min/max dimensions */
09970     xmin = thee->pmgp->xcent - (xlen/2.0);
09971     ymin = thee->pmgp->ycent - (ylen/2.0);
09972     zmin = thee->pmgp->zcent - (zlen/2.0);
09973     xmax = thee->pmgp->xcent + (xlen/2.0);
09974     ymax = thee->pmgp->ycent + (ylen/2.0);
09975     zmax = thee->pmgp->zcent + (zlen/2.0);
09976
09977     /* This is a floating point parameter related to the non-zero nature of the
09978      * bulk ionic strength. If the ionic strength is greater than zero; this
09979      * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
09980      * Otherwise, this parameter is set to 0.0 */
09981     if (ionstr > VPMGSMALL) ionmask = 1.0;
09982     else ionmask = 0.0;
09983
09984     /* Reset the kappa, epsx, epsy, and epsz arrays */
09985     for (i=0; i<(nx*ny*nz); i++) {
09986         thee->kappa[i] = 1.0;
09987         thee->epsx[i] = 1.0;
09988         thee->epsy[i] = 1.0;
09989         thee->epsz[i] = 1.0;
09990     }
09991
09992     /* Loop through the atoms and do assign the dielectric */
09993     for (iatom=0; iatom<Valist_getNumberAtoms(alist);
09994         iatom++) {
09995         atom = Valist_getAtom(alist, iatom);
09996         apos = Vatom_getPosition(atom);
09997         arad = Vatom_getRadius(atom);
09998
09999         b = arad - splineWin;
10000         e = arad + splineWin;
10001         e2 = e * e;
10002         e3 = e2 * e;
10003         e4 = e3 * e;
10004         e5 = e4 * e;
10005         e6 = e5 * e;
10006         e7 = e6 * e;
10007         b2 = b * b;
10008         b3 = b2 * b;
10009         b4 = b3 * b;
10010         b5 = b4 * b;
10011         b6 = b5 * b;
10012         b7 = b6 * b;
10013         denom = e7 - 7.0*b*e6 + 21.0*b2*e5 - 35.0*e4*b3
10014             + 35.0*e3*b4 - 21.0*b5*e2 + 7.0*e*b6 - b7;

```

```

10015      c0 = b4*(35.0*e3 - 21.0*b*e2 + 7*e*b2 - b3)/denom;
10016      c1 = -140.0*b3*e3/denom;
10017      c2 = 210.0*e2*b2*(e + b)/denom;
10018      c3 = -140.0*e*b*(e2 + 3.0*b*e + b2)/denom;
10019      c4 = 35.0*(e3 + 9.0*b*e2 + + 9.0*e*b2 + b3)/denom;
10020      c5 = -84.0*(e2 + 3.0*b*e + b2)/denom;
10021      c6 = 70.0*(e + b)/denom;
10022      c7 = -20.0/denom;
10023
10024      b = irad + arad - splineWin;
10025      e = irad + arad + splineWin;
10026      e2 = e * e;
10027      e3 = e2 * e;
10028      e4 = e3 * e;
10029      e5 = e4 * e;
10030      e6 = e5 * e;
10031      e7 = e6 * e;
10032      b2 = b * b;
10033      b3 = b2 * b;
10034      b4 = b3 * b;
10035      b5 = b4 * b;
10036      b6 = b5 * b;
10037      b7 = b6 * b;
10038      denom = e7 - 7.0*b*e6 + 21.0*b2*e5 - 35.0*e4*b3
10039      + 35.0*e3*b4 - 21.0*b5*e2 + 7.0*e*b6 - b7;
10040      ic0 = b4*(35.0*e3 - 21.0*b*e2 + 7*e*b2 - b3)/denom;
10041      ic1 = -140.0*b3*e3/denom;
10042      ic2 = 210.0*e2*b2*(e + b)/denom;
10043      ic3 = -140.0*e*b*(e2 + 3.0*b*e + b2)/denom;
10044      ic4 = 35.0*(e3 + 9.0*b*e2 + + 9.0*e*b2 + b3)/denom;
10045      ic5 = -84.0*(e2 + 3.0*b*e + b2)/denom;
10046      ic6 = 70.0*(e + b)/denom;
10047      ic7 = -20.0/denom;
10048
10049      /* Make sure we're on the grid */
10050      if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
10051          (apos[1]<=ymin) || (apos[1]>=ymax) || \
10052          (apos[2]<=zmin) || (apos[2]>=zmax)) {
10053          if ((thee->pmpg->bpcf != BCFL_FOCUS) &&
10054              (thee->pmpg->bpcf != BCFL_MAP)) {
10055              Vnm_print(2, "Vpmg_fillco: Atom #%d at (%4.3f, %4.3f,\n"
10056              "%4.3f) is off the mesh (ignoring):\n",
10057                  iatom, apos[0], apos[1], apos[2]);
10058              Vnm_print(2, "Vpmg_fillco: xmin = %g, xmax = %g\n",
10059                  xmin, xmax);
10060              Vnm_print(2, "Vpmg_fillco: ymin = %g, ymax = %g\n",
10061                  ymin, ymax);
10062              Vnm_print(2, "Vpmg_fillco: zmin = %g, zmax = %g\n",
10063                  zmin, zmax);
10064          }
10065          fflush(stderr);
10066
10067      } else if (arad > VPMGSMALL ) { /* if we're on the mesh */
10068
10069          /* Convert the atom position to grid reference frame */
10070          position[0] = apos[0] - xmin;
10071          position[1] = apos[1] - ymin;
10072          position[2] = apos[2] - zmin;
10073
10074          /* MARK ION ACCESSIBILITY AND DIELECTRIC VALUES FOR LATER
10075             * ASSIGNMENT (Steps #1-3) */
10076          itot = irad + arad + splineWin;
10077          itot2 = VSQR(itot);
10078          ictot = VMAX2(0, (irad + arad - splineWin));
10079          ictot2 = VSQR(ictot);
10080          stot = arad + splineWin;
10081          stot2 = VSQR(stot);
10082          sctot = VMAX2(0, (arad - splineWin));
10083          sctot2 = VSQR(sctot);
10084
10085          /* We'll search over grid points which are in the greater of
10086             * these two radii */
10087          rtot = VMAX2(itot, stot);
10088          rtot2 = VMAX2(itot2, stot2);
10089          dx = rtot + 0.5*hx;
10090          dy = rtot + 0.5*hy;
10091          dz = rtot + 0.5*hzed;
10092          imin = VMAX2(0, (int)floor((position[0] - dx)/hx));
10093          imax = VMIN2(nx-1, (int)ceil((position[0] + dx)/hx));
10094          jmin = VMAX2(0, (int)floor((position[1] - dy)/hy));
10095          jmax = VMIN2(ny-1, (int)ceil((position[1] + dy)/hy));

```

```

10096     kmin = VMAX2(0, (int)floor((position[2] - dz)/hzed));
10097     kmax = VMIN2(nz-1,(int)ceil((position[2] + dz)/hzed));
10098     for (i=iimin; i<=imax; i++) {
10099         dx2 = VSQR(position[0] - hx*i);
10100        for (j=jmin; j<=jmax; j++) {
10101            dy2 = VSQR(position[1] - hy*j);
10102            for (k=kmin; k<=kmax; k++) {
10103                dz2 = VSQR(position[2] - k*hzed);
10104
10105             /* ASSIGN CCF */
10106             if (thee->kappa[IJK(i,j,k)] > VPMGSMALL)
10107             {
10108                 dist2 = dz2 + dy2 + dx2;
10109                 if (dist2 >= itot2) {
10110                     ;
10111                 }
10112                 if (dist2 <= ictot2) {
10113                     thee->kappa[IJK(i,j,k)] = 0.0;
10114                 }
10115                 if ((dist2 < itot2) && (dist2 > ictot2)) {
10116                     dist = VSQRT(dist2);
10117                     sm = dist;
10118                     sm2 = dist2;
10119                     sm3 = sm2 * sm;
10120                     sm4 = sm3 * sm;
10121                     sm5 = sm4 * sm;
10122                     sm6 = sm5 * sm;
10123                     sm7 = sm6 * sm;
10124                     value = ic0 + ic1*sm + ic2*sm2 + ic3*sm3
10125                         + ic4*sm4 + ic5*sm5 + ic6*sm6 + ic7*sm7;
10126                     if (value > 1.0) {
10127                         value = 1.0;
10128                     } else if (value < 0.0){
10129                         value = 0.0;
10130                     }
10131                     thee->kappa[IJK(i,j,k)] *= value;
10132                 }
10133
10134             /* ASSIGN A1CF */
10135             if (thee->epsx[IJK(i,j,k)] > VPMGSMALL) {
10136                 dist2 = dz2+dy2+VSQR(position[0]-(i+0.5)*hx);
10137                 if (dist2 >= stot2) {
10138                     thee->epsx[IJK(i,j,k)] *= 1.0;
10139                 }
10140                 if (dist2 <= sctot2) {
10141                     thee->epsx[IJK(i,j,k)] = 0.0;
10142                 }
10143                 if ((dist2 > sctot2) && (dist2 < stot2)) {
10144                     dist = VSQRT(dist2);
10145                     sm = dist;
10146                     sm2 = VSQR(sm);
10147                     sm3 = sm2 * sm;
10148                     sm4 = sm3 * sm;
10149                     sm5 = sm4 * sm;
10150                     sm6 = sm5 * sm;
10151                     sm7 = sm6 * sm;
10152                     value = c0 + c1*sm + c2*sm2 + c3*sm3
10153                         + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
10154                     if (value > 1.0) {
10155                         value = 1.0;
10156                     } else if (value < 0.0){
10157                         value = 0.0;
10158                     }
10159                     thee->epsx[IJK(i,j,k)] *= value;
10160                 }
10161             }
10162
10163             /* ASSIGN A2CF */
10164             if (thee->epsy[IJK(i,j,k)] > VPMGSMALL) {
10165                 dist2 = dz2+dx2+VSQR(position[1]-(j+0.5)*hy);
10166                 if (dist2 >= stot2) {
10167                     thee->epsy[IJK(i,j,k)] *= 1.0;
10168                 }
10169                 if (dist2 <= sctot2) {
10170                     thee->epsy[IJK(i,j,k)] = 0.0;
10171                 }
10172                 if ((dist2 > sctot2) && (dist2 < stot2)) {
10173                     dist = VSQRT(dist2);
10174                     sm = dist;
10175                     sm2 = VSQR(sm);

```

```

10176                     sm3 = sm2 * sm;
10177                     sm4 = sm3 * sm;
10178                     sm5 = sm4 * sm;
10179                     sm6 = sm5 * sm;
10180                     sm7 = sm6 * sm;
10181                     value = c0 + c1*sm + c2*sm2 + c3*sm3
10182                         + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
10183                     if (value > 1.0) {
10184                         value = 1.0;
10185                     } else if (value < 0.0){
10186                         value = 0.0;
10187                     }
10188                     thee->epsy[IJK(i,j,k)] *= value;
10189                 }
10190             }
10191
10192             /* ASSIGN A3CF */
10193             if (thee->epsz[IJK(i,j,k)] > VPMGSMALL) {
10194                 dist2 = dy2+dx2+VSQR(position[2]-(k+0.5)*hzed);
10195                 if (dist2 >= stot2) {
10196                     thee->epsz[IJK(i,j,k)] *= 1.0;
10197                 }
10198                 if (dist2 <= sctot2) {
10199                     thee->epsz[IJK(i,j,k)] = 0.0;
10200                 }
10201                 if ((dist2 > sctot2) && (dist2 < stot2)) {
10202                     dist = VSQRT(dist2);
10203                     sm = dist;
10204                     sm2 = dist2;
10205                     sm3 = sm2 * sm;
10206                     sm4 = sm3 * sm;
10207                     sm5 = sm4 * sm;
10208                     sm6 = sm5 * sm;
10209                     sm7 = sm6 * sm;
10210                     value = c0 + c1*sm + c2*sm2 + c3*sm3
10211                         + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
10212                     if (value > 1.0) {
10213                         value = 1.0;
10214                     } else if (value < 0.0){
10215                         value = 0.0;
10216                     }
10217                     thee->epsz[IJK(i,j,k)] *= value;
10218                 }
10219             }
10220
10221             } /* k loop */
10222             } /* j loop */
10223         } /* i loop */
10224     } /* endif (on the mesh) */
10225 } /* endfor (over all atoms) */
10226
10227 Vnm_print(0, "Vpmg_fillco: filling coefficient arrays\n");
10228 /* Interpret markings and fill the coefficient arrays */
10229 for (k=0; k<nz; k++) {
10230     for (j=0; j<ny; j++) {
10231         for (i=0; i<nx; i++) {
10232
10233             thee->kappa[IJK(i,j,k)] = ionmask*thee->kappa[IJK(i,j
10234 ,k)];
10235             thee->epsx[IJK(i,j,k)] = (epsw-epsp)*thee->epsx[IJK(i,j
10236 ,k)]
10237                 + epsp;
10238             thee->epsy[IJK(i,j,k)] = (epsw-epsp)*thee->epsy[IJK(i,j
10239 ,k)]
10240                 + epsp;
10241             thee->epsz[IJK(i,j,k)] = (epsw-epsp)*thee->epsz[IJK(i,j
10242 ,k)]
10243                 + epsp;
10244         } /* i loop */
10245     } /* j loop */
10246 } /* k loop */
10247
10248 VPUBLIC void fillcoPermanentInduced(Vpmg *thee) {
10249     Valist *alist;
10250     Vpbe *pbe;
10251     Vatom *atom;

```

```

10253 /* Coversion */
10254 double zmagic, f;
10255 /* Grid */
10256 double xmin, xmax, ymin, ymax, zmin, zmax;
10257 double xlabel, ylabel, zlabel, position[3], ifloat, jfloat, kfloat;
10258 double hx, hy, hzed, *apos;
10259 /* Multipole */
10260 double charge, *dipole, *quad;
10261 double c, ux, uy, uz, qx, qy, qx, qy, qx, qy, qz, qave;
10262 /* B-spline weights */
10263 double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz;
10264 double m1, mj, mk;
10265 /* Loop variables */
10266 int i, ii, jj, kk, nx, ny, nz, iatom;
10267 int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
10268
10269 VASSERT(thee != VNULL);
10270
10271 /* Get PBE info */
10272 pbe = thee->pbe;
10273 alist = pbe->alist;
10274 zmagic = Vpbe_getZmagic(pbe);
10275
10276 /* Mesh info */
10277 nx = thee->pmgp->nx;
10278 ny = thee->pmgp->ny;
10279 nz = thee->pmgp->nz;
10280 hx = thee->pmgp->hx;
10281 hy = thee->pmgp->hy;
10282 hzed = thee->pmgp->hzed;
10283
10284 /* Conversion */
10285 f = zmagic/(hx*hy*hzed);
10286
10287 /* Define the total domain size */
10288 xlabel = thee->pmgp->xlen;
10289 ylabel = thee->pmgp->ylen;
10290 zlabel = thee->pmgp->zlen;
10291
10292 /* Define the min/max dimensions */
10293 xmin = thee->pmgp->xcent - (xlen/2.0);
10294 ymin = thee->pmgp->ycent - (ylen/2.0);
10295 zmin = thee->pmgp->zcent - (zlen/2.0);
10296 xmax = thee->pmgp->xcent + (xlen/2.0);
10297 ymax = thee->pmgp->ycent + (ylen/2.0);
10298 zmax = thee->pmgp->zcent + (zlen/2.0);
10299
10300 /* Fill in the source term (permanent atomic multipoles
10301     and induced dipoles) */
10302 Vnm_print(0, "fillcoPermanentInduced: filling in source term.\n");
10303 for (iatom=0; iatom<Valist_getNumberAtoms(alist);
10304 iatom++) {
10305     atom = Valist_getAtom(alist, iatom);
10306     apos = Vatom_getPosition(atom);
10307
10308     c = Vatom_getCharge(atom)*f;
10309
10310 #if defined(WITH_TINKER)
10311     dipole = Vatom_getDipole(atom);
10312     ux = dipole[0]/hx*f;
10313     uy = dipole[1]/hy*f;
10314     uz = dipole[2]/hzed*f;
10315     dipole = Vatom_getInducedDipole(atom);
10316     ux = ux + dipole[0]/hx*f;
10317     uy = uy + dipole[1]/hy*f;
10318     uz = uz + dipole[2]/hzed*f;
10319     quad = Vatom_getQuadrupole(atom);
10320     qx = (1.0/3.0)*quad[0]/(hx*hx)*f;
10321     qy = (2.0/3.0)*quad[3]/(hx*hy)*f;
10322     qy = (1.0/3.0)*quad[4]/(hy*hy)*f;
10323     qx = (2.0/3.0)*quad[6]/(hzed*hx)*f;
10324     qz = (2.0/3.0)*quad[7]/(hzed*hy)*f;
10325     qz = (1.0/3.0)*quad[8]/(hzed*hzed)*f;
10326 #else
10327     ux = 0.0;
10328     uy = 0.0;
10329     uz = 0.0;
10330     qx = 0.0;
10331     qy = 0.0;
10332     qz = 0.0;

```

```

10333     qzx = 0.0;
10334     qzy = 0.0;
10335     qzz = 0.0;
10336 #endif /* if defined(WITH_TINKER) */
10337
10338     /* Make sure we're on the grid */
10339     if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx)) || \
10340         (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy)) || \
10341         (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
10342         Vnm_print(2, "fillcoPermanentMultipole: Atom #%"d" at (%4.3f, %4.3f,
10343             %4.3f) is off the mesh (ignoring this atom):\n", iatom, apos[0], apos[1], apos[2
10344             ]);
10345         Vnm_print(2, "fillcoPermanentMultipole: xmin = %g, xmax = %g\n",
10346             xmin, xmax);
10347         Vnm_print(2, "fillcoPermanentMultipole: ymin = %g, ymax = %g\n",
10348             ymin, ymax);
10349         Vnm_print(2, "fillcoPermanentMultipole: zmin = %g, zmax = %g\n",
10350             zmin, zmax);
10351         fflush(stderr);
10352     } else {
10353
10354         /* Convert the atom position to grid reference frame */
10355         position[0] = apos[0] - xmin;
10356         position[1] = apos[1] - ymin;
10357         position[2] = apos[2] - zmin;
10358
10359         /* Figure out which vertices we're next to */
10360         ifloat = position[0]/hx;
10361         jfloat = position[1]/hy;
10362         kfloat = position[2]/hzed;
10363
10364         ip1 = (int)ceil(ifloat);
10365         ip2 = ip1 + 2;
10366         im1 = (int)floor(ifloat);
10367         im2 = im1 - 2;
10368         jp1 = (int)ceil(jfloat);
10369         jp2 = jp1 + 2;
10370         jm1 = (int)floor(jfloat);
10371         jm2 = jm1 - 2;
10372         kp1 = (int)ceil(kfloat);
10373         kp2 = kp1 + 2;
10374         km1 = (int)floor(kfloat);
10375         km2 = km1 - 2;
10376
10377         /* This step shouldn't be necessary, but it saves nasty debugging
10378          * later on if something goes wrong */
10379         ip2 = VMIN2(ip2,nx-1);
10380         ip1 = VMIN2(ip1,nx-1);
10381         im1 = VMAX2(im1,0);
10382         im2 = VMAX2(im2,0);
10383         jp2 = VMIN2(jp2,ny-1);
10384         jp1 = VMIN2(jp1,ny-1);
10385         jm1 = VMAX2(jm1,0);
10386         jm2 = VMAX2(jm2,0);
10387         kp2 = VMIN2(kp2,nz-1);
10388         kp1 = VMIN2(kp1,nz-1);
10389         km1 = VMAX2(km1,0);
10390         km2 = VMAX2(km2,0);
10391
10392         /* Now assign fractions of the charge to the nearby verts */
10393         for (ii=im2; ii<ip2; ii++) {
10394             mi = VFCHI4(ii,ifloat);
10395             mx = bspline4(mi);
10396             dmx = dbspline4(mi);
10397             d2mx = d2bspline4(mi);
10398             for (jj=jm2; jj<jp2; jj++) {
10399                 mj = VFCHI4(jj,jfloat);
10400                 my = bspline4(mj);
10401                 dmy = dbspline4(mj);
10402                 d2my = d2bspline4(mj);
10403                 for (kk=km2; kk<kp2; kk++) {
10404                     mk = VFCHI4(kk,kfloat);
10405                     mz = bspline4(mk);
10406                     dmz = dbspline4(mk);
10407                     d2mz = d2bspline4(mk);
10408                     charge = mx*my*mz*c -
10409                         dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz +
10410                         d2mx*my*mz*qxx +
10411                         dmx*dmy*mz*qyx + mx*d2my*mz*qyy +
10412                         dmx*my*dmz*qzx + mx*dmy*dmz*qzy + mx*my*d2mz*qzz;
10413                     thee->charge[IJK(ii,jj,kk)] += charge;

```

```

10409
10410         }
10411     }
10412   }
10413 } /* endif (on the mesh) */
10414
10415 } /* endfor (each atom) */
10416 }
10417
10418 VPRIVATE void fillcoCoefSpline3(Vpmg *thee) {
10419
10420     Valist *alist;
10421     Vpbe *pbe;
10422     Vatom *atom;
10423     double xmin, xmax, ymin, ymax, zmin, zmax, ionmask,
10424     ionstr, dist2;
10425     double xlabel, ylabel, zlabel, position[3], itot, stot, ictot, ictot2
10426     , sctot;
10427     double irad, dx, dy, dz, epsw, epsp, w2i;
10428     double hx, hy, hzed, *apos, arad, sctot2;
10429     double dx2, dy2, dz2, stot2, itot2, rtot, rtot2, splineWin;
10430     double dist, value, denom, sm, sm2, sm3, sm4, sm5;
10431     double e, e2, e3, e4, e5;
10432     double b, b2, b3, b4, b5;
10433     double c0, c1, c2, c3, c4, c5;
10434     double ic0, ic1, ic2, ic3, ic4, ic5;
10435     int i, j, k, nx, ny, nz, iatom;
10436     int imin, imax, jmax, kmin, kmax;
10437
10438     VASSERT(thee != VNULL);
10439     splineWin = thee->splineWin;
10440
10441     /* Get PBE info */
10442     pbe = thee->pbe;
10443     alist = pbe->alist;
10444     irad = Vpbe_getMaxIonRadius(pbe);
10445     ionstr = Vpbe_getBulkIonicStrength(pbe);
10446     epsw = Vpbe_getSolventDiel(pbe);
10447     epsp = Vpbe_getSoluteDiel(pbe);
10448
10449     /* Mesh info */
10450     nx = thee->pmgp->nx;
10451     ny = thee->pmgp->ny;
10452     nz = thee->pmgp->nz;
10453     hx = thee->pmgp->hx;
10454     hy = thee->pmgp->hy;
10455     hzed = thee->pmgp->hzed;
10456
10457     /* Define the total domain size */
10458     xlabel = thee->pmgp->xlabel;
10459     ylabel = thee->pmgp->ylabel;
10460     zlabel = thee->pmgp->zlabel;
10461
10462     /* Define the min/max dimensions */
10463     xmin = thee->pmgp->xcent - (xlabel/2.0);
10464     ymin = thee->pmgp->ycent - (ylabel/2.0);
10465     zmin = thee->pmgp->zcent - (zlabel/2.0);
10466     xmax = thee->pmgp->xcent + (xlabel/2.0);
10467     ymax = thee->pmgp->ycent + (ylabel/2.0);
10468     zmax = thee->pmgp->zcent + (zlabel/2.0);
10469
10470     /* This is a floating point parameter related to the non-zero nature of the
10471      * bulk ionic strength. If the ionic strength is greater than zero; this
10472      * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
10473      * Otherwise, this parameter is set to 0.0 */
10474     if (ionstr > VPMGSMALL) ionmask = 1.0;
10475     else ionmask = 0.0;
10476
10477     /* Reset the kappa, epsx, epsy, and epsz arrays */
10478     for (i=0; i<(nx*ny*nz); i++) {
10479         thee->kappa[i] = 1.0;
10480         thee->epsx[i] = 1.0;
10481         thee->epsy[i] = 1.0;
10482         thee->epsz[i] = 1.0;
10483     }
10484
10485     /* Loop through the atoms and do assign the dielectric */
10486     for (iatom=0; iatom<Valist_getNumberAtoms(alist);
10487          iatom++) {
10488         atom = Valist_getAtom(alist, iatom);

```

```

10487     apos = Vatom_getPosition(atom);
10488     arad = Vatom_getRadius(atom);
10489
10490     b = arad - splineWin;
10491     e = arad + splineWin;
10492     e2 = e * e;
10493     e3 = e2 * e;
10494     e4 = e3 * e;
10495     e5 = e4 * e;
10496     b2 = b * b;
10497     b3 = b2 * b;
10498     b4 = b3 * b;
10499     b5 = b4 * b;
10500     denom = pow((e - b), 5.0);
10501     c0 = -10.0*e2*b3 + 5.0*e*b4 - b5;
10502     c1 = 30.0*e2*b2;
10503     c2 = -30.0*(e2*b + e*b2);
10504     c3 = 10.0*(e2 + 4.0*e*b + b2);
10505     c4 = -15.0*(e + b);
10506     c5 = 6;
10507     c0 = c0/denom;
10508     c1 = c1/denom;
10509     c2 = c2/denom;
10510     c3 = c3/denom;
10511     c4 = c4/denom;
10512     c5 = c5/denom;
10513
10514     b = irad + arad - splineWin;
10515     e = irad + arad + splineWin;
10516     e2 = e * e;
10517     e3 = e2 * e;
10518     e4 = e3 * e;
10519     e5 = e4 * e;
10520     b2 = b * b;
10521     b3 = b2 * b;
10522     b4 = b3 * b;
10523     b5 = b4 * b;
10524     denom = pow((e - b), 5.0);
10525     ic0 = -10.0*e2*b3 + 5.0*e*b4 - b5;
10526     ic1 = 30.0*e2*b2;
10527     ic2 = -30.0*(e2*b + e*b2);
10528     ic3 = 10.0*(e2 + 4.0*e*b + b2);
10529     ic4 = -15.0*(e + b);
10530     ic5 = 6;
10531     ic0 = c0/denom;
10532     ic1 = c1/denom;
10533     ic2 = c2/denom;
10534     ic3 = c3/denom;
10535     ic4 = c4/denom;
10536     ic5 = c5/denom;
10537
10538     /* Make sure we're on the grid */
10539     if ((apos[0]<=xmin) || (apos[0]>=xmax) || \
10540         (apos[1]<=ymin) || (apos[1]>=ymax) || \
10541         (apos[2]<=zmin) || (apos[2]>=zmax)) {
10542     if ((thee->pmgp->bclf != BCFL_FOCUS) &&
10543         (thee->pmgp->bclf != BCFL_MAP)) {
10544         Vnm_print(2, "Vpmg_fillco: Atom #d at (%4.3f, %4.3f,\n"
10545         "%4.3f) is off the mesh (ignoring):\n",
10546         iatom, apos[0], apos[1], apos[2]);
10547         Vnm_print(2, "Vpmg_fillco: xmin = %g, xmax = %g\n",
10548             xmin, xmax);
10549         Vnm_print(2, "Vpmg_fillco: ymin = %g, ymax = %g\n",
10550             ymin, ymax);
10551         Vnm_print(2, "Vpmg_fillco: zmin = %g, zmax = %g\n",
10552             zmin, zmax);
10553     }
10554     fflush(stderr);
10555
10556     } else if (arad > VPMGSMALL ) { /* if we're on the mesh */
10557
10558         /* Convert the atom position to grid reference frame */
10559         position[0] = apos[0] - xmin;
10560         position[1] = apos[1] - ymin;
10561         position[2] = apos[2] - zmin;
10562
10563         /* MARK ION ACCESSIBILITY AND DIELECTRIC VALUES FOR LATER
10564          * ASSIGNMENT (Steps #1-3) */
10565         itot = irad + arad + splineWin;
10566         itot2 = VSQR(itot);
10567         ictot = VMAX2(0, (irad + arad - splineWin));

```

```

10568     ictot2 = VSQR(ictot);
10569     stot = arad + splineWin;
10570     stot2 = VSQR(stot);
10571     sctot = VMAX2(0, (arad - splineWin));
10572     sctot2 = VSQR(sctot);
10573
10574     /* We'll search over grid points which are in the greater of
10575      * these two radii */
10576     rtot = VMAX2(itot, stot);
10577     rtot2 = VMAX2(itot2, stot2);
10578     dx = rtot + 0.5*hx;
10579     dy = rtot + 0.5*hy;
10580     dz = rtot + 0.5*hzed;
10581     imin = VMAX2(0, (int)floor((position[0] - dx)/hx));
10582     imax = VMIN2(nx-1, (int)ceil((position[0] + dx)/hx));
10583     jmin = VMAX2(0, (int)floor((position[1] - dy)/hy));
10584     jmax = VMIN2(ny-1, (int)ceil((position[1] + dy)/hy));
10585     kmin = VMAX2(0, (int)floor((position[2] - dz)/hzed));
10586     kmax = VMIN2(nz-1, (int)ceil((position[2] + dz)/hzed));
10587     for (i=imin; i<imax; i++) {
10588         dx2 = VSQR(position[0] - hx*i);
10589         for (j=jmin; j<=jmax; j++) {
10590             dy2 = VSQR(position[1] - hy*j);
10591             for (k=kmin; k<=kmax; k++) {
10592                 dz2 = VSQR(position[2] - k*hzed);
10593
10594                 /* ASSIGN CCF */
10595                 if (thee->kappa[IJK(i,j,k)] > VPMGSMALL)
10596                 {
10597                     dist2 = dz2 + dy2 + dx2;
10598                     if (dist2 >= itot2) {
10599                         ;
10600                     }
10601                     if (dist2 <= ictot2) {
10602                         thee->kappa[IJK(i,j,k)] = 0.0;
10603                     }
10604                     if ((dist2 < itot2) && (dist2 > ictot2)) {
10605                         dist = VSQRT(dist2);
10606                         sm = dist;
10607                         sm2 = dist2;
10608                         sm3 = sm2 * sm;
10609                         sm4 = sm3 * sm;
10610                         sm5 = sm4 * sm;
10611                         value = ic0 + ic1*sm + ic2*sm2 + ic3*sm3
10612                         + ic4*sm4 + ic5*sm5;
10613                         if (value > 1.0) {
10614                             value = 1.0;
10615                         } else if (value < 0.0){
10616                             value = 0.0;
10617                         }
10618                         thee->kappa[IJK(i,j,k)] *= value;
10619                     }
10620
10621                 /* ASSIGN A1CF */
10622                 if (thee->epsx[IJK(i,j,k)] > VPMGSMALL) {
10623                     dist2 = dz2+dy2+VSQR(position[0]-(i+0.5)*hx);
10624                     if (dist2 >= stot2) {
10625                         thee->epsx[IJK(i,j,k)] *= 1.0;
10626                     }
10627                     if (dist2 <= sctot2) {
10628                         thee->epsx[IJK(i,j,k)] = 0.0;
10629                     }
10630                     if ((dist2 > sctot2) && (dist2 < stot2)) {
10631                         dist = VSQRT(dist2);
10632                         sm = dist;
10633                         sm2 = VSQR(sm);
10634                         sm3 = sm2 * sm;
10635                         sm4 = sm3 * sm;
10636                         sm5 = sm4 * sm;
10637                         value = c0 + c1*sm + c2*sm2 + c3*sm3
10638                         + c4*sm4 + c5*sm5;
10639                         if (value > 1.0) {
10640                             value = 1.0;
10641                         } else if (value < 0.0){
10642                             value = 0.0;
10643                         }
10644                         thee->epsx[IJK(i,j,k)] *= value;
10645                     }
10646                 }
10647             }

```

```

10648             /* ASSIGN A2CF */
10649             if (thee->epsy[IJK(i,j,k)] > VPMGSMALL) {
10650                 dist2 = dz2+dx2+VSQR(position[1]-(j+0.5)*hy);
10651                 if (dist2 >= stot2) {
10652                     thee->epsy[IJK(i,j,k)] *= 1.0;
10653                 }
10654                 if (dist2 <= sctot2) {
10655                     thee->epsy[IJK(i,j,k)] = 0.0;
10656                 }
10657                 if ((dist2 > sctot2) && (dist2 < stot2)) {
10658                     dist = VSQRT(dist2);
10659                     sm = dist;
10660                     sm2 = VSQR(sm);
10661                     sm3 = sm2 * sm;
10662                     sm4 = sm3 * sm;
10663                     sm5 = sm4 * sm;
10664                     value = c0 + c1*sm + c2*sm2 + c3*sm3
10665                         + c4*sm4 + c5*sm5;
10666                     if (value > 1.0) {
10667                         value = 1.0;
10668                     } else if (value < 0.0){
10669                         value = 0.0;
10670                     }
10671                     thee->epsy[IJK(i,j,k)] *= value;
10672                 }
10673             }
10674
10675             /* ASSIGN A3CF */
10676             if (thee->epsz[IJK(i,j,k)] > VPMGSMALL) {
10677                 dist2 = dy2+dx2+VSQR(position[2]-(k+0.5)*hzed);
10678                 if (dist2 >= stot2) {
10679                     thee->epsz[IJK(i,j,k)] *= 1.0;
10680                 }
10681                 if (dist2 <= sctot2) {
10682                     thee->epsz[IJK(i,j,k)] = 0.0;
10683                 }
10684                 if ((dist2 > sctot2) && (dist2 < stot2)) {
10685                     dist = VSQRT(dist2);
10686                     sm = dist;
10687                     sm2 = dist2;
10688                     sm3 = sm2 * sm;
10689                     sm4 = sm3 * sm;
10690                     sm5 = sm4 * sm;
10691                     value = c0 + c1*sm + c2*sm2 + c3*sm3
10692                         + c4*sm4 + c5*sm5;
10693                     if (value > 1.0) {
10694                         value = 1.0;
10695                     } else if (value < 0.0){
10696                         value = 0.0;
10697                     }
10698                     thee->epsz[IJK(i,j,k)] *= value;
10699                 }
10700             }
10701
10702             } /* k loop */
10703         } /* j loop */
10704     } /* i loop */
10705 } /* endif (on the mesh) */
10706 } /* endfor (over all atoms) */
10707
10708 Vnm_print(0, "Vpmg_fillco: filling coefficient arrays\n");
10709 /* Interpret markings and fill the coefficient arrays */
10710 for (k=0; k<nz; k++) {
10711     for (j=0; j<ny; j++) {
10712         for (i=0; i<nx; i++) {
10713
10714             thee->kappa[IJK(i,j,k)] = ionmask*thee->kappa[IJK(i,j
10715 ,k)];
10716             thee->epsx[IJK(i,j,k)] = (epsw-epsp)*thee->epsx[IJK(i,j
10717 ,k)]
10718                 + epsp;
10719             thee->epsy[IJK(i,j,k)] = (epsw-epsp)*thee->epsy[IJK(i,j
10720 ,k)]
10721                 + epsp;
10722
10723         } /* i loop */
10724     } /* j loop */

```

```

10725     } /* k loop */
10726
10727 }
10728
10729 VPRIVATE void bcolcomp(int *iparm, double *rparm, int *iwork, double *
10730     rwork,
10731     double *values, int *rowind, int *colptr, int *flag) {
10732     int nrow, ncol, nonzero, i;
10733     int nxc, nyc, nzc, nf, nc, narr, narrc, n_rpc;
10734     int n_iz, n_ipc, iretot, iintot;
10735     int nrwk, niwk, nx, ny, nz, nlev, ierror, maxlev, mxlv;
10736     int mgcoar, mgdisc, mgsolv;
10737     int k_iz;
10738     int k_ipc, k_rpc, k_ac, k_cc, k_fc, k_pc;
10739
10740     ANNOUNCE_FUNCTION;
10741     WARN_UNTESTED;
10742
10743     // Decode some parameters
10744     nrwk = VAT(iparm, 1);
10745     niwk = VAT(iparm, 2);
10746     nx = VAT(iparm, 3);
10747     ny = VAT(iparm, 4);
10748     nz = VAT(iparm, 5);
10749     nlev = VAT(iparm, 6);
10750
10751     // Some checks on input
10752     mxlv = Vmaxlev(nx, ny, nz);
10753
10754     // Basic grid sizes, etc.
10755     mgcoar = VAT(iparm, 18);
10756     mgdisc = VAT(iparm, 19);
10757     mgsolv = VAT(iparm, 21);
10758     Vmgsz(&mgcoar, &mgdisc, &mgsolv,
10759             &nx, &ny, &nz,
10760             &nlev,
10761             &nxc, &nyc, &nzc,
10762             &nf, &nc,
10763             &narr, &narrc,
10764             &n_rpc, &n_iz, &n_ipc,
10765             &iretot, &iintot);
10766
10767     // Split up the integer work array
10768     k_iz = 1;
10769     k_ipc = k_iz + n_iz;
10770
10771     // Split up the real work array
10772     k_rpc = 1;
10773     k_cc = k_rpc + n_rpc;
10774     k_fc = k_cc + narr;
10775     k_pc = k_fc + narr;
10776     k_ac = k_pc + 27*narrc;
10777
10778     bcolcomp2(iparm, rparm,
10779             &nx, &ny, &nz, RAT(iwork, k_iz),
10780             RAT(iwork, k_ipc), RAT(rwork, k_rpc),
10781             RAT(rwork, k_ac), RAT(rwork, k_cc),
10782             values, rowind, colptr, flag);
10783
10784 VPRIVATE void bcolcomp2(int *iparm, double *rparm,
10785     int *nx, int *ny, int *nz,
10786     int *iz, int *ipc, double *rpc,
10787     double *ac, double *cc, double *values,
10788     int *rowind, int *colptr, int *flag) {
10789
10790     int nlev = 1;
10791     int lev = VAT(iparm, 6);
10792
10793     MAT2(iz, 50, nlev);
10794
10795     ANNOUNCE_FUNCTION;
10796     WARN_UNTESTED;
10797
10798     /*
10799      * Build the multigrid data structure in iz
10800      * THIS MAY HAVE BEEN DONE ALREADY, BUT IT'S OK TO DO IT AGAIN,
10801      * RIGHT?
10802      * call buildstr (nx,ny,nz,nlev,iz)
10803      *
10804      * We're interested in the finest level

```

```

10805      */
10806      bcolcomp3(nx, ny, nz,
10807          RAT(ipc, VAT2(iz, 5, lev)), RAT(rpc, VAT2(iz, 6, lev)),
10808          RAT(ac, VAT2(iz, 7, lev)), RAT(cc, VAT2(iz, 1, lev)),
10809          values, rowind, colptr, flag);
10810 }
10811
10812 /*****
10813 * Routine: bcolcomp3
10814 * Purpose: Build a column-compressed matrix in Harwell-Boeing format
10815 * Args:    flag  0 ==> Use Poisson operator only
10816 *           1 ==> Use linearization of full operator around current
10817 *           solution
10818 * Author:  Nathan Baker (mostly ripped off from Harwell-Boeing format
10819 *           documentation)
10820 ****/
10821 VPRIIVATE void bcolcomp3(int *nx, int *ny, int *nz,
10822     int *ipc, double *rpc,
10823     double *ac, double *cc,
10824     double *values, int *rowind, int *colptr, int *flag) {
10825
10826     MAT2(ac, *nx * *ny * *nz, 1);
10827
10828     ANNOUNCE_FUNCTION;
10829     WARN_UNTESTED;
10830
10831     bcolcomp4(nx, ny, nz,
10832         ipc, rpc,
10833         RAT2(ac, 1, 1), cc,
10834         RAT2(ac, 1, 2), RAT2(ac, 1, 3), RAT2(ac, 1, 4),
10835         values, rowind, colptr, flag);
10836 }
10837
10838
10839
10840 /*****
10841 * Routine: bcolcomp4
10842 * Purpose: Build a column-compressed matrix in Harwell-Boeing format
10843 * Args:    flag  0 ==> Use Poisson operator only
10844 *           1 ==> Use linearization of full operator around current
10845 *           solution
10846 * Author:  Nathan Baker (mostly ripped off from Harwell-Boeing format
10847 *           documentation)
10848 ****/
10849 VPRIIVATE void bcolcomp4(int *nx, int *ny, int *nz,
10850     int *ipc, double *rpc,
10851     double *oC, double *cc, double *oE, double *oN, double *uC,
10852     double *values, int *rowind, int *colptr, int *flag) {
10853
10854     int nxm2, nym2, nzm2;
10855     int ii, jj, kk, ll;
10856     int i, j, k, l;
10857     int inonz, iirow, nn, nrow, ncol, nonz, irow, n;
10858
10859     int doit;
10860
10861     MAT3(oE, *nx, *ny, *nz);
10862     MAT3(oN, *nx, *ny, *nz);
10863     MAT3(uC, *nx, *ny, *nz);
10864     MAT3(cc, *nx, *ny, *nz);
10865     MAT3(oC, *nx, *ny, *nz);
10866
10867     ANNOUNCE_FUNCTION;
10868     WARN_UNTESTED;
10869
10870     // Get some column, row, and nonzero information
10871     n = *nx * *ny * *nz;
10872     nxm2 = *nx - 2;
10873     nym2 = *ny - 2;
10874     nzm2 = *nz - 2;
10875     nn = nxm2 * nym2 * nzm2;
10876     ncol = nn;
10877     nrow = nn;
10878     nonz = 7 * nn - 2 * nxm2 * nym2 - 2 * nxm2 - 2;
10879
10880     // Initialize some pointers
10881     inonz = 1;
10882
10883     /*
10884      * Run over the dimensions of the matrix (non-zero only in the interior
10885      * of the mesh

```

```

10886      */
10887  for (k=2; k<=*nz-1; k++) {
10888      // Offset the index to the output grid index
10889      kk = k - 1;
10890
10891      for (j=2; j<=*ny-1; j++) {
10892          // Offset the index to the output grid index
10893          jj = j - 1;
10894
10895          for (i=2; i<=*nx-1; i++) {
10896              // Offset the index to the output grid index
10897              ii = i - 1;
10898
10899              // Get the output (i,j,k) row number in natural ordering
10900              ll = (kk - 1) * nxm2 * nym2 + (jj - 1) * nxm2 + (ii - 1) + 1;
10901              l = (k - 1) * *nx * *ny + (j - 1) * *nx + (i - 1) + 1;
10902
10903              // Store where this column starts
10904              VAT(colptr,ll) = inonz;
10905
10906              // SUB-DIAGONAL 3
10907              iirow = ll - nxm2 * nym2;
10908              irow = l - *nx * *ny;
10909
10910              doit = (iirow >= 1) && (iirow <= nn);
10911              doit = doit && (irow >= 1) && (irow <= n);
10912
10913              if (doit) {
10914                  VAT(values, inonz) = -VAT3(uC, i, j, k-1);
10915                  VAT(rowind, inonz) = iirow;
10916                  inonz++;
10917              }
10918
10919
10920
10921              // SUB-DIAGONAL 2
10922              iirow = ll - nxm2;
10923              irow = l - *nx;
10924
10925              doit = (iirow >= 1) && (iirow <= nn);
10926              doit = doit && (irow >= 1) && (irow <= n);
10927
10928              if (doit) {
10929                  VAT(values, inonz) = -VAT3(oN, i, j-1, k);
10930                  VAT(rowind, inonz) = iirow;
10931                  inonz++;
10932              }
10933
10934
10935
10936              // SUB-DIAGONAL 1
10937              iirow = ll - 1;
10938              irow = l - 1;
10939
10940              doit = (iirow >= 1) && (iirow <= nn);
10941              doit = doit && (irow <= 1) && (irow <= n);
10942              if (doit) {
10943                  VAT(values, inonz) = -VAT3(oE, i-1, j, k);
10944                  VAT(rowind, inonz) = iirow;
10945                  inonz++;
10946              }
10947
10948
10949
10950              // DIAGONAL
10951              iirow = ll;
10952              irow = l;
10953
10954              if (*flag == 0) {
10955                  VAT(values, inonz) = VAT3(oC, i, j, k);
10956              } else if (*flag == 1) {
10957                  VAT(values, inonz) = VAT3(oC, i, j, k)
10958                      + VAT3(cc, i, j, k);
10959              } else {
10960                  VABORT_MSG0("PMGFI");
10961              }
10962
10963              VAT(rowind, inonz) = iirow;
10964              inonz++;
10965
10966              // SUPER-DIAGONAL 1

```

```

10967             iirow = ll + 1;
10968             irow  = 1 + 1;
10969             doit = (iirow >= 1) && (iirow <= nn);
10970             doit = doit && (irow >= 1) && (irow <= n);
10971             if (doit) {
10972                 VAT(values, inonz) = -VAT3(oE, i, j, k);
10973                 VAT(rowind, inonz) = iirow;
10974                 inonz++;
10975             }
10976
10977
10978         // SUPER-DIAGONAL 2
10979         iirow = ll + nxm2;
10980         irow  = 1 + *nx;
10981         doit = (iirow >= 1) && (iirow <= nn);
10982         doit = doit && (irow >= 1) && (irow <= n);
10983         if (doit) {
10984             VAT(values, inonz) = -VAT3(oN, i, j, k);
10985             VAT(rowind, inonz) = iirow;
10986             inonz++;
10987         }
10988
10989
10990
10991         // SUPER-DIAGONAL 3
10992         iirow = ll + nxm2 * nym2;
10993         irow  = 1 + *nx * *ny;
10994         doit = (iirow >= 1) && (iirow <= nn);
10995         doit = doit && (irow >= 1) && (irow <= n);
10996         if (doit) {
10997             VAT(values, inonz) = -VAT3(uC, i, j, k);
10998             VAT(rowind, inonz) = iirow;
10999             inonz++;
11000         }
11001     }
11002 }
11003 }
11004 }
11005 VAT(colptr, ncol + 1) = inonz;
11006
11007 if (inonz != (nonz + 1)) {
11008     VABORT_MSG2("BCOLCOMP4:  ERROR -- INONZ = %d, NONZ = %d",
11009     inonz, nonz);
11010 }
11011 }
11012
11013
11014
11015 VPRIIVATE void pcolcomp(int *nrow, int *ncol, int *nnzero,
11016     double *values, int *rowind, int *colptr,
11017     char *path, char *title, char *mxtype) {
11018
11019     char key[] = "key";
11020     char ptrfmt[] = "(10I8)";
11021     char indfmt[] = "(10I8)";
11022     char valfmt[] = "(SE15.8)";
11023     char rhsfmt[] = "(5E15.8)";
11024
11025     int i, totcrd, ptrcrd, indcrd, valcrd, neltbl, rhscrd;
11026
11027     FILE *outFile;
11028
11029     ANNOUNCE_FUNCTION;
11030     WARN_UNTESTED;
11031
11032     // Open the file for reading
11033     outFile = fopen(path, "w");
11034
11035     // Set some default values
11036     ptrcrd = (int)(*ncol / 10 + 1) - 1;
11037     indcrd = (int)(*nnzero / 10 + 1) - 1;
11038     valcrd = (int)(*nnzero / 10 + 1) - 1;
11039     totcrd = ptrcrd + indcrd + valcrd;
11040     rhscrd = 0;
11041     neltbl = 0;
11042
11043     // Print the header
11044     fprintf(outFile, "%72s%8s\n",
11045             title, key);
11046     fprintf(outFile, "%14d%14d%14d%14d%14d\n",

```

```

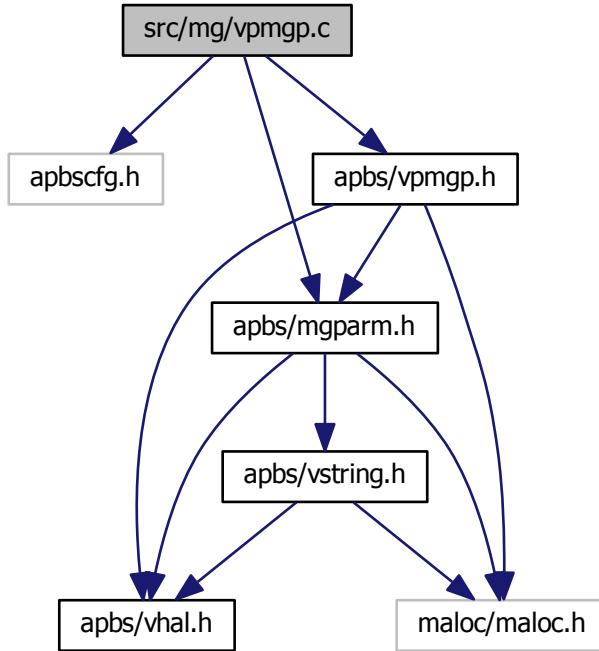
11047     totcrd, ptrcrd, inccrd, valcrd, rhscrd);
11048     fprintf(outFile, "%3s\n", mxtype);
11049     fprintf(outFile, " %14d%14d%14d%14d\n",
11050         *nrow, *ncol, *nnzero, nelv1);
11051     fprintf(outFile, "%16s%16s%20s%20s\n",
11052         ptrfmt, indfmt, valfmt, rhsfmt);
11053
11054 // Write the matrix structure
11055 for (i=1; i<=*ncol+1; i++)
11056     fprintf(outFile, "%8d", VAT(colptr, i));
11057     fprintf(outFile, "\n");
11058
11059 for (i=1; i<=*nnzero; i++)
11060     fprintf(outFile, "%8d", VAT(rowind, i));
11061     fprintf(outFile, "\n");
11062
11063 // Write out the values
11064 if (valcrd > 0) {
11065     for (i=1; i<=*nnzero; i++)
11066         fprintf(outFile, "%15.8e", VAT(values, i));
11067     fprintf(outFile, "\n");
11068 }
11069
11070 // Close the file
11071 fclose (outFile);
11072 }
```

9.99 src/mg/vpmgp.c File Reference

Class Vpmgp methods.

```
#include "apbscfg.h"
#include "apbs/vpmgp.h"
#include "apbs/mgparm.h"
```

Include dependency graph for vpmgp.c:



Functions

- VPUBLIC [Vpmgp * Vpmgp_ctor \(MGparm *mgparm\)](#)
Construct PMG parameter object and initialize to default values.
- VPUBLIC int [Vpmgp_ctor2 \(Vpmgp *thee, MGparm *mgparm\)](#)
FORTRAN stub to construct PMG parameter object and initialize to default values.
- VPUBLIC void [Vpmgp_dtor \(Vpmgp **thee\)](#)
Object destructor.
- VPUBLIC void [Vpmgp_dtor2 \(Vpmgp *thee\)](#)
FORTRAN stub for object destructor.
- VPUBLIC void [Vpmgp_size \(Vpmgp *thee\)](#)
Determine array sizes and parameters for multigrid solver.
- VPRIVATE int **coarsenThis** (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.

9.99.1 Detailed Description

Class Vpmgp methods.

Author

Nathan Baker

Version**Id:**

[vpmgp.c](#) 1750 2012-07-18 18:34:27Z tuckerbeck

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](#).

9.100 vpmgp.c

```

00001
00057 #include "apbscfg.h"
00058 #include "apbs/vpmgp.h"
00059 #include "apbs/mgparm.h"
00060
00061 VEMBED(rcsid="$Id: vpmgp.c 1750 2012-07-18 18:34:27Z tuckerbeck $")
00062
00063 /* //////////////////////////////// */
00064 // Class Vpmgp: Inlineable methods
00065 #if !defined(VINLINE_VACC)
00066 #endif /* if !defined(VINLINE_VACC) */
00067
00068 /* //////////////////////////////// */
00069 // Class Vpmgp: Non-inlineable methods
00070
00071 /* //////////////////////////////// */
00072 // Routine: Vpmgp_ctor
00073
00074 /*
00075 // 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 /* //////////////////////////////// */
00090 // Routine: Vpmgp_ctor2
00091
00092 /*
00093 // Author: Nathan Baker
00094 VPUBLIC int Vpmgp_ctor2(Vpmgp *thee, MGparm *mgparm) {
00095
00096     /* Specified parameters */
00097     thee->nx = mgparm->dime[0];
00098     thee->ny = mgparm->dime[1];
00099     thee->nz = mgparm->dime[2];
00100     thee->hx = mgparm->grid[0];
00101     thee->hy = mgparm->grid[1];
00102     thee->hzed = mgparm->grid[2];
00103     thee->xlen = ((double)(mgparm->dime[0]-1))*mgparm->grid[0];
00104     thee->ylen = ((double)(mgparm->dime[1]-1))*mgparm->grid[1];
00105     thee->zlen = ((double)(mgparm->dime[2]-1))*mgparm->grid[2];
00106     thee->nlev = mgparm->nlev;
00107
00108     thee->nonlin = mgparm->nonlintype;
00109     thee->meth = mgparm->method;
00110
00111 #ifdef DEBUG_MAC OSX_OCL
00112 #include "mach_chud.h"
00113     if(kOpenCLAvailable)
00114         thee->meth = 4;
00115     #endif
00116
00117     if (thee->nonlin == NONLIN_LPBE) thee->ipkey = IPKEY_LPBE; /* LPBE case */
00118     else if(thee->nonlin == NONLIN_SMPBE) thee->ipkey = IPKEY_SMPBE; /* SMPBE case */
00119     else thee->ipkey = IPKEY_NPBE; /* NPBE standard case */
00120
00121     /* Default parameters */
00122     if (mgparm->setetol) { /* If etol is set by the user in APBS input file,
00123     then use this custom-defined etol */
00124         thee->errtol = mgparm->etol;
00125         Vnm_print(1, " Error tolerance (etol) is now set to user-defined \
00126 value: %g \n", thee->errtol);
00127         Vnm_print(0, "Error tolerance (etol) is now set to user-defined \
00128 value: %g \n", thee->errtol);
00129         } else thee->errtol = 1.0e-6; /* Here are a few comments. Mike had this
00130         set to
00131         * 1e-9; conventional wisdom sets this at 1e-6 for
00132         * the PBE; Ray Luo sets this at 1e-3 for his
00133         * accelerated PBE (for dynamics, etc.) */
00134         thee->itmax = 200;

```

```

00134     thee->istop = 1;
00135     thee->iinfo = 1;           /* I'd recommend either 1 (for debugging LPBE) or
00136     2 (for debugging NPBE), higher values give too much output */
00137     thee->bcfl = BCFL_SDH;
00138     thee->key = 0;
00139     thee->iperf = 0;
00140     thee->mgcoar = 2;
00141     thee->mgkey = 0;
00142     thee->nul = 2;
00143     thee->nu2 = 2;
00144     thee->mgprol = 0;
00145     thee->mgdisc = 0;
00146     thee->omegal = 19.4e-1;
00147     thee->omegan = 9.0e-1;
00148     thee->ipcon = 3;
00149     thee->irite = 8;
00150     thee->xcent = 0.0;
00151     thee->ycent = 0.0;
00152     thee->zcent = 0.0;
00153
00154     /* Default value for all APBS runs */
00155     thee->mgsmoo = 1;
00156     if (thee->nonlin == NONLIN_NPBE || thee->nonlin == NONLIN_SMPBE) {
00157     /* SMPBE Added - SMPBE needs to mimic NPBE */
00158     Vnm_print(0, "Vpmp_ctor2: Using meth = 1, mgsolv = 0\n");
00159     thee->mgsolv = 0;
00160 } else {
00161     /* Most rigorous (good for testing) */
00162     Vnm_print(0, "Vpmp_ctor2: Using meth = 2, mgsolv = 1\n");
00163     thee->mgsolv = 1;
00164 }
00165
00166 /* TEMPORARY USEAQUA */
00167 /* If we are using aqua, our solution method is either VSOL_CGMGAqua or
VSOL_NewtonAqua
00168 * so we need to temporarily override the mgsolve value and set it to 0
00169 */
00170 if (mgparm->useAqua == 1) thee->mgsolv = 0;
00171
00172 return 1;
00173 }
00174
00175 /* /////////////////////////////////
00176 // Routine: Vpmgp_dtor
00177 //
00178 // Author: Nathan Baker
00179 VPUBLIC void Vpmgp_dtor(Vpmgp **thee) {
00180
00181     if ((*thee) != VNULL) {
00182         Vpmgp_dtor2(*thee);
00183         Vmem_free(VNULL, 1, sizeof(Vpmgp), (void **)thee);
00184         (*thee) = VNULL;
00185     }
00186 }
00187
00188 }
00189
00190 /* /////////////////////////////////
00191 // Routine: Vpmgp_dtor2
00192 //
00193 // Author: Nathan Baker
00194 VPUBLIC void Vpmgp_dtor2(Vpmgp *thee) { ; }
00195
00196
00197
00198 VPUBLIC void Vpmgp_size(
00199     Vpmgp *thee
00200 )
00201 {
00202
00203     int num_nf = 0;
00204     int num_narr = 2;
00205     int num_narrc = 27;
00206     int nxf, nyf, nzf, level, num_nf_oper, num_narcc_oper, n_band,
nc_band,
num_band, iretot;
00207
00208     thee->nf = thee->nx * thee->ny * thee->nz;
00209     thee->narr = thee->nf;
00210     nxf = thee->nx;
00211     nyf = thee->ny;
00212     nzf = thee->nz;
00213     thee->nxc = thee->nx;

```

```

00214     thee->nyc = thee->ny;
00215     thee->nzc = thee->nz;
00216
00217     for (level=2; level<=thee->nlev; level++) {
00218         Vpmgp_makeCoarse(1, nxf, nyf, nzf, &(thee->nxc), &(thee->nyc), &(thee->nzc));
00219         /* NAB TO-DO -- implement this function and check which variables need to be
00220         passed by reference... */
00221         nxf = thee->nxc;
00222         nyf = thee->nyc;
00223         nzf = thee->nzc;
00224         thee->narr = thee->narr + (nxf * nyf * nzf);
00225     }
00226
00227     thee->nc = thee->nxc * thee->nyc * thee->nzc;
00228     thee->narrc = thee->narr - thee->nf;
00229
00230     /* Box or FEM discretization on fine grid? */
00231     switch (thee->mgdisc) { /* NAB TO-DO: This needs to be changed into an
00232     enumeration */
00233     case 0:
00234         num_nf_oper = 4;
00235         break;
00236     case 1:
00237         num_nf_oper = 14;
00238         break;
00239     default:
00240         Vnm_print(2, "Vpmgp_size: Invalid mgdisc value (%d)!\n", thee->mgdisc);
00241         VASSERT(0);
00242     }
00243
00244     /* Galerkin or standard coarsening? */
00245     switch (thee->mgcoar) { /* NAB TO-DO: This needs to be changed into an
00246     enumeration */
00247     case 0:
00248         if (thee->mgdisc != 0) {
00249             Vnm_print(2, "Vpmgp_size: Invalid mgcoar value (%d); must be used with
00250             mgdisc 0!\n", thee->mgcoar);
00251             VASSERT(0);
00252         }
00253         num_narrc_oper = 4;
00254         break;
00255     case 1:
00256         if (thee->mgdisc != 0) {
00257             Vnm_print(2, "Vpmgp_size: Invalid mgcoar value (%d); must be used with
00258             mgdisc 0!\n", thee->mgcoar);
00259             VASSERT(0);
00260         }
00261         num_narrc_oper = 14;
00262         break;
00263     default:
00264         Vnm_print(2, "Vpmgp_size: Invalid mgcoar value (%d)!\n", thee->mgcoar);
00265         VASSERT(0);
00266     }
00267
00268     /* LINPACK storage on coarse grid */
00269     switch (thee->mgsolv) { /* NAB TO-DO: This needs to be changed into an
00270     enumeration */
00271     case 0:
00272         n_band = 0;
00273         break;
00274     case 1:
00275         if ( ( (thee->mgcoar == 0) || (thee->mgcoar == 1) ) && (thee->mgdisc == 0) ) {
00276             num_band = 1 + (thee->nxc-2)*(thee->nyc-2);
00277         } else {
00278             num_band = 1 + (thee->nxc-2)*(thee->nyc-2) + (thee->nxc-2) + 1;
00279         }
00280         nc_band = (thee->nxc-2)*(thee->nyc-2)*(thee->nzc-2);
00281         n_band = nc_band * num_band;
00282         break;
00283     default:
00284         Vnm_print(2, "Vpmgp_size: Invalid mgsolv value (%d)!\n", thee->mgsolv);
00285         VASSERT(0);
00286     }
00287
00288     /* Real storage parameters */
00289     thee->n_rpc = 100*(thee->nlev+1);
00290
00291     /* Resulting total required for real storage */

```

```
00288     thee->nwk = num_narr*thee->narr + (num_nf + num_nf_oper)*thee->nf + (
00289         num_narcc + num_narcc_oper)*thee->narcc + n_band + thee->n_rpc;
00290     /* Integer storage parameters */
00291     thee->n_iz = 50*(thee->nlev+1);
00292     thee->n_ipc = 100*(thee->nlev+1);
00293     thee->n_iwk = thee->n_iz + thee->n_ipc;
00294 }
00295
00296 VPRIIVATE int coarsenThis(int nOld) {
00297
00298     int nOut;
00299
00300     nOut = (nOld - 1) / 2 + 1;
00301
00302     if (((nOut-1)*2) != (nOld-1)) {
00303         Vnm_print(2, "Vpmgp_makeCoarse: Warning! The grid dimensions you have
00304 chosen are not consistent with the nlev you have specified!\n");
00305         Vnm_print(2, "Vpmgp_makeCoarse: This calculation will only work if you are
00306 running with mg-dummy type.\n");
00307     }
00308     if (nOut < 1) {
00309         Vnm_print(2, "D'oh! You coarsened the grid below zero! How did you do that?
00310 \n");
00311         VASSERT(0);
00312     }
00313
00314 VPUBLIC void Vpmgp_makeCoarse(
00315     int numLevel,
00316     int nxOld,
00317     int nyOld,
00318     int nzOld,
00319     int *nxNew,
00320     int *nyNew,
00321     int *nzNew
00322 )
00323 {
00324     int nxtmp, nytmp, nzttmp, iLevel;
00325
00326     for (iLevel=0; iLevel<numLevel; iLevel++) {
00327         nxtmp = *nxNew;
00328         nytmp = *nyNew;
00329         nzttmp = *nzNew;
00330         *nxNew = coarsenThis(nxtmp);
00331         *nyNew = coarsenThis(nytmp);
00332         *nzNew = coarsenThis(nzttmp);
00333     }
00334
00335
00336 }
```