APBS

1.4.0

Generated by Doxygen 1.8.1.2

Wed Aug 1 2012 07:08:38

Contents

1 APBS Programmers Guide **1**

1.1 Table of Contents . . . 1

1.2 License . . . 1

1.3 Programming Style . . . 2

1.4 Application programming interface documentation . . . 3

2 Deprecated List **5**

3 Bug List **7**

4 Module Index **11**

4.1 Modules . . . 11

5 Data Structure Index **13**

5.1 Data Structures . . . 13

6 File Index **15**

6.1 File List . . . 15

7 Module Documentation **19**

7.1 Vcsm class . . . 19

7.1.1 Detailed Description . . . 20

7.1.2 Function Documentation . . . 20

7.1.2.1 Gem_setExternalUpdateFunction . . . 20

7.1.2.2 Vcsm_ctor . . . 21

7.1.2.3 Vcsm_ctor2 . . . 22

7.1.2.4 Vcsm_dtor . . . 23

7.1.2.5 Vcsm_dtor2 . . . 23

7.1.2.6 Vcsm_getAtom . . . 24

7.1.2.7 Vcsm_getAtomIndex . . . 25

7.1.2.8 Vcsm_getNumberAtoms . . . 25

7.1.2.9 Vcsm_getNumberSimplices . . . 26
7.1.2.10 Vcsm_getSimplex . . . 26
7.1.2.11 Vcsm_getSimplexIndex . . . 27
7.1.2.12 Vcsm_getValist . . . 27
7.1.2.13 Vcsm_init . . . 28
7.1.2.14 Vcsm_memChk . . . 28
7.1.2.15 Vcsm_update . . . 29
7.2 Vfetk class . . . 31
7.2.1 Detailed Description . . . 34
7.2.2 Enumeration Type Documentation . . . 35
7.2.2.1 eVfetk_GuessType . . . 35
7.2.2.2 eVfetk_LsolvType . . . 35
7.2.2.3 eVfetk_MeshLoad . . . 35
7.2.2.4 eVfetk_NsolvType . . . 35
7.2.2.5 eVfetk_PrecType . . . 36
7.2.3 Function Documentation . . . 36
7.2.3.1 Bmat_printHB . . . 36
7.2.3.2 Vfetk_ctor . . . 37
7.2.3.3 Vfetk_ctor2 . . . 38
7.2.3.4 Vfetk_dqmEnergy . . . 39
7.2.3.5 Vfetk_dtor . . . 40
7.2.3.6 Vfetk_dtor2 . . . 40
7.2.3.7 Vfetk_dumpLocalVar . . . 41
7.2.3.8 Vfetk_energy . . . 41
7.2.3.9 Vfetk_externalUpdateFunction . . . 42
7.2.3.10 Vfetk_fillArray . . . 43
7.2.3.11 Vfetk_genCube . . . 44
7.2.3.12 Vfetk_getAM . . . 45
7.2.3.13 Vfetk_getAtomColor . . . 45
7.2.3.14 Vfetk_getGem . . . 46
7.2.3.15 Vfetk_getSolution . . . 46
7.2.3.16 Vfetk_getVcsm . . . 47
7.2.3.17 Vfetk_getVpbe . . . 48
7.2.3.18 Vfetk_loadGem . . . 48
7.2.3.19 Vfetk_loadMesh . . . 48
7.2.3.20 Vfetk_memChk . . . 49
7.2.3.21 Vfetk_PDE_bisectEdge . . . 50

7.2.3.22 Vfetk_PDE_ctor . . . 50
7.2.3.23 Vfetk_PDE_ctor2 . . . 52
7.2.3.24 Vfetk_PDE_delta . . . 53
7.2.3.25 Vfetk_PDE_DFu_wv . . . 55
7.2.3.26 Vfetk_PDE_dtor . . . 55
7.2.3.27 Vfetk_PDE_dtor2 . . . 56
7.2.3.28 Vfetk_PDE_Fu . . . 57
7.2.3.29 Vfetk_PDE_Fu_v . . . 58
7.2.3.30 Vfetk_PDE_initAssemble . . . 58
7.2.3.31 Vfetk_PDE_initElement . . . 59
7.2.3.32 Vfetk_PDE_initFace . . . 60
7.2.3.33 Vfetk_PDE_initPoint . . . 60
7.2.3.34 Vfetk_PDE_Ju . . . 61
7.2.3.35 Vfetk_PDE_mapBoundary . . . 62
7.2.3.36 Vfetk_PDE_markSimplex . . . 62
7.2.3.37 Vfetk_PDE_oneChart . . . 63
7.2.3.38 Vfetk_PDE_simplexBasisForm . . . 64
7.2.3.39 Vfetk_PDE_simplexBasisInit . . . 65
7.2.3.40 Vfetk_PDE_u_D . . . 66
7.2.3.41 Vfetk_PDE_u_T . . . 66
7.2.3.42 Vfetk_qfEnergy . . . 67
7.2.3.43 Vfetk_readMesh . . . 68
7.2.3.44 Vfetk_setAtomColors . . . 69
7.2.3.45 Vfetk_setParameters . . . 70
7.2.3.46 Vfetk_write . . . 70
7.3 Vpee class . . . 72
7.3.1 Detailed Description . . . 72
7.3.2 Function Documentation . . . 73
7.3.2.1 Vpee_ctor . . . 73
7.3.2.2 Vpee_ctor2 . . . 73
7.3.2.3 Vpee_dtor . . . 74
7.3.2.4 Vpee_dtor2 . . . 74
7.3.2.5 Vpee_markRefine . . . 74
7.3.2.6 Vpee_numSS . . . 75
7.4 APOLparm class . . . 77
7.4.1 Detailed Description . . . 78
7.4.2 Enumeration Type Documentation . . . 78

7.4.2.1 eAPOLparm_calcEnergy . . . 78
7.4.2.2 eAPOLparm_calcForce . . . 78
7.4.2.3 eAPOLparm_doCalc . . . 79
7.4.3 Function Documentation . . . 79
7.4.3.1 APOLparm_check . . . 79
7.4.3.2 APOLparm_copy . . . 79
7.4.3.3 APOLparm_ctor . . . 80
7.4.3.4 APOLparm_ctor2 . . . 81
7.4.3.5 APOLparm_dtor . . . 81
7.4.3.6 APOLparm_dtor2 . . . 82
7.5 FEMparm class . . . 83
7.5.1 Detailed Description . . . 84
7.5.2 Typedef Documentation . . . 84
7.5.2.1 FEMparm_EtolType . . . 84
7.5.3 Enumeration Type Documentation . . . 84
7.5.3.1 eFEMparm_CalcType . . . 84
7.5.3.2 eFEMparm_EstType . . . 84
7.5.3.3 eFEMparm_EtolType . . . 85
7.5.4 Function Documentation . . . 85
7.5.4.1 FEMparm_check . . . 85
7.5.4.2 FEMparm_copy . . . 86
7.5.4.3 FEMparm_ctor . . . 86
7.5.4.4 FEMparm_ctor2 . . . 87
7.5.4.5 FEMparm_dtor . . . 88
7.5.4.6 FEMparm_dtor2 . . . 89
7.6 MGparm class . . . 90
7.6.1 Detailed Description . . . 91
7.6.2 Enumeration Type Documentation . . . 91
7.6.2.1 eMGparm_CalcType . . . 91
7.6.2.2 eMGparm_CentMeth . . . 92
7.6.3 Function Documentation . . . 92
7.6.3.1 APOLparm_parseToken . . . 92
7.6.3.2 FEMparm_parseToken . . . 93
7.6.3.3 MGparm_check . . . 93
7.6.3.4 MGparm_copy . . . 94
7.6.3.5 MGparm_ctor . . . 94
7.6.3.6 MGparm_ctor2 . . . 95

7.6.3.7 MGparm_dtor . . . 96
7.6.3.8 MGparm_dtor2 . . . 97
7.6.3.9 MGparm_getCenterX . . . 97
7.6.3.10 MGparm_getCenterY . . . 97
7.6.3.11 MGparm_getCenterZ . . . 98
7.6.3.12 MGparm_getHx . . . 98
7.6.3.13 MGparm_getHy . . . 98
7.6.3.14 MGparm_getHz . . . 99
7.6.3.15 MGparm_getNx . . . 99
7.6.3.16 MGparm_getNy . . . 99
7.6.3.17 MGparm_getNz . . . 100
7.6.3.18 MGparm_parseToken . . . 100
7.6.3.19 MGparm_setCenterX . . . 101
7.6.3.20 MGparm_setCenterY . . . 101
7.6.3.21 MGparm_setCenterZ . . . 102
7.7 NOsh class . . . 103
7.7.1 Detailed Description . . . 105
7.7.2 Enumeration Type Documentation . . . 105
7.7.2.1 eNOsh_CalcType . . . 105
7.7.2.2 eNOsh_MolFormat . . . 106
7.7.2.3 eNOsh_ParmFormat . . . 106
7.7.2.4 eNOsh_PrintType . . . 106
7.7.3 Function Documentation . . . 106
7.7.3.1 NOsh_apol2calc . . . 106
7.7.3.2 NOsh_calc_copy . . . 107
7.7.3.3 NOsh_calc_ctor . . . 107
7.7.3.4 NOsh_calc_dtor . . . 108
7.7.3.5 NOsh_ctor . . . 109
7.7.3.6 NOsh_ctor2 . . . 110
7.7.3.7 NOsh_dtor . . . 111
7.7.3.8 NOsh_dtor2 . . . 111
7.7.3.9 NOsh_elec2calc . . . 112
7.7.3.10 NOsh_elecname . . . 112
7.7.3.11 NOsh_getCalc . . . 113
7.7.3.12 NOsh_getChargefmt . . . 113
7.7.3.13 NOsh_getChargepath . . . 114
7.7.3.14 NOsh_getDielfmt . . . 114

7.7.3.15 NOsh_getDielXpath . . . 114
7.7.3.16 NOsh_getDielYpath . . . 115
7.7.3.17 NOsh_getDielZpath . . . 115
7.7.3.18 NOsh_getKappafmt . . . 116
7.7.3.19 NOsh_getKappapath . . . 116
7.7.3.20 NOsh_getMolpath . . . 116
7.7.3.21 NOsh_getPotfmt . . . 117
7.7.3.22 NOsh_getPotpath . . . 117
7.7.3.23 NOsh_parseInput . . . 117
7.7.3.24 NOsh_parseInputFile . . . 118
7.7.3.25 NOsh_printCalc . . . 119
7.7.3.26 NOsh_printNarg . . . 119
7.7.3.27 NOsh_printOp . . . 120
7.7.3.28 NOsh_printWhat . . . 121
7.7.3.29 NOsh_setupApolCalc . . . 121
7.7.3.30 NOsh_setupElecCalc . . . 122
7.8 PBEparm class . . . 123
7.8.1 Detailed Description . . . 124
7.8.2 Enumeration Type Documentation . . . 124
7.8.2.1 ePBEparm_calcEnergy . . . 124
7.8.2.2 ePBEparm_calcForce . . . 124
7.8.3 Function Documentation . . . 125
7.8.3.1 PBEparm_check . . . 125
7.8.3.2 PBEparm_copy . . . 125
7.8.3.3 PBEparm_ctor . . . 125
7.8.3.4 PBEparm_ctor2 . . . 126
7.8.3.5 PBEparm_dtor . . . 127
7.8.3.6 PBEparm_dtor2 . . . 127
7.8.3.7 PBEparm_getIonCharge . . . 128
7.8.3.8 PBEparm_getIonConc . . . 128
7.8.3.9 PBEparm_getIonRadius . . . 129
7.8.3.10 PBEparm_parseToken . . . 129
7.9 Vacc class . . . 131
7.9.1 Detailed Description . . . 133
7.9.2 Function Documentation . . . 133
7.9.2.1 Vacc_atomdSASA . . . 133
7.9.2.2 Vacc_atomdSAV . . . 133

7.9.2.3 Vacc_atomSASA . . . 134
7.9.2.4 Vacc_atomSASPoints . . . 135
7.9.2.5 Vacc_atomSurf . . . 135
7.9.2.6 Vacc_ctor . . . 137
7.9.2.7 Vacc_ctor2 . . . 138
7.9.2.8 Vacc_dtor . . . 139
7.9.2.9 Vacc_dtor2 . . . 140
7.9.2.10 Vacc_fastMolAcc . . . 140
7.9.2.11 Vacc_ivdwAcc . . . 141
7.9.2.12 Vacc_memChk . . . 142
7.9.2.13 Vacc_molAcc . . . 143
7.9.2.14 Vacc_SASA . . . 144
7.9.2.15 Vacc_splineAcc . . . 145
7.9.2.16 Vacc_splineAccAtom . . . 146
7.9.2.17 Vacc_splineAccGrad . . . 147
7.9.2.18 Vacc_splineAccGradAtomNorm . . . 148
7.9.2.19 Vacc_splineAccGradAtomNorm3 . . . 149
7.9.2.20 Vacc_splineAccGradAtomNorm4 . . . 150
7.9.2.21 Vacc_splineAccGradAtomUnnorm . . . 151
7.9.2.22 Vacc_totalAtomdSASA . . . 151
7.9.2.23 Vacc_totalAtomdSAV . . . 152
7.9.2.24 Vacc_totalSASA . . . 153
7.9.2.25 Vacc_totalSAV . . . 154
7.9.2.26 Vacc_vdwAcc . . . 155
7.9.2.27 Vacc_wcaEnergy . . . 156
7.9.2.28 Vacc_wcaEnergyAtom . . . 157
7.9.2.29 Vacc_wcaForceAtom . . . 158
7.9.2.30 VaccSurf_ctor . . . 158
7.9.2.31 VaccSurf_ctor2 . . . 159
7.9.2.32 VaccSurf_dtor . . . 160
7.9.2.33 VaccSurf_dtor2 . . . 160
7.9.2.34 VaccSurf_refSphere . . . 161
7.10 Valist class . . . 163
7.10.1 Detailed Description . . . 164
7.10.2 Function Documentation . . . 164
7.10.2.1 Valist_ctor . . . 164
7.10.2.2 Valist_ctor2 . . . 164

7.10.2.3 Valist_dtor . . . 165
7.10.2.4 Valist_dtor2 . . . 165
7.10.2.5 Valist_getAtom . . . 166
7.10.2.6 Valist_getAtomList . . . 167
7.10.2.7 Valist_getCenterX . . . 168
7.10.2.8 Valist_getCenterY . . . 168
7.10.2.9 Valist_getCenterZ . . . 168
7.10.2.10 Valist_getNumberAtoms . . . 169
7.10.2.11 Valist_getStatistics . . . 170
7.10.2.12 Valist_memChk . . . 171
7.10.2.13 Valist_readPDB . . . 171
7.10.2.14 Valist_readPQR . . . 172
7.10.2.15 Valist_readXML . . . 173
7.11 Vatom class . . . 175
7.11.1 Detailed Description . . . 176
7.11.2 Macro Definition Documentation . . . 176
7.11.2.1 VMAX_RECLEN . . . 176
7.11.3 Function Documentation . . . 177
7.11.3.1 Vatom_copyFrom . . . 177
7.11.3.2 Vatom_copyTo . . . 177
7.11.3.3 Vatom_ctor . . . 178
7.11.3.4 Vatom_ctor2 . . . 178
7.11.3.5 Vatom_dtor . . . 179
7.11.3.6 Vatom_dtor2 . . . 179
7.11.3.7 Vatom_getAtomID . . . 180
7.11.3.8 Vatom_getAtomName . . . 180
7.11.3.9 Vatom_getCharge . . . 181
7.11.3.10 Vatom_getEpsilon . . . 182
7.11.3.11 Vatom_getPartID . . . 182
7.11.3.12 Vatom_getPosition . . . 182
7.11.3.13 Vatom_getRadius . . . 183
7.11.3.14 Vatom_getResName . . . 184
7.11.3.15 Vatom_memChk . . . 185
7.11.3.16 Vatom_setAtomID . . . 185
7.11.3.17 Vatom_setAtomName . . . 186
7.11.3.18 Vatom_setCharge . . . 186
7.11.3.19 Vatom_setEpsilon . . . 187

7.11.3.20 Vatom_setPartID . . . 187
7.11.3.21 Vatom_setPosition . . . 188
7.11.3.22 Vatom_setRadius . . . 189
7.11.3.23 Vatom_setResName . . . 190
7.12 Vcap class . . . 191
7.12.1 Detailed Description . . . 191
7.12.2 Function Documentation . . . 191
7.12.2.1 Vcap_cosh . . . 191
7.12.2.2 Vcap_exp . . . 192
7.12.2.3 Vcap_sinh . . . 193
7.13 Vclist class . . . 194
7.13.1 Detailed Description . . . 195
7.13.2 Enumeration Type Documentation . . . 195
7.13.2.1 eVclist_DomainMode . . . 195
7.13.3 Function Documentation . . . 195
7.13.3.1 Vclist_ctor . . . 195
7.13.3.2 Vclist_ctor2 . . . 196
7.13.3.3 Vclist_dtor . . . 197
7.13.3.4 Vclist_dtor2 . . . 198
7.13.3.5 Vclist_getCell . . . 199
7.13.3.6 Vclist_maxRadius . . . 199
7.13.3.7 Vclist_memChk . . . 200
7.13.3.8 VclistCell_ctor . . . 200
7.13.3.9 VclistCell_ctor2 . . . 201
7.13.3.10 VclistCell_dtor . . . 201
7.13.3.11 VclistCell_dtor2 . . . 202
7.14 Vgreen class . . . 203
7.14.1 Detailed Description . . . 204
7.14.2 Function Documentation . . . 205
7.14.2.1 Vgreen_coulomb . . . 205
7.14.2.2 Vgreen_coulomb_direct . . . 206
7.14.2.3 Vgreen_coulombD . . . 207
7.14.2.4 Vgreen_coulombD_direct . . . 208
7.14.2.5 Vgreen_ctor . . . 209
7.14.2.6 Vgreen_ctor2 . . . 210
7.14.2.7 Vgreen_dtor . . . 211
7.14.2.8 Vgreen_dtor2 . . . 211

7.14.2.9 Vgreen_getValist . . . 212
7.14.2.10 Vgreen_helmholtz . . . 212
7.14.2.11 Vgreen_helmholtzD . . . 213
7.14.2.12 Vgreen_memChk . . . 214
7.15 Vhal class . . . 215
7.15.1 Detailed Description . . . 218
7.15.2 Macro Definition Documentation . . . 218
7.15.2.1 MAX_SPHERE_PTS . . . 218
7.15.2.2 VABORT_MSG0 . . . 218
7.15.2.3 VABORT_MSG1 . . . 218
7.15.2.4 VABORT_MSG2 . . . 219
7.15.2.5 VAPBS_BACK . . . 219
7.15.2.6 VAPBS_DOWN . . . 219
7.15.2.7 VAPBS_FRONT . . . 220
7.15.2.8 VAPBS_LEFT . . . 220
7.15.2.9 VAPBS_RIGHT . . . 220
7.15.2.10 VAPBS_UP . . . 220
7.15.2.11 VASSERT_MSG0 . . . 220
7.15.2.12 VASSERT_MSG1 . . . 221
7.15.2.13 VASSERT_MSG2 . . . 221
7.15.2.14 VEMBED . . . 222
7.15.2.15 VFLOOR . . . 222
7.15.2.16 VWARN_MSG0 . . . 222
7.15.2.17 VWARN_MSG1 . . . 222
7.15.2.18 VWARN_MSG2 . . . 223
7.15.3 Enumeration Type Documentation . . . 223
7.15.3.1 eVbcfl . . . 223
7.15.3.2 eVchrg_Meth . . . 224
7.15.3.3 eVchrg_Src . . . 224
7.15.3.4 eVdata_Format . . . 225
7.15.3.5 eVdata_Type . . . 225
7.15.3.6 eVhal_IPKEYType . . . 226
7.15.3.7 eVhal_PBEType . . . 226
7.15.3.8 eVoutput_Format . . . 226
7.15.3.9 eVrc_Codes . . . 226
7.15.3.10 eVsol_Meth . . . 227
7.15.3.11 eVsurf_Meth . . . 227

7.16 Matrix wrapper class . . . 228
7.16.1 Detailed Description . . . 228
7.17 Vparam class . . . 229
7.17.1 Detailed Description . . . 230
7.17.2 Function Documentation . . . 231
7.17.2.1 readFlatFileLine . . . 231
7.17.2.2 readXMLFileAtom . . . 231
7.17.2.3 Vparam_AtomData_copyFrom . . . 232
7.17.2.4 Vparam_AtomData_copyTo . . . 233
7.17.2.5 Vparam_AtomData_ctor . . . 233
7.17.2.6 Vparam_AtomData_ctor2 . . . 234
7.17.2.7 Vparam_AtomData_dtor . . . 234
7.17.2.8 Vparam_AtomData_dtor2 . . . 235
7.17.2.9 Vparam_ctor . . . 235
7.17.2.10 Vparam_ctor2 . . . 236
7.17.2.11 Vparam_dtor . . . 236
7.17.2.12 Vparam_dtor2 . . . 237
7.17.2.13 Vparam_getAtomData . . . 238
7.17.2.14 Vparam_getResData . . . 239
7.17.2.15 Vparam_memChk . . . 240
7.17.2.16 Vparam_readFlatFile . . . 240
7.17.2.17 Vparam_readXMLFile . . . 241
7.17.2.18 Vparam_ResData_copyTo . . . 242
7.17.2.19 Vparam_ResData_ctor . . . 243
7.17.2.20 Vparam_ResData_ctor2 . . . 244
7.17.2.21 Vparam_ResData_dtor . . . 244
7.17.2.22 Vparam_ResData_dtor2 . . . 245
7.18 Vpbe class . . . 246
7.18.1 Detailed Description . . . 247
7.18.2 Function Documentation . . . 248
7.18.2.1 Vpbe_ctor . . . 248
7.18.2.2 Vpbe_ctor2 . . . 249
7.18.2.3 Vpbe_dtor . . . 250
7.18.2.4 Vpbe_dtor2 . . . 250
7.18.2.5 Vpbe_getBulkIonicStrength . . . 251
7.18.2.6 Vpbe_getCoulombEnergy1 . . . 252
7.18.2.7 Vpbe_getDeblen . . . 253

7.18.2.8 Vpbe_getGamma . . . 254
7.18.2.9 Vpbe_getIons . . . 254
7.18.2.10 Vpbe_getLmem . . . 255
7.18.2.11 Vpbe_getMaxIonRadius . . . 255
7.18.2.12 Vpbe_getmembraneDiel . . . 256
7.18.2.13 Vpbe_getmemv . . . 257
7.18.2.14 Vpbe_getSoluteCenter . . . 257
7.18.2.15 Vpbe_getSoluteCharge . . . 257
7.18.2.16 Vpbe_getSoluteDiel . . . 258
7.18.2.17 Vpbe_getSoluteRadius . . . 258
7.18.2.18 Vpbe_getSoluteXlen . . . 259
7.18.2.19 Vpbe_getSoluteYlen . . . 259
7.18.2.20 Vpbe_getSoluteZlen . . . 260
7.18.2.21 Vpbe_getSolventDiel . . . 260
7.18.2.22 Vpbe_getSolventRadius . . . 261
7.18.2.23 Vpbe_getTemperature . . . 261
7.18.2.24 Vpbe_getVacc . . . 262
7.18.2.25 Vpbe_getValist . . . 263
7.18.2.26 Vpbe_getXkappa . . . 264
7.18.2.27 Vpbe_getZkappa2 . . . 264
7.18.2.28 Vpbe_getZmagic . . . 265
7.18.2.29 Vpbe_getzmem . . . 266
7.18.2.30 Vpbe_memChk . . . 267
7.19 Vstring class . . . 268
7.19.1 Detailed Description . . . 268
7.19.2 Function Documentation . . . 268
7.19.2.1 Vstring_isdigit . . . 268
7.19.2.2 Vstring_strcasecmp . . . 268
7.19.2.3 Vstring_wrappedtext . . . 270
7.20 Vunit class . . . 271
7.20.1 Detailed Description . . . 271
7.21 Vgrid class . . . 272
7.21.1 Detailed Description . . . 273
7.21.2 Function Documentation . . . 274
7.21.2.1 Vgrid_ctor . . . 274
7.21.2.2 Vgrid_ctor2 . . . 274
7.21.2.3 Vgrid_curvature . . . 275

7.21.2.4 Vgrid_dtor . . . 276
7.21.2.5 Vgrid_dtor2 . . . 276
7.21.2.6 Vgrid_gradient . . . 277
7.21.2.7 Vgrid_integrate . . . 277
7.21.2.8 Vgrid_memChk . . . 278
7.21.2.9 Vgrid_normH1 . . . 278
7.21.2.10 Vgrid_normL1 . . . 279
7.21.2.11 Vgrid_normL2 . . . 279
7.21.2.12 Vgrid_normLinf . . . 280
7.21.2.13 Vgrid_readDX . . . 280
7.21.2.14 Vgrid_readGZ . . . 281
7.21.2.15 Vgrid_seminormH1 . . . 281
7.21.2.16 Vgrid_value . . . 282
7.21.2.17 Vgrid_writeDX . . . 283
7.21.2.18 Vgrid_writeUHBD . . . 283
7.22 Vmgrid class . . . 285
7.22.1 Detailed Description . . . 286
7.22.2 Function Documentation . . . 286
7.22.2.1 Vmgrid_addGrid . . . 286
7.22.2.2 Vmgrid_ctor . . . 286
7.22.2.3 Vmgrid_ctor2 . . . 286
7.22.2.4 Vmgrid_curvature . . . 287
7.22.2.5 Vmgrid_dtor . . . 287
7.22.2.6 Vmgrid_dtor2 . . . 287
7.22.2.7 Vmgrid_getGridByNum . . . 288
7.22.2.8 Vmgrid_getGridByPoint . . . 288
7.22.2.9 Vmgrid_gradient . . . 288
7.22.2.10 Vmgrid_value . . . 289
7.23 Multi-grid class . . . 290
7.23.1 Detailed Description . . . 290
7.24 Vopot class . . . 291
7.24.1 Detailed Description . . . 291
7.24.2 Function Documentation . . . 292
7.24.2.1 Vopot_ctor . . . 292
7.24.2.2 Vopot_ctor2 . . . 292
7.24.2.3 Vopot_curvature . . . 292
7.24.2.4 Vopot_dtor . . . 293

7.24.2.5 Vopot_dtor2 . . . 293
7.24.2.6 Vopot_gradient . . . 293
7.24.2.7 Vopot_pot . . . 294
7.25 Vpmg class . . . 295
7.25.1 Detailed Description . . . 298
7.25.2 Function Documentation . . . 298
7.25.2.1 bcolcomp . . . 298
7.25.2.2 bcolcomp2 . . . 299
7.25.2.3 bcolcomp3 . . . 300
7.25.2.4 bcolcomp4 . . . 301
7.25.2.5 pcolcomp . . . 302
7.25.2.6 Vpackmg . . . 302
7.25.2.7 Vpmg_ctor . . . 303
7.25.2.8 Vpmg_ctor2 . . . 304
7.25.2.9 Vpmg_dbDirectPolForce . . . 306
7.25.2.10 Vpmg_dbForce . . . 306
7.25.2.11 Vpmg_dbMutualPolForce . . . 307
7.25.2.12 Vpmg_dbNLDirectPolForce . . . 308
7.25.2.13 Vpmg_dbPermanentMultipoleForce . . . 308
7.25.2.14 Vpmg_dielEnergy . . . 308
7.25.2.15 Vpmg_dielGradNorm . . . 309
7.25.2.16 Vpmg_dtor . . . 310
7.25.2.17 Vpmg_dtor2 . . . 311
7.25.2.18 Vpmg_energy . . . 311
7.25.2.19 Vpmg_fieldSpline4 . . . 312
7.25.2.20 Vpmg_fillArray . . . 313
7.25.2.21 Vpmg_fillco . . . 313
7.25.2.22 Vpmg_force . . . 315
7.25.2.23 Vpmg_ibDirectPolForce . . . 317
7.25.2.24 Vpmg_ibForce . . . 317
7.25.2.25 Vpmg_ibMutualPolForce . . . 318
7.25.2.26 Vpmg_ibNLDirectPolForce . . . 319
7.25.2.27 Vpmg_ibPermanentMultipoleForce . . . 319
7.25.2.28 Vpmg_memChk . . . 319
7.25.2.29 Vpmg_printColComp . . . 320
7.25.2.30 Vpmg_qfAtomEnergy . . . 321
7.25.2.31 Vpmg_qfDirectPolForce . . . 321

7.25.2.32 Vpmg_qfEnergy . . . 322
7.25.2.33 Vpmg_qfForce . . . 323
7.25.2.34 Vpmg_qfMutualPolForce . . . 325
7.25.2.35 Vpmg_qfNLDirectPolForce . . . 325
7.25.2.36 Vpmg_qfPermanentMultipoleEnergy . . . 325
7.25.2.37 Vpmg_qfPermanentMultipoleForce . . . 326
7.25.2.38 Vpmg_qmEnergy . . . 326
7.25.2.39 Vpmg_setPart . . . 327
7.25.2.40 Vpmg_solve . . . 328
7.25.2.41 Vpmg_solveLaplace . . . 330
7.25.2.42 Vpmg_unsetPart . . . 330
7.26 Vpmgp class . . . 332
7.26.1 Detailed Description . . . 332
7.26.2 Function Documentation . . . 333
7.26.2.1 Vpmgp_ctor . . . 333
7.26.2.2 Vpmgp_ctor2 . . . 333
7.26.2.3 Vpmgp_dtor . . . 333
7.26.2.4 Vpmgp_dtor2 . . . 334
7.26.2.5 Vpmgp_makeCoarse . . . 334
7.26.2.6 Vpmgp_size . . . 334
7.27 C translation of Holst group PMG code . . . 336
7.27.1 Detailed Description . . . 340
7.27.2 Macro Definition Documentation . . . 340
7.27.2.1 HARMO2 . . . 340
7.27.2.2 MAXIONS . . . 341
7.27.3 Function Documentation . . . 342
7.27.3.1 Vaxrand . . . 342
7.27.3.2 Vazeros . . . 342
7.27.3.3 VbuildA . . . 343
7.27.3.4 Vbuildband . . . 345
7.27.3.5 Vbuildband1_27 . . . 348
7.27.3.6 Vbuildband1_7 . . . 349
7.27.3.7 VbuildG . . . 350
7.27.3.8 VbuildG_1 . . . 352
7.27.3.9 VbuildG_27 . . . 354
7.27.3.10 VbuildG_7 . . . 357
7.27.3.11 Vbuildgaler0 . . . 359

7.27.3.12 Vbuildops . . . 360
7.27.3.13 VbuildP . . . 363
7.27.3.14 Vbuildstr . . . 365
7.27.3.15 Vc_vec . . . 366
7.27.3.16 Vc_vecpmg . . . 367
7.27.3.17 Vc_vecsmpbe . . . 367
7.27.3.18 Vcghs . . . 368
7.27.3.19 Vdc_vec . . . 370
7.27.3.20 Vdpbsl . . . 371
7.27.3.21 Vextrac . . . 373
7.27.3.22 VfboundPMG . . . 374
7.27.3.23 VfboundPMG00 . . . 375
7.27.3.24 Vfmvfas . . . 376
7.27.3.25 Vfnewton . . . 380
7.27.3.26 Vgetjac . . . 384
7.27.3.27 Vgsrb . . . 385
7.27.3.28 VinterpPMG . . . 388
7.27.3.29 Vipower . . . 388
7.27.3.30 Vmatvec . . . 391
7.27.3.31 Vmgdriv . . . 393
7.27.3.32 Vmgdriv2 . . . 397
7.27.3.33 Vmgsz . . . 400
7.27.3.34 Vmresid . . . 402
7.27.3.35 Vmvcs . . . 403
7.27.3.36 Vmvfas . . . 408
7.27.3.37 Vmypdefinitlpbe . . . 411
7.27.3.38 Vmypdefinitnpbe . . . 412
7.27.3.39 Vmypdefinitsmpbe . . . 413
7.27.3.40 Vnewdriv . . . 414
7.27.3.41 Vnewdriv2 . . . 418
7.27.3.42 Vnewton . . . 421
7.27.3.43 Vnmatvec . . . 424
7.27.3.44 Vnmresid . . . 425
7.27.3.45 Vnsmooth . . . 425
7.27.3.46 Vpower . . . 426
7.27.3.47 Vprtmatd . . . 429
7.27.3.48 Vrestrc . . . 430

7.27.3.49 Vsmooth . . . 431
7.27.3.50 Vxaxpy . . . 434
7.27.3.51 Vxcopy . . . 435
7.27.3.52 Vxcopy_large . . . 437
7.27.3.53 Vxcopy_small . . . 438
7.27.3.54 Vxdot . . . 439
7.27.3.55 Vxnrm1 . . . 439
7.27.3.56 Vxnrm2 . . . 440
7.27.3.57 Vxscal . . . 441

8 Data Structure Documentation **443**

8.1 sAPOLparm Struct Reference . . . 443
8.1.1 Detailed Description . . . 444
8.1.2 Field Documentation . . . 444
8.1.2.1 bconc . . . 444
8.1.2.2 calcenergy . . . 444
8.1.2.3 calcforce . . . 444
8.1.2.4 dpos . . . 444
8.1.2.5 gamma . . . 444
8.1.2.6 grid . . . 445
8.1.2.7 molid . . . 445
8.1.2.8 parsed . . . 445
8.1.2.9 press . . . 445
8.1.2.10 sasa . . . 445
8.1.2.11 sav . . . 445
8.1.2.12 sdens . . . 445
8.1.2.13 setbconc . . . 445
8.1.2.14 setcalcenergy . . . 446
8.1.2.15 setcalcforce . . . 446
8.1.2.16 setdpos . . . 446
8.1.2.17 setgamma . . . 446
8.1.2.18 setgrid . . . 446
8.1.2.19 setmolid . . . 447
8.1.2.20 setpress . . . 447
8.1.2.21 setsdens . . . 447
8.1.2.22 setsrad . . . 447
8.1.2.23 setsrfm . . . 447

8.1.2.24 setswin . . . 448
8.1.2.25 settemp . . . 448
8.1.2.26 setwat . . . 448
8.1.2.27 srad . . . 448
8.1.2.28 srfm . . . 448
8.1.2.29 swin . . . 448
8.1.2.30 temp . . . 448
8.1.2.31 totForce . . . 449
8.1.2.32 watepsilon . . . 449
8.1.2.33 watsigma . . . 449
8.1.2.34 wcaEnergy . . . 449
8.2 sFEMparm Struct Reference . . . 449
8.2.1 Detailed Description . . . 450
8.2.2 Field Documentation . . . 450
8.2.2.1 akeyPRE . . . 450
8.2.2.2 akeySOLVE . . . 450
8.2.2.3 ekey . . . 450
8.2.2.4 etol . . . 450
8.2.2.5 glen . . . 450
8.2.2.6 maxsolve . . . 451
8.2.2.7 maxvert . . . 451
8.2.2.8 meshID . . . 451
8.2.2.9 parsed . . . 451
8.2.2.10 pkey . . . 451
8.2.2.11 setakeyPRE . . . 451
8.2.2.12 setakeySOLVE . . . 451
8.2.2.13 setekey . . . 451
8.2.2.14 setetol . . . 451
8.2.2.15 setglen . . . 452
8.2.2.16 setmaxsolve . . . 452
8.2.2.17 setmaxvert . . . 452
8.2.2.18 settargetNum . . . 452
8.2.2.19 settargetRes . . . 452
8.2.2.20 settype . . . 452
8.2.2.21 targetNum . . . 452
8.2.2.22 targetRes . . . 452
8.2.2.23 type . . . 453

8.2.2.24 useMesh . . . 453
8.3 sMGparm Struct Reference . . . 453
8.3.1 Detailed Description . . . 454
8.3.2 Field Documentation . . . 454
8.3.2.1 async . . . 454
8.3.2.2 ccenter . . . 454
8.3.2.3 ccentmol . . . 455
8.3.2.4 ccmeth . . . 455
8.3.2.5 center . . . 455
8.3.2.6 centmol . . . 455
8.3.2.7 cglen . . . 455
8.3.2.8 chgm . . . 455
8.3.2.9 chgs . . . 455
8.3.2.10 cmeth . . . 455
8.3.2.11 dime . . . 456
8.3.2.12 etol . . . 456
8.3.2.13 fcenter . . . 456
8.3.2.14 fcentmol . . . 456
8.3.2.15 fcmeth . . . 456
8.3.2.16 fglen . . . 456
8.3.2.17 glen . . . 456
8.3.2.18 grid . . . 456
8.3.2.19 method . . . 456
8.3.2.20 nlev . . . 457
8.3.2.21 nonlintype . . . 457
8.3.2.22 ofrac . . . 457
8.3.2.23 parsed . . . 457
8.3.2.24 partDisjCenter . . . 457
8.3.2.25 partDisjLength . . . 457
8.3.2.26 partDisjOwnSide . . . 457
8.3.2.27 pdime . . . 457
8.3.2.28 proc_rank . . . 458
8.3.2.29 proc_size . . . 458
8.3.2.30 setasync . . . 458
8.3.2.31 setcgcent . . . 458
8.3.2.32 setcglen . . . 458
8.3.2.33 setchgm . . . 458

8.3.2.34 setdime . . . 459
8.3.2.35 setetol . . . 459
8.3.2.36 setfgcent . . . 459
8.3.2.37 setfglen . . . 459
8.3.2.38 setgcent . . . 459
8.3.2.39 setglen . . . 460
8.3.2.40 setgrid . . . 460
8.3.2.41 setmethod . . . 460
8.3.2.42 setnlev . . . 460
8.3.2.43 setnonlintype . . . 460
8.3.2.44 setofrac . . . 461
8.3.2.45 setpdime . . . 461
8.3.2.46 setrank . . . 461
8.3.2.47 setsize . . . 461
8.3.2.48 setUseAqua . . . 461
8.3.2.49 type . . . 462
8.3.2.50 useAqua . . . 462
8.4 sNOsh Struct Reference . . . 462
8.4.1 Detailed Description . . . 463
8.4.2 Field Documentation . . . 464
8.4.2.1 alist . . . 464
8.4.2.2 apol . . . 464
8.4.2.3 apol2calc . . . 464
8.4.2.4 apolname . . . 464
8.4.2.5 bogus . . . 464
8.4.2.6 calc . . . 464
8.4.2.7 chargefmt . . . 464
8.4.2.8 chargepath . . . 464
8.4.2.9 dielfmt . . . 465
8.4.2.10 dielXpath . . . 465
8.4.2.11 dielYpath . . . 465
8.4.2.12 dielZpath . . . 465
8.4.2.13 elec . . . 465
8.4.2.14 elec2calc . . . 465
8.4.2.15 elecname . . . 465
8.4.2.16 gotparm . . . 465
8.4.2.17 ispara . . . 466

8.4.2.18 kappafmt . . . 466
8.4.2.19 kappapath . . . 466
8.4.2.20 meshfmt . . . 466
8.4.2.21 meshpath . . . 466
8.4.2.22 molfmt . . . 466
8.4.2.23 molpath . . . 466
8.4.2.24 napol . . . 466
8.4.2.25 ncalc . . . 466
8.4.2.26 ncharge . . . 467
8.4.2.27 ndiel . . . 467
8.4.2.28 nelec . . . 467
8.4.2.29 nkappa . . . 467
8.4.2.30 nmesh . . . 467
8.4.2.31 nmol . . . 467
8.4.2.32 npot . . . 467
8.4.2.33 nprint . . . 467
8.4.2.34 parmfmt . . . 467
8.4.2.35 parmpath . . . 468
8.4.2.36 parsed . . . 468
8.4.2.37 potfmt . . . 468
8.4.2.38 potpath . . . 468
8.4.2.39 printcalc . . . 468
8.4.2.40 printnarg . . . 468
8.4.2.41 printop . . . 468
8.4.2.42 printwhat . . . 468
8.4.2.43 proc_rank . . . 469
8.4.2.44 proc_size . . . 469
8.5 sNOsh_calc Struct Reference . . . 469
8.5.1 Detailed Description . . . 469
8.5.2 Field Documentation . . . 470
8.5.2.1 apolparm . . . 470
8.5.2.2 calctype . . . 470
8.5.2.3 femparm . . . 470
8.5.2.4 mgparm . . . 470
8.5.2.5 pbeparm . . . 470
8.6 sPBEparm Struct Reference . . . 470
8.6.1 Detailed Description . . . 472

8.6.2 Field Documentation . . . 472
8.6.2.1 bcfl . . . 472
8.6.2.2 calcenergy . . . 472
8.6.2.3 calcforce . . . 472
8.6.2.4 chargeMapID . . . 472
8.6.2.5 dielMapID . . . 473
8.6.2.6 ionc . . . 473
8.6.2.7 ionq . . . 473
8.6.2.8 ionr . . . 473
8.6.2.9 kappaMapID . . . 473
8.6.2.10 Lmem . . . 473
8.6.2.11 mdie . . . 473
8.6.2.12 memv . . . 473
8.6.2.13 molid . . . 473
8.6.2.14 nion . . . 474
8.6.2.15 numwrite . . . 474
8.6.2.16 parsed . . . 474
8.6.2.17 pbetype . . . 474
8.6.2.18 pdie . . . 474
8.6.2.19 potMapID . . . 474
8.6.2.20 sdens . . . 474
8.6.2.21 sdie . . . 474
8.6.2.22 setbcfl . . . 474
8.6.2.23 setcalcenergy . . . 475
8.6.2.24 setcalcforce . . . 475
8.6.2.25 setion . . . 475
8.6.2.26 setLmem . . . 475
8.6.2.27 setmdie . . . 475
8.6.2.28 setmemv . . . 475
8.6.2.29 setmolid . . . 476
8.6.2.30 setnion . . . 476
8.6.2.31 setpbetype . . . 476
8.6.2.32 setpdie . . . 476
8.6.2.33 setsdens . . . 476
8.6.2.34 setsdie . . . 477
8.6.2.35 setsmsize . . . 477
8.6.2.36 setsmvolume . . . 477

8.6.2.37 setsrad . . . 477
8.6.2.38 setsrfm . . . 477
8.6.2.39 setswin . . . 478
8.6.2.40 settemp . . . 478
8.6.2.41 setwritemat . . . 478
8.6.2.42 setzmem . . . 478
8.6.2.43 smsize . . . 478
8.6.2.44 smvolume . . . 478
8.6.2.45 srad . . . 478
8.6.2.46 srfm . . . 479
8.6.2.47 swin . . . 479
8.6.2.48 temp . . . 479
8.6.2.49 useChargeMap . . . 479
8.6.2.50 useDielMap . . . 479
8.6.2.51 useKappaMap . . . 479
8.6.2.52 usePotMap . . . 479
8.6.2.53 writefmt . . . 479
8.6.2.54 writemat . . . 479
8.6.2.55 writematflag . . . 480
8.6.2.56 writematstem . . . 480
8.6.2.57 writestem . . . 480
8.6.2.58 writetype . . . 480
8.6.2.59 zmem . . . 480
8.7 sVacc Struct Reference . . . 480
8.7.1 Detailed Description . . . 481
8.7.2 Field Documentation . . . 482
8.7.2.1 acc . . . 482
8.7.2.2 alist . . . 482
8.7.2.3 atomFlags . . . 482
8.7.2.4 clist . . . 482
8.7.2.5 mem . . . 482
8.7.2.6 refSphere . . . 482
8.7.2.7 surf . . . 482
8.7.2.8 surf_density . . . 482
8.8 sVaccSurf Struct Reference . . . 483
8.8.1 Detailed Description . . . 483
8.8.2 Field Documentation . . . 483

8.8.2.1 area . . . 483
8.8.2.2 bpts . . . 483
8.8.2.3 mem . . . 483
8.8.2.4 npts . . . 483
8.8.2.5 probe_radius . . . 484
8.8.2.6 xpts . . . 484
8.8.2.7 ypts . . . 484
8.8.2.8 zpts . . . 484
8.9 sValist Struct Reference . . . 484
8.9.1 Detailed Description . . . 485
8.9.2 Field Documentation . . . 485
8.9.2.1 atoms . . . 485
8.9.2.2 center . . . 485
8.9.2.3 charge . . . 485
8.9.2.4 maxcrd . . . 485
8.9.2.5 maxrad . . . 485
8.9.2.6 mincrd . . . 486
8.9.2.7 number . . . 486
8.9.2.8 vmem . . . 486
8.10 sVatom Struct Reference . . . 486
8.10.1 Detailed Description . . . 486
8.10.2 Field Documentation . . . 487
8.10.2.1 atomName . . . 487
8.10.2.2 charge . . . 487
8.10.2.3 epsilon . . . 487
8.10.2.4 id . . . 487
8.10.2.5 partID . . . 487
8.10.2.6 position . . . 487
8.10.2.7 radius . . . 487
8.10.2.8 resName . . . 487
8.11 sVclist Struct Reference . . . 488
8.11.1 Detailed Description . . . 488
8.11.2 Field Documentation . . . 489
8.11.2.1 alist . . . 489
8.11.2.2 cells . . . 489
8.11.2.3 lower_corner . . . 489
8.11.2.4 max_radius . . . 489

8.11.2.5 mode . . . 489
8.11.2.6 n . . . 489
8.11.2.7 npts . . . 489
8.11.2.8 spacs . . . 489
8.11.2.9 upper_corner . . . 490
8.11.2.10 vmem . . . 490
8.12 sVclistCell Struct Reference . . . 490
8.12.1 Detailed Description . . . 490
8.12.2 Field Documentation . . . 491
8.12.2.1 atoms . . . 491
8.12.2.2 natoms . . . 491
8.13 sVcsm Struct Reference . . . 491
8.13.1 Detailed Description . . . 492
8.13.2 Field Documentation . . . 492
8.13.2.1 alist . . . 492
8.13.2.2 gm . . . 492
8.13.2.3 initFlag . . . 492
8.13.2.4 msimp . . . 492
8.13.2.5 natom . . . 493
8.13.2.6 nqsm . . . 493
8.13.2.7 nsimp . . . 493
8.13.2.8 nsqm . . . 493
8.13.2.9 qsm . . . 493
8.13.2.10 sqm . . . 493
8.13.2.11 vmem . . . 493
8.14 sVfetk Struct Reference . . . 493
8.14.1 Detailed Description . . . 495
8.14.2 Field Documentation . . . 495
8.14.2.1 am . . . 495
8.14.2.2 aprx . . . 495
8.14.2.3 csm . . . 495
8.14.2.4 feparm . . . 495
8.14.2.5 gm . . . 495
8.14.2.6 gues . . . 495
8.14.2.7 level . . . 496
8.14.2.8 lkey . . . 496
8.14.2.9 lmax . . . 496

8.14.2.10 lprec . . . 496
8.14.2.11 ltol . . . 496
8.14.2.12 nkey . . . 496
8.14.2.13 nmax . . . 496
8.14.2.14 ntol . . . 496
8.14.2.15 pbe . . . 496
8.14.2.16 pbeparm . . . 497
8.14.2.17 pde . . . 497
8.14.2.18 pjac . . . 497
8.14.2.19 type . . . 497
8.14.2.20 vmem . . . 497
8.15 sVfetk_LocalVar Struct Reference . . . 497
8.15.1 Detailed Description . . . 499
8.15.2 Field Documentation . . . 499
8.15.2.1 A . . . 499
8.15.2.2 B . . . 499
8.15.2.3 d2W . . . 499
8.15.2.4 DB . . . 500
8.15.2.5 delta . . . 500
8.15.2.6 DFu_wv . . . 500
8.15.2.7 dU . . . 500
8.15.2.8 dW . . . 500
8.15.2.9 F . . . 500
8.15.2.10 fetk . . . 500
8.15.2.11 fType . . . 500
8.15.2.12 Fu_v . . . 500
8.15.2.13 green . . . 501
8.15.2.14 initGreen . . . 501
8.15.2.15 ionConc . . . 501
8.15.2.16 ionQ . . . 501
8.15.2.17 ionRadii . . . 501
8.15.2.18 ionstr . . . 501
8.15.2.19 jumpDiel . . . 501
8.15.2.20 nion . . . 501
8.15.2.21 nvec . . . 501
8.15.2.22 nverts . . . 502
8.15.2.23 simp . . . 502

8.15.2.24 sType . . . 502
8.15.2.25 U . . . 502
8.15.2.26 u_D . . . 502
8.15.2.27 u_T . . . 502
8.15.2.28 verts . . . 502
8.15.2.29 vx . . . 502
8.15.2.30 W . . . 502
8.15.2.31 xq . . . 503
8.15.2.32 zkappa2 . . . 503
8.15.2.33 zks2 . . . 503
8.16 sVgreen Struct Reference . . . 503
8.16.1 Detailed Description . . . 504
8.16.2 Field Documentation . . . 504
8.16.2.1 alist . . . 504
8.16.2.2 np . . . 504
8.16.2.3 qp . . . 504
8.16.2.4 vmem . . . 504
8.16.2.5 xp . . . 504
8.16.2.6 yp . . . 505
8.16.2.7 zp . . . 505
8.17 sVgrid Struct Reference . . . 505
8.17.1 Detailed Description . . . 505
8.17.2 Field Documentation . . . 506
8.17.2.1 ctordata . . . 506
8.17.2.2 data . . . 506
8.17.2.3 hx . . . 506
8.17.2.4 hy . . . 506
8.17.2.5 hzed . . . 506
8.17.2.6 mem . . . 506
8.17.2.7 nx . . . 506
8.17.2.8 ny . . . 506
8.17.2.9 nz . . . 507
8.17.2.10 readdata . . . 507
8.17.2.11 xmax . . . 507
8.17.2.12 xmin . . . 507
8.17.2.13 ymax . . . 507
8.17.2.14 ymin . . . 507

8.17.2.15 zmax . . . 507
8.17.2.16 zmin . . . 507
8.18 sVmgrid Struct Reference . . . 508
8.18.1 Detailed Description . . . 508
8.18.2 Field Documentation . . . 508
8.18.2.1 grids . . . 508
8.18.2.2 ngrids . . . 508
8.19 sVmultigrid Struct Reference . . . 509
8.19.1 Detailed Description . . . 510
8.19.2 Field Documentation . . . 510
8.19.2.1 coarse_count . . . 510
8.19.2.2 coarse_solver . . . 510
8.19.2.3 coarsen_method . . . 510
8.19.2.4 discrete_method . . . 510
8.19.2.5 fine_count . . . 511
8.19.2.6 integer_workspace_limit . . . 511
8.19.2.7 integer_workspace_size . . . 511
8.19.2.8 ipcon . . . 511
8.19.2.9 irite . . . 511
8.19.2.10 level_count . . . 511
8.19.2.11 nonlin . . . 511
8.19.2.12 prolong_method . . . 512
8.19.2.13 real_workspace_limit . . . 512
8.19.2.14 real_workspace_size . . . 512
8.19.2.15 smooth_method . . . 512
8.19.2.16 stop_criterion . . . 512
8.19.2.17 verbosity . . . 513
8.19.2.18 x_size . . . 513
8.19.2.19 y_size . . . 513
8.19.2.20 z_size . . . 513
8.20 sVopot Struct Reference . . . 513
8.20.1 Detailed Description . . . 514
8.20.2 Field Documentation . . . 515
8.20.2.1 bcfl . . . 515
8.20.2.2 mgrid . . . 515
8.20.2.3 pbe . . . 515
8.21 sVparam_AtomData Struct Reference . . . 515

8.21.1 Detailed Description . . . 515
8.21.2 Field Documentation . . . 516
8.21.2.1 atomName . . . 516
8.21.2.2 charge . . . 516
8.21.2.3 epsilon . . . 516
8.21.2.4 radius . . . 516
8.21.2.5 resName . . . 516
8.22 sVpbe Struct Reference . . . 517
8.22.1 Detailed Description . . . 518
8.22.2 Field Documentation . . . 518
8.22.2.1 acc . . . 518
8.22.2.2 alist . . . 518
8.22.2.3 bulkIonicStrength . . . 519
8.22.2.4 clist . . . 519
8.22.2.5 deblen . . . 519
8.22.2.6 ionConc . . . 519
8.22.2.7 ionQ . . . 519
8.22.2.8 ionRadii . . . 519
8.22.2.9 ipkey . . . 519
8.22.2.10 L . . . 519
8.22.2.11 maxIonRadius . . . 519
8.22.2.12 membraneDiel . . . 520
8.22.2.13 numIon . . . 520
8.22.2.14 param2Flag . . . 520
8.22.2.15 paramFlag . . . 520
8.22.2.16 smsize . . . 520
8.22.2.17 smvolume . . . 520
8.22.2.18 soluteCenter . . . 520
8.22.2.19 soluteCharge . . . 520
8.22.2.20 soluteDiel . . . 520
8.22.2.21 soluteRadius . . . 521
8.22.2.22 soluteXlen . . . 521
8.22.2.23 soluteYlen . . . 521
8.22.2.24 soluteZlen . . . 521
8.22.2.25 solventDiel . . . 521
8.22.2.26 solventRadius . . . 521
8.22.2.27 T . . . 521

8.22.2.28 V . . . 521
8.22.2.29 vmem . . . 521
8.22.2.30 xkappa . . . 522
8.22.2.31 z_mem . . . 522
8.22.2.32 zkappa2 . . . 522
8.22.2.33 zmagic . . . 522
8.23 sVpee Struct Reference . . . 522
8.23.1 Detailed Description . . . 522
8.23.2 Field Documentation . . . 523
8.23.2.1 gm . . . 523
8.23.2.2 killFlag . . . 523
8.23.2.3 killParam . . . 523
8.23.2.4 localPartCenter . . . 523
8.23.2.5 localPartID . . . 523
8.23.2.6 localPartRadius . . . 523
8.23.2.7 mem . . . 523
8.24 sVpmg Struct Reference . . . 524
8.24.1 Detailed Description . . . 526
8.24.2 Field Documentation . . . 526
8.24.2.1 a1cf . . . 526
8.24.2.2 a2cf . . . 526
8.24.2.3 a3cf . . . 526
8.24.2.4 ccf . . . 526
8.24.2.5 charge . . . 526
8.24.2.6 chargeMap . . . 526
8.24.2.7 chargeMeth . . . 527
8.24.2.8 chargeSrc . . . 527
8.24.2.9 dielXMap . . . 527
8.24.2.10 dielYMap . . . 527
8.24.2.11 dielZMap . . . 527
8.24.2.12 epsx . . . 527
8.24.2.13 epsy . . . 527
8.24.2.14 epsz . . . 527
8.24.2.15 extDiEnergy . . . 527
8.24.2.16 extNpEnergy . . . 528
8.24.2.17 extQfEnergy . . . 528
8.24.2.18 extQmEnergy . . . 528

8.24.2.19 fcf . . . 528
8.24.2.20 filled . . . 528
8.24.2.21 gxcf . . . 528
8.24.2.22 gycf . . . 528
8.24.2.23 gzcf . . . 528
8.24.2.24 iparm . . . 528
8.24.2.25 iwork . . . 529
8.24.2.26 kappa . . . 529
8.24.2.27 kappaMap . . . 529
8.24.2.28 pbe . . . 529
8.24.2.29 pmgp . . . 529
8.24.2.30 pot . . . 529
8.24.2.31 potMap . . . 529
8.24.2.32 pvec . . . 529
8.24.2.33 rparm . . . 529
8.24.2.34 rwork . . . 530
8.24.2.35 splineWin . . . 530
8.24.2.36 surfMeth . . . 530
8.24.2.37 tcf . . . 530
8.24.2.38 u . . . 530
8.24.2.39 useChargeMap . . . 530
8.24.2.40 useDielXMap . . . 530
8.24.2.41 useDielYMap . . . 530
8.24.2.42 useDielZMap . . . 530
8.24.2.43 useKappaMap . . . 531
8.24.2.44 usePotMap . . . 531
8.24.2.45 vmem . . . 531
8.24.2.46 xf . . . 531
8.24.2.47 yf . . . 531
8.24.2.48 zf . . . 531
8.25 sVpmgp Struct Reference . . . 531
8.25.1 Detailed Description . . . 533
8.25.2 Field Documentation . . . 533
8.25.2.1 bcfl . . . 533
8.25.2.2 errtol . . . 533
8.25.2.3 hx . . . 533
8.25.2.4 hy . . . 533

8.25.2.5 hzed . . . 533
8.25.2.6 iinfo . . . 533
8.25.2.7 ipcon . . . 534
8.25.2.8 iperf . . . 534
8.25.2.9 ipkey . . . 534
8.25.2.10 irite . . . 534
8.25.2.11 istop . . . 535
8.25.2.12 itmax . . . 535
8.25.2.13 key . . . 535
8.25.2.14 meth . . . 535
8.25.2.15 mgcoar . . . 536
8.25.2.16 mgdisc . . . 536
8.25.2.17 mgkey . . . 536
8.25.2.18 mgprol . . . 536
8.25.2.19 mgsmoo . . . 536
8.25.2.20 mgsolv . . . 537
8.25.2.21 n_ipc . . . 537
8.25.2.22 n_iz . . . 537
8.25.2.23 n_rpc . . . 537
8.25.2.24 narr . . . 537
8.25.2.25 narrc . . . 537
8.25.2.26 nc . . . 537
8.25.2.27 nf . . . 538
8.25.2.28 niwk . . . 538
8.25.2.29 nlev . . . 538
8.25.2.30 nonlin . . . 538
8.25.2.31 nrwk . . . 538
8.25.2.32 nu1 . . . 538
8.25.2.33 nu2 . . . 538
8.25.2.34 nx . . . 538
8.25.2.35 nxc . . . 539
8.25.2.36 ny . . . 539
8.25.2.37 nyc . . . 539
8.25.2.38 nz . . . 539
8.25.2.39 nzc . . . 539
8.25.2.40 omegal . . . 539
8.25.2.41 omegan . . . 539

8.25.2.42 xcent . . . 539
8.25.2.43 xlen . . . 539
8.25.2.44 xmax . . . 540
8.25.2.45 xmin . . . 540
8.25.2.46 ycent . . . 540
8.25.2.47 ylen . . . 540
8.25.2.48 ymax . . . 540
8.25.2.49 ymin . . . 540
8.25.2.50 zcent . . . 540
8.25.2.51 zlen . . . 540
8.25.2.52 zmax . . . 540
8.25.2.53 zmin . . . 541
8.26 Vparam Struct Reference . . . 541
8.26.1 Detailed Description . . . 541
8.26.2 Field Documentation . . . 542
8.26.2.1 nResData . . . 542
8.26.2.2 resData . . . 542
8.26.2.3 vmem . . . 542
8.27 Vparam_ResData Struct Reference . . . 542
8.27.1 Detailed Description . . . 543
8.27.2 Field Documentation . . . 543
8.27.2.1 atomData . . . 543
8.27.2.2 name . . . 543
8.27.2.3 nAtomData . . . 543
8.27.2.4 vmem . . . 543

9 File Documentation **545**
9.1 doc/license/LICENSE.h File Reference . . . 545
9.1.1 Detailed Description . . . 545
9.2 LICENSE.h . . . 546
9.3 src/aaa_inc/apbs/apbs.h File Reference . . . 546
9.3.1 Detailed Description . . . 547
9.4 apbs.h . . . 548
9.5 src/aaa_lib/apbs_link.c File Reference . . . 549
9.5.1 Detailed Description . . . 549
9.6 apbs_link.c . . . 551
9.7 src/fem/apbs/vcsm.h File Reference . . . 551

9.7.1 Detailed Description . . . 553
9.8 vcsm.h . . . 554
9.9 src/fem/apbs/vfetk.h File Reference . . . 556
9.9.1 Detailed Description . . . 560
9.10 vfetk.h . . . 561
9.11 src/fem/apbs/vpee.h File Reference . . . 566
9.11.1 Detailed Description . . . 568
9.12 vpee.h . . . 569
9.13 src/fem/dummy.c File Reference . . . 570
9.13.1 Detailed Description . . . 570
9.14 dummy.c . . . 571
9.15 src/fem/vcsm.c File Reference . . . 572
9.15.1 Detailed Description . . . 573
9.16 vcsm.c . . . 574
9.17 src/fem/vfetk.c File Reference . . . 580
9.17.1 Detailed Description . . . 584
9.17.2 Variable Documentation . . . 585
9.17.2.1 diriCubeString . . . 585
9.17.2.2 lgr_2DP1 . . . 585
9.17.2.3 lgr_2DP1x . . . 586
9.17.2.4 lgr_2DP1y . . . 586
9.17.2.5 lgr_2DP1z . . . 586
9.17.2.6 lgr_3DP1 . . . 586
9.17.2.7 lgr_3DP1x . . . 587
9.17.2.8 lgr_3DP1y . . . 587
9.17.2.9 lgr_3DP1z . . . 587
9.17.2.10 neumCubeString . . . 587
9.18 vfetk.c . . . 588
9.19 src/fem/vpee.c File Reference . . . 619
9.19.1 Detailed Description . . . 620
9.20 vpee.c . . . 621
9.21 src/generic/apbs/femparm.h File Reference . . . 628
9.21.1 Detailed Description . . . 630
9.22 femparm.h . . . 632
9.23 src/generic/apbs/mgparm.h File Reference . . . 633
9.23.1 Detailed Description . . . 636
9.24 mgparm.h . . . 637

9.25 src/generic/apbs/nosh.h File Reference . . . 638
9.25.1 Detailed Description . . . 642
9.26 nosh.h . . . 643
9.27 src/generic/apbs/pbeparm.h File Reference . . . 646
9.27.1 Detailed Description . . . 647
9.28 pbeparm.h . . . 648
9.29 src/generic/apbs/vacc.h File Reference . . . 650
9.29.1 Detailed Description . . . 653
9.30 vacc.h . . . 655
9.31 src/generic/apbs/valist.h File Reference . . . 658
9.31.1 Detailed Description . . . 660
9.32 valist.h . . . 661
9.33 src/generic/apbs/vatom.h File Reference . . . 663
9.33.1 Detailed Description . . . 665
9.34 vatom.h . . . 666
9.35 src/generic/apbs/vcap.h File Reference . . . 668
9.35.1 Detailed Description . . . 669
9.36 vcap.h . . . 670
9.37 src/generic/apbs/vclist.h File Reference . . . 670
9.37.1 Detailed Description . . . 672
9.38 vclist.h . . . 673
9.39 src/generic/apbs/vgreen.h File Reference . . . 675
9.39.1 Detailed Description . . . 677
9.40 vgreen.h . . . 677
9.41 src/generic/apbs/vhal.h File Reference . . . 678
9.41.1 Detailed Description . . . 682
9.41.2 Macro Definition Documentation . . . 683
9.41.2.1 PRINT_DBL . . . 683
9.41.2.2 PRINT_FUNC . . . 683
9.41.2.3 PRINT_INT . . . 683
9.41.2.4 VCOPY . . . 683
9.41.2.5 VFILL . . . 683
9.41.2.6 VMESSAGE0 . . . 684
9.41.2.7 VMESSAGE1 . . . 684
9.41.2.8 VMESSAGE2 . . . 684
9.42 vhal.h . . . 684
9.43 src/generic/apbs/vmatrix.h File Reference . . . 690

9.43.1 Detailed Description . . . 691
9.43.2 Macro Definition Documentation . . . 692
9.43.2.1 MAT2 . . . 692
9.43.2.2 MAT3 . . . 692
9.44 vmatrix.h . . . 692
9.45 src/generic/apbs/vparam.h File Reference . . . 693
9.45.1 Detailed Description . . . 695
9.46 vparam.h . . . 696
9.47 src/generic/apbs/vpbe.h File Reference . . . 698
9.47.1 Detailed Description . . . 700
9.48 vpbe.h . . . 701
9.49 src/generic/apbs/vstring.h File Reference . . . 704
9.49.1 Detailed Description . . . 705
9.50 vstring.h . . . 706
9.51 src/generic/apbs/vunit.h File Reference . . . 706
9.51.1 Detailed Description . . . 708
9.52 vunit.h . . . 709
9.53 src/generic/apolparm.c File Reference . . . 709
9.53.1 Detailed Description . . . 711
9.54 apolparm.c . . . 712
9.55 src/generic/femparm.c File Reference . . . 719
9.55.1 Detailed Description . . . 721
9.56 femparm.c . . . 722
9.57 src/generic/mgparm.c File Reference . . . 727
9.57.1 Detailed Description . . . 729
9.58 mgparm.c . . . 730
9.59 src/generic/nosh.c File Reference . . . 741
9.59.1 Detailed Description . . . 744
9.60 nosh.c . . . 745
9.61 src/generic/pbeparm.c File Reference . . . 774
9.61.1 Detailed Description . . . 776
9.62 pbeparm.c . . . 777
9.63 src/generic/vacc.c File Reference . . . 793
9.63.1 Detailed Description . . . 795
9.63.2 Function Documentation . . . 796
9.63.2.1 ivdwAccExclus . . . 796
9.63.2.2 splineAcc . . . 797

9.63.2.3 Vacc_allocate . . . 798
9.63.2.4 Vacc_storeParms . . . 799
9.64 vacc.c . . . 800
9.65 src/generic/valist.c File Reference . . . 824
9.65.1 Detailed Description . . . 825
9.66 valist.c . . . 826
9.67 src/generic/vatom.c File Reference . . . 838
9.67.1 Detailed Description . . . 839
9.68 vatom.c . . . 840
9.69 src/generic/vcap.c File Reference . . . 843
9.69.1 Detailed Description . . . 844
9.70 vcap.c . . . 845
9.71 src/generic/vclist.c File Reference . . . 846
9.71.1 Detailed Description . . . 848
9.72 vclist.c . . . 849
9.73 src/generic/vgreen.c File Reference . . . 855
9.73.1 Detailed Description . . . 856
9.74 vgreen.c . . . 857
9.75 src/generic/vparam.c File Reference . . . 864
9.75.1 Detailed Description . . . 866
9.76 vparam.c . . . 867
9.77 src/generic/vpbe.c File Reference . . . 876
9.77.1 Detailed Description . . . 879
9.78 vpbe.c . . . 880
9.79 src/generic/vstring.c File Reference . . . 886
9.79.1 Detailed Description . . . 887
9.80 vstring.c . . . 888
9.81 src/mg/apbs/vgrid.h File Reference . . . 890
9.81.1 Detailed Description . . . 893
9.81.2 Function Documentation . . . 894
9.81.2.1 Vgrid_writeGZ . . . 894
9.82 vgrid.h . . . 894
9.83 src/mg/apbs/vmgrid.h File Reference . . . 895
9.83.1 Detailed Description . . . 897
9.84 vmgrid.h . . . 898
9.85 src/mg/apbs/vmultigrid.h File Reference . . . 899
9.85.1 Detailed Description . . . 901

9.85.2 Enumeration Type Documentation . . . 902
9.85.2.1 eVmultigrid_boundary_method . . . 902
9.85.2.2 eVmultigrid_coarse_solver . . . 903
9.85.2.3 eVmultigrid_coarsen_method . . . 903
9.85.2.4 eVmultigrid_discrete_method . . . 903
9.85.2.5 eVmultigrid_iterate_method . . . 903
9.85.2.6 eVmultigrid_linear . . . 904
9.85.2.7 eVmultigrid_method . . . 904
9.85.2.8 eVmultigrid_operator_analysis . . . 904
9.85.2.9 eVmultigrid_prolong_method . . . 905
9.85.2.10 eVmultigrid_smooth_method . . . 905
9.85.2.11 eVmultigrid_stop_criterion . . . 905
9.85.2.12 eVmultigrid_verbosity . . . 905
9.86 vmultigrid.h . . . 906
9.87 src/mg/apbs/vopot.h File Reference . . . 909
9.87.1 Detailed Description . . . 910
9.88 vopot.h . . . 911
9.89 src/mg/apbs/vpmg.h File Reference . . . 912
9.89.1 Detailed Description . . . 918
9.89.2 Function Documentation . . . 919
9.89.2.1 bcCalc . . . 919
9.89.2.2 bcfl1 . . . 920
9.89.2.3 bspline2 . . . 920
9.89.2.4 bspline4 . . . 921
9.89.2.5 d2bspline4 . . . 921
9.89.2.6 d3bspline4 . . . 922
9.89.2.7 dbspline2 . . . 922
9.89.2.8 dbspline4 . . . 923
9.89.2.9 fillcoCharge . . . 924
9.89.2.10 fillcoChargeMap . . . 925
9.89.2.11 fillcoChargeSpline1 . . . 926
9.89.2.12 fillcoChargeSpline2 . . . 926
9.89.2.13 fillcoCoef . . . 927
9.89.2.14 fillcoCoefMap . . . 928
9.89.2.15 fillcoCoefMol . . . 929
9.89.2.16 fillcoCoefMolDiel . . . 930
9.89.2.17 fillcoCoefMolDielNoSmooth . . . 931

9.89.2.18 fillcoCoefMolDielSmooth . . . 932
9.89.2.19 fillcoCoefMolIon . . . 933
9.89.2.20 fillcoCoefSpline . . . 934
9.89.2.21 fillcoCoefSpline3 . . . 936
9.89.2.22 fillcoCoefSpline4 . . . 937
9.89.2.23 fillcoInducedDipole . . . 938
9.89.2.24 fillcoNLInducedDipole . . . 938
9.89.2.25 fillcoPermanentMultipole . . . 938
9.89.2.26 markSphere . . . 940
9.89.2.27 multipolebc . . . 940
9.89.2.28 qfForceSpline1 . . . 941
9.89.2.29 qfForceSpline2 . . . 941
9.89.2.30 qfForceSpline4 . . . 942
9.89.2.31 VFCHI4 . . . 943
9.89.2.32 Vpmg_polarizEnergy . . . 944
9.89.2.33 Vpmg_qfEnergyPoint . . . 945
9.89.2.34 Vpmg_qfEnergyVolume . . . 946
9.89.2.35 Vpmg_qmEnergySMPBE . . . 947
9.89.2.36 Vpmg_splineSelect . . . 948
9.89.2.37 zlapSolve . . . 949
9.90 vpmg.h . . . 949
9.91 src/mg/apbs/vpmgp.h File Reference . . . 958
9.91.1 Detailed Description . . . 960
9.92 vpmgp.h . . . 961
9.93 src/mg/vgrid.c File Reference . . . 963
9.93.1 Detailed Description . . . 965
9.93.2 Function Documentation . . . 966
9.93.2.1 Vgrid_writeGZ . . . 966
9.94 vgrid.c . . . 966
9.95 src/mg/vopot.c File Reference . . . 985
9.95.1 Detailed Description . . . 985
9.96 vopot.c . . . 987
9.97 src/mg/vpmg.c File Reference . . . 991
9.97.1 Detailed Description . . . 994
9.97.2 Function Documentation . . . 996
9.97.2.1 bcCalc . . . 996
9.97.2.2 bcfl1 . . . 997

9.97.2.3 bspline2 . . . 997
9.97.2.4 bspline4 . . . 998
9.97.2.5 d2bspline4 . . . 998
9.97.2.6 d3bspline4 . . . 999
9.97.2.7 dbspline2 . . . 999
9.97.2.8 dbspline4 . . . 1000
9.97.2.9 fillcoCharge . . . 1001
9.97.2.10 fillcoChargeMap . . . 1002
9.97.2.11 fillcoChargeSpline1 . . . 1003
9.97.2.12 fillcoChargeSpline2 . . . 1003
9.97.2.13 fillcoCoef . . . 1004
9.97.2.14 fillcoCoefMap . . . 1005
9.97.2.15 fillcoCoefMol . . . 1006
9.97.2.16 fillcoCoefMolDiel . . . 1007
9.97.2.17 fillcoCoefMolDielNoSmooth . . . 1008
9.97.2.18 fillcoCoefMolDielSmooth . . . 1009
9.97.2.19 fillcoCoefMolIon . . . 1010
9.97.2.20 fillcoCoefSpline . . . 1011
9.97.2.21 fillcoCoefSpline3 . . . 1013
9.97.2.22 fillcoCoefSpline4 . . . 1014
9.97.2.23 fillcoPermanentMultipole . . . 1015
9.97.2.24 markSphere . . . 1016
9.97.2.25 multipolebc . . . 1016
9.97.2.26 qfForceSpline1 . . . 1017
9.97.2.27 qfForceSpline2 . . . 1018
9.97.2.28 qfForceSpline4 . . . 1019
9.97.2.29 VFCHI4 . . . 1020
9.97.2.30 Vpmg_polarizEnergy . . . 1020
9.97.2.31 Vpmg_qfEnergyPoint . . . 1021
9.97.2.32 Vpmg_qfEnergyVolume . . . 1022
9.97.2.33 Vpmg_qmEnergySMPBE . . . 1023
9.97.2.34 Vpmg_splineSelect . . . 1024
9.97.2.35 zlapSolve . . . 1025
9.98 vpmg.c . . . 1026
9.99 src/mg/vpmgp.c File Reference . . . 1167
9.99.1 Detailed Description . . . 1168
9.100 vpmgp.c . . . 1170

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 instatiated objects.

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

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

would be declared as

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

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

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

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

would be declared as

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

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

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

in C++ would be

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

in Clean OO C.

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

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

1.4 Application programming interface documentation

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

- `Class overview`
- `Class declarations`
- `Class members`
- `Class methods`

Chapter 2

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 dV[][VAPBS_DIM])

This function is not thread-safe

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

This function is not thread-safe

This function is not implemented (sets error to zero)

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

This function is not thread-safe

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

This function is not thread-safe

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

This function is not thread-safe

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

This function is not thread-safe

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

Global Vfetk_PDE_Ju (PDE *thee, int key)

This function is not thread-safe.

Global Vfetk_PDE_markSimplex (int dim, int dimII, int simplexType, int faceType[VAPBS_NVS], int vertexType[VAPBS_NVS], int 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-Hü functions.

This function is not thread-safe.

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

This function is not thread-safe.

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

Some values of format are not implemented

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

Not implemented yet

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

Not implemented yet

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

This routine does not respect partition information

Global 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)

Can this path variable be replaced with a Vio socket?

Global Vpbe_ctor2 (Vpbe *thee, Valist *alist, int ionNum, double *ionConc, double *ionRadii, double *ionQ, double T, double soluteDiel, double solventDiel, double solventRadius, int focusFlag, double sdens, double z_mem, double L, double membraneDiel, double V)

The focusing flag is currently not used!!

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 . . . 19
Vfetk class . . . 31
Vpee class . . . 72
APOLparm class . . . 77
FEMparm class . . . 83
MGparm class . . . 90
NOsh class . . . 103
PBEparm class . . . 123
Vacc class . . . 131
Valist class . . . 163
Vatom class . . . 175
Vcap class . . . 191
Vclist class . . . 194
Vgreen class . . . 203
Vhal class . . . 215
Matrix wrapper class . . . 228
Vparam class . . . 229
Vpbe class . . . 246
Vstring class . . . 268
Vunit class . . . 271
Vgrid class . . . 272
Vmgrid class . . . 285
Vopot class . . . 291
Vpmg class . . . 295
Vpmgp class . . . 332
C translation of Holst group PMG code . . . 336

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 . . . 443
sFEMparm
Parameter structure for FEM-specific variables from input files . . . 449
sMGparm
Parameter structure for MG-specific variables from input files . . . 453
sNOsh
Class for parsing fixed format input files . . . 462
sNOsh_calc
Calculation class for use when parsing fixed format input files . . . 469
sPBEparm
Parameter structure for PBE variables from input files . . . 470
sVacc
Oracle for solvent- and ion-accessibility around a biomolecule . . . 480
sVaccSurf
Surface object list of per-atom surface points . . . 483
sValist
Container class for list of atom objects . . . 484
sVatom
Contains public data members for Vatom class/module . . . 486
sVclist
Atom cell list . . . 488
sVclistCell
Atom cell list cell . . . 490
sVcsm
Charge-simplex map class . . . 491
sVfetk
Contains public data members for Vfetk class/module . . . 493
sVfetk_LocalVar
Vfetk LocalVar subclass . . . 497
sVgreen
Contains public data members for Vgreen class/module . . . 503
sVgrid
Electrostatic potential oracle for Cartesian mesh data . . . 505

sVmgrid
Multiresoltion oracle for Cartesian mesh data 508
sVopot
Electrostatic potential oracle for Cartesian mesh data 513
sVparam_AtomData
AtomData sub-class; stores atom data 515
sVpbe
Contains public data members for Vpbe class/module 517
sVpee
Contains public data members for Vpee class/module 522
sVpmg
Contains public data members for Vpmg class/module 524
sVpmgp
Contains public data members for Vpmgp class/module 531
Vparam
Reads and assigns charge/radii parameters 541
Vparam_ResData
ResData sub-class; stores residue data 542

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 546
doc/programmer/**mainpage.h** . . . . . **??**
src/aaa_inc/apbs/apbs.h
Top-level header for APBS 548
src/aaa_lib/apbs_link.c
Autoconf linkage assistance for packages built on top of APBS 551
src/fem/dummy.c
Give libtool something to do 571
src/fem/vcsm.c
Class Vcsm methods 574
src/fem/vfetk.c
Class Vfetk methods 588
src/fem/vpee.c
Class Vpee methods 621
src/fem/apbs/vcsm.h
Contains declarations for the Vcsm class 554
src/fem/apbs/vfetk.h
Contains declarations for class Vfetk 561
src/fem/apbs/vpee.h
Contains declarations for class Vpee 569
src/generic/apolparm.c
Class APOLparm methods 712
src/generic/femparm.c
Class FEMparm methods 722
src/generic/mgparm.c
Class MGparm methods 730
src/generic/nosh.c
Class NOsh methods 745
src/generic/pbeparm.c
Class PBEparm methods 777
src/generic/vacc.c
Class Vacc methods 800

src/generic/valist.c
Class Valist methods . . . 826
src/generic/vatom.c
Class Vatom methods . . . 840
src/generic/vcap.c
Class Vcap methods . . . 845
src/generic/vclist.c
Class Vclist methods . . . 849
src/generic/vgreen.c
Class Vgreen methods . . . 857
src/generic/vparam.c
Class Vparam methods . . . 867
src/generic/vpbe.c
Class Vpbe methods . . . 880
src/generic/vstring.c
Class Vstring methods . . . 888
src/generic/apbs/**apolparm.h** . . . **??**
src/generic/apbs/femparm.h
Contains declarations for class APOLparm . . . 632
src/generic/apbs/mgparm.h
Contains declarations for class MGparm . . . 637
src/generic/apbs/nosh.h
Contains declarations for class NOsh . . . 643
src/generic/apbs/pbeparm.h
Contains declarations for class PBEparm . . . 648
src/generic/apbs/vacc.h
Contains declarations for class Vacc . . . 655
src/generic/apbs/valist.h
Contains declarations for class Valist . . . 661
src/generic/apbs/vatom.h
Contains declarations for class Vatom . . . 666
src/generic/apbs/vcap.h
Contains declarations for class Vcap . . . 670
src/generic/apbs/vclist.h
Contains declarations for class Vclist . . . 673
src/generic/apbs/vgreen.h
Contains declarations for class Vgreen . . . 677
src/generic/apbs/vhal.h
Contains generic macro definitions for APBS . . . 684
src/generic/apbs/vmatrix.h
Contains inclusions for matrix data wrappers . . . 692
src/generic/apbs/vparam.h
Contains declarations for class Vparam . . . 696
src/generic/apbs/vpbe.h
Contains declarations for class Vpbe . . . 701
src/generic/apbs/vstring.h
Contains declarations for class Vstring . . . 706
src/generic/apbs/vunit.h
Contains a collection of useful constants and conversion factors . . . 709
src/mg/vgrid.c
Class Vgrid methods . . . 966
src/mg/vopot.c
Class Vopot methods . . . 987

src/mg/vpmg.c
Class Vpmg methods . . . 1026
src/mg/vpmgp.c
Class Vpmgp methods . . . 1170
src/mg/apbs/vgrid.h
Potential oracle for Cartesian mesh data . . . 894
src/mg/apbs/vmgrid.h
Multiresolution oracle for Cartesian mesh data . . . 898
src/mg/apbs/vopot.h
Potential oracle for Cartesian mesh data . . . 911
src/mg/apbs/vpmg.h
Contains declarations for class Vpmg . . . 949
src/mg/apbs/vpmgp.h
Contains declarations for class Vpmgp . . . 961
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/**mgdrvd.c** . . . **??**
src/pmgc/**mgfasd.c** . . . **??**
src/pmgc/**mgsubd.c** . . . **??**
src/pmgc/**mikpckd.c** . . . **??**
src/pmgc/**mlinpckd.c** . . . **??**
src/pmgc/**mypdec.c** . . . **??**
src/pmgc/**newdrvd.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/**mgdrvd.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/**newdrvd.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)

Get ID of particular atom in a simplex.

- VEXTERNC int Vcsm_getNumberSimplices (Vcsm ∗thee, int iatom)

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

  Get particular simplex associated with an atom.
- VEXTERNC int Vcsm_getSimplexIndex (Vcsm ∗thee, int isimp, int iatom)

  Get index particular simplex associated with an atom.
- VEXTERNC unsigned long int Vcsm_memChk (Vcsm ∗thee)

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

  Construct Vcsm object.
- VEXTERNC int Vcsm_ctor2 (Vcsm ∗thee, Valist ∗alist, Gem ∗gm)

  FORTRAN stub to construct Vcsm object.
- VEXTERNC void Vcsm_dtor (Vcsm ∗∗thee)

  Destroy Vcsm object.
- VEXTERNC void Vcsm_dtor2 (Vcsm ∗thee)

  FORTRAN stub to destroy Vcsm object.
- VEXTERNC void Vcsm_init (Vcsm ∗thee)

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

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

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

thee	The FEtk geometry manager
externalUpdate	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

alist	List of atoms
gm	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

thee	The Vcsm object
alist	The list of atoms
gm	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

thee	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

thee	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

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

Definition at line 81 of file vcsm.c.

Here is the call graph for this function:

Here is the caller graph for this function:

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

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

thee	The Vcsm object
isimp	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

thee	The Vcsm object
iatom	The Valist atom index

Definition at line 103 of file vcsm.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

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

Definition at line 113 of file vcsm.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

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

thee	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

thee	The Vcsm object

Definition at line 174 of file vcsm.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

thee	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

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

Definition at line 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_BT/e) at the finest mesh level.
- VEXTERNC void Vfetk_setParameters (Vfetk ∗thee, PBEparm ∗pbeparm, FEMparm ∗feparm)

  Set the parameter objects.
- VEXTERNC double Vfetk_energy (Vfetk ∗thee, int color, int nonlin)

  Return the total electrostatic energy.
- VEXTERNC double Vfetk_dqmEnergy (Vfetk ∗thee, int color)

  Get the "mobile charge" and "polarization" contributions to the electrostatic energy.
- VEXTERNC double Vfetk_qfEnergy (Vfetk ∗thee, int color)

  Get the "fixed charge" contribution to the electrostatic energy.
- VEXTERNC unsigned long int Vfetk_memChk (Vfetk ∗thee)

  Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC void Vfetk_setAtomColors (Vfetk ∗thee)

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

- VEXTERNC void Bmat_printHB (Bmat ∗thee, char ∗fname)

  Writes a Bmat to disk in Harwell-Boeing sparse matrix format.
- VEXTERNC Vrc_Codes Vfetk_genCube (Vfetk ∗thee, double center[3], double length[3], Vfetk_MeshLoad meshType)

  Construct a rectangular mesh (in the current Vfetk object)
- VEXTERNC Vrc_Codes Vfetk_loadMesh (Vfetk ∗thee, double center[3], double length[3], Vfetk_MeshLoad meshType, Vio ∗sock)

  Loads a mesh into the Vfetk (and associated) object(s).
- VEXTERNC PDE ∗ Vfetk_PDE_ctor (Vfetk ∗fetk)

  Constructs the FEtk PDE object.
- VEXTERNC int Vfetk_PDE_ctor2 (PDE ∗thee, Vfetk ∗fetk)

  Intializes the FEtk PDE object.
- VEXTERNC void Vfetk_PDE_dtor (PDE ∗∗thee)

  Destroys FEtk PDE object.
- VEXTERNC void Vfetk_PDE_dtor2 (PDE ∗thee)

  FORTRAN stub: destroys FEtk PDE object.
- VEXTERNC void Vfetk_PDE_initAssemble (PDE ∗thee, int ip[], double rp[])

  Do once-per-assembly initialization.
- VEXTERNC void Vfetk_PDE_initElement (PDE ∗thee, int elementType, int chart, double tvx[][VAPBS_DIM], void ∗data)

  Do once-per-element initialization.
- VEXTERNC void Vfetk_PDE_initFace (PDE ∗thee, int faceType, int chart, double tnvec[])

  Do once-per-face initialization.
- VEXTERNC void Vfetk_PDE_initPoint (PDE ∗thee, int pointType, int chart, double txq[], double tU[], double tdU[][VAPBS_DIM])

  Do once-per-point initialization.
- VEXTERNC void Vfetk_PDE_Fu (PDE ∗thee, int key, double F[])

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

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

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

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

  where $n(x)$ is the normal to the simplex face and the term represents the jump in dielectric displacement across the face. There is no outer-boundary contribution for this problem.
- VEXTERNC double Vfetk_PDE_Fu_v (PDE ∗thee, int key, double V[], double dV[][VAPBS_DIM])

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

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

  where $b(u)$ denotes the mobile ion term.
- VEXTERNC double Vfetk_PDE_DFu_wv (PDE ∗thee, int key, double W[], double dW[][VAPBS_DIM], double V[], double dV[][VAPBS_DIM])

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

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

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

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

  Evaluate a (discretized) delta function source term at the given point.
- VEXTERNC void Vfetk_PDE_u_D (PDE *thee, int type, int chart, double txq[], double F[])

  Evaluate the Dirichlet boundary condition at the given point.
- VEXTERNC void Vfetk_PDE_u_T (PDE *thee, int type, int chart, double txq[], double F[])

  Evaluate the "true solution" at the given point for comparison with the numerical solution.
- VEXTERNC void Vfetk_PDE_bisectEdge (int dim, int dimII, int edgeType, int chart[], double vx[][VAPBS_DIM])

  Define the way manifold edges are bisected.
- VEXTERNC void Vfetk_PDE_mapBoundary (int dim, int dimII, int vertexType, int chart, double vx[VAPBS_DIM])

  Map a boundary point to some pre-defined shape.
- VEXTERNC int Vfetk_PDE_markSimplex (int dim, int dimII, int simplexType, int faceType[VAPBS_NVS], int vertexType[VAPBS_NVS], int chart[], double vx[][VAPBS_DIM], void *simplex)

  User-defined error estimator – in our case, a geometry-based refinement method; forcing simplex refinement at the dielectric boundary and (for non-regularized PBE) the charges.
- VEXTERNC void Vfetk_PDE_oneChart (int dim, int dimII, int objType, int chart[], double vx[][VAPBS_DIM], int dimV)

  Unify the chart for different coordinate systems – a no-op for us.
- VEXTERNC double Vfetk_PDE_Ju (PDE *thee, int key)

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

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

  for a 1:1 electrolyte where c is the output from Vpbe_getZmagic.
- VEXTERNC void Vfetk_externalUpdateFunction (SS **simps, int num)

  External hook to simplex subdivision routines in Gem. Called each time a simplex is subdivided (we use it to update the charge-simplex map)
- VEXTERNC int Vfetk_PDE_simplexBasisInit (int key, int dim, int comp, int *ndof, int dof[])

  Initialize the bases for the trial or the test space, for a particular component of the system, at all quadrature points on the master simplex element.
- VEXTERNC void Vfetk_PDE_simplexBasisForm (int key, int dim, int comp, int pdkey, double xq[], double basis[])

  Evaluate the bases for the trial or test space, for a particular component of the system, at all quadrature points on the master simplex element.
- VEXTERNC void Vfetk_readMesh (Vfetk *thee, int skey, Vio *sock)

  Read in mesh and initialize associated internal structures.
- VEXTERNC void Vfetk_dumpLocalVar ()

  Debugging routine to print out local variables used by PDE object.
- VEXTERNC int Vfetk_fillArray (Vfetk *thee, Bvec *vec, Vdata_Type type)

  Fill an array with the specified data.
- VEXTERNC int Vfetk_write (Vfetk *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, Bvec *vec, Vdata_Format format)

  Write out data.
- VEXTERNC Vrc_Codes Vfetk_loadGem (Vfetk *thee, Gem *gm)

  Load a Gem geometry manager object into Vfetk.

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

thee	The matrix to write
fname	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

pbe	Vpbe (PBE manager object)
type	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

thee	Vfetk object memory
pbe	PBE manager object
type	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:

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

for the NPBE and

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

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

Author

Nathan Baker

Parameters

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

Returns

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

Parameters

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

Definition at line 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

thee	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

thee	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

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

Definition at line 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

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

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

thee	Vfetk object
center	Center for mesh, which the new mesh will adjust to
length	Mesh lengths, which the new mesh will adjust to
meshType	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

thee	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

thee	The Vfetk object
iatom	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

thee	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

thee	Vfetk object with solution
length	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

thee	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

thee	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

thee	Destination
gm	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

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

Definition at line 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

thee	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 *dimII*, int *edgeType*, int *chart[ ]*, double *vx[ ][VAPBS_DIM]*)

Define the way manifold edges are bisected.

Author

Nathan Baker and Mike Holst

Note

This function is thread-safe.

Parameters

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

Here is the caller graph for this function:

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

Constructs the FEtk PDE object.

Author

Nathan Baker

Returns

Newly-allocated PDE object

Bug Not thread-safe

Parameters

fetk	The Vfetk object

Definition at line 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*)

Intializes the FEtk PDE object.

Author

Nathan Baker (with code by Mike Holst)

Returns

1 if successful, 0 otherwise

Bug Not thread-safe

Parameters

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

Definition at line 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

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

Definition at line 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:

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

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

Author

Nathan Baker and Mike Holst

Returns

Integrand value

Bug This function is not thread-safe

Parameters

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

Here is the caller graph for this function:

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

Destroys FEtk PDE object.

Author

Nathan Baker

Note

Thread-safe

Parameters

thee	Pointer to PDE object memory

Definition at line 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

thee	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:

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

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

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

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

Author

Nathan Baker

Bug This function is not thread-safe

This function is not implemented (sets error to zero)

Parameters

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

Definition at line 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

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

Definition at line 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

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

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

Here is the caller graph for this function:

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

Do once-per-face initialization.

Author

Nathan Baker and Mike Holst

Bug This function is not thread-safe

Parameters

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

Definition at line 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

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

Here is the caller graph for this function:

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:

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

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

Author

Nathan Baker

Returns

Energy value (in kT)

Bug This function is not thread-safe.

Parameters

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

Definition at line 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 *dimII*, int *vertexType*, int *chart*, double *vx[VAPBS_DIM]*)

Map a boundary point to some pre-defined shape.

Author

Nathan Baker and Mike Holst

Note

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

Parameters

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

Here is the caller graph for this function:

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

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

Author

Nathan Baker

Returns

1 if mark simplex for refinement, 0 otherwise

Bug This function is not thread-safe

Parameters

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

Here is the caller graph for this function:

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

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

Author

Nathan Baker

Note

Thread-safe; a no-op

Parameters

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

Here is the caller graph for this function:

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

key	Basis type to evaluate (0 = trial, 1 = test, 2 = trialB, 3 = testB)
dim	Spatial dimension
comp	Which component of elliptic system to produce basis for
pdkey	Basis partial differential equation evaluation key: • 0 = evaluate basis(x,y,z) • 1 = evaluate basis_x(x,y,z) • 2 = evaluate basis_y(x,y,z) • 3 = evaluate basis_z(x,y,z) • 4 = evaluate basis_xx(x,y,z) • 5 = evaluate basis_yy(x,y,z) • 6 = evaluate basis_zz(x,y,z) • 7 = etc...
xq	Set to quad pt coordinate
basis	Set to all basis functions evaluated at all quadrature pts

Definition at line 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, eletromagnetics, or simply to play around with
*   different elements.
*
*   This function is called by MC to build new master elements whenever it
*   reads in a new mesh.  Therefore, this function does not have to be all
*   that fast, and e.g.  could involve symbolic computation.
*
```

Parameters

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

Definition at line 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

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

thee	PDE object
type	Point type
chart	Chart for point coordinates
txq	Point coordinates
F	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

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

Returns

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

Parameters

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

Definition at line 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

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

thee	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

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

thee	Vfetk object
vec	FEtk Bvec vector to use
format	Format for data

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

Note

This function is thread-safe

Bug Some values of format are not implemented

Returns

1 if successful, 0 otherwise

Parameters

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

Definition at line 2469 of file vfetk.c.

7.3 Vpee class

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

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

gm	FEtk geometry manager object
localPartID	ID of the local partition (focus of refinement)
killFlag	A flag to indicate how error estimates are to be attenuated outside the local partition: • 0: no attenuation • 1: all error outside the local partition set to zero • 2: all error is set to zero outside a sphere of radius (killParam*partRadius), where partRadius is the radius of the sphere circumscribing the local partition • 3: all error is set to zero except for the local partition and its immediate neighbors
killParam	

See Also

killFlag for usage

Definition at line 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

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

killFlag	A flag to indicate how error estimates are to be attenuated outside the local partition: • 0: no attenuation • 1: all error outside the local partition set to zero • 2: all error is set to zero outside a sphere of radius (killParam∗partRadius), where partRadius is the radius of the sphere circumscribing the local partition • 3: all error is set to zero except for the local partition and its immediate neighbors
killParam	

See Also

killFlag for usage

Definition at line 121 of file vpee.c.

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

Object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of the Vpee object

Definition at line 232 of file vpee.c.

7.3.2.4 VEXTERNC void Vpee_dtor2 (Vpee ∗ *thee*)

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to object to be destroyed

Definition at line 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

thee	The Vpee object
am	The FEtk algebra manager currently used to solve the PB
level	The current level of the multigrid hierarchy
akey	The marking method: • -1: Reset markings –> killFlag has no effect. • 0: Uniform. • 1: User defined (geometry-based). • >1: A numerical estimate for the error has already been set in am and should be attenuated according to killFlag and used, in conjunction with etol, to mark simplices for refinement.
rcol	The ID of the main parition on which to mark (or -1 if all partitions should be marked). NOte that we shouldhave (rcol == thee->localPartID) for (thee->killFlag == 2 or 3)
etol	The error tolerance criterion for marking
bkey	How the error tolerance is interpreted: • 0: Simplex marked if error > etol. • 1: Simplex marked if error > sqrt(etol^2/L) where L$ is the number of simplices

Definition at line 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

thee	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

thee	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

thee	Destination object
source	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

thee	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

thee	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

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

  Adaptive refinment error estimator method.
- enum eFEMparm_CalcType { FCT_MANUAL, FCT_NONE }

  Calculation type.

Functions

- VEXTERNC FEMparm ∗ FEMparm_ctor (FEMparm_CalcType type)

  Construct FEMparm.
- VEXTERNC int FEMparm_ctor2 (FEMparm ∗thee, FEMparm_CalcType type)

  FORTRAN stub to construct FEMparm.
- VEXTERNC void FEMparm_dtor (FEMparm ∗∗thee)

  Object destructor.
- VEXTERNC void FEMparm_dtor2 (FEMparm ∗thee)

  FORTRAN stub for object destructor.
- VEXTERNC int FEMparm_check (FEMparm ∗thee)

  Consistency check for parameter values stored in object.
- VEXTERNC void FEMparm_copy (FEMparm ∗thee, FEMparm ∗source)

  Copy target object into thee.

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 refinment error estimator method.

Note

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

Author

Nathan Baker

Enumerator:

FRT_UNIF Uniform refinement

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

FRT_RESI Nonlinear residual estimate-based refinement

FRT_DUAL Dual-solution weight nonlinear residual estimate-based refinement

FRT_LOCA Local problem error estimate-based refinement

Definition at line 95 of file femparm.h.

7.5.3.3 enum eFEMparm_EtolType

Adaptive refinment error estimate tolerance key.

Author

Nathan Baker

Enumerator:

FET_SIMP per-simplex error tolerance

FET_GLOB global error tolerance

FET_FRAC fraction of simplices we want to have refined

Definition at line 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

thee	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

thee	Destination object
source	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

type	FEM calculation type

Returns

Newly allocated and initialized Vpmgp object

Definition at line 67 of file femparm.c.

Here is the call graph for this function:

Here is the caller graph for this function:

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

FORTRAN stub to construct FEMparm.

Author

Nathan Baker

Parameters

thee	Pointer to allocated FEMparm object
type	FEM calculation type

Returns

1 if successful, 0 otherwise

Definition at line 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

thee	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

thee	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

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

Returns

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

Definition at line 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

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

Returns

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

Definition at line 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

thee	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

thee	MGparm object (target for copy)
parm	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

type	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

thee	Space for MGparm object
type	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

thee	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

thee	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

thee	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

thee	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

thee	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

thee	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

thee	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

thee	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

thee	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

thee	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

thee	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

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

Returns

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

Definition at line 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

thee	MGparm object
x	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

thee	MGparm object
y	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

thee	MGparm object
z	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

thee	NOsh object to use
icalc	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

thee	Target object
source	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

calcType	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

thee	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

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

Returns

Newly allocated and initialized NOsh object

Definition at line 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

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

Returns

1 if successful, 0 otherwise

Definition at line 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

thee	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

thee	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

thee	NOsh object to use
icalc	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

thee	NOsh object to use
ielec	ID of ELEC statement

Returns

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

See Also

elec2calc)

Definition at line 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

thee	Pointer to NOsh object
icalc	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

thee	Pointer to NOsh object
imap	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

thee	Pointer to NOsh object
imap	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

thee	Pointer to NOsh object
imap	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

thee	Pointer to NOsh object
imap	Map ID of interest

Returns

Path string

Definition at line 145 of file nosh.c.

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

Returns path to specified y-shifted dielectric map.

Author

Nathan Baker

Parameters

thee	Pointer to NOsh object
imap	Map ID of interest

Returns

Path string

Definition at line 150 of file nosh.c.

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

Returns path to specified z-shifted dielectric map.

Author

Nathan Baker

Parameters

thee	Pointer to NOsh object
imap	Map ID of interest

Returns

Path string

Definition at line 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

thee	Pointer to NOsh object
imap	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

thee	Pointer to NOsh object
imap	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

thee	Pointer to NOsh object
imol	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

thee	Pointer to NOsh object
imap	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

thee	Pointer to NOsh object
imap	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

thee	Pointer to NOsh object
sock	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

thee	Pointer to NOsh object
filename	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

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

Returns

Calculation ID for specified PRINT statement (

See Also

printcalc)

Definition at line 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

thee	NOsh object to use
iprint	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

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

Returns

Integer ID for specified operation (

See Also

printop)

Definition at line 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

thee	NOsh object to use
iprint	ID of PRINT statement

Returns

An integer ID of the observable to print (

See Also

printwhat)

Definition at line 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

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

Returns

1 if successful, 0 otherwise

Parameters

thee	NOsh object
alist	Atom list for calculation

Definition at line 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

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

Returns

1 if successful, 0 otherwise

Parameters

thee	NOsh object
alist	Atom list for calculation

Definition at line 1213 of file nosh.c.

7.8 PBEparm class

Parameter structure for PBE variables independent of solver.

Files

- file pbeparm.h

  Contains declarations for class PBEparm.
- file pbeparm.c

  Class PBEparm methods.

Data Structures

- struct sPBEparm

  Parameter structure for PBE variables from input files.

Macros

- #define PBEPARM_MAXWRITE 20

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

Typedefs

- typedef enum ePBEparm_calcEnergy PBEparm_calcEnergy

  Define ePBEparm_calcEnergy enumeration as PBEparm_calcEnergy.
- typedef enum ePBEparm_calcForce PBEparm_calcForce

  Define ePBEparm_calcForce enumeration as PBEparm_calcForce.
- typedef struct sPBEparm PBEparm

  Declaration of the PBEparm class as the PBEparm structure.

Enumerations

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

  Define energy calculation enumeration.
- enum ePBEparm_calcForce { PCF_NO = 0, PCF_TOTAL = 1, PCF_COMPS = 2 }

  Define force calculation enumeration.

Functions

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

  Get charge (e) of specified ion species.
- VEXTERNC double PBEparm_getIonConc (PBEparm *thee, int iion)

  Get concentration (M) of specified ion species.
- VEXTERNC double PBEparm_getIonRadius (PBEparm *thee, int iion)

  Get radius (A) of specified ion species.
- VEXTERNC PBEparm * PBEparm_ctor ()

Construct PBEparm object.

- VEXTERNC int PBEparm_ctor2 (PBEparm *thee)

  FORTRAN stub to construct PBEparm object.

- VEXTERNC void PBEparm_dtor (PBEparm **thee)

  Object destructor.

- VEXTERNC void PBEparm_dtor2 (PBEparm *thee)

  FORTRAN stub for object destructor.

- VEXTERNC int PBEparm_check (PBEparm *thee)

  Consistency check for parameter values stored in object.

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

  Copy PBEparm object into thee.

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

  Parse a keyword from an input file.

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 PBEparm_check (PBEparm * *thee*)

Consistency check for parameter values stored in object.

Author

Nathan Baker

Returns

1 if OK, 0 otherwise

Parameters

thee	Object to be checked

Definition at line 185 of file pbeparm.c.

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

Copy PBEparm object into thee.

Author

Nathan Baker

Parameters

thee	Target for copy
parm	Source for copy

Definition at line 285 of file pbeparm.c.

Here is the caller graph for this function:

7.8.3.3 VEXTERNC PBEparm* PBEparm_ctor ( )

Construct PBEparm object.

Author

Nathan Baker

Returns

Newly allocated and initialized PBEparm object

Definition at line 106 of file pbeparm.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.8.3.4 VEXTERNC int PBEparm_ctor2 (PBEparm * *thee*)

FORTRAN stub to construct PBEparm object.

Author

Nathan Baker

Returns

1 if succesful, 0 otherwise

Parameters

thee	Memory location for object

Definition at line 117 of file pbeparm.c.

Here is the caller graph for this function:

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

Object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of object

Definition at line 175 of file pbeparm.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.8.3.6 VEXTERNC void PBEparm_dtor2 (PBEparm * *thee*)

FORTRAN stub for object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to object to be destroyed

Definition at line 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

thee	PBEparm object
iion	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

thee	PBEparm object
iion	Ion species ID/index

Definition at line 73 of file pbeparm.c.

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

Get radius (A) of specified ion species.

Author

Nathan Baker

Returns

Radius of ion species (A)

Parameters

thee	PBEparm object
iion	Ion species ID/index

Definition at line 79 of file pbeparm.c.

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

Parse a keyword from an input file.

Author

Nathan Baker

Returns

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

Parameters

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

Definition at line 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 derivatve of solvent accessible volume.

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

  Get the derivatve of solvent accessible area.

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

  Testing purposes only.

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

  Total solvent accessible volume.
- VEXTERNC double Vacc_totalSAV (Vacc *thee, Vclist *clist, APOLparm *apolparm, double radius)

  Return the total solvent accessible volume (SAV)
- VEXTERNC int Vacc_wcaEnergy (Vacc *thee, APOLparm *apolparm, Valist *alist, Vclist *clist)

  Return the WCA integral energy.
- VEXTERNC int Vacc_wcaForceAtom (Vacc *thee, APOLparm *apolparm, Vclist *clist, Vatom *atom, double *force)

  Return the WCA integral force.
- VEXTERNC int Vacc_wcaEnergyAtom (Vacc *thee, APOLparm *apolparm, Valist *alist, Vclist *clist, int iatom, double *value)

  Calculate the WCA energy for an atom.

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 derivatve of solvent accessible area.

Author

Jason Wagoner, David Gohara, Nathan Baker

Parameters

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

Definition at line 1338 of file vacc.c.

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

Get the derivatve of solvent accessible volume.

Author

Jason Wagoner, Nathan Baker

Parameters

thee	Acessibility object
radius	Probe radius (Å)

atom	Atom of interest
dSA	Array holding answers of calc

Definition at line 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

thee	Accessibility object
radius	Probe molecule radius (Å)
atom	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

thee	Accessibility object
radius	Probe molecule radius (Å)
atom	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

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

See Also

VaccSurf_refSphere

Parameters

probe_radius	Probe radius (in A)

Definition at line 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

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

Definition at line 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

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

Definition at line 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

thee	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

thee	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

thee	Accessibility object
center	Probe center coordinates
radius	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

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

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

thee	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

thee	Accessibility object
center	Probe center coordinates
radius	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

thee	Accessibility object
radius	Probe molecule radius (Å)

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 *infrad*)

Report spline-based accessibility.

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

Author

Nathan Baker

Returns

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

Parameters

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

Definition at line 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 evalation; basically a cubic spline.

Author

Nathan Baker

Returns

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

Parameters

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

Definition at line 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

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

Definition at line 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

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

Definition at line 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

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

Definition at line 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

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

Definition at line 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 evalation; basically a cubic spline.

Author

Nathan Baker

Parameters

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

Definition at line 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

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

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

Definition at line 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

thee	Accessibility object
radius	Probe molecule radius (Å)

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 (A^3)

Parameters

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

Definition at line 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

thee	Accessibility object
center	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

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

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

Definition at line 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

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

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

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

Definition at line 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

thee	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

thee	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

mem	Memory object
npts	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

thee	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

thee	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

thee	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

thee	Atom list object
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

thee	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

thee	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

thee	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

thee	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

thee	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

thee	Atom list object

Definition at line 130 of file valist.c.

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

Fill atom list with information from a PDB file.

Author

Nathan Baker, Todd Dolinsky, Yong Huang

Returns

Success enumeration

Note

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

Parameters

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

Definition at line 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

thee	Atom list object
param	A pre-initialized parameter object
sock	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 VEXTERNC Vrc_Codes Valist_readXML (Valist * *thee*, Vparam * *param*, Vio * *sock*)

Fill atom list with information from an XML file.

Author

Todd Dolinsky, Yong Huang

Returns

Success enumeration

Note

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

Parameters

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

Definition at line 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

thee	Destination for atom information
src	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

thee	Source for atom information
dest	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

thee	Pointer to Vatom allocated memory location

Returns

1 if succesful, 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

thee	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

thee	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

thee	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

thee	Vatom object
atomName	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

thee	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

thee	Vatom object

Returns

Atomic epsilon (in Å)

7.11.3.11 VEXTERNC double Vatom_getPartID (Vatom * *thee*)

Get partition ID.

Author

Nathan Baker

Parameters

thee	Vatom object

Returns

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

Definition at line 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

thee	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

thee	Vatom object
resName	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

thee	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

thee	Vatom object
id	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

thee	Vatom object
atomName	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

thee	Vatom object
charge	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

thee	Vatom object
epsilon	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

thee	Vatom object
partID	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

thee	Vatom object to be modified
position	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

thee	Vatom object
radius	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

thee	Vatom object
resName	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

x	Argument to cosh()
ichop	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 Vcap_exp() exceeds EXPMAX or EXPMIN, then we
return exp(EXPMAX) or exp(EXPMIN) rather than exp(x).
```

Note

Original FORTRAN routine from PMG library by Mike Holst Original notes: to control overflow in the hyperbolic and exp functions, note that the following are the argument limits of the various functions on various machines after which overflow occurs: Convex C240, Sun 3/60, Sun SPARC, IBM RS/6000: sinh, cosh, exp: maximal argument (abs value) = 88.0d0 dsinh, dcosh, dexp: maximal argument (abs value) = 709.0d0

Author

Nathan Baker (based on FORTRAN code by Mike Holst)

Returns

exp(x) or capped equivalent

Parameters

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

Definition at line 60 of file vcap.c.

Here is the caller graph for this function:

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(EXPMAX) or sinh(EXPMIN) rather than sinh(x).
```

Note

Original FORTRAN routine from PMG library by Mike Holst Original notes: to control overflow in the hyperbolic and exp functions, note that the following are the argument limits of the various functions on various machines after which overflow occurs: Convex C240, Sun 3/60, Sun SPARC, IBM RS/6000: sinh, cosh, exp: maximal argument (abs value) = 88.0d0 dsinh, dcosh, dexp: maximal argument (abs value) = 709.0d0

Author

Nathan Baker (based on FORTRAN code by Mike Holst)

Returns

sinh(x) or capped equivalent

Parameters

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

Definition at line 76 of file vcap.c.

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

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

Definition at line 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

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

Definition at line 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

thee	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

thee	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

thee	Pointer to Vclist cell list
position	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

thee	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

thee	Memory location for object
natoms	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

thee	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

thee	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

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

Returns

1 if successful, 0 otherwise

Definition at line 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

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

Returns

1 if successful, 0 otherwise

Definition at line 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

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

Returns

1 if successful, 0 otherwise

Definition at line 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

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

Returns

1 if successful, 0 otherwise

Definition at line 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

alist	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

thee	Pointer to memory allocated for object
alist	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

thee	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

thee	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

thee	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 &Aring;)
```

q_i is the atomic charge (in e), and r_i r_i is the distance from atom i to the observation point r. The potential is scaled to units of V.

Author

Nathan Baker

Bug Not implemented yet

Note

Not implemented yet

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 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 &Aring;).
```

q_i is the atomic charge (in e), and r_i r_i is the distance from atom i to the observation point r. The potential is scaled to units of V/Å.

Author

Nathan Baker

Bug Not implemented yet

Note

Not implemented yet

Parameters

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

Returns

int 1 if sucessful, 0 otherwise

Definition at line 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

thee	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

Face definition for a volume.

- #define VAPBS_LEFT 3

  Face definition for a volume.

- #define VAPBS_BACK 4

  Face definition for a volume.

- #define VAPBS_DOWN 5

  Face definition for a volume.

- #define VPMGSMALL 1e-12

  A small number used in Vpmg to decide if points are on/off grid-lines or non-zer0 (etc.)

- #define SINH_MIN -85.0

  Used to set the min values acceptable for sinh chopping.

- #define SINH_MAX 85.0

  Used to set the max values acceptable for sinh chopping.

- #define VF77_MANGLE(name, NAME) name

  Name-mangling macro for using FORTRAN functions in C code.

- #define VFLOOR(value) floor(value)

  Wrapped floor to fix floating point issues in the Intel compiler.

- #define VEMBED(rctag)

  Allows embedding of RCS ID tags in object files.

Typedefs

- typedef enum eVhal_PBEType Vhal_PBEType

  Declaration of the Vhal_PBEType type as the Vhal_PBEType enum.

- typedef enum eVhal_IPKEYType Vhal_IPKEYType

  Declaration of the Vhal_IPKEYType type as the Vhal_IPKEYType enum.

- typedef enum eVhal_NONLINType Vhal_NONLINType

  Declaration of the Vhal_NONLINType type as the Vhal_NONLINType enum.

- typedef enum eVoutput_Format Voutput_Format

  Declaration of the Voutput_Format type as the VOutput_Format enum.

- typedef enum eVbcfl Vbcfl

  Declare Vbcfl type.

- typedef enum eVsurf_Meth Vsurf_Meth

  Declaration of the Vsurf_Meth type as the Vsurf_Meth enum.

- typedef enum eVchrg_Meth Vchrg_Meth

  Declaration of the Vchrg_Meth type as the Vchrg_Meth enum.

- typedef enum eVchrg_Src Vchrg_Src

  Declaration of the Vchrg_Src type as the Vchrg_Meth enum.

- typedef enum eVdata_Type Vdata_Type

  Declaration of the Vdata_Type type as the Vdata_Type enum.

- typedef enum eVdata_Format Vdata_Format

  Declaration of the Vdata_Format type as the Vdata_Format enum.

Enumerations

- enum eVrc_Codes { **VRC_WARNING** = -1, VRC_FAILURE = 0, VRC_SUCCESS = 1 }

  Return code enumerations.
- enum eVsol_Meth {
  VSOL_CGMG, **VSOL_Newton**, **VSOL_MG**, **VSOL_CG**,
  VSOL_SOR, **VSOL_RBGS**, **VSOL_WJ**, **VSOL_Richardson**,
  VSOL_CGMGAqua, **VSOL_NewtonAqua** }

  Solution Method enumerations.
- enum eVsurf_Meth {
  VSM_MOL = 0, VSM_MOLSMOOTH = 1, VSM_SPLINE = 2, VSM_SPLINE3 = 3,
  VSM_SPLINE4 = 4 }

  Types of molecular surface definitions.
- enum eVhal_PBEType {
  PBE_LPBE, PBE_NPBE, PBE_LRPBE, **PBE_NRPBE**,
  PBE_SMPBE }

  Version of PBE to solve.
- enum eVhal_IPKEYType { IPKEY_SMPBE = -2, IPKEY_LPBE, IPKEY_NPBE }

  Type of ipkey to use for MG methods.
- enum eVhal_NONLINType {
  NONLIN_LPBE = 0, **NONLIN_NPBE**, **NONLIN_SMPBE**, **NONLIN_LPBEAQUA**,
  NONLIN_NPBEAQUA }

  Type of nonlinear to use for MG methods.
- enum eVoutput_Format { OUTPUT_NULL, OUTPUT_FLAT }

  Output file format.
- enum eVbcfl {
  BCFL_ZERO = 0, BCFL_SDH = 1, BCFL_MDH = 2, BCFL_UNUSED = 3,
  BCFL_FOCUS = 4, BCFL_MEM = 5, BCFL_MAP = 6 }

  Types of boundary conditions.
- enum eVchrg_Meth { VCM_TRIL = 0, VCM_BSPL2 = 1, VCM_BSPL4 = 2 }

  Types of charge discretization methods.
- enum eVchrg_Src { VCM_CHARGE = 0, VCM_PERMANENT = 1, VCM_INDUCED = 2, VCM_NLINDUCED = 3 }

  Charge source.
- enum eVdata_Type {
  VDT_CHARGE, VDT_POT, VDT_ATOMPOT, VDT_SMOL,
  VDT_SSPL, VDT_VDW, VDT_IVDW, VDT_LAP,
  VDT_EDENS, VDT_NDENS, VDT_QDENS, VDT_DIELX,
  VDT_DIELY, VDT_DIELZ, VDT_KAPPA }

  Types of (scalar) data that can be written out of APBS.
- enum eVdata_Format {
  VDF_DX = 0, VDF_UHBD = 1, VDF_AVS = 2, VDF_MCSF = 3,
  VDF_GZ = 4, VDF_FLAT = 5 }

  Format of data for APBS I/O.

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 415 of file vhal.h.

7.15.2.2 #define VAPBS_BACK 4

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line 439 of file vhal.h.

7.15.2.3 #define VAPBS_DOWN 5

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line 445 of file vhal.h.

7.15.2.4 #define VAPBS_FRONT 1

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line 421 of file vhal.h.

7.15.2.5 #define VAPBS_LEFT 3

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line 433 of file vhal.h.

7.15.2.6 #define VAPBS_RIGHT 0

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line 409 of file vhal.h.

7.15.2.7 #define VAPBS_UP 2

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line 427 of file vhal.h.

7.15.2.8 #define VEMBED(*rctag*)

Value:

```
VPRIVATE const char* rctag; \
         static void* use_rcsid=(0 ? &use_rcsid : (void**)&rcsid);
```

Allows embedding of RCS ID tags in object files.

Author

Mike Holst

Definition at line 563 of file vhal.h.

7.15.2.9 #define VFLOOR(*value*) floor(value)

Wrapped floor to fix floating point issues in the Intel compiler.

Author

Todd Dolinksy

Definition at line 554 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 209 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 232 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 253 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_AVS AVS UCD format

VDF_MCSF FEtk MC Simplex Format (MCSF)

VDF_GZ Binary file (GZip)

VDF_FLAT Write flat file

Definition at line 311 of file vhal.h.

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/A^2)

VDT_EDENS Energy density $\varepsilon(\nabla u)^2$, where u is potential (kT/e/A)^2

VDT_NDENS Ion number density $\sum c_i \exp(-q_i u)^2$, where u is potential (output in M)

VDT_QDENS Ion charge density $\sum q_i c_i \exp(-q_i u)^2$, where u is potential (output in $e_c M$)

VDT_DIELX Dielectric x-shifted map as calculated with the currently specified scheme (dimensionless)

VDT_DIELY Dielectric y-shifted map as calculated with the currently specified scheme (dimensionless)

VDT_DIELZ Dielectric y-shifted map as calculated with the currently specified scheme (dimensionless)

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

Definition at line 271 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 159 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 141 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 193 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 68 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 83 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 104 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.

- VPRIVATE int readFlatFileLine (Vio *sock, Vparam_AtomData *atom)

  Read a single line of the flat file database.

- VPRIVATE int readXMLFileAtom (Vio *sock, Vparam_AtomData *atom)

  Read atom information from an XML file.

Variables

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

  Whitespace characters for socket reads.

- VPRIVATE char * MCcommChars = "#%"

  Comment characters for socket reads.

- VPRIVATE char * MCxmlwhiteChars = " =,;\t\n\r<>"

  Whitespace characters for XML socket reads.

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

sock	Socket ready for reading
atom	Atom to hold parsed data

Returns

1 if successful, 0 otherwise

Definition at line 696 of file vparam.c.

Here is the caller graph for this function:

7.17.2.2 VPRIVATE int readXMLFileAtom (Vio * *sock*, Vparam_AtomData * *atom*)

Read atom information from an XML file.

Author

Todd Dolinsky

Parameters

sock	Socket ready for reading
atom	Atom to hold parsed data

Returns

1 if successful, 0 otherwise

Definition at line 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

thee	Pointer to destination object
src	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

thee	Pointer to source object
dest	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

thee	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

thee	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

thee	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

thee	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

thee	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

thee	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

thee	Vparam object
resName	Residue name
atomName	Atom name

Returns

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

Note

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

See Also

Vparam_readFlatFile)

Definition at line 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

thee	Vparam object
resName	Residue name

Returns

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

Note

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

See Also

Vparam_readFlatFile)

Definition at line 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

thee	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

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

Returns

1 if successful, 0 otherwise

Note

The database file should have the following format:

```
RESIDUE ATOM CHARGE RADIUS EPSILON
```

where RESIDUE is the residue name string, ATOM is the atom name string, CHARGE is the charge in e, RADIUS is the van der Waals radius (σ_i) in Å, and EPSILON is the van der Waals well-depth (ε_i) in kJ/mol. See the Vparam structure documentation for the precise definitions of σ_i and ε_i.

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

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

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

Definition at line 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 * *iofmt*, const char * *thost*, const char * *fname*)

Read an XML format parameter database.

Author

Todd Dolinsky

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 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

thee	Pointer to source object
dest	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

thee	Allocated memory
mem	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

thee	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

thee	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)

Get coordinates of solute center.

- VEXTERNC double Vpbe_getSoluteCharge (Vpbe *thee)

  Get total solute charge.
- VEXTERNC double Vpbe_getSolventDiel (Vpbe *thee)

  Get solvent dielectric constant.
- VEXTERNC double Vpbe_getSolventRadius (Vpbe *thee)

  Get solvent molecule radius.
- VEXTERNC double Vpbe_getXkappa (Vpbe *thee)

  Get Debye-Huckel parameter.
- VEXTERNC double Vpbe_getDeblen (Vpbe *thee)

  Get Debye-Huckel screening length.
- VEXTERNC double Vpbe_getZkappa2 (Vpbe *thee)

  Get modified squared Debye-Huckel parameter.
- VEXTERNC double Vpbe_getZmagic (Vpbe *thee)

  Get charge scaling factor.
- VEXTERNC double Vpbe_getzmem (Vpbe *thee)

  Get z position of the membrane bottom.
- VEXTERNC double Vpbe_getLmem (Vpbe *thee)

  Get length of the membrane (A)
  aauthor Michael Grabe.
- VEXTERNC double Vpbe_getmembraneDiel (Vpbe *thee)

  Get membrane dielectric constant.
- VEXTERNC double Vpbe_getmemv (Vpbe *thee)

  Get membrane potential (kT)
- VEXTERNC Vpbe * Vpbe_ctor (Valist *alist, int ionNum, double *ionConc, double *ionRadii, double *ionQ, double T, double soluteDiel, double solventDiel, double solventRadius, int focusFlag, double sdens, double z_mem, double L, double membraneDiel, double V)

  Construct Vpbe object.
- VEXTERNC int Vpbe_ctor2 (Vpbe *thee, Valist *alist, int ionNum, double *ionConc, double *ionRadii, double *ionQ, double T, double soluteDiel, double solventDiel, double solventRadius, int focusFlag, double sdens, double z_mem, double L, double membraneDiel, double V)

  FORTRAN stub to construct Vpbe objct.
- VEXTERNC int Vpbe_getIons (Vpbe *thee, int *nion, double ionConc[MAXION], double ionRadii[MAXION], double ionQ[MAXION])

  Get information about the counterion species present.
- VEXTERNC void Vpbe_dtor (Vpbe **thee)

  Object destructor.
- VEXTERNC void Vpbe_dtor2 (Vpbe *thee)

  FORTRAN stub object destructor.
- VEXTERNC double Vpbe_getCoulombEnergy1 (Vpbe *thee)

  Calculate coulombic energy of set of charges.
- VEXTERNC unsigned long int Vpbe_memChk (Vpbe *thee)

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

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:

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

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

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

and the factor of 10^{-16} results from converting cm^2 to angstroms^2, noting that the 1000 in the denominator has converted m^3 to cm^3, since the ionic strength I_s is assumed to have been provided in moles per liter, which is moles per 1000 cm^3.

Returns

Pointer to newly allocated Vpbe object

Parameters

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

Definition at line 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 objct.

Author

Nathan Baker and Mike Holst and Michael Grabe

Note

This is partially based on some of Mike Holst's PMG code. Here are a few of the original function comments: kappa is defined as follows:

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

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

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

and the factor of 10^{-16} results from converting cm^2 to angstroms^2, noting that the 1000 in the denominator has converted m^3 to cm^3, since the ionic strength I_s is assumed to have been provided in moles per liter, which is moles per 1000 cm^3.

Bug The focusing flag is currently not used!!

Returns

1 if successful, 0 otherwise

Parameters

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

Definition at line 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

thee	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

thee	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

thee	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

thee	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-Huckel screening length.

Author

Nathan Baker

Parameters

thee	Vpbe object

Returns

Debye-Huckel 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

thee	Vpbe object

Returns

Apolar coefficent (kJ/mol/A^2)

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

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

Returns

Number of ions

Definition at line 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)

aauthor Michael Grabe.

Parameters

thee	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

thee	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

thee	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

thee	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

thee	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

thee	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

thee	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

thee	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

thee	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

thee	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

thee	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

thee	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

thee	Vpbe 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

thee	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

thee	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

thee	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-Huckel parameter.

Author

Nathan Baker

Parameters

thee	Vpbe object

Returns

Bulk Debye-Huckel parameter (Å)

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-Huckel parameter.

Author

Nathan Baker

Parameters

thee	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

thee	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

thee	Vpbe 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

thee	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

tok	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

s1	First string for comparison
s2	Second string for comparison

Returns

An integer less than, equal to, or greater than zero if s1 is found, respectively, to be less than, to match, or be greater than s2. (Source: Linux man pages)

Definition at line 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

str	The input string to wrap and indent
right_margin	The number of characters to the right margin
left_padding	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

  $e_c{}^2/$ *in ESU units => kcal/mol*
- #define Vunit_esu_kb 1.9871913e-03

  k_b *in ESU units => 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

nx	Number grid points in x direction
ny	Number grid points in y direction
nz	Number grid points in z direction
hx	Grid spacing in x direction
hy	Grid spacing in y direction
hzed	Grid spacing in z direction
xmin	x coordinate of lower grid corner
ymin	y coordinate of lower grid corner
zmin	z coordinate of lower grid corner
data	nx*ny*nz array of data. This can be VNULL if you are planning to read in data later with one of the read routines

Returns

Newly allocated and initialized Vgrid object

Definition at line 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

thee	Pointer to newly allocated Vgrid object
nx	Number grid points in x direction
ny	Number grid points in y direction
nz	Number grid points in z direction
hx	Grid spacing in x direction
hy	Grid spacing in y direction
hzed	Grid spacing in z direction
xmin	x coordinate of lower grid corner
ymin	y coordinate of lower grid corner
zmin	z coordinate of lower grid corner
data	nx∗ny∗nz array of data. This can be VNULL if you are planning to read in data later with one of the read routines

Returns

Newly allocated and initialized Vgrid object

Definition at line 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

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

Returns

1 if successful, 0 if off grid

Definition at line 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

thee	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

thee	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

thee	Pointer to Vgrid object
pt	Location to evaluate gradient
grad	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

thee	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

thee	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

thee	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

thee	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

thee	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

thee	Vgrid object

Returns

$L\infty$ 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

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

Returns

1 if sucessful, 0 otherwise

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

thee	Object with grid data to write
fname	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

thee	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

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

Returns

1 if successful, 0 if off grid

Definition at line 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

thee	Grid object
iodev	Output device type (FILE/BUFF/UNIX/INET)
iofmt	Output device format (ASCII/XDR)
thost	Output hostname (for sockets)
fname	Output FILE/BUFF/UNIX/INET name
title	Title to be inserted in grid file
pvec	Partition weight (if 1: point in current partition, if 0 point not in current partition if $>$ 0 && $<$ 1 point on/near boundary)

Definition at line 1002 of file vgrid.c.

7.21.2.18 VEXTERNC void Vgrid_writeUHBD (Vgrid * *thee*, const char * *iodev*, const char * *iofmt*, const char * *thost*, const char * *fname*, char * *title*, double * *pvec*)

Write out the data in UHBD grid format.

Note

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

Author

Nathan Baker

Parameters

thee	Grid object
iodev	Output device type (FILE/BUFF/UNIX/INET)
iofmt	Output device format (ASCII/XDR)
thost	Output hostname (for sockets)
fname	Output FILE/BUFF/UNIX/INET name
title	Title to be inserted in grid file
pvec	Partition weight (if 1: point in current partition, if 0 point not in current partition if $>$ 0 && $<$ 1 point on/near boundary)

Bug This routine does not respect partition information

Definition at line 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

  Multiresoltion oracle for Cartesian mesh data.

Macros

- #define VMGRIDMAX 20

  The maximum number of levels in the grid hiearchy.

Typedefs

- typedef struct sVmgrid Vmgrid

  Declaration of the Vmgrid class as the Vgmrid structure.

Functions

- VEXTERNC Vmgrid ∗ Vmgrid_ctor ()

  Construct Vmgrid object.
- VEXTERNC int Vmgrid_ctor2 (Vmgrid ∗thee)

  Initialize Vmgrid object.
- VEXTERNC int Vmgrid_value (Vmgrid ∗thee, double x[3], double ∗value)

  Get potential value (from mesh or approximation) at a point.
- VEXTERNC void Vmgrid_dtor (Vmgrid ∗∗thee)

  Object destructor.
- VEXTERNC void Vmgrid_dtor2 (Vmgrid ∗thee)

  FORTRAN stub object destructor.
- VEXTERNC int Vmgrid_addGrid (Vmgrid ∗thee, Vgrid ∗grid)

  Add a grid to the hierarchy.
- VEXTERNC int Vmgrid_curvature (Vmgrid ∗thee, double pt[3], int cflag, double ∗curv)

  Get second derivative values at a point.
- VEXTERNC int Vmgrid_gradient (Vmgrid ∗thee, double pt[3], double grad[3])

  Get first derivative values at a point.
- VEXTERNC Vgrid ∗ Vmgrid_getGridByNum (Vmgrid ∗thee, int num)

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

  Get grid in hiearchy which contains specified point or VNULL.

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

thee	Pointer to object to be destroyed
grid	Grid to be added. As mentioned above, we would prefer to have the finest grid added first, next-finest second, ..., coarsest last – this is how the grid will be searched when looking up values for points. However, this is not enforced to provide flexibility for cases where the dataset is decomposed into disjoint partitions, etc.

Returns

1 if successful, 0 otherwise

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

thee	Newly allocated Vmgrid 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

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

Returns

1 if successful, 0 if off grid

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

Object destructor.

Author

Nathan Baker

Parameters

thee	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

thee	Pointer to object to be destroyed

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

Get specific grid in hiearchy.

Author

Nathan Baker

Parameters

thee	Pointer to Vmgrid object
num	Number of grid in hiearchy

Returns

Pointer to specified grid

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

Get grid in hiearchy which contains specified point or VNULL.

Author

Nathan Baker

Parameters

thee	Pointer to Vmgrid object
pt	Point to check

Returns

Pointer to specified grid

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

thee	Pointer to Vmgrid object
pt	Location to evaluate gradient
grad	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

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

Returns

1 if successful, 0 if off grid

7.23 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.

7.23.1 Detailed Description

Potential oracle for Cartesian mesh data.

7.23.2 Function Documentation

7.23.2.1 VEXTERNC Vopot* Vopot_ctor (Vmgrid * *mgrid*, Vpbe * *pbe*, Vbcfl *bcfl*)

Construct Vopot object with values obtained from Vpmg_readDX (for example)

Author

Nathan Baker

Parameters

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

Returns

Newly allocated and initialized Vopot object

Definition at line 67 of file vopot.c.

7.23.2.2 VEXTERNC int Vopot_ctor2 (Vopot * *thee*, Vmgrid * *mgrid*, Vpbe * *pbe*, Vbcfl *bcfl*)

Initialize Vopot object with values obtained from Vpmg_readDX (for example)

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 82 of file vopot.c.

7.23.2.3 VEXTERNC int Vopot_curvature (Vopot * *thee*, double *pt[3]*, int *cflag*, double * *curv*)

Get second derivative values at a point.

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 216 of file vopot.c.

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

Object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of object to be destroyed

Definition at line 96 of file vopot.c.

7.23.2.5 VEXTERNC void Vopot_dtor2 (Vopot * *thee*)

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to object to be destroyed

Definition at line 109 of file vopot.c.

7.23.2.6 VEXTERNC int Vopot_gradient (Vopot * *thee*, double *pt[3]*, double *grad[3]*)

Get first derivative values at a point.

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 302 of file vopot.c.

7.23.2.7 VEXTERNC int Vopot_pot (Vopot * *thee*, double *x[3]*, double * *pot*)

Get potential value (from mesh or approximation) at a point.

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 116 of file vopot.c.

7.24 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_qfEnergy (Vpmg ∗thee, int extFlag)

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

- VEXTERNC double Vpmg_qfAtomEnergy (Vpmg *thee, Vatom *atom)

  Get the per-atom "fixed charge" contribution to the electrostatic energy.
- VEXTERNC double Vpmg_qmEnergy (Vpmg *thee, int extFlag)

  Get the "mobile charge" contribution to the electrostatic energy.
- VEXTERNC double Vpmg_dielEnergy (Vpmg *thee, int extFlag)

  Get the "polarization" contribution to the electrostatic energy.
- VEXTERNC double Vpmg_dielGradNorm (Vpmg *thee)

  Get the integral of the gradient of the dielectric function.
- VEXTERNC int Vpmg_force (Vpmg *thee, double *force, int atomID, Vsurf_Meth srfm, Vchrg_Meth chgm)

  Calculate the total force on the specified atom in units of k_B T/AA.
- VEXTERNC int Vpmg_qfForce (Vpmg *thee, double *force, int atomID, Vchrg_Meth chgm)

  Calculate the "charge-field" force on the specified atom in units of k_B T/AA.
- VEXTERNC int Vpmg_dbForce (Vpmg *thee, double *dbForce, int atomID, Vsurf_Meth srfm)

  Calculate the dielectric boundary forces on the specified atom in units of k_B T/AA.
- VEXTERNC int Vpmg_ibForce (Vpmg *thee, double *force, int atomID, Vsurf_Meth srfm)

  Calculate the osmotic pressure on the specified atom in units of k_B T/AA.
- VEXTERNC void Vpmg_setPart (Vpmg *thee, double lowerCorner[3], double upperCorner[3], int bflags[6])

  Set partition information which restricts the calculation of observables to a (rectangular) subset of the problem domain.
- VEXTERNC void Vpmg_unsetPart (Vpmg *thee)

  Remove partition restrictions.
- VEXTERNC int Vpmg_fillArray (Vpmg *thee, double *vec, Vdata_Type type, double parm, Vhal_PBEType pbetype, PBEparm *pbeparm)

  Fill the specified array with accessibility values.
- VPUBLIC void Vpmg_fieldSpline4 (Vpmg *thee, int atomID, double field[3])

  Computes the field at an atomic center using a stencil based on the first derivative of a 5th order B-spline.
- VEXTERNC double Vpmg_qfPermanentMultipoleEnergy (Vpmg *thee, int atomID)

  Computes the permanent multipole electrostatic hydration energy (the polarization component of the hydration energy currently computed in TINKER).
- VEXTERNC void Vpmg_qfPermanentMultipoleForce (Vpmg *thee, int atomID, double force[3], double torque[3])

  Computes the q-Phi Force for permanent multipoles based on 5th order B-splines.
- VEXTERNC void Vpmg_ibPermanentMultipoleForce (Vpmg *thee, int atomID, double force[3])

  Compute the ionic boundary force for permanent multipoles.
- VEXTERNC void Vpmg_dbPermanentMultipoleForce (Vpmg *thee, int atomID, double force[3])

  Compute the dielectric boundary force for permanent multipoles.
- VEXTERNC void Vpmg_qfDirectPolForce (Vpmg *thee, Vgrid *perm, Vgrid *induced, int atomID, double force[3], double torque[3])

  q-Phi direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.
- VEXTERNC void Vpmg_qfNLDirectPolForce (Vpmg *thee, Vgrid *perm, Vgrid *nlInduced, int atomID, double force[3], double torque[3])

  q-Phi direct polarization force between permanent multipoles and non-local induced dipoles based on 5th Order B-Splines. Keep in mind that the "non-local" induced dipooles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.
- VEXTERNC void Vpmg_ibDirectPolForce (Vpmg *thee, Vgrid *perm, Vgrid *induced, int atomID, double force[3])

  Ionic boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.

- VEXTERNC void Vpmg_ibNLDirectPolForce (Vpmg *thee, Vgrid *perm, Vgrid *nlInduced, int atomID, double force[3])

  Ionic boundary direct polarization force between permanent multipoles and non-local induced dipoles based on 5th order Keep in mind that the "non-local" induced dipooles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

- VEXTERNC void Vpmg_dbDirectPolForce (Vpmg *thee, Vgrid *perm, Vgrid *induced, int atomID, double force[3])

  Dielectric boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.

- VEXTERNC void Vpmg_dbNLDirectPolForce (Vpmg *thee, Vgrid *perm, Vgrid *nlInduced, int atomID, double force[3])

  Dielectric bounday direct polarization force between permanent multipoles and non-local induced dipoles. Keep in mind that the "non-local" induced dipooles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

- VEXTERNC void Vpmg_qfMutualPolForce (Vpmg *thee, Vgrid *induced, Vgrid *nlInduced, int atomID, double force[3])

  Mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

- VEXTERNC void Vpmg_ibMutualPolForce (Vpmg *thee, Vgrid *induced, Vgrid *nlInduced, int atomID, double force[3])

  Ionic boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

- VEXTERNC void Vpmg_dbMutualPolForce (Vpmg *thee, Vgrid *induced, Vgrid *nlInduced, int atomID, double force[3])

  Dielectric boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

- VEXTERNC void Vpmg_printColComp (Vpmg *thee, char path[72], char title[72], char mxtype[3], int flag)

  Print out a column-compressed sparse matrix in Harwell-Boeing format.

- VPRIVATE void bcolcomp (int *iparm, double *rparm, int *iwork, double *rwork, double *values, int *rowind, int *colptr, int *flag)

  Build a column-compressed matrix in Harwell-Boeing format.

- VPRIVATE void bcolcomp2 (int *iparm, double *rparm, int *nx, int *ny, int *nz, int *iz, int *ipc, double *rpc, double *ac, double *cc, double *values, int *rowind, int *colptr, int *flag)

  Build a column-compressed matrix in Harwell-Boeing format.

- VPRIVATE void bcolcomp3 (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *ac, double *cc, double *values, int *rowind, int *colptr, int *flag)

  Build a column-compressed matrix in Harwell-Boeing format.

- VPRIVATE void bcolcomp4 (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *oC, double *cc, double *oE, double *oN, double *uC, double *values, int *rowind, int *colptr, int *flag)

  Build a column-compressed matrix in Harwell-Boeing format.

- VPRIVATE void pcolcomp (int *nrow, int *ncol, int *nnzero, double *values, int *rowind, int *colptr, char *path, char *title, char *mxtype)

  Print a column-compressed matrix in Harwell-Boeing format.

- VEXTERNC void Vpackmg (int *iparm, double *rparm, 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.

7.24.1 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.

7.24.2 Function Documentation

7.24.2.1 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

iparm	
rparm	
iwork	
rwork	
values	
rowind	
colptr	
flag	Operation selection parameter 0 = Use Poisson operator only 1 = Use linearization of full operation around current solution.

Definition at line 10732 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.24.2.2 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

iparm	
rparm	
nx	
ny	
nz	
iz	
ipc	
rpc	
ac	
cc	
values	
rowind	
colptr	
flag	Operation selection parameter 0 = Use Poisson operator only 1 = Use linearization of full operation around current solution.

Definition at line 10786 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.24.2.3 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

nx	
ny	
nz	
ipc	
rpc	
ac	
cc	
values	
rowind	
colptr	
flag	

Definition at line 10822 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.24.2.4 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

nx	
ny	
nz	
ipc	
rpc	
oC	
cc	
oE	
oN	
uC	
values	
rowind	
colptr	
flag	

Definition at line 10849 of file vpmg.c.

Here is the caller graph for this function:

7.24.2.5 VPRIVATE void pcolcomp (int * *nrow,* int * *ncol,* int * *nnzero,* double * *values,* int * *rowind,* int * *colptr,* char * *path,* char * *title,* char * *mxtype*)

Print a column-compressed matrix in Harwell-Boeing format.

Author

Tucker Beck [C Translation] Nathan Baker [Original] (mostly ripped off from Harwell-Boeing format documentation)Michael Schnieders)

Parameters

nrow	
ncol	
nnzero	
values	
rowind	
colptr	
path	
title	
mxtype	

Definition at line 11014 of file vpmg.c.

Here is the caller graph for this function:

7.24.2.6 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.

Author

Nathan Baker

Bug Can this path variable be replaced with a Vio socket?

Definition at line 550 of file mgsubd.c.

Here is the caller graph for this function:

7.24.2.7 VEXTERNC Vpmg* Vpmg_ctor (Vpmgp * *parms*, Vpbe * *pbe*, int *focusFlag*, Vpmg * *pmgOLD*, MGparm * *mgparm*, PBEparm_calcEnergy *energyFlag*)

Constructor for the Vpmg class (allocates new memory)

Author

Nathan Baker

Returns

Pointer to newly allocated Vpmg object

Parameters

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

Definition at line 142 of file vpmg.c.

Here is the call graph for this function:

7.24.2.8 VEXTERNC int Vpmg_ctor2 (Vpmg * *thee*, Vpmgp * *parms*, Vpbe * *pbe*, int *focusFlag*, Vpmg * *pmgOLD*, MGparm * *mgparm*, PBEparm_calcEnergy *energyFlag*)

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

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Note

this is common to both replace/noreplace options
The fortran replacement functions are run along side the old fortran functions. This is due to the use of common variables in the fortran sub-routines. Once the fortran code has been successfully excised, these functions will no longer need to be called in tandem, and the fortran version may be dropped

Parameters

thee	Memory location for object
parms	PMG parameter object
pbe	PBE-specific variables

focusFlag	1 for focusing, 0 otherwise
pmgOLD	Old Vpmg object to use for boundary conditions (can be VNULL if focusFlag = 0)
mgparm	MGparm parameter object for boundary conditions (can be VNULL if focusFlag = 0)
energyFlag	What types of energies to calculate (ignored if focusFlag = 0)

Definition at line 154 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.24.2.9 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

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

7.24.2.10 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

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

Definition at line 6001 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.24.2.11 VEXTERNC void Vpmg_dbMutualPolForce (Vpmg * *thee*, Vgrid * *induced*, Vgrid * *nlInduced*, int *atomID*, double *force[3]*)

Dielectric boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

Author

Michael Schnieders

Parameters

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

7.24.2.12 VEXTERNC void Vpmg_dbNLDirectPolForce (Vpmg * *thee*, Vgrid * *perm*, Vgrid * *nlInduced*, int *atomID*, double *force[3]*)

Dielectric bounday direct polarization force between permanent multipoles and non-local induced dipoles. Keep in mind that the "non-local" induced dipooles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

Author

Michael Schnieders

Parameters

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

7.24.2.13 VEXTERNC void Vpmg_dbPermanentMultipoleForce (Vpmg * *thee*, int *atomID*, double *force[3]*)

Compute the dielectric boundary force for permanent multipoles.

Author

Michael Schnieders

Parameters

thee	Vpmg object
atomID	Atom index
force	(returned) force

7.24.2.14 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:
```

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

where epsilon is the dielectric parameter and u(x) is the dimensionless electrostatic potential. The energy is scaled to units of k_b T.

Author

Nathan Baker

Note

The value of this observable may be modified by setting restrictions on the subdomain over which it is calculated. Such limits can be set via Vpmg_setPart and are generally useful for parallel runs.

Returns

The polarization electrostatic energy in units of k_B T.

Parameters

thee	Vpmg object
extFlag	If this was a focused calculation, include (1 – for serial calculations) or ignore (0 – for parallel calculations) energy contributions from outside the focusing domain

Definition at line 1285 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.24.2.15 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):
```

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

where epsilon is the dielectric parameter. The integral is returned in units of A^2.

Author

Nathan Baker restrictions on the subdomain over which it is calculated. Such limits can be set via Vpmg_setPart and are generally useful for parallel runs.

Returns

The integral in units of A^2.

Parameters

thee	Vpmg object

Definition at line 1348 of file vpmg.c.

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

Object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of object to be destroyed

Definition at line 560 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.24.2.17 VEXTERNC void Vpmg_dtor2 (Vpmg * *thee*)

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to object to be destroyed

Definition at line 570 of file vpmg.c.

Here is the caller graph for this function:

7.24.2.18 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

thee	Vpmg object
extFlag	If this was a focused calculation, include (1 – for serial calculations) or ignore (0 – for parallel calculations) energy contributions from outside the focusing domain

Definition at line 1254 of file vpmg.c.

Here is the call graph for this function:

7.24.2.19 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

thee	Vpmg object
atomID	Atom index
field	The (returned) electric field

7.24.2.20 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

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

Definition at line 891 of file vpmg.c.

Here is the call graph for this function:

7.24.2.21 VEXTERNC int Vpmg_fillco (Vpmg * *thee*, Vsurf_Meth *surfMeth*, double *splineWin*, Vchrg_Meth *chargeMeth*, int *useDielXMap*, Vgrid * *dielXMap*, int *useDielYMap*, Vgrid * *dielYMap*, int *useDielZMap*, Vgrid * *dielZMap*, int *useKappaMap*, Vgrid * *kappaMap*, int *usePotMap*, Vgrid * *potMap*, int *useChargeMap*, Vgrid * *chargeMap*)

Fill the coefficient arrays prior to solving the equation.

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

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

Parameters

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

Definition at line 5646 of file vpmg.c.

Here is the call graph for this function:

7.24.2.22 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/AA.

Author

Nathan Baker

Note

- Using the force evaluation methods of Im et al (Roux group), Comput Phys Commun, 111, 59–75 (1998). However, this gives the whole (self-interactions included) force – reaction field forces will have to be calculated at higher level.

- No contributions are made from higher levels of focusing.

Returns

1 if successful, 0 otherwise

Parameters

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

Definition at line 5813 of file vpmg.c.

Here is the call graph for this function:

7.24.2.23 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

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

7.24.2.24 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

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

Definition at line 5836 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.24.2.25 VEXTERNC void Vpmg_ibMutualPolForce (Vpmg * *thee*, Vgrid * *induced*, Vgrid * *nlInduced*, int *atomID*, double *force[3]*)

Ionic boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

Author

Michael Schnieders

Parameters

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

7.24.2.26 VEXTERNC void Vpmg_ibNLDirectPolForce (Vpmg * *thee*, Vgrid * *perm*, Vgrid * *nlInduced*, int *atomID*, double *force[3]*)

Ionic boundary direct polarization force between permanent multipoles and non-local induced dipoles based on 5th order Keep in mind that the "non-local" induced dipooles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

Author

Michael Schnieders

Parameters

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

7.24.2.27 VEXTERNC void Vpmg_ibPermanentMultipoleForce (Vpmg * *thee*, int *atomID*, double *force[3]*)

Compute the ionic boundary force for permanent multipoles.

Author

Michael Schnieders

Parameters

thee	Vpmg object
atomID	Atom index
force	(returned) force

7.24.2.28 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

thee	Object for memory check

Definition at line 80 of file vpmg.c.

7.24.2.29 VEXTERNC void Vpmg_printColComp (Vpmg * *thee*, char *path[72]*, char *title[72]*, char *mxtype[3]*, int *flag*)

Print out a column-compressed sparse matrix in Harwell-Boeing format.

Author

Nathan Baker

Bug Can this path variable be replaced with a Vio socket?

Parameters

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

Definition at line 88 of file vpmg.c.

Here is the call graph for this function:

7.24.2.30 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: \form#65 where q$ is the
charge and r is the location of the atom of interest.  The
result is returned in units of k_B T.  Clearly, no
self-interaction terms are removed.  A factor a 1/2 has to be
included to convert this to a real energy.
```

Author

Nathan Baker

Note

The value of this observable may be modified by setting restrictions on the subdomain over which it is calculated. Such limits can be set via Vpmg_setPart and are generally useful for parallel runs.

Returns

The fixed charge electrostatic energy in units of k_B T.

Parameters

thee	The Vpmg object
atom	The atom for energy calculations

Definition at line 1797 of file vpmg.c.

Here is the call graph for this function:

7.24.2.31 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

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

7.24.2.32 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

thee	Vpmg object
extFlag	If this was a focused calculation, include (1 – for serial calculations) or ignore (0 – for parallel calculations) energy contributions from outside the focusing domain

Definition at line 1693 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.24.2.33 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

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

Definition at line 6258 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.24.2.34 VEXTERNC void Vpmg_qfMutualPolForce (Vpmg * *thee,* Vgrid * *induced,* Vgrid * *nlInduced,* int *atomID,* double *force[3]*)

Mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

Author

Michael Schnieders

Parameters

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

7.24.2.35 VEXTERNC void Vpmg_qfNLDirectPolForce (Vpmg * *thee,* Vgrid * *perm,* Vgrid * *nlInduced,* int *atomID,* double *force[3],* double *torque[3]*)

q-Phi direct polarization force between permanent multipoles and non-local induced dipoles based on 5th Order B--Splines. Keep in mind that the "non-local" induced dipooles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

Author

Michael Schnieders

Parameters

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

7.24.2.36 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

thee	Vpmg object
atomID	Atom index

7.24.2.37 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

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

7.24.2.38 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

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

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

Author

Nathan Baker

Note

The value of this observable may be modified by setting restrictions on the subdomain over which it is calculated. Such limits can be set via Vpmg_setPart and are generally useful for parallel runs.

Returns

The mobile charge electrostatic energy in units of k_B T.

Parameters

thee	Vpmg object
extFlag	If this was a focused calculation, include (1 – for serial calculations) or ignore (0 – for parallel calculations) energy contributions from outside the focusing domain

Definition at line 1392 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.24.2.39 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

thee	Vpmg object
lowerCorner	Partition lower corner
upperCorner	Partition upper corner
bflags	Booleans indicating whether a particular processor is on the boundary with another partition. 0 if the face is not bounded (next to) another partition, and 1 otherwise.

Definition at line 626 of file vpmg.c.

Here is the call graph for this function:

7.24.2.40 VEXTERNC int Vpmg_solve (Vpmg * *thee*)

Solve the PBE using PMG.

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Parameters

thee	Vpmg object

Definition at line 400 of file vpmg.c.

Here is the call graph for this function:

7.24.2.41 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

thee	Vpmg object

Definition at line 7033 of file vpmg.c.

Here is the call graph for this function:

7.24.2.42 VEXTERNC void Vpmg_unsetPart (Vpmg * *thee*)

Remove partition restrictions.

Author

Nathan Baker

Parameters

thee	Vpmg object

Definition at line 871 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.25 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.

7.25.1 Detailed Description

Parameter structure for Mike Holst's PMGP code.

Note

Variables and many default values taken directly from PMG

7.25.2 Function Documentation

7.25.2.1 VEXTERNC Vpmgp* Vpmgp_ctor (MGparm * *mgparm*)

Construct PMG parameter object and initialize to default values.

Author

Nathan Baker

Parameters

mgparm	MGParm object containing parameters to be used in setup

Returns

Newly allocated and initialized Vpmgp object

Definition at line 78 of file vpmgp.c.

7.25.2.2 VEXTERNC int Vpmgp_ctor2 (Vpmgp * *thee*, MGparm * *mgparm*)

FORTRAN stub to construct PMG parameter object and initialize to default values.

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 95 of file vpmgp.c.

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

Object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to memory location for Vpmgp object

Definition at line 180 of file vpmgp.c.

7.25.2.4 VEXTERNC void Vpmgp_dtor2 (Vpmgp * *thee*)

FORTRAN stub for object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to Vpmgp object

Definition at line 195 of file vpmgp.c.

7.25.2.5 VEXTERNC void Vpmgp_makeCoarse (int *numLevel,* int *nxOld,* int *nyOld,* int *nzOld,* int * *nxNew,* int * *nyNew,* int * *nzNew*)

Coarsen the grid by the desired number of levels and determine the resulting numbers of grid points.

Author

Mike Holst and Nathan Baker

Parameters

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

Definition at line 314 of file vpmgp.c.

7.25.2.6 VEXTERNC void Vpmgp_size (Vpmgp * *thee*)

Determine array sizes and parameters for multigrid solver.

Author

Mike Holst and Nathan Baker

Parameters

thee	Object to be sized

Definition at line 198 of file vpmgp.c.

Here is the caller graph for this function:

7.26 C translation of Holst group PMG code

C translation of Holst group PMG code.

Macros

- #define HARMO2(a, b) (2.0 ∗ (a) ∗ (b) / ((a) + (b)))

  Multigrid subroutines.
- #define MAXIONS 50

  Specifies the PDE definition for PMG to solve.

Functions

- 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 ∗nxf, int ∗nyf, int ∗nzf, int ∗nxc, int ∗nyc, int ∗nzc, int ∗numdia, double ∗pcFF, double ∗acFF, double ∗ac)

  Build Galerkin matrix structures.
- 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.
- 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 ∗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 ∗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.

- VEXTERNC 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.
- 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)

  Guass-Seidel solver.
- VEXTERNC void Vmatvec (int ∗nx, int ∗ny, int ∗nz, int ∗ipc, double ∗rpc, double ∗ac, double ∗cc, double ∗x, double ∗y)

  Matrix-vector multiplication routines.
- VEXTERNC void Vnmatvec (int ∗nx, int ∗ny, int ∗nz, int ∗ipc, double ∗rpc, double ∗ac, double ∗cc, double ∗x, double ∗y, double ∗w1)

  Break the matrix data-structure into diagonals and then call the matrix-vector routine.
- VEXTERNC void Vmresid (int ∗nx, int ∗ny, int ∗nz, int ∗ipc, double ∗rpc, double ∗ac, double ∗cc, double ∗fc, double ∗x, double ∗r)

  Break the matrix data-structure into diagonals and then call the residual routine.
- VEXTERNC void Vnmresid (int ∗nx, int ∗ny, int ∗nz, int ∗ipc, double ∗rpc, double ∗ac, double ∗cc, double ∗fc, double ∗x, double ∗r, double ∗w1)

  Break the matrix data-structure into diagonals and then call the residual routine.
- VEXTERNC void Vrestrc (int ∗nxf, int ∗nyf, int ∗nzf, int ∗nxc, int ∗nyc, int ∗nzc, double ∗xin, double ∗xout, double ∗pc)

  Apply the restriction operator.
- VEXTERNC void VinterpPMG (int ∗nxc, int ∗nyc, int ∗nzc, int ∗nxf, int ∗nyf, int ∗nzf, double ∗xin, double ∗xout, double ∗pc)

  Apply the prolongation operator.
- VEXTERNC void Vextrac (int ∗nxf, int ∗nyf, int ∗nzf, int ∗nxc, int ∗ny, int ∗nzc, double ∗xin, double ∗xout)

  Simple injection of a fine grid function into coarse grid.
- VEXTERNC void Vmvcs (int ∗nx, int ∗ny, int ∗nz, double ∗x, int ∗iz, double ∗w0, double ∗w1, double ∗w2, double ∗w3, int ∗istop, int ∗itmax, int ∗iters, int ∗ierror, int ∗nlev, int ∗ilev, int ∗nlev_real, int ∗mgsolv, int ∗iok, int ∗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.
- VEXTERNC void Vmgdriv (int ∗iparm, double ∗rparm, int ∗iwork, double ∗rwork, double ∗u, double ∗xf, double ∗yf, double ∗zf, double ∗gxcf, double ∗gycf, double ∗gzcf, double ∗a1cf, double ∗a2cf, double ∗a3cf, double ∗ccf, double ∗fcf, double ∗tcf)

  Multilevel solver driver.
- VEXTERNC void Vmgdriv2 (int ∗iparm, double ∗rparm, int ∗nx, int ∗ny, int ∗nz, double ∗u, int ∗iz, int ∗ipc, double ∗rpc, double ∗pc, double ∗ac, double ∗cc, double ∗fc, double ∗xf, double ∗yf, double ∗zf, double ∗gxcf, double ∗gycf, double ∗gzcf, double ∗a1cf, double ∗a2cf, double ∗a3cf, double ∗ccf, double ∗fcf, double ∗tcf)

  Solves the pde using the multi-grid method.

- VEXTERNC void Vmgsz (int *mgcoar, int *mgdisc, int *mgsolv, int *nx, int *ny, int *nz, int *nlev, int *nxc, int *nyc, int *nzc, int *nf, int *nc, int *narr, int *narrc, int *n_rpc, int *n_iz, int *n_ipc, int *iretot, int *iintot)

  This routine computes the required sizes of the real and integer work arrays for the multigrid code. these two sizes are a (complicated) function of input parameters.
- VEXTERNC void Vfmvfas (int *nx, int *ny, int *nz, double *x, int *iz, double *w0, double *w1, double *w2, double *w3, double *w4, int *istop, int *itmax, int *iters, int *ierror, int *nlev, int *ilev, int *nlev_real, int *mgsolv, int *iok, int *iinfo, double *epsiln, double *errtol, double *omega, int *nu1, int *nu2, int *mgsmoo, int *ipc, double *rpc, double *pc, double *ac, double *cc, double *fc, double *tru)

  Multigrid nonlinear solve iteration routine.
- VEXTERNC void Vmvfas (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 *gxcf, double *gycf, double *gzcf, double *a1cf, double *a2cf, double *a3cf, double *ccf, double *fcf, double *tcf)

  Build operators, boundary arrays, modify affine vectors ido==0: do only fine level ido==1: do only coarse levels (including second op at coarsest) ido==2: do all levels ido==3: rebuild the second operator at the coarsest level.
- VEXTERNC void Vbuildstr (int *nx, int *ny, int *nz, int *nlev, int *iz)

  Build the nexted operator framework in the array iz.
- VEXTERNC void Vbuildgaler0 (int *nxf, int *nyf, int *nzf, int *nxc, int *nyc, int *nzc, int *ipkey, int *numdia, double *pcFF, int *ipcFF, double *rpcFF, double *acFF, double *ccFF, double *fcFF, int *ipc, double *rpc, double *ac, double *cc, double *fc)

  Form the Galerkin coarse grid system.
- 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)

  saxpy 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 Vprtmatd (int ∗nx, int ∗ny, int ∗nz, int ∗ipc, double ∗rpc, double ∗ac)
- VEXTERNC void Vdpbsl (double ∗abd, int ∗lda, int ∗n, int ∗m, double ∗b)

  LINPACK interface.
- VEXTERNC void Vmypdefinitlpbe (int ∗tnion, double ∗tcharge, double ∗tsconc)

  Set up the ionic species to be used in later calculations. This must be called before any other of the routines in this file.
- 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.
- VEXTERNC void Vmypdefinitsmpbe (int ∗tnion, double ∗tcharge, double ∗tsconc, double ∗smvolume, double ∗smsize)

  Set up the ionic species to be used in later calculations. This must be called before any other of the routines in this file.
- VEXTERNC void Vc_vec (double ∗coef, double ∗uin, double ∗uout, int ∗nx, int ∗ny, int ∗nz, int ∗ipkey)

  Define the nonlinearity (vector version)
- VEXTERNC void Vdc_vec (double ∗coef, double ∗uin, double ∗uout, int ∗nx, int ∗ny, int ∗nz, int ∗ipkey)

  Define the derivative of the nonlinearity (vector version)
- VEXTERNC void Vc_vecpmg (double ∗coef, double ∗uin, double ∗uout, int ∗nx, int ∗ny, int ∗nz, int ∗ipkey)

  Define the nonlinearity (vector version)
- VEXTERNC void Vc_vecsmpbe (double ∗coef, double ∗uin, double ∗uout, int ∗nx, int ∗ny, int ∗nz, int ∗ipkey)

  Define the nonlinearity (vector version)
- VEXTERNC void Vnewdriv (int ∗iparm, double ∗rparm, int ∗iwork, double ∗rwork, double ∗u, double ∗xf, double ∗yf, double ∗zf, double ∗gxcf, double ∗gycf, double ∗gzcf, double ∗a1cf, double ∗a2cf, double ∗a3cf, double ∗ccf, double ∗fcf, double ∗tcf)

  Driver for the Newton Solver.
- VEXTERNC void Vnewdriv2 (int ∗iparm, double ∗rparm, int ∗nx, int ∗ny, int ∗nz, double ∗u, int ∗iz, double ∗w1, double ∗w2, int ∗ipc, double ∗rpc, double ∗pc, double ∗ac, double ∗cc, double ∗fc, double ∗xf, double ∗yf, double ∗zf, double ∗gxcf, double ∗gycf, double ∗gzcf, double ∗a1cf, double ∗a2cf, double ∗a3cf, double ∗ccf, double ∗fcf, double ∗tcf)

  Solves using Newton's Method.
- VPUBLIC void Vfnewton (int ∗nx, int ∗ny, int ∗nz, double ∗x, int ∗iz, double ∗w0, double ∗w1, double ∗w2, double ∗w3, int ∗istop, int ∗itmax, int ∗iters, int ∗ierror, int ∗nlev, int ∗ilev, int ∗nlev_real, int ∗mgsolv, int ∗iok, int ∗iinfo, double ∗epsiln, double ∗errtol, double ∗omega, int ∗nu1, int ∗nu2, int ∗mgsmoo, double ∗cprime, double ∗rhs, double ∗xtmp, int ∗ipc, double ∗rpc, double ∗pc, double ∗ac, double ∗cc, double ∗fc, double ∗tru)

  Driver routines for the Newton method.
- VEXTERNC void Vnewton (int ∗nx, int ∗ny, int ∗nz, double ∗x, int ∗iz, double ∗w0, double ∗w1, double ∗w2, double ∗w3, int ∗istop, int ∗itmax, int ∗iters, int ∗ierror, int ∗nlev, int ∗ilev, int ∗nlev_real, int ∗mgsolv, int ∗iok, int ∗iinfo, double ∗epsiln, double ∗errtol, double ∗omega, int ∗nu1, int ∗nu2, int ∗mgsmoo, double ∗cprime, double ∗rhs, double ∗xtmp, int ∗ipc, double ∗rpc, double ∗pc, double ∗ac, double ∗cc, double ∗fc, double ∗tru)

  Inexact-newton-multilevel method.
- VEXTERNC void Vgetjac (int ∗nx, int ∗ny, int ∗nz, int ∗nlev_real, int ∗iz, int ∗lev, int ∗ipkey, double ∗x, double ∗r, double ∗cprime, double ∗rhs, double ∗cc, double ∗pc)

  Form the jacobian system.
- VEXTERNC void Vpower (int ∗nx, int ∗ny, int ∗nz, int ∗iz, int ∗ilev, int ∗ipc, double ∗rpc, double ∗ac, double ∗cc, double ∗w1, double ∗w2, double ∗w3, double ∗w4, double ∗eigmax, double ∗eigmax_model, double ∗tol, int ∗itmax, int ∗iters, int ∗iinfo)

  Power methods for eigenvalue estimation.
- VEXTERNC void Vipower (int ∗nx, int ∗ny, int ∗nz, double ∗u, int ∗iz, double ∗w0, double ∗w1, double ∗w2, double ∗w3, double ∗w4, double ∗eigmin, double ∗eigmin_model, double ∗tol, int ∗itmax, int ∗iters, int ∗nlev, int ∗ilev, int ∗nlev_real, int ∗mgsolv, int ∗iok, int ∗iinfo, double ∗epsiln, double ∗errtol, double ∗omega, int ∗nu1, int ∗nu2, int ∗mgsmoo, int ∗ipc, double ∗rpc, double ∗pc, double ∗ac, double ∗cc, double ∗tru)

Standard inverse power method for minimum eigenvalue estimation.

- VEXTERNC void Vsmooth (int ∗nx, int ∗ny, int ∗nz, int ∗ipc, double ∗rpc, double ∗ac, double ∗cc, double ∗fc, double ∗x, double ∗w1, double ∗w2, double ∗r, int ∗itmax, int ∗iters, double ∗errtol, double ∗omega, int ∗iresid, int ∗iadjoint, int ∗meth)

  Multigrid smoothing functions.

- VEXTERNC void Vnsmooth (int ∗nx, int ∗ny, int ∗nz, int ∗ipc, double ∗rpc, double ∗ac, double ∗cc, double ∗fc, double ∗x, double ∗w1, double ∗w2, double ∗r, int ∗itmax, int ∗iters, double ∗errtol, double ∗omega, int ∗iresid, int ∗iadjoint, int ∗meth)

  call the appropriate non-linear smoothing routine.

7.26.1 Detailed Description

C translation of Holst group PMG code.

7.26.2 Macro Definition Documentation

7.26.2.1 #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.

7.26.2.2 #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.

7.26.3 Function Documentation

7.26.3.1 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

nx	The size of the x dimension of the 3d matrix
ny	The size of the y dimension of the 3d matrix
nz	The size of the z dimension of the 3d matrix
x	The 3d matrix to fill

Definition at line 286 of file mikpckd.c.

Here is the caller graph for this function:

7.26.3.2 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

nx	The size of the x dimension of the 3d matrix
ny	The size of the x dimension of the 3d matrix
nz	The size of the x dimension of the 3d matrix
x	The matrix to zero out

Definition at line 190 of file mikpckd.c.

Here is the caller graph for this function:

7.26.3.3 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

nx	
ny	
nz	
ipkey	
mgdisc	
numdia	
ipc	
rpc	
ac	
cc	
fc	
xf	
yf	
zf	
gxcf	
gycf	
gzcf	
a1cf	
a2cf	
a3cf	
ccf	
fcf	

Definition at line 55 of file buildAd.c.

Here is the caller graph for this function:

7.26.3.4 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

key	
nx	
ny	
nz	
ipc	
rpc	
ac	
ipcB	
rpcB	
acB	

Definition at line 54 of file buildBd.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.26.3.5 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

nx	
ny	
nz	
ipc	
rpc	
oC	
oE	
oN	

uC	
oNE	
oNW	
uE	
uW	
uN	
uS	
uNE	
uNW	
uSE	
uSW	
ipcB	
rpcB	
acB	
n	
m	
lda	

Definition at line 180 of file buildBd.c.

Here is the caller graph for this function:

7.26.3.6 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	

oE	
oN	
uC	
ipcB	
rpcB	
acB	
n	
m	
lda	

Definition at line 116 of file buildBd.c.

Here is the caller graph for this function:

7.26.3.7 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

nxf	
nyf	
nzf	
nxc	
nyc	
nzc	
numdia	
pcFF	
acFF	
ac	

Definition at line 52 of file buildGd.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.26.3.8 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

nxf	
nyf	
nzf	
nx	
ny	
nz	
oPC	
oPN	
oPS	
oPE	
oPW	
oPNE	
oPNW	
oPSE	
oPSW	
uPC	
uPN	
uPS	
uPE	
uPW	

uPNE	
uPNW	
uPSE	
uPSW	
dPC	
dPN	
dPS	
dPE	
dPW	
dPNE	
dPNW	
dPSE	
dPSW	
oC	
XoC	
XoE	
XoN	
XuC	
XoNE	
XoNW	
XuE	
XuW	
XuN	
XuS	
XuNE	
XuNW	
XuSE	
XuSW	

Definition at line 140 of file buildGd.c.

Here is the caller graph for this function:

7.26.3.9 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

nxf	
nyf	
nzf	
nx	
ny	
nz	
oPC	
oPN	
oPS	

oPE	
oPW	
oPNE	
oPNW	
oPSE	
oPSW	
uPC	
uPN	
uPS	
uPE	
uPW	
uPNE	
uPNW	
uPSE	
uPSW	
dPC	
dPN	
dPS	
dPE	
dPW	
dPNE	
dPNW	
dPSE	
dPSW	
oC	
oE	
oN	
uC	
oNE	
oNW	
uE	
uW	
uN	
uS	
uNE	
uNW	
uSE	
uSW	
XoC	
XoE	
XoN	
XuC	
XoNE	
XoNW	
XuE	
XuW	
XuN	
XuS	
XuNE	
XuNW	
XuSE	

XuSW	

Definition at line 1247 of file buildGd.c.

Here is the caller graph for this function:

7.26.3.10 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 ∗ *XuW,* double ∗ *XuN,* double ∗ *XuS,* double ∗ *XuNE,* double ∗ *XuNW,* double ∗ *XuSE,* double ∗ *XuSW*)

Computes a 27-point galerkin coarse grid matrix from a 7-point fine grid matrix.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces buildG_7 from buildGd.f

Expressions for the galerkin coarse grid stencil XA in terms of the fine grid matrix stencil A and the interpolation operator stencil P. these stencils have the form:

XA := array([

matrix([[-XdNW(i,j,k), -XdN(i,j,k), -XdNE(i,j,k)], [-XdW(i,j,k), -XdC(i,j,k), -XdE(i,j,k)], [-XdSW(i,j,k), -XdS(i,j,k), -XdS-E(i,j,k)]]),

matrix([[-XoNW(i,j,k), -XoN(i,j,k), -XoNE(i,j,k)], [-XoW(i,j,k), XoC(i,j,k), -XoE(i,j,k)], [-XoSW(i,j,k), -XoS(i,j,k), -XoS-E(i,j,k)]]),

matrix([[-XuNW(i,j,k), -XuN(i,j,k), -XuNE(i,j,k)], [-XuW(i,j,k), -XuC(i,j,k), -XuE(i,j,k)], [-XuSW(i,j,k), -XuS(i,j,k), -XuS-E(i,j,k)]])]):

A := array([

matrix([[0, 0, 0], [0, -dC(i,j,k), 0], [0, 0, 0]]),

matrix([[0, -oN(i,j,k), 0], [-oW(i,j,k), oC(i,j,k), -oE(i,j,k)], [0, -oS(i,j,k), 0]]),

matrix([[0, 0, 0], [0, -uC(i,j,k), 0], [0, 0, 0]])

P := array([

matrix([[dPNW(i,j,k), dPN(i,j,k), dPNE(i,j,k)], [dPW(i,j,k), dPC(i,j,k), dPE(i,j,k)], [dPSW(i,j,k), dPS(i,j,k), dPSE(i,j,k)]]),

matrix([[oPNW(i,j,k), oPN(i,j,k), oPNE(i,j,k)], [oPW(i,j,k), oPC(i,j,k), oPE(i,j,k)], [oPSW(i,j,k), oPS(i,j,k), oPSE(i,j,k)]]),

matrix([[uPNW(i,j,k), uPN(i,j,k), uPNE(i,j,k)], [uPW(i,j,k), uPC(i,j,k), uPE(i,j,k)], [uPSW(i,j,k), uPS(i,j,k), uPSE(i,j,k)]])]):

in addition, A is assumed to be symmetric so that:

oS := proc(x,y,z) RETURN(oN(x,y-1,z)): end: oW := proc(x,y,z) RETURN(oE(x-1,y,z)): end: dC := proc(x,y,z) RETURN(uC(x,y,z-1)): end:

Parameters

nxf	
nyf	
nzf	
nx	
ny	
nz	
oPC	
oPN	
oPS	
oPE	
oPW	
oPNE	
oPNW	
oPSE	
oPSW	
uPC	
uPN	
uPS	
uPE	
uPW	
uPNE	
uPNW	
uPSE	
uPSW	
dPC	
dPN	
dPS	
dPE	
dPW	
dPNE	
dPNW	
dPSE	
dPSW	
oC	
oE	
oN	
uC	
XoC	
XoE	
XoN	
XuC	
XoNE	
XoNW	

XuE	
XuW	
XuN	
XuS	
XuNE	
XuNW	
XuSE	
XuSW	

Definition at line 445 of file buildGd.c.

Here is the caller graph for this function:

7.26.3.11 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

nxf	
nyf	
nzf	
nxc	
nyc	
nzc	
ipkey	
numdia	
pcFF	

ipcFF	
rpcFF	
acFF	
ccFF	
fcFF	
ipc	
rpc	
ac	
cc	
fc	

Definition at line 336 of file mgsubd.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.26.3.12 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 ∗ *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.

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

nx	
ny	
nz	
nlev	
ipkey	
iinfo	
ido	
iz	
mgprol	
mgcoar	
mgsolv	
mgdisc	
ipc	
rpc	
pc	
ac	
cc	
fc	
xf	
yf	
zf	
gxcf	
gycf	
gzcf	
a1cf	
a2cf	
a3cf	
ccf	
fcf	
tcf	

Definition at line 52 of file mgsubd.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.26.3.13 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

nxf	
nyf	
nzf	
nxc	
nyc	
nzc	
mgprol	
ipc	
rpc	
pc	
ac	
xf	
yf	
zf	

Definition at line 53 of file buildPd.c.

Here is the caller graph for this function:

7.26.3.14 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

nx	
ny	
nz	
nlev	
iz	

Definition at line 252 of file mgsubd.c.

Here is the caller graph for this function:

7.26.3.15 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

coef	
uin	
uout	
nx	
ny	
nz	
ipkey	

Definition at line 116 of file mypdec.c.

Here is the call graph for this function:

7.26.3.16 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

coef	
uin	
uout	
nx	
ny	
nz	
ipkey	

Definition at line 128 of file mypdec.c.

Here is the caller graph for this function:

7.26.3.17 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

coef	
uin	
uout	
nx	
ny	
nz	
ipkey	

Definition at line 208 of file mypdec.c.

Here is the caller graph for this function:

7.26.3.18 VPUBLIC void Vcghs (int * *nx*, int * *ny*, int * *nz*, int * *ipc*, double * *rpc*, double * *ac*, double * *cc*, double * *fc*, double * *x*, double * *p*, double * *ap*, double * *r*, int * *itmax*, int * *iters*, double * *errtol*, double * *omega*, int * *iresid*, int * *iadjoint*)

A collection of useful low-level routines (timing, etc).

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

nx	
ny	
nz	
ipc	
rpc	
ac	
cc	
fc	
x	
p	
ap	
r	
itmax	
iters	
errtol	
omega	
iresid	
iadjoint	

Definition at line 52 of file cgd.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.26.3.19 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

coef	
uin	
uout	
nx	
ny	
nz	
ipkey	

Definition at line 336 of file mypdec.c.

Here is the caller graph for this function:

7.26.3.20 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 imporper subroutine arugments. 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

abd	The output from dpbco or dpbfa
lda	The leading dimension of the array abd
n	The order of the matrix a
m	The number of diagonals above the main diagonal
b	The right hand side vector

Definition at line 53 of file mlinpckd.c.

Here is the caller graph for this function:

7.26.3.21 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

nxf	
nyf	
nzf	
nxc	
ny	
nzc	
xin	
xout	

Definition at line 1074 of file matvecd.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.26.3.22 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

ibound	
nx	
ny	
nz	
x	
gxc	
gyc	
gzc	

Definition at line 204 of file mikpckd.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.26.3.23 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

nx	The size of the x dimension of the 3d matrix
ny	The size of the y dimension of the 3d matrix
nz	The size of the z dimension of the 3d matrix
x	The 3d matrix to initialize

Definition at line 253 of file mikpckd.c.

Here is the caller graph for this function:

7.26.3.24 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

nx	
ny	
nz	
x	
iz	
w0	
w1	
w2	
w3	
w4	
istop	
itmax	
iters	
ierror	
nlev	
ilev	
nlev_real	
mgsolv	
iok	
iinfo	
epsiln	
errtol	
omega	
nu1	
nu2	
mgsmoo	
ipc	
rpc	
pc	
ac	
cc	
fc	
tru	

Definition at line 52 of file mgfasd.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.26.3.25 VPUBLIC void Vfnewton (int * *nx*, int * *ny*, int * *nz*, double * *x*, int * *iz*, double * *w0*, double * *w1*, double * *w2*, double * *w3*, int * *istop*, int * *itmax*, int * *iters*, int * *ierror*, int * *nlev*, int * *ilev*, int * *nlev_real*, int * *mgsolv*, int * *iok*, int * *iinfo*, double * *epsiln*, double * *errtol*, double * *omega*, int * *nu1*, int * *nu2*, int * *mgsmoo*, double * *cprime*, double * *rhs*, double * *xtmp*, int * *ipc*, double * *rpc*, double * *pc*, double * *ac*, double * *cc*, double * *fc*, double * *tru*)

Driver routines for the Newton method.

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

nx	
ny	
nz	
x	
iz	

w0	
w1	
w2	
w3	
istop	
itmax	
iters	
ierror	
nlev	
ilev	
nlev_real	
mgsolv	
iok	
iinfo	
epsiln	
errtol	
omega	
nu1	
nu2	
mgsmoo	
cprime	
rhs	
xtmp	
ipc	
rpc	
pc	
ac	
cc	
fc	
tru	

Definition at line 58 of file newtond.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.26.3.26 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

nx	
ny	
nz	
nlev_real	
iz	
lev	
ipkey	
x	
r	
cprime	
rhs	
cc	
pc	

Definition at line 550 of file newtond.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.26.3.27 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 * *iresid*, 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 gsrb 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

nx	
ny	
nz	
ipc	
rpc	
ac	
cc	
fc	
x	
w1	
w2	
r	
itmax	
iters	
errtol	
omega	
iresid	
iadjoint	

Definition at line 52 of file gsd.c.

Here is the caller graph for this function:

7.26.3.28 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

nxc	
nyc	
nzc	
nxf	
nyf	
nzf	
xin	
xout	
pc	

Definition at line 911 of file matvecd.c.

Here is the caller graph for this function:

7.26.3.29 VEXTERNC void Vipower (int * *nx*, int * *ny*, int * *nz*, double * *u*, int * *iz*, double * *w0*, double * *w1*, double * *w2*, double * *w3*, double * *w4*, double * *eigmin*, double * *eigmin_model*, double * *tol*, int * *itmax*, int * *iters*, int * *nlev*, int * *ilev*, int * *nlev_real*, int * *mgsolv*, int * *iok*, int * *iinfo*, double * *epsiln*, double * *errtol*, double * *omega*, int * *nu1*, int * *nu2*, int * *mgsmoo*, 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

nx	
ny	
nz	
u	
iz	
w0	
w1	
w2	
w3	
w4	
eigmin	
eigmin_model	
tol	
itmax	
iters	
nlev	
ilev	
nlev_real	
mgsolv	
iok	
iinfo	
epsiln	
errtol	
omega	
nu1	
nu2	
mgsmoo	
ipc	
rpc	
pc	
ac	
cc	
tru	

Definition at line 160 of file powerd.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.26.3.30 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

nx	
ny	
nz	
ipc	
rpc	
ac	
cc	
x	
y	

Definition at line 52 of file matvecd.c.

Here is the caller graph for this function:

7.26.3.31 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 mgdriv from mgdrvd.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

iparm	
rparm	
iwork	
rwork	
u	
xf	
yf	
zf	
gxcf	
gycf	
gzcf	
a1cf	
a2cf	
a3cf	
ccf	
fcf	
tcf	

Definition at line 52 of file mgdrvd.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.26.3.32 VEXTERNC void Vmgdriv2 (int * *iparm,* double * *rparm,* int * *nx,* int * *ny,* int * *nz,* double * *u,* int * *iz,* int * *ipc,* double * *rpc,* double * *pc,* double * *ac,* double * *cc,* double * *fc,* double * *xf,* double * *yf,* double * *zf,* double * *gxcf,* double * *gycf,* double * *gzcf,* double * *a1cf,* double * *a2cf,* double * *a3cf,* double * *ccf,* double * *fcf,* double * *tcf*)

Solves the pde using the multi-grid method.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Replaces mgdriv2 from mgdrvd.f

This routine uses a multigrid method to solve the following three-dimensional, 2nd order elliptic partial differential equation:

```
lu = f, u in omega
 u = g, u on boundary of omega
```

where

```
omega = [xmin,xmax]x[ymin,ymax]x[zmin,zmax]
```

the multigrid code requires the operator in the form:

```
- \nabla \cdot (a \nabla u) + c(u) = f
```

with

```
a(x,y,z),f(x,y,z), scalar functions (possibly discontinuous)
on 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

iparm	
rparm	
nx	
ny	
nz	
u	
iz	
ipc	
rpc	
pc	
ac	
cc	
fc	
xf	
yf	
zf	
gxcf	
gycf	
gzcf	
a1cf	
a2cf	
a3cf	
ccf	
fcf	
tcf	

Definition at line 185 of file mgdrvd.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.26.3.33 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 mgdrvd.f

Parameters

mgcoar	
mgdisc	
mgsolv	
nx	
ny	
nz	
nlev	
nxc	
nyc	
nzc	
nf	
nc	
narr	
narrc	
n_rpc	
n_iz	
n_ipc	
iretot	
iintot	

Definition at line 557 of file mgdrvd.c.

Here is the caller graph for this function:

7.26.3.34 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

nx	
ny	
nz	
ipc	
rpc	
ac	
cc	
fc	
x	
r	

Definition at line 422 of file matvecd.c.

Here is the caller graph for this function:

7.26.3.35 VEXTERNC void Vmvcs (int * *nx*, int * *ny*, int * *nz*, double * *x*, int * *iz*, double * *w0*, double * *w1*, double * *w2*, double * *w3*, int * *istop*, int * *itmax*, int * *iters*, int * *ierror*, int * *nlev*, int * *ilev*, int * *nlev_real*, int * *mgsolv*, int * *iok*, int * *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 = 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 = 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 reveresed; 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

nx	
ny	
nz	
x	
iz	
w0	
w1	
w2	
w3	
istop	
itmax	
iters	
ierror	
nlev	
ilev	
nlev_real	
mgsolv	
iok	
iinfo	
epsiln	
errtol	
omega	
nu1	
nu2	
mgsmoo	
ipc	
rpc	

pc	
ac	
cc	
fc	
tru	

Definition at line 53 of file mgcsd.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.26.3.36 VEXTERNC void Vmvfas (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.

Note

Replaces mvfas from mgfasd.f

Author

Tucker Beck [C Translation], Michael Holst [Original]

Algorithm: nonlinear multigrid iteration (fas)

multigrid v-cycle solver.

input: (1) fine and coarse grid discrete nonlinear operators: L_h, L_H (2) fine grid source function: f_h (3) fine grid approximate solution: u_h

output: (1) fine grid improved solution: u_h

the two-grid algorithm is: (1) pre-smooth: u1_h = smooth(L_h,f_h,u_h) (2) restrict defect: d_H = r(L_h(u1_h) - f_h) restrict solution: u_H = r(u1_h) (3) form coarse grid rhs: f_H = L_H(u_H) - d_H solve for correction: c_H = L_H^{-1}(f_H) (4) prolongate and correct: u2_h = u1_h - p(c_H - u_H) (5) post-smooth: u_h = smooth(L_h,f_h,u2_h)

(of course, c_H is determined with another two-grid algorithm)

implementation notes: (0) "u1_h" and "u_H" must be kept on each level until "c_H" is computed, and then all three are used to compute "u2_h". (1) "u_h" (and then "u1_h") on all levels is stored in the "x" array. (2) "u_H" on all levels is stored in the "e" array. (3) "c_h" is identically "u_h" for u_h on the next coarser grid. (4) "d_H" is stored in the "r" array. (5) "f_h" and "f_H" are stored in the "fc" array. (6) "L_h" on all levels is stored in the "ac" array. (7) signs may be reveresed; i.e., residual is used in place of the defect in places, etc.

Parameters

nx	
ny	
nz	
x	
iz	
w0	
w1	

w2	
w3	
w4	
istop	
itmax	
iters	
ierror	
nlev	
ilev	
nlev_real	
mgsolv	
iok	
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:

7.26.3.37 VPUBLIC void Vmypdefinitlpbe (int * *tnion,* double * *tcharge,* double * *tsconc*)

Set up the ionic species to be used in later calculations. This must be called before any other of the routines in this file.

Author

Tucker Beck [C Translation], Nathan Baker [Original]

Note

Replaces mypdefinitlpbe 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

tnion	The number if ionic species
tcharge	The charge in electrons
tsconc	Prefactor for conterion Bolzmann distribution terms. Basically a scaled concentration -(ion concentration/bulkIonicStrength)/2

Definition at line 52 of file mypdec.c.

Here is the caller graph for this function:

7.26.3.38 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

tnion	The number if ionic species
tcharge	The charge in electrons
tsconc	Prefactor for conterion Bolzmann distribution terms. Basically a scaled concentration -(ion concentration/bulkIonicStrength)/2

Definition at line 70 of file mypdec.c.

Here is the caller graph for this function:

7.26.3.39 VEXTERNC void Vmypdefinitsmpbe (int * *tnion*, double * *tcharge*, double * *tsconc*, double * *smvolume*, double * *smsize*)

Set up the ionic species to be used in later calculations. This must be called before any other of the routines in this file.

Author

Tucker Beck [C Translation], Nathan Baker [Original]

Note

Replaces mypdefinitsmpbe from mypde.f

Parameters

tnion	The number if ionic species
tcharge	The charge in electrons
tsconc	Prefactor for conterion Bolzmann distribution terms. Basically a scaled concentration -(ion concentration/bulkIonicStrength)/2
smvolume	
smsize	

Definition at line 88 of file mypdec.c.

Here is the caller graph for this function:

7.26.3.40 VEXTERNC void Vnewdriv (int * *iparm,* double * *rparm,* int * *iwork,* double * *rwork,* double * *u,* double * *xf,* double * *yf,* double * *zf,* double * *gxcf,* double * *gycf,* double * *gzcf,* double * *a1cf,* double * *a2cf,* double * *a3cf,* double * *ccf,* double * *fcf,* double * *tcf*)

Driver for the Newton Solver.

Author

Tucker Beck [fortran ->c translation], Michael Holst [original]

Version

$Id:

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory,
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Driver for a screaming inexact-newton-multilevel solver.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces newdriv from newdrvd.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

iparm	
rparm	
iwork	
rwork	
u	
xf	

yf	
zf	
gxcf	
gycf	
gzcf	
a1cf	
a2cf	
a3cf	
ccf	
fcf	
tcf	

Definition at line 52 of file newdrvd.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.26.3.41 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 newdriv2 from newdrvd.f

Parameters

iparm	
rparm	
nx	
ny	
nz	
u	
iz	
w1	
w2	
ipc	
rpc	
pc	
ac	
cc	
fc	
xf	
yf	
zf	
gxcf	
gycf	
gzcf	
a1cf	
a2cf	
a3cf	
ccf	
fcf	
tcf	

Definition at line 164 of file newdrvd.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.26.3.42 VEXTERNC void Vnewton (int * *nx*, int * *ny*, int * *nz*, double * *x*, int * *iz*, double * *w0*, double * *w1*, double * *w2*, double * *w3*, int * *istop*, int * *itmax*, int * *iters*, int * *ierror*, int * *nlev*, int * *ilev*, int * *nlev_real*, int * *mgsolv*, int * *iok*, int * *iinfo*, double * *epsiln*, double * *errtol*, double * *omega*, int * *nu1*, int * *nu2*, int * *mgsmoo*, double * *cprime*, double * *rhs*, double * *xtmp*, int * *ipc*, double * *rpc*, double * *pc*, double * *ac*, double * *cc*, double * *fc*, double * *tru*)

Inexact-newton-multilevel method.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces newton from newtond.f

Parameters

nx	
ny	
nz	
x	
iz	
w0	
w1	
w2	
w3	
istop	
itmax	
iters	
ierror	
nlev	
ilev	
nlev_real	
mgsolv	
iok	
iinfo	
epsiln	
errtol	
omega	

nu1	
nu2	
mgsmoo	
cprime	
rhs	
xtmp	
ipc	
rpc	
pc	
ac	
cc	
fc	
tru	

Definition at line 162 of file newtond.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.26.3.43 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

nx	
ny	
nz	
ipc	
rpc	
ac	
cc	
x	
y	
w1	

Definition at line 228 of file matvecd.c.

Here is the caller graph for this function:

7.26.3.44 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

nx	
ny	
nz	
ipc	
rpc	
ac	
cc	
fc	
x	
r	
w1	

Definition at line 594 of file matvecd.c.

Here is the caller graph for this function:

7.26.3.45 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

nx	
ny	
nz	
ipc	
rpc	
ac	
cc	
fc	
x	
w1	
w2	
r	
itmax	
iters	
errtol	
omega	
iresid	
iadjoint	
meth	

Definition at line 94 of file smoothd.c.

Here is the caller graph for this function:

7.26.3.46 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

nx	
ny	
nz	
iz	
ilev	
ipc	
rpc	
ac	
cc	
w1	
w2	
w3	
w4	
eigmax	
eigmax_model	
tol	
itmax	
iters	
iinfo	

Definition at line 52 of file powerd.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.26.3.47 VEXTERNC void Vprtmatd (int * *nx*, int * *ny*, int * *nz*, int * *ipc*, double * *rpc*, double * *ac*)

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces prtmatd from mikpckd.f

Parameters

nx	The size of the x dimension of the 3d matrix
ny	The size of the y dimension of the 3d matrix
nz	The size of the z dimension of the 3d matrix
ipc	Integer parameters
rpc	Double parameters
ac	

Definition at line 327 of file mikpckd.c.

Here is the caller graph for this function:

7.26.3.48 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

nxf	
nyf	
nzf	
nxc	
nyc	
nzc	
xin	
xout	
pc	

Definition at line 778 of file matvecd.c.

Here is the caller graph for this function:

7.26.3.49 VEXTERNC void Vsmooth (int * *nx*, int * *ny*, int * *nz*, int * *ipc*, double * *rpc*, double * *ac*, double * *cc*, double * *fc*, double * *x*, double * *w1*, double * *w2*, double * *r*, int * *itmax*, int * *iters*, double * *errtol*, double * *omega*, int * *iresid*, int * *iadjoint*, 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@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory,
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
```

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

call the appropriate linear smoothing routine.

Author

Tucker Beck [C Translation], Michael Holst [Original]

Note

Replaces smooth from smoothd.f

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

nx	
ny	
nz	
ipc	
rpc	
ac	
cc	
fc	
x	
w1	
w2	
r	
itmax	
iters	
errtol	
omega	
iresid	
iadjoint	
meth	

Definition at line 54 of file smoothd.c.

Here is the call graph for this function:

Here is the caller graph for this function:

7.26.3.50 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

nx	The size of the x dimension of the 3d matrix
ny	The size of the y dimension of the 3d matrix
nz	The size of the z dimension of the 3d matrix
alpha	
x	The source matrix from which to copy data
y	The destination matrix to receive copied data

Definition at line 107 of file mikpckd.c.

Here is the caller graph for this function:

7.26.3.51 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

nx	The size of the x dimension of the 3d matrix
ny	The size of the y dimension of the 3d matrix
nz	The size of the z dimension of the 3d matrix
x	The source matrix from which to copy data
y	The destination matrix to receive copied data

Definition at line 52 of file mikpckd.c.

Here is the caller graph for this function:

7.26.3.52 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

nx	The size of the x dimension of the 3d matrix
ny	The size of the y dimension of the 3d matrix
nz	The size of the z dimension of the 3d matrix
x	The source matrix from which to copy data
y	The destination matrix to receive copied data

Definition at line 86 of file mikpckd.c.

Here is the caller graph for this function:

7.26.3.53 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

nx	The size of the x dimension of the 3d matrix
ny	The size of the y dimension of the 3d matrix
nz	The size of the z dimension of the 3d matrix
x	The source matrix from which to copy data
y	The destination matrix to receive copied data

Definition at line 70 of file mikpckd.c.

Here is the caller graph for this function:

7.26.3.54 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

nx	The size of the x dimension of the 3d matrix
ny	The size of the y dimension of the 3d matrix
nz	The size of the z dimension of the 3d matrix
x	The first vector
y	The second vector

Definition at line 168 of file mikpckd.c.

Here is the caller graph for this function:

7.26.3.55 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

nx	The size of the x dimension of the 3d matrix
ny	The size of the y dimension of the 3d matrix
nz	The size of the z dimension of the 3d matrix
x	The matrix to normalize

Definition at line 126 of file mikpckd.c.

Here is the caller graph for this function:

7.26.3.56 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

nx	The size of the x dimension of the 3d matrix
ny	The size of the y dimension of the 3d matrix
nz	The size of the z dimension of the 3d matrix
x	The matrix to normalize

Definition at line 147 of file mikpckd.c.

Here is the caller graph for this function:

7.26.3.57 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

nx	The size of the x dimension of the 3d matrix
ny	The size of the y dimension of the 3d matrix
nz	The size of the z dimension of the 3d matrix
fac	The scaling factor
x	The 3d matrix to scale

Definition at line 313 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/APB-
S/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/A^2)

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 refinment 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 refinment 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 refinment 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 setgcent
- double cglen [3]
- int setcglen
- double fglen [3]
- int setfglen
- MGparm_CentMeth ccmeth
- double ccenter [3]
- int ccentmol

- int setcgcent
- MGparm_CentMeth fcmeth
- double fcenter [3]
- int fcentmol
- int setfgcent
- double partDisjCenter [3]
- double partDisjLength [3]
- int partDisjOwnSide [6]
- int pdime [3]
- int setpdime
- int proc_rank
- int setrank
- int proc_size
- int setsize
- double ofrac
- int setofrac
- int async
- int setasync
- int nonlintype
- int setnonlintype
- int method
- int setmethod
- int useAqua
- int setUseAqua

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 fcentmol

Particular molecule on which we want to center the grid. This should be the appropriate index in an array of molecules, not the positive definite integer specified by the user.

Definition at line 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

cglen

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/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 calcualtions. This can really confuse operations which work on specific calculations further down the road (like PRINT). Therefore this array is the initial point of entry for any calculation-specific operation. It points to a specific entry in the calc array.

Definition at line 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 VhaI_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/A^2)

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 (xmin - xmax)/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 thee->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/APB-
S/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 Laplacia

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/APB-
S/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

Multiresoltion 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

Multiresoltion 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 hiearchy. Our convention will be to have the finest grid first, however, this will not be enforced as it may be useful to search multiple grids for parallel datasets, etc.

Definition at line 87 of file vmgrid.h.

8.18.2.2 int ngrids

Number of grids in hiearchy

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 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

8.19.1 Detailed Description

Electrostatic potential oracle for Cartesian mesh data.

Author

Nathan Baker

Definition at line 82 of file vopot.h.

8.19.2 Field Documentation

8.19.2.1 Vbcfl bcfl

Boundary condition flag for returning potential values at points off the grid.

Definition at line 87 of file vopot.h.

8.19.2.2 Vmgrid* mgrid

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

Definition at line 84 of file vopot.h.

8.19.2.3 Vpbe* pbe

Pointer to PBE object

Definition at line 86 of file vopot.h.

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

- src/mg/apbs/vopot.h

8.20 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

8.20.1 Detailed Description

AtomData sub-class; stores atom data.

Author

Nathan Baker

Note

The epsilon and radius members of this class refer use the following formula for calculating the van der Waals energy of atom i interacting with atom j:

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

where $\varepsilon_{ij} = \sqrt{\varepsilon_i \varepsilon_j}$ is the well-depth (in the desired energy units), r_{ij} is the distance between atoms i and j, and $\sigma_{ij} = \sigma_i + \sigma_j$ is the sum of the van der Waals radii.

Definition at line 87 of file vparam.h.

8.20.2 Field Documentation

8.20.2.1 char atomName[VMAX_ARGLEN]

Atom name

Definition at line 88 of file vparam.h.

8.20.2.2 double charge

Atom charge (in e)

Definition at line 90 of file vparam.h.

8.20.2.3 double epsilon

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

Definition at line 92 of file vparam.h.

8.20.2.4 double radius

Atom VdW radius (σ_i above; in Å)

Definition at line 91 of file vparam.h.

8.20.2.5 char resName[VMAX_ARGLEN]

Residue name

Definition at line 89 of file vparam.h.

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

- src/generic/apbs/vparam.h

8.21 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 maxIonRadius
- int numIon
- double ionConc [MAXION]
- double ionRadii [MAXION]
- double ionQ [MAXION]
- double xkappa
- double deblen
- double zkappa2
- double zmagic
- double soluteCenter [3]
- double soluteRadius
- double soluteXlen
- double soluteYlen
- double soluteZlen
- double soluteCharge
- double smvolume
- double smsize
- int ipkey
- int paramFlag
- double z_mem
- double L
- double membraneDiel
- double V
- int param2Flag

8.21.1 Detailed Description

Contains public data members for Vpbe class/module.

Author

Nathan Baker

Definition at line 84 of file vpbe.h.

8.21.2 Field Documentation

8.21.2.1 Vacc* acc

Accessibility object

Definition at line 90 of file vpbe.h.

8.21.2.2 Valist* alist

Atom (charge) list

Definition at line 88 of file vpbe.h.

8.21.2.3 double bulkIonicStrength

Bulk ionic strength (M)

Definition at line 99 of file vpbe.h.

8.21.2.4 Vclist* clist

Atom location cell list

Definition at line 89 of file vpbe.h.

8.21.2.5 double deblen

Debye length (bulk)

Definition at line 109 of file vpbe.h.

8.21.2.6 double ionConc[MAXION]

Concentration (M) of each species

Definition at line 104 of file vpbe.h.

8.21.2.7 double ionQ[MAXION]

Charge (e) of each species

Definition at line 106 of file vpbe.h.

8.21.2.8 double ionRadii[MAXION]

Ionic radius (A) of each species

Definition at line 105 of file vpbe.h.

8.21.2.9 int ipkey

PBE calculation type (this is a cached copy it should not be used directly in code)

Definition at line 122 of file vpbe.h.

8.21.2.10 double L

Length of the membrane (A)

Definition at line 132 of file vpbe.h.

8.21.2.11 double maxIonRadius

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

Definition at line 100 of file vpbe.h.

8.21.2.12 double membraneDiel

Membrane dielectric constant

Definition at line 133 of file vpbe.h.

8.21.2.13 int numIon

Total number of ion species

Definition at line 103 of file vpbe.h.

8.21.2.14 int param2Flag

Check to see if bcfl=3 parms have been set

Definition at line 135 of file vpbe.h.

8.21.2.15 int paramFlag

Check to see if the parameters have been set

Definition at line 125 of file vpbe.h.

8.21.2.16 double smsize

Size-Modified PBE size

Definition at line 121 of file vpbe.h.

8.21.2.17 double smvolume

Size-Modified PBE relative volume

Definition at line 120 of file vpbe.h.

8.21.2.18 double soluteCenter[3]

Center of solute molecule (A)

Definition at line 113 of file vpbe.h.

8.21.2.19 double soluteCharge

Charge of solute molecule (e)

Definition at line 118 of file vpbe.h.

8.21.2.20 double soluteDiel

Solute dielectric constant (unitless)

Definition at line 93 of file vpbe.h.

8.21.2.21 double soluteRadius

Radius of solute molecule (A)

Definition at line 114 of file vpbe.h.

8.21.2.22 double soluteXlen

Solute length in x-direction

Definition at line 115 of file vpbe.h.

8.21.2.23 double soluteYlen

Solute length in y-direction

Definition at line 116 of file vpbe.h.

8.21.2.24 double soluteZlen

Solute length in z-direction

Definition at line 117 of file vpbe.h.

8.21.2.25 double solventDiel

Solvent dielectric constant (unitless)

Definition at line 94 of file vpbe.h.

8.21.2.26 double solventRadius

Solvent probe radius (angstroms) for accessibility; determining defining volumes for the dielectric coefficient

Definition at line 96 of file vpbe.h.

8.21.2.27 double T

Temperature (K)

Definition at line 92 of file vpbe.h.

8.21.2.28 double V

Membrane potential

Definition at line 134 of file vpbe.h.

8.21.2.29 Vmem* vmem

Memory management object

Definition at line 86 of file vpbe.h.

8.21.2.30 double xkappa

Debye-Huckel parameter (bulk)

Definition at line 108 of file vpbe.h.

8.21.2.31 double z_mem

Z value of the botton of the membrane (A)

Definition at line 131 of file vpbe.h.

8.21.2.32 double zkappa2

Square of modified Debye-Huckel parameter (bulk)

Definition at line 110 of file vpbe.h.

8.21.2.33 double zmagic

Delta function scaling parameter

Definition at line 111 of file vpbe.h.

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

- src/generic/apbs/vpbe.h

8.22 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

- Gem * gm
- int localPartID
- double localPartCenter [3]
- double localPartRadius
- int killFlag
- double killParam
- Vmem * mem

8.22.1 Detailed Description

Contains public data members for Vpee class/module.

Author

Nathan Baker

Definition at line 88 of file vpee.h.

8.22.2 Field Documentation

8.22.2.1 Gem* gm

Grid manager

Definition at line 90 of file vpee.h.

8.22.2.2 int killFlag

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

Definition at line 98 of file vpee.h.

8.22.2.3 double killParam

A parameter for the error estimate attenuation method

Definition at line 101 of file vpee.h.

8.22.2.4 double localPartCenter[3]

The coordinates of the center of the local partition

Definition at line 94 of file vpee.h.

8.22.2.5 int localPartID

The local partition ID: i.e. the partition whose boundary simplices we're keeping track of

Definition at line 91 of file vpee.h.

8.22.2.6 double localPartRadius

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

Definition at line 96 of file vpee.h.

8.22.2.7 Vmem* mem

Memory manager

Definition at line 103 of file vpee.h.

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

- src/fem/apbs/vpee.h

8.23 sVpmg Struct Reference

Contains public data members for Vpmg class/module.

```
#include <C:/Users/bake113/Documents/Projects/APBS & PDB2PQR/Source code/APB-
S/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

8.23.1 Detailed Description

Contains public data members for Vpmg class/module.

Author

Nathan Baker Many of the routines and macros are borrowed from the main.c driver (written by Mike Holst) provided with the PMG code.

Definition at line 120 of file vpmg.h.

8.23.2 Field Documentation

8.23.2.1 double* a1cf

Operator coefficient values (a11) – this array can be overwritten

Definition at line 142 of file vpmg.h.

8.23.2.2 double* a2cf

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

Definition at line 144 of file vpmg.h.

8.23.2.3 double* a3cf

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

Definition at line 146 of file vpmg.h.

8.23.2.4 double* ccf

Helmholtz term – this array can be overwritten

Definition at line 148 of file vpmg.h.

8.23.2.5 double* charge

Charge map

Definition at line 136 of file vpmg.h.

8.23.2.6 Vgrid* chargeMap

External charge distribution map

Definition at line 192 of file vpmg.h.

8.23.2.7 Vchrg_Meth chargeMeth

Charge discretization method

Definition at line 169 of file vpmg.h.

8.23.2.8 Vchrg_Src chargeSrc

Charge source

Definition at line 170 of file vpmg.h.

8.23.2.9 Vgrid* dielXMap

External x-shifted dielectric map

Definition at line 176 of file vpmg.h.

8.23.2.10 Vgrid* dielYMap

External y-shifted dielectric map

Definition at line 179 of file vpmg.h.

8.23.2.11 Vgrid* dielZMap

External z-shifted dielectric map

Definition at line 182 of file vpmg.h.

8.23.2.12 double* epsx

X-shifted dielectric map

Definition at line 131 of file vpmg.h.

8.23.2.13 double* epsy

Y-shifted dielectric map

Definition at line 132 of file vpmg.h.

8.23.2.14 double* epsz

Y-shifted dielectric map

Definition at line 133 of file vpmg.h.

8.23.2.15 double extDiEnergy

Stores contributions to the dielectric energy from regions outside the problem domain

Definition at line 159 of file vpmg.h.

8.23.2.16 double extNpEnergy

Stores contributions to the apolar energy from regions outside the problem domain

Definition at line 165 of file vpmg.h.

8.23.2.17 double extQfEnergy

Stores contributions to the fixed charge energy from regions outside the problem domain

Definition at line 163 of file vpmg.h.

8.23.2.18 double extQmEnergy

Stores contributions to the mobile ion energy from regions outside the problem domain

Definition at line 161 of file vpmg.h.

8.23.2.19 double* fcf

Right-hand side – this array can be overwritten

Definition at line 149 of file vpmg.h.

8.23.2.20 int filled

Indicates whether Vpmg_fillco has been called

Definition at line 172 of file vpmg.h.

8.23.2.21 double* gxcf

Boundary conditions for x faces

Definition at line 155 of file vpmg.h.

8.23.2.22 double* gycf

Boundary conditions for y faces

Definition at line 156 of file vpmg.h.

8.23.2.23 double* gzcf

Boundary conditions for z faces

Definition at line 157 of file vpmg.h.

8.23.2.24 int* iparm

Passing int parameters to FORTRAN

Definition at line 138 of file vpmg.h.

8.23.2.25 int* iwork

Work array

Definition at line 140 of file vpmg.h.

8.23.2.26 double* kappa

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

Definition at line 134 of file vpmg.h.

8.23.2.27 Vgrid* kappaMap

External kappa map

Definition at line 185 of file vpmg.h.

8.23.2.28 Vpbe* pbe

Information about the PBE system

Definition at line 124 of file vpmg.h.

8.23.2.29 Vpmgp* pmgp

Parameters

Definition at line 123 of file vpmg.h.

8.23.2.30 double* pot

Potential map

Definition at line 135 of file vpmg.h.

8.23.2.31 Vgrid* potMap

External potential map

Definition at line 188 of file vpmg.h.

8.23.2.32 double* pvec

Partition mask array

Definition at line 158 of file vpmg.h.

8.23.2.33 double* rparm

Passing real parameters to FORTRAN

Definition at line 139 of file vpmg.h.

8.23.2.34 double* rwork

Work array

Definition at line 141 of file vpmg.h.

8.23.2.35 double splineWin

Spline window parm for surf defs

Definition at line 168 of file vpmg.h.

8.23.2.36 Vsurf_Meth surfMeth

Surface definition method

Definition at line 167 of file vpmg.h.

8.23.2.37 double* tcf

True solution

Definition at line 150 of file vpmg.h.

8.23.2.38 double* u

Solution

Definition at line 151 of file vpmg.h.

8.23.2.39 int useChargeMap

Indicates whether Vpmg_fillco was called with an external charge distribution map

Definition at line 190 of file vpmg.h.

8.23.2.40 int useDielXMap

Indicates whether Vpmg_fillco was called with an external x-shifted dielectric map

Definition at line 174 of file vpmg.h.

8.23.2.41 int useDielYMap

Indicates whether Vpmg_fillco was called with an external y-shifted dielectric map

Definition at line 177 of file vpmg.h.

8.23.2.42 int useDielZMap

Indicates whether Vpmg_fillco was called with an external z-shifted dielectric map

Definition at line 180 of file vpmg.h.

8.23.2.43 int useKappaMap

Indicates whether Vpmg_fillco was called with an external kappa map

Definition at line 183 of file vpmg.h.

8.23.2.44 int usePotMap

Indicates whether Vpmg_fillco was called with an external potential map

Definition at line 186 of file vpmg.h.

8.23.2.45 Vmem* vmem

Memory management object for this class

Definition at line 122 of file vpmg.h.

8.23.2.46 double* xf

Mesh point x coordinates

Definition at line 152 of file vpmg.h.

8.23.2.47 double* yf

Mesh point y coordinates

Definition at line 153 of file vpmg.h.

8.23.2.48 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

8.24 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

8.24.1 Detailed Description

Contains public data members for Vpmgp class/module.

Author

Nathan Baker

Bug Value ipcon does not currently allow for preconditioning in PMG

Definition at line 78 of file vpmgp.h.

8.24.2 Field Documentation

8.24.2.1 Vbcfl bcfl

Boundary condition method [default = BCFL_SDH]

Definition at line 133 of file vpmgp.h.

8.24.2.2 double errtol

Desired error tolerance [default = 1e-9]

Definition at line 119 of file vpmgp.h.

8.24.2.3 double hx

Grid x spacings [no default]

Definition at line 85 of file vpmgp.h.

8.24.2.4 double hy

Grid y spacings [no default]

Definition at line 86 of file vpmgp.h.

8.24.2.5 double hzed

Grid z spacings [no default]

Definition at line 87 of file vpmgp.h.

8.24.2.6 int iinfo

Runtime status messages [default = 1]

- 0: none
- 1: some
- 2: lots
- 3: more

Definition at line 128 of file vpmgp.h.

8.24.2.7 int ipcon

Preconditioning method [default = 3]

- 0: diagonal
- 1: ICCG

- 2: ICCGDW
- 3: MICCGDW
- 4: none

Definition at line 181 of file vpmgp.h.

8.24.2.8 int iperf

Analysis of the operator [default = 0]

- 0: no
- 1: condition number
- 2: spectral radius
- 3: cond. number & spectral radius

Definition at line 137 of file vpmgp.h.

8.24.2.9 int ipkey

Toggles nonlinearity (set by nonlin)

- -2: Size-Modified PBE
- -1: Linearized PBE
- 0: Nonlinear PBE with capped sinh term [default]
- >1: Polynomial approximation to sinh, note that ipkey must be odd

Definition at line 107 of file vpmgp.h.

8.24.2.10 int irite

FORTRAN output unit [default = 8]

Definition at line 180 of file vpmgp.h.

8.24.2.11 int istop

Stopping criterion [default = 1]

- 0: residual
- 1: relative residual
- 2: diff
- 3: errc
- 4: errd
- 5: aerrd

Definition at line 121 of file vpmgp.h.

8.24.2.12 int itmax

Maximum number of iters [default = 100]

Definition at line 120 of file vpmgp.h.

8.24.2.13 int key

Print solution to file [default = 0]

- 0: no
- 1: yes

Definition at line 134 of file vpmgp.h.

8.24.2.14 int meth

Solution method [default = 2]

- 0: conjugate gradient multigrid
- 1: newton
- 2: multigrid
- 3: conjugate gradient
- 4: sucessive overrelaxation
- 5: red-black gauss-seidel
- 6: weighted jacobi
- 7: richardson
- 8: conjugate gradient multigrid aqua
- 9: newton aqua

Definition at line 142 of file vpmgp.h.

8.24.2.15 int mgcoar

Coarsening method [default = 2]

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

Definition at line 168 of file vpmgp.h.

8.24.2.16 int mgdisc

Discretization method [default = 0]

- 0: finite volume
- 1: finite element

Definition at line 175 of file vpmgp.h.

8.24.2.17 int mgkey

Multigrid method [default = 0]

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

Definition at line 153 of file vpmgp.h.

8.24.2.18 int mgprol

Prolongation method [default = 0]

- 0: trilinear
- 1: operator-based
- 2: mod. operator-based

Definition at line 164 of file vpmgp.h.

8.24.2.19 int mgsmoo

Smoothing method [default = 1]

- 0: weighted jacobi
- 1: gauss-seidel
- 2: SOR
- 3: richardson
- 4: cghs

Definition at line 158 of file vpmgp.h.

8.24.2.20 int mgsolv

Coarse equation solve method [default = 1]

- 0: cghs
- 1: banded linpack

Definition at line 172 of file vpmgp.h.

8.24.2.21 int n_ipc

Integer info work array required storage

Definition at line 102 of file vpmgp.h.

8.24.2.22 int n_iz

Integer storage parameter (index max)

Definition at line 101 of file vpmgp.h.

8.24.2.23 int n_rpc

Real info work array required storage

Definition at line 100 of file vpmgp.h.

8.24.2.24 int narr

Array work storage

Definition at line 106 of file vpmgp.h.

8.24.2.25 int narrc

Size of vector on coarse level

Definition at line 99 of file vpmgp.h.

8.24.2.26 int nc

Number of coarse grid unknowns

Definition at line 98 of file vpmgp.h.

8.24.2.27 int nf

Number of fine grid unknowns

Definition at line 97 of file vpmgp.h.

8.24.2.28 int niwk

Integer work storage

Definition at line 105 of file vpmgp.h.

8.24.2.29 int nlev

Number of mesh levels [no default]

Definition at line 84 of file vpmgp.h.

8.24.2.30 int nonlin

Problem type [no default]

- 0: linear
- 1: nonlinear
- 2: linear then nonlinear

Definition at line 88 of file vpmgp.h.

8.24.2.31 int nrwk

Real work storage

Definition at line 104 of file vpmgp.h.

8.24.2.32 int nu1

Number of pre-smoothings [default = 2]

Definition at line 156 of file vpmgp.h.

8.24.2.33 int nu2

Number of post-smoothings [default = 2]

Definition at line 157 of file vpmgp.h.

8.24.2.34 int nx

Grid x dimensions [no default]

Definition at line 81 of file vpmgp.h.

8.24.2.35 int nxc

Coarse level grid x dimensions

Definition at line 94 of file vpmgp.h.

8.24.2.36 int ny

Grid y dimensions [no default]

Definition at line 82 of file vpmgp.h.

8.24.2.37 int nyc

Coarse level grid y dimensions

Definition at line 95 of file vpmgp.h.

8.24.2.38 int nz

Grid z dimensions [no default]

Definition at line 83 of file vpmgp.h.

8.24.2.39 int nzc

Coarse level grid z dimensions

Definition at line 96 of file vpmgp.h.

8.24.2.40 double omegal

Linear relax parameter [default = 8e-1]

Definition at line 178 of file vpmgp.h.

8.24.2.41 double omegan

Nonlin relax parameter [default = 9e-1]

Definition at line 179 of file vpmgp.h.

8.24.2.42 double xcent

Grid x center [0]

Definition at line 116 of file vpmgp.h.

8.24.2.43 double xlen

Domain x length

Definition at line 187 of file vpmgp.h.

8.24.2.44 double xmax

Domain upper x corner

Definition at line 193 of file vpmgp.h.

8.24.2.45 double xmin

Domain lower x corner

Definition at line 190 of file vpmgp.h.

8.24.2.46 double ycent

Grid y center [0]

Definition at line 117 of file vpmgp.h.

8.24.2.47 double ylen

Domain y length

Definition at line 188 of file vpmgp.h.

8.24.2.48 double ymax

Domain upper y corner

Definition at line 194 of file vpmgp.h.

8.24.2.49 double ymin

Domain lower y corner

Definition at line 191 of file vpmgp.h.

8.24.2.50 double zcent

Grid z center [0]

Definition at line 118 of file vpmgp.h.

8.24.2.51 double zlen

Domain z length

Definition at line 189 of file vpmgp.h.

8.24.2.52 double zmax

Domain upper z corner

Definition at line 195 of file vpmgp.h.

8.24.2.53 double zmin

Domain lower z corner

Definition at line 192 of file vpmgp.h.

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

- src/mg/apbs/vpmgp.h

8.25 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

8.25.1 Detailed Description

Reads and assigns charge/radii parameters.

Author

Nathan Baker

Definition at line 130 of file vparam.h.

8.25.2 Field Documentation

8.25.2.1 int nResData

Number of Vparam_ResData objects associated with this object

Definition at line 133 of file vparam.h.

8.25.2.2 Vparam_ResData* resData

Array of nResData Vparam_ResData objects

Definition at line 135 of file vparam.h.

8.25.2.3 Vmem* vmem

Memory management object for this class

Definition at line 132 of file vparam.h.

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

- src/generic/apbs/vparam.h

8.26 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

8.26.1 Detailed Description

ResData sub-class; stores residue data.

Author

Nathan Baker

Definition at line 109 of file vparam.h.

8.26.2 Field Documentation

8.26.2.1 Vparam_AtomData* atomData

Array of Vparam_AtomData natom objects

Definition at line 114 of file vparam.h.

8.26.2.2 char name[VMAX_ARGLEN]

Residue name

Definition at line 111 of file vparam.h.

8.26.2.3 int nAtomData

Number of Vparam_AtomData objects associated with this object

Definition at line 112 of file vparam.h.

8.26.2.4 Vmem* vmem

Pointer to memory manager from Vparam master class

Definition at line 110 of file vparam.h.

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

- 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/maloc.h"
#include "apbs/femparm.h"
#include "apbs/mgparm.h"
#include "apbs/nosh.h"
#include "apbs/pbeparm.h"
#include "apbs/vacc.h"
#include "apbs/valist.h"
#include "apbs/vatom.h"
#include "apbs/vcap.h"
#include "apbs/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 1774 2012-07-19 18:46:14Z sobolevnrm

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
```

```
/* MALOC headers */
#include "maloc/maloc.h"

/* Generic headers */
#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"

/* MG headers */
#include "apbs/vgrid.h"
#include "apbs/vmgrid.h"
#include "apbs/vopot.h"
#include "apbs/vpmg.h"
#include "apbs/vpmgp.h"

/* FEM headers */
#if defined(HAVE_MC_H)
#include "apbs/vfetk.h"
#include "apbs/vpee.h"
#endif

#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

```
#include "apbscfg.h"

#if defined(HAVE_MC_H)
    void apbs_needs_mc(void) { }
#endif
#if !defined(USE_PMG_BLAS)
    void apbs_needs_blas(void) { }
#endif

void apbs_link(void)
{
}

```

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)

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.

9.7.1 Detailed Description

Contains declarations for the Vcsm class.

Version

Id:

vcsm.h 1776 2012-07-19 22:29:10Z 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 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 /* ////////////////////////////////////////////////////////////////////////
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(
```

```
            Vcsm *thee,
            int isimp
            );

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

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

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

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

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

    VEXTERNC unsigned long int Vcsm_memChk(
            Vcsm *thee
            );

#else /* if defined(VINLINE_VCSM) */
#   define Vcsm_getValist(thee) ((thee)->alist)
#   define Vcsm_getNumberAtoms(thee, isimp) ((thee)->nsqm[isimp])
#   define Vcsm_getAtom(thee, iatom, isimp) (Valist_getAtom((thee)->alist,
 ((thee)->sqm)[isimp][iatom]))
#   define Vcsm_getAtomIndex(thee, iatom, isimp) (((thee)->sqm)[isimp][iatom])
#   define Vcsm_getNumberSimplices(thee, iatom) (((thee)->nqsm)[iatom])
#   define Vcsm_getSimplex(thee, isimp, iatom) (Gem_SS((thee)->gm,
 ((thee)->qsm)[iatom][isimp]))
#   define Vcsm_getSimplexIndex(thee, isimp, iatom)
 (((thee)->qsm)[iatom][isimp])
#   define Vcsm_memChk(thee) (Vmem_bytes((thee)->vmem))
#endif /* if !defined(VINLINE_VCSM) */

/* ////////////////////////////////////////////////////////////////////////////
// Class Vcsm: Non-Inlineable methods (vcsm.c)

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

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

VEXTERNC void Vcsm_dtor(
        Vcsm **thee
        );

VEXTERNC void Vcsm_dtor2(
        Vcsm *thee
        );

VEXTERNC void Vcsm_init(
        Vcsm *thee
        );

VEXTERNC int Vcsm_update(
        Vcsm *thee,
        SS **simps,
        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:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct sVfetk

  Contains public data members for Vfetk class/module.
- struct sVfetk_LocalVar

  Vfetk LocalVar subclass.

Typedefs

- typedef enum eVfetk_LsolvType Vfetk_LsolvType

  Declare FEMparm_LsolvType type.
- typedef enum eVfetk_MeshLoad Vfetk_MeshLoad

  Declare FEMparm_GuessType type.
- typedef enum eVfetk_NsolvType Vfetk_NsolvType

  Declare FEMparm_NsolvType type.
- typedef enum eVfetk_GuessType Vfetk_GuessType

  Declare FEMparm_GuessType type.
- typedef enum eVfetk_PrecType Vfetk_PrecType

  Declare FEMparm_GuessType type.
- typedef struct sVfetk Vfetk

  Declaration of the Vfetk class as the Vfetk structure.
- typedef struct sVfetk_LocalVar Vfetk_LocalVar

  Declaration of the Vfetk_LocalVar subclass as the Vfetk_LocalVar structure.

Enumerations

- enum eVfetk_LsolvType { VLT_SLU = 0, VLT_MG = 1, VLT_CG = 2, VLT_BCG = 3 }

  Linear solver type.
- enum eVfetk_MeshLoad { VML_DIRICUBE, VML_NEUMCUBE, VML_EXTERNAL }

  Mesh loading operation.
- enum eVfetk_NsolvType { VNT_NEW = 0, VNT_INC = 1, VNT_ARC = 2 }

  Non-linear solver type.
- enum eVfetk_GuessType { VGT_ZERO = 0, VGT_DIRI = 1, VGT_PREV = 2 }

  Initial guess type.
- enum eVfetk_PrecType { VPT_IDEN = 0, VPT_DIAG = 1, VPT_MG = 2 }

  Preconditioner type.

Functions

- VEXTERNC Gem ∗ Vfetk_getGem (Vfetk ∗thee)

  Get a pointer to the Gem (grid manager) object.
- VEXTERNC AM ∗ Vfetk_getAM (Vfetk ∗thee)

  Get a pointer to the AM (algebra manager) object.
- VEXTERNC Vpbe ∗ Vfetk_getVpbe (Vfetk ∗thee)

  Get a pointer to the Vpbe (PBE manager) object.
- VEXTERNC Vcsm ∗ Vfetk_getVcsm (Vfetk ∗thee)

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

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

  Get the partition information for a particular atom.
- VEXTERNC Vfetk * Vfetk_ctor (Vpbe *pbe, Vhal_PBEType type)

  Constructor for Vfetk object.
- VEXTERNC int Vfetk_ctor2 (Vfetk *thee, Vpbe *pbe, Vhal_PBEType type)

  FORTRAN stub constructor for Vfetk object.
- VEXTERNC void Vfetk_dtor (Vfetk **thee)

  Object destructor.
- VEXTERNC void Vfetk_dtor2 (Vfetk *thee)

  FORTRAN stub object destructor.
- VEXTERNC double * Vfetk_getSolution (Vfetk *thee, int *length)

  Create an array containing the solution (electrostatic potential in units of k_BT/e) at the finest mesh level.
- VEXTERNC void Vfetk_setParameters (Vfetk *thee, PBEparm *pbeparm, FEMparm *feparm)

  Set the parameter objects.
- VEXTERNC double Vfetk_energy (Vfetk *thee, int color, int nonlin)

  Return the total electrostatic energy.
- VEXTERNC double Vfetk_dqmEnergy (Vfetk *thee, int color)

  Get the "mobile charge" and "polarization" contributions to the electrostatic energy.
- VEXTERNC double Vfetk_qfEnergy (Vfetk *thee, int color)

  Get the "fixed charge" contribution to the electrostatic energy.
- VEXTERNC unsigned long int Vfetk_memChk (Vfetk *thee)

  Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC void Vfetk_setAtomColors (Vfetk *thee)

  Transfer color (partition ID) information frmo a partitioned mesh to the atoms.
- VEXTERNC void Bmat_printHB (Bmat *thee, char *fname)

  Writes a Bmat to disk in Harwell-Boeing sparse matrix format.
- VEXTERNC Vrc_Codes Vfetk_genCube (Vfetk *thee, double center[3], double length[3], Vfetk_MeshLoad meshType)

  Construct a rectangular mesh (in the current Vfetk object)
- VEXTERNC Vrc_Codes Vfetk_loadMesh (Vfetk *thee, double center[3], double length[3], Vfetk_MeshLoad meshType, Vio *sock)

  Loads a mesh into the Vfetk (and associated) object(s).
- VEXTERNC PDE * Vfetk_PDE_ctor (Vfetk *fetk)

  Constructs the FEtk PDE object.
- VEXTERNC int Vfetk_PDE_ctor2 (PDE *thee, Vfetk *fetk)

  Intializes the FEtk PDE object.
- VEXTERNC void Vfetk_PDE_dtor (PDE **thee)

  Destroys FEtk PDE object.
- VEXTERNC void Vfetk_PDE_dtor2 (PDE *thee)

  FORTRAN stub: destroys FEtk PDE object.
- VEXTERNC void Vfetk_PDE_initAssemble (PDE *thee, int ip[], double rp[])

  Do once-per-assembly initialization.
- VEXTERNC void Vfetk_PDE_initElement (PDE *thee, int elementType, int chart, double tvx[][VAPBS_DIM], void *data)

  Do once-per-element initialization.
- VEXTERNC void Vfetk_PDE_initFace (PDE *thee, int faceType, int chart, double tnvec[])

  Do once-per-face initialization.

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

  Do once-per-point initialization.
- VEXTERNC void Vfetk_PDE_Fu (PDE *thee, int key, double F[])

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

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

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

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

  where $n(x)$ is the normal to the simplex face and the term represents the jump in dielectric displacement across the face. There is no outer-boundary contribution for this problem.
- VEXTERNC double Vfetk_PDE_Fu_v (PDE *thee, int key, double V[], double dV[][VAPBS_DIM])

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

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

  where $b(u)$ denotes the mobile ion term.
- VEXTERNC double Vfetk_PDE_DFu_wv (PDE *thee, int key, double W[], double dW[][VAPBS_DIM], double V[], double dV[][VAPBS_DIM])

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

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

  where $b'(u)$ denotes the functional derivation of the mobile ion term.
- VEXTERNC void Vfetk_PDE_delta (PDE *thee, int type, int chart, double txq[], void *user, double F[])

  Evaluate a (discretized) delta function source term at the given point.
- VEXTERNC void Vfetk_PDE_u_D (PDE *thee, int type, int chart, double txq[], double F[])

  Evaluate the Dirichlet boundary condition at the given point.
- VEXTERNC void Vfetk_PDE_u_T (PDE *thee, int type, int chart, double txq[], double F[])

  Evaluate the "true solution" at the given point for comparison with the numerical solution.
- VEXTERNC void Vfetk_PDE_bisectEdge (int dim, int dimII, int edgeType, int chart[], double vx[][VAPBS_DIM])

  Define the way manifold edges are bisected.
- VEXTERNC void Vfetk_PDE_mapBoundary (int dim, int dimII, int vertexType, int chart, double vx[VAPBS_DIM])

  Map a boundary point to some pre-defined shape.
- VEXTERNC int Vfetk_PDE_markSimplex (int dim, int dimII, int simplexType, int faceType[VAPBS_NVS], int vertexType[VAPBS_NVS], int chart[], double vx[][VAPBS_DIM], void *simplex)

  User-defined error estimator – in our case, a geometry-based refinement method; forcing simplex refinement at the dielectric boundary and (for non-regularized PBE) the charges.
- VEXTERNC void Vfetk_PDE_oneChart (int dim, int dimII, int objType, int chart[], double vx[][VAPBS_DIM], int dimV)

  Unify the chart for different coordinate systems – a no-op for us.
- VEXTERNC double Vfetk_PDE_Ju (PDE *thee, int key)

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

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

  for a 1:1 electrolyte where c is the output from Vpbe_getZmagic.
- VEXTERNC void Vfetk_externalUpdateFunction (SS **simps, int num)

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

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

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

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

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

- VEXTERNC void Vfetk_readMesh (Vfetk ∗thee, int skey, Vio ∗sock)

  Read in mesh and initialize associated internal structures.

- VEXTERNC void Vfetk_dumpLocalVar ()

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

- VEXTERNC int Vfetk_fillArray (Vfetk ∗thee, Bvec ∗vec, Vdata_Type type)

  Fill an array with the specified data.

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

  Write out data.

- VEXTERNC Vrc_Codes Vfetk_loadGem (Vfetk ∗thee, Gem ∗gm)

  Load a Gem geometry manager object into Vfetk.

9.9.1 Detailed Description

Contains declarations for class Vfetk.

Version

Id:

vfetk.h 1776 2012-07-19 22:29:10Z 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 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,
```

```
    VNT_ARC=2
};

typedef enum eVfetk_NsolvType Vfetk_NsolvType;

enum eVfetk_GuessType {
    VGT_ZERO=0,
    VGT_DIRI=1,
    VGT_PREV=2
};

typedef enum eVfetk_GuessType Vfetk_GuessType;

enum eVfetk_PrecType {
    VPT_IDEN=0,
    VPT_DIAG=1,
    VPT_MG=2
};

typedef enum eVfetk_PrecType Vfetk_PrecType;

struct sVfetk {

  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;
};

typedef struct sVfetk Vfetk;

struct sVfetk_LocalVar {
    double nvec[VAPBS_DIM];
    double vx[4][VAPBS_DIM];
    double xq[VAPBS_DIM];
    double U[MAXV];
    double dU[MAXV][VAPBS_DIM];
    double W;
    double dW[VAPBS_DIM];
    double d2W;
    int sType;
    int fType;
    double diel;
    double ionacc;
    double A;
    double F;
    double B;
    double DB;
    double jumpDiel;
    Vfetk *fetk;
    Vgreen *green;
    int initGreen;
    SS *simp;
    VV *verts[4];
    int nverts;
    double ionConc[MAXION];
    double ionQ[MAXION];
    double ionRadii[MAXION];
    double zkappa2;
    double zks2;
    double ionstr;
    int nion;
    double Fu_v;
    double DFu_wv;
    double delta;
```

```
    double u_D;
    double u_T;
};

typedef struct sVfetk_LocalVar Vfetk_LocalVar;

#if !defined(VINLINE_VFETK)

    VEXTERNC Gem* Vfetk_getGem(
            Vfetk *thee
            );

    VEXTERNC AM* Vfetk_getAM(
            Vfetk *thee
            );

    VEXTERNC Vpbe* Vfetk_getVpbe(
            Vfetk *thee
            );

    VEXTERNC Vcsm* Vfetk_getVcsm(
            Vfetk *thee
            );

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

#else /* if defined(VINLINE_VFETK) */
#   define Vfetk_getGem(thee) ((thee)->gm)
#   define Vfetk_getAM(thee) ((thee)->am)
#   define Vfetk_getVpbe(thee) ((thee)->pbe)
#   define Vfetk_getVcsm(thee) ((thee)->csm)
#   define Vfetk_getAtomColor(thee, iatom)
 (Vatom_getPartID(Valist_getAtom(Vpbe_getValist(thee->pbe), iatom)))
#endif /* if !defined(VINLINE_VFETK) */

/* ///////////////////////////////////////////////////////////////////////////
// Class Vfetk: Non-Inlineable methods (vfetk.c)

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

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

VEXTERNC void Vfetk_dtor(
        Vfetk **thee
        );

VEXTERNC void Vfetk_dtor2(
        Vfetk *thee
        );

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

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

VEXTERNC double  Vfetk_energy(
        Vfetk *thee,
        int color,
        int nonlin
        );

VEXTERNC double  Vfetk_dqmEnergy(
        Vfetk *thee,
        int color
        );
```

```
VEXTERNC double Vfetk_qfEnergy(
        Vfetk *thee,
        int color
        );

VEXTERNC unsigned long int Vfetk_memChk(
        Vfetk *thee
        );

VEXTERNC void Vfetk_setAtomColors(
        Vfetk *thee
        );

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

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

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

VEXTERNC PDE* Vfetk_PDE_ctor(
        Vfetk *fetk
        );

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

VEXTERNC void Vfetk_PDE_dtor(
        PDE **thee
        );

VEXTERNC void Vfetk_PDE_dtor2(
        PDE *thee
        );

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

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

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

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

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

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

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

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

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

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

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

VEXTERNC void Vfetk_PDE_mapBoundary(
        int dim,
        int dimII,
        int vertexType,
        int chart,
        double vx[VAPBS_DIM]
        );

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
        );

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

```
        );

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

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


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

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

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

VEXTERNC void Vfetk_dumpLocalVar();

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

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

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


#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 1776 2012-07-19 22:29:10Z 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 vpee.h.

9.12 vpee.h

```
00001
00076 #ifndef _VPEE_H
00077 #define _VPEE_H
00078
00079 /* Generic headers */
00080 #include "maloc/maloc.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
```

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

VEXTERNC int Vpee_numSS(
        Vpee *thee
        );

#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 1776 2012-07-19 22:29:10Z 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 dummy.c.

9.14 dummy.c

```
00001
00056 #include "apbscfg.h"
00057
00058 int APBSFEM_dummy(int i) {
00059
00060     int j;
00061
00062     j = i;
00063
00064     return j;
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 1776 2012-07-19 22:29:10Z 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 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
      isimp) {
00082
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][
      iatom]);
00089
00090 }
00091
00092 VPUBLIC int Vcsm_getAtomIndex(Vcsm *thee, int iatom, int
      isimp) {
00093
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
```

```
      iatom) {

   VASSERT(thee != VNULL);
   VASSERT(thee->initFlag);

   return (thee->nqsm)[iatom];

}

VPUBLIC SS* Vcsm_getSimplex(Vcsm *thee, int isimp, int iatom
      ) {


   VASSERT(thee != VNULL);
   VASSERT(thee->initFlag);

   return Gem_SS(thee->gm, (thee->qsm)[iatom][isimp]);

}

VPUBLIC int Vcsm_getSimplexIndex(Vcsm *thee, int isimp,
       int iatom) {


   VASSERT(thee != VNULL);
   VASSERT(thee->initFlag);

   return (thee->qsm)[iatom][isimp];

}

VPUBLIC unsigned long int Vcsm_memChk(Vcsm *thee) {
    if (thee == VNULL) return 0;
    return Vmem_bytes(thee->vmem);
}

#endif /* if !defined(VINLINE_VCSM) */

VPUBLIC Vcsm* Vcsm_ctor(Valist *alist, Gem *gm) {

    /* Set up the structure */
    Vcsm *thee = VNULL;
    thee = (Vcsm*)Vmem_malloc(VNULL, 1, sizeof(Vcsm) );
    VASSERT( thee != VNULL);
    VASSERT( Vcsm_ctor2(thee, alist, gm));

    return thee;
}

VPUBLIC int Vcsm_ctor2(Vcsm *thee, Valist *alist, Gem *gm
      ) {

    VASSERT( thee != VNULL );

    /* Memory management object */
    thee->vmem = Vmem_ctor("APBS:VCSM");

    /* Set up the atom list and grid manager */
    if( alist == VNULL) {
        Vnm_print(2,"Vcsm_ctor2: got null pointer to Valist object!\n");
        return 0;
    }
    thee->alist = alist;
    if( gm == VNULL) {
        Vnm_print(2,"Vcsm_ctor2: got a null pointer to the Gem object!\n");
        return 0;
    }
    thee->gm = gm;

    thee->initFlag = 0;
    return 1;
}

VPUBLIC void Vcsm_init(Vcsm *thee) {

    /* Counters */
    int iatom,
        jatom,
        isimp,
        jsimp,
```

```
        gotSimp;
    /* Atomic information */
    Vatom *atom;
    double *position;
    /* Simplex/Vertex information */
    SS *simplex;
    /* Basis function values */

    if (thee == VNULL) {
        Vnm_print(2, "Vcsm_init:  Error!  Got NULL thee!\n");
        VASSERT(0);
    }
    if (thee->gm == VNULL) {
        Vnm_print(2, "Vcsm_init:  Error!  Got NULL thee->gm!\n");
        VASSERT(0);
    }
    thee->nsimp = Gem_numSS(thee->gm);
    if (thee->nsimp <= 0) {
        Vnm_print(2, "Vcsm_init:  Error!  Got %d simplices!\n", thee->nsimp);
        VASSERT(0);
    }
    thee->natom = Valist_getNumberAtoms(thee->alist);

    /* Allocate and initialize space for the first dimensions of the
     * simplex-charge map, the simplex array, and the counters */
    thee->sqm = (int**)Vmem_malloc(thee->vmem, thee->nsimp, sizeof(int *));
    VASSERT(thee->sqm != VNULL);
    thee->nsqm = (int*)Vmem_malloc(thee->vmem, thee->nsimp, sizeof(int));
    VASSERT(thee->nsqm != VNULL);
    for (isimp=0; isimp<thee->nsimp; isimp++) (thee->nsqm)[isimp] = 0;

    /* Count the number of charges per simplex. */
    for (iatom=0; iatom<thee->natom; iatom++) {
        atom = Valist_getAtom(thee->alist, iatom);
        position = Vatom_getPosition(atom);
        gotSimp = 0;
        for (isimp=0; isimp<thee->nsimp; isimp++) {
            simplex = Gem_SS(thee->gm, isimp);
            if (Gem_pointInSimplex(thee->gm, simplex, position)) {
                (thee->nsqm)[isimp]++;
                gotSimp = 1;
             }
        }
    }

    /* @todo Combine the following two loops? - PCE */
    /* Allocate the space for the simplex-charge map */
    for (isimp=0; isimp<thee->nsimp; isimp++) {
        if ((thee->nsqm)[isimp] > 0) {
            thee->sqm[isimp] = (int*)Vmem_malloc(thee->vmem, (thee->nsqm)[isimp],
              sizeof(int));
            VASSERT(thee->sqm[isimp] != VNULL);
        }
    }

    /* Finally, set up the map */
    for (isimp=0; isimp<thee->nsimp; isimp++) {
        jsimp = 0;
        simplex = Gem_SS(thee->gm, isimp);
        for (iatom=0; iatom<thee->natom; iatom++) {
            atom = Valist_getAtom(thee->alist, iatom);
            position = Vatom_getPosition(atom);
            /* Check to see if the atom's in this simplex */
            if (Gem_pointInSimplex(thee->gm, simplex, position)) {
                /* Assign the entries in the next vacant spot */
                (thee->sqm)[isimp][jsimp] = iatom;
                jsimp++;
            }
        }
    }

    thee->msimp = thee->nsimp;

    /* Allocate space for the charge-simplex map */
    thee->qsm = (int**)Vmem_malloc(thee->vmem, thee->natom, sizeof(int *));
```

```
    VASSERT(thee->qsm != VNULL);
    thee->nqsm = (int*)Vmem_malloc(thee->vmem, thee->natom, sizeof(int));
    VASSERT(thee->nqsm != VNULL);
    for (iatom=0; iatom<thee->natom; iatom++) (thee->nqsm)[iatom] = 0;
    /* Loop through the list of simplices and count the number of times
     * each atom appears */
    for (isimp=0; isimp<thee->nsimp; isimp++) {
        for (iatom=0; iatom<thee->nsqm[isimp]; iatom++) {
            jatom = thee->sqm[isimp][iatom];
            thee->nqsm[jatom]++;
        }
    }
    /* Do a TIME-CONSUMING SANITY CHECK to make sure that each atom was
     * placed in at simplex */
    for (iatom=0; iatom<thee->natom; iatom++) {
        if (thee->nqsm[iatom] == 0) {
            Vnm_print(2, "Vcsm_init: Atom %d not placed in simplex!\n", iatom);
            VASSERT(0);
        }
    }
    /* Allocate the appropriate amount of space for each entry in the
     * charge-simplex map and clear the counter for re-use in assignment */
    for (iatom=0; iatom<thee->natom; iatom++) {
        thee->qsm[iatom] = (int*)Vmem_malloc(thee->vmem, (thee->nqsm)[iatom],
          sizeof(int));
        VASSERT(thee->qsm[iatom] != VNULL);
        thee->nqsm[iatom] = 0;
    }
    /* Assign the simplices to atoms */
    for (isimp=0; isimp<thee->nsimp; isimp++) {
        for (iatom=0; iatom<thee->nsqm[isimp]; iatom++) {
            jatom = thee->sqm[isimp][iatom];
            thee->qsm[jatom][thee->nqsm[jatom]] = isimp;
            thee->nqsm[jatom]++;
        }
    }

    thee->initFlag = 1;
}

VPUBLIC void Vcsm_dtor(Vcsm **thee) {
    if ((*thee) != VNULL) {
        Vcsm_dtor2(*thee);
        Vmem_free(VNULL, 1, sizeof(Vcsm), (void **)thee);
        (*thee) = VNULL;
    }
}

VPUBLIC void Vcsm_dtor2(Vcsm *thee) {
    int i;

    if ((thee != VNULL) && thee->initFlag) {

        for (i=0; i<thee->msimp; i++) {
            if (thee->nsqm[i] > 0) Vmem_free(thee->vmem, thee->nsqm[i],
              sizeof(int), (void **)&(thee->sqm[i]));
        }
        for (i=0; i<thee->natom; i++) {
            if (thee->nqsm[i] > 0) Vmem_free(thee->vmem, thee->nqsm[i],
              sizeof(int), (void **)&(thee->qsm[i]));
        }
        Vmem_free(thee->vmem, thee->msimp, sizeof(int *),
          (void **)&(thee->sqm));
        Vmem_free(thee->vmem, thee->msimp, sizeof(int),
          (void **)&(thee->nsqm));
        Vmem_free(thee->vmem, thee->natom, sizeof(int *),
          (void **)&(thee->qsm));
        Vmem_free(thee->vmem, thee->natom, sizeof(int),
          (void **)&(thee->nqsm));

    }
    Vmem_dtor(&(thee->vmem));
}

VPUBLIC int Vcsm_update(Vcsm *thee, SS **simps, int num) {

    /* Counters */
```

```
    int isimp, jsimp, iatom, jatom, atomID, simpID;
    int nsimps, gotMem;
    /* Object info */
    Vatom *atom;
    SS *simplex;
    double *position;
    /* Lists */
    int *qParent, nqParent;
    int **sqmNew, *nsqmNew;
    int *affAtoms, nAffAtoms;
    int *dnqsm, *nqsmNew, **qsmNew;

    VASSERT(thee != VNULL);
    VASSERT(thee->initFlag);

    /* If we don't have enough memory to accommodate the new entries,
     * add more by doubling the existing amount */
    isimp = thee->nsimp + num - 1;
    gotMem = 0;
    while (!gotMem) {
        if (isimp > thee->msimp) {
            isimp = 2 * isimp;
            thee->nsqm = (int*)Vmem_realloc(thee->vmem, thee->msimp
, sizeof(int),
              (void **)&(thee->nsqm), isimp);
            VASSERT(thee->nsqm != VNULL);
            thee->sqm = (int**)Vmem_realloc(thee->vmem, thee->msimp
, sizeof(int *),
              (void **)&(thee->sqm), isimp);
            VASSERT(thee->sqm != VNULL);
            thee->msimp = isimp;
        } else gotMem = 1;
    }
    /* Initialize the nsqm entires we just allocated */
    for (isimp = thee->nsimp; isimp<thee->nsimp+num-1 ; isimp++) {
       thee->nsqm[isimp] = 0;
    }

    thee->nsimp = thee->nsimp + num - 1;

    /* There's a simple case to deal with:  if simps[0] didn't have a
     * charge in the first place */
    isimp = SS_id(simps[0]);
    if (thee->nsqm[isimp] == 0) {
        for (isimp=1; isimp<num; isimp++) {
            thee->nsqm[SS_id(simps[isimp])] = 0;
        }
        return 1;
    }

    /* The more complicated case has occured; the parent simplex had one or
     * more charges.  First, generate the list of affected charges. */
    isimp = SS_id(simps[0]);
    nqParent = thee->nsqm[isimp];
    qParent = thee->sqm[isimp];

    sqmNew = (int**)Vmem_malloc(thee->vmem, num, sizeof(int *));
    VASSERT(sqmNew != VNULL);
    nsqmNew = (int*)Vmem_malloc(thee->vmem, num, sizeof(int));
    VASSERT(nsqmNew != VNULL);
    for (isimp=0; isimp<num; isimp++) nsqmNew[isimp] = 0;

    /* Loop throught the affected atoms to determine how many atoms each
     * simplex will get. */
    for (iatom=0; iatom<nqParent; iatom++) {

        atomID = qParent[iatom];
        atom = Valist_getAtom(thee->alist, atomID);
        position = Vatom_getPosition(atom);
        nsimps = 0;

        jsimp = 0;

        for (isimp=0; isimp<num; isimp++) {
            simplex = simps[isimp];
            if (Gem_pointInSimplex(thee->gm, simplex, position)) {
                nsqmNew[isimp]++;
                jsimp = 1;
            }
        }

```

```
        VASSERT(jsimp != 0);
    }

    /* Sanity check that we didn't lose any atoms... */
    iatom = 0;
    for (isimp=0; isimp<num; isimp++) iatom += nsqmNew[isimp];
    if (iatom < nqParent) {
        Vnm_print(2,"Vcsm_update: Lost %d (of %d) atoms!\n",
            nqParent - iatom, nqParent);
        VASSERT(0);
    }

    /* Allocate the storage */
    for (isimp=0; isimp<num; isimp++) {
        if (nsqmNew[isimp] > 0) {
            sqmNew[isimp] = (int*)Vmem_malloc(thee->vmem, nsqmNew[isimp],
              sizeof(int));
            VASSERT(sqmNew[isimp] != VNULL);
        }
    }

    /* Assign charges to simplices */
    for (isimp=0; isimp<num; isimp++) {

        jsimp = 0;
        simplex = simps[isimp];

        /* Loop over the atoms associated with the parent simplex */
        for (iatom=0; iatom<nqParent; iatom++) {

            atomID = qParent[iatom];
            atom = Valist_getAtom(thee->alist, atomID);
            position = Vatom_getPosition(atom);
            if (Gem_pointInSimplex(thee->gm, simplex, position)) {
                sqmNew[isimp][jsimp] = atomID;
                jsimp++;
            }
        }
    }

    /* Update the QSM map using the old and new SQM lists */
    /* The affected atoms are those contained in the parent simplex; i.e.
     * thee->sqm[SS_id(simps[0])] */
    affAtoms = thee->sqm[SS_id(simps[0])];
    nAffAtoms = thee->nsqm[SS_id(simps[0])];
    /* Each of these atoms will go somewhere else; i.e., the entries in
     * thee->qsm are never destroyed and thee->nqsm never decreases.
     * However, it is possible that a subdivision could cause an atom to be
     * shared by two child simplices.  Here we record the change, if any,
     * in the number of simplices associated with each atom. */
    dnqsm = (int*)Vmem_malloc(thee->vmem, nAffAtoms, sizeof(int));
    VASSERT(dnqsm != VNULL);
    nqsmNew = (int*)Vmem_malloc(thee->vmem, nAffAtoms, sizeof(int));
    VASSERT(nqsmNew != VNULL);
    qsmNew = (int**)Vmem_malloc(thee->vmem, nAffAtoms, sizeof(int*));
    VASSERT(qsmNew != VNULL);
    for (iatom=0; iatom<nAffAtoms; iatom++) {
        dnqsm[iatom] = -1;
        atomID = affAtoms[iatom];
        for (isimp=0; isimp<num; isimp++) {
            for (jatom=0; jatom<nsqmNew[isimp]; jatom++) {
                if (sqmNew[isimp][jatom] == atomID) dnqsm[iatom]++;
            }
        }
        VASSERT(dnqsm[iatom] > -1);
    }
    /* Setup the new entries in the array */
    for (iatom=0;iatom<nAffAtoms; iatom++) {
        atomID = affAtoms[iatom];
        qsmNew[iatom] = (int*)Vmem_malloc(thee->vmem,
                (dnqsm[iatom] + thee->nqsm[atomID]),
                sizeof(int));
        nqsmNew[iatom] = 0;
        VASSERT(qsmNew[iatom] != VNULL);
    }
    /* Fill the new entries in the array */
    /* First, do the modified entries */
    for (isimp=0; isimp<num; isimp++) {
        simpID = SS_id(simps[isimp]);
        for (iatom=0; iatom<nsqmNew[isimp]; iatom++) {
            atomID = sqmNew[isimp][iatom];
```

```
            for (jatom=0; jatom<nAffAtoms; jatom++) {
                if (atomID == affAtoms[jatom]) break;
            }
            if (jatom < nAffAtoms) {
                qsmNew[jatom][nqsmNew[jatom]] = simpID;
                nqsmNew[jatom]++;
            }
        }
    }
    /* Now do the unmodified entries */
    for (iatom=0; iatom<nAffAtoms; iatom++) {
        atomID = affAtoms[iatom];
        for (isimp=0; isimp<thee->nqsm[atomID]; isimp++) {
            for (jsimp=0; jsimp<num; jsimp++) {
                simpID = SS_id(simps[jsimp]);
                if (thee->qsm[atomID][isimp] == simpID) break;
            }
            if (jsimp == num) {
                qsmNew[iatom][nqsmNew[iatom]] = thee->qsm[atomID][isimp];
                nqsmNew[iatom]++;
            }
        }
    }

    /* Replace the existing entries in the table.  Do the QSM entires
     * first, since they require affAtoms = thee->sqm[simps[0]] */
    for (iatom=0; iatom<nAffAtoms; iatom++) {
        atomID = affAtoms[iatom];
        Vmem_free(thee->vmem, thee->nqsm[atomID], sizeof(int),
          (void **)&(thee->qsm[atomID]));
        thee->qsm[atomID] = qsmNew[iatom];
        thee->nqsm[atomID] = nqsmNew[iatom];
    }
    for (isimp=0; isimp<num; isimp++) {
        simpID = SS_id(simps[isimp]);
        if (thee->nsqm[simpID] > 0) Vmem_free(thee->vmem, thee->nsqm
[simpID],
          sizeof(int), (void **)&(thee->sqm[simpID]));
        thee->sqm[simpID] = sqmNew[isimp];
        thee->nsqm[simpID] = nsqmNew[isimp];
    }

    Vmem_free(thee->vmem, num, sizeof(int *), (void **)&sqmNew);
    Vmem_free(thee->vmem, num, sizeof(int), (void **)&nsqmNew);
    Vmem_free(thee->vmem, nAffAtoms, sizeof(int *), (void **)&qsmNew);
    Vmem_free(thee->vmem, nAffAtoms, sizeof(int), (void **)&nqsmNew);
    Vmem_free(thee->vmem, nAffAtoms, sizeof(int), (void **)&dnqsm);


    return 1;


}

#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, PBEparm ∗pbeparm, FEMparm ∗feparm)

  Set the parameter objects.
- VPUBLIC void Vfetk_dtor (Vfetk ∗∗thee)

  Object destructor.
- VPUBLIC void Vfetk_dtor2 (Vfetk ∗thee)

  FORTRAN stub object destructor.
- VPUBLIC double ∗ Vfetk_getSolution (Vfetk ∗thee, int ∗length)

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

- VEXTERNC double Vfetk_energy (Vfetk *thee, int color, int nonlin)

  Return the total electrostatic energy.
- VPUBLIC double Vfetk_qfEnergy (Vfetk *thee, int color)

  Get the "fixed charge" contribution to the electrostatic energy.
- VPUBLIC double Vfetk_dqmEnergy (Vfetk *thee, int color)

  Get the "mobile charge" and "polarization" contributions to the electrostatic energy.
- VPUBLIC void Vfetk_setAtomColors (Vfetk *thee)

  Transfer color (partition ID) information frmo a partitioned mesh to the atoms.
- VPUBLIC unsigned long int Vfetk_memChk (Vfetk *thee)

  Return the memory used by this structure (and its contents) in bytes.
- VPUBLIC Vrc_Codes Vfetk_genCube (Vfetk *thee, double center[3], double length[3], Vfetk_MeshLoad meshType)

  Construct a rectangular mesh (in the current Vfetk object)
- VPUBLIC Vrc_Codes Vfetk_loadMesh (Vfetk *thee, double center[3], double length[3], Vfetk_MeshLoad meshType, Vio *sock)

  Loads a mesh into the Vfetk (and associated) object(s).
- VPUBLIC void Bmat_printHB (Bmat *thee, char *fname)

  Writes a Bmat to disk in Harwell-Boeing sparse matrix format.
- VPUBLIC PDE * Vfetk_PDE_ctor (Vfetk *fetk)

  Constructs the FEtk PDE object.
- VPUBLIC int Vfetk_PDE_ctor2 (PDE *thee, Vfetk *fetk)

  Intializes the FEtk PDE object.
- VPUBLIC void Vfetk_PDE_dtor (PDE **thee)

  Destroys FEtk PDE object.
- VPUBLIC void Vfetk_PDE_dtor2 (PDE *thee)

  FORTRAN stub: destroys FEtk PDE object.
- VPUBLIC void Vfetk_PDE_initAssemble (PDE *thee, int ip[], double rp[])

  Do once-per-assembly initialization.
- VPUBLIC void **Vfetk_PDE_initElement** (PDE *thee, int elementType, int chart, double tvx[][3], void *data)
- VPUBLIC void Vfetk_PDE_initFace (PDE *thee, int faceType, int chart, double tnvec[])

  Do once-per-face initialization.
- VPUBLIC void **Vfetk_PDE_initPoint** (PDE *thee, int pointType, int chart, double txq[], double tU[], double tdU[][3])
- VPUBLIC void Vfetk_PDE_Fu (PDE *thee, int key, double F[])

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

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

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

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

  where $n(x)$ is the normal to the simplex face and the term represents the jump in dielectric displacement across the face. There is no outer-boundary contribution for this problem.
- VPUBLIC double Vfetk_PDE_Fu_v (PDE *thee, int key, double V[], double dV[][VAPBS_DIM])

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

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

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

- VPUBLIC double **Vfetk_PDE_DFu_wv** (PDE *thee, int key, double W[], double dW[][VAPBS_DIM], double V[], double dV[][3])
- VPUBLIC void Vfetk_PDE_delta (PDE *thee, int type, int chart, double txq[], void *user, double F[])

  Evaluate a (discretized) delta function source term at the given point.
- VPUBLIC void Vfetk_PDE_u_D (PDE *thee, int type, int chart, double txq[], double F[])

  Evaluate the Dirichlet boundary condition at the given point.
- VPUBLIC void Vfetk_PDE_u_T (PDE *thee, int type, int chart, double txq[], double F[])

  Evaluate the "true solution" at the given point for comparison with the numerical solution.
- VPUBLIC void **Vfetk_PDE_bisectEdge** (int dim, int dimII, int edgeType, int chart[], double vx[][3])
- VPUBLIC void **Vfetk_PDE_mapBoundary** (int dim, int dimII, int vertexType, int chart, double vx[3])
- VPUBLIC int **Vfetk_PDE_markSimplex** (int dim, int dimII, int simplexType, int faceType[VAPBS_NVS], int vertexType[VAPBS_NVS], int chart[], double vx[][3], void *simplex)
- VPUBLIC void **Vfetk_PDE_oneChart** (int dim, int dimII, int objType, int chart[], double vx[][3], int dimV)
- VPUBLIC double Vfetk_PDE_Ju (PDE *thee, int key)

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

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

  for a 1:1 electrolyte where c is the output from Vpbe_getZmagic.
- VPUBLIC void Vfetk_externalUpdateFunction (SS **simps, int num)

  External hook to simplex subdivision routines in Gem. Called each time a simplex is subdivided (we use it to update the charge-simplex map)
- VPUBLIC int Vfetk_PDE_simplexBasisInit (int key, int dim, int comp, int *ndof, int dof[])

  Initialize the bases for the trial or the test space, for a particular component of the system, at all quadrature points on the master simplex element.
- VPUBLIC void Vfetk_PDE_simplexBasisForm (int key, int dim, int comp, int pdkey, double xq[], double basis[])

  Evaluate the bases for the trial or test space, for a particular component of the system, at all quadrature points on the master simplex element.
- VPUBLIC void Vfetk_dumpLocalVar ()

  Debugging routine to print out local variables used by PDE object.
- VPUBLIC int Vfetk_fillArray (Vfetk *thee, Bvec *vec, Vdata_Type type)

  Fill an array with the specified data.
- VPUBLIC int Vfetk_write (Vfetk *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, Bvec *vec, Vdata_Format format)

  Write out data.

Variables

- VPRIVATE Vfetk_LocalVar **var**
- VPRIVATE char * **diriCubeString**
- VPRIVATE char * **neumCubeString**
- VPRIVATE int **dim_2DP1** = 3
- VPRIVATE int **lgr_2DP1** [3][VMAXP]
- VPRIVATE int **lgr_2DP1x** [3][VMAXP]
- VPRIVATE int **lgr_2DP1y** [3][VMAXP]
- VPRIVATE int **lgr_2DP1z** [3][VMAXP]
- VPRIVATE int **dim_3DP1** = VAPBS_NVS
- VPRIVATE int **lgr_3DP1** [VAPBS_NVS][VMAXP]
- VPRIVATE int **lgr_3DP1x** [VAPBS_NVS][VMAXP]
- VPRIVATE int **lgr_3DP1y** [VAPBS_NVS][VMAXP]

- VPRIVATE int **lgr_3DP1z** [VAPBS_NVS][VMAXP]
- VPRIVATE const int **P_DEG** = 1
- VPRIVATE int **numP**
- VPRIVATE double **c** [VMAXP][VMAXP]
- VPRIVATE double **cx** [VMAXP][VMAXP]
- VPRIVATE double **cy** [VMAXP][VMAXP]
- VPRIVATE double **cz** [VMAXP][VMAXP]

9.17.1 Detailed Description

Class Vfetk methods.

Author

Nathan Baker

Version

Id:

vfetk.c 1776 2012-07-19 22:29:10Z 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 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\
\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 },
{  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
}
```

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 },
{  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
}
```

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

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,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
}
```

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

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

Definition at line 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 },
{  2,  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\
];\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  Calculuate 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\
```

```
5 0  0.5 -0.5  0.5\n\
6 0 -0.5  0.5  0.5\n\
7 0  0.5  0.5  0.5\n\
];\n\
\n\
simp=[\n\
0 0 0 0 1 0 1 0 5 1 2\n\
1 0 0 0 1 1 0 0 5 2 4\n\
2 0 0 0 1 0 1 1 5 3 2\n\
3 0 0 0 1 0 1 3 5 7 2\n\
4 0 0 1 1 0 0 2 5 7 6\n\
5 0 0 1 1 0 0 2 5 6 4\n\
];\n\
\n\
mcsf_end=1;\n\
\n\
";

/*
 * @brief  MCSF-format cube mesh (all Neumann)
 * @ingroup Vfetk
 * @author  Based on mesh by Mike Holst
 */
VPRIVATE char *neumCubeString =
"mcsf_begin=1;\n\
\n\
dim=3;\n\
dimii=3;\n\
vertices=8;\n\
simplices=6;\n\
\n\
vert=[\n\
0 0 -0.5 -0.5 -0.5\n\
1 0  0.5 -0.5 -0.5\n\
2 0 -0.5  0.5 -0.5\n\
3 0  0.5  0.5 -0.5\n\
4 0 -0.5 -0.5  0.5\n\
5 0  0.5 -0.5  0.5\n\
6 0 -0.5  0.5  0.5\n\
7 0  0.5  0.5  0.5\n\
];\n\
\n\
simp=[\n\
0 0 0 0 2 0 2 0 5 1 2\n\
1 0 0 0 2 2 0 0 5 2 4\n\
2 0 0 0 2 0 2 1 5 3 2\n\
3 0 0 0 2 0 2 3 5 7 2\n\
4 0 0 2 2 0 0 2 5 7 6\n\
5 0 0 2 2 0 0 2 5 6 4\n\
];\n\
\n\
mcsf_end=1;\n\
\n\
";

/*
 * @brief  Return the smoothed value of the dielectric coefficient at the
 * current point using a fast, chart-based method
 * @ingroup  Vfetk
 * @author  Nathan Baker
 * @returns  Value of dielectric coefficient
 * @bug  Not thread-safe
 */
VPRIVATE double diel();

/*
 * @brief  Return the smoothed value of the ion accessibility at the
 * current point using a fast, chart-based method
 * @ingroup  Vfetk
 * @author  Nathan Baker
 * @returns  Value of mobile ion coefficient
 * @bug  Not thread-safe
 */
VPRIVATE double ionacc();

/*
 * @brief  Smooths a mesh-based coefficient with a simple harmonic function
 * @ingroup  Vfetk
 * @author  Nathan Baker
 * @param  meth  Method for smoothing
 *   \li  0 ==> arithmetic mean (gives bad results)
```

```
 *   \li  1 ==> geometric mean
 * @param  nverts  Number of vertices
 * @param  dist  distance from point to each vertex
 * @param  coeff  coefficient value at each vertex
 * @note  Thread-safe
 * @return smoothed value of coefficieent at point of interest */
VPRIVATE double smooth(
        int nverts,
        double dist[VAPBS_NVS],
        double coeff[VAPBS_NVS],
        int meth
        );


/*
 * @brief  Return the analytical multi-sphere Debye-Huckel approximation (in
 * kT/e) at the specified point
 * @ingroup  Vfetk
 * @author  Nathan Baker
 * @param  pbe  Vpbe object
 * @param  d  Dimension of x
 * @param  x  Coordinates of point of interest (in &Aring;)
 * @note  Thread-safe
 * @returns  Multi-sphere Debye-Huckel potential in kT/e
 */
VPRIVATE double debye_U(
        Vpbe *pbe,
        int d,
        double x[]
        );

/*
 * @brief  Return the difference between the analytical multi-sphere
 * Debye-Huckel approximation and Coulomb's law (in kT/e) at the specified
 * point
 * @ingroup  Vfetk
 * @author  Nathan Baker
 * @param  pbe  Vpbe object
 * @param  d  Dimension of x
 * @param  x  Coordinates of point of interest (in &Aring;)
 * @note  Thread-safe
 * @returns  Multi-sphere Debye-Huckel potential in kT/e */
VPRIVATE double debye_Udiff(
        Vpbe *pbe,
        int d,
        double x[]
        );

/*
 * @brief  Calculate the Coulomb's
 * Debye-Huckel approximation and Coulomb's law (in kT/e) at the specified
 * point
 * @ingroup  Vfetk
 * @author  Nathan Baker
 * @param  pbe  Vpbe object
 * @param  d  Dimension of x
 * @param  x  Coordinates of point of interest (in &Aring;)
 * @param  eps  Dielectric constant
 * @param  U  Set to potential (in kT/e)
 * @param  dU  Set to potential gradient (in kT/e/&Aring;)
 * @param  d2U  Set to Laplacian of potential (in \f$kT e^{-1} \AA^{-2}\f$)
 * @returns  Multi-sphere Debye-Huckel potential in kT/e */
VPRIVATE void coulomb(
        Vpbe *pbe,
        int d,
        double x[],
        double eps,
        double *U,
        double dU[],
        double *d2U
        );

/*
 * @brief  2D linear master simplex information generator
 * @ingroup  Vfetk
 * @author  Mike Holst
 * @param dimIS  dunno
 * @param ndof  dunno
 * @param dof  dunno
 * @param c  dunno
 * @param cx  dunno
```

```
 * @note  Trust in Mike */
VPRIVATE void init_2DP1(
        int dimIS[],
        int *ndof,
        int dof[],
        double c[][VMAXP],
        double cx[][VMAXP],
        double cy[][VMAXP],
        double cz[][VMAXP]
        );

/*
 * @brief  3D linear master simplex information generator
 * @ingroup  Vfetk
 * @author  Mike Holst
 * @param  dimIS  dunno
 * @param ndof dunno
 * @param dof dunno
 * @param c dunno
 * @param cx dunno
 * @param cy dunno
 * @param cz dunno
 * @note  Trust in Mike */
VPRIVATE void init_3DP1(
        int dimIS[],
        int *ndof,
        int dof[],
        double c[][VMAXP],
        double cx[][VMAXP],
        double cy[][VMAXP],
        double cz[][VMAXP]
        );

/*
 * @brief  Setup coefficients of polynomials from integer table data
 * @ingroup  Vfetk
 * @author  Mike Holst
 * @param numP  dunno
 * @param c  dunno
 * @param cx  dunno
 * @param cy  dunno
 * @param cz  dunno
 * @param ic  dunno
 * @param icx  dunno
 * @param icy  dunno
 * @param icz  dunno
 * @note  Trust in Mike */
VPRIVATE void setCoef(
        int numP,
        double c[][VMAXP],
        double cx[][VMAXP],
        double cy[][VMAXP],
        double cz[][VMAXP],
        int ic[][VMAXP],
        int icx[][VMAXP],
        int icy[][VMAXP],
        int icz[][VMAXP]
        );

/*
 * @brief  Evaluate a collection of at most cubic polynomials at a
 * specified point in at most R^3.
 * @ingroup  Vfetk
 * @author  Mike Holst
 * @param numP  the number of polynomials to evaluate
 * @param p  the results of the evaluation
 * @param c   the coefficients of each polynomial
 * @param xv  the point (x,y,z) to evaluate the polynomials.
 * @note  Mike says:
 * <pre>
 *  Note that "VMAXP" must be >= 19 for cubic polynomials.
 *  The polynomials are build from the coefficients c[][] as
 *  follows.  To build polynomial "k", fix k and set:
 *
 *  c0=c[k][0], c1=c[k][1], .... , cp=c[k][p]
 *
 *  Then evaluate as:
 *
 *  p3(x,y,z) = c0 + c1*x + c2*y + c3*z
 *            + c4*x*x + c5*y*y + c6*z*z + c7*x*y + c8*x*z + c9*y*z
 *            + c10*x*x*x + c11*y*y*y + c12*z*z*z
```

```
 *            + c13*x*x*y + c14*x*x*z + c15*x*y*y
 *            + c16*y*y*z + c17*x*z*z + c18*y*z*z
 * </pre>
 */
VPRIVATE void polyEval(
        int numP,
        double p[],
        double c[][VMAXP],
        double xv[]
        );

/*
 * @brief  I have no clue what this variable does, but we need it to initialize
 * the simplices
 * @ingroup  Vfetk
 * @author  Mike Holst */
VPRIVATE int dim_2DP1 = 3;

/*
 * @brief  I have no clue what these variable do, but we need it to initialize
 * the simplices
 * @ingroup  Vfetk
 * @author  Mike Holst
 * @note  Mike says:
 * <pre>
 *  2D-P1 Basis:
 *
 *  p1(x,y) = c0 + c1*x + c2*y
 *
 *  Lagrange Point    Lagrange Basis Function Definition
 *  --------------    -----------------------------------
 *  (0, 0)            p[0](x,y) = 1 - x - y
 *  (1, 0)            p[1](x,y) = x
 *  (0, 1)            p[2](x,y) = y
 *  </pre>
 */
VPRIVATE int lgr_2DP1[3][VMAXP] = {
/*c0  c1  c2  c3
* ------------------------------------------------------------ */
/* 1   x   y   z
* ------------------------------------------------------------ */
{  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 }
};
VPRIVATE int lgr_2DP1x[3][VMAXP] = {
/*c0 ------------------------------------------------------------------------ */
/* 1 ------------------------------------------------------------------------ */
{ -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
};
VPRIVATE int lgr_2DP1y[3][VMAXP] = {
/*c0 ------------------------------------------------------------------------ */
/* 1 ------------------------------------------------------------------------ */
{ -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
};
VPRIVATE int lgr_2DP1z[3][VMAXP] = {
/*c0 ------------------------------------------------------------------------ */
/* 1 ------------------------------------------------------------------------ */
{  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
};


/*
 * @brief  I have no clue what these variable do, but we need it to initialize
 * the simplices
 * @ingroup  Vfetk
 * @author  Mike Holst
 * @note  Mike says:
 * <pre>
 * 3D-P1 Basis:
 *
 * p1(x,y,z) = c0 + c1*x + c2*y + c3*z
 *
 * Lagrange Point    Lagrange Basis Function Definition
 * --------------    -----------------------------------
```

```
 * (0, 0, 0)          p[0](x,y,z) = 1 - x - y - z
 * (1, 0, 0)          p[1](x,y,z) = x
 * (0, 1, 0)          p[2](x,y,z) = y
 * (0, 0, 1)          p[3](x,y,z) = z
 * </pre>
 */
VPRIVATE int dim_3DP1 = VAPBS_NVS;
VPRIVATE int lgr_3DP1[VAPBS_NVS][VMAXP] = {
/*c0  c1  c2  c3 ------------------------------------------------------------ */
/* 1   x   y   z ------------------------------------------------------------ */
{  2, -2, -2, -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
};
VPRIVATE int lgr_3DP1x[VAPBS_NVS][VMAXP] = {
/*c0 ------------------------------------------------------------------------ */
/* 1 ------------------------------------------------------------------------ */
{ -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
};
VPRIVATE int lgr_3DP1y[VAPBS_NVS][VMAXP] = {
/*c0 ------------------------------------------------------------------------ */
/* 1 ------------------------------------------------------------------------ */
{ -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
};
VPRIVATE int lgr_3DP1z[VAPBS_NVS][VMAXP] = {
/*c0 ------------------------------------------------------------------------ */
/* 1 ------------------------------------------------------------------------ */
{ -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
};

/*
 * @brief  Another Holst variable
 * @ingroup  Vfetk
 * @author  Mike Holst
 * @note  Mike says: 1 = linear, 2 = quadratic */
VPRIVATE const int P_DEG=1;

/*
 * @brief  Another Holst variable
 * @ingroup  Vfetk
 * @author  Mike Holst */
VPRIVATE int numP;
VPRIVATE double c[VMAXP][VMAXP];
VPRIVATE double cx[VMAXP][VMAXP];
VPRIVATE double cy[VMAXP][VMAXP];
VPRIVATE double cz[VMAXP][VMAXP];

#if !defined(VINLINE_VFETK)

VPUBLIC Gem* Vfetk_getGem(Vfetk *thee) {

   VASSERT(thee != VNULL);
   return thee->gm;

}

VPUBLIC AM* Vfetk_getAM(Vfetk *thee) {

   VASSERT(thee != VNULL);
   return thee->am;
}

VPUBLIC Vpbe* Vfetk_getVpbe(Vfetk *thee) {

   VASSERT(thee != VNULL);
   return thee->pbe;

}

VPUBLIC Vcsm* Vfetk_getVcsm(Vfetk *thee) {

```

```
    VASSERT(thee != VNULL);
    return thee->csm;

}

VPUBLIC int Vfetk_getAtomColor(Vfetk *thee,
                               int iatom
                              ) {

    int natoms;

    VASSERT(thee != VNULL);

    natoms = Valist_getNumberAtoms(Vpbe_getValist
(thee->pbe));
    VASSERT(iatom < natoms);

    return Vatom_getPartID(Valist_getAtom(
Vpbe_getValist(thee->pbe), iatom));
}
#endif /* if !defined(VINLINE_VFETK) */

VPUBLIC Vfetk* Vfetk_ctor(Vpbe *pbe,
                          Vhal_PBEType type
                         ) {

    /* Set up the structure */
    Vfetk *thee = VNULL;
    thee = (Vfetk*)Vmem_malloc(VNULL, 1, sizeof(Vfetk) );
    VASSERT(thee != VNULL);
    VASSERT(Vfetk_ctor2(thee, pbe, type));

    return thee;
}

VPUBLIC int Vfetk_ctor2(Vfetk *thee,
                        Vpbe *pbe,
                        Vhal_PBEType type
                       ) {

    int i;
    double center[VAPBS_DIM];

    /* Make sure things have been properly initialized & store them */
    VASSERT(pbe != VNULL);
    thee->pbe = pbe;
    VASSERT(pbe->alist != VNULL);
    VASSERT(pbe->acc != VNULL);

    /* Store PBE type */
    thee->type = type;

    /* Set up memory management object */
    thee->vmem = Vmem_ctor("APBS::VFETK");

    /* Set up FEtk objects */
 Vnm_print(0, "Vfetk_ctor2:  Constructing PDE...\n");
    thee->pde = Vfetk_PDE_ctor(thee);
    Vnm_print(0, "Vfetk_ctor2:  Constructing Gem...\n");
    thee->gm = Gem_ctor(thee->vmem, thee->pde);
    Vnm_print(0, "Vfetk_ctor2:  Constructing Aprx...\n");
    thee->aprx = Aprx_ctor(thee->vmem, thee->gm, thee->pde);
    Vnm_print(0, "Vfetk_ctor2:  Constructing Aprx...\n");
    thee->am = AM_ctor(thee->vmem, thee->aprx);

    /* Reset refinement level */
    thee->level = 0;

    /* Set default solver variables */
    thee->lkey = VLT_MG;
    thee->lmax = 1000000;
    thee->ltol = 1e-5;
    thee->lprec = VPT_MG;
    thee->nkey = VNT_NEW;
    thee->nmax = 1000000;
    thee->ntol = 1e-5;
    thee->gues = VGT_ZERO;
    thee->pjac = -1;

    /* Store local copy of myself */
    var.fetk = thee;
```

```
    var.initGreen = 0;

 /* Set up the external Gem subdivision hook */
    Gem_setExternalUpdateFunction(thee->gm, Vfetk_externalUpdateFunction);

    /* Set up ion-related variables */
    var.zkappa2 = Vpbe_getZkappa2(var.fetk->pbe);
    var.ionstr = Vpbe_getBulkIonicStrength(var.fetk->pbe);
    if (var.ionstr > 0.0) var.zks2 = 0.5*var.zkappa2/var.ionstr;
    else var.zks2 = 0.0;
    Vpbe_getIons(var.fetk->pbe, &(var.nion), var.ionConc, var.ionRadii,
      var.ionQ);
    for (i=0; i<var.nion; i++) {
        var.ionConc[i] = var.zks2 * var.ionConc[i] * var.ionQ[i];
    }

    /* Set uninitialized objects to NULL */
    thee->pbeparm = VNULL;
    thee->feparm = VNULL;
    thee->csm = VNULL;

    return 1;
}

VPUBLIC void Vfetk_setParameters(Vfetk *thee,
                                 PBEparm *pbeparm,
                                 FEMparm *feparm
                                ) {

    VASSERT(thee != VNULL);
    thee->feparm = feparm;
    thee->pbeparm = pbeparm;
}

VPUBLIC void Vfetk_dtor(Vfetk **thee) {
    if ((*thee) != VNULL) {
        Vfetk_dtor2(*thee);
        //Vmem_free(VNULL, 1, sizeof(Vfetk), (void **)thee);
        (*thee) = VNULL;
    }
}

VPUBLIC void Vfetk_dtor2(Vfetk *thee) {
    Vcsm_dtor(&(thee->csm));
    AM_dtor(&(thee->am));
    Aprx_dtor(&(thee->aprx));
    Vfetk_PDE_dtor(&(thee->pde));
    Vmem_dtor(&(thee->vmem));
}

VPUBLIC double* Vfetk_getSolution(Vfetk *thee,
                                  int *length
                                 ) {

   int i;
   double *solution,
          *theAnswer;
   AM *am;

   VASSERT(thee != VNULL);

   /* Get the AM object */
   am = thee->am;
   /* Copy the solution into the w0 vector */
   Bvec_copy(am->w0, am->u);
   /* Add the Dirichlet conditions */
   Bvec_axpy(am->w0, am->ud, 1.);
   /* Get the data from the Bvec */
   solution = Bvec_addr(am->w0);
   /* Get the length of the data from the Bvec */
   *length = Bvec_numRT(am->w0);
   /* Make sure that we got scalar data (only one block) for the solution
    * to the FETK */
   VASSERT(1 == Bvec_numB(am->w0));
   /* Allocate space for the returned vector and copy the solution into it */
   theAnswer = VNULL;
```

```
    theAnswer = (double*)Vmem_malloc(VNULL, *length, sizeof(double));
    VASSERT(theAnswer != VNULL);
    for (i=0; i<(*length); i++) theAnswer[i] = solution[i];

    return theAnswer;
}


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

     double totEnergy = 0.0,
            qfEnergy = 0.0,
            dqmEnergy = 0.0;
     VASSERT(thee != VNULL);

     if (nonlin && (Vpbe_getBulkIonicStrength(thee->pbe) > 0.)) {
         Vnm_print(0, "Vfetk_energy:  calculating full PBE energy\n");
         Vnm_print(0, "Vfetk_energy:  bulk ionic strength = %g M\n",
           Vpbe_getBulkIonicStrength(thee->pbe));
         dqmEnergy = Vfetk_dqmEnergy(thee, color);
         Vnm_print(0, "Vfetk_energy:  dqmEnergy = %g kT\n", dqmEnergy);
         qfEnergy = Vfetk_qfEnergy(thee, color);
         Vnm_print(0, "Vfetk_energy:  qfEnergy = %g kT\n", qfEnergy);

         totEnergy = qfEnergy - dqmEnergy;
     } else {
         Vnm_print(0, "Vfetk_energy:  calculating only q-phi energy\n");
         dqmEnergy = Vfetk_dqmEnergy(thee, color);
         Vnm_print(0, "Vfetk_energy:  dqmEnergy = %g kT (NOT USED)\n", dqmEnergy);
         qfEnergy = Vfetk_qfEnergy(thee, color);
         Vnm_print(0, "Vfetk_energy:  qfEnergy = %g kT\n", qfEnergy);
         totEnergy = 0.5*qfEnergy;
     }

     return totEnergy;

}


VPUBLIC double Vfetk_qfEnergy(Vfetk *thee,
                              int color
                             ) {

     double *sol,
            energy = 0.0;
     int nsol,
         iatom,
         natoms;
     AM *am;

     VASSERT(thee != VNULL);
     am = thee->am;

     /* Get the finest level solution */
     sol= VNULL;
     sol = Vfetk_getSolution(thee, &nsol);
     VASSERT(sol != VNULL);

     /* Make sure the number of entries in the solution array matches the
      * number of vertices currently in the mesh */
     if (nsol != Gem_numVV(thee->gm)) {
        Vnm_print(2, "Vfetk_qfEnergy: Number of unknowns in solution does not match\n");
        Vnm_print(2, "Vfetk_qfEnergy: number of vertices in mesh!!!  Bailing out!\n");
        VASSERT(0);
     }

     /* Now we do the sum over atoms... */
     natoms = Valist_getNumberAtoms(thee->pbe->alist);
     for (iatom=0; iatom<natoms; iatom++) {

         energy = energy + Vfetk_qfEnergyAtom(thee, iatom, color, sol);

     } /* end for iatom */
```

```c

    /* Destroy the finest level solution */
    Vmem_free(VNULL, nsol, sizeof(double), (void **)&sol);

    /* Return the energy */
    return energy;
}

VPRIVATE double Vfetk_qfEnergyAtom(
        Vfetk *thee,
        int iatom,
        int color,
        double *sol) {

    Vatom *atom;
    double charge,
           phi[VAPBS_NVS],
           phix[VAPBS_NVS][3],
           *position,
           uval,
           energy = 0.0;
    int isimp,
        nsimps,
        icolor,
        ivert,
        usingColor;
    SS *simp;


    /* Get atom information */
    atom = Valist_getAtom(thee->pbe->alist, iatom);
    icolor = Vfetk_getAtomColor(thee, iatom);
    charge = Vatom_getCharge(atom);
    position = Vatom_getPosition(atom);

    /* Find out if we're using colors */
    usingColor = (color >= 0);

    if (usingColor && (icolor<0)) {
        Vnm_print(2, "Vfetk_qfEnergy: Atom colors not set!\n");
        VASSERT(0);
    }

    /* Check if this atom belongs to the specified partition */
    if ((icolor==color) || (!usingColor)) {
        /* Loop over the simps associated with this atom */
        nsimps =  Vcsm_getNumberSimplices(thee->csm, iatom);

        /* Get the first simp of the correct color; we can use just one
         * simplex for energy evaluations, but not for force
         * evaluations */
        for (isimp=0; isimp<nsimps; isimp++) {

            simp = Vcsm_getSimplex(thee->csm, isimp, iatom);

            /* If we've asked for a particular partition AND if the atom
             * is our partition, then compute the energy */
            if ((SS_chart(simp)==color)||(color<0)) {
                /* Get the value of each basis function evaluated at this
                 * point */
                Gem_pointInSimplexVal(thee->gm, simp, position, phi, phix);
                for (ivert=0; ivert<SS_dimVV(simp); ivert++) {
                    uval = sol[VV_id(SS_vertex(simp,ivert))];
                    energy += (charge*phi[ivert]*uval);
                } /* end for ivert */
                /* We only use one simplex of the appropriate color for
                 * energy calculations, so break here */
                break;
            } /* endif (color) */
        } /* end for isimp */
    }

   return energy;
}


VPUBLIC double Vfetk_dqmEnergy(Vfetk *thee,
                               int color) {

    return AM_evalJ(thee->am);
```

```
}

VPUBLIC void Vfetk_setAtomColors(Vfetk *thee) {

    SS *simp;
    Vatom *atom;
    int i,
        natoms;

    VASSERT(thee != VNULL);

    natoms = Valist_getNumberAtoms(thee->pbe->alist);
    for (i=0; i<natoms; i++) {
        atom = Valist_getAtom(thee->pbe->alist, i);
        simp = Vcsm_getSimplex(thee->csm, 0, i);
        Vatom_setPartID(atom, SS_chart(simp));
    }

}

VPUBLIC unsigned long int Vfetk_memChk(Vfetk *thee) {

    int memUse = 0;

    if (thee == VNULL) return 0;

    memUse = memUse + sizeof(Vfetk);
    memUse = memUse + Vcsm_memChk(thee->csm);

    return memUse;
}

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

    VASSERT(thee != VNULL);

    AM *am = VNULL; /* @todo - no idea what this is */
    Gem *gm = VNULL; /* Geometry manager */

    int skey = 0,  /* Simplex format */
        bufsize = 0, /* Buffer size */
        i, /* Loop counter */
        j; /* Loop counter */
    char *key = "r",  /* Read */
         *iodev = "BUFF",  /* Buffer */
         *iofmt = "ASC",  /* ASCII */
         *iohost = "localhost",  /* localhost (dummy) */
         *iofile = "0",  /*< socket 0 (dummy) */
         buf[VMAX_BUFSIZE]; /* Socket buffer */
    Vio *sock = VNULL; /* Socket object */
    VV *vx = VNULL; /* @todo - no idea what this is */
    double x;

    am = thee->am;
    VASSERT(am != VNULL);
    gm = thee->gm;
    VASSERT(gm != VNULL);

    /* @note This code is based on Gem_makeCube by Mike Holst */
    /* Write mesh string to buffer and read back */
  switch (meshType) {
   case VML_DIRICUBE:
       /* Create a new copy of the DIRICUBE mesh (see globals higher in this file) */
    bufsize = strlen(diriCubeString);
    VASSERT( bufsize <= VMAX_BUFSIZE );
    strncpy(buf, diriCubeString, VMAX_BUFSIZE);
    break;
   case VML_NEUMCUBE:
       /* Create a new copy of the NEUMCUBE mesh (see globals higher in this file) */
    bufsize = strlen(neumCubeString);
    Vnm_print(2, "Vfetk_genCube:  WARNING!  USING EXPERIMENTAL NEUMANN BOUNDARY CONDITIONS!\n");
    VASSERT( bufsize <= VMAX_BUFSIZE );
```

```
    strncpy(buf, neumCubeString, VMAX_BUFSIZE);
    break;
   case VML_EXTERNAL:
    Vnm_print(2, "Vfetk_genCube:  Got request for external mesh!\n");
    Vnm_print(2, "Vfetk_genCube:  How did we get here?\n");
    return VRC_FAILURE;
   default:
    Vnm_print(2, "Vfetk_genCube:  Unknown mesh type (%d)\n", meshType);
    return VRC_FAILURE;
  }

     VASSERT( VNULL != (sock=Vio_socketOpen(key,iodev,iofmt,iohost,iofile)) ); /* Open socket */
     Vio_bufTake(sock, buf, bufsize); /* Initialize internal buffer for socket */
     AM_read(am, skey, sock); /* Take the initial mesh from the socket and load
                                 into internal AM data structure with simplex
                                 format */
  Vio_connectFree(sock); /* Purge output buffers */
     Vio_bufGive(sock); /* Get pointer to output buffer?  No assignment of return value... */
     Vio_dtor(&sock); /* Destroy output buffer */

  /* @todo - could the following be done in a single pass? - PCE */
     /* Scale (unit) cube - for each vertex, set the new coordinates of that
        vertex based on the vertex length */
     for (i=0; i<Gem_numVV(gm); i++) {
         vx = Gem_VV(gm, i);
         for (j=0; j<3; j++) {
             x = VV_coord(vx, j);
             x *= length[j];
             VV_setCoord(vx, j, x);
         }
     }

     /* Add new center - for each vertex, set a new center for the vertex */
     for (i=0; i<Gem_numVV(gm); i++) {
         vx = Gem_VV(gm, i);
         for (j=0; j<3; j++) {
             x = VV_coord(vx, j);
             x += center[j];
             VV_setCoord(vx, j, x);
         }
     }

     return VRC_SUCCESS;
}

VPUBLIC Vrc_Codes Vfetk_loadMesh(Vfetk *thee, /* Vfetk object to load into */
                                 double center[3], /* Center for mesh (if constructed) */
                                 double length[3], /* Mesh lengths (if constructed) */
                                 Vfetk_MeshLoad meshType, /* Type of mesh to load */
                                 Vio *sock /* Socket for external mesh data (NULL otherwise) */
                                ) {

  Vrc_Codes vrc; /* Function return codes - see vhal.h for enum */
  int skey = 0;  /* Simplex format */

  /* Load mesh from socket if external mesh, otherwise generate mesh */
  switch (meshType) {
   case VML_EXTERNAL:
    if (sock == VNULL) {
     Vnm_print(2, "Vfetk_loadMesh:  Got NULL socket!\n");
     return VRC_FAILURE;
    }
    AM_read(thee->am, skey, sock);
    Vio_connectFree(sock);
    Vio_bufGive(sock);
    Vio_dtor(&sock);
    break;
   case VML_DIRICUBE:
   case VML_NEUMCUBE:
       /* Create new mesh and store in thee */
    vrc = Vfetk_genCube(thee, center, length, meshType);
    if (vrc == VRC_FAILURE) return VRC_FAILURE;
    break;
```

```
  default:
   Vnm_print(2, "Vfetk_loadMesh:  unrecognized mesh type (%d)!\n",
       meshType);
   return VRC_FAILURE;
 };

 /* Setup charge-simplex map */
    Vnm_print(0, "Vfetk_ctor2:  Constructing Vcsm...\n");
    thee->csm = VNULL;
    /* Construct a new Vcsm with the atom list and gem data */
    thee->csm = Vcsm_ctor(Vpbe_getValist(thee->pbe
), thee->gm);
    VASSERT(thee->csm != VNULL);
    Vcsm_init(thee->csm);

 return VRC_SUCCESS;
}


VPUBLIC void Bmat_printHB(Bmat *thee,
                          char *fname
                         ) {

    Mat *Ablock;
    MATsym pqsym;
    int i, j, jj;
    int *IA, *JA;
    double *D, *L, *U;
    FILE *fp;

    char mmtitle[72];
    char mmkey[] = {"8charkey"};
    int totc = 0, ptrc = 0, indc = 0, valc = 0;
    char mxtyp[] = {"RUA"}; /* Real Unsymmetric Assembled */
    int nrow = 0, ncol = 0, numZ = 0;
    int numZdigits = 0, nrowdigits = 0;
    int nptrline = 8, nindline = 8, nvalline = 5;
    char ptrfmt[] = {"(8I10)          "}, ptrfmtstr[] = {"%10d"};
    char indfmt[] = {"(8I10)          "}, indfmtstr[] = {"%10d"};
    char valfmt[] = {"(5E16.8)            "}, valfmtstr[] = {"%16.8E"};

    VASSERT( thee->numB == 1 );             /* HARDWIRE FOR NOW */
    Ablock = thee->AD[0][0];

    VASSERT( Mat_format( Ablock ) == DRC_FORMAT );  /* HARDWIRE FOR NOW */

    pqsym = Mat_sym( Ablock );

    if ( pqsym == IS_SYM ) {
        mxtyp[1] = 'S';
    } else if ( pqsym == ISNOT_SYM ) {
        mxtyp[1] = 'U';
    } else {
        VASSERT( 0 ); /* NOT VALID */
    }

    nrow = Bmat_numRT( thee ); /* Number of rows */
    ncol = Bmat_numCT( thee ); /* Number of cols */
    numZ = Bmat_numZT( thee ); /* Number of entries */

    nrowdigits = (int) (log( nrow )/log( 10 )) + 1;
    numZdigits = (int) (log( numZ )/log( 10 )) + 1;

    nptrline = (int) ( 80 / (numZdigits + 1) );
    nindline = (int) ( 80 / (nrowdigits + 1) );

    sprintf(ptrfmt,"(%dI%d)",nptrline,numZdigits+1);
    sprintf(ptrfmtstr,"%%%dd",numZdigits+1);
    sprintf(indfmt,"(%dI%d)",nindline,nrowdigits+1);
    sprintf(indfmtstr,"%%%dd",nrowdigits+1);

    ptrc = (int) ( ( (ncol + 1) - 1 ) / nptrline ) + 1;
    indc = (int) ( (numZ - 1) / nindline ) + 1;
    valc = (int) ( (numZ - 1) / nvalline ) + 1;

    totc = ptrc + indc + valc;

    sprintf( mmtitle, "Sparse '%s' Matrix - Harwell-Boeing Format - '%s'",
             thee->name, fname );

   /* Step 0:  Open the file for writing */
```

```
    fp = fopen( fname, "w" );
    if (fp == VNULL) {
        Vnm_print(2,"Bmat_printHB:  Ouch couldn't open file <%s>\n",fname);
        return;
    }

    /* Step 1:  Print the header information */

    fprintf( fp, "%-72s%-8s\n", mmtitle, mmkey );
    fprintf( fp, "%14d%14d%14d%14d%14d\n", totc, ptrc, indc, valc, 0 );
    fprintf( fp, "%3s%11s%14d%14d%14d\n", mxtyp, " ", nrow, ncol, numZ );
    fprintf( fp, "%-16s%-16s%-20s%-20s\n", ptrfmt, indfmt, valfmt, "6E13.5" );

    IA = Ablock->IA;
    JA = Ablock->JA;
    D = Ablock->diag;
    L = Ablock->offL;
    U = Ablock->offU;

    if ( pqsym == IS_SYM ) {

        /* Step 2:  Print the pointer information */

        for (i=0; i<(ncol+1); i++) {
            fprintf( fp, ptrfmtstr, Ablock->IA[i] + (i+1) );
            if ( ( (i+1) % nptrline ) == 0 ) {
                fprintf( fp, "\n" );
            }
        }

        if ( ( (ncol+1) % nptrline ) != 0 ) {
            fprintf( fp, "\n" );
        }

        /* Step 3:  Print the index information */

        j = 0;
        for (i=0; i<ncol; i++) {
            fprintf( fp, indfmtstr, i+1); /* diagonal */
            if ( ( (j+1) % nindline ) == 0 ) {
                fprintf( fp, "\n" );
            }
            j++;
            for (jj=IA[i]; jj<IA[i+1]; jj++) {
                fprintf( fp, indfmtstr, JA[jj] + 1 ); /* lower triangle */
                if ( ( (j+1) % nindline ) == 0 ) {
                    fprintf( fp, "\n" );
                }
                j++;
            }
        }

        if ( ( j % nindline ) != 0 ) {
            fprintf( fp, "\n" );
        }

        /* Step 4:  Print the value information */

        j = 0;
        for (i=0; i<ncol; i++) {
            fprintf( fp, valfmtstr, D[i] );
            if ( ( (j+1) % nvalline ) == 0 ) {
                fprintf( fp, "\n" );
            }
            j++;
            for (jj=IA[i]; jj<IA[i+1]; jj++) {
                fprintf( fp, valfmtstr, L[jj] );
                if ( ( (j+1) % nvalline ) == 0 ) {
                    fprintf( fp, "\n" );
                }
                j++;
            }
        }

        if ( ( j % nvalline ) != 0 ) {
            fprintf( fp, "\n" );
        }

    } else { /* ISNOT_SYM */

```

```
        VASSERT( 0 ); /* NOT CODED YET */
    }

    /* Step 5:  Close the file */
    fclose( fp );
}

VPUBLIC PDE* Vfetk_PDE_ctor(Vfetk *fetk) {

    PDE *thee = VNULL;

    thee = (PDE*)Vmem_malloc(fetk->vmem, 1, sizeof(PDE));
    VASSERT(thee != VNULL);
    VASSERT(Vfetk_PDE_ctor2(thee, fetk));

    return thee;
}

VPUBLIC int Vfetk_PDE_ctor2(PDE *thee,
                            Vfetk *fetk
                           ) {

    int i;

    if (thee == VNULL) {
        Vnm_print(2, "Vfetk_PDE_ctor2:  Got NULL thee!\n");
        return 0;
    }

    /* Store a local copy of the Vfetk class */
    var.fetk = fetk;

    /* PDE-specific parameters and function pointers */
    thee->initAssemble = Vfetk_PDE_initAssemble;
    thee->initElement  = Vfetk_PDE_initElement;
    thee->initFace     = Vfetk_PDE_initFace;
    thee->initPoint    = Vfetk_PDE_initPoint;
    thee->Fu           = Vfetk_PDE_Fu;
    thee->Fu_v         = Vfetk_PDE_Fu_v;
    thee->DFu_wv       = Vfetk_PDE_DFu_wv;
    thee->delta        = Vfetk_PDE_delta;
    thee->u_D          = Vfetk_PDE_u_D;
    thee->u_T          = Vfetk_PDE_u_T;
    thee->Ju           = Vfetk_PDE_Ju;
    thee->vec          = 1; /* FIX! */
    thee->sym[0][0]    = 1;
    thee->est[0]       = 1.0;
    for (i=0; i<VMAX_BDTYPE; i++) thee->bmap[0][i] = i;

    /* Manifold-specific function pointers */
    thee->bisectEdge  = Vfetk_PDE_bisectEdge;
    thee->mapBoundary = Vfetk_PDE_mapBoundary;
    thee->markSimplex = Vfetk_PDE_markSimplex;
    thee->oneChart    = Vfetk_PDE_oneChart;

    /* Element-specific function pointers */
    thee->simplexBasisInit = Vfetk_PDE_simplexBasisInit;
    thee->simplexBasisForm = Vfetk_PDE_simplexBasisForm;

    return 1;
}

VPUBLIC void Vfetk_PDE_dtor(PDE **thee) {

    if ((*thee) != VNULL) {
        Vfetk_PDE_dtor2(*thee);
  /* TODO: The following line is commented out because at the moment,
   there is a seg fault when deallocating at the end of a run. Since
   this routine is called only once at the very end, we'll leave it
   commented out. However, this could be a memory leak.
   */
        /* Vmem_free(var.fetk->vmem, 1, sizeof(PDE), (void **)thee); */
        (*thee) = VNULL;
    }

}

VPUBLIC void Vfetk_PDE_dtor2(PDE *thee) {
    var.fetk = VNULL;
```

```
}

VPRIVATE double smooth(int nverts, double dist[VAPBS_NVS], double coeff[VAPBS_NVS], int meth) {

    int i;
    double weight;
    double num = 0.0;
    double den = 0.0;

    for (i=0; i<nverts; i++) {
        if (dist[i] < VSMALL) return coeff[i];
        weight = 1.0/dist[i];
        if (meth == 0) {
            num += (weight * coeff[i]);
            den += weight;
        } else if (meth == 1) {
            /* Small coefficients reset the average to 0; we need to break out
             * of the loop */
            if (coeff[i] < VSMALL) {
                num = 0.0;
                break;
            } else {
                num += weight; den += (weight/coeff[i]);
            }
        } else VASSERT(0);
    }

    return (num/den);

}

VPRIVATE double diel() {

    int i, j;
    double eps, epsp, epsw, dist[5], coeff[5], srad, swin, *vx;
    Vsurf_Meth srfm;
    Vacc *acc;
    PBEparm *pbeparm;

    epsp = Vpbe_getSoluteDiel(var.fetk->pbe);
    epsw = Vpbe_getSolventDiel(var.fetk->pbe);
    VASSERT(var.fetk->pbeparm != VNULL);
    pbeparm = var.fetk->pbeparm;
    srfm = pbeparm->srfm;
    srad = pbeparm->srad;
    swin = pbeparm->swin;
    acc = var.fetk->pbe->acc;

    eps = 0;

    if (VABS(epsp - epsw) < VSMALL) return epsp;
    switch (srfm) {
        case VSM_MOL:
            eps = ((epsw-epsp)*Vacc_molAcc(acc, var.xq, srad) + epsp);
            break;
        case VSM_MOLSMOOTH:
            for (i=0; i<var.nverts; i++) {
                dist[i] = 0;
                vx = var.vx[i];
                for (j=0; j<3; j++) {
                    dist[i] += VSQR(var.xq[j] - vx[j]);
                }
                dist[i] = VSQRT(dist[i]);
                coeff[i] = (epsw-epsp)*Vacc_molAcc(acc, var.xq, srad) + epsp;
            }
            eps = smooth(var.nverts, dist, coeff, 1);
            break;
        case VSM_SPLINE:
            eps = ((epsw-epsp)*Vacc_splineAcc(acc, var.xq, swin, 0.0) + epsp);
            break;
        default:
            Vnm_print(2, "Undefined surface method (%d)!\n", srfm);
            VASSERT(0);
    }

    return eps;
}
```

```
VPRIVATE double ionacc() {

    int i, j;
    double dist[5], coeff[5], irad, swin, *vx, accval;
    Vsurf_Meth srfm;
    Vacc *acc = VNULL;
    PBEparm *pbeparm = VNULL;

    VASSERT(var.fetk->pbeparm != VNULL);
    pbeparm = var.fetk->pbeparm;
    srfm = pbeparm->srfm;
    irad = Vpbe_getMaxIonRadius(var.fetk->pbe);
    swin = pbeparm->swin;
    acc = var.fetk->pbe->acc;

    if (var.zks2 < VSMALL) return 0.0;
    switch (srfm) {
        case VSM_MOL:
            accval = Vacc_ivdwAcc(acc, var.xq, irad);
            break;
        case VSM_MOLSMOOTH:
            for (i=0; i<var.nverts; i++) {
                dist[i] = 0;
                vx = var.vx[i];
                for (j=0; j<3; j++) {
                    dist[i] += VSQR(var.xq[j] - vx[j]);
                }
                dist[i] = VSQRT(dist[i]);
                coeff[i] = Vacc_ivdwAcc(acc, var.xq, irad);
            }
            accval = smooth(var.nverts, dist, coeff, 1);
            break;
        case VSM_SPLINE:
            accval = Vacc_splineAcc(acc, var.xq, swin, irad);
            break;
        default:
            Vnm_print(2, "Undefined surface method (%d)!\n", srfm);
            VASSERT(0);
    }

    return accval;
}

VPRIVATE double debye_U(Vpbe *pbe, int d, double x[]) {

    double size, *position, charge, xkappa, eps_w, dist, T, pot, val;
    int iatom, i;
    Valist *alist;
    Vatom *atom;

    eps_w = Vpbe_getSolventDiel(pbe);
    xkappa = (1.0e10)*Vpbe_getXkappa(pbe);
    T = Vpbe_getTemperature(pbe);
    alist = Vpbe_getValist(pbe);
    val = 0;
    pot = 0;

    for (iatom=0; iatom<Valist_getNumberAtoms(alist);
iatom++) {
        atom = Valist_getAtom(alist, iatom);
        position = Vatom_getPosition(atom);
        charge = Vunit_ec*Vatom_getCharge(atom);
        size = (1e-10)*Vatom_getRadius(atom);
        dist = 0;
        for (i=0; i<d; i++) {
            dist += VSQR(position[i] - x[i]);
        }
        dist = (1.0e-10)*VSQRT(dist);
        val = (charge)/(4*VPI*Vunit_eps0*eps_w*dist);
        if (xkappa != 0.0) {
            val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
        }
        pot = pot + val;
    }
    pot = pot*Vunit_ec/(Vunit_kb*T);

    return pot;
}

VPRIVATE double debye_Udiff(Vpbe *pbe, int d, double x[]) {
```

```
    double size, *position, charge, eps_p, dist, T, pot, val;
    double Ufull;
    int iatom, i;
    Valist *alist;
    Vatom *atom;

    Ufull = debye_U(pbe, d, x);

    eps_p = Vpbe_getSoluteDiel(pbe);
    T = Vpbe_getTemperature(pbe);
    alist = Vpbe_getValist(pbe);
    val = 0;
    pot = 0;

    for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
        atom = Valist_getAtom(alist, iatom);
        position = Vatom_getPosition(atom);
        charge = Vunit_ec*Vatom_getCharge(atom);
        size = (1e-10)*Vatom_getRadius(atom);
        dist = 0;
        for (i=0; i<d; i++) {
            dist += VSQR(position[i] - x[i]);
        }
        dist = (1.0e-10)*VSQRT(dist);
        val = (charge)/(4*VPI*Vunit_eps0*eps_p*dist);
        pot = pot + val;
    }
    pot = pot*Vunit_ec/(Vunit_kb*T);

    pot = Ufull - pot;

    return pot;
}

VPRIVATE void coulomb(Vpbe *pbe, int d, double pt[], double eps, double *U,
  double dU[], double *d2U) {

    int iatom, i;
    double T, pot, fx, fy, fz, x, y, z, scale;
    double *position, charge, dist, dist2, val, vec[3], dUold[3], Uold;
    Valist *alist;
    Vatom *atom;

    /* Initialize variables */
    T = Vpbe_getTemperature(pbe);
    alist = Vpbe_getValist(pbe);
    pot = 0;  fx = 0; fy = 0; fz = 0;
    x = pt[0]; y = pt[1]; z = pt[2];

    /* Calculate */
    if (!Vgreen_coulombD(var.green, 1, &x, &y, &z, &pot, &fx, &fy, &fz)) {
        Vnm_print(2, "Error calculating Green's function!\n");
        VASSERT(0);
    }


    /* Scale the results */
    scale = Vunit_ec/(eps*Vunit_kb*T);
    *U = pot*scale;
    *d2U = 0.0;
    dU[0] = -fx*scale;
    dU[1] = -fy*scale;
    dU[2] = -fz*scale;

#if 0
    /* Compare with old results */
    val = 0.0;
    Uold = 0.0; dUold[0] = 0.0; dUold[1] = 0.0; dUold[2] = 0.0;
    for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
        atom = Valist_getAtom(alist, iatom);
        position = Vatom_getPosition(atom);
        charge = Vatom_getCharge(atom);
        dist2 = 0;
        for (i=0; i<d; i++) {
            vec[i] = (position[i] - pt[i]);
            dist2 += VSQR(vec[i]);
        }
```

```
        dist = VSQRT(dist2);

        /* POTENTIAL */
        Uold = Uold + charge/dist;

        /* GRADIENT */
        for (i=0; i<d; i++) dUold[i] = dUold[i] + vec[i]*charge/(dist2*dist);

    }
    Uold = Uold*VSQR(Vunit_ec)*(1.0e10)/(4*VPI*Vunit_eps0*eps*Vunit_kb*T);
    for (i=0; i<d; i++) {
        dUold[i] = dUold[i]*VSQR(Vunit_ec)*(1.0e10)/(4*VPI*Vunit_eps0*eps*Vunit_kb*T);
    }

    printf("Unew - Uold = %g - %g = %g\n", *U, Uold, (*U - Uold));
    printf("||dUnew - dUold||^2 = %g\n", (VSQR(dU[0] - dUold[0])
                + VSQR(dU[1] - dUold[1]) + VSQR(dU[2] - dUold[2])));
    printf("dUnew[0] = %g, dUold[0] = %g\n", dU[0], dUold[0]);
    printf("dUnew[1] = %g, dUold[1] = %g\n", dU[1], dUold[1]);
    printf("dUnew[2] = %g, dUold[2] = %g\n", dU[2], dUold[2]);

#endif

}

VPUBLIC void Vfetk_PDE_initAssemble(PDE *thee, int ip[], double rp[]) {

#if 1
    /* Re-initialize the Green's function oracle in case the atom list has
     * changed */
    if (var.initGreen) {
        Vgreen_dtor(&(var.green));
        var.initGreen = 0;
    }
    var.green = Vgreen_ctor(var.fetk->pbe->alist);
    var.initGreen = 1;
#else
    if (!var.initGreen) {
        var.green = Vgreen_ctor(var.fetk->pbe->alist);
        var.initGreen = 1;
    }
#endif

}

VPUBLIC void Vfetk_PDE_initElement(PDE *thee, int elementType, int chart,
  double tvx[][3], void *data) {

    int i, j;
    double epsp, epsw;

    /* We assume that the simplex has been passed in as the void *data * *
     * argument.  Store it */
    VASSERT(data != NULL);
    var.simp = (SS *)data;

    /* save the element type */
    var.sType = elementType;

    /* Grab the vertices from this simplex */
    var.nverts = thee->dim+1;
    for (i=0; i<thee->dim+1; i++) var.verts[i] = SS_vertex(var.simp, i);

    /* Vertex locations of this simplex */
    for (i=0; i<thee->dim+1; i++) {
        for (j=0; j<thee->dim; j++) {
            var.vx[i][j] = tvx[i][j];
        }
    }

    /* Set the dielectric constant for this element for use in the jump term *
     * of the residual-based error estimator.  The value is set to the average
     * * value of the vertices */
    var.jumpDiel = 0;  /* NOT IMPLEMENTED YET! */
}
```

```c

VPUBLIC void Vfetk_PDE_initFace(PDE *thee, int faceType, int chart,
  double tnvec[]) {

    int i;

    /* unit normal vector of this face */
    for (i=0; i<thee->dim; i++) var.nvec[i] = tnvec[i];

    /* save the face type */
    var.fType = faceType;
}

VPUBLIC void Vfetk_PDE_initPoint(PDE *thee, int pointType, int chart,
  double txq[], double tU[], double tdU[][3]) {

    int i, j, ichop;
    double u2, coef2, eps_p;
    Vhal_PBEType pdetype;
    Vpbe *pbe = VNULL;

    eps_p = Vpbe_getSoluteDiel(var.fetk->pbe);
    pdetype = var.fetk->type;
    pbe = var.fetk->pbe;

    /* the point, the solution value and gradient, and the Coulomb value and *
     * gradient at the point */
    if ((pdetype == PBE_LRPBE) || (pdetype == PBE_NRPBE)) {
        coulomb(pbe, thee->dim, txq, eps_p, &(var.W), var.dW, &(var.d2W));
    }
    for (i=0; i<thee->vec; i++) {
        var.U[i] = tU[i];
        for (j=0; j<thee->dim; j++) {
            var.xq[j] = txq[j];
            var.dU[i][j] = tdU[i][j];
        }
    }

    /* interior form case */
    if (pointType == 0) {

        /* Get the dielectric values */
        var.diel  = diel();
        var.ionacc  = ionacc();
        var.A = var.diel;
        var.F = (var.diel - eps_p);

        switch (pdetype) {

            case PBE_LPBE:
                var.DB = var.ionacc*var.zkappa2*var.ionstr;
                var.B = var.DB*var.U[0];
                break;

            case PBE_NPBE:

                var.B  = 0;
                var.DB  = 0;
                if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
                    for (i=0; i<var.nion; i++) {
                        u2 = -1.0 * var.U[0] * var.ionQ[i];

                        /* NONLINEAR TERM */
                        coef2 = -1.0 * var.ionacc * var.zks2 * var.ionConc[i];
                        var.B += (coef2 * Vcap_exp(u2, &ichop));
                        /* LINEARIZED TERM */
                        coef2 = -1.0 * var.ionQ[i] * coef2;
                        var.DB += (coef2 * Vcap_exp(u2, &ichop));
                    }
                }
                break;

            case PBE_LRPBE:
                var.DB = var.ionacc*var.zkappa2*var.ionstr;
                var.B = var.DB*(var.U[0]+var.W);
                break;

```

```
            case PBE_NRPBE:

                var.B  = 0;
                var.DB  = 0;
                if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
                    for (i=0; i<var.nion; i++) {
                        u2 = -1.0 * (var.U[0] + var.W) * var.ionQ[i];

                        /* NONLINEAR TERM */
                        coef2 = -1.0 * var.ionacc * var.zks2 * var.ionConc[i];
                        var.B += (coef2 * Vcap_exp(u2, &ichop));

                        /* LINEARIZED TERM */
                        coef2 = -1.0 * var.ionQ[i] * coef2;
                        var.DB += (coef2 * Vcap_exp(u2, &ichop));
                    }
                }
                break;

   case PBE_SMPBE: /* SMPBE Temp */

                var.B  = 0;
                var.DB  = 0;
                if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
                    for (i=0; i<var.nion; i++) {
                        u2 = -1.0 * var.U[0] * var.ionQ[i];

                        /* NONLINEAR TERM */
                        coef2 = -1.0 * var.ionacc * var.zks2 * var.ionConc[i];
                        var.B += (coef2 * Vcap_exp(u2, &ichop));
                        /* LINEARIZED TERM */
                        coef2 = -1.0 * var.ionQ[i] * coef2;
                        var.DB += (coef2 * Vcap_exp(u2, &ichop));
                    }
                }
     break;
            default:
                Vnm_print(2, "Vfetk_PDE_initPoint:  Unknown PBE type (%d)!\n",
                  pdetype);
                VASSERT(0);
                break;
        }


    /* boundary form case */
    } else {
#ifdef DONEUMANN
        ;
#else
        Vnm_print(2, "Vfetk:  Whoa!  I just got a boundary point to evaluate (%d)!\n", pointType);
        Vnm_print(2, "Vfetk:  Did you do that on purpose?\n");
#endif
    }

#if 0 /* THIS IS VERY NOISY! */
    Vfetk_dumpLocalVar();
#endif

}

VPUBLIC void Vfetk_PDE_Fu(PDE *thee, int key, double F[]) {

    //Vnm_print(2, "Vfetk_PDE_Fu:  Setting error to zero!\n");

    F[0] = 0.;

}

VPUBLIC double Vfetk_PDE_Fu_v(
        PDE *thee,
        int key,
        double V[],
        double dV[][VAPBS_DIM]
        ) {

    Vhal_PBEType type;
    int i;
    double value = 0.;
```

```

    type = var.fetk->type;

    /* interior form case */
    if (key == 0) {

        for (i=0; i<thee->dim; i++) value += ( var.A * var.dU[0][i] * dV[0][i] );
        value += var.B * V[0];

        if ((type == PBE_LRPBE) || (type == PBE_NRPBE)) {
            for (i=0; i<thee->dim; i++) {
                if (var.F > VSMALL) value += (var.F * var.dW[i] * dV[0][i]);
            }
        }

    /* boundary form case */
    } else {
#ifdef DONEUMANN
        value = 0.0;
#else
        Vnm_print(2, "Vfetk:  Whoa! I was just asked to evaluate a boundary weak form for point type %d!\n", key);
#endif
    }

    var.Fu_v = value;
    return value;
}

VPUBLIC double Vfetk_PDE_DFu_wv(
        PDE *thee,
        int key,
        double W[],
        double dW[][VAPBS_DIM],
        double V[],
        double dV[][3]
        ) {

    Vhal_PBEType type;
    int i;
    double value = 0.;

    type = var.fetk->type;

    /* Interior form */
    if (key == 0) {
            value += var.DB * W[0] * V[0];
            for (i=0; i<thee->dim; i++) value += ( var.A * dW[0][i] * dV[0][i] );

    /* boundary form case */
    } else {
#ifdef DONEUMANN
        value = 0.0;
#else
        Vnm_print(2, "Vfetk:  Whoa! I was just asked to evaluate a boundary weak form for point type %d!\n", key);
#endif
    }

    var.DFu_wv = value;
    return value;
}

#define VRINGMAX 1000

#define VATOMMAX 1000000
VPUBLIC void Vfetk_PDE_delta(PDE *thee, int type, int chart, double txq[],
  void *user, double F[]) {

    int iatom, jatom, natoms, atomIndex, atomList[VATOMMAX], nAtomList;
    int gotAtom, numSring, isimp, ivert, sid;
    double *position, charge, phi[VAPBS_NVS], phix[VAPBS_NVS][3], value;
    Vatom *atom;
    Vhal_PBEType pdetype;
    SS *sring[VRINGMAX];
    VV *vertex = (VV *)user;
```

```
    pdetype = var.fetk->type;

    F[0] = 0.0;

    if ((pdetype == PBE_LPBE) || (pdetype == PBE_NPBE) || (pdetype == PBE_SMPBE) /* SMPBE Added */) {
        VASSERT( vertex != VNULL);
        numSring = 0;
        sring[numSring] = VV_firstSS(vertex);
        while (sring[numSring] != VNULL) {
            numSring++;
            sring[numSring] = SS_link(sring[numSring-1], vertex);
        }
        VASSERT( numSring > 0 );
        VASSERT( numSring <= VRINGMAX );

        /* Move around the simplex ring and determine the charge locations */
        F[0] = 0.;
        charge = 0.;
        nAtomList = 0;
        for (isimp=0; isimp<numSring; isimp++) {
            sid = SS_id(sring[isimp]);
            natoms = Vcsm_getNumberAtoms(Vfetk_getVcsm(var.fetk), sid);
            for (iatom=0; iatom<natoms; iatom++) {
                /* Get the delta function information * */
                atomIndex = Vcsm_getAtomIndex(Vfetk_getVcsm(var.fetk),
                  iatom, sid);
                gotAtom = 0;
                for (jatom=0; jatom<nAtomList; jatom++) {
                    if (atomList[jatom] == atomIndex) {
                        gotAtom = 1;
                        break;
                    }
                }
                if (!gotAtom) {
                    VASSERT(nAtomList < VATOMMAX);
                    atomList[nAtomList] = atomIndex;
                    nAtomList++;

                    atom = Vcsm_getAtom(Vfetk_getVcsm(var.fetk), iatom, sid);
                    charge = Vatom_getCharge(atom);
                    position = Vatom_getPosition(atom);

                    /* Get the test function value at the delta function I
                     * used to do a VASSERT to make sure the point was in the
                     * simplex (i.e., make sure round-off error isn't an
                     * issue), but round off errors became an issue */
                    if (!Gem_pointInSimplexVal(Vfetk_getGem(var.fetk),
                      sring[isimp], position, phi, phix)) {
                        if (!Gem_pointInSimplex(Vfetk_getGem(var.fetk),
                          sring[isimp], position)) {
                            Vnm_print(2, "delta: Both Gem_pointInSimplexVal \
and Gem_pointInSimplex detected misplaced point charge!\n");
                            Vnm_print(2, "delta: I think you have problems: \
phi = {");
                            for (ivert=0; ivert<Gem_dimVV(Vfetk_getGem(var.fetk)); ivert++) Vnm_print(2, "%e ", phi[ivert]);

                             Vnm_print(2, "}\n");
                        }
                    }
                    value = 0;
                    for (ivert=0; ivert<Gem_dimVV(Vfetk_getGem(var.fetk)); ivert++) {
                        if (VV_id(SS_vertex(sring[isimp], ivert)) == VV_id(vertex)) value += phi[ivert];
                    }

                    F[0] += (value * Vpbe_getZmagic(var.fetk->pbe) * charge);
                } /* if !gotAtom */
            } /* for iatom */
        } /* for isimp */

    } else if ((pdetype == PBE_LRPBE) || (pdetype == PBE_NRPBE)) {
```

```
        F[0] = 0.0;
    } else { VASSERT(0); }

    var.delta = F[0];

}

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

    if ((var.fetk->type == PBE_LPBE) || (var.fetk->type
 == PBE_NPBE) || (var.fetk->type == PBE_SMPBE) /* SMPBE
 Added */) {
        F[0] = debye_U(var.fetk->pbe, thee->dim, txq);
    } else if ((var.fetk->type == PBE_LRPBE) || (var.fetk
->type == PBE_NRPBE)) {
        F[0] = debye_Udiff(var.fetk->pbe, thee->dim, txq);
    } else VASSERT(0);

    var.u_D = F[0];

}

VPUBLIC void Vfetk_PDE_u_T(PDE *thee, int type, int chart, double
txq[],
  double F[]) {
/*VPUBLIC void Vfetk_PDE_u_T(sPDE *thee,
                           int type,
                           int chart,
                           double txq[],
                           double F[],
                           double dF[][3]
                          ) { */

    F[0] = 0.0;
    var.u_T = F[0];

}


VPUBLIC void Vfetk_PDE_bisectEdge(int dim, int dimII, int
edgeType,
  int chart[], double vx[][3]) {

    int i;

    for (i=0; i<dimII; i++) vx[2][i] = .5 * (vx[0][i] + vx[1][i]);
    chart[2] = chart[0];

}

VPUBLIC void Vfetk_PDE_mapBoundary(int dim, int dimII, int
 vertexType,
  int chart, double vx[3]) {

}

VPUBLIC int Vfetk_PDE_markSimplex(int dim, int dimII, int
simplexType,
  int faceType[VAPBS_NVS], int vertexType[VAPBS_NVS], int chart[], double vx[][
3],
  void *simplex) {

    double targetRes, edgeLength, srad, swin, myAcc, refAcc;
    int i, natoms;
    Vsurf_Meth srfm;
    Vhal_PBEType type;
    FEMparm *feparm = VNULL;
    PBEparm *pbeparm = VNULL;
    Vpbe *pbe = VNULL;
    Vacc *acc = VNULL;
    Vcsm *csm = VNULL;
    SS *simp = VNULL;

    VASSERT(var.fetk->feparm != VNULL);
    feparm = var.fetk->feparm;
    VASSERT(var.fetk->pbeparm != VNULL);;
    pbeparm = var.fetk->pbeparm;
    pbe = var.fetk->pbe;
    csm = Vfetk_getVcsm(var.fetk);
```

```
    acc = pbe->acc;
    targetRes = feparm->targetRes;
    srfm = pbeparm->srfm;
    srad = pbeparm->srad;
    swin = pbeparm->swin;
    simp = (SS *)simplex;
    type = var.fetk->type;

    /* Check to see if this simplex is smaller than the target size */
    /* NAB WARNING:  I am providing face=-1 here to conform to the new MC API; however, I'm not sure if this is the correct behavior. */
    Gem_longestEdge(var.fetk->gm, simp, -1, &edgeLength);
    if (edgeLength < targetRes) return 0;

    /* For non-regularized PBE, check charge-simplex map */
    if ((type == PBE_LPBE) || (type == PBE_NPBE) || (type == PBE_SMPBE)  /* SMPBE Added */) {
        natoms = Vcsm_getNumberAtoms(csm, SS_id(simp));
        if (natoms > 0) {
            return 1;
        }
    }

    /* We would like to resolve the mesh between the van der Waals surface the
     * max distance from this surface where there could be coefficient
     * changes */
    switch(srfm) {
        case VSM_MOL:
            refAcc = Vacc_molAcc(acc, vx[0], srad);
            for (i=1; i<(dim+1); i++) {
                myAcc = Vacc_molAcc(acc, vx[i], srad);
                if (myAcc != refAcc) {
                    return 1;
                }
            }
            break;
        case VSM_MOLSMOOTH:
            refAcc = Vacc_molAcc(acc, vx[0], srad);
            for (i=1; i<(dim+1); i++) {
                myAcc = Vacc_molAcc(acc, vx[i], srad);
                if (myAcc != refAcc) {
                    return 1;
                }
            }
            break;
        case VSM_SPLINE:
            refAcc = Vacc_splineAcc(acc, vx[0], swin, 0.0);
            for (i=1; i<(dim+1); i++) {
                myAcc = Vacc_splineAcc(acc, vx[i], swin, 0.0);
                if (myAcc != refAcc) {
                    return 1;
                }
            }
            break;
        default:
            VASSERT(0);
            break;
    }

    return 0;
}

VPUBLIC void Vfetk_PDE_oneChart(int dim, int dimII, int objType, int chart[],
  double vx[][3], int dimV) {

}

VPUBLIC double Vfetk_PDE_Ju(PDE *thee, int key) {

    int i, ichop;
    double dielE, qmE, coef2, u2;
    double value = 0.;
    Vhal_PBEType type;

    type = var.fetk->type;

    /* interior form case */
    if (key == 0) {
        dielE = 0;
        for (i=0; i<3; i++) dielE += VSQR(var.dU[0][i]);
```

```
        dielE = dielE*var.diel;

        switch (type) {
            case PBE_LPBE:
                if ((var.ionacc > VSMALL) && (var.zkappa2 > VSMALL)) {
                    qmE = var.ionacc*var.zkappa2*VSQR(var.U[0]);
                } else qmE = 0;
                break;
            case PBE_NPBE:
                if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
                    qmE = 0.;
                    for (i=0; i<var.nion; i++) {
                        coef2 = var.ionacc * var.zks2 * var.ionConc[i] * var.ionQ[i];
                        u2 = -1.0 * (var.U[0]) * var.ionQ[i];
                        qmE += (coef2 * (Vcap_exp(u2, &ichop) - 1.0));
                    }
                } else qmE = 0;
                break;
            case PBE_LRPBE:
                if ((var.ionacc > VSMALL) && (var.zkappa2 > VSMALL)) {
                    qmE = var.ionacc*var.zkappa2*VSQR((var.U[0] + var.W));
                } else qmE = 0;
                break;
            case PBE_NRPBE:
                if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
                    qmE = 0.;
                    for (i=0; i<var.nion; i++) {
                        coef2 = var.ionacc * var.zks2 * var.ionConc[i] * var.ionQ[i];
                        u2 = -1.0 * (var.U[0] + var.W) * var.ionQ[i];
                        qmE += (coef2 * (Vcap_exp(u2, &ichop) - 1.0));
                    }
                } else qmE = 0;
                break;
   case PBE_SMPBE: /* SMPBE Temp */
                if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
                    qmE = 0.;
                    for (i=0; i<var.nion; i++) {
                        coef2 = var.ionacc * var.zks2 * var.ionConc[i] * var.ionQ[i];
                        u2 = -1.0 * (var.U[0]) * var.ionQ[i];
                        qmE += (coef2 * (Vcap_exp(u2, &ichop) - 1.0));
                    }
                } else qmE = 0;
                break;
            default:
                Vnm_print(2, "Vfetk_PDE_Ju:  Invalid PBE type (%d)!\n", type);
                VASSERT(0);
                break;
        }

        value = 0.5*(dielE + qmE)/Vpbe_getZmagic(var.fetk->pbe);

    /* boundary form case */
    } else if (key == 1) {
        value = 0.0;

    /* how did we get here? */
    } else VASSERT(0);

    return value;

}

VPUBLIC void Vfetk_externalUpdateFunction(SS **simps, int num) {

    Vcsm *csm = VNULL;
    int rc;

    VASSERT(var.fetk != VNULL);
    csm = Vfetk_getVcsm(var.fetk);
    VASSERT(csm != VNULL);

    rc = Vcsm_update(csm, simps, num);

    if (!rc) {
        Vnm_print(2, "Error while updating charge-simplex map!\n");
```

```
        VASSERT(0);
    }
}

VPRIVATE void polyEval(int numP, double p[], double c[][VMAXP], double xv[]) {
    int i;
    double x, y, z;

    x = xv[0];
    y = xv[1];
    z = xv[2];
    for (i=0; i<numP; i++) {
        p[i] = c[i][0]
             + c[i][1]  * x
             + c[i][2]  * y
             + c[i][3]  * z
             + c[i][4]  * x*x
             + c[i][5]  * y*y
             + c[i][6]  * z*z
             + c[i][7]  * x*y
             + c[i][8]  * x*z
             + c[i][9]  * y*z
             + c[i][10] * x*x*x
             + c[i][11] * y*y*y
             + c[i][12] * z*z*z
             + c[i][13] * x*x*y
             + c[i][14] * x*x*z
             + c[i][15] * x*y*y
             + c[i][16] * y*y*z
             + c[i][17] * x*z*z
             + c[i][18] * y*z*z;
    }
}

VPRIVATE void setCoef(int numP, double c[][VMAXP], double cx[][VMAXP],
  double cy[][VMAXP], double cz[][VMAXP], int ic[][VMAXP], int icx[][VMAXP],
  int icy[][VMAXP], int icz[][VMAXP]) {

    int i, j;
    for (i=0; i<numP; i++) {
        for (j=0; j<VMAXP; j++) {
            c[i][j]  = 0.5 * (double)ic[i][j];
            cx[i][j] = 0.5 * (double)icx[i][j];
            cy[i][j] = 0.5 * (double)icy[i][j];
            cz[i][j] = 0.5 * (double)icz[i][j];
        }
    }
}

VPUBLIC int Vfetk_PDE_simplexBasisInit(int key, int
    dim, int comp, int *ndof,
  int dof[]) {

    int qorder, bump, dimIS[VAPBS_NVS];

    /* necessary quadrature order to return at the end */
    qorder = P_DEG;

    /* deal with bump function requests */
    if ((key == 0) || (key == 1)) {
        bump = 0;
    } else if ((key == 2) || (key == 3)) {
        bump = 1;
    } else { VASSERT(0); }

    /* for now use same element for all components, both trial and test */
    if (dim==2) {
        /* 2D simplex dimensions */
        dimIS[0] = 3;  /* number of vertices             */
        dimIS[1] = 3;  /* number of edges                */
        dimIS[2] = 0;  /* number of faces (3D only)      */
        dimIS[3] = 1;  /* number of simplices (always=1) */
        if (bump==0) {
            if (P_DEG==1) {
                init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
            } else if (P_DEG==2) {
                init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
            } else if (P_DEG==3) {
                init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
            } else Vnm_print(2, "..bad order..");
        } else if (bump==1) {
```

```
            if (P_DEG==1) {
                init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
            } else Vnm_print(2, "..bad order..");
        } else Vnm_print(2, "..bad bump..");
    } else if (dim==3) {
        /* 3D simplex dimensions */
        dimIS[0] = 4;  /* number of vertices             */
        dimIS[1] = 6;  /* number of edges                */
        dimIS[2] = 4;  /* number of faces (3D only)      */
        dimIS[3] = 1;  /* number of simplices (always=1) */
        if (bump==0) {
            if (P_DEG==1) {
                init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
            } else if (P_DEG==2) {
                init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
            } else if (P_DEG==3) {
                init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
            } else Vnm_print(2, "..bad order..");
        } else if (bump==1) {
            if (P_DEG==1) {
                init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
            } else Vnm_print(2, "..bad order..");
        } else Vnm_print(2, "..bad bump..");
    } else Vnm_print(2, "..bad dimension..");

    /* save number of DF */
    numP = *ndof;

    /* return the required quarature order */
    return qorder;
}

VPUBLIC void Vfetk_PDE_simplexBasisForm(int key, int dim, int comp, int pdkey,
  double xq[], double basis[]) {

    if (pdkey == 0) {
        polyEval(numP, basis, c, xq);
    } else if (pdkey == 1) {
        polyEval(numP, basis, cx, xq);
    } else if (pdkey == 2) {
        polyEval(numP, basis, cy, xq);
    } else if (pdkey == 3) {
        polyEval(numP, basis, cz, xq);
    } else { VASSERT(0); }
}

VPRIVATE void init_2DP1(int dimIS[], int *ndof, int dof[], double c[][VMAXP],
  double cx[][VMAXP], double cy[][VMAXP], double cz[][VMAXP]) {

    int i;

    /* dof number and locations */
    dof[0] = 1;
    dof[1] = 0;
    dof[2] = 0;
    dof[3] = 0;
    *ndof  = 0;
    for (i=0; i<VAPBS_NVS; i++) *ndof += dimIS[i] * dof[i];
    VASSERT( *ndof == dim_2DP1 );
    VASSERT( *ndof <= VMAXP );

    /* coefficients of the polynomials */
    setCoef( *ndof, c, cx, cy, cz, lgr_2DP1, lgr_2DP1x, lgr_2DP1y, lgr_2DP1z );
}

VPRIVATE void init_3DP1(int dimIS[], int *ndof, int dof[], double c[][VMAXP],
  double cx[][VMAXP], double cy[][VMAXP], double cz[][VMAXP]) {

    int i;

    /* dof number and locations */
    dof[0] = 1;
    dof[1] = 0;
    dof[2] = 0;
    dof[3] = 0;
    *ndof  = 0;
    for (i=0; i<VAPBS_NVS; i++) *ndof += dimIS[i] * dof[i];
    VASSERT( *ndof == dim_3DP1 );
    VASSERT( *ndof <= VMAXP );

```

```
    /* coefficients of the polynomials */
    setCoef( *ndof, c, cx, cy, cz, lgr_3DP1, lgr_3DP1x, lgr_3DP1y, lgr_3DP1z );
}

VPUBLIC void Vfetk_dumpLocalVar() {

    int i;

    Vnm_print(1, "DEBUG: nvec = (%g, %g, %g)\n", var.nvec[0], var.nvec[1],
      var.nvec[2]);
    Vnm_print(1, "DEBUG: nverts = %d\n", var.nverts);
    for (i=0; i<var.nverts; i++) {
        Vnm_print(1, "DEBUG: verts[%d] ID = %d\n", i, VV_id(var.verts[i]));
        Vnm_print(1, "DEBUG: vx[%d] = (%g, %g, %g)\n", i, var.vx[i][0],
          var.vx[i][1], var.vx[i][2]);
    }
    Vnm_print(1, "DEBUG: simp ID = %d\n", SS_id(var.simp));
    Vnm_print(1, "DEBUG: sType = %d\n", var.sType);
    Vnm_print(1, "DEBUG: fType = %d\n", var.fType);
    Vnm_print(1, "DEBUG: xq = (%g, %g, %g)\n", var.xq[0], var.xq[1], var.xq[2]);
    Vnm_print(1, "DEBUG: U[0] = %g\n", var.U[0]);
    Vnm_print(1, "DEBUG: dU[0] = (%g, %g, %g)\n", var.dU[0][0], var.dU[0][1],
      var.dU[0][2]);
    Vnm_print(1, "DEBUG: W = %g\n", var.W);
    Vnm_print(1, "DEBUG: d2W = %g\n", var.d2W);
    Vnm_print(1, "DEBUG: dW = (%g, %g, %g)\n", var.dW[0], var.dW[1], var.dW[2]);
    Vnm_print(1, "DEBUG: diel = %g\n", var.diel);
    Vnm_print(1, "DEBUG: ionacc = %g\n", var.ionacc);
    Vnm_print(1, "DEBUG: A = %g\n", var.A);
    Vnm_print(1, "DEBUG: F = %g\n", var.F);
    Vnm_print(1, "DEBUG: B = %g\n", var.B);
    Vnm_print(1, "DEBUG: DB = %g\n", var.DB);
    Vnm_print(1, "DEBUG: nion = %d\n", var.nion);
    for (i=0; i<var.nion; i++) {
        Vnm_print(1, "DEBUG: ionConc[%d] = %g\n", i, var.ionConc[i]);
        Vnm_print(1, "DEBUG: ionQ[%d] = %g\n", i, var.ionQ[i]);
        Vnm_print(1, "DEBUG: ionRadii[%d] = %g\n", i, var.ionRadii[i]);
    }
    Vnm_print(1, "DEBUG: zkappa2 = %g\n", var.zkappa2);
    Vnm_print(1, "DEBUG: zks2 = %g\n", var.zks2);
    Vnm_print(1, "DEBUG: Fu_v = %g\n", var.Fu_v);
    Vnm_print(1, "DEBUG: DFu_wv = %g\n", var.DFu_wv);
    Vnm_print(1, "DEBUG: delta = %g\n", var.delta);
    Vnm_print(1, "DEBUG: u_D = %g\n", var.u_D);
    Vnm_print(1, "DEBUG: u_T = %g\n", var.u_T);

};

VPUBLIC int Vfetk_fillArray(Vfetk *thee, Bvec *vec, Vdata_Type type) {

    int i, j, ichop;
    double coord[3], chi, q, conc, val;
    VV *vert;
    Bvec *u, *u_d;
    AM *am;
    Gem *gm;
    PBEparm *pbeparm;
    Vacc *acc;
    Vpbe *pbe;

    gm = thee->gm;
    am = thee->am;
    pbe = thee->pbe;
    pbeparm = thee->pbeparm;
    acc = pbe->acc;

    /* Make sure vec has enough rows to accomodate the vertex data */
    if (Bvec_numRB(vec, 0) != Gem_numVV(gm)) {
        Vnm_print(2, "Vfetk_fillArray:  insufficient space in Bvec!\n");
        Vnm_print(2, "Vfetk_fillArray:  Have %d, need %d!\n", Bvec_numRB(vec, 0),
          Gem_numVV(gm));
        return 0;
    }

```

```
    switch (type) {

        case VDT_CHARGE:
            Vnm_print(2, "Vfetk_fillArray:  can't write out charge distribution!\n");
            return 0;
            break;

        case VDT_POT:
            u = am->u;
            u_d = am->ud;
            /* Copy in solution */
            Bvec_copy(vec, u);
            /* Add dirichlet condition */
            Bvec_axpy(vec, u_d, 1.0);
            break;

        case VDT_SMOL:
            for (i=0; i<Gem_numVV(gm); i++) {
                vert = Gem_VV(gm, i);
                for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
                chi = Vacc_molAcc(acc, coord, pbe->solventRadius);
                Bvec_set(vec, i, chi);
            }
            break;

        case VDT_SSPL:
            for (i=0; i<Gem_numVV(gm); i++) {
                vert = Gem_VV(gm, i);
                for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
                chi = Vacc_splineAcc(acc, coord, pbeparm->swin, 0.0);
                Bvec_set(vec, i, chi);
            }
            break;

        case VDT_VDW:
            for (i=0; i<Gem_numVV(gm); i++) {
                vert = Gem_VV(gm, i);
                for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
                chi = Vacc_vdwAcc(acc, coord);
                Bvec_set(vec, i, chi);
            }
            break;

        case VDT_IVDW:
            for (i=0; i<Gem_numVV(gm); i++) {
                vert = Gem_VV(gm, i);
                for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
                chi = Vacc_ivdwAcc(acc, coord, pbe->maxIonRadius);
                Bvec_set(vec, i, chi);
            }
            break;

        case VDT_LAP:
            Vnm_print(2, "Vfetk_fillArray:  can't write out Laplacian!\n");
            return 0;
            break;

        case VDT_EDENS:
            Vnm_print(2, "Vfetk_fillArray:  can't write out energy density!\n");
            return 0;
            break;

        case VDT_NDENS:
            u = am->u;
            u_d = am->ud;
            /* Copy in solution */
            Bvec_copy(vec, u);
            /* Add dirichlet condition */
            Bvec_axpy(vec, u_d, 1.0);
            /* Load up ions */
            ichop = 0;
            for (i=0; i<Gem_numVV(gm); i++) {
                val = 0;
                for (j=0; j<pbe->numIon; j++) {
                    q = pbe->ionQ[j];
                    conc = pbe->ionConc[j];
```

```
                    if (thee->type == PBE_NPBE || thee->type == PBE_SMPBE /* SMPBE Added */) {
                        val += (conc*Vcap_exp(-q*Bvec_val(vec, i), &ichop));
                    } else if (thee->type == PBE_LPBE) {
                        val += (conc * ( 1 - q*Bvec_val(vec, i)));
                    }
                }
                Bvec_set(vec, i, val);
            }
            break;

        case VDT_QDENS:
            u = am->u;
            u_d = am->ud;
            /* Copy in solution */
            Bvec_copy(vec, u);
            /* Add dirichlet condition */
            Bvec_axpy(vec, u_d, 1.0);
            /* Load up ions */
            ichop = 0;
            for (i=0; i<Gem_numVV(gm); i++) {
                val = 0;
                for (j=0; j<pbe->numIon; j++) {
                    q = pbe->ionQ[j];
                    conc = pbe->ionConc[j];
                    if (thee->type == PBE_NPBE || thee->type == PBE_SMPBE /* SMPBE Added */) {
                        val += (q*conc*Vcap_exp(-q*Bvec_val(vec, i), &ichop));
                    } else if (thee->type == PBE_LPBE) {
                        val += (q*conc*(1 - q*Bvec_val(vec, i)));
                    }
                }
                Bvec_set(vec, i, val);
            }
            break;

        case VDT_DIELX:
            Vnm_print(2, "Vfetk_fillArray:  can't write out x-shifted diel!\n");
            return 0;
            break;

        case VDT_DIELY:
            Vnm_print(2, "Vfetk_fillArray:  can't write out y-shifted diel!\n");
            return 0;
            break;

        case VDT_DIELZ:
            Vnm_print(2, "Vfetk_fillArray:  can't write out z-shifted diel!\n");
            return 0;
            break;

        case VDT_KAPPA:
            Vnm_print(2, "Vfetk_fillArray:  can't write out kappa!\n");
            return 0;
            break;

        default:
            Vnm_print(2, "Vfetk_fillArray:  invalid data type (%d)!\n", type);
            return 0;
            break;
    }

    return 1;
}

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

    int i, j, ichop;
    Aprx *aprx;
    Gem *gm;
    Vio *sock;

    VASSERT(thee != VNULL);
```

```
    aprx = thee->aprx;
    gm = thee->gm;

    sock = Vio_ctor(iodev,iofmt,thost,fname,"w");
    if (sock == VNULL) {
        Vnm_print(2, "Vfetk_write: Problem opening virtual socket %s\n",
          fname);
        return 0;
    }
    if (Vio_connect(sock, 0) < 0) {
        Vnm_print(2, "Vfetk_write: Problem connecting to virtual socket %s\n",
          fname);
        return 0;
    }

    /* Make sure vec has enough rows to accomodate the vertex data */
    if (Bvec_numRB(vec, 0) != Gem_numVV(gm)) {
        Vnm_print(2, "Vfetk_fillArray:  insufficient space in Bvec!\n");
        Vnm_print(2, "Vfetk_fillArray:  Have %d, need %d!\n", Bvec_numRB(vec, 0
    ),
          Gem_numVV(gm));
        return 0;
    }

    switch (format) {

        case VDF_DX:
            Aprx_writeSOL(aprx, sock, vec, "DX");
            break;
        case VDF_AVS:
            Aprx_writeSOL(aprx, sock, vec, "UCD");
            break;
        case VDF_UHBD:
            Vnm_print(2, "Vfetk_write:  UHBD format not supported!\n");
            return 0;
        default:
            Vnm_print(2, "Vfetk_write:  Invalid data format (%d)!\n", format);
            return 0;
    }


    Vio_connectFree(sock);
    Vio_dtor(&sock);

    return 1;
}

#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 1776 2012-07-19 22:29:10Z 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 vpee.c.

9.20 vpee.c

```
00001
00058 #include "apbscfg.h"
00059
```

```
#if defined(HAVE_MC_H)
#if defined(HAVE_MCX_H)

#include "mc/mc.h"
#include "apbs/vpee.h"

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
                                    );

/* ////////////////////////////////////////////////////////////////////////
// Class Vpee: Non-inlineable methods

/* ////////////////////////////////////////////////////////////////////////
// Routine:  Vpee_ctor
//
// Author:   Nathan Baker
VPUBLIC Vpee* Vpee_ctor(Gem *gm,
                        int localPartID,
                        int killFlag,
                        double killParam
                        ) {

    Vpee *thee = VNULL;

    /* Set up the structure */
    thee = Vmem_malloc(VNULL, 1, sizeof(Vpee) );
    VASSERT( thee != VNULL);
    VASSERT( Vpee_ctor2(thee, gm, localPartID, killFlag
, killParam));

    return thee;
}

/* ////////////////////////////////////////////////////////////////////////
// Routine:  Vpee_ctor2
//
// Author:   Nathan Baker
VPUBLIC int Vpee_ctor2(Vpee *thee,
                       Gem *gm,
                       int localPartID,
                       int killFlag,
                       double killParam
                       ) {

    int ivert,
        nLocalVerts;
    SS *simp;
    VV *vert;
    double radius,
           dx,
           dy,
           dz;

    VASSERT(thee != VNULL);

    /* Sanity check on input values */
    if (killFlag == 0) {
        Vnm_print(0, "Vpee_ctor2: No error attenuation outside partition.\n");
    } else if (killFlag == 1) {
```

```
        Vnm_print(0, "Vpee_ctor2: Error outside local partition ignored.\n");
    } else if (killFlag == 2) {
        Vnm_print(0, "Vpee_ctor2: Error ignored outside sphere with radius %4.3f times the radius of the circumscribing sphere\n", killParam);
        if (killParam < 1.0) {
          Vnm_print(2, "Vpee_ctor2: Warning! Parameter killParam = %4.3 < 1.0!\n",
            killParam);
          Vnm_print(2, "Vpee_ctor2: This may result in non-optimal marking and refinement!\n");
        }
    } else if (killFlag == 3) {
        Vnm_print(0, "Vpee_ctor2: Error outside local partition and immediate neighbors ignored [NOT IMPLEMENTED].\n");
    } else {
        Vnm_print(2, "Vpee_ctor2: UNRECOGNIZED killFlag PARAMETER! BAILING!.\n");
        VASSERT(0);
    }

    thee->gm = gm;
    thee->localPartID = localPartID;
    thee->killFlag = killFlag;
    thee->killParam = killParam;
    thee->mem = Vmem_ctor("APBS::VPEE");

    /* Now, figure out the center of geometry for the local partition.  The
     * general plan is to loop through the vertices, loop through the
     * vertices' simplex lists and find the vertices with simplices containing
     * chart values we're interested in. */
    thee->localPartCenter[0] = 0.0;
    thee->localPartCenter[1] = 0.0;
    thee->localPartCenter[2] = 0.0;
    nLocalVerts = 0;
    for (ivert=0; ivert<Gem_numVV(thee->gm); ivert++) {
        vert = Gem_VV(thee->gm, ivert);
        simp = VV_firstSS(vert);
        VASSERT(simp != VNULL);
        while (simp != VNULL) {
            if (SS_chart(simp) == thee->localPartID) {
                thee->localPartCenter[0] += VV_coord(vert, 0);
                thee->localPartCenter[1] += VV_coord(vert, 1);
                thee->localPartCenter[2] += VV_coord(vert, 2);
                nLocalVerts++;
                break;
            }
            simp = SS_link(simp, vert);
        }
    }
    VASSERT(nLocalVerts > 0);
    thee->localPartCenter[0] =
      thee->localPartCenter[0]/((double)(nLocalVerts));
    thee->localPartCenter[1] =
      thee->localPartCenter[1]/((double)(nLocalVerts));
    thee->localPartCenter[2] =
      thee->localPartCenter[2]/((double)(nLocalVerts));
    Vnm_print(0, "Vpee_ctor2: Part %d centered at (%4.3f, %4.3f, %4.3f)\n",
      thee->localPartID, thee->localPartCenter[0], thee->localPartCenter[1],
      thee->localPartCenter[2]);


    /* Now, figure out the radius of the sphere circumscribing the local
     * partition.  We need to keep track of vertices so we don't double count
     * them. */
    thee->localPartRadius = 0.0;
    for (ivert=0; ivert<Gem_numVV(thee->gm); ivert++) {
        vert = Gem_VV(thee->gm, ivert);
        simp = VV_firstSS(vert);
        VASSERT(simp != VNULL);
        while (simp != VNULL) {
            if (SS_chart(simp) == thee->localPartID) {
                dx = thee->localPartCenter[0] - VV_coord(vert, 0);
                dy = thee->localPartCenter[1] - VV_coord(vert, 1);
                dz = thee->localPartCenter[2] - VV_coord(vert, 2);
                radius = dx*dx + dy*dy + dz*dz;
                if (radius > thee->localPartRadius) thee->localPartRadius =
                  radius;
                break;
            }
            simp = SS_link(simp, vert);
        }
```

```
    }
    thee->localPartRadius = VSQRT(thee->localPartRadius);
    Vnm_print(0, "Vpee_ctor2: Part %d has circumscribing sphere of radius %4.3f
\n",
      thee->localPartID, thee->localPartRadius);

    return 1;
}

/* ////////////////////////////////////////////////////////////////////////
// Routine:  Vpee_dtor
//
// Author:   Nathan Baker
VPUBLIC void Vpee_dtor(Vpee **thee) {

    if ((*thee) != VNULL) {
        Vpee_dtor2(*thee);
        Vmem_free(VNULL, 1, sizeof(Vpee), (void **)thee);
        (*thee) = VNULL;
    }

}

/* ////////////////////////////////////////////////////////////////////////
// Routine:  Vpee_dtor2
//
// Author:   Nathan Baker
VPUBLIC void Vpee_dtor2(Vpee *thee) {
    Vmem_dtor(&(thee->mem));
}

/* ////////////////////////////////////////////////////////////////////////
// Routine:  Vpee_markRefine
//
// Author:   Nathan Baker (and Michael Holst: the author of AM_markRefine, on
//           which this is based)
VPUBLIC int Vpee_markRefine(Vpee *thee,
                            AM *am,
                            int level,
                            int akey,
                            int rcol,
                            double etol,
                            int bkey
                            ) {

    Aprx *aprx;
    int marked = 0,
        markMe,
        i,
        smid,
        count,
        currentQ;
    double minError = 0.0,
           maxError = 0.0,
           errEst = 0.0,
           mlevel,
           barrier;
    SS *sm;


    VASSERT(thee != VNULL);

    /* Get the Aprx object from AM */
    aprx = am->aprx;

    /* input check and some i/o */
    if ( ! ((-1 <= akey) && (akey <= 4)) ) {
        Vnm_print(0,"Vpee_markRefine: bad refine key; simplices marked = %d\n",
            marked);
        return marked;
    }

    /* For uniform markings, we have no effect */
    if ((-1 <= akey) && (akey <= 0)) {
        marked = Gem_markRefine(thee->gm, akey, rcol);
        return marked;
    }

    /* Informative I/O */
    if (akey == 2) {
        Vnm_print(0,"Vpee_estRefine: using Aprx_estNonlinResid().\n");
```

```
    } else if (akey == 3) {
        Vnm_print(0,"Vpee_estRefine: using Aprx_estLocalProblem().\n");
    } else if (akey == 4) {
        Vnm_print(0,"Vpee_estRefine: using Aprx_estDualProblem().\n");
    } else {
        Vnm_print(0,"Vpee_estRefine: bad key given; simplices marked = %d\n",
            marked);
        return marked;
    }
    if (thee->killFlag == 0) {
        Vnm_print(0, "Vpee_markRefine: No error attenuation -- simplices in all partitions will be marked.\n");
    } else if (thee->killFlag == 1) {
        Vnm_print(0, "Vpee_markRefine: Maximum error attenuation -- only simplices in local partition will be marked.\n");
    } else if (thee->killFlag == 2) {
        Vnm_print(0, "Vpee_markRefine: Spherical error attenutation -- simplices within a sphere of %4.3f times the size of the partition will be marked\n",
          thee->killParam);
    } else if (thee->killFlag == 2) {
        Vnm_print(0, "Vpee_markRefine: Neighbor-based error attenuation -- simplices in the local and neighboring partitions will be marked [NOT IMPLEMENTED]!\n");
        VASSERT(0);
    } else {
        Vnm_print(2,"Vpee_markRefine: bogus killFlag given; simplices marked = %d\n",
            marked);
        return marked;
    }

    /* set the barrier type */
    mlevel = (etol*etol) / Gem_numSS(thee->gm);
    if (bkey == 0) {
        barrier = (etol*etol);
        Vnm_print(0,"Vpee_estRefine: forcing [err per S] < [TOL] = %g\n",
            barrier);
    } else if (bkey == 1) {
        barrier = mlevel;
        Vnm_print(0,"Vpee_estRefine: forcing [err per S] < [(TOL^2/numS)^{1/2}] = %g\n",
            VSQRT(barrier));
    } else {
        Vnm_print(0,"Vpee_estRefine: bad bkey given; simplices marked = %d\n",
            marked);
        return marked;
    }

    /* timer */
    Vnm_tstart(30, "error estimation");

    /* count = num generations to produce from marked simplices (minimally) */
    count = 1; /* must be >= 1 */

    /* check the refinement Q for emptyness */
    currentQ = 0;
    if (Gem_numSQ(thee->gm,currentQ) > 0) {
        Vnm_print(0,"Vpee_markRefine: non-empty refinement Q%d....clearing..",
            currentQ);
        Gem_resetSQ(thee->gm,currentQ);
        Vnm_print(0,"..done.\n");
    }
    if (Gem_numSQ(thee->gm,!currentQ) > 0) {
        Vnm_print(0,"Vpee_markRefine: non-empty refinement Q%d....clearing..",
            !currentQ);
        Gem_resetSQ(thee->gm,!currentQ);
        Vnm_print(0,"..done.\n");
    }
    VASSERT( Gem_numSQ(thee->gm,currentQ)  == 0 );
    VASSERT( Gem_numSQ(thee->gm,!currentQ) == 0 );

    /* clear everyone's refinement flags */
    Vnm_print(0,"Vpee_markRefine: clearing all simplex refinement flags..");
    for (i=0; i<Gem_numSS(thee->gm); i++) {
        if ( (i>0) && (i % VPRTKEY) == 0 ) Vnm_print(0,"[MS:%d]",i);
        sm = Gem_SS(thee->gm,i);
        SS_setRefineKey(sm,currentQ,0);
        SS_setRefineKey(sm,!currentQ,0);
        SS_setRefinementCount(sm,0);
    }
    Vnm_print(0,"..done.\n");
```

```
    /* NON-ERROR-BASED METHODS */
    /* Simplex flag clearing */
    if (akey == -1) return marked;
    /* Uniform & user-defined refinement*/
    if ((akey == 0) || (akey == 1)) {
        smid = 0;
        while ( smid < Gem_numSS(thee->gm)) {
            /* Get the simplex and find out if it's markable */
            sm = Gem_SS(thee->gm,smid);
            markMe = Vpee_ourSimp(thee, sm, rcol);
            if (markMe) {
                if (akey == 0) {
                    marked++;
                    Gem_appendSQ(thee->gm,currentQ, sm);
                    SS_setRefineKey(sm,currentQ,1);
                    SS_setRefinementCount(sm,count);
                } else if (Vpee_userDefined(thee, sm)) {
                    marked++;
                    Gem_appendSQ(thee->gm,currentQ, sm);
                    SS_setRefineKey(sm,currentQ,1);
                    SS_setRefinementCount(sm,count);
                }
            }
            smid++;
        }
    }

    /* ERROR-BASED METHODS */
    /* gerror = global error accumulation */
    aprx->gerror = 0.;

    /* traverse the simplices and process the error estimates */
    Vnm_print(0,"Vpee_markRefine: estimating error..");
    smid = 0;
    while ( smid < Gem_numSS(thee->gm)) {

        /* Get the simplex and find out if it's markable */
        sm = Gem_SS(thee->gm,smid);
        markMe = Vpee_ourSimp(thee, sm, rcol);

        if ( (smid>0) && (smid % VPRTKEY) == 0 ) Vnm_print(0,"[MS:%d]",smid);

        /* Produce an error estimate for this element if it is in the set */
        if (markMe) {
            if (akey == 2) {
                errEst = Aprx_estNonlinResid(aprx, sm, am->u,am->ud,am->f);
            } else if (akey == 3) {
                errEst = Aprx_estLocalProblem(aprx, sm, am->u,am->ud,am->f);
            } else if (akey == 4) {
                errEst = Aprx_estDualProblem(aprx, sm, am->u,am->ud,am->f);
            }
            VASSERT( errEst >= 0. );

            /* if error estimate above tol, mark element for refinement */
            if ( errEst > barrier ) {
                marked++;
                Gem_appendSQ(thee->gm,currentQ, sm); /*add to refinement Q*/
                SS_setRefineKey(sm,currentQ,1);      /* note now on refine Q */
                SS_setRefinementCount(sm,count);     /* refine X many times? */
            }

            /* keep track of min/max errors over the mesh */
            minError = VMIN2( VSQRT(VABS(errEst)), minError );
            maxError = VMAX2( VSQRT(VABS(errEst)), maxError );

            /* store the estimate */
            Bvec_set( aprx->wev, smid, errEst );

            /* accumlate into global error (errEst is SQUAREd already) */
            aprx->gerror += errEst;

        /* otherwise store a zero for the estimate */
        } else {
            Bvec_set( aprx->wev, smid, 0. );
        }

        smid++;
    }

    /* do some i/o */
```

```
    Vnm_print(0,"..done.  [marked=<%d/%d>]\n",marked,Gem_numSS(thee->gm));
    Vnm_print(0,"Vpee_estRefine: TOL=<%g>  Global_Error=<%g>\n",
        etol, aprx->gerror);
    Vnm_print(0,"Vpee_estRefine: (TOL^2/numS)^{1/2}=<%g>  Max_Ele_Error=<%g>\n"
,
        VSQRT(mlevel),maxError);
    Vnm_tstop(30, "error estimation");

    /* check for making the error tolerance */
    if ((bkey == 1) && (aprx->gerror <= etol)) {
        Vnm_print(0,
            "Vpee_estRefine: *******************************************\n");
        Vnm_print(0,
            "Vpee_estRefine: Global Error criterion met; setting marked=0.\n");
        Vnm_print(0,
            "Vpee_estRefine: *******************************************\n");
        marked = 0;
    }


    /* return */
    return marked;

}

/* ///////////////////////////////////////////////////////////////////////
// Routine:  Vpee_numSS
//
// Author:   Nathan Baker
VPUBLIC int Vpee_numSS(Vpee *thee) {
    int num = 0;
    int isimp;

    for (isimp=0; isimp<Gem_numSS(thee->gm); isimp++) {
        if (SS_chart(Gem_SS(thee->gm, isimp)) == thee->localPartID) num++;
    }

    return num;
}

/* ///////////////////////////////////////////////////////////////////////
// Routine:  Vpee_userDefined
//
// Purpose:  Reduce code bloat by wrapping up the common steps for getting the
//           user-defined error estimate
//
// Author:   Nathan Baker
VPRIVATE int Vpee_userDefined(Vpee *thee,
                              SS *sm
                              ) {

    int ivert,
        icoord,
        chart[4],
        fType[4],
        vType[4];
    double vx[4][3];

    for (ivert=0; ivert<Gem_dimVV(thee->gm); ivert++) {
        fType[ivert] = SS_faceType(sm,ivert);
        vType[ivert] = VV_type(SS_vertex(sm,ivert) );
        chart[ivert] = VV_chart(SS_vertex(sm,ivert) );
        for (icoord=0; icoord<Gem_dimII(thee->gm); icoord++) {
            vx[ivert][icoord] = VV_coord(SS_vertex(sm,ivert), icoord );
        }
    }
    return thee->gm->pde->markSimplex(Gem_dim(thee->gm), Gem_dimII(thee->gm),
             SS_type(sm), fType, vType, chart, vx, sm);
}

/* ///////////////////////////////////////////////////////////////////////
// Routine:  Vpee_ourSimp
//
// Purpose:  Reduce code bloat by wrapping up the common steps for determining
//           whether the given simplex can be marked (i.e., belongs to our
//           partition or overlap region)
//
// Returns:  1 if could be marked, 0 otherwise
//
// Author:   Nathan Baker
VPRIVATE int Vpee_ourSimp(Vpee *thee,
```

```
                          SS *sm,
                          int rcol
                          ) {

    int ivert;
    double dist,
           dx,
           dy,
           dz;

    if (thee->killFlag == 0) return 1;
    else if (thee->killFlag == 1) {
        if ((SS_chart(sm) == rcol) || (rcol < 0)) return 1;
    } else if (thee->killFlag == 2) {
        if (rcol < 0) return 1;
        else {
            /* We can only do distance-based searches on the local partition */
            VASSERT(rcol == thee->localPartID);
            /* Find the closest distance between this simplex and the
             * center of the local partition and check it against
             * (thee->localPartRadius*thee->killParam) */
            dist = 0;
            for (ivert=0; ivert<SS_dimVV(sm); ivert++) {
                dx = VV_coord(SS_vertex(sm, ivert), 0) -
                  thee->localPartCenter[0];
                dy = VV_coord(SS_vertex(sm, ivert), 1) -
                  thee->localPartCenter[1];
                dz = VV_coord(SS_vertex(sm, ivert), 2) -
                  thee->localPartCenter[2];
                dist = VSQRT((dx*dx + dy*dy + dz*dz));
            }
            if (dist < thee->localPartRadius*thee->killParam)
    return 1;
        }
    } else if (thee->killFlag == 3) VASSERT(0);
    else VASSERT(0);

    return 0;

}

#endif
#endif
```

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

Contains declarations for class APOLparm.

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

Include dependency graph for femparm.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct sFEMparm

  Parameter structure for FEM-specific variables from input files.

Typedefs

- typedef enum eFEMparm_EtolType FEMparm_EtolType

  Declare FEparm_EtolType type.
- typedef enum eFEMparm_EstType FEMparm_EstType

Declare FEMparm_EstType type.

- typedef enum eFEMparm_CalcType FEMparm_CalcType

  Declare FEMparm_CalcType type.
- typedef struct sFEMparm FEMparm

  Declaration of the FEMparm class as the FEMparm structure.

Enumerations

- enum eFEMparm_EtolType { FET_SIMP = 0, FET_GLOB = 1, FET_FRAC = 2 }

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

  Adaptive refinment error estimator method.
- enum eFEMparm_CalcType { FCT_MANUAL, FCT_NONE }

  Calculation type.

Functions

- VEXTERNC FEMparm ∗ FEMparm_ctor (FEMparm_CalcType type)

  Construct FEMparm.
- VEXTERNC int FEMparm_ctor2 (FEMparm ∗thee, FEMparm_CalcType type)

  FORTRAN stub to construct FEMparm.
- VEXTERNC void FEMparm_dtor (FEMparm ∗∗thee)

  Object destructor.
- VEXTERNC void FEMparm_dtor2 (FEMparm ∗thee)

  FORTRAN stub for object destructor.
- VEXTERNC int FEMparm_check (FEMparm ∗thee)

  Consistency check for parameter values stored in object.
- VEXTERNC void FEMparm_copy (FEMparm ∗thee, FEMparm ∗source)

  Copy target object into thee.
- VEXTERNC Vrc_Codes FEMparm_parseToken (FEMparm ∗thee, char tok[VMAX_BUFSIZE], Vio ∗sock)

  Parse an MG keyword from an input file.

9.21.1 Detailed Description

Contains declarations for class APOLparm. Contains declarations for class FEMparm.

Version

Id:

apolparm.h 1774 2012-07-19 18:46:14Z sobolevnrm

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 1774 2012-07-19 18:46:14Z sobolevnrm

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_EtolType {
00077     FET_SIMP=0,
00078     FET_GLOB=1,
00079     FET_FRAC=2
00080 };
```

```
00081
00087 typedef enum eFEMparm_EtolType FEMparm_EtolType
      ;
00088
00095 enum eFEMparm_EstType {
00096     FRT_UNIF=0,
00097     FRT_GEOM=1,
00098     FRT_RESI=2,
00099     FRT_DUAL=3,
00101     FRT_LOCA=4
00102 };
00103
00108 typedef enum eFEMparm_EstType FEMparm_EstType;
00109
00114 enum eFEMparm_CalcType {
00115     FCT_MANUAL,
00116  FCT_NONE
00117 };
00118
00123 typedef enum eFEMparm_CalcType FEMparm_CalcType
      ;
00124
00130 struct sFEMparm {
00131
00132     int parsed;
00135     FEMparm_CalcType type;
00136     int settype;
00137     double glen[3];
00138     int setglen;
00139     double etol;
00141     int setetol;
00142     FEMparm_EtolType ekey;
00144     int setekey;
00145     FEMparm_EstType akeyPRE;
00148     int setakeyPRE;
00149     FEMparm_EstType akeySOLVE;
00151     int setakeySOLVE;
00152     int targetNum;
00156     int settargetNum;
00157     double targetRes;
00161     int settargetRes;
00162     int maxsolve;
00163     int setmaxsolve;
00164     int maxvert;
00166     int setmaxvert;
00167  int pkey;
00170  int useMesh;
00171  int meshID;
00173 };
00174
00179 typedef struct sFEMparm FEMparm;
00180
00181 /* ////////////////////////////////////////////////////////////////////////
00182 // Class NOsh: Non-inlineable methods (nosh.c)
00184
00191 VEXTERNC FEMparm* FEMparm_ctor(FEMparm_CalcType type);
00192
00200 VEXTERNC int FEMparm_ctor2(FEMparm *thee, FEMparm_CalcType type);
00201
00207 VEXTERNC void FEMparm_dtor(FEMparm **thee);
00208
00214 VEXTERNC void FEMparm_dtor2(FEMparm *thee);
00215
00223 VEXTERNC int FEMparm_check(FEMparm *thee);
00224
00231 VEXTERNC void FEMparm_copy(FEMparm *thee, FEMparm *source);
00232
00243 VEXTERNC Vrc_Codes FEMparm_parseToken(FEMparm *thee, char tok[VMAX_BUFSIZE],
00244   Vio *sock);
00245
00246 #endif
00247
```

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

Contains declarations for class MGparm.

```
#include "maloc/maloc.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 1774 2012-07-19 18:46:14Z sobolevnrm

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 "maloc/maloc.h"
00068 #include "apbs/vhal.h"
00069 #include "apbs/vstring.h"
00070
00075 enum eMGparm_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 eMGparm_CalcType MGparm_CalcType;
00088
00093 enum eMGparm_CentMeth {
00094     MCM_POINT=0,
00095     MCM_MOLECULE=1,
00096  MCM_FOCUS=2
00097 };
00098
00103 typedef enum eMGparm_CentMeth MGparm_CentMeth;
00112 struct sMGparm {
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];
```

```
    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;
    /* ********* TYPE 2 PARAMETERS (PARALLEL AUTO-FOCUS) ******** */
    double partDisjCenter[3];
    double partDisjLength[3];
    int partDisjOwnSide[6];
    int pdime[3];
    int setpdime;
    int proc_rank;
    int setrank;
    int proc_size;
    int setsize;
    double ofrac;
    int setofrac;
    int async;
    int setasync;
 int nonlintype;
 int setnonlintype;
 int method;
 int setmethod;
 int useAqua;
 int setUseAqua;
};

typedef struct sMGparm MGparm;

VEXTERNC int MGparm_getNx(MGparm *thee);

VEXTERNC int MGparm_getNy(MGparm *thee);

VEXTERNC int MGparm_getNz(MGparm *thee);

VEXTERNC double MGparm_getHx(MGparm *thee);

VEXTERNC double MGparm_getHy(MGparm *thee);

VEXTERNC double MGparm_getHz(MGparm *thee);

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

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

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

VEXTERNC double MGparm_getCenterX(MGparm *thee);

VEXTERNC double MGparm_getCenterY(MGparm *thee);

VEXTERNC double MGparm_getCenterZ(MGparm *thee);

VEXTERNC MGparm*  MGparm_ctor(MGparm_CalcType type);

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

VEXTERNC void     MGparm_dtor(MGparm **thee);

VEXTERNC void     MGparm_dtor2(MGparm *thee);

VEXTERNC Vrc_Codes      MGparm_check(MGparm *thee);

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

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

#endif
```

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

Contains declarations for class NOsh.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/pbeparm.h"
#include "apbs/mgparm.h"
#include "apbs/femparm.h"
#include "apbs/apolparm.h"
#include "apbs/valist.h"
```

Include dependency graph for nosh.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct sNOsh_calc

  Calculation class for use when parsing fixed format input files.
- struct sNOsh

  Class for parsing fixed format input files.

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)

  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.

9.25.1 Detailed Description

Contains declarations for class NOsh.

Version

Id:

nosh.h 1774 2012-07-19 18:46:14Z sobolevnrm

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
00078 #define NOSH_MAXMOL 20
00079
00082 #define NOSH_MAXCALC 20
00083
00086 #define NOSH_MAXPRINT 20
00087
00090 #define NOSH_MAXPOP 20
00091
00096 enum eNOsh_MolFormat {
00097     NMF_PQR=0,
00098     NMF_PDB=1,
00099     NMF_XML=2
00100 };
00101
00106 typedef enum eNOsh_MolFormat NOsh_MolFormat;
00107
00112 enum eNOsh_CalcType {
00113     NCT_MG=0,
00114     NCT_FEM=1,
00115  NCT_APOL=2
00116 };
00117
00122 typedef enum eNOsh_CalcType NOsh_CalcType;
00123
00128 enum eNOsh_ParmFormat {
00129     NPF_FLAT=0,
00130     NPF_XML=1
00131 };
00132
00137 typedef enum eNOsh_ParmFormat NOsh_ParmFormat;
00138
00143 enum eNOsh_PrintType {
00144     NPT_ENERGY=0,
00145     NPT_FORCE=1,
00146  NPT_ELECENERGY,
00147     NPT_ELECFORCE,
00148  NPT_APOLENERGY,
00149     NPT_APOLFORCE
00150 };
00151
00156 typedef enum eNOsh_PrintType NOsh_PrintType;
00157
```

```
00163 struct sNOsh_calc {
00164  MGparm *mgparm;
00165  FEMparm *femparm;
00166  PBEparm *pbeparm;
00167  APOLparm *apolparm;
00168  NOsh_CalcType calctype;
00169 };
00170
00175 typedef struct sNOsh_calc NOsh_calc;
00176
00182 struct sNOsh {
00183
00184  NOsh_calc *calc[NOSH_MAXCALC];
00187     int ncalc;
00189  NOsh_calc *elec[NOSH_MAXCALC];
00192     int nelec;
00195  NOsh_calc *apol[NOSH_MAXCALC];
00198     int napol;
00201     int ispara;
00202     int proc_rank;
00203     int proc_size;
00204     int bogus;
00208     int elec2calc[NOSH_MAXCALC];
00216  int apol2calc[NOSH_MAXCALC];
00218     int nmol;
00219     char molpath[NOSH_MAXMOL][VMAX_ARGLEN];
00220     NOsh_MolFormat molfmt[NOSH_MAXMOL];
00221  Valist *alist[NOSH_MAXMOL];
00223     int gotparm;
00224     char parmpath[VMAX_ARGLEN];
00225     NOsh_ParmFormat parmfmt;
00226     int ndiel;
00227     char dielXpath[NOSH_MAXMOL][VMAX_ARGLEN];
00229     char dielYpath[NOSH_MAXMOL][VMAX_ARGLEN];
00231     char dielZpath[NOSH_MAXMOL][VMAX_ARGLEN];
00233     Vdata_Format dielfmt[NOSH_MAXMOL];
00234     int nkappa;
00235     char kappapath[NOSH_MAXMOL][VMAX_ARGLEN];
00236     Vdata_Format kappafmt[NOSH_MAXMOL];
00237  int npot;
00238     char potpath[NOSH_MAXMOL][VMAX_ARGLEN];
00239     Vdata_Format potfmt[NOSH_MAXMOL];
00240     int ncharge;
00241     char chargepath[NOSH_MAXMOL][VMAX_ARGLEN];
00242     Vdata_Format chargefmt[NOSH_MAXMOL];
00243  int nmesh;
00244  char meshpath[NOSH_MAXMOL][VMAX_ARGLEN];
00245     Vdata_Format meshfmt[NOSH_MAXMOL];
00246     int nprint;
00247     NOsh_PrintType printwhat[NOSH_MAXPRINT]
    ;
00249     int printnarg[NOSH_MAXPRINT];
00250     int printcalc[NOSH_MAXPRINT][NOSH_MAXPOP];

00251     int printop[NOSH_MAXPRINT][NOSH_MAXPOP];
00253     int parsed;
00254     char elecname[NOSH_MAXCALC][VMAX_ARGLEN];
00256  char apolname[NOSH_MAXCALC][VMAX_ARGLEN];
00258 };
00259
00264 typedef struct sNOsh NOsh;
00265
00266 /* ////////////////////////////////////////////////////////////////////////
00267    // Class NOsh: Inlineable methods (mcsh.c)
00269 #if !defined(VINLINE_NOSH)
00277 VEXTERNC char* NOsh_getMolpath(NOsh *thee, int imol);
00278
00286 VEXTERNC char* NOsh_getDielXpath(NOsh *thee, int imap);
00287
00295 VEXTERNC char* NOsh_getDielYpath(NOsh *thee, int imap);
00296
00304 VEXTERNC char* NOsh_getDielZpath(NOsh *thee, int imap);
00305
00313 VEXTERNC char* NOsh_getKappapath(NOsh *thee, int imap);
00314
00322 VEXTERNC char* NOsh_getPotpath(NOsh *thee, int imap);
00323
00331 VEXTERNC char* NOsh_getChargepath(NOsh *thee, int imap);
00332
00340 VEXTERNC NOsh_calc* NOsh_getCalc(NOsh *thee, int icalc);
00341
```

```
VEXTERNC int NOsh_getDielfmt(NOsh *thee, int imap);

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

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

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

#else

#   define NOsh_getMolpath(thee, imol) ((thee)->molpath[(imol)])
#   define NOsh_getDielXpath(thee, imol) ((thee)->dielXpath[(imol)])
#   define NOsh_getDielYpath(thee, imol) ((thee)->dielYpath[(imol)])
#   define NOsh_getDielZpath(thee, imol) ((thee)->dielZpath[(imol)])
#   define NOsh_getKappapath(thee, imol) ((thee)->kappapath[(imol)])
#   define NOsh_getPotpath(thee, imol) ((thee)->potpath[(imol)])
#   define NOsh_getChargepath(thee, imol) ((thee)->chargepath[(imol)])
#   define NOsh_getCalc(thee, icalc) ((thee)->calc[(icalc)])
#   define NOsh_getDielfmt(thee, imap) ((thee)->dielfmt[(imap)])
#   define NOsh_getKappafmt(thee, imap) ((thee)->kappafmt[(imap)])
#   define NOsh_getPotfmt(thee, imap) ((thee)->potfmt[(imap)])
#   define NOsh_getChargefmt(thee, imap) ((thee)->chargefmt[(imap)])

#endif


/* ///////////////////////////////////////////////////////////////////////////
   // Class NOsh: Non-inlineable methods (mcsh.c)

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

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

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

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

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

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

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

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

VEXTERNC NOsh_calc* NOsh_calc_ctor(
         NOsh_CalcType calcType
         );

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

VEXTERNC void  NOsh_calc_dtor(NOsh_calc **thee);

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

VEXTERNC void  NOsh_dtor(NOsh **thee);

VEXTERNC void  NOsh_dtor2(NOsh *thee);

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

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

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

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

#endif
```

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

Contains declarations for class PBEparm.

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

Include dependency graph for pbeparm.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct sPBEparm

  Parameter structure for PBE variables from input files.

Macros

- #define PBEPARM_MAXWRITE 20

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

Typedefs

- typedef enum ePBEparm_calcEnergy PBEparm_calcEnergy

  Define ePBEparm_calcEnergy enumeration as PBEparm_calcEnergy.
- typedef enum ePBEparm_calcForce PBEparm_calcForce

  Define ePBEparm_calcForce enumeration as PBEparm_calcForce.
- typedef struct sPBEparm PBEparm

  Declaration of the PBEparm class as the PBEparm structure.

Enumerations

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

  Define energy calculation enumeration.
- enum ePBEparm_calcForce { PCF_NO = 0, PCF_TOTAL = 1, PCF_COMPS = 2 }

  Define force calculation enumeration.

Functions

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

  Get charge (e) of specified ion species.
- VEXTERNC double PBEparm_getIonConc (PBEparm *thee, int iion)

  Get concentration (M) of specified ion species.
- VEXTERNC double PBEparm_getIonRadius (PBEparm *thee, int iion)

  Get radius (A) of specified ion species.
- VEXTERNC PBEparm * PBEparm_ctor ()

  Construct PBEparm object.
- VEXTERNC int PBEparm_ctor2 (PBEparm *thee)

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

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

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

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

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

  Parse a keyword from an input file.

9.27.1 Detailed Description

Contains declarations for class PBEparm.

Version

Id:

pbeparm.h 1774 2012-07-19 18:46:14Z sobolevnrm

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

```
#ifndef _PBEPARM_H_
#define _PBEPARM_H_

/* Generic headers */
#include "maloc/maloc.h"

/* Headers specific to this file */
#include "apbs/vhal.h"

#define PBEPARM_MAXWRITE 20

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

typedef enum ePBEparm_calcEnergy PBEparm_calcEnergy
;

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

typedef enum ePBEparm_calcForce PBEparm_calcForce
;

struct sPBEparm {

    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;
 /*-----------------------------------------------------------------*/
 /* Added by Michael Grabe                                          */
 /*-----------------------------------------------------------------*/

```

```
    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;
};

typedef struct sPBEparm PBEparm;

/* ////////////////////////////////////////////////////////////////////////
// Class NOsh: Non-inlineable methods (mcsh.c)

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

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

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


VEXTERNC PBEparm* PBEparm_ctor();

VEXTERNC int PBEparm_ctor2(
        PBEparm *thee
        );

VEXTERNC void PBEparm_dtor(
        PBEparm **thee
        );

VEXTERNC void PBEparm_dtor2(
        PBEparm *thee
        );

VEXTERNC int PBEparm_check(
        PBEparm *thee
        );

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

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


#endif
```

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

Contains declarations for class Vacc.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/valist.h"
#include "apbs/vclist.h"
#include "apbs/vatom.h"
#include "apbs/vunit.h"
#include "apbs/apolparm.h"
```

Include dependency graph for vacc.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct sVaccSurf

  Surface object list of per-atom surface points.

- struct sVacc

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

Typedefs

- typedef struct sVaccSurf VaccSurf

  Declaration of the VaccSurf class as the VaccSurf structure.
- typedef struct sVacc Vacc

  Declaration of the Vacc class as the Vacc structure.

Functions

- VEXTERNC unsigned long int Vacc_memChk (Vacc ∗thee)

  Get number of bytes in this object and its members.
- VEXTERNC VaccSurf ∗ VaccSurf_ctor (Vmem ∗mem, double probe_radius, int nsphere)

  Allocate and construct the surface object; do not assign surface points to positions.
- VEXTERNC int VaccSurf_ctor2 (VaccSurf ∗thee, Vmem ∗mem, double probe_radius, int nsphere)

  Construct the surface object using previously allocated memory; do not assign surface points to positions.
- VEXTERNC void VaccSurf_dtor (VaccSurf ∗∗thee)

  Destroy the surface object and free its memory.
- VEXTERNC void VaccSurf_dtor2 (VaccSurf ∗thee)

  Destroy the surface object.
- VEXTERNC VaccSurf ∗ VaccSurf_refSphere (Vmem ∗mem, int npts)

  Set up an array of points for a reference sphere of unit radius.
- VEXTERNC VaccSurf ∗ Vacc_atomSurf (Vacc ∗thee, Vatom ∗atom, VaccSurf ∗ref, double probe_radius)

  Set up an array of points corresponding to the SAS due to a particular atom.
- VEXTERNC Vacc ∗ Vacc_ctor (Valist ∗alist, Vclist ∗clist, double surf_density)

  Construct the accessibility object.
- VEXTERNC int Vacc_ctor2 (Vacc ∗thee, Valist ∗alist, Vclist ∗clist, double surf_density)

  FORTRAN stub to construct the accessibility object.
- VEXTERNC void Vacc_dtor (Vacc ∗∗thee)

  Destroy object.
- VEXTERNC void Vacc_dtor2 (Vacc ∗thee)

  FORTRAN stub to destroy object.
- VEXTERNC double Vacc_vdwAcc (Vacc ∗thee, double center[VAPBS_DIM])

  Report van der Waals accessibility.
- VEXTERNC double Vacc_ivdwAcc (Vacc ∗thee, double center[VAPBS_DIM], double radius)

  Report inflated van der Waals accessibility.
- VEXTERNC double Vacc_molAcc (Vacc ∗thee, double center[VAPBS_DIM], double radius)

  Report molecular accessibility.
- VEXTERNC double Vacc_fastMolAcc (Vacc ∗thee, double center[VAPBS_DIM], double radius)

  Report molecular accessibility quickly.
- VEXTERNC double Vacc_splineAcc (Vacc ∗thee, double center[VAPBS_DIM], double win, double infrad)

  Report spline-based accessibility.
- VEXTERNC void Vacc_splineAccGrad (Vacc ∗thee, double center[VAPBS_DIM], double win, double infrad, double ∗grad)

  Report gradient of spline-based accessibility.

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

  Report spline-based accessibility for a given atom.

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

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

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

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

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

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

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

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

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

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

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

  Return the total solvent accessible surface area (SASA)

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

  Return the atomic solvent accessible surface area (SASA)

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

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

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

  Get the derivatve of solvent accessible volume.

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

  Get the derivatve of solvent accessible area.

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

  Testing purposes only.

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

  Total solvent accessible volume.

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

  Return the total solvent accessible volume (SAV)

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

  Return the WCA integral energy.

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

  Return the WCA integral force.

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

  Calculate the WCA energy for an atom.

9.29.1 Detailed Description

Contains declarations for class Vacc.

Version

Id:

vacc.h 1774 2012-07-19 18:46:14Z sobolevnrm

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
```

```
        );

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

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

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

VEXTERNC void Vacc_dtor(
        Vacc **thee
        );

VEXTERNC void Vacc_dtor2(
        Vacc *thee
        );

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

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

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

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

VEXTERNC double Vacc_splineAcc(
        Vacc *thee,
        double center[VAPBS_DIM],
        double win,
        double infrad
        );

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

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

VEXTERNC void Vacc_splineAccGradAtomUnnorm(
        Vacc *thee,
        double center[VAPBS_DIM],
        double win,
        double infrad,
```

```
        Vatom *atom,
        double *force
        );

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

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

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


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

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

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

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

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

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

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

VEXTERNC void Vacc_totalAtomdSAV(
         Vacc *thee,
         double dpos,
         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,
00580  double radius
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)

  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.

9.31.1 Detailed Description

Contains declarations for class Valist.

Version

Id:

valist.h 1774 2012-07-19 18:46:14Z sobolevnrm

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
00078 struct sValist {
00079
```

```
  int number;
  double center[3];
  double mincrd[3];
  double maxcrd[3];
  double maxrad;
  double charge;
  Vatom *atoms;
  Vmem *vmem;
};

typedef struct sValist Valist;

#if !defined(VINLINE_VATOM)

VEXTERNC Vatom* Valist_getAtomList(
        Valist *thee
        );

VEXTERNC double Valist_getCenterX(
        Valist *thee
        );

VEXTERNC double Valist_getCenterY(
        Valist *thee
        );

VEXTERNC double Valist_getCenterZ(
        Valist *thee
        );

VEXTERNC int Valist_getNumberAtoms(
        Valist *thee
        );

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

VEXTERNC unsigned long int Valist_memChk(
        Valist *thee
        );

#else /* if defined(VINLINE_VATOM) */
#   define Valist_getAtomList(thee) ((thee)->atoms)
#   define Valist_getNumberAtoms(thee) ((thee)->number)
#   define Valist_getAtom(thee, i) (&((thee)->atoms[i]))
#   define Valist_memChk(thee) (Vmem_bytes((thee)->vmem))
#   define Valist_getCenterX(thee) ((thee)->center[0])
#   define Valist_getCenterY(thee) ((thee)->center[1])
#   define Valist_getCenterZ(thee) ((thee)->center[2])
#endif /* if !defined(VINLINE_VATOM) */

VEXTERNC Valist* Valist_ctor();

VEXTERNC Vrc_Codes Valist_ctor2(
        Valist *thee
        );

VEXTERNC void Valist_dtor(
        Valist **thee
        );

VEXTERNC void Valist_dtor2(
        Valist *thee
        );

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

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

VEXTERNC Vrc_Codes Valist_readXML(
        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/maloc.h"`
`#include "apbs/vhal.h"`
Include dependency graph for vatom.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct sVatom

Contains public data members for Vatom class/module.

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 1774 2012-07-19 18:46:14Z sobolevnrm

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 "maloc/maloc.h"
00066 #include "apbs/vhal.h"
00067
00074 #define VMAX_RECLEN     64
00075
00081 struct sVatom {
00082
00083     double position[3];
00084     double radius;
00085     double charge;
00086     double partID;
00088     double epsilon;
00090  int id;
00094  char resName[VMAX_RECLEN];
00095     char atomName[VMAX_RECLEN];
00097 #if defined(WITH_TINKER)
00098
00099     double dipole[3];
00100     double quadrupole[9];
00101     double inducedDipole[3];
00102     double nlInducedDipole[3];
00104 #endif /* if defined(WITH_TINKER) */
00105 };
00106
00111 typedef struct sVatom Vatom;
00112
00113 #if !defined(VINLINE_VATOM)
00114
00121     VEXTERNC double* Vatom_getPosition(Vatom *thee);
00122
00129     VEXTERNC void    Vatom_setRadius(Vatom *thee, double
      radius);
00130
00137     VEXTERNC double  Vatom_getRadius(Vatom *thee);
00138
00146     VEXTERNC void    Vatom_setPartID(Vatom *thee, int
      partID);
00147
00155     VEXTERNC double     Vatom_getPartID(Vatom *thee);
00156
00163     VEXTERNC void Vatom_setAtomID(Vatom *thee, int id);
```

```
00164
00171     VEXTERNC double Vatom_getAtomID(Vatom *thee);
00172
00179     VEXTERNC void    Vatom_setCharge(Vatom *thee, double
      charge);
00180
00187     VEXTERNC double  Vatom_getCharge(Vatom *thee);
00188
00195  VEXTERNC void    Vatom_setEpsilon(Vatom *thee, double
      epsilon);
00196
00203  VEXTERNC double  Vatom_getEpsilon(Vatom *thee);
00204
00212     VEXTERNC unsigned long int Vatom_memChk(Vatom *thee);
00213
00214 #else /* if defined(VINLINE_VATOM) */
00215 #   define Vatom_getPosition(thee) ((thee)->position)
00216 #   define Vatom_setRadius(thee, tRadius) ((thee)->radius = (tRadius))
00217 #   define Vatom_getRadius(thee) ((thee)->radius)
00218 #   define Vatom_setPartID(thee, tpartID) ((thee)->partID = (double)(tpartID))
00219 #   define Vatom_getPartID(thee) ((thee)->partID)
00220 #   define Vatom_setAtomID(thee, tatomID) ((thee)->id = (tatomID))
00221 #   define Vatom_getAtomID(thee) ((thee)->id)
00222 #   define Vatom_setCharge(thee, tCharge) ((thee)->charge = (tCharge))
00223 #   define Vatom_getCharge(thee) ((thee)->charge)
00224 #   define Vatom_setEpsilon(thee, tEpsilon) ((thee)->epsilon = (tEpsilon))
00225 #   define Vatom_getEpsilon(thee) ((thee)->epsilon)
00226 #   define Vatom_memChk(thee) (sizeof(Vatom))
00227 #endif /* if !defined(VINLINE_VATOM) */
00228
00229 /* ////////////////////////////////////////////////////////////////////////
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 1774 2012-07-19 18:46:14Z sobolevnrm

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

```
#ifndef _VCAP_H_
#define _VCAP_H_

#define EXPMAX  85.00

#define EXPMIN -85.00

#include "maloc/maloc.h"

VEXTERNC double Vcap_exp(
        double x,
        int *ichop
        );


VEXTERNC double Vcap_sinh(
        double x,
        int *ichop
        );

VEXTERNC double Vcap_cosh(
        double x,
        int *ichop
        );

#endif    /* ifndef _VCAP_H_ */
```

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

Contains declarations for class Vclist.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/valist.h"
#include "apbs/vatom.h"
#include "apbs/vunit.h"
```

Include dependency graph for vclist.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct sVclistCell

  Atom cell list cell.
- struct sVclist

  Atom cell list.

Typedefs

- typedef enum eVclist_DomainMode Vclist_DomainMode

  Declaration of Vclist_DomainMode enumeration type.
- typedef struct sVclistCell VclistCell

  Declaration of the VclistCell class as the VclistCell structure.
- typedef struct sVclist Vclist

  Declaration of the Vclist class as the Vclist structure.

Enumerations

- enum eVclist_DomainMode { CLIST_AUTO_DOMAIN, CLIST_MANUAL_DOMAIN }

  Atom cell list domain setup mode.

Functions

- VEXTERNC unsigned long int Vclist_memChk (Vclist ∗thee)

  Get number of bytes in this object and its members.
- VEXTERNC double Vclist_maxRadius (Vclist ∗thee)

  Get the max probe radius value (in A) the cell list was constructed with.
- VEXTERNC Vclist ∗ Vclist_ctor (Valist ∗alist, double max_radius, int npts[VAPBS_DIM], Vclist_DomainMode mode, double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM])

  Construct the cell list object.
- VEXTERNC Vrc_Codes Vclist_ctor2 (Vclist ∗thee, Valist ∗alist, double max_radius, int npts[VAPBS_DIM], Vclist_DomainMode mode, double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM])

  FORTRAN stub to construct the cell list object.
- VEXTERNC void Vclist_dtor (Vclist ∗∗thee)

  Destroy object.
- VEXTERNC void Vclist_dtor2 (Vclist ∗thee)

  FORTRAN stub to destroy object.
- VEXTERNC VclistCell ∗ Vclist_getCell (Vclist ∗thee, double position[VAPBS_DIM])

  Return cell corresponding to specified position or return VNULL.
- VEXTERNC VclistCell ∗ VclistCell_ctor (int natoms)

  Allocate and construct a cell list cell object.
- VEXTERNC Vrc_Codes VclistCell_ctor2 (VclistCell ∗thee, int natoms)

  Construct a cell list object.
- VEXTERNC void VclistCell_dtor (VclistCell ∗∗thee)

  Destroy object.
- VEXTERNC void VclistCell_dtor2 (VclistCell ∗thee)

  FORTRAN stub to destroy object.

9.37.1 Detailed Description

Contains declarations for class Vclist.

Version

Id:

vclist.h 1774 2012-07-19 18:46:14Z sobolevnrm

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 "maloc/maloc.h"
00067 #include "apbs/vhal.h"
00068
00069 /* Headers specific to this file */
00070 #include "apbs/valist.h"
00071 #include "apbs/vatom.h"
00072 #include "apbs/vunit.h"
00073
00079 enum eVclist_DomainMode {
00080     CLIST_AUTO_DOMAIN,
00082     CLIST_MANUAL_DOMAIN
00084 };
00085
00091 typedef enum eVclist_DomainMode Vclist_DomainMode
      ;
00092
00098 struct sVclistCell {
00099     Vatom **atoms;
00100     int natoms;
00101 };
00102
00107 typedef struct sVclistCell VclistCell;
00108
00114 struct sVclist {
00115
00116   Vmem *vmem;
00117   Valist *alist;
00118   Vclist_DomainMode mode;
00119   int npts[VAPBS_DIM];
00120   int n;
00121   double max_radius;
00122   VclistCell *cells;
00123   double lower_corner[VAPBS_DIM];
00124   double upper_corner[VAPBS_DIM];
00125   double spacs[VAPBS_DIM];
00127 };
00128
00133 typedef struct sVclist Vclist;
00134
00135 #if !defined(VINLINE_VCLIST)
00136
00142     VEXTERNC unsigned long int Vclist_memChk(
00143             Vclist *thee
00144             );
00145
00153     VEXTERNC double Vclist_maxRadius(
00154             Vclist *thee
00155             );
00156
00157 #else /* if defined(VINLINE_VCLIST) */
00158
00159 #   define Vclist_memChk(thee) (Vmem_bytes((thee)->vmem))
00160 #   define Vclist_maxRadius(thee) ((thee)->max_radius)
00161
00162 #endif /* if !defined(VINLINE_VCLIST) */
00163
00164 /* ////////////////////////////////////////////////////////////////////////////
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],
```

```
        Vclist_DomainMode mode,
        double lower_corner[VAPBS_DIM],
        double upper_corner[VAPBS_DIM]
        );

VEXTERNC void Vclist_dtor(
        Vclist **thee
        );

VEXTERNC void Vclist_dtor2(
        Vclist *thee
        );

VEXTERNC VclistCell* Vclist_getCell(
        Vclist *thee,
        double position[VAPBS_DIM]
        );

VEXTERNC VclistCell* VclistCell_ctor(
        int natoms
        );

VEXTERNC Vrc_Codes VclistCell_ctor2(
        VclistCell *thee,
        int natoms
        );

VEXTERNC void VclistCell_dtor(
        VclistCell **thee
        );

VEXTERNC void VclistCell_dtor2(
        VclistCell *thee
        );

#endif    /* ifndef _VCLIST_H_ */
```

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

Contains declarations for class Vgreen.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/vunit.h"
#include "apbs/vatom.h"
#include "apbs/valist.h"
```

Include dependency graph for vgreen.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct sVgreen

  Contains public data members for Vgreen class/module.

Typedefs

- typedef struct sVgreen Vgreen

  Declaration of the Vgreen class as the Vgreen structure.

Functions

- VEXTERNC Valist * Vgreen_getValist (Vgreen *thee)

  Get the atom list associated with this Green's function object.
- VEXTERNC unsigned long int Vgreen_memChk (Vgreen *thee)

  Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC Vgreen * Vgreen_ctor (Valist *alist)

  Construct the Green's function oracle.
- VEXTERNC int Vgreen_ctor2 (Vgreen *thee, Valist *alist)

  FORTRAN stub to construct the Green's function oracle.
- VEXTERNC void Vgreen_dtor (Vgreen **thee)

  Destruct the Green's function oracle.
- VEXTERNC void Vgreen_dtor2 (Vgreen *thee)

  FORTRAN stub to destruct the Green's function oracle.
- VEXTERNC int Vgreen_helmholtz (Vgreen *thee, int npos, double *x, double *y, double *z, double *val, double kappa)

  Get the Green's function for Helmholtz's equation integrated over the atomic point charges.
- VEXTERNC int Vgreen_helmholtzD (Vgreen *thee, int npos, double *x, double *y, double *z, double *gradx, double *grady, double *gradz, double kappa)

  Get the gradient of Green's function for Helmholtz's equation integrated over the atomic point charges.
- VEXTERNC int Vgreen_coulomb_direct (Vgreen *thee, int npos, double *x, double *y, double *z, double *val)

  Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.
- VEXTERNC int Vgreen_coulomb (Vgreen *thee, int npos, double *x, double *y, double *z, double *val)

  Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation or H. E. Johnston, R. Krasny FMM library (if available)
- VEXTERNC int Vgreen_coulombD_direct (Vgreen *thee, int npos, double *x, double *y, double *z, double *pot, double *gradx, double *grady, double *gradz)

  Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.
- VEXTERNC int Vgreen_coulombD (Vgreen *thee, int npos, double *x, double *y, double *z, double *pot, double *gradx, double *grady, double *gradz)

  Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using either direct summation or H. E. Johnston/R. Krasny FMM library (if available)

9.39.1 Detailed Description

Contains declarations for class Vgreen.

Version

Id:

vgreen.h 1774 2012-07-19 18:46:14Z sobolevnrm

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;
00089   double *yp;
00091   double *zp;
00093   double *qp;
00095   int np;
00096 };
00097
00102 typedef struct sVgreen Vgreen;
00103
00104 /* ///////////////////////////////////////////////////////////////////////
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 /* ///////////////////////////////////////////////////////////////////////
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
```

```
VEXTERNC int Vgreen_coulombD(Vgreen *thee, int npos, double *x, double *y,
  double *z, double *pot, double *gradx, double *grady, double *gradz);

#endif /* ifndef _VGREEN_H_ */
```

9.41 src/generic/apbs/vhal.h File Reference

Contains generic macro definitions for APBS.

`#include "stdio.h"`

Include dependency graph for vhal.h:

This graph shows which files directly or indirectly include this file:

Macros

- #define APBS_TIMER_WALL_CLOCK 26

  APBS total execution timer ID.
- #define APBS_TIMER_SETUP 27

  APBS setup timer ID.
- #define APBS_TIMER_SOLVER 28

  APBS solver timer ID.
- #define APBS_TIMER_ENERGY 29

  APBS energy timer ID.
- #define APBS_TIMER_FORCE 30

  APBS force timer ID.
- #define APBS_TIMER_TEMP1 31

  APBS temp timer #1 ID.

- #define APBS_TIMER_TEMP2 32

  APBS temp timer #2 ID.
- #define MAXMOL 5

  The maximum number of molecules that can be involved in a single PBE calculation.
- #define MAXION 10

  The maximum number of ion species that can be involved in a single PBE calculation.
- #define MAXFOCUS 5

  The maximum number of times an MG calculation can be focused.
- #define VMGNLEV 4

  Minimum number of levels in a multigrid calculations.
- #define VREDFRAC 0.25

  Maximum reduction of grid spacing during a focusing calculation.
- #define VAPBS_NVS 4

  Number of vertices per simplex (hard-coded to 3D)
- #define VAPBS_DIM 3

  Our dimension.
- #define VAPBS_RIGHT 0

  Face definition for a volume.
- #define MAX_SPHERE_PTS 50000

  Maximum number of points on a sphere.
- #define VAPBS_FRONT 1

  Face definition for a volume.
- #define VAPBS_UP 2

  Face definition for a volume.
- #define VAPBS_LEFT 3

  Face definition for a volume.
- #define VAPBS_BACK 4

  Face definition for a volume.
- #define VAPBS_DOWN 5

  Face definition for a volume.
- #define VPMGSMALL 1e-12

  A small number used in Vpmg to decide if points are on/off grid-lines or non-zer0 (etc.)
- #define SINH_MIN -85.0

  Used to set the min values acceptable for sinh chopping.
- #define SINH_MAX 85.0

  Used to set the max values acceptable for sinh chopping.
- #define VF77_MANGLE(name, NAME) name

  Name-mangling macro for using FORTRAN functions in C code.
- #define VFLOOR(value) floor(value)

  Wrapped floor to fix floating point issues in the Intel compiler.
- #define VEMBED(rctag)

  Allows embedding of RCS ID tags in object files.
- #define PRINT_FUNC __PRETTY_FUNCTION__
- #define **OS_SEP_STR** "/"
- #define **OS_SEP_CHAR** '/'
- #define **ANNOUNCE_FUNCTION**
- #define **WARN_UNTESTED**

- #define **WARN_PARTTESTED**
- #define **VCHANNELEDMESSAGE0**(channel, msg)
- #define **VCHANNELEDMESSAGE1**(channel, msg, arg0)
- #define **VCHANNELEDMESSAGE2**(channel, msg, arg0, arg1)
- #define **VCHANNELEDMESSAGE3**(channel, msg, arg0, arg1, arg2)
- #define **VMESSAGE0**(msg) VCHANNELEDMESSAGE0(0, msg)
- #define **VMESSAGE1**(msg, arg0) VCHANNELEDMESSAGE1(0, msg, arg0)
- #define **VMESSAGE2**(msg, arg0, arg1) VCHANNELEDMESSAGE2(0, msg, arg0, arg1)
- #define **VMESSAGE3**(msg, arg0, arg1, arg2) VCHANNELEDMESSAGE3(0, msg, arg0, arg1, arg2)
- #define **VERRMSG0**(msg) VCHANNELEDMESSAGE0(2, msg)
- #define **VERRMSG1**(msg, arg0) VCHANNELEDMESSAGE1(2, msg, arg0)
- #define **VERRMSG2**(msg, arg0, arg1) VCHANNELEDMESSAGE2(2, msg, arg0, arg1)
- #define **VERRMSG3**(msg, arg0, arg1, arg2) VCHANNELEDMESSAGE3(2, msg, arg0, arg1, arg2)
- #define **VASSERT_MSG0**(cnd, msg)
- #define **VASSERT_MSG1**(cnd, msg, arg)
- #define **VASSERT_MSG2**(cnd, msg, arg0, arg1)
- #define **VWARN_MSG0**(cnd, msg)
- #define **VWARN_MSG1**(cnd, msg, arg0)
- #define **VWARN_MSG2**(cnd, msg, arg0, arg1)
- #define **VABORT_MSG0**(msg)
- #define **VABORT_MSG1**(msg, arg)
- #define **VABORT_MSG2**(msg, arg0, arg1)
- #define **PRINT_INT**(expr)
- #define **PRINT_DBL**(expr)
- #define **VMALLOC**(vmem, n, type) ((type∗)Vmem_malloc(vmem, n, sizeof(type)))
- #define **VFREE**(vmem, n, type, ptr) (Vmem_free(vmem, n, sizeof(type), (void ∗∗)&(ptr)))
- #define **VFILL**(vec, n, val)
- #define **VCOPY**(srcvec, dstvec, i, n)
- #define **VAT**(array, i) ((array)[(i) - 1])
- #define **RAT**(array, i) ((array) + i - 1)

Typedefs

- typedef enum eVrc_Codes **Vrc_Codes**
- typedef enum eVsol_Meth **Vsol_Meth**
- typedef enum eVsurf_Meth Vsurf_Meth

  Declaration of the Vsurf_Meth type as the Vsurf_Meth enum.
- typedef enum eVhal_PBEType Vhal_PBEType

  Declaration of the Vhal_PBEType type as the Vhal_PBEType enum.
- typedef enum eVhal_IPKEYType Vhal_IPKEYType

  Declaration of the Vhal_IPKEYType type as the Vhal_IPKEYType enum.
- typedef enum eVhal_NONLINType Vhal_NONLINType

  Declaration of the Vhal_NONLINType type as the Vhal_NONLINType enum.
- typedef enum eVoutput_Format Voutput_Format

  Declaration of the Voutput_Format type as the VOutput_Format enum.
- typedef enum eVbcfl Vbcfl

  Declare Vbcfl type.
- typedef enum eVchrg_Meth Vchrg_Meth

  Declaration of the Vchrg_Meth type as the Vchrg_Meth enum.

- typedef enum eVchrg_Src Vchrg_Src

  Declaration of the Vchrg_Src type as the Vchrg_Meth enum.
- typedef enum eVdata_Type Vdata_Type

  Declaration of the Vdata_Type type as the Vdata_Type enum.
- typedef enum eVdata_Format Vdata_Format

  Declaration of the Vdata_Format type as the Vdata_Format enum.

Enumerations

- enum eVrc_Codes { **VRC_WARNING** = -1, VRC_FAILURE = 0, VRC_SUCCESS = 1 }

  Return code enumerations.
- enum eVsol_Meth {
  VSOL_CGMG, **VSOL_Newton**, **VSOL_MG**, **VSOL_CG**,
  VSOL_SOR, **VSOL_RBGS**, **VSOL_WJ**, **VSOL_Richardson**,
  VSOL_CGMGAqua, **VSOL_NewtonAqua** }

  Solution Method enumerations.
- enum eVsurf_Meth {
  VSM_MOL = 0, VSM_MOLSMOOTH = 1, VSM_SPLINE = 2, VSM_SPLINE3 = 3,
  VSM_SPLINE4 = 4 }

  Types of molecular surface definitions.
- enum eVhal_PBEType {
  PBE_LPBE, PBE_NPBE, PBE_LRPBE, **PBE_NRPBE**,
  PBE_SMPBE }

  Version of PBE to solve.
- enum eVhal_IPKEYType { IPKEY_SMPBE = -2, IPKEY_LPBE, IPKEY_NPBE }

  Type of ipkey to use for MG methods.
- enum eVhal_NONLINType {
  NONLIN_LPBE = 0, **NONLIN_NPBE**, **NONLIN_SMPBE**, **NONLIN_LPBEAQUA**,
  NONLIN_NPBEAQUA }

  Type of nonlinear to use for MG methods.
- enum eVoutput_Format { OUTPUT_NULL, OUTPUT_FLAT }

  Output file format.
- enum eVbcfl {
  BCFL_ZERO = 0, BCFL_SDH = 1, BCFL_MDH = 2, BCFL_UNUSED = 3,
  BCFL_FOCUS = 4, BCFL_MEM = 5, BCFL_MAP = 6 }

  Types of boundary conditions.
- enum eVchrg_Meth { VCM_TRIL = 0, VCM_BSPL2 = 1, VCM_BSPL4 = 2 }

  Types of charge discretization methods.
- enum eVchrg_Src { VCM_CHARGE = 0, VCM_PERMANENT = 1, VCM_INDUCED = 2, VCM_NLINDUCED = 3 }

  Charge source.
- enum eVdata_Type {
  VDT_CHARGE, VDT_POT, VDT_ATOMPOT, VDT_SMOL,
  VDT_SSPL, VDT_VDW, VDT_IVDW, VDT_LAP,
  VDT_EDENS, VDT_NDENS, VDT_QDENS, VDT_DIELX,
  VDT_DIELY, VDT_DIELZ, VDT_KAPPA }

  Types of (scalar) data that can be written out of APBS.
- enum eVdata_Format {
  VDF_DX = 0, VDF_UHBD = 1, VDF_AVS = 2, VDF_MCSF = 3,
  VDF_GZ = 4, VDF_FLAT = 5 }

  Format of data for APBS I/O.

Functions

- char * **wrap_text** (char *str, int right_margin, int left_padding)

Variables

- FILE * **data**

9.41.1 Detailed Description

Contains generic macro definitions for APBS.

Version

Id:

vhal.h 1794 2012-08-01 03:34:53Z tuckerbeck

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2012 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory,
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
```

```
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 *
```

Definition in file vhal.h.

9.41.2 Macro Definition Documentation

9.41.2.1 #define PRINT_FUNC __PRETTY_FUNCTION__

OS specific flags and etcetera

Definition at line 578 of file vhal.h.

9.41.2.2 #define VABORT_MSG0(*msg*)

Value:

```
do {                                    \
        Vnm_print(2, "%[%s()]: ABORTING:\n" \
                     "    %s\n\n",          \
                 __FUNCTION__, msg);        \
        abort();                            \
    } while(0)
```

Definition at line 906 of file vhal.h.

9.41.2.3 #define VABORT_MSG1(*msg, arg*)

Value:

```
do {                                    \
        char buff[1000];                    \
        snprintf( buff, 1000, msg, arg );   \
        Vnm_print(2, "[%s()]: ABORTING:\n" \
                     "    %s\n\n",          \
                 __FUNCTION__, buff);       \
        abort();                            \
    } while(0)
```

Definition at line 914 of file vhal.h.

9.41.2.4 #define VABORT_MSG2(*msg, arg0, arg1*)

Value:

```
do {                                        \
        char buff[1000];                        \
        snprintf( buff, 1000, msg, arg0, arg1); \
        Vnm_print(2, "[%s()]: ABORTING:\n"      \
                     "    %s\n\n",               \
                 __FUNCTION__, buff);           \
        abort();                                \
    } while(0)
```

Definition at line 924 of file vhal.h.

9.41.2.5 #define VASSERT_MSG0(*cnd, msg*)

Value:

```
do {                                                                \
        if( (cnd) == 0 ) {                                          \
            Vnm_print(2, "%[%s()]: ERROR:\n"                        \
                         "    Assertion Failed (%s): %s\n\n",       \
                     __FUNCTION__, #cnd, msg);                      \
            abort();                                                \
        }                                                           \
    } while(0)
```

Definition at line 738 of file vhal.h.

9.41.2.6 #define VASSERT_MSG1(*cnd, msg, arg*)

Value:

```
do {                                                                \
        if( (cnd) == 0 ) {                                          \
            char buff[1000];                                        \
            snprintf( buff, 1000, msg, arg );                       \
            Vnm_print(2, "[%s()]: ERROR:\n"                         \
                         "    Assertion Failed (%s): %s\n\n",       \
                     __FUNCTION__, #cnd, buff);                     \
            abort();                                                \
        }                                                           \
    } while(0)
```

Definition at line 748 of file vhal.h.

9.41.2.7 #define VASSERT_MSG2(*cnd, msg, arg0, arg1*)

Value:

```
do {                                                                \
        if( (cnd) == 0 ) {                                          \
            char buff[1000];                                        \
            snprintf( buff, 1000, msg, arg0, arg1 );                \
            Vnm_print(2, "[%s()]: ERROR:\n"                         \
                         "    Assertion Failed (%s): %s\n\n",       \
                     __FUNCTION__, #cnd, buff);                     \
            abort();                                                \
        }                                                           \
    } while(0)
```

Definition at line 760 of file vhal.h.

9.41.2.8 #define VCHANNELEDMESSAGE0(*channel, msg*)

Value:

```
do {                                                    \
            Vnm_print(channel, "%s: %s\n", __FUNCTION__, msg); \
        } while(0)
```

Definition at line 660 of file vhal.h.

9.41.2.9 #define VCHANNELEDMESSAGE1(*channel, msg, arg0*)

Value:

```
do {                                                    \
            char buff[1000];                                \
            snprintf( buff, 1000, msg, arg0 );              \
            Vnm_print(channel, "%s: %s\n", __FUNCTION__, buff); \
        } while(0)
```

Definition at line 665 of file vhal.h.

9.41.2.10 #define VCHANNELEDMESSAGE2(*channel, msg, arg0, arg1*)

Value:

```
do {                                                    \
            char buff[1000];                                \
            snprintf( buff, 1000, msg, arg0, arg1 );        \
            Vnm_print(channel, "%s: %s\n", __FUNCTION__, buff); \
        } while(0)
```

Definition at line 672 of file vhal.h.

9.41.2.11 #define VCHANNELEDMESSAGE3(*channel, msg, arg0, arg1, arg2*)

Value:

```
do {                                                     \
            char buff[1000];                                 \
            snprintf(buff, 1000, msg, arg0, arg1, arg2);     \
            Vnm_print(channel, "%s: %s\n", __FUNCTION__, buff);  \
        } while(0)
```

Definition at line 679 of file vhal.h.

9.41.2.12 #define VCOPY(*srcvec, dstvec, i, n*)

Value:

```
do {                            \
        for (i = 0; i < n; i++)     \
            dstvec[i] = srcvec[i];  \
    } while(0)
```

Definition at line 966 of file vhal.h.

9.41.2.13 #define VFILL(*vec, n, val*)

Value:

```
do {                                          \
        int fill_idx;                             \
        for (fill_idx = 0; fill_idx < n; fill_idx++) \
            vec[fill_idx] = val;                  \
    } while(0)
```

Definition at line 959 of file vhal.h.

9.41.2.14 #define VWARN_MSG0(*cnd, msg*)

Value:

```
do {                                                                    \
                if( (cnd) == 0 ) {                                      \
                        Vnm_print(                                      \
                                2,                                      \
                                "[%s()]: WARNING:\n"                    \
                                "    %s\n\n",                           \
                                __FUNCTION__,                           \
                                msg                                     \
                                );                                      \
                }                                                       \
        } while(0)
```

Definition at line 832 of file vhal.h.

9.41.2.15 #define VWARN_MSG1(*cnd, msg, arg0*)

Value:

```
do {                                                                    \
                if( (cnd) == 0 ) {                                      \
                        char buff[1000];                                \
                        snprintf(buff, 1000, msg, arg0);                \
                        Vnm_print(                                      \
                                2,                                      \
                                "[%s()]: WARNING:\n"                    \
                                "    %s\n\n",                           \
                                __FUNCTION__,                           \
                                buff                                    \
                                );                                      \
                }                                                       \
        } while(0)
```

Definition at line 845 of file vhal.h.

9.41.2.16 #define VWARN_MSG2(*cnd, msg, arg0, arg1*)

Value:

```
do {                                                                    \
                if( (cnd) == 0 ) {                                      \
                        char buff[1000];                                \
                        snprintf(buff, 1000, msg, arg0, arg1);          \
                        Vnm_print(                                      \
                                2,                                      \
                                "[%s()]: WARNING:\n"                    \
                                "    %s\n\n",                           \
                                __FUNCTION__,                           \
                                buff                                    \
                                );                                      \
                }                                                       \
        } while(0)
```

Definition at line 860 of file vhal.h.

9.42 vhal.h

```
00001
00055 #ifndef _VAPBSHAL_H_
00056 #define _VAPBSHAL_H_
```

```
#include "stdio.h"

FILE *data;

enum eVrc_Codes {

    VRC_WARNING=-1,
    VRC_FAILURE=0,
    VRC_SUCCESS=1
};
typedef enum eVrc_Codes Vrc_Codes;

enum eVsol_Meth {

    VSOL_CGMG,        /* 0: conjugate gradient multigrid */
    VSOL_Newton,    /* 1: newton */
    VSOL_MG,        /* 2: multigrid */
    VSOL_CG,        /* 3: conjugate gradient */
    VSOL_SOR,        /* 4: sucessive overrelaxation */
    VSOL_RBGS,        /* 5: red-black gauss-seidel */
    VSOL_WJ,        /* 6: weighted jacobi */
    VSOL_Richardson,/* 7: richardson  */
    VSOL_CGMGAqua,    /* 8: conjugate gradient multigrid aqua */
    VSOL_NewtonAqua    /* 9: newton aqua */

};
typedef enum eVsol_Meth Vsol_Meth;

enum eVsurf_Meth {
    VSM_MOL=0,
    VSM_MOLSMOOTH=1,
    VSM_SPLINE=2,
    VSM_SPLINE3=3,
    VSM_SPLINE4=4
};

typedef enum eVsurf_Meth Vsurf_Meth;

enum eVhal_PBEType {
    PBE_LPBE,
    PBE_NPBE,
    PBE_LRPBE,
    PBE_NRPBE,
    PBE_SMPBE
};

typedef enum eVhal_PBEType Vhal_PBEType;

enum eVhal_IPKEYType {
    IPKEY_SMPBE = -2,
    IPKEY_LPBE,
    IPKEY_NPBE
};

typedef enum eVhal_IPKEYType Vhal_IPKEYType;

enum eVhal_NONLINType {
    NONLIN_LPBE = 0,
    NONLIN_NPBE,
    NONLIN_SMPBE,
    NONLIN_LPBEAQUA,
    NONLIN_NPBEAQUA
};

typedef enum eVhal_NONLINType Vhal_NONLINType;

enum eVoutput_Format {
    OUTPUT_NULL,
    OUTPUT_FLAT,
};

typedef enum eVoutput_Format Voutput_Format;

enum eVbcfl {
    BCFL_ZERO=0,
    BCFL_SDH=1,
    BCFL_MDH=2,
    BCFL_UNUSED=3,
    BCFL_FOCUS=4,
    BCFL_MEM=5,
```

```
    BCFL_MAP=6
};

typedef enum eVbcfl Vbcfl;

enum eVchrg_Meth {
    VCM_TRIL=0,
    VCM_BSPL2=1,
    VCM_BSPL4=2
};

typedef enum eVchrg_Meth Vchrg_Meth;

enum eVchrg_Src {
    VCM_CHARGE=0,
    VCM_PERMANENT=1,
    VCM_INDUCED=2,
    VCM_NLINDUCED=3
};

typedef enum eVchrg_Src Vchrg_Src;

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
};

typedef enum eVdata_Type Vdata_Type;

enum eVdata_Format {
    VDF_DX=0,
    VDF_UHBD=1,
    VDF_AVS=2,
    VDF_MCSF=3,
    VDF_GZ=4,
    VDF_FLAT=5
};

typedef enum eVdata_Format Vdata_Format;

#define APBS_TIMER_WALL_CLOCK 26

#define APBS_TIMER_SETUP 27

#define APBS_TIMER_SOLVER 28

#define APBS_TIMER_ENERGY 29

#define APBS_TIMER_FORCE 30

#define APBS_TIMER_TEMP1 31

#define APBS_TIMER_TEMP2 32

#define MAXMOL 5

#define MAXION 10

#define MAXFOCUS 5

#define VMGNLEV 4

#define VREDFRAC 0.25

#define VAPBS_NVS 4

#define VAPBS_DIM 3

```

```
#define VAPBS_RIGHT 0

#define MAX_SPHERE_PTS 50000

#define VAPBS_FRONT 1

#define VAPBS_UP    2

#define VAPBS_LEFT  3

#define VAPBS_BACK  4

#define VAPBS_DOWN  5

#define VPMGSMALL 1e-12

#define SINH_MIN -85.0

#define SINH_MAX 85.0

#if defined(VDEBUG)
#   if !defined(APBS_NOINLINE)
#       define APBS_NOINLINE 1
#   endif
#endif

#if !defined(APBS_NOINLINE)

#   define VINLINE_VACC

#   define VINLINE_VATOM

#   define VINLINE_VCSM

#   define VINLINE_VPBE

#   define VINLINE_VPEE

#   define VINLINE_VGREEN

#   define VINLINE_VFETK

#   define VINLINE_VPMG

#   define MAX_HASH_DIM 75

#endif

/* Fortran name mangling */
#if defined(VF77_UPPERCASE)
#   if defined(VF77_NOUNDERSCORE)
#       define VF77_MANGLE(name,NAME) NAME
#   elif defined(VF77_ONEUNDERSCORE)
#       define VF77_MANGLE(name,NAME) NAME ## _
#   else
#       define VF77_MANGLE(name,NAME) name
#   endif
#else
#   if defined(VF77_NOUNDERSCORE)
#       define VF77_MANGLE(name,NAME) name
#   elif defined(VF77_ONEUNDERSCORE)
#       define VF77_MANGLE(name,NAME) name ## _
#   else

#       define VF77_MANGLE(name,NAME) name
#   endif
#endif

/* Floating Point Error */
#if defined(MACHINE_EPS)
#     define VFLOOR(value) \
              ((floor(value) != floor(value + MACHINE_EPS)) ? \
              floor(value + MACHINE_EPS) : floor(value))
#else

#     define VFLOOR(value) floor(value)
#endif

/* String embedding for ident */
#if defined(HAVE_EMBED)

```

```
#    define VEMBED(rctag) \
         VPRIVATE const char* rctag; \
         static void* use_rcsid=(0 ? &use_rcsid : (void**)&rcsid);
#else

#    define VEMBED(rctag)
#endif /* if defined(HAVE_EMBED) */



#if !defined(WIN32) || defined(__MINGW32__)
#define PRINT_FUNC __PRETTY_FUNCTION__
#define OS_SEP_STR "/"
#define OS_SEP_CHAR '/'
#else
#define OS_SEP_STR "\\"
#define OS_SEP_CHAR '\\'
#define PRINT_FUNC __FUNCSIG__
#define snprintf sprintf_s
#endif

#ifdef VERGBOSE_DEBUG
#define ANNOUNCE_FUNCTION                                   \
        do {                                                \
            Vnm_prrint(2, "%s() [%s:%d]\n",                 \
                       PRINT_FUNC, __FILE__, __LINE__ ); \
        } while(0)

#define WARN_UNTESTED                                                 \
        do {                                                          \
            Vnm_print(2, "%s() [%s:%d]: Untested Translation!\n", \
                      __FUNCTION__, __FILE__, __LINE__);          \
        } while(0)

#define WARN_PARTTESTED                                                     \
        do{                                                                 \
            Vnm_print(2, "%s() [%s:%d]: Partially Tested Translation.\n", \
                      __FUNCTION__, __FILE__, __LINE__);                  \
        } while(0)
#else
#define ANNOUNCE_FUNCTION
#define WARN_UNTESTED
#define WARN_PARTTESTED
#endif



/* Utility messages.  Print out messages with location information */
#ifdef DEBUG
#define VCHANNELEDMESSAGE0(channel, msg)                      \
        do {                                                  \
            Vnm_print(channel, "%s:%d [%s()]: MESSAGE:\n"     \
                      "    %s\n\n",                           \
                      __FILE__, __LINE__, __FUNCTION__, msg); \
        } while(0)

#define VCHANNELEDMESSAGE1(channel, msg, arg)                  \
        do {                                                   \
            char buff[1000];                                   \
            snprintf( buff, 1000, msg, arg );                  \
            Vnm_print(channel, "%s:%d [%s()]: MESSAGE:\n"      \
                      "    %s\n\n",                            \
                      __FILE__, __LINE__, __FUNCTION__, buff); \
        } while(0)

#define VCHANNELEDMESSAGE2(channel, msg, arg0, arg1)           \
        do {                                                   \
            char buff[1000];                                   \
            snprintf( buff, 1000, msg, arg0, arg1 );           \
            Vnm_print(channel, "%s:%d [%s()]: MESSAGE:\n"      \
                      "    %s\n\n",                            \
                      __FILE__, __LINE__, __FUNCTION__, buff); \
        } while(0)

#define VCHANNELEDMESSAGE3(channel, msg, arg0, arg1, arg2)     \
        do {                                                   \
            char buff[1000];                                   \
            snprintf(buff, 1000, msg, arg0, arg1, arg2);       \
            Vnm_print(channel, "%s:%d [%s()]: MESSAGE:\n"      \
                      "    %s\n\n",                            \
                      __FILE__, __LINE__, __FUNCTION__, buff); \
```

```
        } while(0)

#define VMESSAGE0(msg)                  VCHANNELEDMESSAGE0(2, msg)
#define VMESSAGE1(msg, arg0)            VCHANNELEDMESSAGE1(2, msg, arg0)
#define VMESSAGE2(msg, arg0, arg1)      VCHANNELEDMESSAGE2(2, msg, arg0, arg1)
#define VMESSAGE3(msg, arg0, arg1, arg2) VCHANNELEDMESSAGE3(2, msg, arg0, arg1, arg2)

#define VERRMSG0(msg)                  VMESSAGE0(msg)
#define VERRMSG1(msg, arg0)            VMESSAGE1(msg, arg0)
#define VERRMSG2(msg, arg0, arg1)      VMESSAGE2(msg, arg0, arg1)
#define VERRMSG3(msg, arg0, arg1, arg2) VMESSAGE3(msg, arg0, arg1, arg2)
#else
#define VCHANNELEDMESSAGE0(channel, msg)                        \
        do {                                                    \
            Vnm_print(channel, "%s: %s\n", __FUNCTION__, msg); \
        } while(0)

#define VCHANNELEDMESSAGE1(channel, msg, arg0)                  \
        do {                                                    \
            char buff[1000];                                    \
            snprintf( buff, 1000, msg, arg0 );                  \
            Vnm_print(channel, "%s: %s\n", __FUNCTION__, buff); \
        } while(0)

#define VCHANNELEDMESSAGE2(channel, msg, arg0, arg1)            \
        do {                                                    \
            char buff[1000];                                    \
            snprintf( buff, 1000, msg, arg0, arg1 );            \
            Vnm_print(channel, "%s: %s\n", __FUNCTION__, buff); \
        } while(0)

#define VCHANNELEDMESSAGE3(channel, msg, arg0, arg1, arg2)       \
        do {                                                     \
            char buff[1000];                                     \
            snprintf(buff, 1000, msg, arg0, arg1, arg2);         \
            Vnm_print(channel, "%s: %s\n", __FUNCTION__, buff);  \
        } while(0)

#define VMESSAGE0(msg)                  VCHANNELEDMESSAGE0(0, msg)
#define VMESSAGE1(msg, arg0)            VCHANNELEDMESSAGE1(0, msg, arg0)
#define VMESSAGE2(msg, arg0, arg1)      VCHANNELEDMESSAGE2(0, msg, arg0, arg1)
#define VMESSAGE3(msg, arg0, arg1, arg2) VCHANNELEDMESSAGE3(0, msg, arg0, arg1, arg2)

#define VERRMSG0(msg)                  VCHANNELEDMESSAGE0(2, msg)
#define VERRMSG1(msg, arg0)            VCHANNELEDMESSAGE1(2, msg, arg0)
#define VERRMSG2(msg, arg0, arg1)      VCHANNELEDMESSAGE2(2, msg, arg0, arg1)
#define VERRMSG3(msg, arg0, arg1, arg2)  VCHANNELEDMESSAGE3(2, msg, arg0, arg1, arg2)
#endif



/* Utility assertions.  If they fail, they print out messages with possible
 * arguments and then abort
 * The do{...} while(0) simply enforces that a semicolon appears at the end
 */
#ifdef DEBUG
#define VASSERT_MSG0(cnd, msg)
                                \
    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)

#define VASSERT_MSG1(cnd, msg, arg)                              \
    do {                                                         \
        if( (cnd) == 0 ) {                                       \
            char buff[1000];                                     \
            snprintf( 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)
```

```

#define VASSERT_MSG2(cnd, msg, arg0, arg1)                                  \
    do {                                                                    \
        if( (cnd) == 0 ) {                                                  \
            char buff[1000];                                                \
            snprintf( 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)
#else
#define VASSERT_MSG0(cnd, msg)
                                 \
    do {                                                                    \
        if( (cnd) == 0 ) {                                                  \
            Vnm_print(2, "%[%s()]: ERROR:\n"                                \
                         "    Assertion Failed (%s): %s\n\n",               \
                      __FUNCTION__, #cnd, msg);                             \
            abort();                                                        \
        }                                                                   \
    } while(0)

#define VASSERT_MSG1(cnd, msg, arg)                                         \
    do {                                                                    \
        if( (cnd) == 0 ) {                                                  \
            char buff[1000];                                                \
            snprintf( buff, 1000, msg, arg );                               \
            Vnm_print(2, "[%s()]: ERROR:\n"                                 \
                         "    Assertion Failed (%s): %s\n\n",               \
                      __FUNCTION__, #cnd, buff);                            \
            abort();                                                        \
        }                                                                   \
    } while(0)

#define VASSERT_MSG2(cnd, msg, arg0, arg1)                                  \
    do {                                                                    \
        if( (cnd) == 0 ) {                                                  \
            char buff[1000];                                                \
            snprintf( buff, 1000, msg, arg0, arg1 );                        \
            Vnm_print(2, "[%s()]: ERROR:\n"                                 \
                         "    Assertion Failed (%s): %s\n\n",               \
                      __FUNCTION__, #cnd, buff);                            \
            abort();                                                        \
        }                                                                   \
    } while(0)
#endif



/* Utility warning.  Tests a condition and if it fails prints out a message
 * with optional arguments
 * The do{...} while(0) simply enforces that a semicolon at the end
 */
#ifdef DEBUG
#define VWARN_MSG0(cnd, msg)                                                \
        do {                                                                \
                if( (cnd) == 0 ) {                                          \
                        Vnm_print(                                          \
                                2,                                          \
                                "%s:%d [%s()]: WARNING:\n"                  \
                                "    Condition Failed (%s):\n    %s\n\n", \
                                __FILE__,                                   \
                                __LINE__,                                   \
                                __FUNCTION__,                               \
                                #cnd,                                       \
                                msg                                         \
                                );                                          \
                }                                                           \
        } while(0)

#define VWARN_MSG1(cnd, msg, arg0)                                          \
        do {                                                                \
                if( (cnd) == 0 ) {                                          \
                        char buff[1000];                                    \
                        snprintf(buff, 1000, msg, arg0);                    \
                        Vnm_print(                                          \
                                2,                                          \
                                "%s:%d [%s()]: WARNING:\n"                  \
                                "    Condition Failed (%s):\n    %s\n\n", \
```

```
                                __FILE__,                                   \
                                __LINE__,                                   \
                                __FUNCTION__,                               \
                                #cnd,                                       \
                                buff                                        \
                                );                                          \
                        }                                                   \
        } while(0)

#define VWARN_MSG2(cnd, msg, arg0, arg1)                                    \
        do {                                                                \
                if( (cnd) == 0 ) {                                          \
                        char buff[1000];                                    \
                        snprintf(buff, 1000, msg, arg0, arg1);              \
                        Vnm_print(                                          \
                                2,                                          \
                                "%s:%d [%s()]: WARNING:\n"                  \
                                "    Condition Failed (%s):\n    %s\n\n", \
                                __FILE__,                                   \
                                __LINE__,                                   \
                                __FUNCTION__,                               \
                                #cnd,                                       \
                                buff                                        \
                                );                                          \
                }                                                           \
        } while(0)
#else
#define VWARN_MSG0(cnd, msg)                                                \
        do {                                                                \
                if( (cnd) == 0 ) {                                          \
                        Vnm_print(                                          \
                                2,                                          \
                                "[%s()]: WARNING:\n"                        \
                                "    %s\n\n",                               \
                                __FUNCTION__,                               \
                                msg                                         \
                                );                                          \
                }                                                           \
        } while(0)

#define VWARN_MSG1(cnd, msg, arg0)                                          \
        do {                                                                \
                if( (cnd) == 0 ) {                                          \
                        char buff[1000];                                    \
                        snprintf(buff, 1000, msg, arg0);                    \
                        Vnm_print(                                          \
                                2,                                          \
                                "[%s()]: WARNING:\n"                        \
                                "    %s\n\n",                               \
                                __FUNCTION__,                               \
                                buff                                        \
                                );                                          \
                }                                                           \
        } while(0)

#define VWARN_MSG2(cnd, msg, arg0, arg1)                                    \
        do {                                                                \
                if( (cnd) == 0 ) {                                          \
                        char buff[1000];                                    \
                        snprintf(buff, 1000, msg, arg0, arg1);              \
                        Vnm_print(                                          \
                                2,                                          \
                                "[%s()]: WARNING:\n"                        \
                                "    %s\n\n",                               \
                                __FUNCTION__,                               \
                                buff                                        \
                                );                                          \
                }                                                           \
        } while(0)
#endif

/* Utility Abort.  Prints a message with optional arugments and aborts */
#ifdef DEBUG
#define VABORT_MSG0(msg)                                        \
        do {                                                    \
            Vnm_print(2, "%s:%d [%s()]: ABORTING:\n"            \
                         "    %s\n\n",                          \
                      __FILE__, __LINE__, __FUNCTION__, msg);   \
            abort();                                            \
        } while(0)

```

```
#define VABORT_MSG1(msg, arg)                                   \
        do {                                                    \
            char buff[1000];                                    \
            snprintf( buff, 1000, msg, arg );                   \
            Vnm_print(2, "%s:%d [%s()]: ABORTING:\n"            \
                         "    %s\n\n",                          \
                      __FILE__, __LINE__, __FUNCTION__, buff); \
            abort();                                            \
        } while(0)

#define VABORT_MSG2(msg, arg0, arg1)                            \
        do {                                                    \
            char buff[1000];                                    \
            snprintf( buff, 1000, msg, arg0, arg1);             \
            Vnm_print(2, "%s:%d [%s()]: ABORTING:\n"            \
                         "    %s\n\n",                          \
                      __FILE__, __LINE__, __FUNCTION__, buff); \
            abort();                                            \
        } while(0)
#else
#define VABORT_MSG0(msg)                          \
        do {                                      \
            Vnm_print(2, "%[%s()]: ABORTING:\n" \
                         "    %s\n\n",            \
                      __FUNCTION__, msg);         \
            abort();                              \
        } while(0)

#define VABORT_MSG1(msg, arg)                    \
        do {                                     \
            char buff[1000];                     \
            snprintf( buff, 1000, msg, arg );  \
            Vnm_print(2, "[%s()]: ABORTING:\n" \
                         "    %s\n\n",           \
                      __FUNCTION__, buff);       \
            abort();                             \
        } while(0)

#define VABORT_MSG2(msg, arg0, arg1)                  \
        do {                                          \
            char buff[1000];                          \
            snprintf( buff, 1000, msg, arg0, arg1); \
            Vnm_print(2, "[%s()]: ABORTING:\n"      \
                         "    %s\n\n",               \
                      __FUNCTION__, buff);            \
            abort();                                  \
        } while(0)
#endif



/* Utility expression printers.  Print the expression and its value */
#ifdef DEBUG
#define PRINT_INT(expr)                                                 \
    do {                                                                \
        Vnm_print(2, "%s:%d [%s()]: %s == %d\n",                        \
                  __FILE__, __LINE__, __FUNCTION__, #expr, expr);   \
    } while(0)

#define PRINT_DBL(expr)                                                \
    do {                                                               \
        Vnm_print(2, "%s:%d [%s()]: %s == %f\n\n",                    \
                  __FILE__, __LINE__, __FUNCTION__, #expr, expr); \
    } while(0)
#else
#define PRINT_INT(expr)
#define PRINT_DBL(expr)
#endif

#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)                          \
    do {                                            \
        int fill_idx;                               \
        for (fill_idx = 0; fill_idx < n; fill_idx++) \
            vec[fill_idx] = val;                    \
    } while(0)

```

```
#define VCOPY(srcvec, dstvec, i, n) \
    do {                            \
        for (i = 0; i < n; i++)     \
            dstvec[i] = srcvec[i];  \
    } while(0)


char* wrap_text( char* str, int right_margin, int left_padding );

#define VAT(array, i) ((array)[(i) - 1])
#define RAT(array, i) ((array) + i - 1)

#endif /* #ifndef _VAPBSHAL_H_ */
```

9.43 src/generic/apbs/vmatrix.h File Reference

Contains inclusions for matrix data wrappers.

This graph shows which files directly or indirectly include this file:

Macros

- #define **MAT2**(mat, dx, dy)
- #define **RAT2**(mat, x, y) &VAT2(mat, x, y)
- #define **VAT2**(mat, x, y) mat[(y - 1) ∗ dx_##mat + (x - 1)]
- #define **MAT3**(mat, dx, dy, dz)
- #define **RAT3**(mat, x, y, z) &VAT3(mat, x, y, z)
- #define **VAT3**(mat, x, y, z)

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, dx, dy*)

Value:

```
int dx_##mat = dx;     \
    int dy_##mat = dy
```

Definition at line 64 of file vmatrix.h.

9.43.2.2 #define MAT3(*mat, dx, dy, dz*)

Value:

```
int dx_##mat = dx;         \
    int dy_##mat = dy;         \
    int dz_##mat = dz
```

Definition at line 76 of file vmatrix.h.

9.43.2.3 #define VAT3(*mat, x, y, z*)

Value:

```
mat[(z - 1) * dy_##mat * dx_##mat + \
        (y - 1) * dx_##mat + \
        (x - 1)]
```

Definition at line 84 of file vmatrix.h.

9.44 vmatrix.h

```
00001
00061 #ifndef _VMATRIX_H_
00062 #define _VMATRIX_H_
00063
00064 #define MAT2(mat, dx, dy) \
00065     int dx_##mat = dx;    \
00066     int dy_##mat = dy
00067
00068 #define RAT2(mat, x, y) \
00069     &VAT2(mat, x, y)
00070
00071 #define VAT2(mat, x, y) \
00072     mat[(y - 1) * dx_##mat + (x - 1)]
00073
00074
00075
00076 #define MAT3(mat, dx, dy, dz) \
00077     int dx_##mat = dx;        \
00078     int dy_##mat = dy;        \
00079     int dz_##mat = dz
00080
00081 #define RAT3(mat, x, y, z) \
00082     &VAT3(mat, x, y, z)
00083
00084 #define VAT3(mat, x, y, z) \
00085     mat[(z - 1) * dy_##mat * dx_##mat + \
00086         (y - 1) * dx_##mat + \
00087         (x - 1)]
00088
00089 #endif /* _VMATRIX_H_ */
```

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

Contains declarations for class Vparam.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/vunit.h"
#include "apbs/vstring.h"
```

Include dependency graph for vparam.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct sVparam_AtomData

  AtomData sub-class; stores atom data.

- struct Vparam_ResData

  ResData sub-class; stores residue data.

- struct Vparam

  Reads and assigns charge/radii parameters.

Typedefs

- typedef struct sVparam_AtomData Vparam_AtomData

  Declaration of the Vparam_AtomData class as the sVparam_AtomData structure.
- typedef struct Vparam_ResData Vparam_ResData

  Declaration of the Vparam_ResData class as the Vparam_ResData structure.
- typedef struct Vparam Vparam

  Declaration of the Vparam class as the Vparam structure.

Functions

- VEXTERNC unsigned long int Vparam_memChk (Vparam ∗thee)

  Get number of bytes in this object and its members.
- VEXTERNC Vparam_AtomData ∗ Vparam_AtomData_ctor ()

  Construct the object.
- VEXTERNC int Vparam_AtomData_ctor2 (Vparam_AtomData ∗thee)

  FORTRAN stub to construct the object.
- VEXTERNC void Vparam_AtomData_dtor (Vparam_AtomData ∗∗thee)

  Destroy object.
- VEXTERNC void Vparam_AtomData_dtor2 (Vparam_AtomData ∗thee)

  FORTRAN stub to destroy object.
- VEXTERNC void Vparam_AtomData_copyTo (Vparam_AtomData ∗thee, Vparam_AtomData ∗dest)

  Copy current atom object to destination.
- VEXTERNC void Vparam_ResData_copyTo (Vparam_ResData ∗thee, Vparam_ResData ∗dest)

  Copy current residue object to destination.
- VEXTERNC void Vparam_AtomData_copyFrom (Vparam_AtomData ∗thee, Vparam_AtomData ∗src)

  Copy current atom object from another.
- VEXTERNC Vparam_ResData ∗ Vparam_ResData_ctor (Vmem ∗mem)

  Construct the object.
- VEXTERNC int Vparam_ResData_ctor2 (Vparam_ResData ∗thee, Vmem ∗mem)

  FORTRAN stub to construct the object.
- VEXTERNC void Vparam_ResData_dtor (Vparam_ResData ∗∗thee)

  Destroy object.
- VEXTERNC void Vparam_ResData_dtor2 (Vparam_ResData ∗thee)

  FORTRAN stub to destroy object.
- VEXTERNC Vparam ∗ Vparam_ctor ()

  Construct the object.
- VEXTERNC int Vparam_ctor2 (Vparam ∗thee)

  FORTRAN stub to construct the object.
- VEXTERNC void Vparam_dtor (Vparam ∗∗thee)

  Destroy object.
- VEXTERNC void Vparam_dtor2 (Vparam ∗thee)

  FORTRAN stub to destroy object.
- VEXTERNC Vparam_ResData ∗ Vparam_getResData (Vparam ∗thee, char resName[VMAX_ARGLEN])

  Get residue data.
- VEXTERNC Vparam_AtomData ∗ Vparam_getAtomData (Vparam ∗thee, char resName[VMAX_ARGLEN], char atomName[VMAX_ARGLEN])

Get atom data.

- VEXTERNC int Vparam_readFlatFile (Vparam *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname)

  Read a flat-file format parameter database.

- VEXTERNC int Vparam_readXMLFile (Vparam *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname)

  Read an XML format parameter database.

9.45.1 Detailed Description

Contains declarations for class Vparam.

Version

Id:

vparam.h 1774 2012-07-19 18:46:14Z sobolevnrm

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
00132   Vmem *vmem;
00133   int nResData;
00135   Vparam_ResData *resData;
00136 };
00137
00142 typedef struct Vparam Vparam;
00143
00144 /* ////////////////////////////////////////////////////////////////////////////
00145 // Class Vparam: Inlineable methods (vparam.c)
00147
00148 #if !defined(VINLINE_VPARAM)
00149
00156     VEXTERNC unsigned long int Vparam_memChk(Vparam *thee);
00157
00158 #else /* if defined(VINLINE_VPARAM) */
00159
00160 #   define Vparam_memChk(thee) (Vmem_bytes((thee)->vmem))
00161
00162 #endif /* if !defined(VINLINE_VPARAM) */
00163
00164 /* ////////////////////////////////////////////////////////////////////////////
00165 // Class Vparam: Non-Inlineable methods (vparam.c)
00167
00172 VEXTERNC Vparam_AtomData* Vparam_AtomData_ctor();
00173
00179 VEXTERNC int Vparam_AtomData_ctor2(Vparam_AtomData *thee);
```

```
VEXTERNC void Vparam_AtomData_dtor(Vparam_AtomData **thee);

VEXTERNC void Vparam_AtomData_dtor2(Vparam_AtomData *thee);

VEXTERNC void Vparam_AtomData_copyTo(Vparam_AtomData *thee,
  Vparam_AtomData *dest);

VEXTERNC void Vparam_ResData_copyTo(Vparam_ResData *thee,
  Vparam_ResData *dest);

VEXTERNC void Vparam_AtomData_copyFrom(Vparam_AtomData *thee,
  Vparam_AtomData *src);

VEXTERNC Vparam_ResData* Vparam_ResData_ctor(Vmem *mem);

VEXTERNC int Vparam_ResData_ctor2(Vparam_ResData *thee, Vmem *mem);

VEXTERNC void Vparam_ResData_dtor(Vparam_ResData **thee);

VEXTERNC void Vparam_ResData_dtor2(Vparam_ResData *thee);

VEXTERNC Vparam* Vparam_ctor();

VEXTERNC int Vparam_ctor2(Vparam *thee);

VEXTERNC void Vparam_dtor(Vparam **thee);

VEXTERNC void Vparam_dtor2(Vparam *thee);

VEXTERNC Vparam_ResData* Vparam_getResData(Vparam *thee,
  char resName[VMAX_ARGLEN]);

VEXTERNC Vparam_AtomData* Vparam_getAtomData(Vparam *thee,
  char resName[VMAX_ARGLEN], char atomName[VMAX_ARGLEN]);

VEXTERNC int Vparam_readFlatFile(Vparam *thee, const char *iodev,
  const char *iofmt, const char *thost, const char *fname);

VEXTERNC int Vparam_readXMLFile(Vparam *thee, const char *iodev,
  const char *iofmt, const char *thost, const char *fname);

#endif    /* ifndef _VPARAM_H_ */
```

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

Contains declarations for class Vpbe.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/vunit.h"
#include "apbs/vatom.h"
#include "apbs/vacc.h"
#include "apbs/vclist.h"
```

Include dependency graph for vpbe.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct sVpbe

  Contains public data members for Vpbe class/module.

Typedefs

- typedef struct sVpbe Vpbe

  Declaration of the Vpbe class as the Vpbe structure.

Functions

- VEXTERNC Valist ∗ Vpbe_getValist (Vpbe ∗thee)

  Get atom list.

- VEXTERNC Vacc * Vpbe_getVacc (Vpbe *thee)

  Get accessibility oracle.
- VEXTERNC double Vpbe_getBulkIonicStrength (Vpbe *thee)

  Get bulk ionic strength.
- VEXTERNC double Vpbe_getMaxIonRadius (Vpbe *thee)

  Get maximum radius of ion species.
- VEXTERNC double Vpbe_getTemperature (Vpbe *thee)

  Get temperature.
- VEXTERNC double Vpbe_getSoluteDiel (Vpbe *thee)

  Get solute dielectric constant.
- VEXTERNC double Vpbe_getGamma (Vpbe *thee)

  Get apolar coefficient.
- VEXTERNC double Vpbe_getSoluteRadius (Vpbe *thee)

  Get sphere radius which bounds biomolecule.
- VEXTERNC double Vpbe_getSoluteXlen (Vpbe *thee)

  Get length of solute in x dimension.
- VEXTERNC double Vpbe_getSoluteYlen (Vpbe *thee)

  Get length of solute in y dimension.
- VEXTERNC double Vpbe_getSoluteZlen (Vpbe *thee)

  Get length of solute in z dimension.
- VEXTERNC double * Vpbe_getSoluteCenter (Vpbe *thee)

  Get coordinates of solute center.
- VEXTERNC double Vpbe_getSoluteCharge (Vpbe *thee)

  Get total solute charge.
- VEXTERNC double Vpbe_getSolventDiel (Vpbe *thee)

  Get solvent dielectric constant.
- VEXTERNC double Vpbe_getSolventRadius (Vpbe *thee)

  Get solvent molecule radius.
- VEXTERNC double Vpbe_getXkappa (Vpbe *thee)

  Get Debye-Huckel parameter.
- VEXTERNC double Vpbe_getDeblen (Vpbe *thee)

  Get Debye-Huckel screening length.
- VEXTERNC double Vpbe_getZkappa2 (Vpbe *thee)

  Get modified squared Debye-Huckel parameter.
- VEXTERNC double Vpbe_getZmagic (Vpbe *thee)

  Get charge scaling factor.
- VEXTERNC double Vpbe_getzmem (Vpbe *thee)

  Get z position of the membrane bottom.
- VEXTERNC double Vpbe_getLmem (Vpbe *thee)

  Get length of the membrane (A)
  aauthor Michael Grabe.
- VEXTERNC double Vpbe_getmembraneDiel (Vpbe *thee)

  Get membrane dielectric constant.
- VEXTERNC double Vpbe_getmemv (Vpbe *thee)

  Get membrane potential (kT)
- VEXTERNC Vpbe * Vpbe_ctor (Valist *alist, int ionNum, double *ionConc, double *ionRadii, double *ionQ, double T, double soluteDiel, double solventDiel, double solventRadius, int focusFlag, double sdens, double z_mem, double L, double membraneDiel, double V)

Construct Vpbe object.

- VEXTERNC int Vpbe_ctor2 (Vpbe ∗thee, Valist ∗alist, int ionNum, double ∗ionConc, double ∗ionRadii, double ∗ionQ, double T, double soluteDiel, double solventDiel, double solventRadius, int focusFlag, double sdens, double z_mem, double L, double membraneDiel, double V)

  FORTRAN stub to construct Vpbe objct.

- VEXTERNC int Vpbe_getIons (Vpbe ∗thee, int ∗nion, double ionConc[MAXION], double ionRadii[MAXION], double ionQ[MAXION])

  Get information about the counterion species present.

- VEXTERNC void Vpbe_dtor (Vpbe ∗∗thee)

  Object destructor.

- VEXTERNC void Vpbe_dtor2 (Vpbe ∗thee)

  FORTRAN stub object destructor.

- VEXTERNC double Vpbe_getCoulombEnergy1 (Vpbe ∗thee)

  Calculate coulombic energy of set of charges.

- VEXTERNC unsigned long int Vpbe_memChk (Vpbe ∗thee)

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

9.47.1 Detailed Description

Contains declarations for class Vpbe.

Version

Id:

vpbe.h 1774 2012-07-19 18:46:14Z sobolevnrm

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
00086  Vmem *vmem;
00088  Valist *alist;
00089  Vclist *clist;
00090  Vacc *acc;
00092  double T;
00093  double soluteDiel;
00094  double solventDiel;
00095  double solventRadius;
00099  double bulkIonicStrength;
00100  double maxIonRadius;
00103  int    numIon;
00104  double ionConc[MAXION];
00105  double ionRadii[MAXION];
00106  double ionQ[MAXION];
00108  double xkappa;
00109  double deblen;
00110  double zkappa2;
00111  double zmagic;
00113  double soluteCenter[3];
00114  double soluteRadius;
00115  double soluteXlen;
00116  double soluteYlen;
00117  double soluteZlen;
00118  double soluteCharge;
00120  double smvolume;
```

```
00121  double smsize;
00122  int ipkey;
00125  int paramFlag;
00127  /*---------------------------------------------------------*/
00128  /* Added by Michael Grabe                                  */
00129  /*---------------------------------------------------------*/
00130
00131  double z_mem;
00132  double L;
00133  double membraneDiel;
00134  double V;
00135  int param2Flag;
00136  /*---------------------------------------------------------*/
00137
00138 };
00139
00144 typedef struct sVpbe Vpbe;
00145
00146 /* ////////////////////////////////////////////////////////////////////////////
00147    // Class Vpbe: Inlineable methods (vpbe.c)
00149
00150 #if !defined(VINLINE_VPBE)
00151
00158 VEXTERNC Valist* Vpbe_getValist(Vpbe *thee);
00159
00166 VEXTERNC Vacc*   Vpbe_getVacc(Vpbe *thee);
00167
00174 VEXTERNC double  Vpbe_getBulkIonicStrength(Vpbe *thee);
00175
00182 VEXTERNC double  Vpbe_getMaxIonRadius(Vpbe *thee);
00183
00190 VEXTERNC double  Vpbe_getTemperature(Vpbe *thee);
00191
00198 VEXTERNC double  Vpbe_getSoluteDiel(Vpbe *thee);
00199
00206 VEXTERNC double  Vpbe_getGamma(Vpbe *thee);
00207
00214 VEXTERNC double  Vpbe_getSoluteRadius(Vpbe *thee);
00215
00222 VEXTERNC double  Vpbe_getSoluteXlen(Vpbe *thee);
00223
00230 VEXTERNC double  Vpbe_getSoluteYlen(Vpbe *thee);
00231
00238 VEXTERNC double  Vpbe_getSoluteZlen(Vpbe *thee);
00239
00246 VEXTERNC double* Vpbe_getSoluteCenter(Vpbe *thee);
00247
00254 VEXTERNC double  Vpbe_getSoluteCharge(Vpbe *thee);
00255
00262 VEXTERNC double  Vpbe_getSolventDiel(Vpbe *thee);
00263
00270 VEXTERNC double  Vpbe_getSolventRadius(Vpbe *thee);
00271
00278 VEXTERNC double  Vpbe_getXkappa(Vpbe *thee);
00279
00286 VEXTERNC double  Vpbe_getDeblen(Vpbe *thee);
00287
00294 VEXTERNC double  Vpbe_getZkappa2(Vpbe *thee);
00295
00302 VEXTERNC double  Vpbe_getZmagic(Vpbe *thee);
00303
00304 /*-------------------------------------------------------------*/
00305 /* Added by Michael Grabe                                      */
00306 /*-------------------------------------------------------------*/
00307
00314 VEXTERNC double  Vpbe_getzmem(Vpbe *thee);
00315
00322 VEXTERNC double  Vpbe_getLmem(Vpbe *thee);
00323
00330 VEXTERNC double  Vpbe_getmembraneDiel(Vpbe *thee);
00331
00337 VEXTERNC double  Vpbe_getmemv(Vpbe *thee);
00338
00339 /*-------------------------------------------------------------*/
00340
00341 #else /* if defined(VINLINE_VPBE) */
00342 #   define Vpbe_getValist(thee) ((thee)->alist)
00343 #   define Vpbe_getVacc(thee) ((thee)->acc)
00344 #   define Vpbe_getBulkIonicStrength(thee) ((thee)->bulkIonicStrength)
00345 #   define Vpbe_getTemperature(thee) ((thee)->T)
00346 #   define Vpbe_getSoluteDiel(thee) ((thee)->soluteDiel)
```

```
#   define Vpbe_getSoluteCenter(thee) ((thee)->soluteCenter)
#   define Vpbe_getSoluteRadius(thee) ((thee)->soluteRadius)
#   define Vpbe_getSoluteXlen(thee) ((thee)->soluteXlen)
#   define Vpbe_getSoluteYlen(thee) ((thee)->soluteYlen)
#   define Vpbe_getSoluteZlen(thee) ((thee)->soluteZlen)
#   define Vpbe_getSoluteCharge(thee) ((thee)->soluteCharge)
#   define Vpbe_getSolventDiel(thee) ((thee)->solventDiel)
#   define Vpbe_getSolventRadius(thee) ((thee)->solventRadius)
#   define Vpbe_getMaxIonRadius(thee) ((thee)->maxIonRadius)
#   define Vpbe_getXkappa(thee) ((thee)->xkappa)
#   define Vpbe_getDeblen(thee) ((thee)->deblen)
#   define Vpbe_getZkappa2(thee) ((thee)->zkappa2)
#   define Vpbe_getZmagic(thee) ((thee)->zmagic)

/*-----------------------------------------------------------*/
/* Added by Michael Grabe                                    */
/*-----------------------------------------------------------*/

#   define Vpbe_getzmem(thee) ((thee)->z_mem)
#   define Vpbe_getLmem(thee) ((thee)->L)
#   define Vpbe_getmembraneDiel(thee) ((thee)->membraneDiel)
#   define Vpbe_getmemv(thee) ((thee)->V)

/*-----------------------------------------------------------*/


#endif /* if !defined(VINLINE_VPBE) */

/* ////////////////////////////////////////////////////////////////////////////
   // Class Vpbe: Non-Inlineable methods (vpbe.c)

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
       );

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
       );

VEXTERNC int     Vpbe_getIons(Vpbe *thee, int *nion, double ionConc[MAXION],
         double ionRadii[MAXION], double ionQ[MAXION]);

VEXTERNC void    Vpbe_dtor(Vpbe **thee);

VEXTERNC void    Vpbe_dtor2(Vpbe *thee);

VEXTERNC double  Vpbe_getCoulombEnergy1(Vpbe *thee);

VEXTERNC unsigned long int Vpbe_memChk(Vpbe *thee);

#endif /* ifndef _VPBE_H_ */
```

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

Contains declarations for class Vstring.

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

Include dependency graph for vstring.h:

This graph shows which files directly or indirectly include this file:

Functions

- VEXTERNC int Vstring_strcasecmp (const char ∗s1, const char ∗s2)

  Case-insensitive string comparison (BSD standard)
- VEXTERNC int Vstring_isdigit (const char ∗tok)

  A modified sscanf that examines the complete string.
- 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 1774 2012-07-19 18:46:14Z sobolevnrm

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 "maloc/maloc.h"
00082 #include "apbs/vhal.h"
00083
00096 VEXTERNC int Vstring_strcasecmp(const char *s1, const char *
      s2);
00097
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

  ${e_c}^2/$ *in ESU units => kcal/mol*
- #define Vunit_esu_kb 1.9871913e-03

  k_b *in ESU units => 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 1774 2012-07-19 18:46:14Z sobolevnrm

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
```

```
#define Vunit_C_to_ec  6.2415065e+18

#define Vunit_ec  1.6021773e-19

#define Vunit_kb  1.3806581e-23

#define Vunit_Na  6.0221367e+23

#define Vunit_pi  VPI

#define Vunit_eps0  8.8541878e-12

#define Vunit_esu_ec2A  3.3206364e+02

#define Vunit_esu_kb            1.9871913e-03

#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 1793 2012-07-28 00:54:24Z sobolevnrm

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 1793 2012-07-28 00:54:24Z sobolevnrm $")
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;
```

```
    thee = (APOLparm*)Vmem_malloc(VNULL, 1, sizeof(APOLparm));
    VASSERT( thee != VNULL);
    VASSERT( APOLparm_ctor2(thee) == VRC_SUCCESS );

    return thee;
}

VPUBLIC Vrc_Codes APOLparm_ctor2(APOLparm *thee) {

    int i;

    if (thee == VNULL) return VRC_FAILURE;

    thee->parsed = 0;

    thee->setgrid = 0;
    thee->setmolid = 0;
    thee->setbconc = 0;
    thee->setsdens = 0;
    thee->setdpos = 0;
    thee->setpress = 0;
    thee->setsrfm = 0;
    thee->setsrad = 0;
    thee->setswin = 0;

 thee->settemp = 0;
    thee->setgamma = 0;

 thee->setwat = 0;

 thee->sav = 0.0;
 thee->sasa = 0.0;
 thee->wcaEnergy = 0.0;

 for(i=0;i<3;i++) thee->totForce[i] = 0.0;

    return VRC_SUCCESS;
}

VPUBLIC void APOLparm_copy(
        APOLparm *thee,
        APOLparm *source
        ) {

 int i;

 thee->parsed = source->parsed;

 for (i=0; i<3; i++) thee->grid[i] = source->grid[i];
 thee->setgrid = source->setgrid;

 thee->molid = source->molid;
 thee->setmolid = source->setmolid;

 thee->bconc = source->bconc ;
 thee->setbconc= source->setbconc ;

 thee->sdens = source->sdens ;
 thee->setsdens= source->setsdens ;

 thee->dpos = source->dpos ;
 thee->setdpos= source->setdpos ;

 thee->press = source->press ;
 thee->setpress = source->setpress ;

 thee->srfm = source->srfm ;
 thee->setsrfm = source->setsrfm ;

 thee->srad = source->srad ;
 thee->setsrad = source->setsrad ;

 thee->swin = source->swin ;
 thee->setswin = source->setswin ;

 thee->temp = source->temp ;
 thee->settemp = source->settemp ;

 thee->gamma = source->gamma ;
 thee->setgamma = source->setgamma ;

```

```
 thee->calcenergy = source->calcenergy ;
 thee->setcalcenergy = source->setcalcenergy ;

 thee->calcforce = source->calcforce ;
 thee->setcalcforce = source->setcalcforce ;

 thee->setwat = source->setwat ;

 thee->sav = source->sav;
 thee->sasa = source->sasa;
 thee->wcaEnergy = source->wcaEnergy;

 for(i=0;i<3;i++) thee->totForce[i] = source->totForce[i];

 return;
}

VPUBLIC void APOLparm_dtor(APOLparm **thee) {
    if ((*thee) != VNULL) {
        APOLparm_dtor2(*thee);
        Vmem_free(VNULL, 1, sizeof(APOLparm), (void **)thee);
        (*thee) = VNULL;
    }

 return;
}

VPUBLIC void APOLparm_dtor2(APOLparm *thee) { ; }

VPUBLIC Vrc_Codes APOLparm_check(APOLparm *thee) {


    Vrc_Codes rc;
    rc = VRC_SUCCESS;

    if (!thee->parsed) {
        Vnm_print(2, "APOLparm_check:  not filled!\n");
        return VRC_FAILURE;
    }
    if (!thee->setgrid) {
        Vnm_print(2, "APOLparm_check:  grid not set!\n");
        rc = VRC_FAILURE;
    }
    if (!thee->setmolid) {
        Vnm_print(2, "APOLparm_check:  molid not set!\n");
        rc = VRC_FAILURE;
    }
    if (!thee->setbconc) {
        Vnm_print(2, "APOLparm_check:  bconc not set!\n");
        rc = VRC_FAILURE;
    }
    if (!thee->setsdens) {
        Vnm_print(2, "APOLparm_check:  sdens not set!\n");
        rc = VRC_FAILURE;
    }
    if (!thee->setdpos) {
        Vnm_print(2, "APOLparm_check:  dpos not set!\n");
        rc = VRC_FAILURE;
    }
    if (!thee->setpress) {
        Vnm_print(2, "APOLparm_check:  press not set!\n");
        rc = VRC_FAILURE;
    }
    if (!thee->setsrfm) {
        Vnm_print(2, "APOLparm_check:  srfm not set!\n");
        rc = VRC_FAILURE;
    }
    if (!thee->setsrad) {
        Vnm_print(2, "APOLparm_check:  srad not set!\n");
        rc = VRC_FAILURE;
    }
    if (!thee->setswin) {
        Vnm_print(2, "APOLparm_check:  swin not set!\n");
        rc = VRC_FAILURE;
    }
    if (!thee->settemp) {
        Vnm_print(2, "APOLparm_check:  temp not set!\n");
        rc = VRC_FAILURE;
    }
 if (!thee->setgamma) {
        Vnm_print(2, "APOLparm_check:  gamma not set!\n");
```

```
        rc = VRC_FAILURE;
    }
    return rc;

}

VPRIVATE Vrc_Codes APOLparm_parseGRID(APOLparm *thee, Vio *sock) {

 char tok[VMAX_BUFSIZE];
 double tf;

 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
 if (sscanf(tok, "%lf", &tf) == 0) {
  Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GRID \
keyword!\n", tok);
  return VRC_WARNING;
 } else thee->grid[0] = tf;
 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
 if (sscanf(tok, "%lf", &tf) == 0) {
  Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GRID \
keyword!\n", tok);
  return VRC_WARNING;
 } else thee->grid[1] = tf;
 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
 if (sscanf(tok, "%lf", &tf) == 0) {
  Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GRID \
keyword!\n", tok);
  return VRC_WARNING;
 } else thee->grid[2] = tf;
    thee->setgrid = 1;
    return VRC_SUCCESS;

VERROR1:
        Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
 return VRC_WARNING;
}

VPRIVATE Vrc_Codes APOLparm_parseMOL(APOLparm *thee, Vio *sock) {
    int ti;
    char tok[VMAX_BUFSIZE];

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%d", &ti) == 0) {
        Vnm_print(2, "NOsh:  Read non-int (%s) while parsing MOL \
keyword!\n", tok);
        return VRC_WARNING;
    }
    thee->molid = ti;
    thee->setmolid = 1;
    return VRC_SUCCESS;

VERROR1:
        Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
 return VRC_WARNING;
}

VPRIVATE Vrc_Codes APOLparm_parseSRFM(APOLparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);

    if (Vstring_strcasecmp(tok, "sacc") == 0) {
        thee->srfm = VSM_MOL;
        thee->setsrfm = 1;
        return VRC_SUCCESS;
    } else {
        Vnm_print(2, "parseAPOL: Unrecongnized keyword (%s) when parsing srfm!
\n", tok);
  Vnm_print(2, "parseAPOL: Accepted values for srfm = sacc\n");
        return VRC_WARNING;
    }

    return VRC_FAILURE;

VERROR1:
        Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
 return VRC_WARNING;
}

VPRIVATE Vrc_Codes APOLparm_parseSRAD(APOLparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
```

```
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SRAD \
keyword!\n", tok);
        return VRC_WARNING;
    }
    thee->srad = tf;
    thee->setsrad = 1;
    return VRC_SUCCESS;

VERROR1:
        Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
 return VRC_WARNING;
}

VPRIVATE Vrc_Codes APOLparm_parseSWIN(APOLparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SWIN \
keyword!\n", tok);
        return VRC_WARNING;
    }
    thee->swin = tf;
    thee->setswin = 1;
    return VRC_SUCCESS;

VERROR1:
        Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
 return VRC_WARNING;
}

VPRIVATE Vrc_Codes APOLparm_parseTEMP(APOLparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing TEMP \
keyword!\n", tok);
        return VRC_WARNING;
    }
    thee->temp = tf;
    thee->settemp = 1;
    return VRC_SUCCESS;

VERROR1:
        Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
 return VRC_WARNING;
}

VPRIVATE Vrc_Codes APOLparm_parseGAMMA(APOLparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GAMMA \
keyword!\n", tok);
        return VRC_WARNING;
    }
    thee->gamma = tf;
    thee->setgamma = 1;
    return VRC_SUCCESS;

VERROR1:
        Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
 return VRC_WARNING;
}

VPRIVATE Vrc_Codes APOLparm_parseCALCENERGY(APOLparm *thee, Vio *sock)
{
    char tok[VMAX_BUFSIZE];
    int ti;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    /* Parse number */
```

```
    if (sscanf(tok, "%d", &ti) == 1) {
        thee->calcenergy = (APOLparm_calcEnergy)ti
;
        thee->setcalcenergy = 1;

        Vnm_print(2, "parseAPOL:  Warning -- parsed deprecated \"calcenergy \
%d\" statement.\n", ti);
        Vnm_print(2, "parseAPOL:  Please use \"calcenergy ");
        switch (thee->calcenergy) {
            case ACE_NO:
                Vnm_print(2, "no");
                break;
            case ACE_TOTAL:
                Vnm_print(2, "total");
                break;
            case ACE_COMPS:
                Vnm_print(2, "comps");
                break;
            default:
                Vnm_print(2, "UNKNOWN");
                break;
        }
        Vnm_print(2, "\" instead.\n");
        return VRC_SUCCESS;
    } else if (Vstring_strcasecmp(tok, "no") == 0) {
        thee->calcenergy = ACE_NO;
        thee->setcalcenergy = 1;
        return VRC_SUCCESS;
    } else if (Vstring_strcasecmp(tok, "total") == 0) {
        thee->calcenergy = ACE_TOTAL;
        thee->setcalcenergy = 1;
        return VRC_SUCCESS;
    } else if (Vstring_strcasecmp(tok, "comps") == 0) {
        thee->calcenergy = ACE_COMPS;
        thee->setcalcenergy = 1;
        return VRC_SUCCESS;
    } else {
        Vnm_print(2, "NOsh:  Unrecognized parameter (%s) while parsing \
calcenergy!\n", tok);
        return VRC_WARNING;
    }
    return VRC_FAILURE;

VERROR1:
        Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
 return VRC_WARNING;
}

VPRIVATE Vrc_Codes APOLparm_parseCALCFORCE(APOLparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    int ti;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    /* Parse number */
    if (sscanf(tok, "%d", &ti) == 1) {
        thee->calcforce = (APOLparm_calcForce)ti;
        thee->setcalcforce = 1;

        Vnm_print(2, "parseAPOL:  Warning -- parsed deprecated \"calcforce \
%d\" statement.\n", ti);
        Vnm_print(2, "parseAPOL:  Please use \"calcforce ");
        switch (thee->calcenergy) {
            case ACF_NO:
                Vnm_print(2, "no");
                break;
            case ACF_TOTAL:
                Vnm_print(2, "total");
                break;
            case ACF_COMPS:
                Vnm_print(2, "comps");
                break;
            default:
                Vnm_print(2, "UNKNOWN");
                break;
        }
        Vnm_print(2, "\" instead.\n");
        return VRC_SUCCESS;
    } else if (Vstring_strcasecmp(tok, "no") == 0) {
        thee->calcforce = ACF_NO;
        thee->setcalcforce = 1;
        return VRC_SUCCESS;
```

```
    } else if (Vstring_strcasecmp(tok, "total") == 0) {
        thee->calcforce = ACF_TOTAL;
        thee->setcalcforce = 1;
        return VRC_SUCCESS;
    } else if (Vstring_strcasecmp(tok, "comps") == 0) {
        thee->calcforce = ACF_COMPS;
        thee->setcalcforce = 1;
        return VRC_SUCCESS;
    } else {
        Vnm_print(2, "NOsh:  Unrecognized parameter (%s) while parsing \
calcforce!\n", tok);
        return VRC_WARNING;
    }
    return VRC_FAILURE;

VERROR1:
        Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
 return VRC_WARNING;
}

VPRIVATE Vrc_Codes APOLparm_parseBCONC(APOLparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing BCONC \
keyword!\n", tok);
        return VRC_WARNING;
    }
    thee->bconc = tf;
    thee->setbconc = 1;
    return VRC_SUCCESS;

VERROR1:
        Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
 return VRC_WARNING;
}

VPRIVATE Vrc_Codes APOLparm_parseSDENS(APOLparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SDENS \
keyword!\n", tok);
        return VRC_WARNING;
    }
    thee->sdens = tf;
    thee->setsdens = 1;
    return VRC_SUCCESS;

VERROR1:
        Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
 return VRC_WARNING;
}

VPRIVATE Vrc_Codes APOLparm_parseDPOS(APOLparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SDENS \
keyword!\n", tok);
        return VRC_WARNING;
    }
    thee->dpos = tf;
    thee->setdpos = 1;

 if(thee->dpos < 0.001){
  Vnm_print(1,"\nWARNING WARNING WARNING WARNING WARNING\n");
  Vnm_print(1,"NOsh: dpos is set to a very small value.\n");
  Vnm_print(1,"NOsh: If you are not using a PQR file, you can \
safely ignore this message.\n");
  Vnm_print(1,"NOsh: Otherwise please choose a value greater than \
or equal to 0.001.\n\n");
 }

    return VRC_SUCCESS;
```

```
VERROR1:
        Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
 return VRC_WARNING;
}

VPRIVATE Vrc_Codes APOLparm_parsePRESS(APOLparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing PRESS \
keyword!\n", tok);
        return VRC_WARNING;
    }
    thee->press = tf;
    thee->setpress = 1;
    return VRC_SUCCESS;

VERROR1:
        Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
 return VRC_WARNING;
}

VPUBLIC Vrc_Codes APOLparm_parseToken(APOLparm *thee
, char tok[VMAX_BUFSIZE],
  Vio *sock) {

    if (thee == VNULL) {
        Vnm_print(2, "parseAPOL:  got NULL thee!\n");
        return VRC_WARNING;
    }

    if (sock == VNULL) {
        Vnm_print(2, "parseAPOL:  got NULL socket!\n");
        return VRC_WARNING;
    }

 if (Vstring_strcasecmp(tok, "mol") == 0) {
  return APOLparm_parseMOL(thee, sock);
 } else if (Vstring_strcasecmp(tok, "grid") == 0) {
  return APOLparm_parseGRID(thee, sock);
 } else if (Vstring_strcasecmp(tok, "dime") == 0) {
  Vnm_print(2, "APOLparm_parseToken:  The DIME and GLEN keywords for APOLAR
 have been replaced with GRID.\n");
  Vnm_print(2, "APOLparm_parseToken:  Please see the APBS User Guide for more
 information.\n");
  return VRC_WARNING;
 } else if (Vstring_strcasecmp(tok, "glen") == 0) {
  Vnm_print(2, "APOLparm_parseToken:  The DIME and GLEN keywords for APOLAR
 have been replaced with GRID.\n");
  Vnm_print(2, "APOLparm_parseToken:  Please see the APBS User Guide for more
 information.\n");
  return VRC_WARNING;
 } else if (Vstring_strcasecmp(tok, "bconc") == 0) {
  return APOLparm_parseBCONC(thee, sock);
    } else if (Vstring_strcasecmp(tok, "sdens") == 0) {
        return APOLparm_parseSDENS(thee, sock);
    } else if (Vstring_strcasecmp(tok, "dpos") == 0) {
        return APOLparm_parseDPOS(thee, sock);
    }  else if (Vstring_strcasecmp(tok, "srfm") == 0) {
        return APOLparm_parseSRFM(thee, sock);
    } else if (Vstring_strcasecmp(tok, "srad") == 0) {
        return APOLparm_parseSRAD(thee, sock);
    } else if (Vstring_strcasecmp(tok, "swin") == 0) {
        return APOLparm_parseSWIN(thee, sock);
    } else if (Vstring_strcasecmp(tok, "temp") == 0) {
        return APOLparm_parseTEMP(thee, sock);
    } else if (Vstring_strcasecmp(tok, "gamma") == 0) {
        return APOLparm_parseGAMMA(thee, sock);
    } else if (Vstring_strcasecmp(tok, "press") == 0) {
        return APOLparm_parsePRESS(thee, sock);
    } else if (Vstring_strcasecmp(tok, "calcenergy") == 0) {
        return APOLparm_parseCALCENERGY(thee, sock);
    } else if (Vstring_strcasecmp(tok, "calcforce") == 0) {
        return APOLparm_parseCALCFORCE(thee, sock);
    } else {
        Vnm_print(2, "parseAPOL:  Unrecognized keyword (%s)!\n", tok);
        return VRC_WARNING;
    }
```

```
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 1774 2012-07-19 18:46:14Z sobolevnrm

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

```
#include "apbscfg.h"
#include "apbs/femparm.h"

VEMBED(rcsid="$Id: femparm.c 1774 2012-07-19 18:46:14Z sobolevnrm $")

#if !defined(VINLINE_MGPARM)

#endif /* if !defined(VINLINE_MGPARM) */

VPUBLIC FEMparm* FEMparm_ctor(FEMparm_CalcType
 type) {

    /* Set up the structure */
    FEMparm *thee = VNULL;
    thee = (FEMparm*)Vmem_malloc(VNULL, 1, sizeof(FEMparm));
    VASSERT( thee != VNULL);
    VASSERT( FEMparm_ctor2(thee, type) );

    return thee;
}

VPUBLIC int FEMparm_ctor2(FEMparm *thee,
                          FEMparm_CalcType type
                          ) {

    if (thee == VNULL) return 0;

    thee->parsed = 0;
    thee->type = type;
    thee->settype = 1;

    thee->setglen = 0;
    thee->setetol = 0;
    thee->setekey = 0;
    thee->setakeyPRE = 0;
    thee->setakeySOLVE = 0;
    thee->settargetNum = 0;
    thee->settargetRes = 0;
    thee->setmaxsolve = 0;
```

```
    thee->setmaxvert = 0;
 thee->useMesh = 0;

    return 1;
}

VPUBLIC void FEMparm_copy(
        FEMparm *thee,
        FEMparm *source
        ) {

 int i;

 thee->parsed = source->parsed;
 thee->type = source->type;
 thee->settype = source->settype;
 for (i=0; i<3; i++) thee->glen[i] = source->glen[i];
 thee->setglen = source->setglen;
 thee->etol = source->etol;
 thee->setetol = source->setetol;
 thee->ekey = source->ekey;
 thee->setekey = source->setekey;
 thee->akeyPRE = source->akeyPRE;
 thee->setakeyPRE = source->setakeyPRE;
 thee->akeySOLVE = source->akeySOLVE;
 thee->setakeySOLVE = source->setakeySOLVE;
 thee->targetNum = source->targetNum;
 thee->settargetNum = source->settargetNum;
 thee->targetRes = source->targetRes;
 thee->settargetRes = source->settargetRes;
 thee->maxsolve = source->maxsolve;
 thee->setmaxsolve = source->setmaxsolve;
 thee->maxvert = source->maxvert;
 thee->setmaxvert = source->setmaxvert;
 thee->pkey = source->pkey;
 thee->useMesh = source->useMesh;
 thee->meshID = source->meshID;
}

VPUBLIC void FEMparm_dtor(FEMparm **thee) {
    if ((*thee) != VNULL) {
        FEMparm_dtor2(*thee);
        Vmem_free(VNULL, 1, sizeof(FEMparm), (void **)thee);
        (*thee) = VNULL;
    }
}

VPUBLIC void FEMparm_dtor2(FEMparm *thee) { ; }

VPUBLIC int FEMparm_check(FEMparm *thee) {

    int rc;
    rc = 1;

    if (!thee->parsed) {
        Vnm_print(2, "FEMparm_check:  not filled!\n");
        return 0;
    }
    if (!thee->settype) {
        Vnm_print(2, "FEMparm_check:  type not set!\n");
        rc = 0;
    }
    if (!thee->setglen) {
        Vnm_print(2, "FEMparm_check:  glen not set!\n");
        rc = 0;
    }
    if (!thee->setetol) {
        Vnm_print(2, "FEMparm_check:  etol not set!\n");
        rc = 0;
    }
    if (!thee->setekey) {
        Vnm_print(2, "FEMparm_check:  ekey not set!\n");
        rc = 0;
    }
    if (!thee->setakeyPRE) {
        Vnm_print(2, "FEMparm_check:  akeyPRE not set!\n");
        rc = 0;
    }
    if (!thee->setakeySOLVE) {
        Vnm_print(2, "FEMparm_check:  akeySOLVE not set!\n");
        rc = 0;
```

```
    }
    if (!thee->settargetNum) {
        Vnm_print(2, "FEMparm_check:  targetNum not set!\n");
        rc = 0;
    }
    if (!thee->settargetRes) {
        Vnm_print(2, "FEMparm_check:  targetRes not set!\n");
        rc = 0;
    }
    if (!thee->setmaxsolve) {
        Vnm_print(2, "FEMparm_check:  maxsolve not set!\n");
        rc = 0;
    }
    if (!thee->setmaxvert) {
        Vnm_print(2, "FEMparm_check:  maxvert not set!\n");
        rc = 0;
    }

    return rc;
}

VPRIVATE Vrc_Codes FEMparm_parseDOMAINLENGTH(FEMparm *thee,
                                             Vio *sock
                                             ) {

    int i;
    double tf;
    char tok[VMAX_BUFSIZE];

    for (i=0; i<3; i++) {
        VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
        if (sscanf(tok, "%lf", &tf) == 0) {
            Vnm_print(2, "parseFE:  Read non-double (%s) while parsing \
DOMAINLENGTH keyword!\n", tok);
            return VRC_FAILURE;
        }
        thee->glen[i] = tf;
    }
    thee->setglen = 1;
    return VRC_SUCCESS;
VERROR1:
    Vnm_print(2, "parseFE:  ran out of tokens!\n");
    return VRC_FAILURE;

}

VPRIVATE Vrc_Codes FEMparm_parseETOL(FEMparm *thee,
                                     Vio *sock
                                     ) {

    double tf;
    char tok[VMAX_BUFSIZE];

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "parseFE:  Read non-double (%s) while parsing \
ETOL keyword!\n", tok);
        return VRC_FAILURE;
    }
    thee->etol = tf;
    thee->setetol = 1;
    return VRC_SUCCESS;
VERROR1:
    Vnm_print(2, "parseFE:  ran out of tokens!\n");
    return VRC_FAILURE;


}

VPRIVATE Vrc_Codes FEMparm_parseEKEY(FEMparm *thee,
                                     Vio *sock
                                     ) {

    char tok[VMAX_BUFSIZE];
    Vrc_Codes vrc = VRC_FAILURE;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (Vstring_strcasecmp(tok, "simp") == 0) {
        thee->ekey = FET_SIMP;
        thee->setekey = 1;
        vrc = VRC_SUCCESS;
```

```
    } else if (Vstring_strcasecmp(tok, "glob") == 0) {
        thee->ekey = FET_GLOB;
        thee->setekey = 1;
        vrc = VRC_SUCCESS;
    } else if (Vstring_strcasecmp(tok, "frac") == 0) {
        thee->ekey = FET_FRAC;
        thee->setekey = 1;
        vrc = VRC_SUCCESS;
    } else {
        Vnm_print(2, "parseFE:  undefined value (%s) for ekey!\n", tok);
        vrc = VRC_FAILURE;
    }

    return vrc;
VERROR1:
    Vnm_print(2, "parseFE:  ran out of tokens!\n");
    return VRC_FAILURE;

}

VPRIVATE Vrc_Codes FEMparm_parseAKEYPRE(FEMparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
    Vrc_Codes vrc = VRC_FAILURE;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (Vstring_strcasecmp(tok, "unif") == 0) {
        thee->akeyPRE = FRT_UNIF;
        thee->setakeyPRE = 1;
        vrc =  VRC_SUCCESS;
    } else if (Vstring_strcasecmp(tok, "geom") == 0) {
        thee->akeyPRE = FRT_GEOM;
        thee->setakeyPRE = 1;
        vrc =  VRC_SUCCESS;
    } else {
        Vnm_print(2, "parseFE:  undefined value (%s) for akeyPRE!\n", tok);
        vrc =  VRC_FAILURE;
    }

    return vrc;

VERROR1:
    Vnm_print(2, "parseFE:  ran out of tokens!\n");
    return VRC_FAILURE;

}

VPRIVATE Vrc_Codes FEMparm_parseAKEYSOLVE(FEMparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
    Vrc_Codes vrc = VRC_FAILURE;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (Vstring_strcasecmp(tok, "resi") == 0) {
        thee->akeySOLVE = FRT_RESI;
        thee->setakeySOLVE = 1;
        vrc =  VRC_SUCCESS;
    } else if (Vstring_strcasecmp(tok, "dual") == 0) {
        thee->akeySOLVE = FRT_DUAL;
        thee->setakeySOLVE = 1;
        vrc =  VRC_SUCCESS;
    } else if (Vstring_strcasecmp(tok, "loca") == 0) {
        thee->akeySOLVE = FRT_LOCA;
        thee->setakeySOLVE = 1;
        vrc =  VRC_SUCCESS;
    } else {
        Vnm_print(2, "parseFE:  undefined value (%s) for akeyPRE!\n", tok);
        vrc =  VRC_FAILURE;
    }

    return vrc;
VERROR1:
    Vnm_print(2, "parseFE:  ran out of tokens!\n");
    return VRC_SUCCESS;

}

VPRIVATE Vrc_Codes FEMparm_parseTARGETNUM(FEMparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
    int ti;
```

```c

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%d", &ti) == 0) {
        Vnm_print(2, "parseFE:  read non-int (%s) for targetNum!\n", tok);
        return VRC_FAILURE;
    }
    thee->targetNum = ti;
    thee->settargetNum = 1;
    return VRC_SUCCESS;
VERROR1:
    Vnm_print(2, "parseFE:  ran out of tokens!\n");
    return VRC_FAILURE;

}

VPRIVATE Vrc_Codes FEMparm_parseTARGETRES(FEMparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "parseFE:  read non-double (%s) for targetNum!\n",
          tok);
        return VRC_FAILURE;
    }
    thee->targetRes = tf;
    thee->settargetRes = 1;
    return VRC_SUCCESS;
VERROR1:
    Vnm_print(2, "parseFE:  ran out of tokens!\n");
    return VRC_FAILURE;

}

VPRIVATE Vrc_Codes FEMparm_parseMAXSOLVE(FEMparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
    int ti;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%d", &ti) == 0) {
        Vnm_print(2, "parseFE:  read non-int (%s) for maxsolve!\n", tok);
        return VRC_FAILURE;
    }
    thee->maxsolve = ti;
    thee->setmaxsolve = 1;
    return VRC_SUCCESS;
VERROR1:
    Vnm_print(2, "parseFE:  ran out of tokens!\n");
    return VRC_FAILURE;

}

VPRIVATE Vrc_Codes FEMparm_parseMAXVERT(FEMparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
    int ti;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%d", &ti) == 0) {
        Vnm_print(2, "parseFE:  read non-int (%s) for maxvert!\n", tok);
        return VRC_FAILURE;
    }
    thee->maxvert = ti;
    thee->setmaxvert = 1;
    return VRC_SUCCESS;

VERROR1:
    Vnm_print(2, "parseFE:  ran out of tokens!\n");
    return VRC_FAILURE;

}

VPRIVATE Vrc_Codes FEMparm_parseUSEMESH(FEMparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    int ti;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
 if (sscanf(tok, "%d", &ti) == 0) {
        Vnm_print(2, "parseFE:  read non-int (%s) for usemesh!\n", tok);
```

```
        return VRC_FAILURE;
    }
 thee->useMesh = 1;
 thee->meshID = ti;

 return VRC_SUCCESS;

VERROR1:
 Vnm_print(2, "parsePBE:  ran out of tokens!\n");
 return VRC_FAILURE;
}


VPUBLIC Vrc_Codes FEMparm_parseToken(FEMparm *thee,
char tok[VMAX_BUFSIZE],
  Vio *sock) {

    //int i, ti; // gcc says unused
    //double tf; // gcc says unused

    if (thee == VNULL) {
        Vnm_print(2, "parseFE:  got NULL thee!\n");
        return VRC_FAILURE;
    }

    if (sock == VNULL) {
        Vnm_print(2, "parseFE:  got NULL socket!\n");
        return VRC_FAILURE;
    }

    if (Vstring_strcasecmp(tok, "domainLength") == 0) {
        return FEMparm_parseDOMAINLENGTH(thee, sock);
    } else if (Vstring_strcasecmp(tok, "etol") == 0) {
        return FEMparm_parseETOL(thee, sock);
    } else if (Vstring_strcasecmp(tok, "ekey") == 0) {
        return FEMparm_parseEKEY(thee, sock);
    } else if (Vstring_strcasecmp(tok, "akeyPRE") == 0) {
        return FEMparm_parseAKEYPRE(thee, sock);
    } else if (Vstring_strcasecmp(tok, "akeySOLVE") == 0) {
        return FEMparm_parseAKEYSOLVE(thee, sock);
    } else if (Vstring_strcasecmp(tok, "targetNum") == 0) {
        return FEMparm_parseTARGETNUM(thee, sock);
    } else if (Vstring_strcasecmp(tok, "targetRes") == 0) {
        return FEMparm_parseTARGETRES(thee, sock);
    } else if (Vstring_strcasecmp(tok, "maxsolve") == 0) {
        return FEMparm_parseMAXSOLVE(thee, sock);
    } else if (Vstring_strcasecmp(tok, "maxvert") == 0) {
        return FEMparm_parseMAXVERT(thee, sock);
    } else if (Vstring_strcasecmp(tok, "usemesh") == 0) {
        return FEMparm_parseUSEMESH(thee, sock);
    }

    return VRC_WARNING;

}
```

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 1774 2012-07-19 18:46:14Z sobolevnrm

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
```

```
VEMBED(rcsid="$Id: mgparm.c 1774 2012-07-19 18:46:14Z sobolevnrm $")

#if !defined(VINLINE_MGPARM)

#endif /* if !defined(VINLINE_MGPARM) */

VPUBLIC void MGparm_setCenterX(MGparm *thee, double x) {
    VASSERT(thee != VNULL);
    thee->center[0] = x;
}
VPUBLIC void MGparm_setCenterY(MGparm *thee, double y) {
    VASSERT(thee != VNULL);
    thee->center[1] = y;
}
VPUBLIC void MGparm_setCenterZ(MGparm *thee, double z) {
    VASSERT(thee != VNULL);
    thee->center[2] = z;
}
VPUBLIC double MGparm_getCenterX(MGparm *thee) {
    VASSERT(thee != VNULL);
    return thee->center[0];
}
VPUBLIC double MGparm_getCenterY(MGparm *thee) {
    VASSERT(thee != VNULL);
    return thee->center[1];
}
VPUBLIC double MGparm_getCenterZ(MGparm *thee) {
    VASSERT(thee != VNULL);
    return thee->center[2];
}
VPUBLIC int MGparm_getNx(MGparm *thee) {
    VASSERT(thee != VNULL);
    return thee->dime[0];
}
VPUBLIC int MGparm_getNy(MGparm *thee) {
    VASSERT(thee != VNULL);
    return thee->dime[1];
}
VPUBLIC int MGparm_getNz(MGparm *thee) {
    VASSERT(thee != VNULL);
    return thee->dime[2];
}
VPUBLIC double MGparm_getHx(MGparm *thee) {
    VASSERT(thee != VNULL);
    return thee->grid[0];
}
VPUBLIC double MGparm_getHy(MGparm *thee) {
    VASSERT(thee != VNULL);
    return thee->grid[1];
}
VPUBLIC double MGparm_getHz(MGparm *thee) {
    VASSERT(thee != VNULL);
    return thee->grid[2];
}

VPUBLIC MGparm* MGparm_ctor(MGparm_CalcType
    type) {

    /* Set up the structure */
    MGparm *thee = VNULL;
    thee = (MGparm*)Vmem_malloc(VNULL, 1, sizeof(MGparm));
    VASSERT( thee != VNULL);
    VASSERT( MGparm_ctor2(thee, type) == VRC_SUCCESS );

    return thee;
}

VPUBLIC Vrc_Codes MGparm_ctor2(MGparm *thee, MGparm_CalcType
    type) {

    int i;

    if (thee == VNULL) return VRC_FAILURE;

    for (i=0; i<3; i++) {
        thee->dime[i] = -1;
        thee->pdime[i] = 1;
    }

    thee->parsed = 0;
    thee->type = type;
```

```
    /* *** GENERIC PARAMETERS *** */
    thee->setdime = 0;
    thee->setchgm = 0;

    /* *** TYPE 0 PARAMETERS *** */
    thee->nlev = VMGNLEV;
    thee->setnlev = 1;
    thee->etol = 1.0e-6;
    thee->setetol = 0;
    thee->setgrid = 0;
    thee->setglen = 0;
    thee->setgcent = 0;

    /* *** TYPE 1 & 2 PARAMETERS *** */
    thee->setcglen = 0;
    thee->setfglen = 0;
    thee->setcgcent = 0;
    thee->setfgcent = 0;

    /* *** TYPE 2 PARAMETERS *** */
    thee->setpdime = 0;
    thee->setrank = 0;
    thee->setsize = 0;
    thee->setofrac = 0;
    for (i=0; i<6; i++) thee->partDisjOwnSide[i] = 0;
 thee->setasync = 0;

    /* *** Default parameters for TINKER *** */
    thee->chgs = VCM_CHARGE;

 thee->useAqua = 0;
 thee->setUseAqua = 0;

    return VRC_SUCCESS;
}

VPUBLIC void MGparm_dtor(MGparm **thee) {
    if ((*thee) != VNULL) {
        MGparm_dtor2(*thee);
        Vmem_free(VNULL, 1, sizeof(MGparm), (void **)thee);
        (*thee) = VNULL;
    }
}

VPUBLIC void MGparm_dtor2(MGparm *thee) { ; }

VPUBLIC Vrc_Codes MGparm_check(MGparm *thee) {

    Vrc_Codes rc;
 int i, tdime[3], ti, tnlev[3], nlev;

    rc = VRC_SUCCESS;

 Vnm_print(0, "MGparm_check:  checking MGparm object of type %d.\n",
     thee->type);

    /* Check to see if we were even filled... */
    if (!thee->parsed) {
        Vnm_print(2, "MGparm_check:  not filled!\n");
        return VRC_FAILURE;
    }

    /* Check generic settings */
    if (!thee->setdime) {
        Vnm_print(2, "MGparm_check:  DIME not set!\n");
        rc = VRC_FAILURE;
    }
    if (!thee->setchgm) {
        Vnm_print(2, "MGparm_check: CHGM not set!\n");
        return VRC_FAILURE;
    }


    /* Check sequential manual & dummy settings */
    if ((thee->type == MCT_MANUAL) || (thee->type ==
MCT_DUMMY)) {
        if ((!thee->setgrid) && (!thee->setglen)) {
            Vnm_print(2, "MGparm_check:  Neither GRID nor GLEN set!\n");
            rc = VRC_FAILURE;
        }
```

```
        if ((thee->setgrid) && (thee->setglen)) {
            Vnm_print(2, "MGparm_check:  Both GRID and GLEN set!\n");
            rc = VRC_FAILURE;
        }
        if (!thee->setgcent) {
            Vnm_print(2, "MGparm_check:  GCENT not set!\n");
            rc = VRC_FAILURE;
        }
    }

    /* Check sequential and parallel automatic focusing settings */
    if ((thee->type == MCT_AUTO) || (thee->type == MCT_PARALLEL
    )) {
        if (!thee->setcglen) {
            Vnm_print(2, "MGparm_check:  CGLEN not set!\n");
            rc = VRC_FAILURE;
        }
        if (!thee->setfglen) {
            Vnm_print(2, "MGparm_check:  FGLEN not set!\n");
            rc = VRC_FAILURE;
        }
        if (!thee->setcgcent) {
            Vnm_print(2, "MGparm_check:  CGCENT not set!\n");
            rc = VRC_FAILURE;
        }
        if (!thee->setfgcent) {
            Vnm_print(2, "MGparm_check:  FGCENT not set!\n");
            rc = VRC_FAILURE;
        }
    }

    /* Check parallel automatic focusing settings */
    if (thee->type == MCT_PARALLEL) {
        if (!thee->setpdime) {
            Vnm_print(2, "MGparm_check:  PDIME not set!\n");
            rc = VRC_FAILURE;
        }
        if (!thee->setrank) {
            Vnm_print(2, "MGparm_check:  PROC_RANK not set!\n");
            rc = VRC_FAILURE;
        }
        if (!thee->setsize) {
            Vnm_print(2, "MGparm_check:  PROC_SIZE not set!\n");
            rc = VRC_FAILURE;
        }
        if (!thee->setofrac) {
            Vnm_print(2, "MGparm_check:  OFRAC not set!\n");
            rc = VRC_FAILURE;
        }
    }

    /* Perform a sanity check on nlev and dime, resetting values as necessary
    */
    if (rc == 1) {
 /* Calculate the actual number of grid points and nlev to satisfy the
  * formula:  n = c * 2^(l+1) + 1, where n is the number of grid points,
  * c is an integer, and l is the number of levels */
 if (thee->type != MCT_DUMMY) {
  for (i=0; i<3; i++) {
            /* See if the user picked a reasonable value, if not then fix it */
            ti = thee->dime[i] - 1;
            if (ti == VPOW(2, (thee->nlev+1))) {
                tnlev[i] = thee->nlev;
                tdime[i] = thee->dime[i];
            } else {
                tdime[i] = thee->dime[i];
                ti = tdime[i] - 1;
                tnlev[i] = 0;
                /* Find the maximum number of times this dimension can be
                 * divided by two */
                while (VEVEN(ti)) {
                    (tnlev[i])++;
                    ti = (int)ceil(0.5*ti);
                }
                (tnlev[i])--;
                /* We'd like to have at least VMGNLEV levels in the multigrid
                 * hierarchy.  This means that the dimension needs to be
                 * c*2^VMGNLEV + 1, where c is an integer. */
                if ((tdime[i] > 65) && (tnlev[i] < VMGNLEV)) {
                    Vnm_print(2, "NOsh:  Bad dime[%d]  = %d (%d nlev)!\n",
                      i, tdime[i], tnlev[i]);
```

```
                    ti = (int)(tdime[i]/VPOW(2.,(VMGNLEV+1)));
                    if (ti < 1) ti = 1;
                    tdime[i] = ti*(int)(VPOW(2.,(VMGNLEV+1))) + 1;
                    tnlev[i] = 4;
                    Vnm_print(2, "NOsh:  Reset dime[%d] to %d and (nlev = %d).\n", i, tdime[i], VMGNLEV);
                }
            }
        }
 } else { /* We are a dummy calculation, but we still need positive numbers of points */
  for (i=0; i<3; i++) {
      tnlev[i] = thee->nlev;
   tdime[i] = thee->dime[i];
    if (thee->dime[i] <= 0) {
     Vnm_print(2, "NOsh:  Resetting dime[%d] from %d to 3.\n", i, thee->dime[i]);
     thee->dime[i] = 3;
   }
  }
 }

 /* The actual number of levels we'll be using is the smallest number of
         * possible levels in any dimensions */
        nlev = VMIN2(tnlev[0], tnlev[1]);
        nlev = VMIN2(nlev, tnlev[2]);
        /* Set the number of levels and dimensions */
        Vnm_print(0, "NOsh:  nlev = %d, dime = (%d, %d, %d)\n", nlev, tdime[0],
          tdime[1], tdime[2]);
        thee->nlev = nlev;
  if (thee->nlev <= 0) {
   Vnm_print(2, "MGparm_check:  illegal nlev (%d); check your grid dimensions!\n", thee->nlev);
   rc = VRC_FAILURE;
  }
  if (thee->nlev < 2) {
   Vnm_print(2, "MGparm_check:  you're using a very small nlev (%d) and therefore\n", thee->nlev);
   Vnm_print(2, "MGparm_check:  will not get the optimal performance of the multigrid\n");
   Vnm_print(2, "MGparm_check:  algorithm.  Please check your grid dimensions.\n");
  }
        for (i=0; i<3; i++) thee->dime[i] = tdime[i];
    }

 if (!thee->setUseAqua) thee->useAqua = 0;

    return rc;
}

VPUBLIC void MGparm_copy(MGparm *thee, MGparm *parm) {

    int i;

    VASSERT(thee != VNULL);
    VASSERT(parm != VNULL);


    thee->type = parm->type;
    thee->parsed = parm->parsed;

    /* *** GENERIC PARAMETERS *** */
    for (i=0; i<3; i++) thee->dime[i] = parm->dime[i];
    thee->setdime = parm->setdime;
    thee->chgm = parm->chgm;
    thee->setchgm = parm->setchgm;
    thee->chgs = parm->chgs;

    /* *** TYPE 0 PARMS *** */
    thee->nlev = parm->nlev;
    thee->setnlev = parm->setnlev;
    thee->etol = parm->etol;
    thee->setetol = parm->setetol;
    for (i=0; i<3; i++) thee->grid[i] = parm->grid[i];
    thee->setgrid = parm->setgrid;
    for (i=0; i<3; i++) thee->glen[i] = parm->glen[i];
    thee->setglen = parm->setglen;
    thee->cmeth = parm->cmeth;
    for (i=0; i<3; i++) thee->center[i] = parm->center[i];
    thee->setgcent = parm->setgcent;
```

```
    thee->centmol = parm->centmol;

    /* *** TYPE 1 & 2 PARMS *** */
    for (i=0; i<3; i++) thee->cglen[i] = parm->cglen[i];
    thee->setcglen = parm->setcglen;
    for (i=0; i<3; i++) thee->fglen[i] = parm->fglen[i];
    thee->setfglen = parm->setfglen;
    thee->ccmeth = parm->ccmeth;
    for (i=0; i<3; i++) thee->ccenter[i] = parm->ccenter[i];
    thee->setcgcent = parm->setcgcent;
    thee->ccentmol = parm->ccentmol;
    thee->fcmeth = parm->fcmeth;
    for (i=0; i<3; i++) thee->fcenter[i] = parm->fcenter[i];
    thee->setfgcent = parm->setfgcent;
    thee->fcentmol = parm->fcentmol;

    /* *** TYPE 2 PARMS *** */
    for (i=0; i<3; i++)
      thee->partDisjCenter[i] = parm->partDisjCenter
[i];
    for (i=0; i<3; i++)
      thee->partDisjLength[i] = parm->partDisjLength
[i];
    for (i=0; i<6; i++)
      thee->partDisjOwnSide[i] = parm->partDisjOwnSide
[i];
    for (i=0; i<3; i++) thee->pdime[i] = parm->pdime[i];
    thee->setpdime = parm->setpdime;
    thee->proc_rank = parm->proc_rank;
    thee->setrank = parm->setrank;
    thee->proc_size = parm->proc_size;
    thee->setsize = parm->setsize;
    thee->ofrac = parm->ofrac;
    thee->setofrac = parm->setofrac;
 thee->setasync = parm->setasync;
 thee->async = parm->async;

 thee->nonlintype = parm->nonlintype;
 thee->setnonlintype = parm->setnonlintype;

 thee->method = parm->method;
 thee->method = parm->method;

 thee->useAqua = parm->useAqua;
 thee->setUseAqua = parm->setUseAqua;
}

VPRIVATE Vrc_Codes MGparm_parseDIME(MGparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
    int ti;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%d", &ti) == 0){
        Vnm_print(2, "parseMG:  Read non-integer (%s) while parsing DIME \
keyword!\n", tok);
        return VRC_WARNING;
    } else thee->dime[0] = ti;
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%d", &ti) == 0) {
        Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing DIME \
keyword!\n", tok);
        return VRC_WARNING;
    } else thee->dime[1] = ti;
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%d", &ti) == 0) {
        Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing DIME \
keyword!\n", tok);
        return VRC_WARNING;
    } else thee->dime[2] = ti;
    thee->setdime = 1;
    return VRC_SUCCESS;

    VERROR1:
        Vnm_print(2, "parseMG:  ran out of tokens!\n");
        return VRC_WARNING;
}

VPRIVATE Vrc_Codes MGparm_parseCHGM(MGparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
```

```
    Vchrg_Meth ti;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%d", (int*)(&ti)) == 1) {
        thee->chgm = ti;
        thee->setchgm = 1;
        Vnm_print(2, "NOsh:  Warning -- parsed deprecated statment \"chgm %d\".
\n", ti);
        Vnm_print(2, "NOsh:  Please use \"chgm ");
        switch (thee->chgm) {
            case VCM_TRIL:
                Vnm_print(2, "spl0");
                break;
            case VCM_BSPL2:
                Vnm_print(2, "spl2");
                break;
            case VCM_BSPL4:
                Vnm_print(2, "spl4");
                break;
            default:
                Vnm_print(2, "UNKNOWN");
                break;
        }
        Vnm_print(2, "\" instead!\n");
        return VRC_SUCCESS;
    } else if (Vstring_strcasecmp(tok, "spl0") == 0) {
        thee->chgm = VCM_TRIL;
        thee->setchgm = 1;
        return VRC_SUCCESS;
    } else if (Vstring_strcasecmp(tok, "spl2") == 0) {
        thee->chgm = VCM_BSPL2;
        thee->setchgm = 1;
        return VRC_SUCCESS;
    } else if (Vstring_strcasecmp(tok, "spl4") == 0) {
        thee->chgm = VCM_BSPL4;
        thee->setchgm = 1;
        return VRC_SUCCESS;
    } else {
        Vnm_print(2, "NOsh:  Unrecognized parameter (%s) when parsing \
chgm!\n", tok);
        return VRC_WARNING;
    }
    return VRC_WARNING;

    VERROR1:
        Vnm_print(2, "parseMG:  ran out of tokens!\n");
        return VRC_WARNING;
}

VPRIVATE Vrc_Codes MGparm_parseNLEV(MGparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
    int ti;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%d", &ti) == 0) {
        Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing NLEV \
keyword!\n", tok);
        return VRC_WARNING;
    } else thee->nlev = ti;
    thee->setnlev = 1;
    return VRC_SUCCESS;

    VERROR1:
        Vnm_print(2, "parseMG:  ran out of tokens!\n");
        return VRC_WARNING;
}

VPRIVATE Vrc_Codes MGparm_parseETOL(MGparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing etol \
keyword!\n", tok);
        return VRC_WARNING;
    } else if (tf <= 0.0) {
        Vnm_print(2, "parseMG:  etol must be greater than 0!\n");
        return VRC_WARNING;
```

```
    } else thee->etol = tf;
    thee->setetol = 1;
    return VRC_SUCCESS;

    VERROR1:
        Vnm_print(2, "parseMG:  ran out of tokens!\n");
        return VRC_WARNING;
}


VPRIVATE Vrc_Codes MGparm_parseGRID(MGparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GRID \
keyword!\n", tok);
        return VRC_WARNING;
    } else thee->grid[0] = tf;
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GRID \
keyword!\n", tok);
        return VRC_WARNING;
    } else thee->grid[1] = tf;
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GRID \
keyword!\n", tok);
        return VRC_WARNING;
    } else thee->grid[2] = tf;
    thee->setgrid = 1;
    return VRC_SUCCESS;

    VERROR1:
        Vnm_print(2, "parseMG:  ran out of tokens!\n");
        return VRC_WARNING;
}

VPRIVATE Vrc_Codes MGparm_parseGLEN(MGparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GLEN \
keyword!\n", tok);
        return VRC_WARNING;
    } else thee->glen[0] = tf;
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GLEN \
keyword!\n", tok);
        return VRC_WARNING;
    } else thee->glen[1] = tf;
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GLEN \
keyword!\n", tok);
        return VRC_WARNING;
    } else thee->glen[2] = tf;
    thee->setglen = 1;
    return VRC_SUCCESS;

    VERROR1:
        Vnm_print(2, "parseMG:  ran out of tokens!\n");
        return VRC_WARNING;
}

VPRIVATE Vrc_Codes MGparm_parseGAMMA(MGparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
 Vnm_print(2, "parseMG:  GAMMA keyword deprecated!\n");
 Vnm_print(2, "parseMG:  If you are using PyMOL or VMD and still seeing this
 message,\n");
 Vnm_print(2, "parseMG:  please contact the developers of those programs
```

```
     regarding this message.\n");
 return VRC_SUCCESS;

VERROR1:
 Vnm_print(2, "parseMG:  ran out of tokens!\n");
 return VRC_WARNING;
}

VPRIVATE Vrc_Codes MGparm_parseGCENT(MGparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
    double tf;
    int ti;

    /* If the next token isn't a float, it probably means we want to
     * center on a molecule */
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        if (Vstring_strcasecmp(tok, "mol") == 0) {
            VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
            if (sscanf(tok, "%d", &ti) == 0) {
                Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
GCENT MOL keyword!\n", tok);
                return VRC_WARNING;
            } else {
                thee->cmeth = MCM_MOLECULE;
    /* Subtract 1 here to convert user numbering (1, 2, 3, ...) into
    array index */
                thee->centmol = ti - 1;
            }
        } else {
            Vnm_print(2, "NOsh:  Unexpected keyword (%s) while parsing \
GCENT!\n", tok);
            return VRC_WARNING;
        }
    } else {
        thee->center[0] = tf;
        VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
        if (sscanf(tok, "%lf", &tf) == 0) {
            Vnm_print(2, "NOsh:  Read non-float (%s) while parsing \
GCENT keyword!\n", tok);
            return VRC_WARNING;
        }
        thee->center[1] = tf;
        VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
        if (sscanf(tok, "%lf", &tf) == 0) {
            Vnm_print(2, "NOsh:  Read non-float (%s) while parsing \
GCENT keyword!\n", tok);
            return VRC_WARNING;
        }
        thee->center[2] = tf;
    }
    thee->setgcent = 1;
    return VRC_SUCCESS;

    VERROR1:
        Vnm_print(2, "parseMG:  ran out of tokens!\n");
        return VRC_WARNING;
}

VPRIVATE Vrc_Codes MGparm_parseCGLEN(MGparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing CGLEN \
keyword!\n", tok);
        return VRC_WARNING;
    } else thee->cglen[0] = tf;
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing CGLEN \
keyword!\n", tok);
        return VRC_WARNING;
    } else thee->cglen[1] = tf;
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing CGLEN \
keyword!\n", tok);
```

```
        return VRC_WARNING;
    } else thee->cglen[2] = tf;
    thee->setcglen = 1;
    return VRC_SUCCESS;

    VERROR1:
        Vnm_print(2, "parseMG:  ran out of tokens!\n");
        return VRC_WARNING;
}

VPRIVATE Vrc_Codes MGparm_parseFGLEN(MGparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing FGLEN \
keyword!\n", tok);
        return VRC_WARNING;
    } else thee->fglen[0] = tf;
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing FGLEN \
keyword!\n", tok);
        return VRC_WARNING;
    } else thee->fglen[1] = tf;
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing FGLEN \
keyword!\n", tok);
        return VRC_WARNING;
    } else thee->fglen[2] = tf;
    thee->setfglen = 1;
    return VRC_SUCCESS;

    VERROR1:
        Vnm_print(2, "parseMG:  ran out of tokens!\n");
        return VRC_WARNING;
}

VPRIVATE Vrc_Codes MGparm_parseCGCENT(MGparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
    double tf;
    int ti;

    /* If the next token isn't a float, it probably means we want to
     * center on a molecule */
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        if (Vstring_strcasecmp(tok, "mol") == 0) {
            VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
            if (sscanf(tok, "%d", &ti) == 0) {
                Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
CGCENT MOL keyword!\n", tok);
                return VRC_WARNING;
            } else {
    thee->ccmeth = MCM_MOLECULE;
    /* Subtract 1 here to convert user numbering (1, 2, 3, ...) into
    array index */
                thee->ccentmol = ti - 1;
   }
        } else {
            Vnm_print(2, "NOsh:  Unexpected keyword (%s) while parsing \
CGCENT!\n", tok);
            return VRC_WARNING;
            }
    } else {
        thee->ccenter[0] = tf;
        VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
        if (sscanf(tok, "%lf", &tf) == 0) {
            Vnm_print(2, "NOsh:  Read non-float (%s) while parsing \
CGCENT keyword!\n", tok);
            return VRC_WARNING;
        }
        thee->ccenter[1] = tf;
        VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
        if (sscanf(tok, "%lf", &tf) == 0) {
            Vnm_print(2, "NOsh:  Read non-float (%s) while parsing \
CGCENT keyword!\n", tok);
```

```
            return VRC_WARNING;
        }
        thee->ccenter[2] = tf;
    }
    thee->setcgcent = 1;
    return VRC_SUCCESS;

    VERROR1:
        Vnm_print(2, "parseMG:  ran out of tokens!\n");
        return VRC_WARNING;
}

VPRIVATE Vrc_Codes MGparm_parseFGCENT(MGparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
    double tf;
    int ti;

    /* If the next token isn't a float, it probably means we want to
     * center on a molecule */
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        if (Vstring_strcasecmp(tok, "mol") == 0) {
            VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
            if (sscanf(tok, "%d", &ti) == 0) {
                 Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
FGCENT MOL keyword!\n", tok);
                 return VRC_WARNING;
            } else {
                thee->fcmeth = MCM_MOLECULE;
    /* Subtract 1 here to convert user numbering (1, 2, 3, ...) into
    array index */
                thee->fcentmol = ti - 1;
            }
        } else {
            Vnm_print(2, "NOsh:  Unexpected keyword (%s) while parsing \
FGCENT!\n", tok);
            return VRC_WARNING;
        }
    } else {
        thee->fcenter[0] = tf;
        VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
        if (sscanf(tok, "%lf", &tf) == 0) {
            Vnm_print(2, "NOsh:  Read non-float (%s) while parsing \
FGCENT keyword!\n", tok);
            return VRC_WARNING;
        }
        thee->fcenter[1] = tf;
        VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
        if (sscanf(tok, "%lf", &tf) == 0) {
            Vnm_print(2, "NOsh:  Read non-float (%s) while parsing \
FGCENT keyword!\n", tok);
            return VRC_WARNING;
        }
        thee->fcenter[2] = tf;
    }
    thee->setfgcent = 1;
    return VRC_SUCCESS;

    VERROR1:
        Vnm_print(2, "parseMG:  ran out of tokens!\n");
        return VRC_WARNING;
}

VPRIVATE Vrc_Codes MGparm_parsePDIME(MGparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
    int ti;

    /* Read the number of grid points */
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%d", &ti) == 0) {
        Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing PDIME \
keyword!\n", tok);
        return VRC_WARNING;
    } else {
        thee->pdime[0] = ti;
    }
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%d", &ti) == 0) {
        Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing PDIME \
```

```
keyword!\n", tok);
        return VRC_WARNING;
    } else {
        thee->pdime[1] = ti;
    }
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%d", &ti) == 0) {
        Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing PDIME \
keyword!\n", tok);
        return VRC_WARNING;
    } else {
        thee->pdime[2] = ti;
    }
    thee->setpdime = 1;
    return VRC_SUCCESS;

    VERROR1:
        Vnm_print(2, "parseMG:  ran out of tokens!\n");
        return VRC_WARNING;
}

VPRIVATE Vrc_Codes MGparm_parseOFRAC(MGparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-int (%s) while parsing OFRAC \
keyword!\n", tok);
        return VRC_WARNING;
    }
    thee->ofrac = tf;
    thee->setofrac = 1;
    return VRC_SUCCESS;

    VERROR1:
        Vnm_print(2, "parseMG:  ran out of tokens!\n");
        return VRC_WARNING;
}

VPRIVATE Vrc_Codes MGparm_parseASYNC(MGparm *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
    int ti;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%i", &ti) == 0) {
        Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing ASYNC \
keyword!\n", tok);
        return VRC_WARNING;
    }
    thee->async = ti;
    thee->setasync = 1;
    return VRC_SUCCESS;

    VERROR1:
        Vnm_print(2, "parseMG:  ran out of tokens!\n");
        return VRC_WARNING;
}

VPRIVATE Vrc_Codes MGparm_parseUSEAQUA(MGparm *thee, Vio *sock) {
    Vnm_print(0, "NOsh: parsed useaqua\n");
    thee->useAqua = 1;
    thee->setUseAqua = 1;
    return VRC_SUCCESS;
}

VPUBLIC Vrc_Codes MGparm_parseToken(MGparm *thee, char
tok[VMAX_BUFSIZE],
  Vio *sock) {

    if (thee == VNULL) {
        Vnm_print(2, "parseMG:  got NULL thee!\n");
        return VRC_WARNING;
    }
    if (sock == VNULL) {
        Vnm_print(2, "parseMG:  got NULL socket!\n");
        return VRC_WARNING;
    }

```

```
Vnm_print(0, "MGparm_parseToken:  trying %s...\n", tok);


   if (Vstring_strcasecmp(tok, "dime") == 0) {
       return MGparm_parseDIME(thee, sock);
   } else if (Vstring_strcasecmp(tok, "chgm") == 0) {
       return MGparm_parseCHGM(thee, sock);
   } else if (Vstring_strcasecmp(tok, "nlev") == 0) {
       Vnm_print(2, "Warning: The 'nlev' keyword is now deprecated!\n");
       return MGparm_parseNLEV(thee, sock);
   } else if (Vstring_strcasecmp(tok, "etol") == 0) {
       return MGparm_parseETOL(thee, sock);
   } else if (Vstring_strcasecmp(tok, "grid") == 0) {
       return MGparm_parseGRID(thee, sock);
   } else if (Vstring_strcasecmp(tok, "glen") == 0) {
       return MGparm_parseGLEN(thee, sock);
   } else if (Vstring_strcasecmp(tok, "gcent") == 0) {
       return MGparm_parseGCENT(thee, sock);
   } else if (Vstring_strcasecmp(tok, "cglen") == 0) {
       return MGparm_parseCGLEN(thee, sock);
   } else if (Vstring_strcasecmp(tok, "fglen") == 0) {
       return MGparm_parseFGLEN(thee, sock);
   } else if (Vstring_strcasecmp(tok, "cgcent") == 0) {
       return MGparm_parseCGCENT(thee, sock);
   } else if (Vstring_strcasecmp(tok, "fgcent") == 0) {
       return MGparm_parseFGCENT(thee, sock);
   } else if (Vstring_strcasecmp(tok, "pdime") == 0) {
       return MGparm_parsePDIME(thee, sock);
   } else if (Vstring_strcasecmp(tok, "ofrac") == 0) {
       return MGparm_parseOFRAC(thee, sock);
   } else if (Vstring_strcasecmp(tok, "async") == 0) {
       return MGparm_parseASYNC(thee, sock);
 } else if (Vstring_strcasecmp(tok, "gamma") == 0) {
       return MGparm_parseGAMMA(thee, sock);
 } else if (Vstring_strcasecmp(tok, "useaqua") == 0) {
       return MGparm_parseUSEAQUA(thee, sock);
   } else {
       Vnm_print(2, "parseMG:  Unrecognized keyword (%s)!\n", tok);
       return VRC_WARNING;
   }

   return VRC_FAILURE;

}
```

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)

  Returns format of specified dielectric map.
- VPUBLIC int NOsh_getKappafmt (NOsh *thee, int i)

  Returns format of specified kappa map.
- VPUBLIC int NOsh_getPotfmt (NOsh *thee, int i)

  Returns format of specified potential map.
- VPUBLIC int NOsh_getChargefmt (NOsh *thee, int i)

  Returns format of specified charge map.
- VPUBLIC NOsh_PrintType NOsh_printWhat (NOsh *thee, int iprint)

  Return an integer ID of the observable to print (.
- VPUBLIC int NOsh_printNarg (NOsh *thee, int iprint)

  Return number of arguments to PRINT statement (.
- VPUBLIC int NOsh_elec2calc (NOsh *thee, int icalc)

  Return the name of an elec statement.
- VPUBLIC int NOsh_apol2calc (NOsh *thee, int icalc)

  Return the name of an apol statement.
- VPUBLIC char * NOsh_elecname (NOsh *thee, int ielec)

  Return an integer mapping of an ELEC statement to a calculation ID (.
- VPUBLIC int NOsh_printOp (NOsh *thee, int iprint, int iarg)

  Return integer ID for specified operation (.
- VPUBLIC int NOsh_printCalc (NOsh *thee, int iprint, int iarg)

  Return calculation ID for specified PRINT statement (.
- VPUBLIC NOsh * NOsh_ctor (int rank, int size)

  Construct NOsh.
- VPUBLIC int NOsh_ctor2 (NOsh *thee, int rank, int size)

  FORTRAN stub to construct NOsh.
- VPUBLIC void NOsh_dtor (NOsh **thee)

  Object destructor.
- VPUBLIC void NOsh_dtor2 (NOsh *thee)

  FORTRAN stub for object destructor.
- VPUBLIC NOsh_calc * NOsh_calc_ctor (NOsh_CalcType calctype)

  Construct NOsh_calc.
- VPUBLIC void NOsh_calc_dtor (NOsh_calc **thee)

  Object destructor.
- VPUBLIC int NOsh_calc_copy (NOsh_calc *thee, NOsh_calc *source)

  Copy NOsh_calc object into thee.
- VPUBLIC int NOsh_parseInputFile (NOsh *thee, char *filename)

  Parse an input file only from a file.
- VPUBLIC int NOsh_parseInput (NOsh *thee, Vio *sock)

  Parse an input file from a socket.
- VPRIVATE int **NOsh_parseREAD_MOL** (NOsh *thee, Vio *sock)
- VPRIVATE int **NOsh_parseREAD_PARM** (NOsh *thee, Vio *sock)
- VPRIVATE int **NOsh_parseREAD_DIEL** (NOsh *thee, Vio *sock)
- VPRIVATE int **NOsh_parseREAD_KAPPA** (NOsh *thee, Vio *sock)
- VPRIVATE int **NOsh_parseREAD_POTENTIAL** (NOsh *thee, Vio *sock)
- VPRIVATE int **NOsh_parseREAD_CHARGE** (NOsh *thee, Vio *sock)

- VPRIVATE int **NOsh_parseREAD_MESH** (NOsh *thee, Vio *sock)
- VPUBLIC int NOsh_setupElecCalc (NOsh *thee, Valist *alist[NOSH_MAXMOL])

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

  Setup the series of non-polar calculations.

9.59.1 Detailed Description

Class NOsh methods.

Author

Nathan Baker

Version

Id:

nosh.c 1774 2012-07-19 18:46:14Z sobolevnrm

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

```
#include "apbscfg.h"
#include "apbs/nosh.h"
#include "apbs/vstring.h"
#include "apbs/mgparm.h"
#include "apbs/femparm.h"

VEMBED(rcsid="$Id: nosh.c 1774 2012-07-19 18:46:14Z sobolevnrm $")


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
        );

#if !defined(VINLINE_NOSH)

VPUBLIC char* NOsh_getMolpath(NOsh *thee, int imol) {
 VASSERT(thee != VNULL);
 VASSERT(imol < thee->nmol);
 return thee->molpath[imol];
}
VPUBLIC char* NOsh_getDielXpath(NOsh *thee, int imol) {
 VASSERT(thee != VNULL);
 VASSERT(imol < thee->nmol);
 return thee->dielXpath[imol];
}
VPUBLIC char* NOsh_getDielYpath(NOsh *thee, int imol) {
 VASSERT(thee != VNULL);
 VASSERT(imol < thee->nmol);
 return thee->dielYpath[imol];
}
VPUBLIC char* NOsh_getDielZpath(NOsh *thee, int imol) {
 VASSERT(thee != VNULL);
 VASSERT(imol < thee->nmol);
 return thee->dielZpath[imol];
}
VPUBLIC char* NOsh_getKappapath(NOsh *thee, int imol) {
 VASSERT(thee != VNULL);
 VASSERT(imol < thee->nmol);
 return thee->kappapath[imol];
}
VPUBLIC char* NOsh_getPotpath(NOsh *thee, int imol) {
 VASSERT(thee != VNULL);
 VASSERT(imol < thee->nmol);
 return thee->potpath[imol];
}
VPUBLIC char* NOsh_getChargepath(NOsh *thee, int imol) {
 VASSERT(thee != VNULL);
 VASSERT(imol < thee->nmol);
 return thee->chargepath[imol];
}
VPUBLIC NOsh_calc* NOsh_getCalc(NOsh *thee, int icalc)
 {
 VASSERT(thee != VNULL);
 VASSERT(icalc < thee->ncalc);
 return thee->calc[icalc];
}
VPUBLIC int NOsh_getDielfmt(NOsh *thee, int i) {
 VASSERT(thee != VNULL);
 VASSERT(i < thee->ndiel);
 return (thee->dielfmt[i]);
}
VPUBLIC int NOsh_getKappafmt(NOsh *thee, int i) {
 VASSERT(thee != VNULL);
 VASSERT(i < thee->nkappa);
 return (thee->kappafmt[i]);
}
VPUBLIC int NOsh_getPotfmt(NOsh *thee, int i) {
 VASSERT(thee != VNULL);
 VASSERT(i < thee->npot);
 return (thee->potfmt[i]);
}
VPUBLIC int NOsh_getChargefmt(NOsh *thee, int i) {
 VASSERT(thee != VNULL);
```

```
 VASSERT(i < thee->ncharge);
 return (thee->chargefmt[i]);
}


#endif /* if !defined(VINLINE_NOSH) */

VPUBLIC NOsh_PrintType NOsh_printWhat(NOsh *thee, int iprint) {
 VASSERT(thee != VNULL);
 VASSERT(iprint < thee->nprint);
 return thee->printwhat[iprint];
}

VPUBLIC int NOsh_printNarg(NOsh *thee, int iprint) {
 VASSERT(thee != VNULL);
 VASSERT(iprint < thee->nprint);
 return thee->printnarg[iprint];
}

VPUBLIC int NOsh_elec2calc(NOsh *thee, int icalc) {
 VASSERT(thee != VNULL);
 VASSERT(icalc < thee->ncalc);
 return thee->elec2calc[icalc];
}

VPUBLIC int NOsh_apol2calc(NOsh *thee, int icalc) {
 VASSERT(thee != VNULL);
 VASSERT(icalc < thee->ncalc);
 return thee->apol2calc[icalc];
}

VPUBLIC char* NOsh_elecname(NOsh *thee, int ielec) {
 VASSERT(thee != VNULL);
 VASSERT(ielec < thee->nelec + 1);
 return thee->elecname[ielec];
}

VPUBLIC int NOsh_printOp(NOsh *thee, int iprint, int iarg) {
 VASSERT(thee != VNULL);
 VASSERT(iprint < thee->nprint);
 VASSERT(iarg < thee->printnarg[iprint]);
 return thee->printop[iprint][iarg];
}

VPUBLIC int NOsh_printCalc(NOsh *thee, int iprint, int iarg) {
 VASSERT(thee != VNULL);
 VASSERT(iprint < thee->nprint);
 VASSERT(iarg < thee->printnarg[iprint]);
 return thee->printcalc[iprint][iarg];
}

VPUBLIC NOsh* NOsh_ctor(int rank, int size) {

 /* Set up the structure */
 NOsh *thee = VNULL;
 thee = (NOsh*)Vmem_malloc(VNULL, 1, sizeof(NOsh) );
 VASSERT( thee != VNULL);
 VASSERT( NOsh_ctor2(thee, rank, size) );

 return thee;
}

VPUBLIC int NOsh_ctor2(NOsh *thee, int rank, int size) {

 int i;

 if (thee == VNULL) return 0;

 thee->proc_rank = rank;
 thee->proc_size = size;

 thee->ispara = 0;
    thee->parsed = 0;

    thee->nmol = 0;
    thee->gotparm = 0;
    thee->ncharge = 0;
    thee->ndiel = 0;
    thee->nkappa = 0;
```

```
 thee->npot = 0;
    thee->nprint = 0;

    for (i=0; i<NOSH_MAXCALC; i++) {
  thee->calc[i] = VNULL;
  thee->elec[i] = VNULL;
  thee->apol[i] = VNULL;
 }
 for (i=0; i<NOSH_MAXMOL; i++) {
  thee->alist[i] = VNULL;
 }
 thee->ncalc = 0;
 thee->nelec = 0;
 thee->napol = 0;

    return 1;
}

VPUBLIC void NOsh_dtor(NOsh **thee) {
    if ((*thee) != VNULL) {
        NOsh_dtor2(*thee);
        Vmem_free(VNULL, 1, sizeof(NOsh), (void **)thee);
        (*thee) = VNULL;
    }
}

VPUBLIC void NOsh_dtor2(NOsh *thee) {

 int i;

 if (thee != VNULL) {
  for (i=0; i<(thee->ncalc); i++) NOsh_calc_dtor(&(thee->calc[i]));
  for (i=0; i<(thee->nelec); i++) NOsh_calc_dtor(&(thee->elec[i]));
  for (i=0; i<(thee->napol); i++) NOsh_calc_dtor(&(thee->apol[i]));
    }

}

VPUBLIC NOsh_calc* NOsh_calc_ctor(
          NOsh_CalcType calctype
          ) {
 NOsh_calc *thee;
 thee = (NOsh_calc *)Vmem_malloc(VNULL, 1, sizeof(NOsh_calc));
 thee->calctype = calctype;
 switch (calctype) {
  case NCT_MG:
   thee->mgparm = MGparm_ctor(MCT_NONE);
   thee->femparm = VNULL;
   thee->apolparm = VNULL;
   break;
  case NCT_FEM:
   thee->mgparm = VNULL;
   thee->femparm = FEMparm_ctor(FCT_NONE);
   thee->apolparm = VNULL;
   break;
  case NCT_APOL:
   thee->mgparm = VNULL;
   thee->femparm = VNULL;
   thee->apolparm = APOLparm_ctor();
   break;
  default:
   Vnm_print(2, "NOsh_calc_ctor:  unknown calculation type (%d)!\n",
       calctype);
   VASSERT(0);
 }
 thee->pbeparm = PBEparm_ctor();

 return thee;
}

VPUBLIC void NOsh_calc_dtor(
       NOsh_calc **thee
       ) {

 NOsh_calc *calc = VNULL;
 calc = *thee;
 if (calc == VNULL) return;
```

```
00353
00354  switch (calc->calctype) {
00355   case NCT_MG:
00356    MGparm_dtor(&(calc->mgparm));
00357    break;
00358   case NCT_FEM:
00359    FEMparm_dtor(&(calc->femparm));
00360    break;
00361   case NCT_APOL:
00362    APOLparm_dtor(&(calc->apolparm));
00363    break;
00364   default:
00365    Vnm_print(2, "NOsh_calc_ctor:  unknown calculation type (%d)!\n",
00366        calc->calctype);
00367    VASSERT(0);
00368  }
00369  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");
```

```
  return 0;
 }

 if (Vio_accept(sock, 0) < 0) {
  Vnm_print(2, "NOsh_parseInput:  Problem reading from socket!\n");
  return 0;
 }

 /* Set up the whitespace and comment character definitions */
 Vio_setWhiteChars(sock, MCwhiteChars);
 Vio_setCommChars(sock, MCcommChars);

 /* We parse the file until we run out of tokens */
 Vnm_print(0, "NOsh_parseInput:  Starting file parsing...\n");
 while (Vio_scanf(sock, "%s", tok) == 1) {
  /* At the highest level, we look for keywords that indicate functions like:

  read => Read in a molecule file
  elec => Do an electrostatics calculation
  print => Print some results
  apolar => do a non-polar calculation
  quit => Quit

  These cause the code to go to a lower-level parser routine which
  handles keywords specific to the particular function.  Each
  lower-level parser routine then returns when it hits the "end"
  keyword.  Due to this simple layout, no nesting of these "function"
  sections is allowed.
  */
  if (Vstring_strcasecmp(tok, "read") == 0) {
   Vnm_print(0, "NOsh: Parsing READ section\n");
   if (!NOsh_parseREAD(thee, sock)) return 0;
   Vnm_print(0, "NOsh: Done parsing READ section \
(nmol=%d, ndiel=%d, nkappa=%d, ncharge=%d, npot=%d)\n", thee->nmol, thee->
ndiel,
       thee->nkappa, thee->ncharge,thee->npot);
  } else if (Vstring_strcasecmp(tok, "print") == 0) {
   Vnm_print(0, "NOsh: Parsing PRINT section\n");
   if (!NOsh_parsePRINT(thee, sock)) return 0;
   Vnm_print(0, "NOsh: Done parsing PRINT section\n");
  } else if (Vstring_strcasecmp(tok, "elec") == 0) {
   Vnm_print(0, "NOsh: Parsing ELEC section\n");
   if (!NOsh_parseELEC(thee, sock)) return 0;
   Vnm_print(0, "NOsh: Done parsing ELEC section (nelec = %d)\n",
       thee->nelec);
  } else if (Vstring_strcasecmp(tok, "apolar") == 0) {
   Vnm_print(0, "NOsh: Parsing APOLAR section\n");
   if (!NOsh_parseAPOLAR(thee, sock)) return 0;
   Vnm_print(0, "NOsh: Done parsing APOLAR section (nelec = %d)\n",
       thee->nelec);
  } else if (Vstring_strcasecmp(tok, "quit") == 0) {
   Vnm_print(0, "NOsh: Done parsing file (got QUIT)\n");
   break;
  } else {
   Vnm_print(2, "NOsh_parseInput: Ignoring undefined keyword %s!\n", tok);
  }
 }

 thee->parsed = 1;
 return 1;

}

VPRIVATE int NOsh_parseREAD_MOL(NOsh *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
    NOsh_MolFormat molfmt;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (Vstring_strcasecmp(tok, "pqr") == 0) {
        molfmt = NMF_PQR;
        VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
        if (tok[0]=='"') {
            strcpy(strnew, "");
            while (tok[strlen(tok)-1] != '"') {
                strcat(str, tok);
                strcat(str, " ");
                VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
   }
            strcat(str, tok);
            strncpy(strnew, str+1, strlen(str)-2);
```

```
            strcpy(tok, strnew);
        }
        Vnm_print(0, "NOsh: Storing molecule %d path %s\n",
      thee->nmol, tok);
        thee->molfmt[thee->nmol] = molfmt;
        strncpy(thee->molpath[thee->nmol], tok, VMAX_ARGLEN);
        (thee->nmol)++;
    } else if (Vstring_strcasecmp(tok, "pdb") == 0) {
        molfmt = NMF_PDB;
        VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
        if (tok[0]=='"') {
            strcpy(strnew, "");
            while (tok[strlen(tok)-1] != '"') {
                strcat(str, tok);
                strcat(str, " ");
                VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
   }
            strcat(str, tok);
            strncpy(strnew, str+1, strlen(str)-2);
            strcpy(tok, strnew);
        }
        Vnm_print(0, "NOsh: Storing molecule %d path %s\n",
      thee->nmol, tok);
        thee->molfmt[thee->nmol] = molfmt;
        strncpy(thee->molpath[thee->nmol], tok, VMAX_ARGLEN);
        (thee->nmol)++;
    } else if (Vstring_strcasecmp(tok, "xml") == 0) {
        molfmt = NMF_XML;
        VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
        if (tok[0]=='"') {
            strcpy(strnew, "");
            while (tok[strlen(tok)-1] != '"') {
                strcat(str, tok);
                strcat(str, " ");
                VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
   }
            strcat(str, tok);
            strncpy(strnew, str+1, strlen(str)-2);
            strcpy(tok, strnew);
        }
        Vnm_print(0, "NOsh: Storing molecule %d path %s\n",
      thee->nmol, tok);
        thee->molfmt[thee->nmol] = molfmt;
        strncpy(thee->molpath[thee->nmol], tok, VMAX_ARGLEN);
        (thee->nmol)++;
    } else {
        Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined mol format \
%s!\n", tok);
    }

    return 1;


VERROR1:
        Vnm_print(2, "NOsh_parseREAD_MOL:  Ran out of tokens while parsing READ
 section!\n");
 return 0;

}

VPRIVATE int NOsh_parseREAD_PARM(NOsh *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
    NOsh_ParmFormat parmfmt;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (Vstring_strcasecmp(tok, "flat") == 0) {
        parmfmt = NPF_FLAT;
        VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
        if (tok[0]=='"') {
            strcpy(strnew, "");
            while (tok[strlen(tok)-1] != '"') {
                strcat(str, tok);
                strcat(str, " ");
                VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
   }
            strcat(str, tok);
            strncpy(strnew, str+1, strlen(str)-2);
            strcpy(tok, strnew);
        }
        if (thee->gotparm) {
```

```
            Vnm_print(2, "NOsh:  Hey!  You already specified a parameterfile (%s)!\n", thee->parmpath);
            Vnm_print(2, "NOsh:  I'm going to ignore this one (%s)!\n", tok);
        } else {
            thee->parmfmt = parmfmt;
            thee->gotparm = 1;
            strncpy(thee->parmpath, tok, VMAX_ARGLEN);
        }
    } else if(Vstring_strcasecmp(tok, "xml") == 0) {
        parmfmt = NPF_XML;
        VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
        if (tok[0]=='"') {
            strcpy(strnew, "");
            while (tok[strlen(tok)-1] != '"') {
                strcat(str, tok);
                strcat(str, " ");
                VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
   }
            strcat(str, tok);
            strncpy(strnew, str+1, strlen(str)-2);
            strcpy(tok, strnew);
        }
        if (thee->gotparm) {
            Vnm_print(2, "NOsh:  Hey!  You already specified a parameterfile (%s)!\n", thee->parmpath);
            Vnm_print(2, "NOsh:  I'm going to ignore this one (%s)!\n", tok);
        } else {
            thee->parmfmt = parmfmt;
            thee->gotparm = 1;
            strncpy(thee->parmpath, tok, VMAX_ARGLEN);
        }

    } else {
        Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined parm format \
%s!\n", tok);
    }

    return 1;

VERROR1:
        Vnm_print(2, "NOsh_parseREAD_PARM:  Ran out of tokens while parsing READ section!\n");
 return 0;

}

VPRIVATE int NOsh_parseREAD_DIEL(NOsh *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
    Vdata_Format dielfmt;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (Vstring_strcasecmp(tok, "dx") == 0) {
        dielfmt = VDF_DX;
 } else if (Vstring_strcasecmp(tok, "gz") == 0) {
        dielfmt = VDF_GZ;
 } else {
        Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined format \
      %s!\n", tok);
  return VRC_FAILURE;
 }

 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
 if (tok[0]=='"') {
  strcpy(strnew, "");
  while (tok[strlen(tok)-1] != '"') {
   strcat(str, tok);
   strcat(str, " ");
   VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
  }
  strcat(str, tok);
  strncpy(strnew, str+1, strlen(str)-2);
  strcpy(tok, strnew);
 }
 Vnm_print(0, "NOsh: Storing x-shifted dielectric map %d path \
     %s\n", thee->ndiel, tok);
 strncpy(thee->dielXpath[thee->ndiel], tok, VMAX_ARGLEN);
 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
 Vnm_print(0, "NOsh: Storing y-shifted dielectric map %d path \
     %s\n", thee->ndiel, tok);
 strncpy(thee->dielYpath[thee->ndiel], tok, VMAX_ARGLEN);
```

```
 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
 Vnm_print(0, "NOsh: Storing z-shifted dielectric map %d path \
     %s\n", thee->ndiel, tok);
 strncpy(thee->dielZpath[thee->ndiel], tok, VMAX_ARGLEN);
 thee->dielfmt[thee->ndiel] = dielfmt;
 (thee->ndiel)++;

 return 1;

VERROR1:
        Vnm_print(2, "NOsh_parseREAD_DIEL:  Ran out of tokens while parsing READ \
section!\n");
 return 0;

}

VPRIVATE int NOsh_parseREAD_KAPPA(NOsh *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
    Vdata_Format kappafmt;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (Vstring_strcasecmp(tok, "dx") == 0) {
        kappafmt = VDF_DX;
 } else if (Vstring_strcasecmp(tok, "gz") == 0) {
        kappafmt = VDF_GZ;
 } else {
        Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined format \
      %s!\n", tok);
  return VRC_FAILURE;
 }

 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
 if (tok[0]=='"') {
  strcpy(strnew, "");
  while (tok[strlen(tok)-1] != '"') {
   strcat(str, tok);
   strcat(str, " ");
   VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
  }
  strcat(str, tok);
  strncpy(strnew, str+1, strlen(str)-2);
  strcpy(tok, strnew);
 }
 Vnm_print(0, "NOsh: Storing kappa map %d path %s\n",
     thee->nkappa, tok);
 thee->kappafmt[thee->nkappa] = kappafmt;
 strncpy(thee->kappapath[thee->nkappa], tok, VMAX_ARGLEN);
 (thee->nkappa)++;

    return 1;

VERROR1:
        Vnm_print(2, "NOsh_parseREAD:  Ran out of tokens while parsing READ \
section!\n");
 return 0;

}

VPRIVATE int NOsh_parseREAD_POTENTIAL(NOsh *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
    Vdata_Format potfmt;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (Vstring_strcasecmp(tok, "dx") == 0) {
        potfmt = VDF_DX;
 } else if (Vstring_strcasecmp(tok, "gz") == 0) {
        potfmt = VDF_GZ;
 } else {
        Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined format \
      %s!\n", tok);
  return VRC_FAILURE;
   }

 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
 if (tok[0]=='"') {
  strcpy(strnew, "");
  while (tok[strlen(tok)-1] != '"') {
   strcat(str, tok);
```

```
    strcat(str, " ");
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
   }
   strcat(str, tok);
   strncpy(strnew, str+1, strlen(str)-2);
   strcpy(tok, strnew);
  }
  Vnm_print(0, "NOsh: Storing potential map %d path %s\n",
      thee->npot, tok);
  thee->potfmt[thee->npot] = potfmt;
  strncpy(thee->potpath[thee->npot], tok, VMAX_ARGLEN);
  (thee->npot)++;

  return 1;

VERROR1:
  Vnm_print(2, "NOsh_parseREAD:  Ran out of tokens while parsing READ \
      section!\n");
  return 0;

}

VPRIVATE int NOsh_parseREAD_CHARGE(NOsh *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
    Vdata_Format chargefmt;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (Vstring_strcasecmp(tok, "dx") == 0) {
        chargefmt = VDF_DX;
  } else if (Vstring_strcasecmp(tok, "gz") == 0) {
        chargefmt = VDF_GZ;
  } else {
        Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined format \
      %s!\n", tok);
   return VRC_FAILURE;
  }

  VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
  if (tok[0]=='"') {
   strcpy(strnew, "");
   while (tok[strlen(tok)-1] != '"') {
    strcat(str, tok);
    strcat(str, " ");
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
   }
   strcat(str, tok);
   strncpy(strnew, str+1, strlen(str)-2);
   strcpy(tok, strnew);
  }
  Vnm_print(0, "NOsh: Storing charge map %d path %s\n",
      thee->ncharge, tok);
  thee->chargefmt[thee->ncharge] = chargefmt;
  strncpy(thee->chargepath[thee->ncharge], tok, VMAX_ARGLEN);
  (thee->ncharge)++;

    return 1;

VERROR1:
        Vnm_print(2, "NOsh_parseREAD:  Ran out of tokens while parsing READ \
section!\n");
  return 0;

}

VPRIVATE int NOsh_parseREAD_MESH(NOsh *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
    Vdata_Format meshfmt;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (Vstring_strcasecmp(tok, "mcsf") == 0) {
        meshfmt = VDF_MCSF;
        VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
        if (tok[0]=='"') {
            strcpy(strnew, "");
            while (tok[strlen(tok)-1] != '"') {
                strcat(str, tok);
                strcat(str, " ");
                VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
   }
```

```
            strcat(str, tok);
            strncpy(strnew, str+1, strlen(str)-2);
            strcpy(tok, strnew);
        }
        Vnm_print(0, "NOsh: Storing mesh %d path %s\n",
      thee->nmesh, tok);
        thee->meshfmt[thee->nmesh] = meshfmt;
        strncpy(thee->meshpath[thee->nmesh], tok, VMAX_ARGLEN);
        (thee->nmesh)++;
    } else {
        Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined mesh format \
      %s!\n", tok);
    }

    return 1;

VERROR1:
 Vnm_print(2, "NOsh_parseREAD:  Ran out of tokens while parsing READ \
     section!\n");
 return 0;

}


VPRIVATE int NOsh_parseREAD(NOsh *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];

    if (thee == VNULL) {
        Vnm_print(2, "NOsh_parseREAD:  Got NULL thee!\n");
        return 0;
    }

    if (sock == VNULL) {
        Vnm_print(2, "NOsh_parseREAD:  Got pointer to NULL socket!\n");
        return 0;
    }

    if (thee->parsed) {
        Vnm_print(2, "NOsh_parseREAD:  Already parsed an input file!\n");
        return 0;
    }

    /* Read until we run out of tokens (bad) or hit the "END" keyword (good) */
    while (Vio_scanf(sock, "%s", tok) == 1) {
        if (Vstring_strcasecmp(tok, "end") == 0) {
            Vnm_print(0, "NOsh: Done parsing READ section\n");
            return 1;
        } else if (Vstring_strcasecmp(tok, "mol") == 0) {
            NOsh_parseREAD_MOL(thee, sock);
        } else if (Vstring_strcasecmp(tok, "parm") == 0) {
            NOsh_parseREAD_PARM(thee,sock);
        } else if (Vstring_strcasecmp(tok, "diel") == 0) {
            NOsh_parseREAD_DIEL(thee,sock);
        } else if (Vstring_strcasecmp(tok, "kappa") == 0) {
            NOsh_parseREAD_KAPPA(thee,sock);
  } else if (Vstring_strcasecmp(tok, "pot") == 0) {
            NOsh_parseREAD_POTENTIAL(thee,sock);
        } else if (Vstring_strcasecmp(tok, "charge") == 0) {
            NOsh_parseREAD_CHARGE(thee,sock);
  } else if (Vstring_strcasecmp(tok, "mesh") == 0) {
   NOsh_parseREAD_MESH(thee,sock);
        } else {
            Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined keyword %s!\n",
       tok);
        }
    }

    /* We ran out of tokens! */
    Vnm_print(2, "NOsh_parseREAD:  Ran out of tokens while parsing READ \
section!\n");
    return 0;

}

VPRIVATE int NOsh_parsePRINT(NOsh *thee, Vio *sock) {

    char tok[VMAX_BUFSIZE];
 char name[VMAX_BUFSIZE];
    int ti, idx, expect, ielec, iapol;

```

```
    if (thee == VNULL) {
        Vnm_print(2, "NOsh_parsePRINT:  Got NULL thee!\n");
        return 0;
    }

    if (sock == VNULL) {
        Vnm_print(2, "NOsh_parsePRINT:  Got pointer to NULL socket!\n");
        return 0;
    }

    if (thee->parsed) {
        Vnm_print(2, "NOsh_parsePRINT:  Already parsed an input file!\n");
        return 0;
    }

    idx = thee->nprint;
    if (thee->nprint >= NOSH_MAXPRINT) {
        Vnm_print(2, "NOsh_parsePRINT:  Exceeded max number (%d) of PRINT \
sections\n",
      NOSH_MAXPRINT);
        return 0;
    }


    /* The first thing we read is the thing we want to print */
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (Vstring_strcasecmp(tok, "energy") == 0) {
        thee->printwhat[idx] = NPT_ENERGY;
        thee->printnarg[idx] = 0;
    } else if (Vstring_strcasecmp(tok, "force") == 0) {
        thee->printwhat[idx] = NPT_FORCE;
        thee->printnarg[idx] = 0;
 } else if (Vstring_strcasecmp(tok, "elecEnergy") == 0) {
        thee->printwhat[idx] = NPT_ELECENERGY;
        thee->printnarg[idx] = 0;
 } else if (Vstring_strcasecmp(tok, "elecForce") == 0) {
        thee->printwhat[idx] = NPT_ELECFORCE;
        thee->printnarg[idx] = 0;
    } else if (Vstring_strcasecmp(tok, "apolEnergy") == 0) {
        thee->printwhat[idx] = NPT_APOLENERGY;
        thee->printnarg[idx] = 0;
 } else if (Vstring_strcasecmp(tok, "apolForce") == 0) {
        thee->printwhat[idx] = NPT_APOLFORCE;
        thee->printnarg[idx] = 0;
    } else {
        Vnm_print(2, "NOsh_parsePRINT:  Undefined keyword %s while parsing \
PRINT section!\n", tok);
        return 0;
    }

    expect = 0;   /* We first expect a calculation ID (0) then an op (1) */

    /* Read until we run out of tokens (bad) or hit the "END" keyword (good) */
    while (Vio_scanf(sock, "%s", tok) == 1) {

        /* The next thing we read is either END or an ARG OP ARG statement */
        if (Vstring_strcasecmp(tok, "end") == 0) {
            if (expect != 0) {
                (thee->nprint)++;
                (thee->printnarg[idx])++;
                Vnm_print(0, "NOsh: Done parsing PRINT section\n");
                return 1;
            } else {
                Vnm_print(2, "NOsh_parsePRINT:  Got premature END to PRINT!\n")
;
                return 0;
            }
        } else {

            /* Grab a calculation ID */
            if ((sscanf(tok, "%d", &ti) == 1) &&
    (Vstring_isdigit(tok) == 1)) {
                if (expect == 0) {
                    thee->printcalc[idx][thee->printnarg[idx]
] = ti-1;
                    expect = 1;
                } else {
                    Vnm_print(2, "NOsh_parsePRINT:  Syntax error in PRINT \
section while reading %s!\n", tok);
                    return 0;
                }
```

```
    /* Grab addition operation */
            } else if (Vstring_strcasecmp(tok, "+") == 0) {
                if (expect == 1) {
                    thee->printop[idx][thee->printnarg[idx]] = 0;
                    (thee->printnarg[idx])++;
                    expect = 0;
                    if (thee->printnarg[idx] >= NOSH_MAXPOP) {
                        Vnm_print(2, "NOsh_parsePRINT:  Exceeded max number \(%d) of arguments for PRINT section!\n",
          NOSH_MAXPOP);
                        return 0;
                    }
                } else {
                    Vnm_print(2, "NOsh_parsePRINT:  Syntax error in PRINT \section while reading %s!\n", tok);
                    return 0;
                }
    /* Grab subtraction operation */
            } else if (Vstring_strcasecmp(tok, "-") == 0) {
                if (expect == 1) {
                    thee->printop[idx][thee->printnarg[idx]] = 1;
                    (thee->printnarg[idx])++;
                    expect = 0;
                    if (thee->printnarg[idx] >= NOSH_MAXPOP) {
                        Vnm_print(2, "NOsh_parseREAD:  Exceeded max number \(%d) of arguments for PRINT section!\n",
          NOSH_MAXPOP);
                        return 0;
                    }
                } else {
                    Vnm_print(2, "NOsh_parsePRINT:  Syntax error in PRINT \section while reading %s!\n", tok);
                    return 0;
                }
    /* Grab a calculation name from elec ID */
   } else if (sscanf(tok, "%s", name) == 1) {
       if (expect == 0) {
     for (ielec=0; ielec<thee->nelec; ielec++) {
         if (Vstring_strcasecmp(thee->elecname[ielec], name) == 0) {
       thee->printcalc[idx][thee->printnarg[idx]] = ielec;
       expect = 1;
       break;
         }
        }
     for (iapol=0; iapol<thee->napol; iapol++) {
         if (Vstring_strcasecmp(thee->apolname[iapol], name) == 0) {
       thee->printcalc[idx][thee->printnarg[idx]] = iapol;
       expect = 1;
       break;
         }
        }
     if (expect == 0) {
      Vnm_print(2, "No ELEC or APOL statement has been named %s!\n",
          name);
      return 0;
     }
                } else {
                    Vnm_print(2, "NOsh_parsePRINT:  Syntax error in PRINT \section while reading %s!\n", tok);
                    return 0;
                }
    /* Got bad operation */
            } else {
                Vnm_print(2, "NOsh_parsePRINT:  Undefined keyword %s while \parsing PRINT section!\n", tok);
                return 0;
            }
        } /* end parse token */

    } /* end while */

    VJMPERR1(0);

    /* We ran out of tokens! */
VERROR1:
```

```
        Vnm_print(2, "NOsh_parsePRINT:  Ran out of tokens while parsing PRINT \
section!\n");
        return 0;

}

VPRIVATE int NOsh_parseELEC(NOsh *thee, Vio *sock) {

 NOsh_calc *calc = VNULL;

    char tok[VMAX_BUFSIZE];

    if (thee == VNULL) {
        Vnm_print(2, "NOsh_parseELEC:  Got NULL thee!\n");
        return 0;
    }

    if (sock == VNULL) {
        Vnm_print(2, "NOsh_parseELEC:  Got pointer to NULL socket!\n");
        return 0;
    }

    if (thee->parsed) {
        Vnm_print(2, "NOsh_parseELEC:  Already parsed an input file!\n");
        return 0;
    }

    /* Get a pointer to the latest ELEC calc object and update the ELEC
  statement number */
    if (thee->nelec >= NOSH_MAXCALC) {
        Vnm_print(2, "NOsh:  Too many electrostatics calculations in this \
run!\n");
        Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
      NOSH_MAXCALC);
        return 1;
    }

    /* The next token HAS to be the method OR "name" */
 if (Vio_scanf(sock, "%s", tok) == 1) {
     if (Vstring_strcasecmp(tok, "name") == 0) {
      Vio_scanf(sock, "%s", tok);
            strncpy(thee->elecname[thee->nelec], tok, VMAX_ARGLEN)
;
   if (Vio_scanf(sock, "%s", tok) != 1) {
    Vnm_print(2, "NOsh_parseELEC:  Ran out of tokens while reading \
ELEC section!\n");
    return 0;
   }
  }
  if (Vstring_strcasecmp(tok, "mg-manual") == 0) {
   thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_MG
);
   calc = thee->elec[thee->nelec];
   (thee->nelec)++;
   calc->mgparm->type = MCT_MANUAL;
      return NOsh_parseMG(thee, sock, calc);
  } else if (Vstring_strcasecmp(tok, "mg-auto") == 0) {
   thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_MG
);
   calc = thee->elec[thee->nelec];
   (thee->nelec)++;
   calc->mgparm->type = MCT_AUTO;
      return NOsh_parseMG(thee, sock, calc);
  } else if (Vstring_strcasecmp(tok, "mg-para") == 0) {
   thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_MG
);
   calc = thee->elec[thee->nelec];
   (thee->nelec)++;
   calc->mgparm->type = MCT_PARALLEL;
      return NOsh_parseMG(thee, sock, calc);
  } else if (Vstring_strcasecmp(tok, "mg-dummy") == 0) {
   thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_MG
);
   calc = thee->elec[thee->nelec];
   (thee->nelec)++;
   calc->mgparm->type = MCT_DUMMY;
      return NOsh_parseMG(thee, sock, calc);
  } else if (Vstring_strcasecmp(tok, "fe-manual") == 0) {
   thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_FEM
);
   calc = thee->elec[thee->nelec];
```

```
    (thee->nelec)++;
    calc->femparm->type = FCT_MANUAL;
       return NOsh_parseFEM(thee, sock, calc);
   } else {
       Vnm_print(2, "NOsh_parseELEC: The method (\"mg\" or \"fem\") or \
\"name\" must be the first keyword in the ELEC section\n");
    return 0;
   }
  }

     Vnm_print(2, "NOsh_parseELEC:  Ran out of tokens while reading ELEC
 section!\n");
     return 0;

}

VPRIVATE int NOsh_parseAPOLAR(NOsh *thee, Vio *sock) {

  NOsh_calc *calc = VNULL;

     char tok[VMAX_BUFSIZE];

     if (thee == VNULL) {
         Vnm_print(2, "NOsh_parseAPOLAR:  Got NULL thee!\n");
         return 0;
     }

     if (sock == VNULL) {
         Vnm_print(2, "NOsh_parseAPOLAR:  Got pointer to NULL socket!\n");
         return 0;
     }

     if (thee->parsed) {
         Vnm_print(2, "NOsh_parseAPOLAR:  Already parsed an input file!\n");
         return 0;
     }

  /* Get a pointer to the latest ELEC calc object and update the ELEC
   statement number */
     if (thee->napol >= NOSH_MAXCALC) {
         Vnm_print(2, "NOsh:  Too many non-polar calculations in this \
run!\n");
         Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
       NOSH_MAXCALC);
         return 1;
     }

  /* The next token HAS to be the method OR "name" */
  if (Vio_scanf(sock, "%s", tok) == 1) {
      if (Vstring_strcasecmp(tok, "name") == 0) {
       Vio_scanf(sock, "%s", tok);
             strncpy(thee->apolname[thee->napol], tok, VMAX_ARGLEN)
;

    /* Parse the non-polar parameters */
    thee->apol[thee->napol] = NOsh_calc_ctor(NCT_APOL
);
    calc = thee->apol[thee->napol];
    (thee->napol)++;
    return NOsh_parseAPOL(thee, sock, calc);

    if (Vio_scanf(sock, "%s", tok) != 1) {
     Vnm_print(2, "NOsh_parseAPOLAR:  Ran out of tokens while reading \
APOLAR section!\n");
     return 0;
    }
   }
  }

  return 1;

}

VPUBLIC int NOsh_setupElecCalc(
          NOsh *thee,
          Valist *alist[NOSH_MAXMOL]
          ) {
  int ielec, imol, i;
  NOsh_calc *elec = VNULL;
  MGparm *mgparm = VNULL;
  Valist *mymol = VNULL;
```

```

 VASSERT(thee != VNULL);
 for (imol=0; imol<thee->nmol; imol++) {
  thee->alist[imol] = alist[imol];
 }


 for (ielec=0; ielec<(thee->nelec); ielec++) {
  /* Unload the calculation object containing the ELEC information */
  elec = thee->elec[ielec];

  if (((thee->ndiel != 0) || (thee->nkappa != 0) ||
    (thee->ncharge != 0) || (thee->npot != 0)) &&
    (elec->pbeparm->calcforce != PCF_NO)) {
   Vnm_print(2, "NOsh_setupElecCalc:  Calculation of forces disabled because surface \
map is used!\n");
   elec->pbeparm->calcforce = PCF_NO;
  }

  /* Setup the calculation */
  switch (elec->calctype) {
   case NCT_MG:
    /* Center on the molecules, if requested */
    mgparm = elec->mgparm;
    VASSERT(mgparm != VNULL);
    if (elec->mgparm->cmeth == MCM_MOLECULE) {
     VASSERT(mgparm->centmol >= 0);
     VASSERT(mgparm->centmol < thee->nmol);
     mymol = thee->alist[mgparm->centmol];
     VASSERT(mymol != VNULL);
     for (i=0; i<3; i++) {
      mgparm->center[i] = mymol->center[i];
     }
    }
    if (elec->mgparm->fcmeth == MCM_MOLECULE) {
     VASSERT(mgparm->fcentmol >= 0);
     VASSERT(mgparm->fcentmol < thee->nmol);
     mymol = thee->alist[mgparm->fcentmol];
     VASSERT(mymol != VNULL);
     for (i=0; i<3; i++) {
      mgparm->fcenter[i] = mymol->center[i];
     }
    }
    if (elec->mgparm->ccmeth == MCM_MOLECULE) {
     VASSERT(mgparm->ccentmol >= 0);
     VASSERT(mgparm->ccentmol < thee->nmol);
     mymol = thee->alist[mgparm->ccentmol];
     VASSERT(mymol != VNULL);
     for (i=0; i<3; i++) {
      mgparm->ccenter[i] = mymol->center[i];
     }
    }
    NOsh_setupCalcMG(thee, elec);
    break;
   case NCT_FEM:
    NOsh_setupCalcFEM(thee, elec);
    break;
   default:
    Vnm_print(2, "NOsh_setupCalc:  Invalid calculation type (%d)!\n",
        elec->calctype);
    return 0;
  }

  /* At this point, the most recently-created NOsh_calc object should be the
   one we use for results for this ELEC statement.  Assign it. */
  /* Associate ELEC statement with the calculation */
  thee->elec2calc[ielec] = thee->ncalc-1;
  Vnm_print(0, "NOsh_setupCalc:  Mapping ELEC statement %d (%d) to \
calculation %d (%d)\n", ielec, ielec+1, thee->elec2calc[ielec],
      thee->elec2calc[ielec]+1);
 }

 return 1;
}

VPUBLIC int NOsh_setupApolCalc(
          NOsh *thee,
          Valist *alist[NOSH_MAXMOL]
          ) {
 int iapol, imol;
```

```
 int doCalc = ACD_NO;
 NOsh_calc *calc = VNULL;

 VASSERT(thee != VNULL);
 for (imol=0; imol<thee->nmol; imol++) {
  thee->alist[imol] = alist[imol];
 }

 for (iapol=0; iapol<(thee->napol); iapol++) {
  /* Unload the calculation object containing the APOL information */
  calc = thee->apol[iapol];

  /* Setup the calculation */
  switch (calc->calctype) {
   case NCT_APOL:
    NOsh_setupCalcAPOL(thee, calc);
    doCalc = ACD_YES;
    break;
   default:
    Vnm_print(2, "NOsh_setupCalc:  Invalid calculation type (%d)!\n", calc->calctype);
    return ACD_ERROR;
  }
  /* At this point, the most recently-created NOsh_calc object should be the
   one we use for results for this APOL statement.  Assign it. */
  /* Associate APOL statement with the calculation */
  thee->apol2calc[iapol] = thee->ncalc-1;
  Vnm_print(0, "NOsh_setupCalc:  Mapping APOL statement %d (%d) to calculation %d (%d)\n", iapol, iapol+1, thee->apol2calc[iapol], thee->apol2calc[iapol]+1);
 }

 if(doCalc == ACD_YES){
  return ACD_YES;
 }else{
  return ACD_NO;
 }
}

VPUBLIC int NOsh_parseMG(
       NOsh *thee,
       Vio *sock,
       NOsh_calc *elec
       ) {

 char tok[VMAX_BUFSIZE];
 MGparm *mgparm = VNULL;
 PBEparm *pbeparm = VNULL;
 int rc;

 /* Check the arguments */
 if (thee == VNULL) {
  Vnm_print(2, "NOsh:  Got NULL thee!\n");
  return 0;
 }
 if (sock == VNULL) {
  Vnm_print(2, "NOsh:  Got pointer to NULL socket!\n");
  return 0;
 }
 if (elec == VNULL) {
  Vnm_print(2, "NOsh:  Got pointer to NULL elec object!\n");
  return 0;
 }
 mgparm = elec->mgparm;
 if (mgparm == VNULL) {
  Vnm_print(2, "NOsh:  Got pointer to NULL mgparm object!\n");
  return 0;
 }
 pbeparm = elec->pbeparm;
 if (pbeparm == VNULL) {
  Vnm_print(2, "NOsh:  Got pointer to NULL pbeparm object!\n");
  return 0;
 }

 Vnm_print(0, "NOsh_parseMG: Parsing parameters for MG calculation\n");

 /* Parallel stuff */
 if (mgparm->type == MCT_PARALLEL) {
  mgparm->proc_rank = thee->proc_rank;
  mgparm->proc_size = thee->proc_size;
  mgparm->setrank = 1;
```

```
  mgparm->setsize = 1;
 }


 /* Start snarfing tokens from the input stream */
 rc = 1;
 while (Vio_scanf(sock, "%s", tok) == 1) {

  Vnm_print(0, "NOsh_parseMG:  Parsing %s...\n", tok);

  /* See if it's an END token */
  if (Vstring_strcasecmp(tok, "end") == 0) {
   mgparm->parsed = 1;
   pbeparm->parsed = 1;
   rc = 1;
   break;
  }

  /* Pass the token through a series of parsers */
  rc = PBEparm_parseToken(pbeparm, tok, sock);
  if (rc == -1) {
   Vnm_print(0, "NOsh_parseMG:  parsePBE error!\n");
   break;
  } else if (rc == 0) {
   /* Pass the token to the generic MG parser */
   rc = MGparm_parseToken(mgparm, tok, sock);
   if (rc == -1) {
    Vnm_print(0, "NOsh_parseMG:  parseMG error!\n");
    break;
   } else if (rc == 0) {
    /* We ran out of parsers! */
    Vnm_print(2, "NOsh:  Unrecognized keyword: %s\n", tok);
    break;
   }
  }
 }

 /* Handle various errors arising in the token-snarfing loop -- these all
  just result in simple returns right now */
 if (rc == -1) return 0;
 if (rc == 0) return 0;

 /* Check the status of the parameter objects */
 if ((MGparm_check(mgparm) == VRC_FAILURE) || (!
PBEparm_check(pbeparm))) {
  Vnm_print(2, "NOsh:  MG parameters not set correctly!\n");
  return 0;
 }

 return 1;
}

VPRIVATE int NOsh_setupCalcMG(
         NOsh *thee,
         NOsh_calc *calc
         ) {

 MGparm *mgparm = VNULL;

 VASSERT(thee != VNULL);
 VASSERT(calc != VNULL);
 mgparm = calc->mgparm;
 VASSERT(mgparm != VNULL);


 /* Now we're ready to whatever sorts of post-processing operations that are
  necessary for the various types of calculations */
 switch (mgparm->type) {
  case MCT_MANUAL:
   return NOsh_setupCalcMGMANUAL(thee, calc);
  case MCT_DUMMY:
   return NOsh_setupCalcMGMANUAL(thee, calc);
  case MCT_AUTO:
   return NOsh_setupCalcMGAUTO(thee, calc);
  case MCT_PARALLEL:
   return NOsh_setupCalcMGPARA(thee, calc);
  default:
   Vnm_print(2, "NOsh_setupCalcMG:  undefined MG calculation type (%d)!\n",
       mgparm->type);
   return 0;
 }
```

```
 /* Shouldn't get here */
 return 0;
}

VPRIVATE int NOsh_setupCalcFEM(
          NOsh *thee,
          NOsh_calc *calc
          ) {

 VASSERT(thee != VNULL);
 VASSERT(calc != VNULL);
 VASSERT(calc->femparm != VNULL);

 /* Now we're ready to whatever sorts of post-processing operations that are
  * necessary for the various types of calculations */
 switch (calc->femparm->type) {
  case FCT_MANUAL:
   return NOsh_setupCalcFEMANUAL(thee, calc);
  default:
   Vnm_print(2, "NOsh_parseFEM:  unknown calculation type (%d)!\n",
       calc->femparm->type);
   return 0;
 }

 /* Shouldn't get here */
 return 0;
}


VPRIVATE int NOsh_setupCalcMGMANUAL(
           NOsh *thee,
           NOsh_calc *elec
           ) {

 MGparm *mgparm = VNULL;
 PBEparm *pbeparm = VNULL;
 NOsh_calc *calc = VNULL;

 if (thee == VNULL) {
  Vnm_print(2, "NOsh_setupCalcMGMANUAL:  Got NULL thee!\n");
  return 0;
 }
 if (elec == VNULL) {
  Vnm_print(2, "NOsh_setupCalcMGMANUAL:  Got NULL calc!\n");
  return 0;
 }
 mgparm = elec->mgparm;
 if (mgparm == VNULL) {
  Vnm_print(2, "NOsh_setupCalcMGMANUAL:  Got NULL mgparm -- was this calculation \
set up?\n");
  return 0;
 }
 pbeparm = elec->pbeparm;
 if (pbeparm == VNULL) {
  Vnm_print(2, "NOsh_setupCalcMGMANUAL:  Got NULL pbeparm -- was this calculation \
set up?\n");
  return 0;
 }

 /* Set up missing MG parameters */
 if (mgparm->setgrid == 0) {
  VASSERT(mgparm->setglen);
  mgparm->grid[0] = mgparm->glen[0]/((double)(mgparm->dime[0]-1));
  mgparm->grid[1] = mgparm->glen[1]/((double)(mgparm->dime[1]-1));
  mgparm->grid[2] = mgparm->glen[2]/((double)(mgparm->dime[2]-1));
 }
 if (mgparm->setglen == 0) {
  VASSERT(mgparm->setgrid);
  mgparm->glen[0] = mgparm->grid[0]*((double)(mgparm->dime[0]-1));
  mgparm->glen[1] = mgparm->grid[1]*((double)(mgparm->dime[1]-1));
  mgparm->glen[2] = mgparm->grid[2]*((double)(mgparm->dime[2]-1));
 }

 /* Check to see if he have any room left for this type of calculation, if
  so: set the calculation type, update the number of calculations of this type,
  and parse the rest of the section */
 if (thee->ncalc >= NOSH_MAXCALC) {
```

```
  Vnm_print(2, "NOsh:  Too many calculations in this run!\n");
  Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
      NOSH_MAXCALC);
  return 0;
 }

    /* Get the next calculation object and increment the number of calculations */
 thee->calc[thee->ncalc] = NOsh_calc_ctor(NCT_MG);
 calc = thee->calc[thee->ncalc];
 (thee->ncalc)++;



 /* Copy over contents of ELEC */
 NOsh_calc_copy(calc, elec);


    return 1;
}

VPUBLIC int NOsh_setupCalcMGAUTO(
         NOsh *thee,
         NOsh_calc *elec
         ) {

 NOsh_calc *calcf = VNULL;
 NOsh_calc *calcc = VNULL;
 double fgrid[3], cgrid[3];
 double d[3], minf[3], maxf[3], minc[3], maxc[3];
 double redfrac, redrat[3], td;
 int ifocus, nfocus, tnfocus[3];
 int j;
 int icalc;
 int dofix;

 /* A comment about the coding style in this function.  I use lots and lots
  and lots of pointer deferencing.  I could (and probably should) save
  these in temporary variables.  However, since there are so many MGparm,
  etc. and NOsh_calc, etc. objects running around in this function, the
  current scheme is easiest to debug. */


 if (thee == VNULL) {
  Vnm_print(2, "NOsh_setupCalcMGAUTO:  Got NULL thee!\n");
  return 0;
 }
 if (elec == VNULL) {
  Vnm_print(2, "NOsh_setupCalcMGAUTO:  Got NULL elec!\n");
  return 0;
 }
 if (elec->mgparm == VNULL) {
  Vnm_print(2, "NOsh_setupCalcMGAUTO:  Got NULL mgparm!\n");
  return 0;
 }
 if (elec->pbeparm == VNULL) {
  Vnm_print(2, "NOsh_setupCalcMGAUTO:  Got NULL pbeparm!\n");
  return 0;
 }

 Vnm_print(0, "NOsh_setupCalcMGAUTO(%s, %d):  coarse grid center = %g %g %g\n",
     __FILE__, __LINE__,
     elec->mgparm->ccenter[0],
     elec->mgparm->ccenter[1],
     elec->mgparm->ccenter[2]);
 Vnm_print(0, "NOsh_setupCalcMGAUTO(%s, %d):  fine grid center = %g %g %g\n",
     __FILE__, __LINE__,
     elec->mgparm->fcenter[0],
     elec->mgparm->fcenter[1],
     elec->mgparm->fcenter[2]);

 /* Calculate the grid spacing on the coarse and fine levels */
 for (j=0; j<3; j++) {
  cgrid[j] = (elec->mgparm->cglen[j])/((double)(elec->mgparm->dime[j]-1));
  fgrid[j] = (elec->mgparm->fglen[j])/((double)(elec->mgparm->dime[j]-1));
  d[j] = elec->mgparm->fcenter[j] - elec->mgparm->ccenter[j];
 }
 Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  Coarse grid spacing = %g, %g, %g
```

```
\n",
     __FILE__, __LINE__, cgrid[0], cgrid[1], cgrid[2]);
 Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  Fine grid spacing = %g, %g, %g\n",
     __FILE__, __LINE__, fgrid[0], fgrid[1], fgrid[2]);
 Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  Displacement between fine and \
coarse grids = %g, %g, %g\n", __FILE__, __LINE__, d[0], d[1], d[2]);

 /* Now calculate the number of focusing levels, never reducing the grid
  spacing by more than redfrac at each level */
 for (j=0; j<3; j++) {
  if (fgrid[j]/cgrid[j] < VREDFRAC) {
   redfrac = fgrid[j]/cgrid[j];
   td = log(redfrac)/log(VREDFRAC);
   tnfocus[j] = (int)ceil(td) + 1;
  } else tnfocus[j] = 2;
 }
 nfocus = VMAX2(VMAX2(tnfocus[0], tnfocus[1]), tnfocus[2]);

 /* Now set redrat to the actual value by which the grid spacing is reduced
  at each level of focusing */
 for (j=0; j<3; j++) {
  redrat[j] = VPOW((fgrid[j]/cgrid[j]), 1.0/((double)nfocus-1.0));
 }
 Vnm_print(0, "NOsh:  %d levels of focusing with %g, %g, %g reductions\n",
     nfocus, redrat[0], redrat[1], redrat[2]);

 /* Now that we know how many focusing levels to use, we're ready to set up
  the parameter objects */
 if (nfocus > (NOSH_MAXCALC-(thee->ncalc))) {
  Vnm_print(2, "NOsh:  Require more calculations than max (%d)!\n",
      NOSH_MAXCALC);
  return 0;
 }

 for (ifocus=0; ifocus<nfocus; ifocus++) {

  /* Generate the new calc object */
  icalc = thee->ncalc;
  thee->calc[icalc] = NOsh_calc_ctor(NCT_MG);
  (thee->ncalc)++;

  /* This is the _current_ NOsh_calc object */
  calcf = thee->calc[icalc];
  /* This is the _previous_ Nosh_calc object */
  if (ifocus != 0) {
   calcc = thee->calc[icalc-1];
  } else {
   calcc = VNULL;
  }

  /* Copy over most of the parameters from the ELEC object */
  NOsh_calc_copy(calcf, elec);

  /* Set up the grid lengths and spacings */
  if (ifocus == 0) {
   for (j=0; j<3; j++) {
    calcf->mgparm->grid[j] = cgrid[j];
    calcf->mgparm->glen[j] = elec->mgparm->cglen[j];
   }
  } else {
   for (j=0; j<3; j++) {
    calcf->mgparm->grid[j] = redrat[j]*(calcc->mgparm->grid[j]);
                calcf->mgparm->glen[j] = redrat[j]*(calcc->mgparm->glen[j]);
            }
        }
        calcf->mgparm->setgrid = 1;
        calcf->mgparm->setglen = 1;

  /* Get centers and centering method from coarse and fine meshes */
        if (ifocus == 0) {
   calcf->mgparm->cmeth = elec->mgparm->ccmeth;
   calcf->mgparm->centmol = elec->mgparm->ccentmol;
   for (j=0; j<3; j++) {
    calcf->mgparm->center[j] = elec->mgparm->ccenter[j];
   }
  } else if (ifocus == (nfocus-1)) {
   calcf->mgparm->cmeth = elec->mgparm->fcmeth;
```

```
    calcf->mgparm->centmol = elec->mgparm->fcentmol;
    for (j=0; j<3; j++) {
     calcf->mgparm->center[j] = elec->mgparm->fcenter[j];
    }
   } else {
    calcf->mgparm->cmeth = MCM_FOCUS;
    /* TEMPORARILY move the current grid center
    to the fine grid center.  In general, this will move portions of
    the current mesh off the immediately-coarser mesh.  We'll fix that
    in the next step. */
    for (j=0; j<3; j++) {
     calcf->mgparm->center[j] = elec->mgparm->fcenter[j];
    }
   }


   /* As mentioned above, it is highly likely that the previous "jump"
    to the fine grid center put portions of the current mesh off the
    previous (coarser) mesh.  Fix this by displacing the current mesh
    back onto the previous coarser mesh.  */
   if (ifocus != 0) {
    Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  starting mesh \repositioning.\n", __FILE__, __LINE__);
    Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  coarse mesh center = \%g %g %g\n", __FILE__, __LINE__,
        calcc->mgparm->center[0],
          calcc->mgparm->center[1],
          calcc->mgparm->center[2]);
    Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  coarse mesh upper corner = \%g %g %g\n", __FILE__, __LINE__,
        calcc->mgparm->center[0]+0.5*(calcc->mgparm->glen[0]),
          calcc->mgparm->center[1]+0.5*(calcc->mgparm->glen[1]),
          calcc->mgparm->center[2]+0.5*(calcc->mgparm->glen[2]));
    Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  coarse mesh lower corner = \%g %g %g\n", __FILE__, __LINE__,
        calcc->mgparm->center[0]-0.5*(calcc->mgparm->glen[0]),
          calcc->mgparm->center[1]-0.5*(calcc->mgparm->glen[1]),
          calcc->mgparm->center[2]-0.5*(calcc->mgparm->glen[2]));
    Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  initial fine mesh upper corner = \%g %g %g\n", __FILE__, __LINE__,
        calcf->mgparm->center[0]+0.5*(calcf->mgparm->glen[0]),
          calcf->mgparm->center[1]+0.5*(calcf->mgparm->glen[1]),
          calcf->mgparm->center[2]+0.5*(calcf->mgparm->glen[2]));
    Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  initial fine mesh lower corner = \%g %g %g\n", __FILE__, __LINE__,
        calcf->mgparm->center[0]-0.5*(calcf->mgparm->glen[0]),
          calcf->mgparm->center[1]-0.5*(calcf->mgparm->glen[1]),
          calcf->mgparm->center[2]-0.5*(calcf->mgparm->glen[2]));
    for (j=0; j<3; j++) {
     /* Check if we've fallen off of the lower end of the mesh */
     dofix = 0;
     minf[j] = calcf->mgparm->center[j]
      - 0.5*(calcf->mgparm->glen[j]);
     minc[j] = calcc->mgparm->center[j]
      - 0.5*(calcc->mgparm->glen[j]);
     d[j] = minc[j] - minf[j];
     if (d[j] >= VSMALL) {
      if (ifocus == (nfocus-1)) {
       Vnm_print(2, "NOsh_setupCalcMGAUTO:  Error!  Finest \mesh has fallen off the coarser meshes!\n");
       Vnm_print(2, "NOsh_setupCalcMGAUTO:  difference in min %d-\direction = %g\n", j, d[j]);
       Vnm_print(2, "NOsh_setupCalcMGAUTO:  min fine = %g %g %g\n",
           minf[0], minf[1], minf[2]);
       Vnm_print(2, "NOsh_setupCalcMGAUTO:  min coarse = %g %g %g\n",
```

```
          minc[0], minc[1], minc[2]);
      VASSERT(0);
     } else {
      Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  ifocus = %d, \
fixing mesh min violation (%g in %d-direction).\n", __FILE__, __LINE__, ifocus,

          d[j], j);
      calcf->mgparm->center[j] += d[j];
      dofix = 1;
     }
    }
    /* Check if we've fallen off of the upper end of the mesh */
    maxf[j] = calcf->mgparm->center[j] \
     + 0.5*(calcf->mgparm->glen[j]);
    maxc[j] = calcc->mgparm->center[j] \
     + 0.5*(calcc->mgparm->glen[j]);
    d[j] = maxf[j] - maxc[j];
    if (d[j] >= VSMALL) {
     if (ifocus == (nfocus-1)) {
      Vnm_print(2, "NOsh_setupCalcMGAUTO:  Error!  Finest \
mesh has fallen off the coarser meshes!\n");
      Vnm_print(2, "NOsh_setupCalcMGAUTO:  difference in %d-\
direction = %g\n", j, d[j]);
      VASSERT(0);
     } else {
      /* If we already fixed the lower boundary and we now need
      to fix the upper boundary, we have a serious problem. */
      if (dofix) {
       Vnm_print(2, "NOsh_setupCalcMGAUTO:  Error!  Both \
ends of the finer mesh do not fit in the bigger mesh!\n");
       VASSERT(0);
      }
      Vnm_print(0, "NOsh_setupCalcMGAUTO(%s, %d):  ifocus = %d, \
fixing mesh max violation (%g in %d-direction).\n", __FILE__, __LINE__, ifocus,

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

  /* Finer levels have focusing boundary conditions */
  if (ifocus != 0) calcf->pbeparm->bcfl = BCFL_FOCUS;

  /* Only the finest level handles I/O and needs to worry about disjoint
   partitioning */
  if (ifocus != (nfocus-1)) calcf->pbeparm->numwrite = 0;

  /* Reset boundary flags for everything except parallel focusing */
        if (calcf->mgparm->type != MCT_PARALLEL)  {
   Vnm_print(0, "NOsh_setupMGAUTO:  Resetting boundary flags\n");
   for (j=0; j<6; j++) calcf->mgparm->partDisjOwnSide[j] = 0;
   for (j=0; j<3; j++) {
    calcf->mgparm->partDisjCenter[j] = 0;
    calcf->mgparm->partDisjLength[j] = calcf->mgparm->glen[j];
   }
        }
```

```
        calcf->mgparm->parsed = 1;
    }


    return 1;
}

/* Author:   Nathan Baker and Todd Dolinsky */
VPUBLIC int NOsh_setupCalcMGPARA(
         NOsh *thee,
         NOsh_calc *elec
         ) {

 /* NEW (25-Jul-2006):  This code should produce modify the ELEC statement
 and pass it on to MGAUTO for further processing. */

 MGparm *mgparm = VNULL;
 double ofrac;
 double hx, hy, hzed;
 double xofrac, yofrac, zofrac;
 int rank, size, npx, npy, npz, nproc, ip, jp, kp;
 int xeffGlob, yeffGlob, zeffGlob, xDisj, yDisj, zDisj;
 int xigminDisj, xigmaxDisj, yigminDisj, yigmaxDisj, zigminDisj, zigmaxDisj;
 int xigminOlap, xigmaxOlap, yigminOlap, yigmaxOlap, zigminOlap, zigmaxOlap;
 int xOlapReg, yOlapReg, zOlapReg;
 double xlenDisj, ylenDisj, zlenDisj;
 double xcentDisj, ycentDisj, zcentDisj;
 double xcentOlap, ycentOlap, zcentOlap;
 double xlenOlap, ylenOlap, zlenOlap;
 double xminOlap, xmaxOlap, yminOlap, ymaxOlap, zminOlap, zmaxOlap;
 double xminDisj, xmaxDisj, yminDisj, ymaxDisj, zminDisj, zmaxDisj;
 double xcent, ycent, zcent;

 /* Grab some useful variables */
 VASSERT(thee != VNULL);
 VASSERT(elec != VNULL);
 mgparm = elec->mgparm;
 VASSERT(mgparm != VNULL);

 /* Grab some useful variables */
    ofrac = mgparm->ofrac;
    npx = mgparm->pdime[0];
    npy = mgparm->pdime[1];
    npz = mgparm->pdime[2];
    nproc = npx*npy*npz;

 /* If this is not an asynchronous calculation, then we need to make sure we
  have all the necessary MPI information */
    if (mgparm->setasync == 0) {

#ifndef HAVE_MPI_H

        Vnm_tprint(2, "NOsh_setupCalcMGPARA:  Oops!  You're trying to perform \
an 'mg-para' (parallel) calculation\n");
        Vnm_tprint(2, "NOsh_setupCalcMGPARA:  with a version of APBS that
    wasn't \
compiled with MPI!\n");
  Vnm_tprint(2, "NOsh_setupCalcMGPARA:  Perhaps you meant to use the \
'async' flag?\n");
        Vnm_tprint(2, "NOsh_setupCalcMGPARA:  Bailing out!\n");

        return 0;

#endif

  rank = thee->proc_rank;
  size = thee->proc_size;
  Vnm_print(0, "NOsh_setupCalcMGPARA:  Hello from processor %d of %d\n", rank,
      size);

  /* Check to see if we have too many processors.  If so, then simply set
   this processor to duplicating the work of processor 0. */
  if (rank > (nproc-1)) {
   Vnm_print(2, "NOsh_setupMGPARA:  There are more processors available than\
the %d you requested.\n", nproc);
   Vnm_print(2, "NOsh_setupMGPARA:  Eliminating processor %d\n", rank);
   thee->bogus = 1;
   rank = 0;
  }

  /* Check to see if we have too few processors.  If so, this is a fatal
```

```
    error. */
   if (size < nproc) {
    Vnm_print(2, "NOsh_setupMGPARA:  There are too few processors (%d) to \
satisfy requirements (%d)\n", size, nproc);
    return 0;
   }

   Vnm_print(0, "NOsh_setupMGPARA:  Hello (again) from processor %d of %d\n",
       rank, size);

     } else { /* Setting up for an asynchronous calculation. */

   rank = mgparm->async;

   thee->ispara = 1;
   thee->proc_rank = rank;

         /* Check to see if the async id is greater than the number of
   * processors.  If so, this is a fatal error. */
         if (rank > (nproc-1)) {
             Vnm_print(2, "NOsh_setupMGPARA:  The processor id you requested (%d) \
is not within the range of processors available (0-%d)\n", rank, (nproc-1));
             return 0;
         }
     }

     /* Calculate the processor's coordinates in the processor grid */
  kp = (int)floor(rank/(npx*npy));
  jp = (int)floor((rank-kp*npx*npy)/npx);
  ip = rank - kp*npx*npy - jp*npx;
  Vnm_print(0, "NOsh_setupMGPARA:  Hello world from PE (%d, %d, %d)\n",
      ip, jp, kp);

  /* Calculate effective overlap fractions for uneven processor distributions */
  if (npx == 1) xofrac = 0.0;
  else xofrac = ofrac;
  if (npy == 1) yofrac = 0.0;
  else yofrac = ofrac;
  if (npz == 1) zofrac = 0.0;
  else zofrac = ofrac;

     /* Calculate the global grid size and spacing */
     xDisj = (int)VFLOOR(mgparm->dime[0]/(1 + 2*xofrac) + 0.5);
     xeffGlob = npx*xDisj;
     hx = mgparm->fglen[0]/(double)(xeffGlob-1);
     yDisj = (int)VFLOOR(mgparm->dime[1]/(1 + 2*yofrac) + 0.5);
     yeffGlob = npy*yDisj;
     hy = mgparm->fglen[1]/(double)(yeffGlob-1);
     zDisj = (int)VFLOOR(mgparm->dime[2]/(1 + 2*zofrac) + 0.5);
     zeffGlob = npz*zDisj;
     hzed = mgparm->fglen[2]/(double)(zeffGlob-1);
     Vnm_print(0, "NOsh_setupMGPARA:  Global Grid size = (%d, %d, %d)\n",
               xeffGlob, yeffGlob, zeffGlob);
     Vnm_print(0, "NOsh_setupMGPARA:  Global Grid Spacing = (%.3f, %.3f, %.3f)\n",
               hx, hy, hzed);
     Vnm_print(0, "NOsh_setupMGPARA:  Processor Grid Size = (%d, %d, %d)\n",
               xDisj, yDisj, zDisj);

     /* Calculate the maximum and minimum processor grid points */
     xigminDisj = ip*xDisj;
     xigmaxDisj = xigminDisj + xDisj - 1;
     yigminDisj = jp*yDisj;
     yigmaxDisj = yigminDisj + yDisj - 1;
     zigminDisj = kp*zDisj;
     zigmaxDisj = zigminDisj + zDisj - 1;
     Vnm_print(0, "NOsh_setupMGPARA:  Min Grid Points for this proc. (%d, %d, %d)\n",
               xigminDisj, yigminDisj, zigminDisj);
     Vnm_print(0, "NOsh_setupMGPARA:  Max Grid Points for this proc. (%d, %d, %d)\n",
               xigmaxDisj, yigmaxDisj, zigmaxDisj);


     /* Calculate the disjoint partition length and center displacement */
     xminDisj = VMAX2(hx*(xigminDisj-0.5), 0.0);
     xmaxDisj = VMIN2(hx*(xigmaxDisj+0.5), mgparm->fglen[0]);
     xlenDisj = xmaxDisj - xminDisj;
     yminDisj = VMAX2(hy*(yigminDisj-0.5), 0.0);
     ymaxDisj = VMIN2(hy*(yigmaxDisj+0.5), mgparm->fglen[1]);
```

```
    ylenDisj = ymaxDisj - yminDisj;
    zminDisj = VMAX2(hzed*(zigminDisj-0.5), 0.0);
    zmaxDisj = VMIN2(hzed*(zigmaxDisj+0.5), mgparm->fglen[2]);
    zlenDisj = zmaxDisj - zminDisj;

    xcent = 0.5*mgparm->fglen[0];
    ycent = 0.5*mgparm->fglen[1];
    zcent = 0.5*mgparm->fglen[2];

    xcentDisj = xminDisj + 0.5*xlenDisj - xcent;
    ycentDisj = yminDisj + 0.5*ylenDisj - ycent;
    zcentDisj = zminDisj + 0.5*zlenDisj - zcent;
    if (VABS(xcentDisj) < VSMALL) xcentDisj = 0.0;
    if (VABS(ycentDisj) < VSMALL) ycentDisj = 0.0;
    if (VABS(zcentDisj) < VSMALL) zcentDisj = 0.0;

    Vnm_print(0, "NOsh_setupMGPARA:  Disj part length = (%g, %g, %g)\n",
     xlenDisj, ylenDisj, zlenDisj);
    Vnm_print(0, "NOsh_setupMGPARA:  Disj part center displacement = (%g, %g, %g)\n",
     xcentDisj, ycentDisj, zcentDisj);

    /* Calculate the overlapping partition length and center displacement */
    xOlapReg = 0;
    yOlapReg = 0;
    zOlapReg = 0;
    if (npx != 1) xOlapReg = (int)VFLOOR(xofrac*mgparm->fglen[0]/npx/hx + 0.5) + 1;
    if (npy != 1) yOlapReg = (int)VFLOOR(yofrac*mgparm->fglen[1]/npy/hy + 0.5) + 1;
    if (npz != 1) zOlapReg = (int)VFLOOR(zofrac*mgparm->fglen[2]/npz/hzed + 0.5) + 1;

    Vnm_print(0, "NOsh_setupMGPARA:  No. of Grid Points in Overlap (%d, %d, %d)\n",
              xOlapReg, yOlapReg, zOlapReg);

    if (ip == 0) xigminOlap = 0;
    else if (ip == (npx - 1)) xigminOlap = xeffGlob - mgparm->dime[0];
    else xigminOlap = xigminDisj - xOlapReg;
    xigmaxOlap = xigminOlap + mgparm->dime[0] - 1;

    if (jp == 0) yigminOlap = 0;
    else if (jp == (npy - 1)) yigminOlap = yeffGlob - mgparm->dime[1];
    else yigminOlap = yigminDisj - yOlapReg;
    yigmaxOlap = yigminOlap + mgparm->dime[1] - 1;

    if (kp == 0) zigminOlap = 0;
    else if (kp == (npz - 1)) zigminOlap = zeffGlob - mgparm->dime[2];
    else zigminOlap = zigminDisj - zOlapReg;
    zigmaxOlap = zigminOlap + mgparm->dime[2] - 1;

    Vnm_print(0, "NOsh_setupMGPARA:  Min Grid Points with Overlap (%d, %d, %d)\n",
              xigminOlap, yigminOlap, zigminOlap);
    Vnm_print(0, "NOsh_setupMGPARA:  Max Grid Points with Overlap (%d, %d, %d)\n",
              xigmaxOlap, yigmaxOlap, zigmaxOlap);

    xminOlap = hx * xigminOlap;
    xmaxOlap = hx * xigmaxOlap;
    yminOlap = hy * yigminOlap;
    ymaxOlap = hy * yigmaxOlap;
    zminOlap = hzed * zigminOlap;
    zmaxOlap = hzed * zigmaxOlap;

    xlenOlap = xmaxOlap - xminOlap;
    ylenOlap = ymaxOlap - yminOlap;
    zlenOlap = zmaxOlap - zminOlap;

    xcentOlap = (xminOlap + 0.5*xlenOlap) - xcent;
    ycentOlap = (yminOlap + 0.5*ylenOlap) - ycent;
    zcentOlap = (zminOlap + 0.5*zlenOlap) - zcent;
    if (VABS(xcentOlap) < VSMALL) xcentOlap = 0.0;
    if (VABS(ycentOlap) < VSMALL) ycentOlap = 0.0;
    if (VABS(zcentOlap) < VSMALL) zcentOlap = 0.0;

    Vnm_print(0, "NOsh_setupMGPARA:  Olap part length = (%g, %g, %g)\n",
     xlenOlap, ylenOlap, zlenOlap);
    Vnm_print(0, "NOsh_setupMGPARA:  Olap part center displacement = (%g, %g, %g)\n",
```

```
     xcentOlap, ycentOlap, zcentOlap);


    /* Calculate the boundary flags:
  Flags are set to 1 when another processor is present along the boundary
  Flags are otherwise set to 0. */

    if (ip == 0) mgparm->partDisjOwnSide[VAPBS_LEFT] = 0;
 else mgparm->partDisjOwnSide[VAPBS_LEFT] = 1;
 if (ip == (npx-1)) mgparm->partDisjOwnSide[VAPBS_RIGHT] = 0;
 else mgparm->partDisjOwnSide[VAPBS_RIGHT] = 1;
 if (jp == 0) mgparm->partDisjOwnSide[VAPBS_BACK] = 0;
 else mgparm->partDisjOwnSide[VAPBS_BACK] = 1;
 if (jp == (npy-1)) mgparm->partDisjOwnSide[VAPBS_FRONT] = 0;
 else mgparm->partDisjOwnSide[VAPBS_FRONT] = 1;
 if (kp == 0) mgparm->partDisjOwnSide[VAPBS_DOWN] = 0;
 else mgparm->partDisjOwnSide[VAPBS_DOWN] = 1;
 if (kp == (npz-1)) mgparm->partDisjOwnSide[VAPBS_UP] = 0;
 else mgparm->partDisjOwnSide[VAPBS_UP] = 1;

 Vnm_print(0, "NOsh_setupMGPARA:  partDisjOwnSide[LEFT] = %d\n",
     mgparm->partDisjOwnSide[VAPBS_LEFT]);
 Vnm_print(0, "NOsh_setupMGPARA:  partDisjOwnSide[RIGHT] = %d\n",
     mgparm->partDisjOwnSide[VAPBS_RIGHT]);
 Vnm_print(0, "NOsh_setupMGPARA:  partDisjOwnSide[FRONT] = %d\n",
     mgparm->partDisjOwnSide[VAPBS_FRONT]);
 Vnm_print(0, "NOsh_setupMGPARA:  partDisjOwnSide[BACK] = %d\n",
     mgparm->partDisjOwnSide[VAPBS_BACK]);
 Vnm_print(0, "NOsh_setupMGPARA:  partDisjOwnSide[UP] = %d\n",
     mgparm->partDisjOwnSide[VAPBS_UP]);
 Vnm_print(0, "NOsh_setupMGPARA:  partDisjOwnSide[DOWN] = %d\n",
     mgparm->partDisjOwnSide[VAPBS_DOWN]);

 /* Set the mesh parameters */
 mgparm->fglen[0] = xlenOlap;
 mgparm->fglen[1] = ylenOlap;
 mgparm->fglen[2] = zlenOlap;
 mgparm->partDisjLength[0] = xlenDisj;
 mgparm->partDisjLength[1] = ylenDisj;
 mgparm->partDisjLength[2] = zlenDisj;
 mgparm->partDisjCenter[0] = mgparm->fcenter[0] + xcentDisj;
 mgparm->partDisjCenter[1] = mgparm->fcenter[1] + ycentDisj;
 mgparm->partDisjCenter[2] = mgparm->fcenter[2] + zcentDisj;
 mgparm->fcenter[0] += xcentOlap;
 mgparm->fcenter[1] += ycentOlap;
 mgparm->fcenter[2] += zcentOlap;

 Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d):  Set up *relative* partition \
centers...\n", __FILE__, __LINE__);
 Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d):  Absolute centers will be set \
in  NOsh_setupMGAUTO\n", __FILE__, __LINE__);
 Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d):  partDisjCenter = %g %g %g\n",
     __FILE__, __LINE__,
     mgparm->partDisjCenter[0],
     mgparm->partDisjCenter[1],
     mgparm->partDisjCenter[2]);
 Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d):  ccenter = %g %g %g\n",
     __FILE__, __LINE__,
     mgparm->ccenter[0],
     mgparm->ccenter[1],
     mgparm->ccenter[2]);
 Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d):  fcenter = %g %g %g\n",
     __FILE__, __LINE__,
     mgparm->fcenter[0],
     mgparm->fcenter[1],
     mgparm->fcenter[2]);


 /* Setup the automatic focusing calculations associated with this processor */
 return NOsh_setupCalcMGAUTO(thee, elec);

}

VPUBLIC int NOsh_parseFEM(
```

```
        NOsh *thee,
        Vio *sock,
        NOsh_calc *elec
        ) {

 char tok[VMAX_BUFSIZE];
 FEMparm *feparm = VNULL;
 PBEparm *pbeparm = VNULL;
 int rc;
 Vrc_Codes vrc;

 /* Check the arguments */
 if (thee == VNULL) {
  Vnm_print(2, "NOsh_parseFEM:  Got NULL thee!\n");
  return 0;
 }
 if (sock == VNULL) {
  Vnm_print(2, "NOsh_parseFEM:  Got pointer to NULL socket!\n");
  return 0;
 }
 if (elec == VNULL) {
  Vnm_print(2, "NOsh_parseFEM:  Got pointer to NULL elec object!\n");
  return 0;
 }
 feparm = elec->femparm;
 if (feparm == VNULL) {
  Vnm_print(2, "NOsh_parseFEM:  Got pointer to NULL feparm object!\n");
  return 0;
 }
 pbeparm = elec->pbeparm;
 if (feparm == VNULL) {
  Vnm_print(2, "NOsh_parseFEM:  Got pointer to NULL pbeparm object!\n");
  return 0;
 }

 Vnm_print(0, "NOsh_parseFEM: Parsing parameters for FEM calculation\n");

 /* Start snarfing tokens from the input stream */
 rc = 1;
 while (Vio_scanf(sock, "%s", tok) == 1) {

  Vnm_print(0, "NOsh_parseFEM:  Parsing %s...\n", tok);

  /* See if it's an END token */
  if (Vstring_strcasecmp(tok, "end") == 0) {
   feparm->parsed = 1;
   pbeparm->parsed = 1;
   rc = 1;
   break;
  }

  /* Pass the token through a series of parsers */
  rc = PBEparm_parseToken(pbeparm, tok, sock);
  if (rc == -1) {
   Vnm_print(0, "NOsh_parseFEM:  parsePBE error!\n");
   break;
  } else if (rc == 0) {
   /* Pass the token to the generic MG parser */
   vrc = FEMparm_parseToken(feparm, tok, sock);
   if (vrc == VRC_FAILURE) {
    Vnm_print(0, "NOsh_parseFEM:  parseMG error!\n");
    break;
   } else if (vrc == VRC_WARNING) {
    /* We ran out of parsers! */
    Vnm_print(2, "NOsh:  Unrecognized keyword: %s\n", tok);
    break;
   }
  }
 }

 /* Handle various errors arising in the token-snarfing loop -- these all
  * just result in simple returns right now */
 if (rc == -1) return 0;
 if (rc == 0) return 0;

 /* Check the status of the parameter objects */
 if ((!FEMparm_check(feparm)) || (!PBEparm_check(
pbeparm))) {
  Vnm_print(2, "NOsh:  FEM parameters not set correctly!\n");
  return 0;
 }
```

```
 return 1;

}

VPRIVATE int NOsh_setupCalcFEMANUAL(
           NOsh *thee,
           NOsh_calc *elec
           ) {

 FEMparm *feparm = VNULL;
 PBEparm *pbeparm = VNULL;
 NOsh_calc *calc = VNULL;

 VASSERT(thee != VNULL);
 VASSERT(elec != VNULL);
 feparm = elec->femparm;
 VASSERT(feparm != VNULL);
 pbeparm = elec->pbeparm;
 VASSERT(pbeparm);

 /* Check to see if he have any room left for this type of
  * calculation, if so: set the calculation type, update the number
  * of calculations of this type, and parse the rest of the section
  */
 if (thee->ncalc >= NOSH_MAXCALC) {
  Vnm_print(2, "NOsh:  Too many calculations in this run!\n");
  Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
      NOSH_MAXCALC);
  return 0;
 }
 thee->calc[thee->ncalc] = NOsh_calc_ctor(NCT_FEM);
 calc = thee->calc[thee->ncalc];
 (thee->ncalc)++;

 /* Copy over contents of ELEC */
 NOsh_calc_copy(calc, elec);


 return 1;
}

VPUBLIC int NOsh_parseAPOL(
        NOsh *thee,
        Vio *sock,
        NOsh_calc *elec
        ) {

 char tok[VMAX_BUFSIZE];
 APOLparm *apolparm = VNULL;
 int rc;

 /* Check the arguments */
 if (thee == VNULL) {
  Vnm_print(2, "NOsh_parseAPOL:  Got NULL thee!\n");
  return 0;
 }
 if (sock == VNULL) {
  Vnm_print(2, "NOsh_parseAPOL:  Got pointer to NULL socket!\n");
  return 0;
 }
 if (elec == VNULL) {
  Vnm_print(2, "NOsh_parseAPOL:  Got pointer to NULL elec object!\n");
  return 0;
 }
 apolparm = elec->apolparm;
 if (apolparm == VNULL) {
  Vnm_print(2, "NOsh_parseAPOL:  Got pointer to NULL feparm object!\n");
  return 0;
 }

 Vnm_print(0, "NOsh_parseAPOL: Parsing parameters for APOL calculation\n");

 /* Start snarfing tokens from the input stream */
 rc = 1;
 while (Vio_scanf(sock, "%s", tok) == 1) {

  Vnm_print(0, "NOsh_parseAPOL:  Parsing %s...\n", tok);
  /* See if it's an END token */
  if (Vstring_strcasecmp(tok, "end") == 0) {
```

```
    apolparm->parsed = 1;
    rc = 1;
    break;
   }

   /* Pass the token through a series of parsers */
   /* Pass the token to the generic non-polar parser */
   rc = APOLparm_parseToken(apolparm, tok, sock);
   if (rc == -1) {
    Vnm_print(0, "NOsh_parseFEM:  parseMG error!\n");
    break;
   } else if (rc == 0) {
    /* We ran out of parsers! */
    Vnm_print(2, "NOsh:  Unrecognized keyword: %s\n", tok);
    break;
   }

  }

  /* Handle various errors arising in the token-snarfing loop -- these all
   * just result in simple returns right now */
  if (rc == -1) return 0;
  if (rc == 0) return 0;

  /* Check the status of the parameter objects */
  if (!APOLparm_check(apolparm)) {
   Vnm_print(2, "NOsh:  APOL parameters not set correctly!\n");
   return 0;
  }

  return 1;

}

VPRIVATE int NOsh_setupCalcAPOL(
        NOsh *thee,
        NOsh_calc *apol
        ) {

  NOsh_calc *calc = VNULL;

  VASSERT(thee != VNULL);
  VASSERT(apol != VNULL);

  /* Check to see if he have any room left for this type of
   * calculation, if so: set the calculation type, update the number
   * of calculations of this type, and parse the rest of the section
   */
  if (thee->ncalc >= NOSH_MAXCALC) {
   Vnm_print(2, "NOsh:  Too many calculations in this run!\n");
   Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
       NOSH_MAXCALC);
   return 0;
  }
  thee->calc[thee->ncalc] = NOsh_calc_ctor(NCT_APOL
 );
  calc = thee->calc[thee->ncalc];
  (thee->ncalc)++;

  /* Copy over contents of APOL */
  NOsh_calc_copy(calc, apol);

  return 1;
}
```

9.61 src/generic/pbeparm.c File Reference

Class PBEparm methods.

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

Include dependency graph for pbeparm.c:

Functions

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

  Get charge (e) of specified ion species.
- VPUBLIC double PBEparm_getIonConc (PBEparm ∗thee, int i)

  Get concentration (M) of specified ion species.
- VPUBLIC double PBEparm_getIonRadius (PBEparm ∗thee, int i)

  Get radius (A) of specified ion species.
- VPUBLIC double **PBEparm_getzmem** (PBEparm ∗thee)
- VPUBLIC double **PBEparm_getLmem** (PBEparm ∗thee)
- VPUBLIC double **PBEparm_getmembraneDiel** (PBEparm ∗thee)
- VPUBLIC double **PBEparm_getmemv** (PBEparm ∗thee)
- VPUBLIC PBEparm ∗ PBEparm_ctor ()

  Construct PBEparm object.
- VPUBLIC int PBEparm_ctor2 (PBEparm ∗thee)

  FORTRAN stub to construct PBEparm object.
- VPUBLIC void PBEparm_dtor (PBEparm ∗∗thee)

  Object destructor.
- VPUBLIC void PBEparm_dtor2 (PBEparm ∗thee)

  FORTRAN stub for object destructor.
- VPUBLIC int PBEparm_check (PBEparm ∗thee)

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

  Copy PBEparm object into thee.

- VPRIVATE int **PBEparm_parseLPBE** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseNPBE** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseMOL** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseLRPBE** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseNRPBE** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseSMPBE** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseBCFL** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseION** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parsePDIE** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseSDENS** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseSDIE** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseSRFM** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseSRAD** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseSWIN** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseTEMP** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseUSEMAP** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseCALCENERGY** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseCALCFORCE** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseZMEM** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseLMEM** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseMDIE** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseMEMV** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseWRITE** (PBEparm *thee, Vio *sock)
- VPRIVATE int **PBEparm_parseWRITEMAT** (PBEparm *thee, Vio *sock)
- VPUBLIC int PBEparm_parseToken (PBEparm *thee, char tok[VMAX_BUFSIZE], Vio *sock)

  Parse a keyword from an input file.

9.61.1 Detailed Description

Class PBEparm methods.

Author

Nathan Baker

Version

Id:

pbeparm.c 1774 2012-07-19 18:46:14Z sobolevnrm

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

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

VEMBED(rcsid="$Id: pbeparm.c 1774 2012-07-19 18:46:14Z sobolevnrm $")

#if !defined(VINLINE_MGPARM)

#endif /* if !defined(VINLINE_MGPARM) */

VPUBLIC double PBEparm_getIonCharge(PBEparm *thee,
int i) {
    VASSERT(thee != VNULL);
```

```
 VASSERT(i < thee->nion);
    return thee->ionq[i];
}

VPUBLIC double PBEparm_getIonConc(PBEparm *thee, int i
) {
    VASSERT(thee != VNULL);
    VASSERT(i < thee->nion);
    return thee->ionc[i];
}

VPUBLIC double PBEparm_getIonRadius(PBEparm *thee,
int i) {
    VASSERT(thee != VNULL);
    VASSERT(i < thee->nion);
    return thee->ionr[i];
}

/*----------------------------------------------------------------------*/
/* Added by Michael Grabe                                               */
/*----------------------------------------------------------------------*/

VPUBLIC double PBEparm_getzmem(PBEparm *thee) {
    VASSERT(thee != VNULL);
    return thee->zmem;
}
VPUBLIC double PBEparm_getLmem(PBEparm *thee) {
    VASSERT(thee != VNULL);
    return thee->Lmem;
}
VPUBLIC double PBEparm_getmembraneDiel(PBEparm *thee) {
    VASSERT(thee != VNULL);
    return thee->mdie;
}
VPUBLIC double PBEparm_getmemv(PBEparm *thee) {
    VASSERT(thee != VNULL);
    return thee->memv;
}

VPUBLIC PBEparm* PBEparm_ctor() {

    /* Set up the structure */
    PBEparm *thee = VNULL;
    thee = (PBEparm*)Vmem_malloc(VNULL, 1, sizeof(PBEparm));
    VASSERT( thee != VNULL);
    VASSERT( PBEparm_ctor2(thee) );

    return thee;
}

VPUBLIC int PBEparm_ctor2(PBEparm *thee) {

    int i;

    if (thee == VNULL) return 0;

    thee->parsed = 0;

    thee->setmolid = 0;
    thee->setpbetype = 0;
    thee->setbcfl = 0;
    thee->setnion = 0;
    for (i=0; i<MAXION; i++){
  thee->setion[i] = 0;
  thee->ionq[i] = 0.0;
  thee->ionc[i] = 0.0;
  thee->ionr[i] = 0.0;
 }
    thee->setpdie = 0;
    thee->setsdie = 0;
    thee->setsrfm = 0;
    thee->setsrad = 0;
    thee->setswin = 0;
    thee->settemp = 0;
    thee->setcalcenergy = 0;
    thee->setcalcforce = 0;
    thee->setsdens = 0;
    thee->numwrite = 0;
    thee->setwritemat = 0;
    thee->nion = 0;
    thee->sdens = 0;
```

```
    thee->swin = 0;
    thee->srad = 1.4;
    thee->useDielMap = 0;
    thee->useKappaMap = 0;
 thee->usePotMap = 0;
    thee->useChargeMap = 0;

 /*----------------------------------------------*/
 /* Added by Michael Grabe                       */
 /*----------------------------------------------*/

    thee->setzmem = 0;
    thee->setLmem = 0;
    thee->setmdie = 0;
    thee->setmemv = 0;

 /*----------------------------------------------*/

 thee->smsize = 0.0;
 thee->smvolume = 0.0;

 thee->setsmsize = 0;
 thee->setsmvolume = 0;

    return 1;
}

VPUBLIC void PBEparm_dtor(PBEparm **thee) {
    if ((*thee) != VNULL) {
        PBEparm_dtor2(*thee);
        Vmem_free(VNULL, 1, sizeof(PBEparm), (void **)thee);
        (*thee) = VNULL;
    }
}

VPUBLIC void PBEparm_dtor2(PBEparm *thee) { ; }

VPUBLIC int PBEparm_check(PBEparm *thee) {

    int i;

    /* Check to see if we were even filled... */
    if (!thee->parsed) {
        Vnm_print(2, "PBEparm_check:  not filled!\n");
        return 0;
    }

    if (!thee->setmolid) {
        Vnm_print(2, "PBEparm_check:  MOL not set!\n");
        return 0;
    }
    if (!thee->setpbetype) {
        Vnm_print(2, "PBEparm_check:  LPBE/NPBE/LRPBE/NRPBE/SMPBE not set!\n");
        return 0;
    }
    if (!thee->setbcfl) {
        Vnm_print(2, "PBEparm_check:  BCFL not set!\n");
        return 0;
    }
    if (!thee->setnion) {
        thee->setnion = 1;
        thee->nion = 0;
    }
    for (i=0; i<thee->nion; i++) {
        if (!thee->setion[i]) {
            Vnm_print(2, "PBEparm_check: ION #%d not set!\n",i);
            return 0;
        }
    }
    if (!thee->setpdie) {
        Vnm_print(2, "PBEparm_check: PDIE not set!\n");
        return 0;
    }
    if (((thee->srfm==VSM_MOL) || (thee->srfm==VSM_MOLSMOOTH
    )) \
      && (!thee->setsdens) && (thee->srad > VSMALL)) {
        Vnm_print(2, "PBEparm_check: SDENS not set!\n");
        return 0;
    }
    if (!thee->setsdie) {
        Vnm_print(2, "PBEparm_check: SDIE not set!\n");
```

```
        return 0;
    }
    if (!thee->setsrfm) {
        Vnm_print(2, "PBEparm_check: SRFM not set!\n");
        return 0;
    }
    if (((thee->srfm==VSM_MOL) || (thee->srfm==VSM_MOLSMOOTH)) \
      && (!thee->setsrad)) {
        Vnm_print(2, "PBEparm_check: SRAD not set!\n");
        return 0;
    }
    if ((thee->srfm==VSM_SPLINE) && (!thee->setswin)) {
        Vnm_print(2, "PBEparm_check: SWIN not set!\n");
        return 0;
    }
    if ((thee->srfm==VSM_SPLINE3) && (!thee->setswin)) {
        Vnm_print(2, "PBEparm_check: SWIN not set!\n");
        return 0;
    }
    if ((thee->srfm==VSM_SPLINE4) && (!thee->setswin)) {
        Vnm_print(2, "PBEparm_check: SWIN not set!\n");
        return 0;
    }
    if (!thee->settemp) {
        Vnm_print(2, "PBEparm_check: TEMP not set!\n");
        return 0;
    }
    if (!thee->setcalcenergy) thee->calcenergy = PCE_NO;
    if (!thee->setcalcforce) thee->calcforce = PCF_NO;
    if (!thee->setwritemat) thee->writemat = 0;

 /*-----------------------------------------------------------*/
 /* Added by Michael Grabe                                    */
 /*-----------------------------------------------------------*/

    if ((!thee->setzmem) && (thee->bcfl == 3)){
        Vnm_print(2, "PBEparm_check: ZMEM not set!\n");
        return 0;
    }
    if ((!thee->setLmem) && (thee->bcfl == 3)){
        Vnm_print(2, "PBEparm_check: LMEM not set!\n");
        return 0;
    }
    if ((!thee->setmdie) && (thee->bcfl == 3)){
        Vnm_print(2, "PBEparm_check: MDIE not set!\n");
        return 0;
    }
    if ((!thee->setmemv) && (thee->bcfl == 3)){
        Vnm_print(2, "PBEparm_check: MEMV not set!\n");
        return 0;
    }

 /*-----------------------------------------------------------*/

    return 1;
}

VPUBLIC void PBEparm_copy(PBEparm *thee, PBEparm *parm) {

    int i, j;

    VASSERT(thee != VNULL);
    VASSERT(parm != VNULL);

    thee->molid = parm->molid;
    thee->setmolid = parm->setmolid;
    thee->useDielMap = parm->useDielMap;
    thee->dielMapID = parm->dielMapID;
    thee->useKappaMap = parm->useKappaMap;
    thee->kappaMapID = parm->kappaMapID;
 thee->usePotMap = parm->usePotMap;
 thee->potMapID = parm->potMapID;
    thee->useChargeMap = parm->useChargeMap;
    thee->chargeMapID = parm->chargeMapID;
    thee->pbetype = parm->pbetype;
    thee->setpbetype = parm->setpbetype;
    thee->bcfl = parm->bcfl;
```

```
    thee->setbcfl = parm->setbcfl;
    thee->nion = parm->nion;
    thee->setnion = parm->setnion;
    for (i=0; i<MAXION; i++) {
        thee->ionq[i] = parm->ionq[i];
        thee->ionc[i] = parm->ionc[i];
        thee->ionr[i] = parm->ionr[i];
        thee->setion[i] = parm->setion[i];
    };
    thee->pdie = parm->pdie;
    thee->setpdie = parm->setpdie;
    thee->sdens = parm->sdens;
    thee->setsdens = parm->setsdens;
    thee->sdie = parm->sdie;
    thee->setsdie = parm->setsdie;
    thee->srfm = parm->srfm;
    thee->setsrfm = parm->setsrfm;
    thee->srad = parm->srad;
    thee->setsrad = parm->setsrad;
    thee->swin = parm->swin;
    thee->setswin = parm->setswin;
    thee->temp = parm->temp;
    thee->settemp = parm->settemp;
    thee->calcenergy = parm->calcenergy;
    thee->setcalcenergy = parm->setcalcenergy;
    thee->calcforce = parm->calcforce;
    thee->setcalcforce = parm->setcalcforce;

 /*-----------------------------------------------------*/
 /* Added by Michael Grabe                              */
 /*-----------------------------------------------------*/

    thee->zmem = parm->zmem;
    thee->setzmem = parm->setzmem;
    thee->Lmem = parm->Lmem;
    thee->setLmem = parm->setLmem;
    thee->mdie = parm->mdie;
    thee->setmdie = parm->setmdie;
    thee->memv = parm->memv;
    thee->setmemv = parm->setmemv;

 /*-----------------------------------------------------*/

    thee->numwrite = parm->numwrite;
    for (i=0; i<PBEPARM_MAXWRITE; i++) {
        thee->writetype[i] = parm->writetype[i];
        thee->writefmt[i] = parm->writefmt[i];
        for (j=0; j<VMAX_ARGLEN; j++)
          thee->writestem[i][j] = parm->writestem[i][j];
    }
    thee->writemat = parm->writemat;
    thee->setwritemat = parm->setwritemat;
    for (i=0; i<VMAX_ARGLEN; i++) thee->writematstem[i] = parm->
writematstem[i];
    thee->writematflag = parm->writematflag;

 thee->smsize = parm->smsize;
 thee->smvolume = parm->smvolume;

 thee->setsmsize = parm->setsmsize;
 thee->setsmvolume = parm->setsmvolume;

    thee->parsed = parm->parsed;

}

VPRIVATE int PBEparm_parseLPBE(PBEparm *thee, Vio *sock) {
    Vnm_print(0, "NOsh: parsed lpbe\n");
    thee->pbetype = PBE_LPBE;
    thee->setpbetype = 1;
    return 1;
}

VPRIVATE int PBEparm_parseNPBE(PBEparm *thee, Vio *sock) {
    Vnm_print(0, "NOsh: parsed npbe\n");
    thee->pbetype = PBE_NPBE;
    thee->setpbetype = 1;
    return 1;
}

VPRIVATE int PBEparm_parseMOL(PBEparm *thee, Vio *sock) {
```

```
    int ti;
    char tok[VMAX_BUFSIZE];

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%d", &ti) == 0) {
        Vnm_print(2, "NOsh:  Read non-int (%s) while parsing MOL \
keyword!\n", tok);
        return -1;
    }
    thee->molid = ti;
    thee->setmolid = 1;
    return 1;

    VERROR1:
        Vnm_print(2, "parsePBE:  ran out of tokens!\n");
        return -1;
}

VPRIVATE int PBEparm_parseLRPBE(PBEparm *thee, Vio *sock) {
    Vnm_print(0, "NOsh: parsed lrpbe\n");
    thee->pbetype = PBE_LRPBE;
    thee->setpbetype = 1;
    return 1;
}

VPRIVATE int PBEparm_parseNRPBE(PBEparm *thee, Vio *sock) {
    Vnm_print(0, "NOsh: parsed nrpbe\n");
    thee->pbetype = PBE_NRPBE;
    thee->setpbetype = 1;
    return 1;
}

VPRIVATE int PBEparm_parseSMPBE(PBEparm *thee, Vio *sock) {

 int i;

 char type[VMAX_BUFSIZE]; /* vol or size (keywords) */
 char value[VMAX_BUFSIZE]; /* floating point value */

 char setVol = 1;
 char setSize = 1;
 char keyValuePairs = 2;

    double size, volume;

 for(i=0;i<keyValuePairs;i++){

  /* The line two tokens at a time */
  VJMPERR1(Vio_scanf(sock, "%s", type) == 1);
  VJMPERR1(Vio_scanf(sock, "%s", value) == 1);

  if(!strcmp(type,"vol")){
   if ((setVol = sscanf(value, "%lf", &volume)) == 0){
    Vnm_print(2,"NOsh:  Read non-float (%s) while parsing smpbe keyword!\n",
value);
    return VRC_FAILURE;
   }
  }else if(!strcmp(type,"size")){
   if ((setSize = sscanf(value, "%lf", &size)) == 0){
    Vnm_print(2,"NOsh:  Read non-float (%s) while parsing smpbe keyword!\n",
value);
    return VRC_FAILURE;
   }
  }else{
   Vnm_print(2,"NOsh:  Read non-float (%s) while parsing smpbe keyword!\n",
value);
   return VRC_FAILURE;
  }
 }

 /* If either the volume or size isn't set, throw an error */
 if((setVol == 0) || (setSize == 0)){
  Vnm_print(2,"NOsh:  Error while parsing smpbe keywords! Only size or vol was
specified.\n");
  return VRC_FAILURE;
 }

 Vnm_print(0, "NOsh: parsed smpbe\n");
    thee->pbetype = PBE_SMPBE;
    thee->setpbetype = 1;

```

```
 thee->smsize = size;
 thee->setsmsize = 1;

 thee->smvolume = volume;
 thee->setsmvolume = 1;

 return VRC_SUCCESS;

VERROR1:
    Vnm_print(2, "parsePBE:  ran out of tokens!\n");
 return VRC_FAILURE;

}

VPRIVATE int PBEparm_parseBCFL(PBEparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    int ti;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);

    /* We can either parse int flag... */
    if (sscanf(tok, "%d", &ti) == 1) {

        thee->bcfl = (Vbcfl)ti;
        thee->setbcfl = 1;
        /* Warn that this usage is deprecated */
        Vnm_print(2, "parsePBE:  Warning -- parsed deprecated \"bcfl %d\" \
statement\n", ti);
        Vnm_print(2, "parsePBE:  Please use \"bcfl ");
        switch (thee->bcfl) {
            case BCFL_ZERO:
                Vnm_print(2, "zero");
                break;
            case BCFL_SDH:
                Vnm_print(2, "sdh");
                break;
            case BCFL_MDH:
                Vnm_print(2, "mdh");
                break;
            case BCFL_FOCUS:
                Vnm_print(2, "focus");
                break;
   case BCFL_MEM:
                Vnm_print(2, "mem");
                break;
   case BCFL_MAP:
                Vnm_print(2, "map");
                break;
            default:
                Vnm_print(2, "UKNOWN");
                break;
        }
        Vnm_print(2, "\" instead.\n");
        return 1;

    /* ...or the word */
    } else {

        if (Vstring_strcasecmp(tok, "zero") == 0) {
            thee->bcfl = BCFL_ZERO;
            thee->setbcfl = 1;
            return 1;
        } else if (Vstring_strcasecmp(tok, "sdh") == 0) {
            thee->bcfl = BCFL_SDH;
            thee->setbcfl = 1;
            return 1;
        } else if (Vstring_strcasecmp(tok, "mdh") == 0) {
            thee->bcfl = BCFL_MDH;
            thee->setbcfl = 1;
            return 1;
        } else if (Vstring_strcasecmp(tok, "focus") == 0) {
            thee->bcfl = BCFL_FOCUS;
            thee->setbcfl = 1;
            return 1;
   } else if (Vstring_strcasecmp(tok, "mem") == 0) {
            thee->bcfl = BCFL_MEM;
            thee->setbcfl = 1;
            return 1;
        } else if (Vstring_strcasecmp(tok, "map") == 0) {
            thee->bcfl = BCFL_MAP;
            thee->setbcfl = 1;
```

```
            return 1;
        } else {
            Vnm_print(2, "NOsh:  parsed unknown BCFL parameter (%s)!\n",
              tok);
            return -1;
        }
    }
    return 0;

    VERROR1:
        Vnm_print(2, "parsePBE:  ran out of tokens!\n");
        return -1;
}

VPRIVATE int PBEparm_parseION(PBEparm *thee, Vio *sock) {

 int i;
 int meth = 0;

 char tok[VMAX_BUFSIZE];
 char value[VMAX_BUFSIZE];

 double tf;
 double charge, conc, radius;

 int setCharge = 0;
 int setConc = 0;
 int setRadius = 0;
 int keyValuePairs = 3;

 /* Get the initial token for the ION statement */
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);

 /* Scan the token once to determine the type (old style or new keyValue pair) */
 meth = sscanf(tok, "%lf", &tf);
 /* If tok is a non-zero float value, we are using the old method */
 if(meth != 0){

  Vnm_print(2, "NOsh:  Deprecated use of ION keyword! Use key-value pairs\n", tok);

  if (sscanf(tok, "%lf", &tf) == 0) {
   Vnm_print(2, "NOsh:  Read non-float (%s) while parsing ION keyword!\n", tok);
   return VRC_FAILURE;
  }
  thee->ionq[thee->nion] = tf;
  VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
  if (sscanf(tok, "%lf", &tf) == 0) {
   Vnm_print(2, "NOsh:  Read non-float (%s) while parsing ION keyword!\n", tok);
   return VRC_FAILURE;
  }
  thee->ionc[thee->nion] = tf;
  VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
  if (sscanf(tok, "%lf", &tf) == 0) {
   Vnm_print(2, "NOsh:  Read non-float (%s) while parsing ION keyword!\n", tok);
   return VRC_FAILURE;
  }
  thee->ionr[thee->nion] = tf;

 }else{

  /* Three key-value pairs (charge, radius and conc) */
  for(i=0;i<keyValuePairs;i++){

   /* Now scan for the value (float) to be used with the key token parsed
    * above the if-else statement */
   VJMPERR1(Vio_scanf(sock, "%s", value) == 1);
   if(!strcmp(tok,"charge")){
    setCharge = sscanf(value, "%lf", &charge);
    if (setCharge == 0){
     Vnm_print(2,"NOsh:  Read non-float (%s) while parsing ION %s keyword!\n", value, tok);
     return VRC_FAILURE;
    }
    thee->ionq[thee->nion] = charge;
   }else if(!strcmp(tok,"radius")){
    setRadius = sscanf(value, "%lf", &radius);
```

```
    if (setRadius == 0){
     Vnm_print(2,"NOsh:  Read non-float (%s) while parsing ION %s keyword!\n", value, tok);
     return VRC_FAILURE;
    }
    thee->ionr[thee->nion] = radius;
   }else if(!strcmp(tok,"conc")){
    setConc = sscanf(value, "%lf", &conc);
    if (setConc == 0){
     Vnm_print(2,"NOsh:  Read non-float (%s) while parsing ION %s keyword!\n", value, tok);
     return VRC_FAILURE;
    }
    thee->ionc[thee->nion] = conc;
   }else{
    Vnm_print(2,"NOsh:  Illegal or missing key-value pair for ION keyword!\n");
    return VRC_FAILURE;
   }

   /* If all three values haven't be set (setValue = 0) then read the next token */
   if((setCharge != 1) || (setConc != 1) || (setRadius != 1)){
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
   }

  } /* end for */
 } /* end if */

 /* Finally set the setion, nion and setnion flags and return success */
 thee->setion[thee->nion] = 1;
 (thee->nion)++;
 thee->setnion = 1;
 return VRC_SUCCESS;

    VERROR1:
        Vnm_print(2, "parsePBE:  ran out of tokens!\n");
        return VRC_FAILURE;
}

VPRIVATE int PBEparm_parsePDIE(PBEparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing PDIE \keyword!\n", tok);
        return -1;
    }
    thee->pdie = tf;
    thee->setpdie = 1;
    return 1;

    VERROR1:
        Vnm_print(2, "parsePBE:  ran out of tokens!\n");
        return -1;
}

VPRIVATE int PBEparm_parseSDENS(PBEparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SDENS \keyword!\n", tok);
        return -1;
    }
    thee->sdens = tf;
    thee->setsdens = 1;
    return 1;

    VERROR1:
        Vnm_print(2, "parsePBE:  ran out of tokens!\n");
        return -1;
}

VPRIVATE int PBEparm_parseSDIE(PBEparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    double tf;

```

```
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SDIE \
keyword!\n", tok);
        return -1;
    }
    thee->sdie = tf;
    thee->setsdie = 1;
    return 1;

    VERROR1:
        Vnm_print(2, "parsePBE:  ran out of tokens!\n");
        return -1;
}

VPRIVATE int PBEparm_parseSRFM(PBEparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    int ti;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);

    /* Parse old-style int arg */
    if (sscanf(tok, "%d", &ti) == 1) {
        thee->srfm = (Vsurf_Meth)ti;
        thee->setsrfm = 1;

        Vnm_print(2, "parsePBE:  Warning -- parsed deprecated \"srfm %d\" \
statement.\n", ti);
        Vnm_print(2, "parsePBE:  Please use \"srfm ");
        switch (thee->srfm) {
            case VSM_MOL:
                Vnm_print(2, "mol");
                break;
            case VSM_MOLSMOOTH:
                Vnm_print(2, "smol");
                break;
            case VSM_SPLINE:
                Vnm_print(2, "spl2");
                break;
            case VSM_SPLINE3:
                Vnm_print(2, "spl3");
                break;
            case VSM_SPLINE4:
                Vnm_print(2, "spl4");
                break;
            default:
                Vnm_print(2, "UNKNOWN");
                break;
        }
        Vnm_print(2, "\" instead.\n");
        return 1;

    /* Parse newer text-based args */
    } else if (Vstring_strcasecmp(tok, "mol") == 0) {
        thee->srfm = VSM_MOL;
        thee->setsrfm = 1;
        return 1;
    } else if (Vstring_strcasecmp(tok, "smol") == 0) {
        thee->srfm = VSM_MOLSMOOTH;
        thee->setsrfm = 1;
        return 1;
    } else if (Vstring_strcasecmp(tok, "spl2") == 0) {
        thee->srfm = VSM_SPLINE;
        thee->setsrfm = 1;
        return 1;
    } else if (Vstring_strcasecmp(tok, "spl3") == 0) {
        thee->srfm = VSM_SPLINE3;
        thee->setsrfm = 1;
        return 1;
 } else if (Vstring_strcasecmp(tok, "spl4") == 0) {
        thee->srfm = VSM_SPLINE4;
        thee->setsrfm = 1;
        return 1;
    } else {
        Vnm_print(2, "NOsh:  Unrecongnized keyword (%s) when parsing \
srfm!\n", tok);
        return -1;
    }

    return 0;

```

```
    VERROR1:
        Vnm_print(2, "parsePBE:  ran out of tokens!\n");
        return -1;
}

VPRIVATE int PBEparm_parseSRAD(PBEparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SRAD \
keyword!\n", tok);
        return -1;
    }
    thee->srad = tf;
    thee->setsrad = 1;
    return 1;

    VERROR1:
        Vnm_print(2, "parsePBE:  ran out of tokens!\n");
        return -1;
}

VPRIVATE int PBEparm_parseSWIN(PBEparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SWIN \
keyword!\n", tok);
        return -1;
    }
    thee->swin = tf;
    thee->setswin = 1;
    return 1;

    VERROR1:
        Vnm_print(2, "parsePBE:  ran out of tokens!\n");
        return -1;
}

VPRIVATE int PBEparm_parseTEMP(PBEparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing TEMP \
keyword!\n", tok);
        return -1;
    }
    thee->temp = tf;
    thee->settemp = 1;
    return 1;

    VERROR1:
        Vnm_print(2, "parsePBE:  ran out of tokens!\n");
        return -1;
}

VPRIVATE int PBEparm_parseUSEMAP(PBEparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    int ti;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    Vnm_print(0, "PBEparm_parseToken:  Read %s...\n", tok);
    if (Vstring_strcasecmp(tok, "diel") == 0) {
        thee->useDielMap = 1;
        VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
        if (sscanf(tok, "%d", &ti) == 0) {
            Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
USEMAP DIEL keyword!\n", tok);
            return -1;
        }
        thee->dielMapID = ti;
        return 1;
    } else if (Vstring_strcasecmp(tok, "kappa") == 0) {
        thee->useKappaMap = 1;
        VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
```

```
        if (sscanf(tok, "%d", &ti) == 0) {
            Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
USEMAP KAPPA keyword!\n", tok);
            return -1;
        }
        thee->kappaMapID = ti;
        return 1;
    } else if (Vstring_strcasecmp(tok, "pot") == 0) {
        thee->usePotMap = 1;
        VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
        if (sscanf(tok, "%d", &ti) == 0) {
            Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
       USEMAP POT keyword!\n", tok);
            return -1;
        }
        thee->potMapID = ti;
        return 1;
    } else if (Vstring_strcasecmp(tok, "charge") == 0) {
        thee->useChargeMap = 1;
        VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
        if (sscanf(tok, "%d", &ti) == 0) {
            Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
USEMAP CHARGE keyword!\n", tok);
            return -1;
        }
        thee->chargeMapID = ti;
        return 1;
    } else {
        Vnm_print(2, "NOsh:  Read undefined keyword (%s) while parsing \
USEMAP statement!\n", tok);
        return -1;
    }
    return 0;

    VERROR1:
        Vnm_print(2, "parsePBE:  ran out of tokens!\n");
        return -1;
}

VPRIVATE int PBEparm_parseCALCENERGY(PBEparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    int ti;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    /* Parse number */
    if (sscanf(tok, "%d", &ti) == 1) {
        thee->calcenergy = (PBEparm_calcEnergy)ti;
        thee->setcalcenergy = 1;

        Vnm_print(2, "parsePBE:  Warning -- parsed deprecated \"calcenergy \
%d\" statement.\n", ti);
        Vnm_print(2, "parsePBE:  Please use \"calcenergy ");
        switch (thee->calcenergy) {
            case PCE_NO:
                Vnm_print(2, "no");
                break;
            case PCE_TOTAL:
                Vnm_print(2, "total");
                break;
            case PCE_COMPS:
                Vnm_print(2, "comps");
                break;
            default:
                Vnm_print(2, "UNKNOWN");
                break;
        }
        Vnm_print(2, "\" instead.\n");
        return 1;
    } else if (Vstring_strcasecmp(tok, "no") == 0) {
        thee->calcenergy = PCE_NO;
        thee->setcalcenergy = 1;
        return 1;
    } else if (Vstring_strcasecmp(tok, "total") == 0) {
        thee->calcenergy = PCE_TOTAL;
        thee->setcalcenergy = 1;
        return 1;
    } else if (Vstring_strcasecmp(tok, "comps") == 0) {
        thee->calcenergy = PCE_COMPS;
        thee->setcalcenergy = 1;
        return 1;
    } else {
```

```
        Vnm_print(2, "NOsh:  Unrecognized parameter (%s) while parsing \
calcenergy!\n", tok);
        return -1;
    }
    return 0;

    VERROR1:
        Vnm_print(2, "parsePBE:  ran out of tokens!\n");
        return -1;
}

VPRIVATE int PBEparm_parseCALCFORCE(PBEparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    int ti;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    /* Parse number */
    if (sscanf(tok, "%d", &ti) == 1) {
        thee->calcforce = (PBEparm_calcForce)ti;
        thee->setcalcforce = 1;

        Vnm_print(2, "parsePBE:  Warning -- parsed deprecated \"calcforce \
%d\" statement.\n", ti);
        Vnm_print(2, "parsePBE:  Please use \"calcforce ");
        switch (thee->calcenergy) {
            case PCF_NO:
                Vnm_print(2, "no");
                break;
            case PCF_TOTAL:
                Vnm_print(2, "total");
                break;
            case PCF_COMPS:
                Vnm_print(2, "comps");
                break;
            default:
                Vnm_print(2, "UNKNOWN");
                break;
        }
        Vnm_print(2, "\" instead.\n");
        return 1;
    } else if (Vstring_strcasecmp(tok, "no") == 0) {
        thee->calcforce = PCF_NO;
        thee->setcalcforce = 1;
        return 1;
    } else if (Vstring_strcasecmp(tok, "total") == 0) {
        thee->calcforce = PCF_TOTAL;
        thee->setcalcforce = 1;
        return 1;
    } else if (Vstring_strcasecmp(tok, "comps") == 0) {
        thee->calcforce = PCF_COMPS;
        thee->setcalcforce = 1;
        return 1;
    } else {
        Vnm_print(2, "NOsh:  Unrecognized parameter (%s) while parsing \
calcforce!\n", tok);
        return -1;
    }
    return 0;

    VERROR1:
        Vnm_print(2, "parsePBE:  ran out of tokens!\n");
        return -1;
}

/*-----------------------------------------------------------*/
/* Added by Michael Grabe                                    */
/*-----------------------------------------------------------*/

VPRIVATE int PBEparm_parseZMEM(PBEparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing ZMEM \
      keyword!\n", tok);
        return -1;
    }
    thee->zmem = tf;
    thee->setzmem = 1;
    return 1;
```

```

VERROR1:
 Vnm_print(2, "parsePBE:  ran out of tokens!\n");
 return -1;
}


VPRIVATE int PBEparm_parseLMEM(PBEparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing LMEM \
      keyword!\n", tok);
        return -1;
    }
    thee->Lmem = tf;
    thee->setLmem = 1;
    return 1;

VERROR1:
 Vnm_print(2, "parsePBE:  ran out of tokens!\n");
 return -1;
}

VPRIVATE int PBEparm_parseMDIE(PBEparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing MDIE \
      keyword!\n", tok);
        return -1;
    }
    thee->mdie = tf;
    thee->setmdie  = 1;
    return 1;

VERROR1:
 Vnm_print(2, "parsePBE:  ran out of tokens!\n");
 return -1;
}

VPRIVATE int PBEparm_parseMEMV(PBEparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE];
    double tf;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &tf) == 0) {
        Vnm_print(2, "NOsh:  Read non-float (%s) while parsing MEMV \
      keyword!\n", tok);
        return -1;
    }
    thee->memv = tf;
    thee->setmemv = 1;
    return 1;

VERROR1:
 Vnm_print(2, "parsePBE:  ran out of tokens!\n");
 return -1;
}

/*-------------------------------------------------------------*/

VPRIVATE int PBEparm_parseWRITE(PBEparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
    Vdata_Type writetype;
    Vdata_Format writefmt;

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (Vstring_strcasecmp(tok, "pot") == 0) {
        writetype = VDT_POT;
    } else if (Vstring_strcasecmp(tok, "atompot") == 0) {
        writetype = VDT_ATOMPOT;
    }  else if (Vstring_strcasecmp(tok, "charge") == 0) {
        writetype = VDT_CHARGE;
    } else if (Vstring_strcasecmp(tok, "smol") == 0) {
        writetype = VDT_SMOL;
    } else if (Vstring_strcasecmp(tok, "dielx") == 0) {
```

```
        writetype = VDT_DIELX;
    } else if (Vstring_strcasecmp(tok, "diely") == 0) {
        writetype = VDT_DIELY;
    } else if (Vstring_strcasecmp(tok, "dielz") == 0) {
        writetype = VDT_DIELZ;
    } else if (Vstring_strcasecmp(tok, "kappa") == 0) {
        writetype = VDT_KAPPA;
    } else if (Vstring_strcasecmp(tok, "sspl") == 0) {
        writetype = VDT_SSPL;
    } else if (Vstring_strcasecmp(tok, "vdw") == 0) {
        writetype = VDT_VDW;
    } else if (Vstring_strcasecmp(tok, "ivdw") == 0) {
        writetype = VDT_IVDW;
    } else if (Vstring_strcasecmp(tok, "lap") == 0) {
        writetype = VDT_LAP;
    } else if (Vstring_strcasecmp(tok, "edens") == 0) {
        writetype = VDT_EDENS;
    } else if (Vstring_strcasecmp(tok, "ndens") == 0) {
        writetype = VDT_NDENS;
    } else if (Vstring_strcasecmp(tok, "qdens") == 0) {
        writetype = VDT_QDENS;
    } else {
        Vnm_print(2, "PBEparm_parse:  Invalid data type (%s) to write!\n",
           tok);
        return -1;
    }
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (Vstring_strcasecmp(tok, "dx") == 0) {
        writefmt = VDF_DX;
    } else if (Vstring_strcasecmp(tok, "uhbd") == 0) {
        writefmt = VDF_UHBD;
    } else if (Vstring_strcasecmp(tok, "avs") == 0) {
        writefmt = VDF_AVS;
    } else if (Vstring_strcasecmp(tok, "gz") == 0) {
        writefmt = VDF_GZ;
    } else if (Vstring_strcasecmp(tok, "flat") == 0) {
        writefmt = VDF_FLAT;
    } else {
        Vnm_print(2, "PBEparm_parse:  Invalid data format (%s) to write!\n",
           tok);
        return -1;
    }
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (tok[0]=='"') {
        strcpy(strnew, "");
        while (tok[strlen(tok)-1] != '"') {
            strcat(str, tok);
            strcat(str, " ");
            VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
        }
        strcat(str, tok);
        strncpy(strnew, str+1, strlen(str)-2);
        strcpy(tok, strnew);
    }
 if (thee->numwrite < (PBEPARM_MAXWRITE-1)) {
  strncpy(thee->writestem[thee->numwrite], tok, VMAX_ARGLEN);
  thee->writetype[thee->numwrite] = writetype;
  thee->writefmt[thee->numwrite] = writefmt;
  (thee->numwrite)++;
 } else {
  Vnm_print(2, "PBEparm_parse:  You have exceeded the maximum number of write
statements!\n");
  Vnm_print(2, "PBEparm_parse:  Ignoring additional write statements!\n");
 }
    return 1;

    VERROR1:
        Vnm_print(2, "parsePBE:  ran out of tokens!\n");
        return -1;
}

VPRIVATE int PBEparm_parseWRITEMAT(PBEparm *thee, Vio *sock) {
    char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (Vstring_strcasecmp(tok, "poisson") == 0) {
        thee->writematflag = 0;
    } else if (Vstring_strcasecmp(tok, "full") == 0) {
        thee->writematflag = 1;
    } else {
        Vnm_print(2, "NOsh:  Invalid format (%s) while parsing \
```

```
WRITEMAT keyword!\n", tok);
        return -1;
    }

    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (tok[0]=='"') {
        strcpy(strnew, "");
        while (tok[strlen(tok)-1] != '"') {
            strcat(str, tok);
            strcat(str, " ");
            VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
        }
        strcat(str, tok);
        strncpy(strnew, str+1, strlen(str)-2);
        strcpy(tok, strnew);
    }
    strncpy(thee->writematstem, tok, VMAX_ARGLEN);
    thee->setwritemat = 1;
    thee->writemat = 1;
    return 1;

    VERROR1:
        Vnm_print(2, "parsePBE:  ran out of tokens!\n");
        return -1;

}

VPUBLIC int PBEparm_parseToken(PBEparm *thee, char tok
    [VMAX_BUFSIZE],
  Vio *sock) {

    if (thee == VNULL) {
        Vnm_print(2, "parsePBE:  got NULL thee!\n");
        return -1;
    }
    if (sock == VNULL) {
        Vnm_print(2, "parsePBE:  got NULL socket!\n");
        return -1;
    }

    Vnm_print(0, "PBEparm_parseToken:  trying %s...\n", tok);

    if (Vstring_strcasecmp(tok, "mol") == 0) {
        return PBEparm_parseMOL(thee, sock);
    } else if (Vstring_strcasecmp(tok, "lpbe") == 0) {
        return PBEparm_parseLPBE(thee, sock);
    } else if (Vstring_strcasecmp(tok, "npbe") == 0) {
        return PBEparm_parseNPBE(thee, sock);
    } else if (Vstring_strcasecmp(tok, "lrpbe") == 0) {
        return PBEparm_parseLRPBE(thee, sock);
    } else if (Vstring_strcasecmp(tok, "nrpbe") == 0) {
        return PBEparm_parseNRPBE(thee, sock);
    } else if (Vstring_strcasecmp(tok, "smpbe") == 0) {
        return PBEparm_parseSMPBE(thee, sock);
    } else if (Vstring_strcasecmp(tok, "bcfl") == 0) {
        return PBEparm_parseBCFL(thee, sock);
    } else if (Vstring_strcasecmp(tok, "ion") == 0) {
        return PBEparm_parseION(thee, sock);
    } else if (Vstring_strcasecmp(tok, "pdie") == 0) {
        return PBEparm_parsePDIE(thee, sock);
    } else if (Vstring_strcasecmp(tok, "sdens") == 0) {
        return PBEparm_parseSDENS(thee, sock);
    } else if (Vstring_strcasecmp(tok, "sdie") == 0) {
        return PBEparm_parseSDIE(thee, sock);
    } else if (Vstring_strcasecmp(tok, "srfm") == 0) {
        return PBEparm_parseSRFM(thee, sock);
    } else if (Vstring_strcasecmp(tok, "srad") == 0) {
        return PBEparm_parseSRAD(thee, sock);
    } else if (Vstring_strcasecmp(tok, "swin") == 0) {
        return PBEparm_parseSWIN(thee, sock);
    } else if (Vstring_strcasecmp(tok, "temp") == 0) {
        return PBEparm_parseTEMP(thee, sock);
    } else if (Vstring_strcasecmp(tok, "usemap") == 0) {
        return PBEparm_parseUSEMAP(thee, sock);
    } else if (Vstring_strcasecmp(tok, "calcenergy") == 0) {
        return PBEparm_parseCALCENERGY(thee, sock);
    } else if (Vstring_strcasecmp(tok, "calcforce") == 0) {
        return PBEparm_parseCALCFORCE(thee, sock);
    } else if (Vstring_strcasecmp(tok, "write") == 0) {
        return PBEparm_parseWRITE(thee, sock);
    } else if (Vstring_strcasecmp(tok, "writemat") == 0) {
```

```
        return PBEparm_parseWRITEMAT(thee, sock);

  /*-------------------------------------------------------*/
  /* Added by Michael Grabe                                */
  /*-------------------------------------------------------*/

    } else if (Vstring_strcasecmp(tok, "zmem") == 0) {
        return PBEparm_parseZMEM(thee, sock);
    } else if (Vstring_strcasecmp(tok, "Lmem") == 0) {
        return PBEparm_parseLMEM(thee, sock);
    } else if (Vstring_strcasecmp(tok, "mdie") == 0) {
        return PBEparm_parseMDIE(thee, sock);
    } else if (Vstring_strcasecmp(tok, "memv") == 0) {
        return PBEparm_parseMEMV(thee, sock);
    }

  /*-------------------------------------------------------*/

    return 0;

}
```

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 derivatve of solvent accessible volume.
- VPRIVATE double **Vacc_SASAPos** (Vacc *thee, double radius)
- VPRIVATE double **Vacc_atomSASAPos** (Vacc *thee, double radius, Vatom *atom, int mode)
- VPUBLIC void Vacc_atomdSASA (Vacc *thee, double dpos, double srad, Vatom *atom, double *dSA)

  Get the derivatve of solvent accessible area.
- VPUBLIC void Vacc_totalAtomdSASA (Vacc *thee, double dpos, double srad, Vatom *atom, double *dSA)

  Testing purposes only.
- VPUBLIC void Vacc_totalAtomdSAV (Vacc *thee, double dpos, double srad, Vatom *atom, double *dSA, Vclist *clist)

  Total solvent accessible volume.
- VPUBLIC double Vacc_totalSAV (Vacc *thee, Vclist *clist, APOLparm *apolparm, double radius)

  Return the total solvent accessible volume (SAV)
- int Vacc_wcaEnergyAtom (Vacc *thee, APOLparm *apolparm, Valist *alist, Vclist *clist, int iatom, double *value)

  Calculate the WCA energy for an atom.
- VPUBLIC int Vacc_wcaEnergy (Vacc *acc, APOLparm *apolparm, Valist *alist, Vclist *clist)

  Return the WCA integral energy.
- VPUBLIC int Vacc_wcaForceAtom (Vacc *thee, APOLparm *apolparm, Vclist *clist, Vatom *atom, double *force)

  Return the WCA integral force.

9.63.1 Detailed Description

Class Vacc methods.

Author

Nathan Baker

Version

Id:

vacc.c 1793 2012-07-28 00:54:24Z sobolevnrm

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

center	Accessibility object
radius	Position to test
atomID	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

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

Definition at line 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 1793 2012-07-28 00:54:24Z sobolevnrm $")
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 VPRIVATE 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
        compute this if we're comparing atom against itself.
00123         if (atom->id == atomID) continue;
00124
```

```
  apos = atom->position;
  dist2 = VSQR(center[0]-apos[0]) + VSQR(center[1]-apos[1])
      + VSQR(center[2]-apos[2]);
  if (dist2 < VSQR(atom->radius+radius)){
            return 0;
  }
 }

    /* If we're still here, then the point is accessible */
    return 1;

}

VPUBLIC Vacc* Vacc_ctor(Valist *alist,
                        Vclist *clist,
                        double surf_density /* Surface density */
                        ) {


    Vacc *thee = VNULL;

    /* Set up the structure */
    thee = (Vacc*)Vmem_malloc(VNULL, 1, sizeof(Vacc) );
    VASSERT( thee != VNULL);
    VASSERT( Vacc_ctor2(thee, alist, clist, surf_density));
    return thee;
}

VPRIVATE int Vacc_storeParms(Vacc *thee,
                             Valist *alist,
                             Vclist *clist,
                             double surf_density /* Surface density */
                             ) {

    int nsphere,
        iatom;
    double maxrad = 0.0,
           maxarea,
           rad;
    Vatom *atom;

    if (alist == VNULL) {
        Vnm_print(2, "Vacc_storeParms:  Got NULL Valist!\n");
        return 0;
    } else thee->alist = alist;
    if (clist == VNULL) {
        Vnm_print(2, "Vacc_storeParms:  Got NULL Vclist!\n");
        return 0;
    } else thee->clist = clist;
    thee->surf_density = surf_density;

    /* Loop through the atoms to determine the maximum radius */
    maxrad = 0.0;
    for (iatom=0; iatom<Valist_getNumberAtoms(alist);
iatom++) {
        atom = Valist_getAtom(alist, iatom);
        rad = Vatom_getRadius(atom);
        if (rad > maxrad) maxrad = rad;
    }
    maxrad = maxrad + Vclist_maxRadius(thee->clist);

    maxarea = 4.0*VPI*maxrad*maxrad;
    nsphere = (int)ceil(maxarea*surf_density);

    Vnm_print(0, "Vacc_storeParms:  Surf. density = %g\n", surf_density);
    Vnm_print(0, "Vacc_storeParms:  Max area = %g\n", maxarea);
    thee->refSphere = VaccSurf_refSphere(thee->mem
, nsphere);
    Vnm_print(0, "Vacc_storeParms:  Using %d-point reference sphere\n",
            thee->refSphere->npts);

    return 1;
}

VPRIVATE int Vacc_allocate(Vacc *thee) {

    int i,
        natoms;

    natoms = Valist_getNumberAtoms(thee->alist);

```

```
    thee->atomFlags = (int*)Vmem_malloc(thee->mem, natoms, sizeof(int));
    if (thee->atomFlags == VNULL) {
        Vnm_print(2,
                "Vacc_allocate:  Failed to allocate %d (int)s for atomFlags!\n",
                 natoms);
        return 0;
    }
    for (i=0; i<natoms; i++) (thee->atomFlags)[i] = 0;

    return 1;
}


VPUBLIC int Vacc_ctor2(Vacc *thee,
                       Valist *alist,
                       Vclist *clist,
                       double surf_density
                       ) {

    /* Check and store parameters */
    if (!Vacc_storeParms(thee, alist, clist, surf_density)) {
        Vnm_print(2, "Vacc_ctor2:  parameter check failed!\n");
        return 0;
    }

    /* Set up memory management object */
    thee->mem = Vmem_ctor("APBS::VACC");
    if (thee->mem == VNULL) {
        Vnm_print(2, "Vacc_ctor2:  memory object setup failed!\n");
        return 0;
    }

    /* Setup and check probe */
    thee->surf = VNULL;

    /* Allocate space */
    if (!Vacc_allocate(thee)) {
        Vnm_print(2, "Vacc_ctor2:  memory allocation failed!\n");
        return 0;
    }

    return 1;
}


VPUBLIC void Vacc_dtor(Vacc **thee) {

    if ((*thee) != VNULL) {
        Vacc_dtor2(*thee);
        Vmem_free(VNULL, 1, sizeof(Vacc), (void **)thee);
        (*thee) = VNULL;
    }

}

VPUBLIC void Vacc_dtor2(Vacc *thee) {

    int i,
        natoms;

    natoms = Valist_getNumberAtoms(thee->alist);
    Vmem_free(thee->mem, natoms, sizeof(int), (void **)&(thee->atomFlags));

    if (thee->refSphere != VNULL) {
        VaccSurf_dtor(&(thee->refSphere));
        thee->refSphere = VNULL;
    }
    if (thee->surf != VNULL) {
        for (i=0; i<natoms; i++) VaccSurf_dtor(&(thee->surf[i]));
        Vmem_free(thee->mem, natoms, sizeof(VaccSurf *),
                (void **)&(thee->surf));
        thee->surf = VNULL;
    }

    Vmem_dtor(&(thee->mem));
}

```

```
VPUBLIC double Vacc_vdwAcc(Vacc *thee,
                           double center[3]
                           ) {

    VclistCell *cell;
    Vatom *atom;
    int iatom;
    double *apos,
           dist2;

    /* Get the relevant cell from the cell list */
    cell = Vclist_getCell(thee->clist, center);

    /* If we have no cell, then no atoms are nearby and we're definitely
     * accessible */
    if (cell == VNULL) return 1.0;

    /* Otherwise, check for overlap with the atoms in the cell */
    for (iatom=0; iatom<cell->natoms; iatom++) {
        atom = cell->atoms[iatom];
        apos = Vatom_getPosition(atom);
        dist2 = VSQR(center[0]-apos[0]) + VSQR(center[1]-apos[1])
               + VSQR(center[2]-apos[2]);
        if (dist2 < VSQR(Vatom_getRadius(atom))) return 0.0;
    }

    /* If we're still here, then the point is accessible */
    return 1.0;
}

VPUBLIC double Vacc_ivdwAcc(Vacc *thee,
                            double center[3],
                            double radius
                            ) {

    return (double)ivdwAccExclus(thee, center, radius, -1);

}

VPUBLIC void Vacc_splineAccGradAtomNorm(Vacc *thee,
                                        double center[VAPBS_DIM],
                                        double win,
                                        double infrad,
                                        Vatom *atom,
                                        double *grad
                                        ) {

    int i;
    double dist,
           *apos,
           arad,
           sm,
           sm2,
           w2i, /* inverse of win squared */
           w3i, /* inverse of win cubed */
           mygrad,
           mychi = 1.0;           /* Char. func. value for given atom */

    VASSERT(thee != NULL);

    /* Inverse squared window parameter */
    w2i = 1.0/(win*win);
    w3i = 1.0/(win*win*win);

    /* The grad is zero by default */
    for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;

    /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
     * *** MAGNITUDE OF THE FORCE *** */
    apos = Vatom_getPosition(atom);
    /* Zero-radius atoms don't contribute */
    if (Vatom_getRadius(atom) > 0.0) {
        arad = Vatom_getRadius(atom) + infrad;
        dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
          + VSQR(apos[2]-center[2]));
        /* If we're inside an atom, the entire characteristic function
         * will be zero and the grad will be zero, so we can stop */
        if (dist < (arad - win)) return;
        /* Likewise, if we're outside the smoothing window, the characteristic
         * function is unity and the grad will be zero, so we can stop */
```

```
        else if (dist > (arad + win)) return;
        /* Account for floating point error at the border
         * NAB:  COULDN'T THESE TESTS BE COMBINED AS BELOW
         * (Vacc_splineAccAtom)? */
        else if ((VABS(dist - (arad - win)) < VSMALL) ||
                 (VABS(dist - (arad + win)) < VSMALL)) return;
        /* If we're inside the smoothing window */
        else {
            sm = dist - arad + win;
            sm2 = VSQR(sm);
            mychi = 0.75*sm2*w2i -0.25*sm*sm2*w3i;
            mygrad = 1.5*sm*w2i - 0.75*sm2*w3i;
        }
        /* Now assemble the grad vector */
        VASSERT(mychi > 0.0);
        for (i=0; i<VAPBS_DIM; i++)
            grad[i] = -(mygrad/mychi)*((center[i] - apos[i])/dist);
    }
}

VPUBLIC void Vacc_splineAccGradAtomUnnorm(Vacc *thee,
                                          double center[VAPBS_DIM],
                                          double win,
                                          double infrad,
                                          Vatom *atom,
                                          double *grad
                                          ) {

    int i;
    double dist,
           *apos,
           arad,
           sm,
           sm2,
           w2i, /* Inverse of win squared */
           w3i, /* Inverse of win cubed */
           mygrad,
           mychi = 1.0;           /* Char. func. value for given atom */

    VASSERT(thee != NULL);

    /* Inverse squared window parameter */
    w2i = 1.0/(win*win);
    w3i = 1.0/(win*win*win);

    /* The grad is zero by default */
    for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;

    /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
     * *** MAGNITUDE OF THE FORCE *** */
    apos = Vatom_getPosition(atom);
    /* Zero-radius atoms don't contribute */
    if (Vatom_getRadius(atom) > 0.0) {
        arad = Vatom_getRadius(atom) + infrad;
        dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
          + VSQR(apos[2]-center[2]));
        /* If we're inside an atom, the entire characteristic function
         * will be zero and the grad will be zero, so we can stop */
        if (dist < (arad - win)) return;
        /* Likewise, if we're outside the smoothing window, the characteristic
         * function is unity and the grad will be zero, so we can stop */
        else if (dist > (arad + win)) return;
        /* Account for floating point error at the border
         * NAB:  COULDN'T THESE TESTS BE COMBINED AS BELOW
         * (Vacc_splineAccAtom)? */
        else if ((VABS(dist - (arad - win)) < VSMALL) ||
                 (VABS(dist - (arad + win)) < VSMALL)) return;
        /* If we're inside the smoothing window */
        else {
            sm = dist - arad + win;
            sm2 = VSQR(sm);
            mychi = 0.75*sm2*w2i -0.25*sm*sm2*w3i;
            mygrad = 1.5*sm*w2i - 0.75*sm2*w3i;
        }
        /* Now assemble the grad vector */
        VASSERT(mychi > 0.0);
        for (i=0; i<VAPBS_DIM; i++)
            grad[i] = -(mygrad)*((center[i] - apos[i])/dist);
    }
}
```

```c

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

    double dist,
           *apos,
           arad,
           sm,
           sm2,
           w2i, /* Inverse of win squared */
           w3i, /* Inverse of win cubed */
           value,
           stot,
           sctot;

    VASSERT(thee != NULL);

    /* Inverse squared window parameter */
    w2i = 1.0/(win*win);
    w3i = 1.0/(win*win*win);

    apos = Vatom_getPosition(atom);
    /* Zero-radius atoms don't contribute */
    if (Vatom_getRadius(atom) > 0.0) {
        arad = Vatom_getRadius(atom) + infrad;
        stot = arad + win;
        sctot = VMAX2(0, (arad - win));
        dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
          + VSQR(apos[2]-center[2]));
        /* If we're inside an atom, the entire characteristic function
         * will be zero */
        if ((dist < sctot) || (VABS(dist - sctot) < VSMALL)){
            value = 0.0;
        /* We're outside the smoothing window */
        } else if ((dist > stot) || (VABS(dist - stot) < VSMALL)) {
            value = 1.0;
        /* We're inside the smoothing window */
        } else {
            sm = dist - arad + win;
            sm2 = VSQR(sm);
            value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
        }
    } else value = 1.0;

    return value;
}

VPRIVATE double splineAcc(
        Vacc *thee,
        double center[VAPBS_DIM],
        double win,
        double infrad,
        VclistCell *cell
        ) {

    int atomID, iatom;
    Vatom *atom;
    double value = 1.0;

    VASSERT(thee != NULL);

    /* Now loop through the atoms assembling the characteristic function */
    for (iatom=0; iatom<cell->natoms; iatom++) {

        atom = cell->atoms[iatom];
        atomID = atom->id;

        /* Check to see if we've counted this atom already */
        if ( !(thee->atomFlags[atomID]) ) {

            thee->atomFlags[atomID] = 1;
            value *= Vacc_splineAccAtom(thee, center, win,
infrad, atom);

            if (value < VSMALL) return value;
        }
    }
```

```
    return value;
}


VPUBLIC double Vacc_splineAcc(Vacc *thee, double center[VAPBS_DIM], double win,
  double infrad) {

    VclistCell *cell;
    Vatom *atom;
    int iatom, atomID;


    VASSERT(thee != NULL);

    if (Vclist_maxRadius(thee->clist) < (win + infrad)) {
        Vnm_print(2, "Vacc_splineAcc:  Vclist has max_radius=%g;\n",
                Vclist_maxRadius(thee->clist));
        Vnm_print(2, "Vacc_splineAcc:  Insufficient for win=%g, infrad=%g\n",
                win, infrad);
        VASSERT(0);
    }

    /* Get a cell or VNULL; in the latter case return 1.0 */
    cell = Vclist_getCell(thee->clist, center);
    if (cell == VNULL) return 1.0;

    /* First, reset the list of atom flags
     * NAB:  THIS SEEMS VERY INEFFICIENT */
    for (iatom=0; iatom<cell->natoms; iatom++) {
        atom = cell->atoms[iatom];
        atomID = atom->id;
        thee->atomFlags[atomID] = 0;
    }

    return splineAcc(thee, center, win, infrad, cell);
}

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

    int iatom, i, atomID;
    double acc = 1.0;
    double tgrad[VAPBS_DIM];
    VclistCell *cell;
    Vatom *atom = VNULL;

    VASSERT(thee != NULL);

    if (Vclist_maxRadius(thee->clist) < (win + infrad)) {
        Vnm_print(2, "Vacc_splineAccGrad: Vclist max_radius=%g;\n",
                Vclist_maxRadius(thee->clist));
        Vnm_print(2, "Vacc_splineAccGrad: Insufficient for win=%g, infrad=%g\n",
                win, infrad);
        VASSERT(0);
    }

    /* Reset the gradient */
    for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;

    /* Get the cell; check for nullity */
    cell = Vclist_getCell(thee->clist, center);
    if (cell == VNULL) return;

    /* Reset the list of atom flags */
    for (iatom=0; iatom<cell->natoms; iatom++) {
        atom = cell->atoms[iatom];
        atomID = atom->id;
        thee->atomFlags[atomID] = 0;
    }

    /* Get the local accessibility */
    acc = splineAcc(thee, center, win, infrad, cell);

    /* Accumulate the gradient of all local atoms */
    if (acc > VSMALL) {
        for (iatom=0; iatom<cell->natoms; iatom++) {
            atom = cell->atoms[iatom];
```

```
            Vacc_splineAccGradAtomNorm(thee, center, win, infrad, atom, tgrad);
        }
        for (i=0; i<VAPBS_DIM; i++) grad[i] += tgrad[i];
    }
    for (i=0; i<VAPBS_DIM; i++) grad[i] *= -acc;
}

VPUBLIC double Vacc_molAcc(Vacc *thee, double center[VAPBS_DIM],
        double radius) {

    double rc;

    /* ******* CHECK IF OUTSIDE ATOM+PROBE RADIUS SURFACE ***** */
    if (Vacc_ivdwAcc(thee, center, radius) == 1.0) {

        /* Vnm_print(2, "DEBUG:  ivdwAcc = 1.0\n"); */
        rc = 1.0;

    /* ******* CHECK IF INSIDE ATOM RADIUS SURFACE ***** */
    } else if (Vacc_vdwAcc(thee, center) == 0.0) {

        /* Vnm_print(2, "DEBUG:  vdwAcc = 0.0\n"); */
        rc = 0.0;

    /* ******* CHECK IF OUTSIDE MOLECULAR SURFACE ***** */
    } else {

        /* Vnm_print(2, "DEBUG:  calling fastMolAcc...\n"); */
        rc = Vacc_fastMolAcc(thee, center, radius);

    }

    return rc;

}

VPUBLIC double Vacc_fastMolAcc(Vacc *thee, double center[VAPBS_DIM],
        double radius) {

    Vatom *atom;
    VaccSurf *surf;
    VclistCell *cell;
    int ipt, iatom, atomID;
    double dist2, rad2;

    rad2 = radius*radius;

    /* Check to see if the SAS has been defined */
    if (thee->surf == VNULL) Vacc_SASA(thee, radius);

    /* Get the cell associated with this point */
    cell = Vclist_getCell(thee->clist, center);
    if (cell == VNULL) {
        Vnm_print(2, "Vacc_fastMolAcc:  unexpected VNULL VclistCell!\n");
        return 1.0;
    }

    /* Loop through all the atoms in the cell */
    for (iatom=0; iatom<cell->natoms; iatom++) {
        atom = cell->atoms[iatom];
        atomID = Vatom_getAtomID(atom);
        surf = thee->surf[atomID];
        /* Loop through all SAS points associated with this atom */
        for (ipt=0; ipt<surf->npts; ipt++) {
            /* See if we're within a probe radius of the point */
            dist2 = VSQR(center[0]-(surf->xpts[ipt]))
                + VSQR(center[1]-(surf->ypts[ipt]))
                + VSQR(center[2]-(surf->zpts[ipt]));
            if (dist2 < rad2) return 1.0;
        }
    }

    /* If all else failed, we are not inside the molecular surface */
    return 0.0;
}


#if defined(HAVE_MC_H)
```

```
VPUBLIC void Vacc_writeGMV(Vacc *thee, double radius, int meth, Gem *gm,
  char *iodev, char *iofmt, char *iohost, char *iofile) {

    double *accVals[MAXV], coord[3];
    Vio *sock;
    int ivert, icoord;

    for (ivert=0; ivert<MAXV; ivert++) accVals[ivert] = VNULL;
    accVals[0] = (void *)Vmem_malloc(thee->mem, Gem_numVV(gm), sizeof(double
    accVals[1] = (void *)Vmem_malloc(thee->mem, Gem_numVV(gm), sizeof(double
    for (ivert=0; ivert<Gem_numVV(gm); ivert++) {
        for (icoord=0;icoord<3;icoord++)
          coord[icoord] = VV_coord(Gem_VV(gm, ivert), icoord);
        if (meth == 0) {
            accVals[0][ivert] = Vacc_molAcc(thee, coord, radius);
            accVals[1][ivert] = Vacc_molAcc(thee, coord, radius);
        } else if (meth == 1) {
            accVals[0][ivert] = Vacc_ivdwAcc(thee, coord, radius);
            accVals[1][ivert] = Vacc_ivdwAcc(thee, coord, radius);
        } else if (meth == 2) {
            accVals[0][ivert] = Vacc_vdwAcc(thee, coord);
            accVals[1][ivert] = Vacc_vdwAcc(thee, coord);
        } else VASSERT(0);
    }
    sock = Vio_ctor(iodev, iofmt, iohost, iofile, "w");
    Gem_writeGMV(gm, sock, 1, accVals);
    Vio_dtor(&sock);
    Vmem_free(thee->mem, Gem_numVV(gm), sizeof(double),
      (void **)&(accVals[0]));
    Vmem_free(thee->mem, Gem_numVV(gm), sizeof(double),
      (void **)&(accVals[1]));
}
#endif /* defined(HAVE_MC_H) */

VPUBLIC double Vacc_SASA(Vacc *thee,
                         double radius
                         ) {

    int i,
        natom;
    double area;
           //*apos; // gcc says unused
    Vatom *atom;
    VaccSurf *asurf;

    time_t ts; // PCE: temp
    ts = clock();

 //unsigned long long mbeg; // gcc says unused

    natom = Valist_getNumberAtoms(thee->alist);

    /* Check to see if we need to build the surface */
    if (thee->surf == VNULL) {
        thee->surf = Vmem_malloc(thee->mem, natom, sizeof(VaccSurf
 *));

#if defined(DEBUG_MAC_OSX_OCL) || defined(DEBUG_MAC_OSX_STANDARD)
#include "mach_chud.h"
  machm_(&mbeg);
#pragma omp parallel for private(i,atom)
#endif
        for (i=0; i<natom; i++) {
            atom = Valist_getAtom(thee->alist, i);
            /* NOTE:  RIGHT NOW WE DO THIS FOR THE ENTIRE MOLECULE WHICH IS
             * INCREDIBLY INEFFICIENT, PARTICULARLY DURING FOCUSING!!! */
            thee->surf[i] = Vacc_atomSurf(thee, atom, thee->
refSphere,
            radius);
        }
    }

    /* Calculate the area */
    area = 0.0;
    for (i=0; i<natom; i++) {
        atom = Valist_getAtom(thee->alist, i);
        asurf = thee->surf[i];
        /* See if this surface needs to be rebuilt */
        if (asurf->probe_radius != radius) {
```

```
            Vnm_print(2, "Vacc_SASA:  Warning -- probe radius changed from %g to %g!\n",
       asurf->probe_radius, radius);
            VaccSurf_dtor2(asurf);
            thee->surf[i] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
            asurf = thee->surf[i];
        }
        area += (asurf->area);
    }

#if defined(DEBUG_MAC_OSX_OCL) || defined(DEBUG_MAC_OSX_STANDARD)
 mets_(&mbeg, "Vacc_SASA - Parallel");
#endif

    Vnm_print(0, "Vacc_SASA: Time elapsed: %f\n", ((double)clock() - ts) / CLOCKS_PER_SEC);
    return area;

}

VPUBLIC double Vacc_totalSASA(Vacc *thee, double radius) {

    return Vacc_SASA(thee, radius);

}

VPUBLIC double Vacc_atomSASA(Vacc *thee, double radius, Vatom *atom) {

    VaccSurf *asurf;
    int id;

    if (thee->surf == VNULL) Vacc_SASA(thee, radius);

    id = Vatom_getAtomID(atom);
    asurf = thee->surf[id];

    /* See if this surface needs to be rebuilt */
    if (asurf->probe_radius != radius) {
        Vnm_print(2, "Vacc_SASA:  Warning -- probe radius changed from %g to %g!\n",
                 asurf->probe_radius, radius);
        VaccSurf_dtor2(asurf);
        thee->surf[id] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
        asurf = thee->surf[id];
    }

    return asurf->area;

}

VPUBLIC VaccSurf* VaccSurf_ctor(Vmem *mem, double probe_radius, int nsphere) {
    VaccSurf *thee;

    //thee = Vmem_malloc(mem, 1, sizeof(Vacc) );
 if (nsphere >= MAX_SPHERE_PTS) {
  Vnm_print(2, "VaccSurf_ctor:  Error!  The requested number of grid points (%d) exceeds the maximum (%d)!\n", nsphere, MAX_SPHERE_PTS);
  Vnm_print(2, "VaccSurf_ctor:  Please check the variable MAX_SPHERE_PTS to reset.\n");
  VASSERT(0);
 }
 thee = (VaccSurf*)calloc(1,sizeof(Vacc));
    VASSERT( VaccSurf_ctor2(thee, mem, probe_radius, nsphere) );

    return thee;
}

VPUBLIC int VaccSurf_ctor2(VaccSurf *thee, Vmem *mem, double probe_radius,
        int nsphere) {

    if (thee == VNULL)
     return 0;

    thee->mem = mem;
    thee->npts = nsphere;
    thee->probe_radius = probe_radius;
```

```
    thee->area = 0.0;

    if (thee->npts > 0) {
        thee->xpts = Vmem_malloc(thee->mem, thee->npts, sizeof(double));
        thee->ypts = Vmem_malloc(thee->mem, thee->npts, sizeof(double));
        thee->zpts = Vmem_malloc(thee->mem, thee->npts, sizeof(double));
        thee->bpts = Vmem_malloc(thee->mem, thee->npts, sizeof(char));
    } else {
        thee->xpts = VNULL;
        thee->ypts = VNULL;
        thee->zpts = VNULL;
        thee->bpts = VNULL;
    }

    return 1;
}

VPUBLIC void VaccSurf_dtor(VaccSurf **thee) {

    Vmem *mem;

    if ((*thee) != VNULL) {
        mem = (*thee)->mem;
        VaccSurf_dtor2(*thee);
        //Vmem_free(mem, 1, sizeof(VaccSurf), (void **)thee);
  free(*thee);
        (*thee) = VNULL;
    }

}

VPUBLIC void VaccSurf_dtor2(VaccSurf *thee) {

    if (thee->npts > 0) {
        Vmem_free(thee->mem, thee->npts, sizeof(double),
                (void **)&(thee->xpts));
        Vmem_free(thee->mem, thee->npts, sizeof(double),
                (void **)&(thee->ypts));
        Vmem_free(thee->mem, thee->npts, sizeof(double),
                (void **)&(thee->zpts));
        Vmem_free(thee->mem, thee->npts, sizeof(char),
                (void **)&(thee->bpts));
    }

}

VPUBLIC VaccSurf* Vacc_atomSurf(Vacc *thee,
                                Vatom *atom,
                                VaccSurf *ref,
                                double prad
                               ) {

    VaccSurf *surf;
    int i,
        j,
        npts,
        atomID;
    double arad,
           rad,
           pos[3],
           *apos;
 char bpts[MAX_SPHERE_PTS];

    /* Get atom information */
    arad = Vatom_getRadius(atom);
    apos = Vatom_getPosition(atom);
    atomID = Vatom_getAtomID(atom);

    if (arad < VSMALL) {
        return VaccSurf_ctor(thee->mem, prad, 0);
    }

    rad = arad + prad;

    /* Determine which points will contribute */
    npts = 0;
    for (i=0; i<ref->npts; i++) {
```

```
        /* Reset point flag: zero-radius atoms do not contribute */
        pos[0] = rad*(ref->xpts[i]) + apos[0];
        pos[1] = rad*(ref->ypts[i]) + apos[1];
        pos[2] = rad*(ref->zpts[i]) + apos[2];
        if (ivdwAccExclus(thee, pos, prad, atomID)) {
            npts++;
            bpts[i] = 1;
        } else {
            bpts[i] = 0;
        }
    }

    /* Allocate space for the points */
    surf = VaccSurf_ctor(thee->mem, prad, npts);

    /* Assign the points */
    j = 0;
    for (i=0; i<ref->npts; i++) {
        if (bpts[i]) {
            surf->bpts[j] = 1;
            surf->xpts[j] = rad*(ref->xpts[i]) + apos[0];
            surf->ypts[j] = rad*(ref->ypts[i]) + apos[1];
            surf->zpts[j] = rad*(ref->zpts[i]) + apos[2];
            j++;
        }
    }

    /* Assign the area */
    surf->area = 4.0*VPI*rad*rad*((double)(surf->npts))/((double)(ref->
npts));

    return surf;

}

VPUBLIC VaccSurf* VaccSurf_refSphere(Vmem *mem, int
npts) {

    VaccSurf *surf;
    int nactual, i, itheta, ntheta, iphi, nphimax, nphi;
    double frac;
    double sintheta, costheta, theta, dtheta;
    double sinphi, cosphi, phi, dphi;

    /* Setup "constants" */
    frac = ((double)(npts))/4.0;
    ntheta = VRINT(VSQRT(Vunit_pi*frac));
    dtheta = Vunit_pi/((double)(ntheta));
    nphimax = 2*ntheta;

    /* Count the actual number of points to be used */
    nactual = 0;
    for (itheta=0; itheta<ntheta; itheta++) {
        theta = dtheta*((double)(itheta));
        sintheta = VSIN(theta);
        costheta = VCOS(theta);
        nphi = VRINT(sintheta*nphimax);
        nactual += nphi;
    }

    /* Allocate space for the points */
    surf = VaccSurf_ctor(mem, 1.0, nactual);

    /* Clear out the boolean array */
    for (i=0; i<nactual; i++) surf->bpts[i] = 1;

    /* Assign the points */
    nactual = 0;
    for (itheta=0; itheta<ntheta; itheta++) {
        theta = dtheta*((double)(itheta));
        sintheta = VSIN(theta);
        costheta = VCOS(theta);
        nphi = VRINT(sintheta*nphimax);
        if (nphi != 0) {
            dphi = 2*Vunit_pi/((double)(nphi));
            for (iphi=0; iphi<nphi; iphi++) {
                phi = dphi*((double)(iphi));
                sinphi = VSIN(phi);
                cosphi = VCOS(phi);
                surf->xpts[nactual] = cosphi * sintheta;
                surf->ypts[nactual] = sinphi * sintheta;
```

```
                surf->zpts[nactual] = costheta;
                nactual++;
            }
        }
    }

    surf->npts = nactual;

    return surf;
}

VPUBLIC VaccSurf* Vacc_atomSASPoints(Vacc *thee, double radius,
        Vatom *atom) {

    VaccSurf *asurf = VNULL;
    int id;

    if (thee->surf == VNULL) Vacc_SASA(thee, radius);
    id = Vatom_getAtomID(atom);

    asurf = thee->surf[id];

    /* See if this surface needs to be rebuilt */
    if (asurf->probe_radius != radius) {
        Vnm_print(2, "Vacc_SASA:  Warning -- probe radius changed from %g to %g!\n",
                asurf->probe_radius, radius);
        VaccSurf_dtor2(asurf);
        thee->surf[id] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
        asurf = thee->surf[id];
    }

    return asurf;

}

VPUBLIC void Vacc_splineAccGradAtomNorm4(Vacc *thee, double center[VAPBS_DIM],
            double win, double infrad, Vatom *atom, double *grad) {

    int i;
    double dist, *apos, arad, sm, sm2, sm3, sm4, sm5, sm6, sm7;
    double e, e2, e3, e4, e5, e6, e7;
    double b, b2, b3, b4, b5, b6, b7;
    double c0, c1, c2, c3, c4, c5, c6, c7;
    double denom, mygrad;
    double mychi = 1.0;           /* Char. func. value for given atom */

    VASSERT(thee != NULL);

    /* The grad is zero by default */
    for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;

    /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
     * *** MAGNITUDE OF THE FORCE *** */
    apos = Vatom_getPosition(atom);
    /* Zero-radius atoms don't contribute */
    if (Vatom_getRadius(atom) > 0.0) {

        arad = Vatom_getRadius(atom);
        arad = arad + infrad;
        b = arad - win;
        e = arad + win;

        e2 = e * e;
        e3 = e2 * e;
        e4 = e3 * e;
        e5 = e4 * e;
        e6 = e5 * e;
        e7 = e6 * e;
        b2 = b * b;
        b3 = b2 * b;
        b4 = b3 * b;
        b5 = b4 * b;
        b6 = b5 * b;
        b7 = b6 * b;

        denom = e7  - 7.0*b*e6 + 21.0*b2*e5 - 35.0*e4*b3
    + 35.0*e3*b4 - 21.0*b5*e2  + 7.0*e*b6 - b7;
```

```
        c0 = b4*(35.0*e3 - 21.0*b*e2 + 7*e*b2 - b3)/denom;
        c1 = -140.0*b3*e3/denom;
        c2 = 210.0*e2*b2*(e + b)/denom;
        c3 = -140.0*e*b*(e2 + 3.0*b*e + b2)/denom;
        c4 =  35.0*(e3 + 9.0*b*e2 + + 9.0*e*b2 + b3)/denom;
        c5 = -84.0*(e2 + 3.0*b*e + b2)/denom;
        c6 =  70.0*(e + b)/denom;
        c7 = -20.0/denom;

        dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
      + VSQR(apos[2]-center[2]));

        /* If we're inside an atom, the entire characteristic function
   * will be zero and the grad will be zero, so we can stop */
        if (dist < (arad - win)) return;
        /* Likewise, if we're outside the smoothing window, the characteristic
   * function is unity and the grad will be zero, so we can stop */
        else if (dist > (arad + win)) return;
        /* Account for floating point error at the border
   * NAB:  COULDN'T THESE TESTS BE COMBINED AS BELOW
   * (Vacc_splineAccAtom)? */
        else if ((VABS(dist - (arad - win)) < VSMALL) ||
                 (VABS(dist - (arad + win)) < VSMALL)) return;
        /* If we're inside the smoothing window */
        else {
            sm = dist;
            sm2 = sm * sm;
            sm3 = sm2 * sm;
            sm4 = sm3 * sm;
            sm5 = sm4 * sm;
            sm6 = sm5 * sm;
            sm7 = sm6 * sm;
            mychi = c0 + c1*sm + c2*sm2 + c3*sm3
    + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
            mygrad = c1 + 2.0*c2*sm  + 3.0*c3*sm2 + 4.0*c4*sm3
    + 5.0*c5*sm4 + 6.0*c6*sm5 + 7.0*c7*sm6;
            if (mychi <= 0.0) {
    /* Avoid numerical round off errors */
    return;
            } else if (mychi > 1.0) {
    /* Avoid numerical round off errors */
    mychi = 1.0;
            }
        }
        /* Now assemble the grad vector */
        VASSERT(mychi > 0.0);
        for (i=0; i<VAPBS_DIM; i++)
            grad[i] = -(mygrad/mychi)*((center[i] - apos[i])/dist);
    }
}

VPUBLIC void Vacc_splineAccGradAtomNorm3(Vacc *
    thee, double center[VAPBS_DIM],
           double win, double infrad, Vatom *atom, double *grad) {

    int i;
    double dist, *apos, arad, sm, sm2, sm3, sm4, sm5;
    double e, e2, e3, e4, e5;
    double b, b2, b3, b4, b5;
    double c0, c1, c2, c3, c4, c5;
    double denom, mygrad;
    double mychi = 1.0;           /* Char. func. value for given atom */

    VASSERT(thee != NULL);

    /* The grad is zero by default */
    for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;

    /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
   * *** MAGNITUDE OF THE FORCE *** */
    apos = Vatom_getPosition(atom);
    /* Zero-radius atoms don't contribute */
    if (Vatom_getRadius(atom) > 0.0) {

        arad = Vatom_getRadius(atom);
        arad = arad + infrad;
        b = arad - win;
        e = arad + win;

        e2 = e * e;
        e3 = e2 * e;
```

```
        e4 = e3 * e;
        e5 = e4 * e;
        b2 = b * b;
        b3 = b2 * b;
        b4 = b3 * b;
        b5 = b4 * b;

        denom = pow((e - b), 5.0);
        c0 = -10.0*e2*b3 + 5.0*e*b4 - b5;
        c1 = 30.0*e2*b2;
        c2 = -30.0*(e2*b + e*b2);
        c3 = 10.0*(e2 + 4.0*e*b + b2);
        c4 = -15.0*(e + b);
        c5 = 6;
        c0 = c0/denom;
        c1 = c1/denom;
        c2 = c2/denom;
        c3 = c3/denom;
        c4 = c4/denom;
        c5 = c5/denom;

        dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
     + VSQR(apos[2]-center[2]));

        /* If we're inside an atom, the entire characteristic function
   * will be zero and the grad will be zero, so we can stop */
        if (dist < (arad - win)) return;
        /* Likewise, if we're outside the smoothing window, the characteristic
   * function is unity and the grad will be zero, so we can stop */
        else if (dist > (arad + win)) return;
        /* Account for floating point error at the border
   * NAB:  COULDN'T THESE TESTS BE COMBINED AS BELOW
   * (Vacc_splineAccAtom)? */
        else if ((VABS(dist - (arad - win)) < VSMALL) ||
                 (VABS(dist - (arad + win)) < VSMALL)) return;
        /* If we're inside the smoothing window */
        else {
            sm = dist;
            sm2 = sm * sm;
            sm3 = sm2 * sm;
            sm4 = sm3 * sm;
            sm5 = sm4 * sm;
            mychi = c0 + c1*sm + c2*sm2 + c3*sm3
    + c4*sm4 + c5*sm5;
            mygrad = c1 + 2.0*c2*sm  + 3.0*c3*sm2 + 4.0*c4*sm3
    + 5.0*c5*sm4;
            if (mychi <= 0.0) {
    /* Avoid numerical round off errors */
    return;
            } else if (mychi > 1.0) {
    /* Avoid numerical round off errors */
    mychi = 1.0;
            }
        }
        /* Now assemble the grad vector */
        VASSERT(mychi > 0.0);
        for (i=0; i<VAPBS_DIM; i++)
            grad[i] = -(mygrad/mychi)*((center[i] - apos[i])/dist);
    }
}

/* ///////////////////////////////////////////////////////////////////////////
   // Routine:  Vacc_atomdSAV
   //
   // Purpose:  Calculates the vector valued atomic derivative of volume
   //
   // Args:     radius  The radius of the solvent probe in Angstroms
   //           iatom   Index of the atom in thee->alist
   //
   // Author:   Jason Wagoner
   //           Nathan Baker (original FORTRAN routine from UHBD by Brock Luty)
VPUBLIC void Vacc_atomdSAV(Vacc *thee,
                           double srad,
                           Vatom *atom,
                           double *dSA
                          ) {

    int ipt, iatom;

    double area;
    double *tPos, tRad, vec[3];
```

```
    double dx,dy,dz;
    VaccSurf *ref;
    dx = 0.0;
    dy = 0.0;
    dz = 0.0;
    /* Get the atom information */
    ref = thee->refSphere;
    iatom = Vatom_getAtomID(atom);

    dSA[0] = 0.0;
    dSA[1] = 0.0;
    dSA[2] = 0.0;

    tPos = Vatom_getPosition(atom);
    tRad = Vatom_getRadius(atom);

 if(tRad == 0.0) return;

    area = 4.0*VPI*(tRad+srad)*(tRad+srad)/((double)(ref->npts));
    for (ipt=0; ipt<ref->npts; ipt++) {
        vec[0] = (tRad+srad)*ref->xpts[ipt] + tPos[0];
        vec[1] = (tRad+srad)*ref->ypts[ipt] + tPos[1];
        vec[2] = (tRad+srad)*ref->zpts[ipt] + tPos[2];
        if (ivdwAccExclus(thee, vec, srad, iatom)) {
   dx = dx+vec[0]-tPos[0];
   dy = dy+vec[1]-tPos[1];
   dz = dz+vec[2]-tPos[2];
     }
    }

    if ((tRad+srad) != 0){
  dSA[0] = dx*area/(tRad+srad);
  dSA[1] = dy*area/(tRad+srad);
  dSA[2] = dz*area/(tRad+srad);
    }

}

/* Note: This is purely test code to make certain that the dSASA code is
   behaving properly. This function should NEVER be called by anyone
   other than an APBS developer at Wash U.
*/
VPRIVATE double Vacc_SASAPos(Vacc *thee, double radius) {

    int i, natom;
    double area;
    Vatom *atom;
    VaccSurf *asurf;

    natom = Valist_getNumberAtoms(thee->alist);

    /* Calculate the area */
    area = 0.0;
    for (i=0; i<natom; i++) {
        atom = Valist_getAtom(thee->alist, i);
        asurf = thee->surf[i];

        VaccSurf_dtor2(asurf);
        thee->surf[i] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
        asurf = thee->surf[i];
        area += (asurf->area);
    }

    return area;

}

VPRIVATE double Vacc_atomSASAPos(Vacc *thee,
                                 double radius,
                                 Vatom *atom, /* The atom being manipulated */
                                 int mode
                                ) {

    VaccSurf *asurf;
    int id;
 static int warned = 0;

    if ((thee->surf == VNULL) || (mode == 1)){
  if(!warned){
```

```
    Vnm_print(2, "WARNING: Recalculating entire surface!!!!\n");
    warned = 1;
   }
   Vacc_SASAPos(thee, radius); // reinitialize before we can do anything about doing a calculation on a repositioned atom
  }

    id = Vatom_getAtomID(atom);
    asurf = thee->surf[id];

    VaccSurf_dtor(&asurf);
    thee->surf[id] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
    asurf = thee->surf[id];

    //printf("%s: Time elapsed: %f\n", __func__, ((double)clock() - ts) / CLOCKS_PER_SEC);

    return asurf->area;
}

/* ///////////////////////////////////////////////////////////////////////////
   // Routine:  Vacc_atomdSASA
   //
   // Purpose:  Calculates the derivative of surface area with respect to atomic
   //           displacement using finite difference methods.
   //
   // Args:     radius  The radius of the solvent probe in Angstroms
   //           iatom   Index of the atom in thee->alist
   //
   // Author:   Jason Wagoner
   //   David Gohara
   //           Nathan Baker (original FORTRAN routine from UHBD by Brock Luty)
VPUBLIC void Vacc_atomdSASA(Vacc *thee,
                            double dpos,
                            double srad,
                            Vatom *atom,
                            double *dSA
                           ) {

    double *temp_Pos,
           tPos[3],
           axb1,
           axt1,
           ayb1,
           ayt1,
           azb1,
           azt1;
    VaccSurf *ref;

    //printf("%s: entering\n", __func__);
    time_t ts;
    ts = clock();

    /* Get the atom information */
    ref = thee->refSphere;
    temp_Pos = Vatom_getPosition(atom); // Get a pointer to the position object.  You actually manipulate the atom doing this...

 tPos[0] = temp_Pos[0];
    tPos[1] = temp_Pos[1];
    tPos[2] = temp_Pos[2];

 /* Shift by pos -/+ on x */
 temp_Pos[0] -= dpos;
    axb1 = Vacc_atomSASAPos(thee, srad, atom,0);
 temp_Pos[0] = tPos[0];

    temp_Pos[0] += dpos;
    axt1 = Vacc_atomSASAPos(thee, srad, atom,0);
 temp_Pos[0] = tPos[0];

 /* Shift by pos -/+ on y */
    temp_Pos[1] -= dpos;
    ayb1 = Vacc_atomSASAPos(thee, srad, atom,0);
 temp_Pos[1] = tPos[1];

    temp_Pos[1] += dpos;
    ayt1 = Vacc_atomSASAPos(thee, srad, atom,0);
 temp_Pos[1] = tPos[1];
```

```

 /* Shift by pos -/+ on z */
    temp_Pos[2] -= dpos;
    azb1 = Vacc_atomSASAPos(thee, srad, atom,0);
 temp_Pos[2] = tPos[2];

    temp_Pos[2] += dpos;
    azt1 = Vacc_atomSASAPos(thee, srad, atom,0);
 temp_Pos[2] = tPos[2];

 /* Reset the atom SASA to zero displacement */
 Vacc_atomSASAPos(thee, srad, atom,0);

 /* Calculate the final value */
    dSA[0] = (axt1-axb1)/(2.0 * dpos);
    dSA[1] = (ayt1-ayb1)/(2.0 * dpos);
    dSA[2] = (azt1-azb1)/(2.0 * dpos);
}

/* Note: This is purely test code to make certain that the dSASA code is
   behaving properly. This function should NEVER be called by anyone
   other than an APBS developer at Wash U.
*/
VPUBLIC void Vacc_totalAtomdSASA(Vacc *thee, double dpos
, double srad, Vatom *atom, double *dSA) {

    int iatom;
    double *temp_Pos, tRad;
    double tPos[3];
 double axb1,axt1,ayb1,ayt1,azb1,azt1;
    VaccSurf *ref;

    /* Get the atom information */
    ref = thee->refSphere;
    temp_Pos = Vatom_getPosition(atom);
    tRad = Vatom_getRadius(atom);
    iatom = Vatom_getAtomID(atom);

    dSA[0] = 0.0;
    dSA[1] = 0.0;
    dSA[2] = 0.0;

 tPos[0] = temp_Pos[0];
    tPos[1] = temp_Pos[1];
    tPos[2] = temp_Pos[2];

 /* Shift by pos -/+ on x */
 temp_Pos[0] -= dpos;
    axb1 = Vacc_atomSASAPos(thee, srad, atom, 1);
 temp_Pos[0] = tPos[0];

    temp_Pos[0] += dpos;
    axt1 = Vacc_atomSASAPos(thee, srad, atom, 1);
 temp_Pos[0] = tPos[0];

 /* Shift by pos -/+ on y */
    temp_Pos[1] -= dpos;
    ayb1 = Vacc_atomSASAPos(thee, srad, atom, 1);
 temp_Pos[1] = tPos[1];

    temp_Pos[1] += dpos;
    ayt1 = Vacc_atomSASAPos(thee, srad, atom, 1);
 temp_Pos[1] = tPos[1];

 /* Shift by pos -/+ on z */
    temp_Pos[2] -= dpos;
    azb1 = Vacc_atomSASAPos(thee, srad, atom, 1);
 temp_Pos[2] = tPos[2];

    temp_Pos[2] += dpos;
    azt1 = Vacc_atomSASAPos(thee, srad, atom, 1);
 temp_Pos[2] = tPos[2];

 /* Calculate the final value */
    dSA[0] = (axt1-axb1)/(2.0 * dpos);
    dSA[1] = (ayt1-ayb1)/(2.0 * dpos);
    dSA[2] = (azt1-azb1)/(2.0 * dpos);
}

/* Note: This is purely test code to make certain that the dSASA code is
   behaving properly. This function should NEVER be called by anyone
```

```
   other than an APBS developer at Wash U.
*/
VPUBLIC void Vacc_totalAtomdSAV(Vacc *thee, double dpos, double srad, Vatom *atom, double *dSA, Vclist *clist) {

    int iatom;
    double *temp_Pos, tRad;
    double tPos[3];
 double axb1,axt1,ayb1,ayt1,azb1,azt1;
    VaccSurf *ref;

    /* Get the atom information */
    ref = thee->refSphere;
    temp_Pos = Vatom_getPosition(atom);
    tRad = Vatom_getRadius(atom);
    iatom = Vatom_getAtomID(atom);

    dSA[0] = 0.0;
    dSA[1] = 0.0;
    dSA[2] = 0.0;

 tPos[0] = temp_Pos[0];
    tPos[1] = temp_Pos[1];
    tPos[2] = temp_Pos[2];

 /* Shift by pos -/+ on x */
 temp_Pos[0] -= dpos;
    axb1 = Vacc_totalSAV(thee,clist, VNULL, srad);
 temp_Pos[0] = tPos[0];

    temp_Pos[0] += dpos;
    axt1 = Vacc_totalSAV(thee,clist, VNULL, srad);
 temp_Pos[0] = tPos[0];

 /* Shift by pos -/+ on y */
    temp_Pos[1] -= dpos;
    ayb1 = Vacc_totalSAV(thee,clist, VNULL, srad);
 temp_Pos[1] = tPos[1];

    temp_Pos[1] += dpos;
    ayt1 = Vacc_totalSAV(thee,clist, VNULL, srad);
 temp_Pos[1] = tPos[1];

 /* Shift by pos -/+ on z */
    temp_Pos[2] -= dpos;
    azb1 = Vacc_totalSAV(thee,clist, VNULL, srad);
 temp_Pos[2] = tPos[2];

    temp_Pos[2] += dpos;
    azt1 = Vacc_totalSAV(thee,clist, VNULL, srad);
 temp_Pos[2] = tPos[2];

 /* Calculate the final value */
    dSA[0] = (axt1-axb1)/(2.0 * dpos);
    dSA[1] = (ayt1-ayb1)/(2.0 * dpos);
    dSA[2] = (azt1-azb1)/(2.0 * dpos);
}

VPUBLIC double Vacc_totalSAV(Vacc *thee, Vclist *clist, APOLparm *apolparm, double radius) {

 int i;
 int npts[3];

 double spacs[3], vec[3];
    double w, wx, wy, wz, len, fn, x, y, z, vol;
 double vol_density,sav;
    double *lower_corner, *upper_corner;

 sav = 0.0;
 vol = 1.0;
 vol_density = 2.0;

 lower_corner = clist->lower_corner;
 upper_corner = clist->upper_corner;

 for (i=0; i<3; i++) {
  len = upper_corner[i] - lower_corner[i];
  vol *= len;
  fn = len*vol_density + 1;
  npts[i] = (int)ceil(fn);
```

```
  spacs[i] = len/((double)(npts[i])-1.0);
  if (apolparm != VNULL) {
   if (apolparm->setgrid) {
    if (apolparm->grid[i] > spacs[i]) {
     Vnm_print(2, "Vacc_totalSAV:  Warning, your GRID value (%g) is larger than the recommended value (%g)!\n",
      apolparm->grid[i], spacs[i]);
    }
    spacs[i] = apolparm->grid[i];

   }
  }
 }

 for (x=lower_corner[0]; x<=upper_corner[0]; x=x+spacs[0]) {
  if ( VABS(x - lower_corner[0]) < VSMALL) {
   wx = 0.5;
  } else if ( VABS(x - upper_corner[0]) < VSMALL) {
   wx = 0.5;
  } else {
   wx = 1.0;
  }
  vec[0] = x;
  for (y=lower_corner[1]; y<=upper_corner[1]; y=y+spacs[1]) {
   if ( VABS(y - lower_corner[1]) < VSMALL) {
    wy = 0.5;
   } else if ( VABS(y - upper_corner[1]) < VSMALL) {
    wy = 0.5;
   } else {
    wy = 1.0;
   }
   vec[1] = y;
   for (z=lower_corner[2]; z<=upper_corner[2]; z=z+spacs[2]) {
    if ( VABS(z - lower_corner[2]) < VSMALL) {
     wz = 0.5;
    } else if ( VABS(z - upper_corner[2]) < VSMALL) {
     wz = 0.5;
    } else {
     wz = 1.0;
    }
    vec[2] = z;

    w = wx*wy*wz;

    sav += (w*(1.0-Vacc_ivdwAcc(thee, vec, radius)));

   } /* z loop */
  } /* y loop */
 } /* x loop */

 w  = spacs[0]*spacs[1]*spacs[2];
 sav *= w;

 return sav;
}

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

 int i;
 int npts[3];
 int pad = 14;

        int xmin, ymin, zmin;
        int xmax, ymax, zmax;

        double sigma6, sigma12;

 double spacs[3], vec[3];
    double w, wx, wy, wz, len, fn, x, y, z, vol;
 double x2,y2,z2,r;
 double vol_density, energy, rho, srad;
 double psig, epsilon, watepsilon, sigma, watsigma, eni, chi;

 double *pos;
    double *lower_corner, *upper_corner;

 Vatom *atom = VNULL;
 VASSERT(apolparm != VNULL);

```

```
energy = 0.0;
vol = 1.0;
vol_density = 2.0;

lower_corner = clist->lower_corner;
upper_corner = clist->upper_corner;

atom = Valist_getAtom(alist, iatom);
pos = Vatom_getPosition(atom);

/* Note:  these are the original temporary water parameters... they have been
 replaced by entries in a parameter file:
watsigma = 1.7683;
watepsilon =  0.1521;
watepsilon = watepsilon*4.184;
*/

srad = apolparm->srad;
rho = apolparm->bconc;
watsigma = apolparm->watsigma;
watepsilon = apolparm->watepsilon;
psig = atom->radius;
epsilon = atom->epsilon;
sigma = psig + watsigma;
   epsilon = VSQRT((epsilon * watepsilon));

/* parameters */
   sigma6 = VPOW(sigma,6);
   sigma12 = VPOW(sigma,12);
   /* OPLS-style radius:  double sigmar = sigma*VPOW(2, (1.0/6.0)); */

xmin = pos[0] - pad;
xmax = pos[0] + pad;
ymin = pos[1] - pad;
ymax = pos[1] + pad;
zmin = pos[2] - pad;
zmax = pos[2] + pad;

for (i=0; i<3; i++) {
 len = (upper_corner[i] + pad) - (lower_corner[i] - pad);
 vol *= len;
 fn = len*vol_density + 1;
 npts[i] = (int)ceil(fn);
 spacs[i] = 0.5;
 if (apolparm->setgrid) {
  if (apolparm->grid[i] > spacs[i]) {
   Vnm_print(2, "Vacc_totalSAV:  Warning, your GRID value (%g) is larger than the recommended value (%g)!\n",
    apolparm->grid[i], spacs[i]);
  }
  spacs[i] = apolparm->grid[i];
 }
}

for (x=xmin; x<=xmax; x=x+spacs[0]) {
 if ( VABS(x - xmin) < VSMALL) {
  wx = 0.5;
 } else if ( VABS(x - xmax) < VSMALL) {
  wx = 0.5;
 } else {
  wx = 1.0;
 }
 vec[0] = x;
 for (y=ymin; y<=ymax; y=y+spacs[1]) {
  if ( VABS(y - ymin) < VSMALL) {
   wy = 0.5;
  } else if ( VABS(y - ymax) < VSMALL) {
   wy = 0.5;
  } else {
   wy = 1.0;
  }
  vec[1] = y;
  for (z=zmin; z<=zmax; z=z+spacs[2]) {
   if ( VABS(z - zmin) < VSMALL) {
    wz = 0.5;
   } else if ( VABS(z - zmax) < VSMALL) {
    wz = 0.5;
   } else {
    wz = 1.0;
   }
   vec[2] = z;
```

```
    w = wx*wy*wz;

    chi = Vacc_ivdwAcc(thee, vec, srad);

    if (VABS(chi) > VSMALL) {

     x2 = VSQR(vec[0]-pos[0]);
     y2 = VSQR(vec[1]-pos[1]);
     z2 = VSQR(vec[2]-pos[2]);
     r = VSQRT(x2+y2+z2);

     if (r <= 14 && r >= sigma) {
      eni = chi*rho*epsilon*(-2.0*sigma6/VPOW(r,6)+sigma12/VPOW(r,12));
     }else if (r <= 14){
      eni = -1.0*epsilon*chi*rho;
     }else{
      eni = 0.0;
     }
    }else{
     eni = 0.0;
    }

    energy += eni*w;

   } /* z loop */
  } /* y loop */
 } /* x loop */

 w  = spacs[0]*spacs[1]*spacs[2];
 energy *= w;

 *value = energy;

 return VRC_SUCCESS;
}

VPUBLIC int Vacc_wcaEnergy(Vacc *acc, APOLparm *apolparm, Valist *alist,
        Vclist *clist){

 int iatom;
 int rc = 0;

    double energy = 0.0;
 double tenergy = 0.0;
 double rho = apolparm->bconc;

 /* Do a sanity check to make sure that watepsilon and watsigma are set
  * If not, return with an error. */
 if(apolparm->setwat == 0){
  Vnm_print(2,"Vacc_wcaEnergy: Error. No value was set for watsigma and watepsilon.\n");
  return VRC_FAILURE;
 }

 if (VABS(rho) < VSMALL) {
  apolparm->wcaEnergy = tenergy;
  return 1;
 }

    for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++){
        rc = Vacc_wcaEnergyAtom(acc, apolparm, alist, clist, iatom, &energy);
  if(rc == 0) return 0;

  tenergy += energy;
    }

 apolparm->wcaEnergy = tenergy;

    return VRC_SUCCESS;

}

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

```
                                ){
    int i,
        si,
        npts[3],
        pad = 14,
        xmin,
        ymin,
        zmin,
        xmax,
        ymax,
        zmax;

    double sigma6,
           sigma12,
           spacs[3],
           vec[3],
           fpt[3],
           w,
           wx,
           wy,
           wz,
           len,
           fn,
           x,
           y,
           z,
           vol,
           x2,
           y2,
           z2,
           r,
           vol_density,
           fo,
           rho,
           srad,
           psig,
           epsilon,
           watepsilon,
           sigma,
           watsigma,
           chi,
           *pos,
           *lower_corner,
           *upper_corner;

    /* Allocate needed variables now that we've asserted required conditions. */
    time_t ts;
    ts = clock();

 VASSERT(apolparm != VNULL);

 /* Do a sanity check to make sure that watepsilon and watsigma are set
  * If not, return with an error. */
 if(apolparm->setwat == 0){
  Vnm_print(2,"Vacc_wcaEnergy: Error. No value was set for watsigma and watepsilon.\n");
  return VRC_FAILURE;
 }

 vol = 1.0;
 vol_density = 2.0;

 lower_corner = clist->lower_corner;
 upper_corner = clist->upper_corner;

 pos = Vatom_getPosition(atom);

 srad = apolparm->srad;
 rho = apolparm->bconc;
 watsigma = apolparm->watsigma;
 watepsilon = apolparm->watepsilon;

 psig = atom->radius;
 epsilon = atom->epsilon;
 sigma = psig + watsigma;
    epsilon = VSQRT((epsilon * watepsilon));

 /* parameters */
    sigma6 = VPOW(sigma,6);
    sigma12 = VPOW(sigma,12);
```

```
    /* OPLS-style radius:  double sigmar = sigma*VPOW(2, (1.0/6.0));  */

 for (i=0; i<3; i++) {
  len = (upper_corner[i] + pad) - (lower_corner[i] - pad);
  vol *= len;
  fn = len*vol_density + 1;
  npts[i] = (int)ceil(fn);
  spacs[i] = 0.5;
  force[i] = 0.0;
  if (apolparm->setgrid) {
   if (apolparm->grid[i] > spacs[i]) {
    Vnm_print(2, "Vacc_totalSAV:  Warning, your GRID value (%g) is larger than the recommended value (%g)!\n",
     apolparm->grid[i], spacs[i]);
   }
   spacs[i] = apolparm->grid[i];
  }
 }

 xmin = pos[0] - pad;
 xmax = pos[0] + pad;
 ymin = pos[1] - pad;
 ymax = pos[1] + pad;
 zmin = pos[2] - pad;
 zmax = pos[2] + pad;

 for (x=xmin; x<=xmax; x=x+spacs[0]) {
  if ( VABS(x - xmin) < VSMALL) {
   wx = 0.5;
  } else if ( VABS(x - xmax) < VSMALL) {
   wx = 0.5;
  } else {
   wx = 1.0;
  }
  vec[0] = x;
  for (y=ymin; y<=ymax; y=y+spacs[1]) {
   if ( VABS(y - ymin) < VSMALL) {
    wy = 0.5;
   } else if ( VABS(y - ymax) < VSMALL) {
    wy = 0.5;
   } else {
    wy = 1.0;
   }
   vec[1] = y;
   for (z=zmin; z<=zmax; z=z+spacs[2]) {
    if ( VABS(z - zmin) < VSMALL) {
     wz = 0.5;
    } else if ( VABS(z - zmax) < VSMALL) {
     wz = 0.5;
    } else {
     wz = 1.0;
    }
    vec[2] = z;

    w = wx*wy*wz;

    chi = Vacc_ivdwAcc(thee, vec, srad);

    if (chi != 0.0) {

     x2 = VSQR(vec[0]-pos[0]);
     y2 = VSQR(vec[1]-pos[1]);
     z2 = VSQR(vec[2]-pos[2]);
     r = VSQRT(x2+y2+z2);

     if (r <= 14 && r >= sigma){

      fo = 12.0*chi*rho*epsilon*(sigma6/VPOW(r,7)-sigma12/VPOW(r,13));

      fpt[0] = -1.0*(pos[0]-vec[0])*fo/r;
      fpt[1] = -1.0*(pos[1]-vec[1])*fo/r;
      fpt[2] = -1.0*(pos[2]-vec[2])*fo/r;

     }else {
      for (si=0; si < 3; si++) fpt[si] = 0.0;
     }
    }else {
     for (si=0; si < 3; si++) fpt[si] = 0.0;
    }

    for(i=0;i<3;i++){
```

```
      force[i] += (w*fpt[i]);
     }

    } /* z loop */
   } /* y loop */
  } /* x loop */

  w  = spacs[0]*spacs[1]*spacs[2];
  for(i=0;i<3;i++) force[i] *= w;

  return VRC_SUCCESS;
 }

```

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)

  Get actual array of atom objects from the list.
- VPUBLIC int Valist_getNumberAtoms (Valist *thee)

  Get number of atoms in the list.
- VPUBLIC Vatom * Valist_getAtom (Valist *thee, int i)

  Get pointer to particular atom in list.
- VPUBLIC unsigned long int Valist_memChk (Valist *thee)

  Get total memory allocated for this object and its members.
- VPUBLIC Valist * Valist_ctor ()

  Construct the atom list object.
- VPUBLIC Vrc_Codes Valist_ctor2 (Valist *thee)

  FORTRAN stub to construct the atom list object.
- VPUBLIC void Valist_dtor (Valist **thee)

  Destroys atom list object.
- VPUBLIC void Valist_dtor2 (Valist *thee)

  FORTRAN stub to destroy atom list object.
- VPRIVATE Vrc_Codes **Valist_readPDBSerial** (Valist *thee, Vio *sock, int *serial)
- VPRIVATE Vrc_Codes **Valist_readPDBAtomName** (Valist *thee, Vio *sock, char atomName[VMAX_ARGLEN])
- VPRIVATE Vrc_Codes **Valist_readPDBResidueName** (Valist *thee, Vio *sock, char resName[VMAX_ARGLEN])
- VPRIVATE Vrc_Codes **Valist_readPDBResidueNumber** (Valist *thee, Vio *sock, int *resSeq)
- VPRIVATE Vrc_Codes **Valist_readPDBAtomCoord** (Valist *thee, Vio *sock, double *coord)
- VPRIVATE Vrc_Codes **Valist_readPDBChargeRadius** (Valist *thee, Vio *sock, double *charge, double *radius)
- VPRIVATE Vrc_Codes **Valist_readPDB_throughXYZ** (Valist *thee, Vio *sock, int *serial, char atomName[VMAX_ARGLEN], char resName[VMAX_ARGLEN], int *resSeq, double *x, double *y, double *z)
- VPRIVATE Vatom * **Valist_getAtomStorage** (Valist *thee, Vatom **plist, int *pnlist, int *pnatoms)
- VPRIVATE Vrc_Codes **Valist_setAtomArray** (Valist *thee, Vatom **plist, int nlist, int natoms)
- VPUBLIC Vrc_Codes Valist_readPDB (Valist *thee, Vparam *param, Vio *sock)

  Fill atom list with information from a PDB file.
- VPUBLIC Vrc_Codes Valist_readPQR (Valist *thee, Vparam *params, Vio *sock)

  Fill atom list with information from a PQR file.
- VPUBLIC Vrc_Codes Valist_readXML (Valist *thee, Vparam *params, Vio *sock)

  Fill atom list with information from an XML file.
- VPUBLIC Vrc_Codes Valist_getStatistics (Valist *thee)

  Load up Valist with various statistics.

Variables

- VPRIVATE char * **Valist_whiteChars** = " \t\r\n"
- VPRIVATE char * **Valist_commChars** = "#%"
- VPRIVATE char * **Valist_xmlwhiteChars** = " \t\r\n<>"

9.65.1 Detailed Description

Class Valist methods.

Author

Nathan Baker

Version

Id:

valist.c 1774 2012-07-19 18:46:14Z sobolevnrm

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

```
#include "apbscfg.h"
#include "apbs/valist.h"

VEMBED(rcsid="$Id: valist.c 1774 2012-07-19 18:46:14Z sobolevnrm $")

VPRIVATE char *Valist_whiteChars = " \t\r\n";
VPRIVATE char *Valist_commChars  = "#%";
VPRIVATE char *Valist_xmlwhiteChars = " \t\r\n<>";

#if !defined(VINLINE_VATOM)

VPUBLIC double Valist_getCenterX(Valist *thee) {

    if (thee == NULL) {
  Vnm_print(2, "Valist_getCenterX:  Found null pointer when getting the center of X coordinate!\n");
  VASSERT(0);
  }
    return thee->center[0];

}

VPUBLIC double Valist_getCenterY(Valist *thee) {

    if (thee == NULL) {
  Vnm_print(2, "Valist_getCenterY:  Found null pointer when getting the center of Y coordinate!\n");
  VASSERT(0);
  }
    return thee->center[1];

}
VPUBLIC double Valist_getCenterZ(Valist *thee) {

    if (thee == NULL) {
  Vnm_print(2, "Valist_getCenterZ:  Found null pointer when getting the center of Z coordinate!\n");
  VASSERT(0);
  }
    return thee->center[2];

}

VPUBLIC Vatom* Valist_getAtomList(Valist *thee) {

    if (thee == NULL) {
  Vnm_print(2, "Valist_getAtomList:  Found null pointer when getting the atom list!\n");
  VASSERT(0);
  }
    return thee->atoms;

}

VPUBLIC int Valist_getNumberAtoms(Valist *thee) {

    if (thee == NULL) {
  Vnm_print(2, "Valist_getNumberAtoms:  Found null pointer when getting the number of atoms!\n");
  VASSERT(0);
  }
    return thee->number;

}

VPUBLIC Vatom* Valist_getAtom(Valist *thee, int i) {

    if (thee == NULL) {
  Vnm_print(2, "Valist_getAtom:  Found null pointer when getting atoms!\n");
  VASSERT(0);
  }
    if (i >= thee->number) {
  Vnm_print(2, "Valist_getAtom:  Requested atom number (%d) outside of atom list range (%d)!\n", i, thee->number);
  VASSERT(0);
  }
    return &(thee->atoms[i]);
```

```
}

VPUBLIC unsigned long int Valist_memChk(Valist *thee) {

    if (thee == NULL) return 0;
    return Vmem_bytes(thee->vmem);

}

#endif /* if !defined(VINLINE_VATOM) */

VPUBLIC Valist* Valist_ctor() {

    /* Set up the structure */
    Valist *thee = VNULL;
    thee = (Valist*)Vmem_malloc(VNULL, 1, sizeof(Valist));
    if ( thee == VNULL) {
  Vnm_print(2, "Valist_ctor:  Got NULL pointer when constructing the atom list object!\n");
  VASSERT(0);
  }
    if ( Valist_ctor2(thee) != VRC_SUCCESS) {
  Vnm_print(2, "Valist_ctor:   Error in constructing the atom list object!\n");
  VASSERT(0);
  }

    return thee;
}

VPUBLIC Vrc_Codes Valist_ctor2(Valist *thee) {

    thee->atoms = VNULL;
    thee->number = 0;

    /* Initialize the memory management object */
    thee->vmem = Vmem_ctor("APBS:VALIST");

    return VRC_SUCCESS;

}

VPUBLIC void Valist_dtor(Valist **thee)
{
    if ((*thee) != VNULL) {
        Valist_dtor2(*thee);
        Vmem_free(VNULL, 1, sizeof(Valist), (void **)thee);
        (*thee) = VNULL;
    }
}

VPUBLIC void Valist_dtor2(Valist *thee) {

    Vmem_free(thee->vmem, thee->number, sizeof(Vatom), (void **)&(thee->atoms));
    thee->atoms = VNULL;
    thee->number = 0;

    Vmem_dtor(&(thee->vmem));
}

/* Read serial number from PDB ATOM/HETATM field */
VPRIVATE Vrc_Codes Valist_readPDBSerial(Valist *thee, Vio *sock, int *serial) {

    char tok[VMAX_BUFSIZE];
    int ti = 0;

    if (Vio_scanf(sock, "%s", tok) != 1) {
        Vnm_print(2, "Valist_readPDB:  Ran out of tokens while parsing serial!\n");
        return VRC_FAILURE;
    }
    if (sscanf(tok, "%d", &ti) != 1) {
        Vnm_print(2, "Valist_readPDB:  Unable to parse serial token (%s) as int!\n",
                tok);
        return VRC_FAILURE;
    }
    *serial = ti;

```

```
    return VRC_SUCCESS;
}

/* Read atom name from PDB ATOM/HETATM field */
VPRIVATE Vrc_Codes Valist_readPDBAtomName(Valist *thee, Vio *sock,
        char atomName[VMAX_ARGLEN]) {

    char tok[VMAX_BUFSIZE];

    if (Vio_scanf(sock, "%s", tok) != 1) {
        Vnm_print(2, "Valist_readPDB:  Ran out of tokens while parsing atom name!\n");
        return VRC_FAILURE;
    }
    if (strlen(tok) < VMAX_ARGLEN) strcpy(atomName, tok);
    else {
        Vnm_print(2, "Valist_readPDB:  Atom name (%s) too long!\n", tok);
        return VRC_FAILURE;
    }
    return VRC_SUCCESS;
}

/* Read residue name from PDB ATOM/HETATM field */
VPRIVATE Vrc_Codes Valist_readPDBResidueName(Valist *thee, Vio *sock,
        char resName[VMAX_ARGLEN]) {

    char tok[VMAX_BUFSIZE];

    if (Vio_scanf(sock, "%s", tok) != 1) {
        Vnm_print(2, "Valist_readPDB:  Ran out of tokens while parsing residue name!\n");
        return VRC_FAILURE;
    }
    if (strlen(tok) < VMAX_ARGLEN) strcpy(resName, tok);
    else {
        Vnm_print(2, "Valist_readPDB:  Residue name (%s) too long!\n", tok);
        return VRC_FAILURE;
    }
    return VRC_SUCCESS;
}

/* Read residue number from PDB ATOM/HETATM field */
VPRIVATE Vrc_Codes Valist_readPDBResidueNumber(
        Valist *thee, Vio *sock, int *resSeq) {

    char tok[VMAX_BUFSIZE];
 char *resstring;
    int ti = 0;

    if (Vio_scanf(sock, "%s", tok) != 1) {
        Vnm_print(2, "Valist_readPDB:  Ran out of tokens while parsing resSeq!\n");
        return VRC_FAILURE;
    }
    if (sscanf(tok, "%d", &ti) != 1) {

     /* One of three things can happen here:
   1)  There is a chainID in the line:    THR A   1
   2)  The chainID is merged with resSeq: THR A1001
   3)  An actual error:                   THR foo

  */

  if (strlen(tok) == 1) {
   /* Case 1: Chain ID Present
                        Read the next field and hope its a float */

   if (Vio_scanf(sock, "%s", tok) != 1) {
          Vnm_print(2, "Valist_readPDB:  Ran out of tokens while parsing resSeq!\n");
          return VRC_FAILURE;
      }
      if (sscanf(tok, "%d", &ti) != 1) {
    Vnm_print(2, "Valist_readPDB:  Unable to parse resSeq token (%s) as int!\n",
                tok);
          return VRC_FAILURE;
   }

  } else {
   /* Case 2: Chain ID, merged string.
```

```
        Move pointer forward past the chainID and check
         */
      //strcpy(resstring, tok);
   resstring = tok;
   resstring++;

   if (sscanf(resstring, "%d", &ti) != 1) {
    /* Case 3:  More than one non-numeral char is present. Error.*/
                Vnm_print(2, "Valist_readPDB:  Unable to parse resSeq token (%s) as int!\n",
                resstring);
            return VRC_FAILURE;
   }
  }
    }
    *resSeq = ti;

    return VRC_SUCCESS;
}

/* Read atom coordinate from PDB ATOM/HETATM field */
VPRIVATE Vrc_Codes Valist_readPDBAtomCoord(Valist *thee, Vio *sock, double *coord) {

    char tok[VMAX_BUFSIZE];
    double tf = 0;

    if (Vio_scanf(sock, "%s", tok) != 1) {
        Vnm_print(2, "Valist_readPDB:  Ran out of tokens while parsing atom coordinate!\n");
        return VRC_FAILURE;
    }
    if (sscanf(tok, "%lf", &tf) != 1) {
        return VRC_FAILURE;
    }
    *coord = tf;

    return VRC_SUCCESS;
}

/* Read charge and radius from PQR ATOM/HETATM field */
VPRIVATE Vrc_Codes Valist_readPDBChargeRadius(Valist *thee, Vio *sock,
        double *charge, double *radius) {

    char tok[VMAX_BUFSIZE];
    double tf = 0;

    if (Vio_scanf(sock, "%s", tok) != 1) {
        Vnm_print(2, "Valist_readPQR:  Ran out of tokens while parsing charge!\n");
        return VRC_FAILURE;
    }
    if (sscanf(tok, "%lf", &tf) != 1) {
        return VRC_FAILURE;
    }
    *charge = tf;

    if (Vio_scanf(sock, "%s", tok) != 1) {
        Vnm_print(2, "Valist_readPQR:  Ran out of tokens while parsing radius!\n");
        return VRC_FAILURE;
    }
    if (sscanf(tok, "%lf", &tf) != 1) {
        return VRC_FAILURE;
    }
    *radius = tf;

    return VRC_SUCCESS;
}

/* Read ATOM/HETATM field of PDB through the X/Y/Z fields */
VPRIVATE Vrc_Codes Valist_readPDB_throughXYZ(
        Valist *thee,
        Vio *sock, /* Socket ready for reading */
        int *serial, /* Set to atom number */
        char atomName[VMAX_ARGLEN], /* Set to atom name */
        char resName[VMAX_ARGLEN], /* Set to residue name */
        int *resSeq, /* Set to residue number */
        double *x, /* Set to x-coordinate */
        double *y, /* Set to y-coordinate */
        double *z  /* Set to z-coordinate */
```

```
        ) {


    int i, njunk, gotit;

    /* Grab serial */
    if (Valist_readPDBSerial(thee, sock, serial) == VRC_FAILURE) {
        Vnm_print(2, "Valist_readPDB:  Error while parsing serial!\n");
    }

    /* Grab atom name */
    if (Valist_readPDBAtomName(thee, sock, atomName) == VRC_FAILURE) {
        Vnm_print(2, "Valist_readPDB:  Error while parsing atom name!\n");
        return VRC_FAILURE;
    }

    /* Grab residue name */
    if (Valist_readPDBResidueName(thee, sock, resName) == VRC_FAILURE) {
        Vnm_print(2, "Valist_readPDB:  Error while parsing residue name!\n");
        return VRC_FAILURE;
    }


    /* Grab residue number */
    if (Valist_readPDBResidueNumber(thee, sock, resSeq) == VRC_FAILURE) {
        Vnm_print(2, "Valist_readPDB:  Error while parsing residue name!\n");
        return VRC_FAILURE;
    }


    /* Read tokens until we find one that can be parsed as an atom
     * x-coordinate.  We will allow njunk=1 intervening field that
     * cannot be parsed as a coordinate */
    njunk = 1;
    gotit = 0;
    for (i=0; i<(njunk+1); i++) {
        if (Valist_readPDBAtomCoord(thee, sock, x) == VRC_SUCCESS) {
            gotit = 1;
            break;
        }
    }
    if (!gotit) {
        Vnm_print(2, "Valist_readPDB:  Can't find x!\n");
        return VRC_FAILURE;
    }
    /* Read y-coordinate */
    if (Valist_readPDBAtomCoord(thee, sock, y) == VRC_FAILURE) {
        Vnm_print(2, "Valist_readPDB:  Can't find y!\n");
        return VRC_FAILURE;
    }
    /* Read z-coordinate */
    if (Valist_readPDBAtomCoord(thee, sock, z) == VRC_FAILURE) {
        Vnm_print(2, "Valist_readPDB:  Can't find z!\n");
        return VRC_FAILURE;
    }

#if 0 /* Set to 1 if you want to debug */
    Vnm_print(1, "Valist_readPDB:  serial = %d\n", *serial);
    Vnm_print(1, "Valist_readPDB:  atomName = %s\n", atomName);
    Vnm_print(1, "Valist_readPDB:  resName = %s\n", resName);
    Vnm_print(1, "Valist_readPDB:  resSeq = %d\n", *resSeq);
    Vnm_print(1, "Valist_readPDB:  pos = (%g, %g, %g)\n",
              *x, *y, *z);
#endif

    return VRC_SUCCESS;
}

/* Get a the next available atom storage location, increasing the storage
 * space if necessary.  Return VNULL if something goes wrong. */
VPRIVATE Vatom* Valist_getAtomStorage(
        Valist *thee,
        Vatom **plist, /* Pointer to existing list of atoms */
        int *pnlist, /* Size of existing list, may be changed */
        int *pnatoms /* Existing number of atoms in list; incremented
                        before exit */
        ) {

```

```
    Vatom *oldList, *newList, *theList;
    Vatom *oldAtom, *newAtom;
    int iatom, inext, oldLength, newLength, natoms;

    newList = VNULL;

    /* See if we need more space */
    if (*pnatoms >= *pnlist) {

        /* Double the storage space */
        oldLength = *pnlist;
        newLength = 2*oldLength;
        newList = (Vatom*)Vmem_malloc(thee->vmem, newLength, sizeof(Vatom));
        oldList = *plist;

        /* Check the allocation */
        if (newList == VNULL) {
            Vnm_print(2, "Valist_readPDB:  failed to allocate space for %d (Vatom)s!\n", newLength);
            return VNULL;
        }

        /* Copy the atoms over */
        natoms = *pnatoms;
        for (iatom=0; iatom<natoms; iatom++) {
            oldAtom = &(oldList[iatom]);
            newAtom = &(newList[iatom]);
            Vatom_copyTo(oldAtom, newAtom);
            Vatom_dtor2(oldAtom);
        }

        /* Free the old list */
        Vmem_free(thee->vmem, oldLength, sizeof(Vatom), (void **)plist);

        /* Copy new list to plist */
        *plist = newList;
        *pnlist = newLength;
    }

    theList = *plist;
    inext = *pnatoms;

    /* Get the next available spot and increment counters */
    newAtom = &(theList[inext]);
    *pnatoms = inext + 1;

    return newAtom;
}

VPRIVATE Vrc_Codes Valist_setAtomArray(Valist *thee,
        Vatom **plist, /* Pointer to list of atoms to store */
        int nlist, /* Length of list */
        int natoms /* Number of real atom entries in list */
        ) {

    Vatom *list, *newAtom, *oldAtom;
    int i;

    list = *plist;

    /* Allocate necessary space */
    thee->number = 0;
    thee->atoms = (Vatom*)Vmem_malloc(thee->vmem, natoms, sizeof(Vatom));
    if (thee->atoms == VNULL) {
        Vnm_print(2, "Valist_readPDB:  Unable to allocate space for %d (Vatom)s!\n",
                  natoms);
        return VRC_FAILURE;
    }
    thee->number = natoms;

    /* Copy over data */
    for (i=0; i<thee->number; i++) {
        newAtom = &(thee->atoms[i]);
        oldAtom = &(list[i]);
        Vatom_copyTo(oldAtom, newAtom);
        Vatom_dtor2(oldAtom);
    }
```

```c

    /* Free old array */
    Vmem_free(thee->vmem, nlist, sizeof(Vatom), (void **)plist);

    return VRC_SUCCESS;
}

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

    /* WE DO NOT DIRECTLY CONFORM TO PDB STANDARDS -- TO ALLOW LARGER FILES, WE
     * REQUIRE ALL FIELDS TO BE WHITESPACE DELIMITED */

    Vatom *atoms = VNULL;
    Vatom *nextAtom = VNULL;
    Vparam_AtomData *atomData = VNULL;

    char tok[VMAX_BUFSIZE];
    char atomName[VMAX_ARGLEN], resName[VMAX_ARGLEN];

 int nlist, natoms, serial, resSeq;

 double x, y, z, charge, radius, epsilon;
    double pos[3];

    if (thee == VNULL) {
  Vnm_print(2, "Valist_readPDB:  Got NULL pointer when reading PDB file!\n");
  VASSERT(0);
 }
    thee->number = 0;

    Vio_setWhiteChars(sock, Valist_whiteChars);
    Vio_setCommChars(sock, Valist_commChars);

    /* Allocate some initial space for the atoms */
    nlist = 200;
    atoms = (Vatom*)Vmem_malloc(thee->vmem, nlist, sizeof(Vatom));

    natoms = 0;
    /* Read until we run out of lines */
    while (Vio_scanf(sock, "%s", tok) == 1) {

        /* Parse only ATOM/HETATOM fields */
        if ((Vstring_strcasecmp(tok, "ATOM") == 0) ||
            (Vstring_strcasecmp(tok, "HETATM") == 0)) {

            /* Read ATOM/HETATM field of PDB through the X/Y/Z fields */
            if (Valist_readPDB_throughXYZ(thee, sock, &serial, atomName,
                        resName, &resSeq, &x, &y, &z) == VRC_FAILURE) {
                Vnm_print(2, "Valist_readPDB:  Error parsing atom %d!\n",
                          serial);
                return VRC_FAILURE;
            }

            /* Try to find the parameters. */
            atomData = Vparam_getAtomData(param, resName, atomName);
            if (atomData == VNULL) {
                Vnm_print(2, "Valist_readPDB:  Couldn't find parameters for \
atom = %s, residue = %s\n", atomName, resName);
                return VRC_FAILURE;
            }
            charge = atomData->charge;
            radius = atomData->radius;
   epsilon = atomData->epsilon;

            /* Get pointer to next available atom position */
            nextAtom = Valist_getAtomStorage(thee, &atoms, &nlist, &natoms);
            if (nextAtom == VNULL) {
                Vnm_print(2, "Valist_readPDB:  Error in allocating spacing for atoms!\n");
                return VRC_FAILURE;
            }

            /* Store the information */
            pos[0] = x; pos[1] = y; pos[2] = z;
            Vatom_setPosition(nextAtom, pos);
            Vatom_setCharge(nextAtom, charge);
            Vatom_setRadius(nextAtom, radius);
```

```
    Vatom_setEpsilon(nextAtom, epsilon);
            Vatom_setAtomID(nextAtom, natoms-1);
    Vatom_setResName(nextAtom, resName);
            Vatom_setAtomName(nextAtom, atomName);

        } /* if ATOM or HETATM */
    } /* while we haven't run out of tokens */

    Vnm_print(0, "Valist_readPDB: Counted %d atoms\n", natoms);
    fflush(stdout);

    /* Store atoms internally */
    if (Valist_setAtomArray(thee, &atoms, nlist, natoms) == VRC_FAILURE
) {
        Vnm_print(2, "Valist_readPDB:  unable to store atoms!\n");
        return VRC_FAILURE;
    }

    return Valist_getStatistics(thee);


}

VPUBLIC Vrc_Codes Valist_readPQR(Valist *thee, Vparam
 *params, Vio *sock) {

    /* WE DO NOT DIRECTLY CONFORM TO PDB STANDARDS -- TO ALLOW LARGER FILES, WE
     * REQUIRE ALL FIELDS TO BE WHITESPACE DELIMITED */


    Vatom *atoms = VNULL;
    Vatom *nextAtom = VNULL;
 Vparam_AtomData *atomData = VNULL;

    char tok[VMAX_BUFSIZE];
    char atomName[VMAX_ARGLEN], resName[VMAX_ARGLEN];

 int use_params = 0;
 int nlist, natoms, serial, resSeq;

    double x, y, z, charge, radius, epsilon;
    double pos[3];

 epsilon = 0.0;

    if (thee == VNULL) {
  Vnm_print(2, "Valist_readPQR:  Got NULL pointer when reading PQR file!\n");
  VASSERT(0);
 }
    thee->number = 0;

    Vio_setWhiteChars(sock, Valist_whiteChars);
    Vio_setCommChars(sock, Valist_commChars);

    /* Allocate some initial space for the atoms */
    nlist = 200;
    atoms = (Vatom*)Vmem_malloc(thee->vmem, nlist, sizeof(Vatom))
;

 /* Check if we are using a parameter file or not */
 if(params != VNULL) use_params = 1;

    natoms = 0;
    /* Read until we run out of lines */
    while (Vio_scanf(sock, "%s", tok) == 1) {

        /* Parse only ATOM/HETATOM fields */
        if ((Vstring_strcasecmp(tok, "ATOM") == 0) ||
            (Vstring_strcasecmp(tok, "HETATM") == 0)) {

            /* Read ATOM/HETATM field of PDB through the X/Y/Z fields */
            if (Valist_readPDB_throughXYZ(thee, sock, &serial, atomName,
                        resName, &resSeq, &x, &y, &z) == VRC_FAILURE
) {
                Vnm_print(2, "Valist_readPQR:  Error parsing atom %d!\n",serial
);
                Vnm_print(2, "Please double check this atom in the pqr file,
 e.g., make sure there are no concatenated fields.\n");
                return VRC_FAILURE;
            }

```

```
            /* Read Q/R fields */
            if (Valist_readPDBChargeRadius(thee, sock, &charge, &radius) == VRC_FAILURE) {
                Vnm_print(2, "Valist_readPQR:  Error parsing atom %d!\n",
                          serial);
                Vnm_print(2, "Please double check this atom in the pqr file, e.g., make sure there are no concatenated fields.\n");
                return VRC_FAILURE;
            }

   if(use_params){
    /* Try to find the parameters. */
    atomData = Vparam_getAtomData(params, resName, atomName);
    if (atomData == VNULL) {
     Vnm_print(2, "Valist_readPDB:  Couldn't find parameters for \
atom = %s, residue = %s\n", atomName, resName);
     return VRC_FAILURE;
    }
    charge = atomData->charge;
    radius = atomData->radius;
    epsilon = atomData->epsilon;
   }

            /* Get pointer to next available atom position */
            nextAtom = Valist_getAtomStorage(thee, &atoms, &nlist, &natoms);
            if (nextAtom == VNULL) {
                Vnm_print(2, "Valist_readPQR:  Error in allocating spacing for atoms!\n");
                return VRC_FAILURE;
            }

            /* Store the information */
            pos[0] = x; pos[1] = y; pos[2] = z;
            Vatom_setPosition(nextAtom, pos);
            Vatom_setCharge(nextAtom, charge);
            Vatom_setRadius(nextAtom, radius);
   Vatom_setEpsilon(nextAtom, epsilon);
            Vatom_setAtomID(nextAtom, natoms-1);
   Vatom_setResName(nextAtom, resName);
            Vatom_setAtomName(nextAtom, atomName);

        } /* if ATOM or HETATM */
    } /* while we haven't run out of tokens */

    Vnm_print(0, "Valist_readPQR: Counted %d atoms\n", natoms);
    fflush(stdout);

    /* Store atoms internally */
    if (Valist_setAtomArray(thee, &atoms, nlist, natoms) == VRC_FAILURE) {
        Vnm_print(2, "Valist_readPDB:  unable to store atoms!\n");
        return VRC_FAILURE;
    }

    return Valist_getStatistics(thee);


}

VPUBLIC Vrc_Codes Valist_readXML(Valist *thee, Vparam *params, Vio *sock) {

    Vatom *atoms = VNULL;
    Vatom *nextAtom = VNULL;

    char tok[VMAX_BUFSIZE];
    char endtag[VMAX_BUFSIZE];

 int nlist, natoms;
    int xset, yset, zset, chgset, radset;

 double x, y, z, charge, radius, dtmp;
    double pos[3];

    if (thee == VNULL) {
  Vnm_print(2, "Valist_readXML:  Got NULL pointer when reading XML file!\n");
  VASSERT(0);
 }
    thee->number = 0;

    Vio_setWhiteChars(sock, Valist_xmlwhiteChars);
```

```
    Vio_setCommChars(sock, Valist_commChars);

    /* Allocate some initial space for the atoms */
    nlist = 200;
    atoms = (Vatom*)Vmem_malloc(thee->vmem, nlist, sizeof(Vatom))
;

    /* Initialize some variables */
    natoms = 0;
    xset = 0;
    yset = 0;
    zset = 0;
    chgset = 0;
    radset = 0;
    strcpy(endtag,"/");

 if(params == VNULL){
  Vnm_print(1,"\nValist_readXML: Warning Warning Warning Warning Warning\n");
  Vnm_print(1,"Valist_readXML: The use of XML input files with parameter\n");
  Vnm_print(1,"Valist_readXML: files is currently not supported.\n");
  Vnm_print(1,"Valist_readXML: Warning Warning Warning Warning Warning\n\n");
 }

    /* Read until we run out of lines */
    while (Vio_scanf(sock, "%s", tok) == 1) {

        /* The first tag taken is the start tag - save it to detect end */
        if (Vstring_strcasecmp(endtag, "/") == 0) strcat(
endtag, tok);

        if (Vstring_strcasecmp(tok, "x") == 0) {
            Vio_scanf(sock, "%s", tok);
            if (sscanf(tok, "%lf", &dtmp) != 1) {
                Vnm_print(2, "Valist_readXML:  Unexpected token (%s) while \
reading x!\n", tok);
                    return VRC_FAILURE;
              }
            x = dtmp;
            xset = 1;
        } else if (Vstring_strcasecmp(tok, "y") == 0) {
            Vio_scanf(sock, "%s", tok);
            if (sscanf(tok, "%lf", &dtmp) != 1) {
                Vnm_print(2, "Valist_readXML:  Unexpected token (%s) while \
reading y!\n", tok);
                    return VRC_FAILURE;
              }
            y = dtmp;
            yset = 1;
        } else if (Vstring_strcasecmp(tok, "z") == 0) {
            Vio_scanf(sock, "%s", tok);
            if (sscanf(tok, "%lf", &dtmp) != 1) {
                Vnm_print(2, "Valist_readXML:  Unexpected token (%s) while \
reading z!\n", tok);
                    return VRC_FAILURE;
              }
            z = dtmp;
            zset = 1;
        } else if (Vstring_strcasecmp(tok, "charge") == 0) {
            Vio_scanf(sock, "%s", tok);
            if (sscanf(tok, "%lf", &dtmp) != 1) {
                Vnm_print(2, "Valist_readXML:  Unexpected token (%s) while \
reading charge!\n", tok);
                    return VRC_FAILURE;
              }
            charge = dtmp;
            chgset = 1;
        } else if (Vstring_strcasecmp(tok, "radius") == 0) {
            Vio_scanf(sock, "%s", tok);
            if (sscanf(tok, "%lf", &dtmp) != 1) {
                Vnm_print(2, "Valist_readXML:  Unexpected token (%s) while \
reading radius!\n", tok);
                    return VRC_FAILURE;
              }
            radius = dtmp;
            radset = 1;
        } else if (Vstring_strcasecmp(tok, "/atom") == 0) {

          /* Get pointer to next available atom position */
            nextAtom = Valist_getAtomStorage(thee, &atoms, &nlist, &natoms);
            if (nextAtom == VNULL) {
                Vnm_print(2, "Valist_readXML:  Error in allocating spacing for
```

```
 atoms!\n");
                return VRC_FAILURE;
            }

            if (xset && yset && zset && chgset && radset){

                /* Store the information */
                pos[0] = x; pos[1] = y; pos[2] = z;
                Vatom_setPosition(nextAtom, pos);
                Vatom_setCharge(nextAtom, charge);
                Vatom_setRadius(nextAtom, radius);
                Vatom_setAtomID(nextAtom, natoms-1);

                /* Reset the necessary flags */
                xset = 0;
                yset = 0;
                zset = 0;
                chgset = 0;
                radset = 0;
            } else {
                Vnm_print(2,  "Valist_readXML:  Missing field(s) in atom tag:\n");
                if (!xset) Vnm_print(2,"\tx value not set!\n");
                if (!yset) Vnm_print(2,"\ty value not set!\n");
                if (!zset) Vnm_print(2,"\tz value not set!\n");
                if (!chgset) Vnm_print(2,"\tcharge value not set!\n");
                if (!radset) Vnm_print(2,"\tradius value not set!\n");
                return VRC_FAILURE;
            }
        } else if (Vstring_strcasecmp(tok, endtag) == 0) break;
    }

    Vnm_print(0, "Valist_readXML: Counted %d atoms\n", natoms);
    fflush(stdout);

    /* Store atoms internally */
    if (Valist_setAtomArray(thee, &atoms, nlist, natoms) == VRC_FAILURE) {
        Vnm_print(2, "Valist_readXML:  unable to store atoms!\n");
        return VRC_FAILURE;
    }

    return Valist_getStatistics(thee);

}

/* Load up Valist with various statistics */
VPUBLIC Vrc_Codes Valist_getStatistics(Valist *thee) {

    Vatom *atom;
    int i, j;

    if (thee == VNULL) {
  Vnm_print(2, "Valist_getStatistics:  Got NULL pointer when loading up Valist with various statistics!\n");
  VASSERT(0);
  }

    thee->center[0] = 0.;
    thee->center[1] = 0.;
    thee->center[2] = 0.;
    thee->maxrad = 0.;
    thee->charge = 0.;

    if (thee->number == 0) return VRC_FAILURE;

    /* Reset stat variables */
    atom = &(thee->atoms[0]);
    for (i=0; i<3; i++) {
        thee->maxcrd[i] = thee->mincrd[i] = atom->position[i];
    }
    thee->maxrad = atom->radius;
    thee->charge = 0.0;

    for (i=0; i<thee->number; i++) {

        atom = &(thee->atoms[i]);
        for (j=0; j<3; j++) {
```

```
            if (atom->position[j] < thee->mincrd[j])
              thee->mincrd[j] = atom->position[j];
            if (atom->position[j] > thee->maxcrd[j])
              thee->maxcrd[j] = atom->position[j];
        }
        if (atom->radius > thee->maxrad) thee->maxrad = atom
    ->radius;
        thee->charge = thee->charge + atom->charge;
    }

    thee->center[0] = 0.5*(thee->maxcrd[0] + thee->mincrd[0])
    ;
    thee->center[1] = 0.5*(thee->maxcrd[1] + thee->mincrd[1])
    ;
    thee->center[2] = 0.5*(thee->maxcrd[2] + thee->mincrd[2])
    ;

 Vnm_print(0, "Valist_getStatistics:  Max atom coordinate:  (%g, %g, %g)\n",
     thee->maxcrd[0], thee->maxcrd[1], thee->maxcrd[2]);
 Vnm_print(0, "Valist_getStatistics:  Min atom coordinate:  (%g, %g, %g)\n",
     thee->mincrd[0], thee->mincrd[1], thee->mincrd[2]);
 Vnm_print(0, "Valist_getStatistics:  Molecule center:  (%g, %g, %g)\n",
     thee->center[0], thee->center[1], thee->center[2]);

    return VRC_SUCCESS;
}
```

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 * Vatom_getPosition (Vatom *thee)

  Get atomic position.
- VPUBLIC double Vatom_getPartID (Vatom *thee)

  Get partition ID.
- VPUBLIC void Vatom_setPartID (Vatom *thee, int partID)

  Set partition ID.
- VPUBLIC double Vatom_getAtomID (Vatom *thee)

  Get atom ID.
- VPUBLIC void Vatom_setAtomID (Vatom *thee, int atomID)

  Set atom ID.
- VPUBLIC void Vatom_setRadius (Vatom *thee, double radius)

  Set atomic radius.
- VPUBLIC double Vatom_getRadius (Vatom *thee)

  Get atomic position.
- VPUBLIC void Vatom_setCharge (Vatom *thee, double charge)

  Set atomic charge.
- VPUBLIC double Vatom_getCharge (Vatom *thee)

  Get atomic charge.
- VPUBLIC unsigned long int Vatom_memChk (Vatom *thee)

  Return the memory used by this structure (and its contents) in bytes.
- VPUBLIC Vatom * Vatom_ctor ()

  Constructor for the Vatom class.
- VPUBLIC int Vatom_ctor2 (Vatom *thee)

  FORTRAN stub constructor for the Vatom class.
- VPUBLIC void Vatom_dtor (Vatom **thee)

  Object destructor.
- VPUBLIC void Vatom_dtor2 (Vatom *thee)

  FORTRAN stub object destructor.
- VPUBLIC void Vatom_setPosition (Vatom *thee, double position[3])

  Set the atomic position.
- VPUBLIC void Vatom_copyTo (Vatom *thee, Vatom *dest)

  Copy information to another atom.
- VPUBLIC void Vatom_copyFrom (Vatom *thee, Vatom *src)

  Copy information to another atom.
- VPUBLIC void Vatom_setResName (Vatom *thee, char resName[VMAX_RECLEN])

  Set residue name.
- VPUBLIC void Vatom_getResName (Vatom *thee, char resName[VMAX_RECLEN])

  Retrieve residue name.
- VPUBLIC void Vatom_setAtomName (Vatom *thee, char atomName[VMAX_RECLEN])

  Set atom name.
- VPUBLIC void Vatom_getAtomName (Vatom *thee, char atomName[VMAX_RECLEN])

  Retrieve atom name.

9.67.1 Detailed Description

Class Vatom methods.

Author

Nathan Baker

Version

Id:

vatom.c 1774 2012-07-19 18:46:14Z sobolevnrm

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

```
#include "apbscfg.h"
#include "apbs/vatom.h"

VEMBED(rcsid="$Id: vatom.c 1774 2012-07-19 18:46:14Z sobolevnrm $")

#if !defined(VINLINE_VATOM)

VPUBLIC double *Vatom_getPosition(Vatom *thee) {

   VASSERT(thee != VNULL);
   return thee->position;

}

VPUBLIC double Vatom_getPartID(Vatom *thee) {

   VASSERT(thee != VNULL);
   return thee->partID;

}

VPUBLIC void Vatom_setPartID(Vatom *thee, int partID) {

   VASSERT(thee != VNULL);
   thee->partID = (double)partID;

}

VPUBLIC double Vatom_getAtomID(Vatom *thee) {

   VASSERT(thee != VNULL);
   return thee->atomID;

}

VPUBLIC void Vatom_setAtomID(Vatom *thee, int atomID) {

   VASSERT(thee != VNULL);
   thee->atomID = atomID;

}

VPUBLIC void Vatom_setRadius(Vatom *thee, double radius) {

   VASSERT(thee != VNULL);
   thee->radius = radius;

}

VPUBLIC double Vatom_getRadius(Vatom *thee) {

   VASSERT(thee != VNULL);
   return thee->radius;

}

VPUBLIC void Vatom_setCharge(Vatom *thee, double charge) {

   VASSERT(thee != VNULL);
   thee->charge = charge;

}

VPUBLIC double Vatom_getCharge(Vatom *thee) {

   VASSERT(thee != VNULL);
   return thee->charge;

}

VPUBLIC unsigned long int Vatom_memChk(Vatom *thee) { return sizeof(Vatom); }

#endif /* if !defined(VINLINE_VATOM) */

VPUBLIC Vatom* Vatom_ctor() {

```

```
    /* Set up the structure */
    Vatom *thee = VNULL;
    thee = (Vatom *)Vmem_malloc( VNULL, 1, sizeof(Vatom) );
    VASSERT( thee != VNULL);
    VASSERT( Vatom_ctor2(thee));

    return thee;
}

VPUBLIC int Vatom_ctor2(Vatom *thee) {
    thee->partID = -1;
    return 1;
}

VPUBLIC void Vatom_dtor(Vatom **thee) {
    if ((*thee) != VNULL) {
        Vatom_dtor2(*thee);
        Vmem_free(VNULL, 1, sizeof(Vatom), (void **)thee);
        (*thee) = VNULL;
    }
}

VPUBLIC void Vatom_dtor2(Vatom *thee) { ; }

VPUBLIC void Vatom_setPosition(Vatom *thee, double position[3]) {

   VASSERT(thee != VNULL);
   (thee->position)[0] = position[0];
   (thee->position)[1] = position[1];
   (thee->position)[2] = position[2];

}

VPUBLIC void Vatom_copyTo(Vatom *thee, Vatom *dest) {

    VASSERT(thee != VNULL);
    VASSERT(dest != VNULL);

    memcpy(dest, thee, sizeof(Vatom));

}

VPUBLIC void Vatom_copyFrom(Vatom *thee, Vatom *src) {

    Vatom_copyTo(src, thee);

}

VPUBLIC void Vatom_setResName(Vatom *thee, char resName[VMAX_RECLEN]) {

 VASSERT(thee != VNULL);
 strcpy(thee->resName, resName);

}

VPUBLIC void Vatom_getResName(Vatom *thee, char resName[VMAX_RECLEN]) {


 VASSERT(thee != VNULL);
 strcpy(resName,thee->resName);

}

VPUBLIC void Vatom_setAtomName(Vatom *thee, char atomName[VMAX_RECLEN]) {

 VASSERT(thee != VNULL);
 strcpy(thee->atomName, atomName);

}

VPUBLIC void Vatom_getAtomName(Vatom *thee, char atomName[VMAX_RECLEN]) {

 VASSERT(thee != VNULL);
 strcpy(atomName,thee->atomName);

}
```

```
#if defined(WITH_TINKER)

VPUBLIC void Vatom_setDipole(Vatom *thee, double dipole[3]) {

   VASSERT(thee != VNULL);
   (thee->dipole)[0] = dipole[0];
   (thee->dipole)[1] = dipole[1];
   (thee->dipole)[2] = dipole[2];

}

VPUBLIC void Vatom_setQuadrupole(Vatom *thee, double quadrupole[9]) {

   int i;
   VASSERT(thee != VNULL);
   for (i=0; i<9; i++)  (thee->quadrupole)[i] = quadrupole[i];
}

VPUBLIC void Vatom_setInducedDipole(Vatom *thee, double dipole[3]) {

   VASSERT(thee != VNULL);
   (thee->inducedDipole)[0] = dipole[0];
   (thee->inducedDipole)[1] = dipole[1];
   (thee->inducedDipole)[2] = dipole[2];
}

VPUBLIC void Vatom_setNLInducedDipole(Vatom *thee, double dipole[3]) {

   VASSERT(thee != VNULL);
   (thee->nlInducedDipole)[0] = dipole[0];
   (thee->nlInducedDipole)[1] = dipole[1];
   (thee->nlInducedDipole)[2] = dipole[2];

}

VPUBLIC double *Vatom_getDipole(Vatom *thee) {

   VASSERT(thee != VNULL);
   return thee->dipole;

}

VPUBLIC double *Vatom_getQuadrupole(Vatom *thee) {

   VASSERT(thee != VNULL);
   return thee->quadrupole;

}

VPUBLIC double *Vatom_getInducedDipole(Vatom *thee) {

   VASSERT(thee != VNULL);
   return thee->inducedDipole;

}

VPUBLIC double *Vatom_getNLInducedDipole(Vatom *thee) {

   VASSERT(thee != VNULL);
   return thee->nlInducedDipole;

}

#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 1774 2012-07-19 18:46:14Z sobolevnrm

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     }
```

```
    /* The normal EXP */
    (*ichop) = 0;
    return VEXP(x);
}

VPUBLIC double Vcap_sinh(double x, int *ichop) {

    /* The two chopped arguments */
    if (x > EXPMAX) {
       (*ichop) = 1;
       return VSINH(EXPMAX);
    } else if (x < EXPMIN) {
       (*ichop) = 1;
       return VSINH(EXPMIN);
    }

    /* The normal SINH */
    (*ichop) = 0;
    return VSINH(x);
}

VPUBLIC double Vcap_cosh(double x, int *ichop) {

    /* The two chopped arguments */
    if (x > EXPMAX) {
       (*ichop) = 1;
       return VCOSH(EXPMAX);
    } else if (x < EXPMIN) {
       (*ichop) = 1;
       return VCOSH(EXPMIN);
    }

    /* The normal COSH */
    (*ichop) = 0;
    return VCOSH(x);
}

```

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 1774 2012-07-19 18:46:14Z sobolevnrm

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 1774 2012-07-19 18:46:14Z sobolevnrm $")
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,
00090                 upper_corner) == VRC_SUCCESS );
00091     return thee;
00092 }
00093
```

```
/* Get the dimensions of the molecule stored in thee->alist */
VPRIVATE void Vclist_getMolDims(
        Vclist *thee,
        double lower_corner[VAPBS_DIM], /* Set to lower corner of molecule */
        double upper_corner[VAPBS_DIM], /* Set to lower corner of molecule */
        double *r_max /* Set to max atom radius */
        ) {

    int i, j;
    double pos;
    Valist *alist;
    Vatom *atom;

    alist = thee->alist;

    /* Initialize */
    for (i=0; i<VAPBS_DIM; i++) {
        lower_corner[i] = VLARGE;
        upper_corner[i] = -VLARGE;
    }
    *r_max = -1.0;

    /* Check each atom */
    for (i=0; i<Valist_getNumberAtoms(alist); i++) {
        atom = Valist_getAtom(alist, i);
  for (j=0; j<VAPBS_DIM; j++) {
            pos = (Vatom_getPosition(atom))[j];
            if ( pos < lower_corner[j] ) lower_corner[j] = pos;
            if ( pos > upper_corner[j] ) upper_corner[j] = pos;
        }
        if (Vatom_getRadius(atom) > *r_max) *r_max = Vatom_getRadius(atom);
    }

}

/* Setup lookup grid */
VPRIVATE Vrc_Codes Vclist_setupGrid(Vclist *thee) {

    /* Inflation factor ~ sqrt(2)*/
    #define VCLIST_INFLATE 1.42

    int i;
    double length[VAPBS_DIM], r_max;

    /* Set up the grid corners */
    switch (thee->mode) {
        case CLIST_AUTO_DOMAIN:
            /* Get molecule dimensions */
            Vclist_getMolDims(thee, thee->lower_corner, thee->upper_corner,
                    &r_max);
            /* Set up grid spacings */
            for (i=0; i<VAPBS_DIM; i++) {
                thee->upper_corner[i] = thee->upper_corner[i]
                    + VCLIST_INFLATE*(r_max+thee->max_radius);
                thee->lower_corner[i] = thee->lower_corner[i]
                    - VCLIST_INFLATE*(r_max+thee->max_radius);
            }
            break;
        case CLIST_MANUAL_DOMAIN:
            /* Grid corners established in constructor */
            break;
        default:
            Vnm_print(2, "Vclist_setupGrid:  invalid setup mode (%d)!\n",
                    thee->mode);
            return VRC_FAILURE;
    }

    /* Set up the grid lengths and spacings */
    for (i=0; i<VAPBS_DIM; i++) {
        length[i] = thee->upper_corner[i] - thee->lower_corner[i];
        thee->spacs[i] = length[i]/((double)(thee->npts[i] - 1));
    }
    Vnm_print(0, "Vclist_setupGrid:  Grid lengths = (%g, %g, %g)\n",
            length[0], length[1], length[2]);

```

```
    Vnm_print(0, "Vclist_setupGrid:  Grid lower corner = (%g, %g, %g)\n",
      (thee->lower_corner)[0], (thee->lower_corner)[1],
      (thee->lower_corner)[2]);

    return VRC_SUCCESS;

    #undef VCLIST_INFLATE
}

/* Check and store parameters passed to constructor */
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] ) {

    int i = 0;

    if (alist == VNULL) {
        Vnm_print(2, "Vclist_ctor2:  Got NULL Valist!\n");
        return VRC_FAILURE;
    } else thee->alist = alist;

    thee->n = 1;
    for (i=0; i<VAPBS_DIM; i++) {
        if (npts[i] < 3) {
            Vnm_print(2,
                    "Vclist_ctor2:  n[%d] (%d) must be greater than 2!\n",
                    i, npts[i]);
            return VRC_FAILURE;
        }
        thee->npts[i] = npts[i];
        thee->n *= npts[i];
    }
    Vnm_print(0, "Vclist_ctor2:  Using %d x %d x %d hash table\n",
            npts[0], npts[1], npts[2]);

    thee->mode = mode;
    switch (thee->mode) {
        case CLIST_AUTO_DOMAIN:
            Vnm_print(0, "Vclist_ctor2:  automatic domain setup.\n");
            break;
        case CLIST_MANUAL_DOMAIN:
            Vnm_print(0, "Vclist_ctor2:  manual domain setup.\n");
            Vnm_print(0, "Vclist_ctor2:  lower corner = [ \n");
            for (i=0; i<VAPBS_DIM; i++) {
                thee->lower_corner[i] = lower_corner[i];
                Vnm_print(0, "%g ", lower_corner[i]);
            }
            Vnm_print(0, "]\n");
            Vnm_print(0, "Vclist_ctor2:  upper corner = [ \n");
            for (i=0; i<VAPBS_DIM; i++) {
                thee->upper_corner[i] = upper_corner[i];
                Vnm_print(0, "%g ", upper_corner[i]);
            }
            Vnm_print(0, "]\n");
            break;
        default:
            Vnm_print(2, "Vclist_ctor2:  invalid setup mode (%d)!\n", mode);
            return VRC_FAILURE;
    }

    thee->max_radius = max_radius;
    Vnm_print(0, "Vclist_ctor2:  Using %g max radius\n", max_radius);

    return VRC_SUCCESS;
}

/* Calculate the gridpoints an atom spans */
VPRIVATE void Vclist_gridSpan(Vclist *thee,
        Vatom *atom, /* Atom */
        int imin[VAPBS_DIM], /* Set to min grid indices */
        int imax[VAPBS_DIM]  /* Set to max grid indices */
        ) {

    int i;
    double *coord, dc, idc, rtot;

    /* Get the position in the grid's frame of reference */
    coord = Vatom_getPosition(atom);

    /* Get the range the atom radius + probe radius spans */
```

```
    rtot = Vatom_getRadius(atom) + thee->max_radius;

    /* Calculate the range of grid points the inflated atom spans in the x
     * direction. */
    for (i=0; i<VAPBS_DIM; i++) {
        dc = coord[i] - (thee->lower_corner)[i];
        idc = (dc + rtot)/(thee->spacs[i]);
        imax[i] = (int)(ceil(idc));
        imax[i] = VMIN2(imax[i], thee->npts[i]-1);
        idc = (dc - rtot)/(thee->spacs[i]);
        imin[i] = (int)(floor(idc));
        imin[i] = VMAX2(imin[i], 0);
    }

}

/* Get the array index for a particular cell based on its i,j,k
 * coordinates */
VPRIVATE int Vclist_arrayIndex(Vclist *thee, int i, int j, int k) {

    return (thee->npts[2])*(thee->npts[1])*i + (thee->npts[2])*j + k;

}


/* Assign atoms to cells */
VPRIVATE Vrc_Codes Vclist_assignAtoms(Vclist *thee) {

    int iatom, i, j, k, ui, inext;
    int imax[VAPBS_DIM], imin[VAPBS_DIM];
    int totatoms;
    Vatom *atom;
    VclistCell *cell;


    /* Find out how many atoms are associated with each grid point */
    totatoms = 0;
    for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {

        /* Get grid span for atom */
        atom = Valist_getAtom(thee->alist, iatom);
        Vclist_gridSpan(thee, atom, imin, imax);

        /* Now find and assign the grid points */
        VASSERT(VAPBS_DIM == 3);
        for ( i = imin[0]; i <= imax[0]; i++) {
            for ( j = imin[1]; j <= imax[1]; j++) {
                for ( k = imin[2]; k <= imax[2]; k++) {
                    /* Get index to array */
                    ui = Vclist_arrayIndex(thee, i, j, k);
                    /* Increment number of atoms for this grid point */
                    cell = &(thee->cells[ui]);
                    (cell->natoms)++;
                    totatoms++;
                }
            }
        }
    }
    Vnm_print(0, "Vclist_assignAtoms:  Have %d atom entries\n", totatoms);

    /* Allocate the space to store the pointers to the atoms */
    for (ui=0; ui<thee->n; ui++) {
        cell = &(thee->cells[ui]);
        if ( VclistCell_ctor2(cell, cell->natoms) == VRC_FAILURE ) {
            Vnm_print(2, "Vclist_assignAtoms:  cell error!\n");
            return VRC_FAILURE;
        }
        /* Clear the counter for later use */
        cell->natoms = 0;
    }

    /* Assign the atoms to grid points */
    for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {

        /* Get grid span for atom */
        atom = Valist_getAtom(thee->alist, iatom);
        Vclist_gridSpan(thee, atom, imin, imax);
```

```c

        /* Now find and assign the grid points */
        for (i = imin[0]; i <= imax[0]; i++) {
            for (j = imin[1]; j <= imax[1]; j++) {
                for (k = imin[2]; k <= imax[2]; k++) {
                    /* Get index to array */
                    ui = Vclist_arrayIndex(thee, i, j, k);
                    cell = &(thee->cells[ui]);
                    /* Index of next available array location */
                    inext = cell->natoms;
                    cell->atoms[inext] = atom;
                    /* Increment number of atoms */
                    (cell->natoms)++;
                }
            }
        }
    }

    return VRC_SUCCESS;
}

/* Main (FORTRAN stub) constructor */
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]) {

    int i;
    VclistCell *cell;

    /* Check and store parameters */
    if ( Vclist_storeParms(thee, alist, max_radius, npts, mode, lower_corner,
                upper_corner) == VRC_FAILURE ) {
        Vnm_print(2, "Vclist_ctor2:  parameter check failed!\n");
        return VRC_FAILURE;
    }

    /* Set up memory */
    thee->vmem = Vmem_ctor("APBS::VCLIST");
    if (thee->vmem == VNULL) {
        Vnm_print(2, "Vclist_ctor2:  memory object setup failed!\n");
        return VRC_FAILURE;
    }

    /* Set up cells */
    thee->cells = (VclistCell*)Vmem_malloc( thee->vmem, thee->n, sizeof(VclistCell) );
    if (thee->cells == VNULL) {
        Vnm_print(2,
                "Vclist_ctor2:  Failed allocating %d VclistCell objects!\n",
                thee->n);
        return VRC_FAILURE;
    }
    for (i=0; i<thee->n; i++) {
        cell = &(thee->cells[i]);
        cell->natoms = 0;
    }

    /* Set up the grid */
    if ( Vclist_setupGrid(thee) == VRC_FAILURE ) {
        Vnm_print(2, "Vclist_ctor2:  grid setup failed!\n");
        return VRC_FAILURE;
    }

    /* Assign atoms to grid cells */
    if (Vclist_assignAtoms(thee) == VRC_FAILURE) {
        Vnm_print(2, "Vclist_ctor2:  atom assignment failed!\n");
        return VRC_FAILURE;
    }




    return VRC_SUCCESS;
}

/* Destructor */
VPUBLIC void Vclist_dtor(Vclist **thee) {

    if ((*thee) != VNULL) {
```

```
        Vclist_dtor2(*thee);
        Vmem_free(VNULL, 1, sizeof(Vclist), (void **)thee);
        (*thee) = VNULL;
    }

}

/* Main (stub) destructor */
VPUBLIC void Vclist_dtor2(Vclist *thee) {

    VclistCell *cell;
    int i;

    for (i=0; i<thee->n; i++) {
        cell = &(thee->cells[i]);
        VclistCell_dtor2(cell);
    }
    Vmem_free(thee->vmem, thee->n, sizeof(VclistCell),
            (void **)&(thee->cells));
    Vmem_dtor(&(thee->vmem));

}

VPUBLIC VclistCell* Vclist_getCell(Vclist *thee,
                                   double pos[VAPBS_DIM]
                                  ) {

    int i,
        ic[VAPBS_DIM],
        ui;
    double c[VAPBS_DIM];

    /* Assert this before we do anything else, since its failure should fail the function */
    VASSERT(VAPBS_DIM == 3);

    /* Convert to grid based coordinates */
    for (i=0; i<VAPBS_DIM; i++) {
        c[i] = pos[i] - (thee->lower_corner)[i];
        ic[i] = (int)(c[i]/thee->spacs[i]);

        if (ic[i] < 0 || ic[i] >= thee->npts[i]) {
            return VNULL;
        }
    }

    /* Get the array index */
    ui = Vclist_arrayIndex(thee, ic[0], ic[1], ic[2]);

    return &(thee->cells[ui]);

}

VPUBLIC VclistCell* VclistCell_ctor(int natoms) {

    VclistCell *thee = VNULL;

    /* Set up the structure */
    thee = (VclistCell*)Vmem_malloc(VNULL, 1, sizeof(VclistCell));
    VASSERT( thee != VNULL);
    VASSERT( VclistCell_ctor2(thee, natoms) == VRC_SUCCESS );

    return thee;
}

VPUBLIC Vrc_Codes VclistCell_ctor2(VclistCell *thee, int natoms) {

    if (thee == VNULL) {
        Vnm_print(2, "VclistCell_ctor2:  NULL thee!\n");
        return VRC_FAILURE;
    }

    thee->natoms = natoms;
    if (thee->natoms > 0) {
        thee->atoms = (Vatom**)Vmem_malloc(VNULL, natoms, sizeof(Vatom *));
        if (thee->atoms == VNULL) {
            Vnm_print(2,
```

```
            "VclistCell_ctor2:  unable to allocate space for %d atom pointers!\n"
,
                natoms);
              return VRC_FAILURE;
          }
      }

      return VRC_SUCCESS;

}

VPUBLIC void VclistCell_dtor(VclistCell **thee) {

      if ((*thee) != VNULL) {
          VclistCell_dtor2(*thee);
          Vmem_free(VNULL, 1, sizeof(VclistCell), (void **)thee);
          (*thee) = VNULL;
      }

}

/* Main (stub) destructor */
VPUBLIC void VclistCell_dtor2(VclistCell *thee) {

      if (thee->natoms > 0) {
          Vmem_free(VNULL, thee->natoms, sizeof(Vatom *),
                  (void **)&(thee->atoms));
      }

}

```

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 **treecleanup** (Vgreen ∗thee)
- VPRIVATE int **treecalc** (Vgreen ∗thee, double ∗xtar, double ∗ytar, double ∗ztar, double ∗qtar, int numtars, double ∗tpengtar, double ∗x, double ∗y, double ∗z, double ∗q, int numpars, double ∗fx, double ∗fy, double ∗fz, int iflag, int farrdim, int arrdim)
- VPUBLIC Valist ∗ Vgreen_getValist (Vgreen ∗thee)

  Get the atom list associated with this Green's function object.
- VPUBLIC unsigned long int Vgreen_memChk (Vgreen ∗thee)

  Return the memory used by this structure (and its contents) in bytes.
- VPUBLIC Vgreen ∗ Vgreen_ctor (Valist ∗alist)

  Construct the Green's function oracle.
- VPUBLIC int Vgreen_ctor2 (Vgreen ∗thee, Valist ∗alist)

  FORTRAN stub to construct the Green's function oracle.
- VPUBLIC void Vgreen_dtor (Vgreen ∗∗thee)

  Destruct the Green's function oracle.
- VPUBLIC void Vgreen_dtor2 (Vgreen ∗thee)

  FORTRAN stub to destruct the Green's function oracle.
- VPUBLIC int Vgreen_helmholtz (Vgreen ∗thee, int npos, double ∗x, double ∗y, double ∗z, double ∗val, double kappa)

Get the Green's function for Helmholtz's equation integrated over the atomic point charges.

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

  Get the gradient of Green's function for Helmholtz's equation integrated over the atomic point charges.

- VPUBLIC int Vgreen_coulomb_direct (Vgreen *thee, int npos, double *x, double *y, double *z, double *val)

  Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.

- VPUBLIC int Vgreen_coulomb (Vgreen *thee, int npos, double *x, double *y, double *z, double *val)

  Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation or H. E. Johnston, R. Krasny FMM library (if available)

- VPUBLIC int Vgreen_coulombD_direct (Vgreen *thee, int npos, double *x, double *y, double *z, double *pot, double *gradx, double *grady, double *gradz)

  Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.

- VPUBLIC int Vgreen_coulombD (Vgreen *thee, int npos, double *x, double *y, double *z, double *pot, double *gradx, double *grady, double *gradz)

  Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using either direct summation or H. E. Johnston/R. Krasny FMM library (if available)

9.73.1 Detailed Description

Class Vgreen methods.

Author

Nathan Baker

Version

Id:

vgreen.c 1774 2012-07-19 18:46:14Z sobolevnrm

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_COMPP VF77_MANGLE(mytree_compp, MYTREE_COMP)
00066 #  define F77TREE_COMPFP VF77_MANGLE(mytree_compfp, MYTREE_COMPFP)
00067 #  define F77CREATE_TREE VF77_MANGLE(mycreate_tree, MYCREATE_TREE)
00068 #  define F77INITLEVELS VF77_MANGLE(myinitlevels, MYINITLEVELS)
00069 #  define F77SETUP VF77_MANGLE(mysetup, MYSETUP)
00070 #endif  /* ifdef HAVE_TREE */
00071
00072 /* Some constants associated with the tree code */
00073 #ifdef HAVE_TREE
00074
00077 #   define FMM_DIST_TOL VSMALL
00078
00083 #   define FMM_IFLAG 2
00084
00087 #   define FMM_ORDER 4
00088
00091 #   define FMM_THETA 0.5
00092
00095 #   define FMM_MAXPARNODE 150
00096
00099 #   define FMM_SHRINK 1
00100
00103 #   define FMM_MINLEVEL 50000
00104
00107 #   define FMM_MAXLEVEL 0
00108 #endif  /* ifdef HAVE_TREE */
00109
00110
00111 /*
00112  * @brief  Setup treecode internal structures
00113  * @ingroup  Vgreen
```

```
 * @author  Nathan Baker
 * @param  thee  Vgreen object
 * @return  1 if successful, 0 otherwise
 */
VPRIVATE int treesetup(Vgreen *thee);

/*
 * @brief  Clean up treecode internal structures
 * @ingroup  Vgreen
 * @author  Nathan Baker
 * @param  thee  Vgreen object
 * @return  1 if successful, 0 otherwise
 */
VPRIVATE int treecleanup(Vgreen *thee);

/*
 * @brief  Calculate forces or potential
 * @ingroup  Vgreen
 * @author  Nathan Baker
 * @param  thee  Vgreen object
 * @return  1 if successful, 0 otherwise
 */
VPRIVATE int treecalc(Vgreen *thee, double *xtar, double *ytar, double *ztar,
        double *qtar, int numtars, double *tpengtar, double *x, double *y,
        double *z, double *q, int numpars, double *fx, double *fy, double *fz,
        int iflag, int farrdim, int arrdim);

#if !defined(VINLINE_VGREEN)

VPUBLIC Valist* Vgreen_getValist(Vgreen *thee) {

   VASSERT(thee != VNULL);
   return thee->alist;

}

VPUBLIC unsigned long int Vgreen_memChk(Vgreen *thee) {
    if (thee == VNULL) return 0;
    return Vmem_bytes(thee->vmem);
}

#endif /* if !defined(VINLINE_VGREEN) */

VPUBLIC Vgreen* Vgreen_ctor(Valist *alist) {

    /* Set up the structure */
    Vgreen *thee = VNULL;
    thee = (Vgreen *)Vmem_malloc(VNULL, 1, sizeof(Vgreen) );
    VASSERT( thee != VNULL);
    VASSERT( Vgreen_ctor2(thee, alist));

    return thee;
}

VPUBLIC int Vgreen_ctor2(Vgreen *thee, Valist *alist) {

    VASSERT( thee != VNULL );

    /* Memory management object */
    thee->vmem = Vmem_ctor("APBS:VGREEN");

    /* Set up the atom list and grid manager */
    if (alist == VNULL) {
        Vnm_print(2,"Vgreen_ctor2: got null pointer to Valist object!\n");
    }

    thee->alist = alist;

    /* Setup FMM tree (if applicable) */
#ifdef HAVE_TREE
    if (!treesetup(thee)) {
        Vnm_print(2, "Vgreen_ctor2:  Error setting up FMM tree!\n");
        return 0;
    }
#endif /* ifdef HAVE_TREE */

    return 1;
}

```

```
VPUBLIC void Vgreen_dtor(Vgreen **thee) {
    if ((*thee) != VNULL) {
        Vgreen_dtor2(*thee);
        Vmem_free(VNULL, 1, sizeof(Vgreen), (void **)thee);
        (*thee) = VNULL;
    }
}

VPUBLIC void Vgreen_dtor2(Vgreen *thee) {

#ifdef HAVE_TREE
    treecleanup(thee);
#endif
    Vmem_dtor(&(thee->vmem));

}

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

    Vnm_print(2, "Error -- Vgreen_helmholtz not implemented yet!\n");
    return 0;
}

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

    Vnm_print(2, "Error -- Vgreen_helmholtzD not implemented yet!\n");
    return 0;

}

VPUBLIC int Vgreen_coulomb_direct(Vgreen *thee, int npos, double *x,
        double *y, double *z, double *val) {

    Vatom *atom;
    double *apos, charge, dist, dx, dy, dz, scale;
    double *q, qtemp, fx, fy, fz;
    int iatom, ipos;

    if (thee == VNULL) {
        Vnm_print(2, "Vgreen_coulomb:  Got NULL thee!\n");
        return 0;
    }

    for (ipos=0; ipos<npos; ipos++) val[ipos] = 0.0;

    for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
        atom = Valist_getAtom(thee->alist, iatom);
        apos = Vatom_getPosition(atom);
        charge = Vatom_getCharge(atom);
        for (ipos=0; ipos<npos; ipos++) {
            dx = apos[0] - x[ipos];
            dy = apos[1] - y[ipos];
            dz = apos[2] - z[ipos];
            dist = VSQRT(VSQR(dx) + VSQR(dy) + VSQR(dz));
            if (dist > VSMALL) val[ipos] += (charge/dist);
        }
    }

    scale = Vunit_ec/(4*Vunit_pi*Vunit_eps0*1.0e-10);
    for (ipos=0; ipos<npos; ipos++) val[ipos] = val[ipos]*scale;

    return 1;
}

VPUBLIC int Vgreen_coulomb(Vgreen *thee, int npos, double *x, double *y,
  double *z, double *val) {

    Vatom *atom;
    double *apos, charge, dist, dx, dy, dz, scale;
    double *q, qtemp, fx, fy, fz;
    int iatom, ipos;

    if (thee == VNULL) {
        Vnm_print(2, "Vgreen_coulomb:  Got NULL thee!\n");
```

```
        return 0;
    }

    for (ipos=0; ipos<npos; ipos++) val[ipos] = 0.0;

#ifdef HAVE_TREE

    /* Allocate charge array (if necessary) */
    if (Valist_getNumberAtoms(thee->alist) > 1) {
        if (npos > 1) {
            q = VNULL;
            q = Vmem_malloc(thee->vmem, npos, sizeof(double));
            if (q == VNULL) {
                Vnm_print(2, "Vgreen_coulomb:  Error allocating charge array!\n");
                return 0;
            }
        } else {
            q = &(qtemp);
        }
        for (ipos=0; ipos<npos; ipos++) q[ipos] = 1.0;

        /* Calculate */
        treecalc(thee, x, y, z, q, npos, val, thee->xp, thee->yp, thee->zp,
          thee->qp, thee->np, &fx, &fy, &fz, 1, 1, thee->np);
    } else return Vgreen_coulomb_direct(thee, npos, x, y, z, val);

    /* De-allocate charge array (if necessary) */
    if (npos > 1) Vmem_free(thee->vmem, npos, sizeof(double), (void **)&q);

    scale = Vunit_ec/(4*Vunit_pi*Vunit_eps0*1.0e-10);
    for (ipos=0; ipos<npos; ipos++) val[ipos] = val[ipos]*scale;

    return 1;

#else /* ifdef HAVE_TREE */

    return Vgreen_coulomb_direct(thee, npos, x, y, z, val);

#endif

}

VPUBLIC int Vgreen_coulombD_direct(Vgreen *thee, int npos,
        double *x, double *y, double *z, double *pot, double *gradx,
        double *grady, double *gradz) {

    Vatom *atom;
    double *apos, charge, dist, dist2, idist3, dy, dz, dx, scale;
    double *q, qtemp;
    int iatom, ipos;

    if (thee == VNULL) {
        Vnm_print(2, "Vgreen_coulombD:  Got VNULL thee!\n");
        return 0;
    }

    for (ipos=0; ipos<npos; ipos++) {
        pot[ipos] = 0.0;
        gradx[ipos] = 0.0;
        grady[ipos] = 0.0;
        gradz[ipos] = 0.0;
    }

    for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
        atom = Valist_getAtom(thee->alist, iatom);
        apos = Vatom_getPosition(atom);
        charge = Vatom_getCharge(atom);
        for (ipos=0; ipos<npos; ipos++) {
            dx = apos[0] - x[ipos];
            dy = apos[1] - y[ipos];
            dz = apos[2] - z[ipos];
            dist2 = VSQR(dx) + VSQR(dy) + VSQR(dz);
            dist = VSQRT(dist2);
            if (dist > VSMALL) {
                idist3 = 1.0/(dist*dist2);
```

```
                gradx[ipos] -= (charge*dx*idist3);
                grady[ipos] -= (charge*dy*idist3);
                gradz[ipos] -= (charge*dz*idist3);
                pot[ipos] += (charge/dist);
            }
        }
    }

    scale = Vunit_ec/(4*VPI*Vunit_eps0*(1.0e-10));
    for (ipos=0; ipos<npos; ipos++) {
        gradx[ipos] = gradx[ipos]*scale;
        grady[ipos] = grady[ipos]*scale;
        gradz[ipos] = gradz[ipos]*scale;
        pot[ipos] = pot[ipos]*scale;
    }

    return 1;
}

VPUBLIC int Vgreen_coulombD(Vgreen *thee, int npos, double *x, double *y,
        double *z, double *pot, double *gradx, double *grady, double *gradz) {

    Vatom *atom;
    double *apos, charge, dist, dist2, idist3, dy, dz, dx, scale;
    double *q, qtemp;
    int iatom, ipos;

    if (thee == VNULL) {
        Vnm_print(2, "Vgreen_coulombD:  Got VNULL thee!\n");
        return 0;
    }

    for (ipos=0; ipos<npos; ipos++) {
        pot[ipos] = 0.0;
        gradx[ipos] = 0.0;
        grady[ipos] = 0.0;
        gradz[ipos] = 0.0;
    }

#ifdef HAVE_TREE

    if (Valist_getNumberAtoms(thee->alist) > 1) {
        if (npos > 1) {
            q = VNULL;
            q = Vmem_malloc(thee->vmem, npos, sizeof(double));
            if (q == VNULL) {
                Vnm_print(2, "Vgreen_coulomb:  Error allocating charge array!\n");
                return 0;
            }
        } else {
            q = &(qtemp);
        }
        for (ipos=0; ipos<npos; ipos++) q[ipos] = 1.0;

        /* Calculate */
        treecalc(thee, x, y, z, q, npos, pot, thee->xp, thee->yp, thee->zp,
                thee->qp, thee->np, gradx, grady, gradz, 2, npos, thee->np);

        /* De-allocate charge array (if necessary) */
        if (npos > 1) Vmem_free(thee->vmem, npos, sizeof(double), (void **)&q);
    } else return Vgreen_coulombD_direct(thee, npos, x, y, z, pot,
            gradx, grady, gradz);

    scale = Vunit_ec/(4*VPI*Vunit_eps0*(1.0e-10));
    for (ipos=0; ipos<npos; ipos++) {
        gradx[ipos] = gradx[ipos]*scale;
        grady[ipos] = grady[ipos]*scale;
        gradz[ipos] = gradz[ipos]*scale;
        pot[ipos] = pot[ipos]*scale;
    }

    return 1;

#else /* ifdef HAVE_TREE */

```

```
    return Vgreen_coulombD_direct(thee, npos, x, y, z, pot,
            gradx, grady, gradz);

#endif

}

VPRIVATE int treesetup(Vgreen *thee) {

#ifdef HAVE_TREE

    double dist_tol = FMM_DIST_TOL;
    int iflag = FMM_IFLAG;
    double order = FMM_ORDER;
    int theta = FMM_THETA;
    int shrink = FMM_SHRINK;
    int maxparnode = FMM_MAXPARNODE;
    int minlevel = FMM_MINLEVEL;
    int maxlevel = FMM_MAXLEVEL;
    int level = 0;
    int one = 1;
    Vatom *atom;
    double xyzminmax[6], *pos;
    int i;

    /* Set up particle arrays with atomic coordinates and charges */
    Vnm_print(0, "treesetup:  Initializing FMM particle arrays...\n");
    thee->np = Valist_getNumberAtoms(thee->alist);
    thee->xp = VNULL;
    thee->xp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(double));
    if (thee->xp == VNULL) {
        Vnm_print(2, "Vgreen_ctor2:  Failed to allocate %d*sizeof(double)!\n",
          thee->np);
        return 0;
    }
    thee->yp = VNULL;
    thee->yp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(double));
    if (thee->yp == VNULL) {
        Vnm_print(2, "Vgreen_ctor2:  Failed to allocate %d*sizeof(double)!\n",
          thee->np);
        return 0;
    }
    thee->zp = VNULL;
    thee->zp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(double));
    if (thee->zp == VNULL) {
        Vnm_print(2, "Vgreen_ctor2:  Failed to allocate %d*sizeof(double)!\n",
          thee->np);
        return 0;
    }
    thee->qp = VNULL;
    thee->qp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(double));
    if (thee->qp == VNULL) {
        Vnm_print(2, "Vgreen_ctor2:  Failed to allocate %d*sizeof(double)!\n",
          thee->np);
        return 0;
    }
    for (i=0; i<thee->np; i++) {
        atom = Valist_getAtom(thee->alist, i);
        pos = Vatom_getPosition(atom);
        thee->xp[i] = pos[0];
        thee->yp[i] = pos[1];
        thee->zp[i] = pos[2];
        thee->qp[i] = Vatom_getCharge(atom);
    }

    Vnm_print(0, "treesetup:  Setting things up...\n");
    F77SETUP(thee->xp, thee->yp, thee->zp, &(thee->np), &order, &theta, &iflag,
            &dist_tol, xyzminmax, &(thee->np));


    Vnm_print(0, "treesetup:  Initializing levels...\n");
    F77INITLEVELS(&minlevel, &maxlevel);

    Vnm_print(0, "treesetup:  Creating tree...\n");
    F77CREATE_TREE(&one, &(thee->np), thee->xp, thee->yp, thee->zp,
```

```
    thee->qp,
00494        &shrink, &maxparnode, xyzminmax, &level, &(thee->np));
00495
00496     return 1;
00497
00498 #else /* ifdef HAVE_TREE */
00499
00500     Vnm_print(2, "treesetup:  Error!  APBS not linked with treecode!\n");
00501     return 0;
00502
00503 #endif /* ifdef HAVE_TREE */
00504 }
00505
00506 VPRIVATE int treecleanup(Vgreen *thee) {
00507
00508 #ifdef HAVE_TREE
00509
00510     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->xp
    ));
00511     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->yp
    ));
00512     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->zp
    ));
00513     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->qp
    ));
00514     F77CLEANUP();
00515
00516     return 1;
00517
00518 #else /* ifdef HAVE_TREE */
00519
00520     Vnm_print(2, "treecleanup:  Error!  APBS not linked with treecode!\n");
00521     return 0;
00522
00523 #endif /* ifdef HAVE_TREE */
00524 }
00525
00526 VPRIVATE int treecalc(Vgreen *thee, double *xtar, double *ytar, double *
    ztar,
00527         double *qtar, int numtars, double *tpengtar, double *x, double *y,
00528         double *z, double *q, int numpars, double *fx, double *fy, double *fz,
00529         int iflag, int farrdim, int arrdim) {
00530
00531 #ifdef HAVE_TREE
00532     int i, level, err, maxlevel, minlevel, one;
00533     double xyzminmax[6];
00534
00535
00536     if (iflag != 1) {
00537         F77TREE_COMPFP(xtar, ytar, ztar, qtar, &numtars, tpengtar, x, y, z, q,
00538                 fx, fy, fz, &numpars, &farrdim, &arrdim);
00539     } else {
00540         F77TREE_COMPP(xtar, ytar, ztar, qtar, &numtars, tpengtar, &farrdim, x,
00541                 y, z, q, &numpars, &arrdim);
00542     }
00543
00544
00545     return 1;
00546
00547 #else /* ifdef HAVE_TREE */
00548
00549     Vnm_print(2, "treecalc:  Error!  APBS not linked with treecode!\n");
00550     return 0;
00551
00552 #endif /* ifdef HAVE_TREE */
00553 }
00554
```

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 1774 2012-07-19 18:46:14Z sobolevnrm

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"
```

```
#if defined(HAVE_MC_H)
#include "mc/mc.h"
#endif

VEMBED(rcsid="$Id: vparam.c 1774 2012-07-19 18:46:14Z sobolevnrm $")


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

VPRIVATE char *MCcommChars  = "#%";

VPRIVATE char *MCxmlwhiteChars = " =,;\t\n\r<>";

VPRIVATE int readFlatFileLine(Vio *sock, Vparam_AtomData *atom);

VPRIVATE int readXMLFileAtom(Vio *sock, Vparam_AtomData *atom);


#if !defined(VINLINE_VPARAM)

VPUBLIC unsigned long int Vparam_memChk(Vparam *thee) {
    if (thee == VNULL) return 0;
    return Vmem_bytes(thee->vmem);
}

#endif /* if !defined(VINLINE_VPARAM) */

VPUBLIC Vparam_AtomData* Vparam_AtomData_ctor() {

    Vparam_AtomData *thee = VNULL;

    /* Set up the structure */
    thee = (Vparam_AtomData*)Vmem_malloc(VNULL, 1, sizeof(Vparam_AtomData) );
    VASSERT(thee != VNULL);
    VASSERT(Vparam_AtomData_ctor2(thee));

    return thee;
}

VPUBLIC int Vparam_AtomData_ctor2(Vparam_AtomData *thee) { return 1; }

VPUBLIC void Vparam_AtomData_dtor(Vparam_AtomData **thee) {

    if ((*thee) != VNULL) {
        Vparam_AtomData_dtor2(*thee);
        Vmem_free(VNULL, 1, sizeof(Vparam_AtomData), (void **)thee);
        (*thee) = VNULL;
    }

}

VPUBLIC void Vparam_AtomData_dtor2(Vparam_AtomData *thee) { ; }

VPUBLIC Vparam_ResData* Vparam_ResData_ctor(Vmem *mem) {

    Vparam_ResData *thee = VNULL;

    /* Set up the structure */
    thee = (Vparam_ResData*)Vmem_malloc(mem, 1, sizeof(Vparam_ResData) );
    VASSERT(thee != VNULL);
    VASSERT(Vparam_ResData_ctor2(thee, mem));

    return thee;
}

VPUBLIC int Vparam_ResData_ctor2(Vparam_ResData *thee, Vmem *mem) {

    if (thee == VNULL) {
        Vnm_print(2, "Vparam_ResData_ctor2:  Got VNULL thee!\n");
```

```
        return 0;
    }
    thee->vmem = mem;
    thee->nAtomData = 0;
    thee->atomData = VNULL;

    return 1;
}

VPUBLIC void Vparam_ResData_dtor(Vparam_ResData **thee) {

    if ((*thee) != VNULL) {
        Vparam_ResData_dtor2(*thee);
        Vmem_free((*thee)->vmem, 1, sizeof(Vparam_ResData), (void **)thee);
        (*thee) = VNULL;
    }

}

VPUBLIC void Vparam_ResData_dtor2(Vparam_ResData *thee) {

    if (thee == VNULL) return;
    if (thee->nAtomData > 0) {
        Vmem_free(thee->vmem, thee->nAtomData, sizeof(Vparam_AtomData),
          (void **)&(thee->atomData));
    }
    thee->nAtomData = 0;
    thee->atomData = VNULL;
}

VPUBLIC Vparam* Vparam_ctor() {

    Vparam *thee = VNULL;

    /* Set up the structure */
    thee = (Vparam*)Vmem_malloc(VNULL, 1, sizeof(Vparam) );
    VASSERT(thee != VNULL);
    VASSERT(Vparam_ctor2(thee));

    return thee;
}

VPUBLIC int Vparam_ctor2(Vparam *thee) {

    if (thee == VNULL) {
        Vnm_print(2, "Vparam_ctor2: got VNULL thee!\n");
        return 0;
    }

    thee->vmem = VNULL;
    thee->vmem = Vmem_ctor("APBS:VPARAM");
    if (thee->vmem == VNULL) {
        Vnm_print(2, "Vparam_ctor2:  failed to init Vmem!\n");
        return 0;
    }

    thee->nResData = 0;
    thee->resData = VNULL;

    return 1;
}

VPUBLIC void Vparam_dtor(Vparam **thee) {

    if ((*thee) != VNULL) {
        Vparam_dtor2(*thee);
        Vmem_free(VNULL, 1, sizeof(Vparam), (void **)thee);
        (*thee) = VNULL;
    }

}

VPUBLIC void Vparam_dtor2(Vparam *thee) {

    int i;

    if (thee == VNULL) return;
```

```
    /* Destroy the residue data */
    for (i=0; i<thee->nResData; i++) Vparam_ResData_dtor2(&(thee->resData[i]));
    if (thee->nResData > 0) Vmem_free(thee->vmem, thee->nResData,
      sizeof(Vparam_ResData), (void **)&(thee->resData));
    thee->nResData = 0;
    thee->resData = VNULL;

    if (thee->vmem != VNULL) Vmem_dtor(&(thee->vmem));
    thee->vmem = VNULL;

}

VPUBLIC Vparam_ResData* Vparam_getResData(Vparam *thee,
  char resName[VMAX_ARGLEN]) {

    int i;
    Vparam_ResData *res = VNULL;

    VASSERT(thee != VNULL);

    if ((thee->nResData == 0) || (thee->resData == VNULL)) {
        res = VNULL;
        return res;
    }

    /* Look for the matching residue */
    for (i=0; i<thee->nResData; i++) {
        res = &(thee->resData[i]);
        if (Vstring_strcasecmp(resName, res->name) == 0) return res;

    }

    /* Didn't find a matching residue */
    res = VNULL;
    Vnm_print(2, "Vparam_getResData:  unable to find res=%s\n", resName);
    return res;
}

VPUBLIC Vparam_AtomData* Vparam_getAtomData(Vparam *thee,
  char resName[VMAX_ARGLEN], char atomName[VMAX_ARGLEN]) {

    int i;
    Vparam_ResData *res = VNULL;
    Vparam_AtomData *atom = VNULL;

    VASSERT(thee != VNULL);

    if ((thee->nResData == 0) || (thee->resData == VNULL)) {
        atom = VNULL;
        return atom;
    }

    /* Look for the matching residue */
    res = Vparam_getResData(thee, resName);
    if (res == VNULL) {
        atom = VNULL;
  Vnm_print(2, "Vparam_getAtomData:  Unable to find residue %s!\n", resName);
        return atom;
    }
    for (i=0; i<res->nAtomData; i++) {
        atom = &(res->atomData[i]);
  if (atom == VNULL) {
   Vnm_print(2, "Vparam_getAtomData:  got NULL atom!\n");
   return VNULL;
  }
        if (Vstring_strcasecmp(atomName, atom->atomName) == 0) {
            return atom;
        }
    }

    /* Didn't find a matching atom/residue */
    atom = VNULL;
    Vnm_print(2, "Vparam_getAtomData:  unable to find atom '%s', res '%s'\n",
      atomName, resName);
```

```
    return atom;
}

VPUBLIC int Vparam_readXMLFile(Vparam *thee, const char *iodev,
 const char *iofmt, const char *thost, const char *fname) {

    int i, ires, natoms, nalloc, ralloc;
    Vparam_AtomData *atoms = VNULL;
    Vparam_AtomData *tatoms = VNULL;
    Vparam_AtomData *atom = VNULL;
    Vparam_ResData *res = VNULL;
    Vparam_ResData *residues = VNULL;
    Vparam_ResData *tresidues = VNULL;
    Vio *sock = VNULL;
    char currResName[VMAX_ARGLEN];
    char tok[VMAX_ARGLEN];
    char endtag[VMAX_ARGLEN];

    VASSERT(thee != VNULL);

    /* Setup communication */
    sock = Vio_ctor(iodev,iofmt,thost,fname,"r");
    if (sock == VNULL) {
        Vnm_print(2, "Vparam_readXMLFile: Problem opening virtual socket %s\n",
          fname);
        return 0;
    }
    if (Vio_accept(sock, 0) < 0) {
        Vnm_print(2, "Vparam_readXMLFile: Problem accepting virtual socket %s\n",
          fname);
        return 0;
    }
    Vio_setWhiteChars(sock, MCxmlwhiteChars);
    Vio_setCommChars(sock, MCcommChars);

    /* Clear existing parameters */
    if (thee->nResData > 0) {
        Vnm_print(2, "WARNING -- CLEARING PARAMETER DATABASE!\n");
        for (i=0; i<thee->nResData; i++) {
            Vparam_ResData_dtor2(&(thee->resData[i]));
        }
        Vmem_free(thee->vmem, thee->nResData,
          sizeof(Vparam_ResData), (void **)&(thee->resData));
    }

    strcpy(endtag,"/");

    /* Set up temporary residue list */

    ralloc = 50;
    residues = (Vparam_ResData*)Vmem_malloc(thee->vmem, ralloc, sizeof(Vparam_ResData));

    /* Read until we run out of entries, allocating space as needed */
    while (1) {

        VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);

        /* The first token should be the start tag */

        if (Vstring_strcasecmp(endtag, "/") == 0) strcat(endtag, tok);

        if (Vstring_strcasecmp(tok, "residue") == 0) {
          if (thee->nResData >= ralloc) {
                tresidues = (Vparam_ResData*)Vmem_malloc(thee->vmem, 2*ralloc, sizeof(Vparam_ResData));
                VASSERT(tresidues != VNULL);
                for (i=0; i<thee->nResData; i++) {
                    Vparam_ResData_copyTo(&(residues[i]), &(tresidues[i]));
                }
                Vmem_free(thee->vmem, ralloc, sizeof(Vparam_ResData),
                          (void **)&(residues));
                residues = tresidues;
                tresidues = VNULL;
```

```
                ralloc = 2*ralloc;
            }

          /* Initial space for this residue's atoms */
          nalloc = 20;
          natoms = 0;
          atoms = (Vparam_AtomData*)Vmem_malloc(thee->vmem, nalloc, sizeof(Vparam_AtomData));

        } else if (Vstring_strcasecmp(tok, "name") == 0) {
            VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);  /* value */
            strcpy(currResName, tok);
            VJMPERR1(Vio_scanf(sock, "%s", tok) == 1); /* </name> */
        } else if (Vstring_strcasecmp(tok, "atom") == 0) {
            if (natoms >= nalloc) {
                tatoms = (Vparam_AtomData*)Vmem_malloc(thee->vmem, 2*nalloc, sizeof(Vparam_AtomData));
                VASSERT(tatoms != VNULL);
                for (i=0; i<natoms; i++) {
                    Vparam_AtomData_copyTo(&(atoms[i]), &(tatoms[i]));
                }
                Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData),
                          (void **)&(atoms));
                atoms = tatoms;
                tatoms = VNULL;
                nalloc = 2*nalloc;
            }
            atom = &(atoms[natoms]);
            if (!readXMLFileAtom(sock, atom)) break;
            natoms++;

        } else if (Vstring_strcasecmp(tok, "/residue") == 0) {

          res = &(residues[thee->nResData]);
          Vparam_ResData_ctor2(res, thee->vmem);
          res->atomData = (Vparam_AtomData*)Vmem_malloc(thee->vmem, natoms,
                                    sizeof(Vparam_AtomData));
          res->nAtomData = natoms;
          strcpy(res->name, currResName);
          for (i=0; i<natoms; i++) {
              strcpy(atoms[i].resName, currResName);
              Vparam_AtomData_copyTo(&(atoms[i]), &(res->atomData[i]));
          }
          Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData), (void **)&(atoms));
          (thee->nResData)++;

        } else if (Vstring_strcasecmp(tok, endtag) == 0) break;
    }

    /* Initialize and copy the residues into the Vparam object */

    thee->resData = (Vparam_ResData*)Vmem_malloc(thee->vmem, thee->nResData,
                                 sizeof(Vparam_ResData));
    for (ires=0; ires<thee->nResData; ires++) {
        Vparam_ResData_copyTo(&(residues[ires]), &(thee->resData[ires]));
    }

    /* Destroy temporary atom space */
    Vmem_free(thee->vmem, ralloc, sizeof(Vparam_ResData), (void **)&(residues));

    /* Shut down communication */
    Vio_acceptFree(sock);
    Vio_dtor(&sock);

    return 1;

VERROR1:
    Vnm_print(2, "Vparam_readXMLFile: Got unexpected EOF reading parameter file!\n");
    return 0;

```

```
}

VPUBLIC int Vparam_readFlatFile(Vparam *thee, const char *iodev,
  const char *iofmt, const char *thost, const char *fname) {

    int i, iatom, jatom, ires, natoms, nalloc;
    Vparam_AtomData *atoms = VNULL;
    Vparam_AtomData *tatoms = VNULL;
    Vparam_AtomData *atom = VNULL;
    Vparam_ResData *res = VNULL;
    Vio *sock = VNULL;
    char currResName[VMAX_ARGLEN];

    VASSERT(thee != VNULL);

    /* Setup communication */
    sock = Vio_ctor(iodev,iofmt,thost,fname,"r");
    if (sock == VNULL) {
        Vnm_print(2, "Vparam_readFlatFile: Problem opening virtual socket %s\n",
          fname);
        return 0;
    }
    if (Vio_accept(sock, 0) < 0) {
        Vnm_print(2, "Vparam_readFlatFile: Problem accepting virtual socket %s\n",
          fname);
        return 0;
    }
    Vio_setWhiteChars(sock, MCwhiteChars);
    Vio_setCommChars(sock, MCcommChars);

    /* Clear existing parameters */
    if (thee->nResData > 0) {
        Vnm_print(2, "WARNING -- CLEARING PARAMETER DATABASE!\n");
        for (i=0; i<thee->nResData; i++) {
            Vparam_ResData_dtor2(&(thee->resData[i]));
        }
        Vmem_free(thee->vmem, thee->nResData,
          sizeof(Vparam_ResData), (void **)&(thee->resData));
    }

    /* Initial space for atoms */
    nalloc = 200;
    natoms = 0;
    atoms = (Vparam_AtomData*)Vmem_malloc(thee->vmem, nalloc, sizeof(Vparam_AtomData));

    /* Read until we run out of entries, allocating space as needed */
    while (1) {
        if (natoms >= nalloc) {
            tatoms = (Vparam_AtomData*)Vmem_malloc(thee->vmem, 2*nalloc, sizeof(Vparam_AtomData));
            VASSERT(tatoms != VNULL);
            for (i=0; i<natoms; i++) {
                Vparam_AtomData_copyTo(&(atoms[i]), &(tatoms[i]));
            }
            Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData),
              (void **)&(atoms));
            atoms = tatoms;
            tatoms = VNULL;
            nalloc = 2*nalloc;
        }
        atom = &(atoms[natoms]);
        if (!readFlatFileLine(sock, atom)) break;
        natoms++;
    }
    if (natoms == 0) return 0;

    /* Count the number of residues */
    thee->nResData = 1;
    strcpy(currResName, atoms[0].resName);
    for (i=1; i<natoms; i++) {
        if (Vstring_strcasecmp(atoms[i].resName, currResName) != 0) {
            strcpy(currResName, atoms[i].resName);
```

```
            (thee->nResData)++;
        }
    }

    /* Create the residues */
    thee->resData = (Vparam_ResData*)Vmem_malloc(thee->vmem, thee->nResData,
      sizeof(Vparam_ResData));
    VASSERT(thee->resData != VNULL);
    for (i=0; i<(thee->nResData); i++) {
        res = &(thee->resData[i]);
        Vparam_ResData_ctor2(res, thee->vmem);
    }

    /* Count the number of atoms per residue */
    ires = 0;
    res = &(thee->resData[ires]);
    res->nAtomData = 1;
    strcpy(res->name, atoms[0].resName);
    for (i=1; i<natoms; i++) {
        if (Vstring_strcasecmp(atoms[i].resName, res->name) != 0) {
            (ires)++;
            res = &(thee->resData[ires]);
            res->nAtomData = 1;
            strcpy(res->name, atoms[i].resName);
        } else (res->nAtomData)++;
    }

    /* Allocate per-residue space for atoms */
    for (ires=0; ires<thee->nResData; ires++) {
        res = &(thee->resData[ires]);
  res->atomData = (Vparam_AtomData*)Vmem_malloc(thee->vmem, res->nAtomData,
          sizeof(Vparam_AtomData));
    }

    /* Copy atoms into residues */
    iatom = 0;
    Vparam_AtomData_copyTo(&(atoms[0]), &(res->atomData[iatom]));
    for (ires=0; ires<thee->nResData; ires++) {
        res = &(thee->resData[ires]);
        for (jatom=0; jatom<res->nAtomData; jatom++) {
            Vparam_AtomData_copyTo(&(atoms[iatom]), &(res->atomData[jatom]));
            iatom++;
        }
    }


    /* Shut down communication */
    Vio_acceptFree(sock);
    Vio_dtor(&sock);

    /* Destroy temporary atom space */
    Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData), (void **)&(atoms));

    return 1;

}

VEXTERNC void Vparam_AtomData_copyTo(Vparam_AtomData *thee,
  Vparam_AtomData *dest) {

    VASSERT(thee != VNULL);
    VASSERT(dest != VNULL);

    strcpy(dest->atomName, thee->atomName);
    strcpy(dest->resName, thee->resName);
    dest->charge = thee->charge;
    dest->radius = thee->radius;
    dest->epsilon = thee->epsilon;

}

VEXTERNC void Vparam_ResData_copyTo(Vparam_ResData *thee,
  Vparam_ResData *dest) {
```

```c

    int i;

    VASSERT(thee != VNULL);
    VASSERT(dest != VNULL);

    strcpy(dest->name, thee->name);
    dest->vmem = thee->vmem;
    dest->nAtomData = thee->nAtomData;

    dest->atomData = (Vparam_AtomData*)Vmem_malloc(thee->vmem, dest->nAtomData,
                                   sizeof(Vparam_AtomData));

    for (i=0; i<dest->nAtomData; i++) {
      Vparam_AtomData_copyTo(&(thee->atomData[i]), &(dest->atomData[i]));
    }
    Vmem_free(thee->vmem, thee->nAtomData, sizeof(Vparam_AtomData),
              (void **)&(thee->atomData));
}

VEXTERNC void Vparam_AtomData_copyFrom(Vparam_AtomData *thee,
  Vparam_AtomData *src) {  Vparam_AtomData_copyTo(src, thee); }

VPRIVATE int readXMLFileAtom(Vio *sock, Vparam_AtomData *atom) {

    double dtmp;
    char tok[VMAX_BUFSIZE];
    int chgflag, radflag, nameflag;

    VASSERT(atom != VNULL);

    if (Vio_scanf(sock, "%s", tok) != 1) return 0;

    chgflag = 0;
    radflag = 0;
    nameflag = 0;

    while (1)
      {
          if (Vstring_strcasecmp(tok, "name") == 0) {
              VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
              if (strlen(tok) > VMAX_ARGLEN) {
                  Vnm_print(2, "Vparam_readXMLFileAtom:  string (%s) too long \
(%d)!\n", tok, strlen(tok));
                  return 0;
              }
              nameflag = 1;
              strcpy(atom->atomName, tok);
          } else if (Vstring_strcasecmp(tok, "charge") == 0) {
              VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
              if (sscanf(tok, "%lf", &dtmp) != 1) {
                  Vnm_print(2, "Vparam_readXMLFileAtom:  Unexpected token (%s) while \
parsing charge!\n", tok);
                  return 0;
              }
              chgflag = 1;
              atom->charge = dtmp;
          }  else if (Vstring_strcasecmp(tok, "radius") == 0) {
              VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
              if (sscanf(tok, "%lf", &dtmp) != 1) {
                  Vnm_print(2, "Vparam_readXMLFileAtom:  Unexpected token (%s) while \
parsing radius!\n", tok);
                  return 0;
              }
              radflag = 1;
              atom->radius = dtmp;
          }  else if (Vstring_strcasecmp(tok, "epsilon") == 0) {
              VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
              if (sscanf(tok, "%lf", &dtmp) != 1) {
                  Vnm_print(2, "Vparam_readXMLFileAtom:  Unexpected token (%s)
```

```
while \
parsing epsilon!\n", tok);
                    return 0;
                }
                atom->epsilon = dtmp;
            } else if ((Vstring_strcasecmp(tok, "/atom") == 0)
||
                        (Vstring_strcasecmp(tok, "atom") == 0)){
                  if (chgflag && radflag && nameflag) return 1;
                  else if (!chgflag) {
                    Vnm_print(2, "Vparam_readXMLFileAtom: Reached end of atom
without \
setting the charge!\n");
                    return 0;
                  } else if (!radflag) {
                    Vnm_print(2, "Vparam_readXMLFileAtom: Reached end of atom
without \
setting the radius!\n");
                    return 0;
                  } else if (!nameflag) {
                    Vnm_print(2, "Vparam_readXMLFileAtom: Reached end of atom
without \
setting the name!\n");
                    return 0;
                  }
            }
            VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
        }

    /* If we get here something wrong has happened */

    VJMPERR1(1);

VERROR1:
    Vnm_print(2, "Vparam_readXMLFileAtom: Got unexpected EOF reading parameter
file!\n");
    return 0;

}

VPRIVATE int readFlatFileLine(Vio *sock, Vparam_AtomData
*atom) {

    double dtmp;
    char tok[VMAX_BUFSIZE];

    VASSERT(atom != VNULL);

    if (Vio_scanf(sock, "%s", tok) != 1) return 0;
    if (strlen(tok) > VMAX_ARGLEN) {
        Vnm_print(2, "Vparam_readFlatFile:  string (%s) too long (%d)!\n",
          tok, strlen(tok));
        return 0;
    }
    strcpy(atom->resName, tok);
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (strlen(tok) > VMAX_ARGLEN) {
        Vnm_print(2, "Vparam_readFlatFile:  string (%s) too long (%d)!\n",
          tok, strlen(tok));
        return 0;
    }
    strcpy(atom->atomName, tok);
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &dtmp) != 1) {
        Vnm_print(2, "Vparam_readFlatFile:  Unexpected token (%s) while \
parsing charge!\n", tok);
        return 0;
    }
    atom->charge = dtmp;
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &dtmp) != 1) {
        Vnm_print(2, "Vparam_readFlatFile:  Unexpected token (%s) while \
parsing radius!\n", tok);
        return 0;
    }
    atom->radius = dtmp;
    VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
    if (sscanf(tok, "%lf", &dtmp) != 1) {
        Vnm_print(2, "Vparam_readFlatFile:  Unexpected token (%s) while \
parsing radius!\n", tok);
        return 0;
```

```
    }
    atom->epsilon = dtmp;

    return 1;

VERROR1:
    Vnm_print(2, "Vparam_readFlatFile: Got unexpected EOF reading parameter file!\n");
    return 0;
}
```

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-Huckel parameter.
- VPUBLIC double Vpbe_getDeblen (Vpbe *thee)

  Get Debye-Huckel screening length.
- VPUBLIC double Vpbe_getZkappa2 (Vpbe *thee)

  Get modified squared Debye-Huckel parameter.
- VPUBLIC double Vpbe_getZmagic (Vpbe *thee)

  Get charge scaling factor.
- VPUBLIC double Vpbe_getSoluteRadius (Vpbe *thee)

  Get sphere radius which bounds biomolecule.
- VPUBLIC double Vpbe_getSoluteXlen (Vpbe *thee)

  Get length of solute in x dimension.
- VPUBLIC double Vpbe_getSoluteYlen (Vpbe *thee)

  Get length of solute in y dimension.
- VPUBLIC double Vpbe_getSoluteZlen (Vpbe *thee)

  Get length of solute in z dimension.
- VPUBLIC double Vpbe_getSoluteCharge (Vpbe *thee)

  Get total solute charge.
- VPUBLIC double Vpbe_getzmem (Vpbe *thee)

  Get z position of the membrane bottom.
- VPUBLIC double Vpbe_getLmem (Vpbe *thee)

  Get length of the membrane (A)
  aauthor Michael Grabe.
- VPUBLIC double Vpbe_getmembraneDiel (Vpbe *thee)

  Get membrane dielectric constant.
- VPUBLIC double Vpbe_getmemv (Vpbe *thee)

  Get membrane potential (kT)
- VPUBLIC Vpbe * Vpbe_ctor (Valist *alist, int ionNum, double *ionConc, double *ionRadii, double *ionQ, double T, double soluteDiel, double solventDiel, double solventRadius, int focusFlag, double sdens, double z_mem, double L, double membraneDiel, double V)

  Construct Vpbe object.
- VPUBLIC int Vpbe_ctor2 (Vpbe *thee, Valist *alist, int ionNum, double *ionConc, double *ionRadii, double *ionQ, double T, double soluteDiel, double solventDiel, double solventRadius, int focusFlag, double sdens, double z_mem, double L, double membraneDiel, double V)

  FORTRAN stub to construct Vpbe objct.
- VPUBLIC void Vpbe_dtor (Vpbe **thee)

Object destructor.

- VPUBLIC void Vpbe_dtor2 (Vpbe *thee)

  FORTRAN stub object destructor.

- VPUBLIC double Vpbe_getCoulombEnergy1 (Vpbe *thee)

  Calculate coulombic energy of set of charges.

- VPUBLIC unsigned long int Vpbe_memChk (Vpbe *thee)

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

- VPUBLIC int Vpbe_getIons (Vpbe *thee, int *nion, double ionConc[MAXION], double ionRadii[MAXION], double ionQ[MAXION])

  Get information about the counterion species present.

9.77.1 Detailed Description

Class Vpbe methods.

Author

Nathan Baker

Version

Id:

vpbe.c 1774 2012-07-19 18:46:14Z sobolevnrm

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
```

```
   VASSERT(thee != VNULL);
   return thee->soluteCenter;
}

VPUBLIC double Vpbe_getSolventDiel(Vpbe *thee) {

   VASSERT(thee != VNULL);
   VASSERT(thee->paramFlag);
   return thee->solventDiel;
}

VPUBLIC double Vpbe_getSolventRadius(Vpbe *thee) {

   VASSERT(thee != VNULL);
   VASSERT(thee->paramFlag);
   return thee->solventRadius;
}

VPUBLIC double Vpbe_getMaxIonRadius(Vpbe *thee) {

   VASSERT(thee != VNULL);
   VASSERT(thee->paramFlag);
   return thee->maxIonRadius;
}

VPUBLIC double Vpbe_getXkappa(Vpbe *thee) {

   VASSERT(thee != VNULL);
   VASSERT(thee->paramFlag);
   return thee->xkappa;
}

VPUBLIC double Vpbe_getDeblen(Vpbe *thee) {

   VASSERT(thee != VNULL);
   VASSERT(thee->paramFlag);
   return thee->deblen;
}

VPUBLIC double Vpbe_getZkappa2(Vpbe *thee) {

   VASSERT(thee != VNULL);
   VASSERT(thee->paramFlag);
   return thee->zkappa2;
}

VPUBLIC double Vpbe_getZmagic(Vpbe *thee) {

   VASSERT(thee != VNULL);
   VASSERT(thee->paramFlag);
   return thee->zmagic;
}

VPUBLIC double Vpbe_getSoluteRadius(Vpbe *thee) {

   VASSERT(thee != VNULL);
   return thee->soluteRadius;
}

VPUBLIC double Vpbe_getSoluteXlen(Vpbe *thee) {

   VASSERT(thee != VNULL);
   return thee->soluteXlen;
}

VPUBLIC double Vpbe_getSoluteYlen(Vpbe *thee) {

   VASSERT(thee != VNULL);
   return thee->soluteYlen;
}

VPUBLIC double Vpbe_getSoluteZlen(Vpbe *thee) {

   VASSERT(thee != VNULL);
   return thee->soluteZlen;
}

VPUBLIC double Vpbe_getSoluteCharge(Vpbe *thee) {

   VASSERT(thee != VNULL);
   return thee->soluteCharge;
```

```
}

/* ////////////////////////////////////////////////////////////////////////
 // Routine:  Vpbe_getzmem
 // Purpose: This routine returns values stored in the structure thee.
 // Author:  Michael Grabe
VPUBLIC double Vpbe_getzmem(Vpbe *thee) {

 VASSERT(thee != VNULL);
 VASSERT(thee->param2Flag);
 return thee->z_mem;
}

/* ////////////////////////////////////////////////////////////////////////
 // Routine:  Vpbe_getLmem
 // Purpose: This routine returns values stored in the structure thee.
 // Author:  Michael Grabe
VPUBLIC double Vpbe_getLmem(Vpbe *thee) {

 VASSERT(thee != VNULL);
 VASSERT(thee->param2Flag);
 return thee->L;
}

/* ////////////////////////////////////////////////////////////////////////
 // Routine:  Vpbe_getmembraneDiel
 // Purpose: This routine returns values stored in the structure thee.
 // Author:  Michael Grabe
VPUBLIC double Vpbe_getmembraneDiel(Vpbe *thee) {

 VASSERT(thee != VNULL);
 VASSERT(thee->param2Flag);
 return thee->mmembraneDiel;
}

/* ////////////////////////////////////////////////////////////////////////
 // Routine:  Vpbe_getmemv
 // Purpose: This routine returns values stored in the structure thee.
 // Author:  Michael Grabe
VPUBLIC double Vpbe_getmemv(Vpbe *thee) {

 VASSERT(thee != VNULL);
 VASSERT(thee->param2Flag);
 return thee->V;
}

#endif /* if !defined(VINLINE_VPBE) */

/* ////////////////////////////////////////////////////////////////////////
// Class Vpbe: Non-inlineable methods

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 ) {

    /* Set up the structure */
    Vpbe *thee = VNULL;
    thee = (Vpbe*)Vmem_malloc(VNULL, 1, sizeof(Vpbe) );
    VASSERT( thee != VNULL);
    VASSERT( Vpbe_ctor2(thee, alist, ionNum, ionConc, ionRadii, ionQ,
      T, soluteDiel, solventDiel, solventRadius, focusFlag, sdens,
      z_mem, L, membraneDiel, V) );

    return thee;
}


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) {

```

```
    int i, iatom, inhash[3];
    double atomRadius;
    Vatom *atom;
    double center[3] = {0.0, 0.0, 0.0};
    double lower_corner[3] = {0.0, 0.0, 0.0};
    double upper_corner[3] = {0.0, 0.0, 0.0};
    double disp[3], dist, radius, charge, xmin, xmax, ymin, ymax, zmin, zmax;
    double x, y, z, netCharge;
    double nhash[3];
    const double N_A = 6.022045000e+23;
    const double e_c = 4.803242384e-10;
    const double k_B = 1.380662000e-16;
    const double pi  = 4. * VATAN(1.);

    /* Set up memory management object */
    thee->vmem = Vmem_ctor("APBS::VPBE");

    VASSERT(thee != VNULL);
    if (alist == VNULL) {
        Vnm_print(2, "Vpbe_ctor2: Got null pointer to Valist object!\n");
        return 0;
    }

    /* **** STUFF THAT GETS DONE FOR EVERYONE **** */
    /* Set pointers */
    thee->alist = alist;
    thee->paramFlag = 0;

    /* Determine solute center */
    center[0] = thee->alist->center[0];
    center[1] = thee->alist->center[1];
    center[2] = thee->alist->center[2];
    thee->soluteCenter[0] = center[0];
    thee->soluteCenter[1] = center[1];
    thee->soluteCenter[2] = center[2];

    /* Determine solute length and charge*/
    radius = 0;
    atom = Valist_getAtom(thee->alist, 0);
    xmin = Vatom_getPosition(atom)[0];
    xmax = Vatom_getPosition(atom)[0];
    ymin = Vatom_getPosition(atom)[1];
    ymax = Vatom_getPosition(atom)[1];
    zmin = Vatom_getPosition(atom)[2];
    zmax = Vatom_getPosition(atom)[2];
    charge = 0;
    for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist
); iatom++) {
        atom = Valist_getAtom(thee->alist, iatom);
        atomRadius = Vatom_getRadius(atom);
        x = Vatom_getPosition(atom)[0];
        y = Vatom_getPosition(atom)[1];
        z = Vatom_getPosition(atom)[2];
        if ((x+atomRadius) > xmax) xmax = x + atomRadius;
        if ((x-atomRadius) < xmin) xmin = x - atomRadius;
        if ((y+atomRadius) > ymax) ymax = y + atomRadius;
        if ((y-atomRadius) < ymin) ymin = y - atomRadius;
        if ((z+atomRadius) > zmax) zmax = z + atomRadius;
        if ((z-atomRadius) < zmin) zmin = z - atomRadius;
        disp[0] = (x - center[0]);
        disp[1] = (y - center[1]);
        disp[2] = (z - center[2]);
        dist = (disp[0]*disp[0]) + (disp[1]*disp[1]) + (disp[2]*disp[2]);
        dist = VSQRT(dist) + atomRadius;
        if (dist > radius) radius = dist;
        charge += Vatom_getCharge(Valist_getAtom(
thee->alist, iatom));
    }
    thee->soluteRadius = radius;
    Vnm_print(0, "Vpbe_ctor2:  solute radius = %g\n", radius);
    thee->soluteXlen = xmax - xmin;
    thee->soluteYlen = ymax - ymin;
    thee->soluteZlen = zmax - zmin;
    Vnm_print(0, "Vpbe_ctor2:  solute dimensions = %g x %g x %g\n",
            thee->soluteXlen, thee->soluteYlen, thee->
soluteZlen);
    thee->soluteCharge = charge;
    Vnm_print(0, "Vpbe_ctor2:  solute charge = %g\n", charge);

    /* Set parameters */
    thee->numIon = ionNum;
```

```
    if (thee->numIon >= MAXION) {
        Vnm_print(2, "Vpbe_ctor2:  Too many ion species (MAX = %d)!\n",
          MAXION);
        return 0;
    }
    thee->bulkIonicStrength = 0.0;
    thee->maxIonRadius = 0.0;
    netCharge = 0.0;
    for (i=0; i<thee->numIon; i++) {
        thee->ionConc[i] = ionConc[i];
        thee->ionRadii[i] = ionRadii[i];
        if (ionRadii[i] > thee->maxIonRadius) thee->maxIonRadius = ionRadii[i];
        thee->ionQ[i] = ionQ[i];
        thee->bulkIonicStrength += (0.5*ionConc[i]*VSQR(ionQ[i]));
        netCharge += (ionConc[i]*ionQ[i]);
    }
#ifndef VAPBSQUIET
    Vnm_print(1, "  Vpbe_ctor:  Using max ion radius (%g A) for exclusion \
function\n", thee->maxIonRadius);
#endif
    if (VABS(netCharge) > VSMALL) {
        Vnm_print(2, "Vpbe_ctor2:  You have a counterion charge imbalance!\n");
        Vnm_print(2, "Vpbe_ctor2:  Net charge conc. = %g M\n", netCharge);
        return 0;
    }
    thee->T = T;
    thee->soluteDiel = soluteDiel;
    thee->solventDiel = solventDiel;
    thee->solventRadius = solventRadius;

    /* Compute parameters:
     *
     * kappa^2 = (8 pi N_A e_c^2) I_s / (1000 eps_w k_B T)
     * kappa   = 0.325567 * I_s^{1/2}   angstroms^{-1}
     * deblen  = 1 / kappa
     *         = 3.071564378 * I_s^{1/2}   angstroms
     * \bar{kappa}^2 = eps_w * kappa^2
     * zmagic  = (4 * pi * e_c^2) / (k_B T)   (we scale the diagonal later)
     *         = 7046.528838
     */
    if (thee->T == 0.0) {
        Vnm_print(2, "Vpbe_ctor2:  You set the temperature to 0 K.\n");
        Vnm_print(2, "Vpbe_ctor2:  That violates the 3rd Law of Thermo.!");
        return 0;
    }
    if (thee->bulkIonicStrength == 0.) {
        thee->xkappa  = 0.;
        thee->deblen  = 0.;
        thee->zkappa2 = 0.;
    } else {
        thee->xkappa  = VSQRT( thee->bulkIonicStrength * 1.0e-16 *
            ((8.0 * pi * N_A * e_c*e_c) /
            (1000.0 * thee->solventDiel * k_B * T))
        );
        thee->deblen  = 1. / thee->xkappa;
        thee->zkappa2 = thee->solventDiel * VSQR(thee->xkappa);
    }
    Vnm_print(0, "Vpbe_ctor2:  bulk ionic strength = %g\n",
            thee->bulkIonicStrength);
    Vnm_print(0, "Vpbe_ctor2:  xkappa = %g\n", thee->xkappa);
    Vnm_print(0, "Vpbe_ctor2:  Debye length = %g\n", thee->deblen);
    Vnm_print(0, "Vpbe_ctor2:  zkappa2 = %g\n", thee->zkappa2);
    thee->zmagic  = ((4.0 * pi * e_c*e_c) / (k_B * thee->T)) * 1.0e+8;
    Vnm_print(0, "Vpbe_ctor2:  zmagic = %g\n", thee->zmagic);

    /* Compute accessibility objects:
     *   - Allow for extra room in the case of spline windowing
     *   - Place some limits on the size of the hash table in the case of very
     *     large molecules
     */
    if (thee->maxIonRadius > thee->solventRadius)
        radius = thee->maxIonRadius + MAX_SPLINE_WINDOW;
    else radius = thee->solventRadius + MAX_SPLINE_WINDOW;

 nhash[0] = (thee->soluteXlen)/0.5;
 nhash[1] = (thee->soluteYlen)/0.5;
 nhash[2] = (thee->soluteZlen)/0.5;
```

```
    for (i=0; i<3; i++) inhash[i] = (int)(nhash[i]);

 for (i=0;i<3;i++){
        if (inhash[i] < 3) inhash[i] = 3;
        if (inhash[i] > MAX_HASH_DIM) inhash[i] = MAX_HASH_DIM;
 }
    Vnm_print(0, "Vpbe_ctor2:  Constructing Vclist with %d x %d x %d table\n",
            inhash[0], inhash[1], inhash[2]);

    thee->clist = Vclist_ctor(thee->alist, radius, inhash,

            CLIST_AUTO_DOMAIN, lower_corner, upper_corner);

    VASSERT(thee->clist != VNULL);
    thee->acc = Vacc_ctor(thee->alist, thee->clist, sdens
);

    VASSERT(thee->acc != VNULL);

 /* SMPBE Added */
 thee->smsize = 0.0;
 thee->smvolume = 0.0;
 thee->ipkey = 0;

    thee->paramFlag = 1;

 /*-----------------------------------------------------------*/
 /* added by Michael Grabe                                    */
 /*-----------------------------------------------------------*/

    thee->z_mem = z_mem;
    thee->L = L;
    thee->membraneDiel = membraneDiel;
    thee->V = V;

 //    if (V != VNULL) thee->param2Flag = 1;
 //    else thee->param2Flag = 0;

 /*-----------------------------------------------------------*/

    return 1;
}

VPUBLIC void Vpbe_dtor(Vpbe **thee) {
    if ((*thee) != VNULL) {
        Vpbe_dtor2(*thee);
        Vmem_free(VNULL, 1, sizeof(Vpbe), (void **)thee);
        (*thee) = VNULL;
    }
}

VPUBLIC void Vpbe_dtor2(Vpbe *thee) {
    Vclist_dtor(&(thee->clist));
    Vacc_dtor(&(thee->acc));
    Vmem_dtor(&(thee->vmem));
}

VPUBLIC double Vpbe_getCoulombEnergy1(Vpbe *thee) {

    int i, j, k, natoms;

    double dist, *ipos, *jpos, icharge, jcharge;
    double energy = 0.0;
    double eps, T;
    Vatom *iatom, *jatom;
    Valist *alist;

    VASSERT(thee != VNULL);
    alist = Vpbe_getValist(thee);
    VASSERT(alist != VNULL);
    natoms = Valist_getNumberAtoms(alist);

    /* Do the sum */
    for (i=0; i<natoms; i++) {
        iatom = Valist_getAtom(alist,i);
        icharge = Vatom_getCharge(iatom);
        ipos = Vatom_getPosition(iatom);
        for (j=i+1; j<natoms; j++) {
            jatom = Valist_getAtom(alist,j);
            jcharge = Vatom_getCharge(jatom);
            jpos = Vatom_getPosition(jatom);
```

```
            dist = 0;
            for (k=0; k<3; k++) dist += ((ipos[k]-jpos[k])*(ipos[k]-jpos[k]));
            dist = VSQRT(dist);
            energy = energy + icharge*jcharge/dist;
        }
    }

    /* Convert the result to J */
    T = Vpbe_getTemperature(thee);
    eps = Vpbe_getSoluteDiel(thee);
    energy = energy*Vunit_ec*Vunit_ec/(4*Vunit_pi*Vunit_eps0*eps*(1.0e-10));

    /* Scale by Boltzmann energy */
    energy = energy/(Vunit_kb*T);

    return energy;
}

VPUBLIC unsigned long int Vpbe_memChk(Vpbe *thee) {

    unsigned long int memUse = 0;

    if (thee == VNULL) return 0;

    memUse = memUse + sizeof(Vpbe);
    memUse = memUse + (unsigned long int)Vacc_memChk(thee->acc);

    return memUse;
}

VPUBLIC int Vpbe_getIons(Vpbe *thee, int *nion, double ionConc[MAXION],
  double ionRadii[MAXION], double ionQ[MAXION]) {

    int i;

    VASSERT(thee != VNULL);

    *nion = thee->numIon;
    for (i=0; i<(*nion); i++) {
        ionConc[i] = thee->ionConc[i];
        ionRadii[i] = thee->ionRadii[i];
        ionQ[i] = thee->ionQ[i];
    }

    return *nion;
}
```

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.
00066 VPUBLIC int Vstring_strcasecmp(const char *s1, const char *s2) {
00067
00068 #if !defined(HAVE_STRCASECMP)
00069     unsigned char charmap[] = {
```

```
      0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
      0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
      0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f,
      0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
      0x28, 0x29, 0x2a, 0x2b, 0x2c, 0x2d, 0x2e, 0x2f,
      0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
      0x38, 0x39, 0x3a, 0x3b, 0x3c, 0x3d, 0x3e, 0x3f,
      0x40, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67,
      0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f,
      0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77,
      0x78, 0x79, 0x7a, 0x5b, 0x5c, 0x5d, 0x5e, 0x5f,
      0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67,
      0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f,
      0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77,
      0x78, 0x79, 0x7a, 0x7b, 0x7c, 0x7d, 0x7e, 0x7f,
      0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
      0x88, 0x89, 0x8a, 0x8b, 0x8c, 0x8d, 0x8e, 0x8f,
      0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97,
      0x98, 0x99, 0x9a, 0x9b, 0x9c, 0x9d, 0x9e, 0x9f,
      0xa0, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7,
      0xa8, 0xa9, 0xaa, 0xab, 0xac, 0xad, 0xae, 0xaf,
      0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7,
      0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0xbd, 0xbe, 0xbf,
      0xc0, 0xe1, 0xe2, 0xe3, 0xe4, 0xc5, 0xe6, 0xe7,
      0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef,
      0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
      0xf8, 0xf9, 0xfa, 0xdb, 0xdc, 0xdd, 0xde, 0xdf,
      0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7,
      0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef,
      0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
      0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff,
    };

    unsigned char u1, u2;

    for ( ; ; s1++, s2++) {
 u1 = (unsigned char) *s1;
 u2 = (unsigned char) *s2;
 if ((u1 == '\0') || (charmap[u1] != charmap[u2])) {
     break;
 }
    }
    return charmap[u1] - charmap[u2];

#else

    return strcasecmp(s1, s2);

#endif

}

/* ////////////////////////////////////////////////////////////////////////
// Routine:  Vstring_isdigit
//
//           Improves upon sscanf to see if a token is an int or not
//
//           Returns isdigit: 1 if a digit, 0 otherwise
VPUBLIC int Vstring_isdigit(const char *tok) {
    int i, isdigit, ti;
    char checkchar[1];
 char name[VMAX_BUFSIZE];
    strcpy(name,tok);
    isdigit = 1;
    for(i=0; ; i++){
      checkchar[0] =  name[i];
      if (name[i] == '\0'){
        break;
      }
      if (sscanf(checkchar, "%d", &ti) != 1){
        isdigit = 0;
        break;
      }
    }
    return isdigit;
}


char* Vstring_wrappedtext(const char* str, int right_margin, int left_padding)
{
```

```
    int span = right_margin - left_padding;
    int i = 0;
    int k = 0;
    int j = 0;
    int line_len = 0;
    int hyphenate = 0;
    char* wrap_str;
    int   wrap_len;
    int len = strlen( str );

    if( len == 0 )
        return VNULL;

    wrap_str = (char*)malloc( len * sizeof(char) );
    wrap_len = len;

    do
    {
        if( str[i] == ' ' )
        {
            i++;
        }
        else
        {
            if( k + right_margin + 2 > wrap_len )
            {
                wrap_len += right_margin + 2;
                wrap_str = (char*)realloc( wrap_str, wrap_len * sizeof( char ) );
            }


            if( i + span >= len )
            {
                hyphenate = 0;
                line_len = len - i;
            }
            else
            {
                j = span;
                do
                {
                    if( str[ i + j ] == ' ' )
                    {
                        hyphenate = 0;
                        line_len = j;
                        break;
                    }
                    else if( j == 0 )
                    {
                        hyphenate = 1;
                        line_len = span - 1;
                        break;
                    }
                    else
                    {
                        j--;
                    }
                } while( 1 );
            }

            memset( wrap_str + k, ' ', left_padding * sizeof( char ) );
            k += left_padding;

            memcpy( wrap_str + k, str + i, line_len * sizeof( char ) );
            k += line_len;
            i += line_len;

            if( hyphenate )
                wrap_str[k++] = '-';

            wrap_str[k++] = '\n';

            wrap_str[k] = '\0';
        }
    } while( i < len );

    return wrap_str;
}
```

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 1774 2012-07-19 18:46:14Z sobolevnrm

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

thee	Object to hold new grid data
iodev	I/O device
iofmt	I/O format
thost	Remote host name
fname	File name
title	Data title
pvec	Masking vector (0 = not written)

Definition at line 808 of file vgrid.c.

9.82 vgrid.h

```
00001
00062 #ifndef _VGRID_H_
00063 #define _VGRID_H_
00064
00065 #include "maloc/maloc.h"
00066 #include "apbs/vhal.h"
00067 #include "apbs/vstring.h"
00068
00069
00072 #define VGRID_DIGITS 6
00073
00079 struct sVgrid {
00080
00081     int nx;
00082     int ny;
00083     int nz;
00084     double hx;
00085     double hy;
00086     double hzed;
00087     double xmin;
00088     double ymin;
00089     double zmin;
00090     double xmax;
00091     double ymax;
00092     double zmax;
00093     double *data;
00094     int readdata;
00095     int ctordata;
00097     Vmem *mem;
00098 };
00099
00104 typedef struct sVgrid Vgrid;
00105
00106 #if !defined(VINLINE_VGRID)
00107
00115     VEXTERNC unsigned long int Vgrid_memChk(Vgrid *thee);
00116
00117 #else /* if defined(VINLINE_VGRID) */
00118
00126 #   define Vgrid_memChk(thee) (Vmem_bytes((thee)->vmem))
00127
00128 #endif /* if !defined(VINLINE_VPMG) */
00129
00147 VEXTERNC Vgrid*  Vgrid_ctor(int nx, int ny, int nz,
00148                   double hx, double hy, double hzed,
00149                   double xmin, double ymin, double zmin,
00150                   double *data);
00151
00170 VEXTERNC int Vgrid_ctor2(Vgrid *thee, int nx, int ny, int
      nz,
00171                   double hx, double hy, double hzed,
00172                   double xmin, double ymin, double zmin,
00173                   double *data);
00174
00183 VEXTERNC int Vgrid_value(Vgrid *thee, double x[3], double *
      value);
00184
00190 VEXTERNC void Vgrid_dtor(Vgrid **thee);
00191
```

```
VEXTERNC void Vgrid_dtor2(Vgrid *thee);

VEXTERNC int Vgrid_curvature(Vgrid *thee, double pt[3], int cflag,
  double *curv);

VEXTERNC int Vgrid_gradient(Vgrid *thee, double pt[3], double grad[3] );

VEXTERNC int Vgrid_readGZ(
        Vgrid *thee,
        const char *fname
        );

VEXTERNC void Vgrid_writeGZ(
       Vgrid *thee,
       const char *iodev,
       const char *iofmt,
       const char *thost,
       const char *fname,
       char *title,
       double *pvec
       );

VEXTERNC void Vgrid_writeUHBD(Vgrid *thee, const char *iodev,
  const char *iofmt, const char *thost, const char *fname, char *title,
  double *pvec);

VEXTERNC void Vgrid_writeDX(Vgrid *thee, const char *iodev,
  const char *iofmt,  const char *thost, const char *fname, char *title,
  double *pvec);

VEXTERNC int Vgrid_readDX(Vgrid *thee, const char *iodev, const char *iofmt,
  const char *thost, const char *fname);

VEXTERNC double Vgrid_integrate(Vgrid *thee);

VEXTERNC double Vgrid_normL1(Vgrid *thee);

VEXTERNC double Vgrid_normL2(Vgrid *thee);

VEXTERNC double Vgrid_normLinf(Vgrid *thee);

VEXTERNC double Vgrid_seminormH1(Vgrid *thee);

VEXTERNC double Vgrid_normH1(Vgrid *thee);

#endif
```

9.83 src/mg/apbs/vmgrid.h File Reference

Multiresolution oracle for Cartesian mesh data.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/vgrid.h"
```

Include dependency graph for vmgrid.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct sVmgrid

  Multiresoltion oracle for Cartesian mesh data.

Macros

- #define VMGRIDMAX 20

  The maximum number of levels in the grid hiearchy.

Typedefs

- typedef struct sVmgrid Vmgrid

  Declaration of the Vmgrid class as the Vgmrid structure.

Functions

- VEXTERNC Vmgrid ∗ Vmgrid_ctor ()

  Construct Vmgrid object.
- VEXTERNC int Vmgrid_ctor2 (Vmgrid ∗thee)

  Initialize Vmgrid object.
- VEXTERNC int Vmgrid_value (Vmgrid ∗thee, double x[3], double ∗value)

  Get potential value (from mesh or approximation) at a point.
- VEXTERNC void Vmgrid_dtor (Vmgrid ∗∗thee)

  Object destructor.
- VEXTERNC void Vmgrid_dtor2 (Vmgrid ∗thee)

  FORTRAN stub object destructor.
- VEXTERNC int Vmgrid_addGrid (Vmgrid ∗thee, Vgrid ∗grid)

  Add a grid to the hierarchy.
- VEXTERNC int Vmgrid_curvature (Vmgrid ∗thee, double pt[3], int cflag, double ∗curv)

  Get second derivative values at a point.
- VEXTERNC int Vmgrid_gradient (Vmgrid ∗thee, double pt[3], double grad[3])

  Get first derivative values at a point.
- VEXTERNC Vgrid ∗ Vmgrid_getGridByNum (Vmgrid ∗thee, int num)

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

  Get grid in hiearchy which contains specified point or VNULL.

9.83.1 Detailed Description

Multiresolution oracle for Cartesian mesh data.

Author

Nathan Baker

Version

Id:

vmgrid.h 1774 2012-07-19 18:46:14Z sobolevnrm

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_
```

```
/* Generic headers */
#include "maloc/maloc.h"
#include "apbs/vhal.h"

/* Headers specific to this file */
#include "apbs/vgrid.h"

#define VMGRIDMAX 20


struct sVmgrid {

    int ngrids;
    Vgrid *grids[VMGRIDMAX];
};

typedef struct sVmgrid Vmgrid;

VEXTERNC Vmgrid*  Vmgrid_ctor();

VEXTERNC int Vmgrid_ctor2(Vmgrid *thee);

VEXTERNC int Vmgrid_value(Vmgrid *thee, double x[3], double *
value);

VEXTERNC void Vmgrid_dtor(Vmgrid **thee);

VEXTERNC void Vmgrid_dtor2(Vmgrid *thee);

VEXTERNC int Vmgrid_addGrid(Vmgrid *thee, Vgrid *grid)
;


VEXTERNC int Vmgrid_curvature(Vmgrid *thee, double pt[3],
 int cflag,
  double *curv);

VEXTERNC int Vmgrid_gradient(Vmgrid *thee, double pt[3],
double grad[3] );

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

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

#endif
```

9.85 src/mg/apbs/vopot.h File Reference

Potential oracle for Cartesian mesh data.

```
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/vatom.h"
#include "apbs/valist.h"
#include "apbs/vmgrid.h"
#include "apbs/vunit.h"
#include "apbs/vpbe.h"
#include "apbs/pbeparm.h"
```

Include dependency graph for vopot.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct sVopot

  Electrostatic potential oracle for Cartesian mesh data.

Typedefs

- typedef struct sVopot Vopot

  Declaration of the Vopot class as the Vopot structure.

Functions

- VEXTERNC Vopot ∗ Vopot_ctor (Vmgrid ∗mgrid, Vpbe ∗pbe, Vbcfl bcfl)

  Construct Vopot object with values obtained from Vpmg_readDX (for example)
- VEXTERNC int Vopot_ctor2 (Vopot ∗thee, Vmgrid ∗mgrid, Vpbe ∗pbe, Vbcfl bcfl)

Initialize Vopot object with values obtained from Vpmg_readDX (for example)

- VEXTERNC int Vopot_pot (Vopot *thee, double x[3], double *pot)

  Get potential value (from mesh or approximation) at a point.
- VEXTERNC void Vopot_dtor (Vopot **thee)

  Object destructor.
- VEXTERNC void Vopot_dtor2 (Vopot *thee)

  FORTRAN stub object destructor.
- VEXTERNC int Vopot_curvature (Vopot *thee, double pt[3], int cflag, double *curv)

  Get second derivative values at a point.
- VEXTERNC int Vopot_gradient (Vopot *thee, double pt[3], double grad[3])

  Get first derivative values at a point.

9.85.1 Detailed Description

Potential oracle for Cartesian mesh data.

Author

Nathan Baker

Version

Id:

vopot.h 1774 2012-07-19 18:46:14Z sobolevnrm

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.

9.86 vopot.h

```
#ifndef _VOPOT_H_
#define _VOPOT_H_

/* Generic headers */
#include "maloc/maloc.h"
#include "apbs/vhal.h"

/* Specific headers */
#include "apbs/vatom.h"
#include "apbs/valist.h"
#include "apbs/vmgrid.h"
#include "apbs/vunit.h"
#include "apbs/vpbe.h"
#include "apbs/pbeparm.h"

struct sVopot {

    Vmgrid *mgrid;
    Vpbe   *pbe;
    Vbcfl bcfl;
};

typedef struct sVopot Vopot;

VEXTERNC Vopot*  Vopot_ctor(Vmgrid *mgrid, Vpbe *
pbe, Vbcfl bcfl);

VEXTERNC int Vopot_ctor2(Vopot *thee, Vmgrid *mgrid,
 Vpbe *pbe, Vbcfl bcfl);

VEXTERNC int Vopot_pot(Vopot *thee, double x[3], double *pot);

VEXTERNC void Vopot_dtor(Vopot **thee);

VEXTERNC void Vopot_dtor2(Vopot *thee);

VEXTERNC int Vopot_curvature(Vopot *thee, double pt[3], int
 cflag, double
  *curv);

VEXTERNC int Vopot_gradient(Vopot *thee, double pt[3],
double grad[3] );


#endif
```

9.87 src/mg/apbs/vpmg.h File Reference

Contains declarations for class Vpmg.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "apbs/vhal.h"
#include "apbs/vpmgp.h"
#include "apbs/vacc.h"
#include "apbs/vcap.h"
#include "apbs/vpbe.h"
#include "apbs/vgrid.h"
#include "apbs/mgparm.h"
#include "apbs/pbeparm.h"
#include "apbs/vmatrix.h"
#include "apbs/mgdrvd.h"
#include "apbs/newdrvd.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-Phi direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.

- VEXTERNC void Vpmg_qfNLDirectPolForce (Vpmg *thee, Vgrid *perm, Vgrid *nlInduced, int atomID, double force[3], double torque[3])

  q-Phi direct polarization force between permanent multipoles and non-local induced dipoles based on 5th Order B-Splines. Keep in mind that the "non-local" induced dipooles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

- VEXTERNC void Vpmg_ibDirectPolForce (Vpmg *thee, Vgrid *perm, Vgrid *induced, int atomID, double force[3])

  Ionic boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.

- VEXTERNC void Vpmg_ibNLDirectPolForce (Vpmg *thee, Vgrid *perm, Vgrid *nlInduced, int atomID, double force[3])

  Ionic boundary direct polarization force between permanent multipoles and non-local induced dipoles based on 5th order Keep in mind that the "non-local" induced dipooles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

- VEXTERNC void Vpmg_dbDirectPolForce (Vpmg *thee, Vgrid *perm, Vgrid *induced, int atomID, double force[3])

Dielectric boundary direct polarization force between permanent multipoles and induced dipoles, which are induced by the sum of the permanent intramolecular field and the permanent reaction field.

- VEXTERNC void Vpmg_dbNLDirectPolForce (Vpmg *thee, Vgrid *perm, Vgrid *nlInduced, int atomID, double force[3])

  Dielectric bounday direct polarization force between permanent multipoles and non-local induced dipoles. Keep in mind that the "non-local" induced dipooles are just a mathematical quantity that result from differentiation of the AMOEBA polarization energy.

- VEXTERNC void Vpmg_qfMutualPolForce (Vpmg *thee, Vgrid *induced, Vgrid *nlInduced, int atomID, double force[3])

  Mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

- VEXTERNC void Vpmg_ibMutualPolForce (Vpmg *thee, Vgrid *induced, Vgrid *nlInduced, int atomID, double force[3])

  Ionic boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

- VEXTERNC void Vpmg_dbMutualPolForce (Vpmg *thee, Vgrid *induced, Vgrid *nlInduced, int atomID, double force[3])

  Dielectric boundary mutual polarization force for induced dipoles based on 5th order B-Splines. This force arises due to self-consistent convergence of the solute induced dipoles and reaction field.

- VEXTERNC void Vpmg_printColComp (Vpmg *thee, char path[72], char title[72], char mxtype[3], int flag)

  Print out a column-compressed sparse matrix in Harwell-Boeing format.

- VPRIVATE void bcolcomp (int *iparm, double *rparm, int *iwork, double *rwork, double *values, int *rowind, int *colptr, int *flag)

  Build a column-compressed matrix in Harwell-Boeing format.

- VPRIVATE void bcolcomp2 (int *iparm, double *rparm, int *nx, int *ny, int *nz, int *iz, int *ipc, double *rpc, double *ac, double *cc, double *values, int *rowind, int *colptr, int *flag)

  Build a column-compressed matrix in Harwell-Boeing format.

- VPRIVATE void bcolcomp3 (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *ac, double *cc, double *values, int *rowind, int *colptr, int *flag)

  Build a column-compressed matrix in Harwell-Boeing format.

- VPRIVATE void bcolcomp4 (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *oC, double *cc, double *oE, double *oN, double *uC, double *values, int *rowind, int *colptr, int *flag)

  Build a column-compressed matrix in Harwell-Boeing format.

- VPRIVATE void pcolcomp (int *nrow, int *ncol, int *nnzero, double *values, int *rowind, int *colptr, char *path, char *title, char *mxtype)

  Print a column-compressed matrix in Harwell-Boeing format.

- VPRIVATE double bspline2 (double x)

  Evaluate a cubic B-spline.

- VPRIVATE double dbspline2 (double x)

  Evaluate a cubic B-spline derivative.

- VPRIVATE double VFCHI4 (int i, double f)

  Return 2.5 plus difference of i - f.

- VPRIVATE double bspline4 (double x)

  Evaluate a 5th Order B-Spline (4th order polynomial)

- VPRIVATE double dbspline4 (double x)

  Evaluate a 5th Order B-Spline derivative (4th order polynomial)

- VPRIVATE double d2bspline4 (double x)

  Evaluate the 2nd derivative of a 5th Order B-Spline.

- VPRIVATE double d3bspline4 (double x)

Evaluate the 3rd derivative of a 5th Order B-Spline.

- VPRIVATE double Vpmg_polarizEnergy (Vpmg ∗thee, int extFlag)

  Determines energy from polarizeable charge and interaction with fixed charges according to Rocchia et al.
- VPRIVATE double Vpmg_qfEnergyPoint (Vpmg ∗thee, int extFlag)

  Calculates charge-potential energy using summation over delta function positions (i.e. something like an Linf norm)
- VPRIVATE double Vpmg_qfEnergyVolume (Vpmg ∗thee, int extFlag)

  Calculates charge-potential energy as integral over a volume.
- VPRIVATE void Vpmg_splineSelect (int 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.
- VPRIVATE void bcfl1 (double size, double ∗apos, double charge, double xkappa, double pre1, double ∗gxcf, double ∗gycf, double ∗gzcf, double ∗xf, double ∗yf, double ∗zf, int nx, int ny, int nz)

  Increment all boundary points by pre1∗(charge/d)∗(exp(-xkappa∗(d-size))/(1+xkappa∗size) to add the effect of the Debye-Huckel potential due to a single charge.
- VPRIVATE void multipolebc (double r, double kappa, double eps_p, double eps_w, double rad, double tsr[3])

  This routine serves bcfl2. It returns (in tsr) the contraction independent portion of the Debye-Huckel potential tensor for a spherical ion with a central charge, dipole and quadrupole. See the code for an in depth description.
- VPRIVATE void bcCalc (Vpmg ∗thee)

  Fill boundary condition arrays.
- VPRIVATE void fillcoCoef (Vpmg ∗thee)

  Top-level driver to fill all operator coefficient arrays.
- VPRIVATE void fillcoCoefMap (Vpmg ∗thee)

  Fill operator coefficient arrays from pre-calculated maps.
- VPRIVATE void fillcoCoefMol (Vpmg ∗thee)

  Fill operator coefficient arrays from a molecular surface calculation.
- VPRIVATE void 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 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)

9.87.1 Detailed Description

Contains declarations for class Vpmg.

Version

Id:

vpmg.h 1774 2012-07-19 18:46:14Z sobolevnrm

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.

9.87.2 Function Documentation

9.87.2.1 VPRIVATE void bcCalc (Vpmg * *thee*)

Fill boundary condition arrays.

Author

Nathan Baker

Definition at line 4373 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.2 VPRIVATE void bcfl1 (double *size*, double * *apos*, double *charge*, double *xkappa*, double *pre1*, double * *gxcf*, double * *gycf*, double * *gzcf*, double * *xf*, double * *yf*, double * *zf*, int *nx*, int *ny*, int *nz*)

Increment all boundary points by pre1∗(charge/d)∗(exp(-xkappa∗(d-size))/(1+xkappa∗size) to add the effect of the Debye-Huckel potential due to a single charge.

Author

Nathan Baker

Parameters

apos	Size of the ion
charge	Position of the ion
xkappa	Charge of the ion
pre1	Exponential screening factor
gxcf	Unit- and dielectric-dependent prefactor
gycf	Set to x-boundary values
gzcf	Set to y-boundary values
xf	Set to z-boundary values
yf	Boundary point x-coordinates
zf	Boundary point y-coordinates
nx	Boundary point z-coordinates
ny	Number of grid points in x-direction
nz	Number of grid points in y-direction Number of grid points in y-direction

Definition at line 2570 of file vpmg.c.

9.87.2.3 VPRIVATE double bspline2 (double *x*)

Evaluate a cubic B-spline.

Author

Nathan Baker

Returns

Cubic B-spline value

Parameters

x	Position

Definition at line 5487 of file vpmg.c.

Here is the caller graph for this function:

9.87.2.4 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 7127 of file vpmg.c.

Here is the caller graph for this function:

9.87.2.5 VPRIVATE double d2bspline4 (double *x*)

Evaluate the 2nd derivative of a 5th Order B-Spline.

Author

: Michael Schnieders

Returns

2nd derivative of a 5th Order B-Spline

Parameters

x	Position

Definition at line 7193 of file vpmg.c.

Here is the caller graph for this function:

9.87.2.6 VPRIVATE double d3bspline4 (double *x*)

Evaluate the 3rd derivative of a 5th Order B-Spline.

Author

: Michael Schnieders

Returns

3rd derivative of a 5th Order B-Spline

Parameters

x	Position

Definition at line 7220 of file vpmg.c.

9.87.2.7 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 5503 of file vpmg.c.

Here is the caller graph for this function:

9.87.2.8 VPRIVATE double dbspline4 (double *x*)

Evaluate a 5th Order B-Spline derivative (4th order polynomial)

Author

: Michael Schnieders

Returns

5th Order B-Spline derivative

Parameters

x	Position

Definition at line 7161 of file vpmg.c.

Here is the caller graph for this function:

9.87.2.9 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 5278 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.10 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 5334 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.11 VPRIVATE void fillcoChargeSpline1 (Vpmg * *thee*)

Fill source term charge array from linear interpolation.

Author

Nathan Baker

Definition at line 5382 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.12 VPRIVATE void fillcoChargeSpline2 (Vpmg * *thee*)

Fill source term charge array from cubic spline interpolation.

Author

Nathan Baker

Definition at line 5519 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.13 VPRIVATE void fillcoCoef (Vpmg * *thee*)

Top-level driver to fill all operator coefficient arrays.

Author

Nathan Baker

Definition at line 5238 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.14 VPRIVATE void fillcoCoefMap (Vpmg * *thee*)

Fill operator coefficient arrays from pre-calculated maps.

Author

Nathan Baker

Definition at line 4480 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.15 VPRIVATE void fillcoCoefMol (Vpmg * *thee*)

Fill operator coefficient arrays from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4603 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.16 VPRIVATE void fillcoCoefMolDiel (Vpmg * *thee*)

Fill differential operator coefficient arrays from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4717 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.17 VPRIVATE void fillcoCoefMolDielNoSmooth (Vpmg * *thee*)

Fill differential operator coefficient arrays from a molecular surface calculation without smoothing.

Author

Nathan Baker

Definition at line 4728 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.18 VPRIVATE void fillcoCoefMolDielSmooth (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 4882 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.19 VPRIVATE void fillcoCoefMolIon (Vpmg * *thee*)

Fill ion (nonlinear) operator coefficient array from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4619 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.20 VPRIVATE void fillcoCoefSpline (Vpmg * *thee*)

Fill operator coefficient arrays from a spline-based surface calculation.

Author

Nathan Baker

Definition at line 5013 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.21 VPRIVATE void fillcoCoefSpline3 (Vpmg * *thee*)

Fill operator coefficient arrays from a 5th order polynomial based surface calculation.

Author

Michael Schnieders

Definition at line 10421 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.22 VPRIVATE void fillcoCoefSpline4 (Vpmg * *thee*)

Fill operator coefficient arrays from a 7th order polynomial based surface calculation.

Author

Michael Schnieders

Definition at line 9930 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.23 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:

9.87.2.24 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:

9.87.2.25 VPRIVATE void fillcoPermanentMultipole (Vpmg * *thee*)

Fill source term charge array for the use of permanent multipoles.

Author

Michael Schnieders

Definition at line 7231 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.26 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

tpos	Sphere radius
nx	Sphere position
ny	Number of grid points
nz	Number of grid points
hx	Number of grid points
hy	Grid spacing
hzed	Grid spacing
xmin	Grid spacing
ymin	Grid lower corner
zmin	Grid lower corner
array	Grid lower corner
markVal	Grid values Value to mark with

Definition at line 6840 of file vpmg.c.

Here is the caller graph for this function:

9.87.2.27 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

kappa	Distance to the boundary
eps_p	Exponential screening factor
eps_w	Solute dielectric
rad	Solvent dielectric
tsr	Radius of the sphere Contraction-independent portion of each tensor

Definition at line 3483 of file vpmg.c.

9.87.2.28 VPRIVATE void qfForceSpline1 (Vpmg * *thee*, double * *force*, int *atomID*)

Charge-field force due to a linear spline charge function.

Author

Nathan Baker

Parameters

atomID	Set to force Valist atom ID

Definition at line 6302 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.29 VPRIVATE void qfForceSpline2 (Vpmg * *thee*, double * *force*, int *atomID*)

Charge-field force due to a cubic spline charge function.

Author

Nathan Baker

Parameters

atomID	Set to force Valist atom ID

Definition at line 6439 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.30 VPRIVATE void qfForceSpline4 (Vpmg * *thee*, double * *force*, int *atomID*)

Charge-field force due to a quintic spline charge function.

Author

Michael Schnieders

Parameters

atomID	Set to force Valist atom ID

Definition at line 6552 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.31 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 7123 of file vpmg.c.

Here is the caller graph for this function:

9.87.2.32 VPRIVATE double Vpmg_polarizEnergy (Vpmg * *thee*, int *extFlag*)

Determines energy from polarizeable charge and interaction with fixed charges according to Rocchia et al.

Author

Nathan Baker

Returns

Energy in kT

Parameters

extFlag	If 1, add external energy contributions to result

Definition at line 1154 of file vpmg.c.

Here is the call graph for this function:

9.87.2.33 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

extFlag	If 1, add external energy contributions to result

Definition at line 1710 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.34 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

extFlag	If 1, add external energy contributions to result

Definition at line 1867 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.35 VPRIVATE double Vpmg_qmEnergySMPBE (Vpmg * *thee*, int *extFlag*)

Vpmg_qmEnergy for SMPBE.

Author

Vincent Chu

Definition at line 1496 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.36 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

acc	Surface method, currently VSM_SPLINE, VSM_SPLINE5, or VSM_SPLINE7
gpos	Accessibility object
win	Position array -> array[3]
infrad	Spline window
atom	Inflation radius
force	Atom object Force array -> array[3]

Definition at line 1899 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.87.2.37 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

source	Solution term vector
work1	Source term vector Work vector

Definition at line 6889 of file vpmg.c.

Here is the caller graph for this function:

9.88 vpmg.h

```
00001
00081 #ifndef _VPMG_H_
00082 #define _VPMG_H_
00083
```

```
/* Generic headers */
#include "apbscfg.h"
#include "maloc/maloc.h"

#include "apbs/vhal.h"
#include "apbs/vpmgp.h"
#include "apbs/vacc.h"
#include "apbs/vcap.h"
#include "apbs/vpbe.h"
#include "apbs/vgrid.h"
#include "apbs/mgparm.h"
#include "apbs/pbeparm.h"
#include "apbs/vmatrix.h"

#include "apbs/mgdrvd.h"
#include "apbs/newdrvd.h"
#include "apbs/mgsubd.h"
#include "apbs/mikpckd.h"
#include "apbs/matvecd.h"



#define VPMGMAXPART 2000

struct sVpmg {

  Vmem *vmem;
  Vpmgp *pmgp;
  Vpbe *pbe;
#ifdef BURY_FORTRAN
  Vpde *pde;
  Vmgdriver *mgdriver;
#endif

  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;
};

```

```
typedef struct sVpmg Vpmg;

/* ////////////////////////////////////////////////////////////////////////
#if !defined(VINLINE_VPMG)

    VEXTERNC unsigned long int Vpmg_memChk(
            Vpmg *thee
            );

#else /* if defined(VINLINE_VPMG) */

#   define Vpmg_memChk(thee) (Vmem_bytes((thee)->vmem))

#endif /* if !defined(VINLINE_VPMG) */

/* ////////////////////////////////////////////////////////////////////////

VEXTERNC Vpmg* Vpmg_ctor(
        Vpmgp *parms,
        Vpbe *pbe,
        int focusFlag,
        Vpmg *pmgOLD,
        MGparm *mgparm,
        PBEparm_calcEnergy energyFlag
        );

VEXTERNC int Vpmg_ctor2(
        Vpmg *thee,
        Vpmgp *parms,
        Vpbe *pbe,
        int focusFlag,
        Vpmg *pmgOLD,
        MGparm *mgparm,
        PBEparm_calcEnergy energyFlag
        );

VEXTERNC void Vpmg_dtor(
        Vpmg **thee
        );

VEXTERNC void Vpmg_dtor2(
        Vpmg *thee
        );

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
        );

VEXTERNC int Vpmg_solve(
        Vpmg *thee
        );

VEXTERNC int Vpmg_solveLaplace(
        Vpmg *thee
        );

VEXTERNC double Vpmg_energy(
        Vpmg *thee,
        int extFlag
        );

VEXTERNC double Vpmg_qfEnergy(
        Vpmg *thee,
        int extFlag
        );
```

```
VEXTERNC double Vpmg_qfAtomEnergy(
        Vpmg *thee,
        Vatom *atom
        );

VEXTERNC double Vpmg_qmEnergy(
        Vpmg *thee,
        int extFlag
        );


VEXTERNC double Vpmg_dielEnergy(
        Vpmg *thee,
        int extFlag
        );


VEXTERNC double Vpmg_dielGradNorm(
        Vpmg *thee
        );

VEXTERNC int Vpmg_force(
        Vpmg *thee,
        double *force,
        int atomID,
        Vsurf_Meth srfm,
        Vchrg_Meth chgm
        );

VEXTERNC int Vpmg_qfForce(
        Vpmg *thee,
        double *force,
        int atomID,
        Vchrg_Meth chgm
        );

VEXTERNC int Vpmg_dbForce(
        Vpmg *thee,
        double *dbForce,
        int atomID,
        Vsurf_Meth srfm
        );

VEXTERNC int Vpmg_ibForce(
        Vpmg *thee,
        double *force,
        int atomID,
        Vsurf_Meth srfm
        );

VEXTERNC void Vpmg_setPart(
        Vpmg *thee,
        double lowerCorner[3],
        double upperCorner[3],
        int bflags[6]
        );

VEXTERNC void Vpmg_unsetPart(
        Vpmg *thee
        );

VEXTERNC int Vpmg_fillArray(
        Vpmg *thee,
        double *vec,
        Vdata_Type type,
        double parm,
        Vhal_PBEType pbetype,
  PBEparm * pbeparm
        );

VPUBLIC void Vpmg_fieldSpline4(
             Vpmg *thee,
             int atomID,
             double field[3]
             );

VEXTERNC double Vpmg_qfPermanentMultipoleEnergy(
             Vpmg *thee,
             int atomID
             );

```

```
VEXTERNC void Vpmg_qfPermanentMultipoleForce(
             Vpmg *thee,
             int atomID,
             double force[3],
             double torque[3]
             );

VEXTERNC void Vpmg_ibPermanentMultipoleForce(
             Vpmg *thee,
             int atomID,
             double force[3]
             );

VEXTERNC void Vpmg_dbPermanentMultipoleForce(
             Vpmg *thee,
             int atomID,
             double force[3]
             );

VEXTERNC void Vpmg_qfDirectPolForce(
             Vpmg *thee,
             Vgrid *perm,
             Vgrid *induced,
             int atomID,
             double force[3],
             double torque[3]
             );

VEXTERNC void Vpmg_qfNLDirectPolForce(
             Vpmg *thee,
             Vgrid *perm,
             Vgrid *nlInduced,
             int atomID,
             double force[3],
             double torque[3]
             );

VEXTERNC void Vpmg_ibDirectPolForce(
             Vpmg *thee,
             Vgrid *perm,
             Vgrid *induced,
             int atomID,
             double force[3]
             );

VEXTERNC void Vpmg_ibNLDirectPolForce(
             Vpmg *thee,
             Vgrid *perm,
             Vgrid *nlInduced,
             int atomID,
             double force[3]
             );

VEXTERNC void Vpmg_dbDirectPolForce(
             Vpmg *thee,
             Vgrid *perm,
             Vgrid *induced,
             int atomID,
             double force[3]
             );

VEXTERNC void Vpmg_dbNLDirectPolForce(
             Vpmg *thee,
             Vgrid *perm,
             Vgrid *nlInduced,
             int atomID,
             double force[3]
             );

VEXTERNC void Vpmg_qfMutualPolForce(
             Vpmg *thee,
             Vgrid *induced,
             Vgrid *nlInduced,
             int atomID,
             double force[3]
             );

VEXTERNC void Vpmg_ibMutualPolForce(
             Vpmg *thee,
             Vgrid *induced,
             Vgrid *nlInduced,
```

```
            int atomID,
            double force[3]
            );

VEXTERNC void Vpmg_dbMutualPolForce(
            Vpmg *thee,
            Vgrid *induced,
            Vgrid *nlInduced,
            int atomID,
            double force[3]
            );

VEXTERNC void Vpmg_printColComp(
        Vpmg *thee,
        char path[72],
        char title[72],
        char mxtype[3],
        int flag
        );



VPRIVATE void bcolcomp(
        int    *iparm,  ///< @todo Document
        double *rparm,  ///< @todo Document
        int    *iwork,  ///< @todo Document
        double *rwork,  ///< @todo Document
        double *values, ///< @todo Document
        int    *rowind, ///< @todo Document
        int    *colptr, ///< @todo Document
        int    *flag
        );



VPRIVATE void bcolcomp2(
        int    *iparm,  ///< @todo Document
        double *rparm,  ///< @todo Document
        int    *nx,     ///< @todo Document
        int    *ny,     ///< @todo Document
        int    *nz,     ///< @todo Document
        int    *iz,     ///< @todo Document
        int    *ipc,    ///< @todo Document
        double *rpc,    ///< @todo Document
        double *ac,     ///< @todo Document
        double *cc,     ///< @todo Document
        double *values, ///< @todo Document
        int    *rowind, ///< @todo Document
        int    *colptr, ///< @todo Document
        int    *flag
        );



VPRIVATE void bcolcomp3(
        int    *nx,     ///< @todo Document
        int    *ny,     ///< @todo Document
        int    *nz,     ///< @todo Document
        int    *ipc,    ///< @todo Document
        double *rpc,    ///< @todo Document
        double *ac,     ///< @todo Document
        double *cc,     ///< @todo Document
        double *values, ///< @todo Document
        int    *rowind, ///< @todo Document
        int    *colptr, ///< @todo Document
        int    *flag    ///< @todo Document
        );



VPRIVATE void bcolcomp4(
        int    *nx,     ///< @todo Document
        int    *ny,     ///< @todo Document
        int    *nz,     ///< @todo Document
        int    *ipc,    ///< @todo Document
        double *rpc,    ///< @todo Document
        double *oC,     ///< @todo Document
        double *cc,     ///< @todo Document
        double *oE,     ///< @todo Document
        double *oN,     ///< @todo Document
        double *uC,     ///< @todo Document
```

```
        double *values, ///< @todo Document
        int    *rowind, ///< @todo Document
        int    *colptr, ///< @todo Document
        int    *flag    ///< @todo Document
        );



VPRIVATE void pcolcomp(
        int    *nrow,   ///< @todo Document
        int    *ncol,   ///< @todo Document
        int    *nnzero, ///< @todo Document
        double *values, ///< @todo Document
        int    *rowind, ///< @todo Document
        int    *colptr, ///< @todo Document
        char   *path,   ///< @todo Document
        char   *title,  ///< @todo Document
        char   *mxtype  ///< @todo Document
        );



/* ///////////////////////////////////////////////////////////////////////
// Internal routines

VPRIVATE double bspline2(
        double x
        );

VPRIVATE double dbspline2(
        double x
        );

VPRIVATE double VFCHI4(
  int i,
  double f
  );

VPRIVATE double bspline4(
         double x
         );

VPRIVATE double dbspline4(
         double x
         );

VPRIVATE double d2bspline4(
         double x
         );

VPRIVATE double d3bspline4(
         double x
         );

VPRIVATE double Vpmg_polarizEnergy(
         Vpmg *thee,
         int extFlag
         );
VPRIVATE double Vpmg_qfEnergyPoint(
        Vpmg *thee,
        int extFlag
        );

VPRIVATE double Vpmg_qfEnergyVolume(
        Vpmg *thee,
        int extFlag
        );

VPRIVATE void Vpmg_splineSelect(
  int srfm,
  Vacc *acc,
  double *gpos,
  double win,
  double infrad,
  Vatom *atom,
  double *force
  );

VPRIVATE void focusFillBound(
        Vpmg *thee,
        Vpmg *pmg
```

```
        );

VPRIVATE void bcfl1(
        double size,
        double *apos,
        double charge,
        double xkappa,
        double pre1,
        double *gxcf,
        double *gycf,
        double *gzcf,
        double *xf,
        double *yf,
        double *zf,
        int nx,
        int ny,
        int nz
        );

VPRIVATE void bcfl2(
        double size,
        double *apos,
        double charge,
        double *dipole,
        double *quad,
        double xkappa,
        double eps_p,
        double eps_w,
        double T,
        double *gxcf,
        double *gycf,
        double *gzcf,
        double *xf,
        double *yf,
        double *zf,
        int nx,
        int ny,
        int nz
        );

VPRIVATE void multipolebc(
        double r,
        double kappa,
        double eps_p,
        double eps_w,
        double rad,
        double tsr[3]
        );

VPRIVATE double bcfl1sp(
        double size,
        double *apos,
        double charge,
        double xkappa,
        double pre1,
        double *pos
        );

VPRIVATE void bcCalc(
        Vpmg *thee
        );

VPRIVATE void fillcoCoef(
        Vpmg *thee
        );

VPRIVATE void fillcoCoefMap(
        Vpmg *thee
        );

VPRIVATE void fillcoCoefMol(
        Vpmg *thee
        );

VPRIVATE void fillcoCoefMolIon(
        Vpmg *thee
        );

VPRIVATE void fillcoCoefMolDiel(
        Vpmg *thee
        );
```

```
VPRIVATE void fillcoCoefMolDielNoSmooth(
        Vpmg *thee
        );

VPRIVATE void fillcoCoefMolDielSmooth(
        Vpmg *thee
        );

VPRIVATE void fillcoCoefSpline(
        Vpmg *thee
        );

VPRIVATE void fillcoCoefSpline3(
  Vpmg *thee
  );

VPRIVATE void fillcoCoefSpline4(
        Vpmg *thee
        );

VPRIVATE Vrc_Codes fillcoCharge(
        Vpmg *thee
        );

VPRIVATE Vrc_Codes fillcoChargeMap(
        Vpmg *thee
        );

VPRIVATE void fillcoChargeSpline1(
        Vpmg *thee
        );

VPRIVATE void fillcoChargeSpline2(
        Vpmg *thee
        );

VPRIVATE void fillcoPermanentMultipole(
        Vpmg *thee
        );

VPRIVATE void fillcoInducedDipole(
        Vpmg *thee
        );

VPRIVATE void fillcoNLInducedDipole(
        Vpmg *thee
        );

VPRIVATE void extEnergy(
        Vpmg *thee,
        Vpmg *pmgOLD,
        PBEparm_calcEnergy extFlag,
        double partMin[3],
        double partMax[3],
        int bflags[6]
        );

VPRIVATE void qfForceSpline1(
        Vpmg *thee,
        double *force,
        int atomID
        );

VPRIVATE void qfForceSpline2(
        Vpmg *thee,
        double *force,
        int atomID
        );

VPRIVATE void qfForceSpline4(
        Vpmg *thee,
        double *force,
        int atomID
        );


VPRIVATE void zlapSolve(
        Vpmg *thee,
        double **solution,
        double **source,
```

```
        double **work1
        );

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
        );

VPRIVATE double Vpmg_qmEnergySMPBE(Vpmg *thee, int extFlag);
VPRIVATE double Vpmg_qmEnergyNONLIN(Vpmg *thee, int extFlag);



// Additional macros and definitions.  May not be needed

// Added by Vincent Chu 9/13/06 for SMPB
#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)))


#endif    /* ifndef _VPMG_H_ */

```

9.89 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.

9.89.1 Detailed Description

Contains declarations for class Vpmgp.

Version

Id:

vpmgp.h 1774 2012-07-19 18:46:14Z sobolevnrm

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.

9.90 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;
```

```
    int nlev;
    double hx;
    double hy;
    double hzed;
    int nonlin;
    /* ********** DERIVED PARAMETERS ********** */
    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;
    /* ********** PARAMETERS WITH DEFAULT VALUES ********** */
    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;
};

typedef struct sVpmgp Vpmgp;

/* ////////////////////////////////////////////////////////////////////////
// Class Vpmgp: Inlineable methods (vpmgp.c)

#if !defined(VINLINE_VPMGP)
#else /* if defined(VINLINE_VPMGP) */
#endif /* if !defined(VINLINE_VPMGP) */

/* ////////////////////////////////////////////////////////////////////////
// Class Vpmgp: Non-Inlineable methods (vpmgp.c)

VEXTERNC Vpmgp* Vpmgp_ctor(MGparm *mgparm);

VEXTERNC int Vpmgp_ctor2(Vpmgp *thee, MGparm *mgparm);

VEXTERNC void Vpmgp_dtor(Vpmgp **thee);

VEXTERNC void Vpmgp_dtor2(Vpmgp *thee);

VEXTERNC void Vpmgp_size(
 Vpmgp *thee
 );

VEXTERNC void Vpmgp_makeCoarse(
 int numLevel,
 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_ */
```

9.91 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]

9.91.1 Detailed Description

Class Vgrid methods.

Author

Nathan Baker

Version

Id:

vgrid.c 1774 2012-07-19 18:46:14Z sobolevnrm

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.

9.91.2 Function Documentation

9.91.2.1 VPUBLIC void Vgrid_writeGZ (Vgrid * *thee*, const char * *iodev*, const char * *iofmt*, const char * *thost*, const char * *fname*, char * *title*, double * *pvec*)

Write out OpenDX data in GZIP format.

Author

Dave Gohara

Parameters

thee	Object to hold new grid data
iodev	I/O device
iofmt	I/O format
thost	Remote host name
fname	File name
title	Data title
pvec	Masking vector (0 = not written)

Definition at line 808 of file vgrid.c.

9.92 vgrid.c

```
00001
00057 #include "apbscfg.h"
```

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

VEMBED(rcsid="$Id: vgrid.c 1774 2012-07-19 18:46:14Z sobolevnrm $")

#if !defined(VINLINE_VGRID)
    VPUBLIC unsigned long int Vgrid_memChk(Vgrid *thee) {
        return Vmem_bytes(thee->mem);
    }
#endif
#define IJK(i,j,k)  (((k)*(nx)*(ny))+((j)*(nx))+(i))

VPRIVATE char *MCwhiteChars = " =,;\t\n";
VPRIVATE char *MCcommChars  = "#%";
VPRIVATE double Vcompare;
VPRIVATE char Vprecision[26];

/* ////////////////////////////////////////////////////////////////////////
// Routine:  Vgrid_ctor
// Author:   Nathan Baker
VPUBLIC Vgrid* Vgrid_ctor(int nx,
                          int ny,
                          int nz,
                          double hx,
                          double hy,
                          double hzed,
                          double xmin,
                          double ymin,
                          double zmin,
                          double *data
                         ) {

    Vgrid *thee = VNULL;

    thee = (Vgrid*)Vmem_malloc(VNULL, 1, sizeof(Vgrid));
    VASSERT(thee != VNULL);
    VASSERT(Vgrid_ctor2(thee, nx, ny, nz, hx, hy, hzed,
                   xmin, ymin, zmin, data));

    return thee;
}

/* ////////////////////////////////////////////////////////////////////////
// Routine:  Vgrid_ctor2
// Author:   Nathan Baker
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) {

    if (thee == VNULL) return 0;
    thee->nx = nx;
    thee->ny = ny;
    thee->nz = nz;
    thee->hx = hx;
    thee->hy = hy;
    thee->hzed = hzed;
    thee->xmin = xmin;
    thee->xmax = xmin + (nx-1)*hx;
    thee->ymin = ymin;
    thee->ymax = ymin + (ny-1)*hy;
    thee->zmin = zmin;
    thee->zmax = zmin + (nz-1)*hzed;
    if (data == VNULL) {
        thee->ctordata = 0;
        thee->readdata = 0;
    } else {
        thee->ctordata = 1;
        thee->readdata = 0;
        thee->data = data;
    }

    thee->mem = Vmem_ctor("APBS:VGRID");

    Vcompare = pow(10,-1*(VGRID_DIGITS - 2));
    sprintf(Vprecision,"%%12.%de %%12.%de %%12.%de", VGRID_DIGITS,
            VGRID_DIGITS, VGRID_DIGITS);

    return 1;
}

/* ////////////////////////////////////////////////////////////////////////
```

```
// Routine:  Vgrid_dtor
// Author:   Nathan Baker
VPUBLIC void Vgrid_dtor(Vgrid **thee) {

    if ((*thee) != VNULL) {
        Vgrid_dtor2(*thee);
        Vmem_free(VNULL, 1, sizeof(Vgrid), (void **)thee);
        (*thee) = VNULL;
    }
}

/* ////////////////////////////////////////////////////////////////////////
// Routine:  Vgrid_dtor2
// Author:   Nathan Baker
VPUBLIC void Vgrid_dtor2(Vgrid *thee) {

    if (thee->readdata) {
        Vmem_free(thee->mem, (thee->nx*thee->ny*thee->nz), sizeof(double),
          (void **)&(thee->data));
    }
    Vmem_dtor(&(thee->mem));

}

/* ////////////////////////////////////////////////////////////////////////
// Routine:  Vgrid_value
// Author:   Nathan Baker
VPUBLIC int Vgrid_value(Vgrid *thee, double pt[3], double *value) {

    int nx, ny, nz, ihi, jhi, khi, ilo, jlo, klo;
    double hx, hy, hzed, xmin, ymin, zmin, ifloat, jfloat, kfloat;
    double xmax, ymax, zmax;
    double u, dx, dy, dz;

    if (thee == VNULL) {
        Vnm_print(2, "Vgrid_value:  Error -- got VNULL thee!\n");
        VASSERT(0);
    }
    if (!(thee->ctordata || thee->readdata)) {
        Vnm_print(2, "Vgrid_value:  Error -- no data available!\n");
        VASSERT(0);
    }

    nx = thee->nx;
    ny = thee->ny;
    nz = thee->nz;
    hx = thee->hx;
    hy = thee->hy;
    hzed = thee->hzed;
    xmin = thee->xmin;
    ymin = thee->ymin;
    zmin = thee->zmin;
    xmax = thee->xmax;
    ymax = thee->ymax;
    zmax = thee->zmax;

    u = 0;

    ifloat = (pt[0] - xmin)/hx;
    jfloat = (pt[1] - ymin)/hy;
    kfloat = (pt[2] - zmin)/hzed;

    ihi = (int)ceil(ifloat);
    jhi = (int)ceil(jfloat);
    khi = (int)ceil(kfloat);
    ilo = (int)floor(ifloat);
    jlo = (int)floor(jfloat);
    klo = (int)floor(kfloat);
 if (VABS(pt[0] - xmin) < Vcompare) ilo = 0;
 if (VABS(pt[1] - ymin) < Vcompare) jlo = 0;
 if (VABS(pt[2] - zmin) < Vcompare) klo = 0;
 if (VABS(pt[0] - xmax) < Vcompare) ihi = nx-1;
 if (VABS(pt[1] - ymax) < Vcompare) jhi = ny-1;
 if (VABS(pt[2] - zmax) < Vcompare) khi = nz-1;

    /* See if we're on the mesh */
    if ((ihi<nx) && (jhi<ny) && (khi<nz) &&
        (ilo>=0) && (jlo>=0) && (klo>=0)) {

        dx = ifloat - (double)(ilo);
        dy = jfloat - (double)(jlo);
```

```
        dz = kfloat - (double)(klo);
        u = dx      *dy      *dz      *(thee->data[IJK(ihi,jhi,khi)])
          + dx      *(1.0-dy)*dz      *(thee->data[IJK(ihi,jlo,khi)])
          + dx      *dy      *(1.0-dz)*(thee->data[IJK(ihi,jhi,klo)])
          + dx      *(1.0-dy)*(1.0-dz)*(thee->data[IJK(ihi,jlo,klo)])
          + (1.0-dx)*dy      *dz      *(thee->data[IJK(ilo,jhi,khi)])
          + (1.0-dx)*(1.0-dy)*dz      *(thee->data[IJK(ilo,jlo,khi)])
          + (1.0-dx)*dy      *(1.0-dz)*(thee->data[IJK(ilo,jhi,klo)])
          + (1.0-dx)*(1.0-dy)*(1.0-dz)*(thee->data[IJK(ilo,jlo,klo)]);

        *value = u;

        if (isnan(u)) {
            Vnm_print(2, "Vgrid_value:  Got NaN!\n");
            Vnm_print(2, "Vgrid_value:  (x, y, z) = (%4.3f, %4.3f, %4.3f)\n",
                    pt[0], pt[1], pt[2]);
            Vnm_print(2, "Vgrid_value:  (ihi, jhi, khi) = (%d, %d, %d)\n",
                    ihi, jhi, khi);
            Vnm_print(2, "Vgrid_value:  (ilo, jlo, klo) = (%d, %d, %d)\n",
                    ilo, jlo, klo);
            Vnm_print(2, "Vgrid_value:  (nx, ny, nz) = (%d, %d, %d)\n",
                    nx, ny, nz);
            Vnm_print(2, "Vgrid_value:  (dx, dy, dz) = (%4.3f, %4.3f, %4.3f)\n"
,
                    dx, dy, dz);
            Vnm_print(2, "Vgrid_value:  data[IJK(ihi,jhi,khi)] = %g\n",
                    thee->data[IJK(ihi,jhi,khi)]);
            Vnm_print(2, "Vgrid_value:  data[IJK(ihi,jlo,khi)] = %g\n",
                    thee->data[IJK(ihi,jlo,khi)]);
            Vnm_print(2, "Vgrid_value:  data[IJK(ihi,jhi,klo)] = %g\n",
                    thee->data[IJK(ihi,jhi,klo)]);
            Vnm_print(2, "Vgrid_value:  data[IJK(ihi,jlo,klo)] = %g\n",
                    thee->data[IJK(ihi,jlo,klo)]);
            Vnm_print(2, "Vgrid_value:  data[IJK(ilo,jhi,khi)] = %g\n",
                    thee->data[IJK(ilo,jhi,khi)]);
            Vnm_print(2, "Vgrid_value:  data[IJK(ilo,jlo,khi)] = %g\n",
                    thee->data[IJK(ilo,jlo,khi)]);
            Vnm_print(2, "Vgrid_value:  data[IJK(ilo,jhi,klo)] = %g\n",
                    thee->data[IJK(ilo,jhi,klo)]);
            Vnm_print(2, "Vgrid_value:  data[IJK(ilo,jlo,klo)] = %g\n",
                    thee->data[IJK(ilo,jlo,klo)]);
        }
        return 1;

    } else {

        *value = 0;
        return 0;

    }

    return 0;

}

/* ////////////////////////////////////////////////////////////////////////
// Routine:  Vgrid_curvature
//
//   Notes:  cflag=0 ==> Reduced Maximal Curvature
//           cflag=1 ==> Mean Curvature (Laplace)
//           cflag=2 ==> Gauss Curvature
//           cflag=3 ==> True Maximal Curvature
//
// Authors:  Stephen Bond and Nathan Baker
VPUBLIC int Vgrid_curvature(Vgrid *thee, double pt[3], int cflag,
  double *value) {

    double hx, hy, hzed, curv;
    double dxx, dyy, dzz;
    double uleft, umid, uright, testpt[3];

    if (thee == VNULL) {
        Vnm_print(2, "Vgrid_curvature:  Error -- got VNULL thee!\n");
        VASSERT(0);
    }
    if (!(thee->ctordata || thee->readdata)) {
        Vnm_print(2, "Vgrid_curvature:  Error -- no data available!\n");
        VASSERT(0);
    }

    hx = thee->hx;
```

```
    hy = thee->hy;
    hzed = thee->hzed;

    curv = 0.0;

    testpt[0] = pt[0];
    testpt[1] = pt[1];
    testpt[2] = pt[2];

    /* Compute 2nd derivative in the x-direction */
    VJMPERR1(Vgrid_value( thee, testpt, &umid));
    testpt[0] = pt[0] - hx;
    VJMPERR1(Vgrid_value( thee, testpt, &uleft));
    testpt[0] = pt[0] + hx;
    VJMPERR1(Vgrid_value( thee, testpt, &uright));
    testpt[0] = pt[0];

    dxx = (uright - 2*umid + uleft)/(hx*hx);

    /* Compute 2nd derivative in the y-direction */
    VJMPERR1(Vgrid_value( thee, testpt, &umid));
    testpt[1] = pt[1] - hy;
    VJMPERR1(Vgrid_value( thee, testpt, &uleft));
    testpt[1] = pt[1] + hy;
    VJMPERR1(Vgrid_value( thee, testpt, &uright));
    testpt[1] = pt[1];

    dyy = (uright - 2*umid + uleft)/(hy*hy);

    /* Compute 2nd derivative in the z-direction */
    VJMPERR1(Vgrid_value( thee, testpt, &umid));
    testpt[2] = pt[2] - hzed;
    VJMPERR1(Vgrid_value( thee, testpt, &uleft));
    testpt[2] = pt[2] + hzed;
    VJMPERR1(Vgrid_value( thee, testpt, &uright));

    dzz = (uright - 2*umid + uleft)/(hzed*hzed);


    if ( cflag == 0 ) {
        curv = fabs(dxx);
        curv = ( curv > fabs(dyy) ) ? curv : fabs(dyy);
        curv = ( curv > fabs(dzz) ) ? curv : fabs(dzz);
    } else if ( cflag == 1 ) {
        curv = (dxx + dyy + dzz)/3.0;
    } else {
        Vnm_print(2, "Vgrid_curvature:  support for cflag = %d not available!\n", cflag);
        VASSERT( 0 ); /* Feature Not Coded Yet! */
    }

    *value = curv;
    return 1;

    VERROR1:
        return 0;

}

/* ////////////////////////////////////////////////////////////////////////////
// Routine:  Vgrid_gradient
//
// Authors:  Nathan Baker and Stephen Bond
VPUBLIC int Vgrid_gradient(Vgrid *thee, double pt[3], double grad[3]) {

    double hx, hy, hzed;
    double uleft, umid, uright, testpt[3];
    int haveleft, haveright;

    if (thee == VNULL) {
        Vnm_print(2, "Vgrid_gradient:  Error -- got VNULL thee!\n");
        VASSERT(0);
    }
    if (!(thee->ctordata || thee->readdata)) {
        Vnm_print(2, "Vgrid_gradient:  Error -- no data available!\n");
        VASSERT(0);
    }

    hx = thee->hx;
    hy = thee->hy;
    hzed = thee->hzed;
```

```

    /* Compute derivative in the x-direction */
    testpt[0] = pt[0];
    testpt[1] = pt[1];
    testpt[2] = pt[2];
    VJMPERR1( Vgrid_value( thee, testpt, &umid));
    testpt[0] = pt[0] - hx;
    if (Vgrid_value( thee, testpt, &uleft)) haveleft = 1;
    else haveleft = 0;
    testpt[0] = pt[0] + hx;
    if (Vgrid_value( thee, testpt, &uright)) haveright = 1;
    else haveright = 0;
    if (haveright && haveleft) grad[0] = (uright - uleft)/(2*hx);
    else if (haveright) grad[0] = (uright - umid)/hx;
    else if (haveleft) grad[0] = (umid - uleft)/hx;
    else VJMPERR1(0);

    /* Compute derivative in the y-direction */
    testpt[0] = pt[0];
    testpt[1] = pt[1];
    testpt[2] = pt[2];
    VJMPERR1(Vgrid_value(thee, testpt, &umid));
    testpt[1] = pt[1] - hy;
    if (Vgrid_value( thee, testpt, &uleft)) haveleft = 1;
    else haveleft = 0;
    testpt[1] = pt[1] + hy;
    if (Vgrid_value( thee, testpt, &uright)) haveright = 1;
    else haveright = 0;
    if (haveright && haveleft) grad[1] = (uright - uleft)/(2*hy);
    else if (haveright) grad[1] = (uright - umid)/hy;
    else if (haveleft) grad[1] = (umid - uleft)/hy;
    else VJMPERR1(0);

    /* Compute derivative in the z-direction */
    testpt[0] = pt[0];
    testpt[1] = pt[1];
    testpt[2] = pt[2];
    VJMPERR1(Vgrid_value(thee, testpt, &umid));
    testpt[2] = pt[2] - hzed;
    if (Vgrid_value( thee, testpt, &uleft)) haveleft = 1;
    else haveleft = 0;
    testpt[2] = pt[2] + hzed;
    if (Vgrid_value( thee, testpt, &uright)) haveright = 1;
    else haveright = 0;
    if (haveright && haveleft) grad[2] = (uright - uleft)/(2*hzed);
    else if (haveright) grad[2] = (uright - umid)/hzed;
    else if (haveleft) grad[2] = (umid - uleft)/hzed;
    else VJMPERR1(0);

    return 1;

    VERROR1:
        return 0;

}

/* ///////////////////////////////////////////////////////////////////////////
 // Routine:  Vgrid_readGZ
 //
 // Author:   David Gohara
#ifdef HAVE_ZLIB
#define off_t long
#include "../../contrib/zlib/zlib.h"
#endif
VPUBLIC int Vgrid_readGZ(Vgrid *thee, const char *fname) {

#ifdef HAVE_ZLIB
 int i, j, k;
 int len; // Temporary counter variable for loop conditionals
 int q, itmp, u, header, incr;
 double *temp;
    double dtmp1, dtmp2, dtmp3;
 gzFile infile;
 char line[VMAX_ARGLEN];

 header = 0;

 /* Check to see if the existing data is null and, if not, clear it out */
    if (thee->data != VNULL) {
        Vnm_print(1, "%s:  destroying existing data!\n", __func__);
  Vmem_free(thee->mem, thee->nx * thee->ny * thee->nz, sizeof(double),
```

```
      (void **)&(thee->data));
        }

    thee->readdata = 1;
    thee->ctordata = 0;

infile = gzopen(fname, "rb");
if (infile == Z_NULL) {
 Vnm_print(2, "%s:  Problem opening compressed file %s\n", __func__, fname);
        return VRC_FAILURE;
    }

thee->hx = 0.0;
thee->hy = 0.0;
thee->hzed = 0.0;

//read data here
while (header < 7) {
 if(gzgets(infile, line, VMAX_ARGLEN) == Z_NULL){
  return VRC_FAILURE;
 }

 // Skip comments and newlines
 if(strncmp(line, "#", 1) == 0) continue;
 if(line[0] == '\n') continue;

 switch (header) {
  case 0:
   sscanf(line, "object 1 class gridpositions counts %d %d %d",
       &(thee->nx),&(thee->ny),&(thee->nz));
   break;
  case 1:
   sscanf(line, "origin %lf %lf %lf",
       &(thee->xmin),&(thee->ymin),&(thee->zmin));
   break;
  case 2:
  case 3:
  case 4:
   sscanf(line, "delta %lf %lf %lf",&dtmp1,&dtmp2,&dtmp3);
   thee->hx += dtmp1;
   thee->hy += dtmp2;
   thee->hzed += dtmp3;
   break;
  default:
   break;
 }

 header++;
}

/* Allocate space for the data */
Vnm_print(0, "%s:  allocating %d x %d x %d doubles for storage\n",
 __func__, thee->nx, thee->ny, thee->nz);
len = thee->nx * thee->ny * thee->nz;

    thee->data = VNULL;
thee->data = Vmem_malloc(thee->mem, len, sizeof(double));
    if (thee->data == VNULL) {
        Vnm_print(2, "%s:  Unable to allocate space for data!\n", __func__);
        return 0;
    }

/* Allocate a temporary buffer to store the compressed
 * data into (column major order). Add 2 to ensure the buffer is
 * big enough to take extra data on the final read loop.
 */
temp = (double *)malloc(len * (2 * sizeof(double)));

for (i = 0; i < len; i += 3){
 memset(&line, 0, sizeof(line));
 gzgets(infile, line, VMAX_ARGLEN);
 sscanf(line, "%lf %lf %lf", &temp[i], &temp[i+1], &temp[i+2]);
}

/* Now move the data to row major order */
incr = 0;
for (i=0; i<thee->nx; i++) {
 for (j=0; j<thee->ny; j++) {
  for (k=0; k<thee->nz; k++) {
   u = k*(thee->nx)*(thee->ny)+j*(thee->nx)+i;
   (thee->data)[u] = temp[incr++];
```

```
    }
   }
  }

  /* calculate grid maxima */
  thee->xmax = thee->xmin + (thee->nx-1)*thee->hx;
  thee->ymax = thee->ymin + (thee->ny-1)*thee->hy;
  thee->zmax = thee->zmin + (thee->nz-1)*thee->hzed;

  /* Close off the socket */
  gzclose(infile);
  free(temp);
#else

  Vnm_print(0, "WARNING\n");
  Vnm_print(0, "Vgrid_readGZ:  gzip read/write support is disabled in this build
 \n");
  Vnm_print(0, "Vgrid_readGZ:  configure and compile without the --disable-zlib
  flag.\n");
  Vnm_print(0, "WARNING\n");
#endif
  return VRC_SUCCESS;
}

VPUBLIC int Vgrid_readDX(Vgrid *thee,
                         const char *iodev,
                         const char *iofmt,
                         const char *thost,
                         const char *fname
                        ) {

    int i,
        j,
        k,
        itmp,
        u;
    double dtmp;
    char tok[VMAX_BUFSIZE];
    Vio *sock;

    /* Check to see if the existing data is null and, if not, clear it out */
    if (thee->data != VNULL) {
        Vnm_print(1, "Vgrid_readDX:  destroying existing data!\n");
  Vmem_free(thee->mem, (thee->nx*thee->ny*thee->nz), sizeof(double),
          (void **)&(thee->data)); }
    thee->readdata = 1;
    thee->ctordata = 0;

    /* Set up the virtual socket */
    sock = Vio_ctor(iodev,iofmt,thost,fname,"r");
    if (sock == VNULL) {
        Vnm_print(2, "Vgrid_readDX: Problem opening virtual socket %s\n",
          fname);
        return 0;
    }
    if (Vio_accept(sock, 0) < 0) {
        Vnm_print(2, "Vgrid_readDX: Problem accepting virtual socket %s\n",
          fname);
        return 0;
    }

    Vio_setWhiteChars(sock, MCwhiteChars);
    Vio_setCommChars(sock, MCcommChars);

    /* Read in the DX regular positions */
    /* Get "object" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "object"));
    /* Get "1" */
    VJMPERR2(1 == Vio_scanf(sock, "%d", &itmp));
    /* Get "class" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "class"));
    /* Get "gridpositions" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "gridpositions"));
    /* Get "counts" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "counts"));
    /* Get nx */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
```

```
    VJMPERR1(1 == sscanf(tok, "%d", &(thee->nx)));
    /* Get ny */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(1 == sscanf(tok, "%d", &(thee->ny)));
    /* Get nz */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(1 == sscanf(tok, "%d", &(thee->nz)));
    Vnm_print(0, "Vgrid_readDX:  Grid dimensions %d x %d x %d grid\n",
     thee->nx, thee->ny, thee->nz);
    /* Get "origin" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "origin"));
    /* Get xmin */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(1 == sscanf(tok, "%lf", &(thee->xmin)));
    /* Get ymin */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(1 == sscanf(tok, "%lf", &(thee->ymin)));
    /* Get zmin */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(1 == sscanf(tok, "%lf", &(thee->zmin)));
    Vnm_print(0, "Vgrid_readDX:  Grid origin = (%g, %g, %g)\n",
      thee->xmin, thee->ymin, thee->zmin);
    /* Get "delta" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "delta"));
    /* Get hx */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(1 == sscanf(tok, "%lf", &(thee->hx)));
    /* Get 0.0 */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
    VJMPERR1(dtmp == 0.0);
    /* Get 0.0 */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
    VJMPERR1(dtmp == 0.0);
    /* Get "delta" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "delta"));
    /* Get 0.0 */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
    VJMPERR1(dtmp == 0.0);
    /* Get hy */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(1 == sscanf(tok, "%lf", &(thee->hy)));
    /* Get 0.0 */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
    VJMPERR1(dtmp == 0.0);
    /* Get "delta" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "delta"));
    /* Get 0.0 */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
    VJMPERR1(dtmp == 0.0);
    /* Get 0.0 */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
    VJMPERR1(dtmp == 0.0);
    /* Get hz */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(1 == sscanf(tok, "%lf", &(thee->hzed)));
    Vnm_print(0, "Vgrid_readDX:  Grid spacings = (%g, %g, %g)\n",
      thee->hx, thee->hy, thee->hzed);
    /* Get "object" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "object"));
    /* Get "2" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    /* Get "class" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "class"));
    /* Get "gridconnections" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "gridconnections"));
    /* Get "counts" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "counts"));
```

```
    /* Get the dimensions again */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    /* Get "object" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "object"));
    /* Get # */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    /* Get "class" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "class"));
    /* Get "array" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "array"));
    /* Get "type" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "type"));
    /* Get "double" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "double"));
    /* Get "rank" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "rank"));
    /* Get # */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    /* Get "items" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "items"));
    /* Get # */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(1 == sscanf(tok, "%d", &itmp));
    VJMPERR1(((thee->nx)*(thee->ny)*(thee->nz)) == itmp);
    /* Get "data" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "data"));
    /* Get "follows" */
    VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
    VJMPERR1(!strcmp(tok, "follows"));

    /* Allocate space for the data */
    Vnm_print(0, "Vgrid_readDX:  allocating %d x %d x %d doubles for storage\n"
,
      thee->nx, thee->ny, thee->nz);
    thee->data = VNULL;
    thee->data = (double*)Vmem_malloc(thee->mem, (thee->nx)*(thee->ny
)*(thee->nz),
      sizeof(double));
    if (thee->data == VNULL) {
        Vnm_print(2, "Vgrid_readDX:  Unable to allocate space for data!\n");
        return 0;
    }

    for (i=0; i<thee->nx; i++) {
        for (j=0; j<thee->ny; j++) {
            for (k=0; k<thee->nz; k++) {
                u = k*(thee->nx)*(thee->ny)+j*(thee->nx)+i;
                VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
                VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
                (thee->data)[u] = dtmp;
            }
        }
    }

    /* calculate grid maxima */
    thee->xmax = thee->xmin + (thee->nx-1)*thee->hx;
    thee->ymax = thee->ymin + (thee->ny-1)*thee->hy;
    thee->zmax = thee->zmin + (thee->nz-1)*thee->hzed;

    /* Close off the socket */
    Vio_acceptFree(sock);
    Vio_dtor(&sock);

    return 1;

  VERROR1:
    Vio_dtor(&sock);
    Vnm_print(2, "Vgrid_readDX:  Format problem with input file <%s>\n",
      fname);
    return 0;

```

```
  VERROR2:
    Vio_dtor(&sock);
    Vnm_print(2, "Vgrid_readDX:  I/O problem with input file <%s>\n",
      fname);
    return 0;



}

/* ////////////////////////////////////////////////////////////////////////
 // Routine:  Vgrid_writeGZ
 //
 // Author:   Nathan Baker
VPUBLIC void Vgrid_writeGZ(Vgrid *thee, const char *iodev, const char *iofmt,
       const char *thost, const char *fname, char *title, double *pvec) {

#ifdef HAVE_ZLIB
 double xmin, ymin, zmin, hx, hy, hzed;

 int nx, ny, nz;
 int icol, i, j, k, u, usepart, nxPART, nyPART, nzPART, gotit;
 double x, y, z, xminPART, yminPART, zminPART;

 int txyz;
 double txmin, tymin, tzmin;

 char header[8196];
 char footer[8196];
 char line[80];
 char newline[] = "\n";
 gzFile outfile;
 char precFormat[VMAX_BUFSIZE];

 if (thee == VNULL) {
  Vnm_print(2, "Vgrid_writeGZ:  Error -- got VNULL thee!\n");
  VASSERT(0);
 }
 if (!(thee->ctordata || thee->readdata)) {
  Vnm_print(2, "Vgrid_writeGZ:  Error -- no data available!\n");
  VASSERT(0);
 }

 hx = thee->hx;
 hy = thee->hy;
 hzed = thee->hzed;
 nx = thee->nx;
 ny = thee->ny;
 nz = thee->nz;
 xmin = thee->xmin;
 ymin = thee->ymin;
 zmin = thee->zmin;

 if (pvec == VNULL) usepart = 0;
 else usepart = 1;

 /* Set up the virtual socket */
 Vnm_print(0, "Vgrid_writeGZ:  Opening file...\n");
 outfile = gzopen(fname, "wb");

 if (usepart) {
  /* Get the lower corner and number of grid points for the local
   * partition */
  xminPART = VLARGE;
  yminPART = VLARGE;
  zminPART = VLARGE;
  nxPART = 0;
  nyPART = 0;
  nzPART = 0;
  /* First, search for the lower corner */
  for (k=0; k<nz; k++) {
   z = k*hzed + zmin;
   for (j=0; j<ny; j++) {
    y = j*hy + ymin;
    for (i=0; i<nx; i++) {
     x = i*hx + xmin;
     if (pvec[IJK(i,j,k)] > 0.0) {
      if (x < xminPART) xminPART = x;
      if (y < yminPART) yminPART = y;
      if (z < zminPART) zminPART = z;
     }
```

```
    }
   }
  }
  /* Now search for the number of grid points in the z direction */
  for (k=0; k<nz; k++) {
   gotit = 0;
   for (j=0; j<ny; j++) {
    for (i=0; i<nx; i++) {
     if (pvec[IJK(i,j,k)] > 0.0) {
      gotit = 1;
      break;
     }
    }
    if (gotit) break;
   }
   if (gotit) nzPART++;
  }
  /* Now search for the number of grid points in the y direction */
  for (j=0; j<ny; j++) {
   gotit = 0;
   for (k=0; k<nz; k++) {
    for (i=0; i<nx; i++) {
     if (pvec[IJK(i,j,k)] > 0.0) {
      gotit = 1;
      break;
     }
    }
    if (gotit) break;
   }
   if (gotit) nyPART++;
  }
  /* Now search for the number of grid points in the x direction */
  for (i=0; i<nx; i++) {
   gotit = 0;
   for (k=0; k<nz; k++) {
    for (j=0; j<ny; j++) {
     if (pvec[IJK(i,j,k)] > 0.0) {
      gotit = 1;
      break;
     }
    }
    if (gotit) break;
   }
   if (gotit) nxPART++;
  }

  if ((nxPART != nx) || (nyPART != ny) || (nzPART != nz)) {
   Vnm_print(0, "Vgrid_writeGZ:  printing only subset of domain\n");
  }

  txyz = (nxPART*nyPART*nzPART);
  txmin = xminPART;
  tymin = yminPART;
  tzmin = zminPART;

 }else {

  txyz = (nx*ny*nz);
  txmin = xmin;
  tymin = ymin;
  tzmin = zmin;

 }

 /* Write off the title (if we're not XDR) */
 sprintf(header,
   "# Data from %s\n" \
   "# \n"       \
   "# %s\n"   \
   "# \n"       \
   "object 1 class gridpositions counts %i %i %i\n" \
   "origin %12.6e %12.6e %12.6e\n" \
   "delta %12.6e 0.000000e+00 0.000000e+00\n"  \
   "delta 0.000000e+00 %12.6e 0.000000e+00\n"  \
   "delta 0.000000e+00 0.000000e+00 %12.6e\n"  \
   "object 2 class gridconnections counts %i %i %i\n"\
   "object 3 class array type double rank 0 items %i data follows\n",
   PACKAGE_STRING,title,nx,ny,nz,txmin,tymin,tzmin,
   hx,hy,hzed,nx,ny,nz,txyz);
 gzwrite(outfile, header, strlen(header)*sizeof(char));

```

```
/* Now write the data */
icol = 0;
for (i=0; i<nx; i++) {
 for (j=0; j<ny; j++) {
  for (k=0; k<nz; k++) {
   u = k*(nx)*(ny)+j*(nx)+i;
   if (pvec[u] > 0.0) {
    sprintf(line, "%12.6e ", thee->data[u]);
    gzwrite(outfile, line, strlen(line)*sizeof(char));
    icol++;
    if (icol == 3) {
     icol = 0;
     gzwrite(outfile, newline, strlen(newline)*sizeof(char));
    }
   }
  }
 }
}
if(icol < 3){
 char newline[] = "\n";
 gzwrite(outfile, newline, strlen(newline)*sizeof(char));
}

/* Create the field */
sprintf(footer, "attribute \"dep\" string \"positions\"\n" \
  "object \"regular positions regular connections\" class field\n" \
  "component \"positions\" value 1\n" \
  "component \"connections\" value 2\n" \
  "component \"data\" value 3\n");
gzwrite(outfile, footer, strlen(footer)*sizeof(char));

gzclose(outfile);
#else

Vnm_print(0, "WARNING\n");
Vnm_print(0, "Vgrid_readGZ:  gzip read/write support is disabled in this build\n");
Vnm_print(0, "Vgrid_readGZ:  configure and compile without the --disable-zlib flag.\n");
Vnm_print(0, "WARNING\n");
#endif
}

/* ///////////////////////////////////////////////////////////////////////////
// Routine:  Vgrid_writeDX
//
// Author:   Nathan Baker
VPUBLIC void Vgrid_writeDX(Vgrid *thee, const char *iodev, const char *iofmt,
  const char *thost, const char *fname, char *title, double *pvec) {

    double xmin, ymin, zmin, hx, hy, hzed;
    int nx, ny, nz;
    int icol, i, j, k, u, usepart, nxPART, nyPART, nzPART, gotit;
    double x, y, z, xminPART, yminPART, zminPART;
    Vio *sock;
    char precFormat[VMAX_BUFSIZE];

    if (thee == VNULL) {
        Vnm_print(2, "Vgrid_writeDX:  Error -- got VNULL thee!\n");
        VASSERT(0);
    }
    if (!(thee->ctordata || thee->readdata)) {
        Vnm_print(2, "Vgrid_writeDX:  Error -- no data available!\n");
        VASSERT(0);
    }

    hx = thee->hx;
    hy = thee->hy;
    hzed = thee->hzed;
    nx = thee->nx;
    ny = thee->ny;
    nz = thee->nz;
    xmin = thee->xmin;
    ymin = thee->ymin;
    zmin = thee->zmin;

    if (pvec == VNULL) usepart = 0;
    else usepart = 1;

    /* Set up the virtual socket */
    Vnm_print(0, "Vgrid_writeDX:  Opening virtual socket...\n");
```

```
    sock = Vio_ctor(iodev,iofmt,thost,fname,"w");
    if (sock == VNULL) {
        Vnm_print(2, "Vgrid_writeDX:  Problem opening virtual socket %s\n",
          fname);
        return;
    }
    if (Vio_connect(sock, 0) < 0) {
        Vnm_print(2, "Vgrid_writeDX: Problem connecting virtual socket %s\n",
          fname);
        return;
    }

    Vio_setWhiteChars(sock, MCwhiteChars);
    Vio_setCommChars(sock, MCcommChars);

    Vnm_print(0, "Vgrid_writeDX:  Writing to virtual socket...\n");

    if (usepart) {
        /* Get the lower corner and number of grid points for the local
         * partition */
        xminPART = VLARGE;
        yminPART = VLARGE;
        zminPART = VLARGE;
        nxPART = 0;
        nyPART = 0;
        nzPART = 0;
        /* First, search for the lower corner */
        for (k=0; k<nz; k++) {
            z = k*hzed + zmin;
            for (j=0; j<ny; j++) {
                y = j*hy + ymin;
                for (i=0; i<nx; i++) {
                    x = i*hx + xmin;
                    if (pvec[IJK(i,j,k)] > 0.0) {
                        if (x < xminPART) xminPART = x;
                        if (y < yminPART) yminPART = y;
                        if (z < zminPART) zminPART = z;
                    }
                }
            }
        }
        /* Now search for the number of grid points in the z direction */
        for (k=0; k<nz; k++) {
            gotit = 0;
            for (j=0; j<ny; j++) {
                for (i=0; i<nx; i++) {
                    if (pvec[IJK(i,j,k)] > 0.0) {
                        gotit = 1;
                        break;
                    }
                }
                if (gotit) break;
            }
            if (gotit) nzPART++;
        }
        /* Now search for the number of grid points in the y direction */
        for (j=0; j<ny; j++) {
            gotit = 0;
            for (k=0; k<nz; k++) {
                for (i=0; i<nx; i++) {
                    if (pvec[IJK(i,j,k)] > 0.0) {
                        gotit = 1;
                        break;
                    }
                }
                if (gotit) break;
            }
            if (gotit) nyPART++;
        }
        /* Now search for the number of grid points in the x direction */
        for (i=0; i<nx; i++) {
            gotit = 0;
            for (k=0; k<nz; k++) {
                for (j=0; j<ny; j++) {
                    if (pvec[IJK(i,j,k)] > 0.0) {
                        gotit = 1;
                        break;
                    }
                }
                if (gotit) break;
            }
```

```
            if (gotit) nxPART++;
        }

        if ((nxPART != nx) || (nyPART != ny) || (nzPART != nz)) {
            Vnm_print(0, "Vgrid_writeDX:  printing only subset of domain\n");
        }


        /* Write off the title (if we're not XDR) */
        if (Vstring_strcasecmp(iofmt, "XDR") == 0) {
            Vnm_print(0, "Vgrid_writeDX:  Skipping comments for XDR format.\n")
;
        } else {
            Vnm_print(0, "Vgrid_writeDX:  Writing comments for %s format.\n",
              iofmt);
            Vio_printf(sock, "# Data from %s\n", PACKAGE_STRING);
            Vio_printf(sock, "# \n");
            Vio_printf(sock, "# %s\n", title);
            Vio_printf(sock, "# \n");
        }

        /* Write off the DX regular positions */
        Vio_printf(sock, "object 1 class gridpositions counts %d %d %d\n",
          nxPART, nyPART, nzPART);

        sprintf(precFormat, Vprecision, xminPART, yminPART, zminPART);
        Vio_printf(sock, "origin %s\n", precFormat);
        sprintf(precFormat, Vprecision, hx, 0.0, 0.0);
        Vio_printf(sock, "delta %s\n", precFormat);
        sprintf(precFormat, Vprecision, 0.0, hy, 0.0);
        Vio_printf(sock, "delta %s\n", precFormat);
        sprintf(precFormat, Vprecision, 0.0, 0.0, hzed);
        Vio_printf(sock, "delta %s\n", precFormat);

        /* Write off the DX regular connections */
        Vio_printf(sock, "object 2 class gridconnections counts %d %d %d\n",
          nxPART, nyPART, nzPART);

        /* Write off the DX data */
        Vio_printf(sock, "object 3 class array type double rank 0 items %d \
data follows\n", (nxPART*nyPART*nzPART));
        icol = 0;
        for (i=0; i<nx; i++) {
            for (j=0; j<ny; j++) {
                for (k=0; k<nz; k++) {
                    u = k*(nx)*(ny)+j*(nx)+i;
                    if (pvec[u] > 0.0) {
                        Vio_printf(sock, "%12.6e ", thee->data[u]);
                        icol++;
                        if (icol == 3) {
                            icol = 0;
                            Vio_printf(sock, "\n");
                        }
                    }
                }
            }
        }

        if (icol != 0) Vio_printf(sock, "\n");

        /* Create the field */
        Vio_printf(sock, "attribute \"dep\" string \"positions\"\n");
        Vio_printf(sock, "object \"regular positions regular connections\" \
class field\n");
        Vio_printf(sock, "component \"positions\" value 1\n");
        Vio_printf(sock, "component \"connections\" value 2\n");
        Vio_printf(sock, "component \"data\" value 3\n");

    } else {
        /* Write off the title (if we're not XDR) */
        if (Vstring_strcasecmp(iofmt, "XDR") == 0) {
            Vnm_print(0, "Vgrid_writeDX:  Skipping comments for XDR format.\n")
;
        } else {
            Vnm_print(0, "Vgrid_writeDX:  Writing comments for %s format.\n",
              iofmt);
            Vio_printf(sock, "# Data from %s\n", PACKAGE_STRING);
            Vio_printf(sock, "# \n");
            Vio_printf(sock, "# %s\n", title);
            Vio_printf(sock, "# \n");
        }
```

```c


        /* Write off the DX regular positions */
        Vio_printf(sock, "object 1 class gridpositions counts %d %d %d\n",
          nx, ny, nz);

        sprintf(precFormat, Vprecision, xmin, ymin, zmin);
        Vio_printf(sock, "origin %s\n", precFormat);
        sprintf(precFormat, Vprecision, hx, 0.0, 0.0);
        Vio_printf(sock, "delta %s\n", precFormat);
        sprintf(precFormat, Vprecision, 0.0, hy, 0.0);
        Vio_printf(sock, "delta %s\n", precFormat);
        sprintf(precFormat, Vprecision, 0.0, 0.0, hzed);
        Vio_printf(sock, "delta %s\n", precFormat);

        /* Write off the DX regular connections */
        Vio_printf(sock, "object 2 class gridconnections counts %d %d %d\n",
          nx, ny, nz);

        /* Write off the DX data */
        Vio_printf(sock, "object 3 class array type double rank 0 items %d \
data follows\n", (nx*ny*nz));
        icol = 0;
        for (i=0; i<nx; i++) {
            for (j=0; j<ny; j++) {
                for (k=0; k<nz; k++) {
                    u = k*(nx)*(ny)+j*(nx)+i;
                    Vio_printf(sock, "%12.6e ", thee->data[u]);
                    icol++;
                    if (icol == 3) {
                        icol = 0;
                        Vio_printf(sock, "\n");
                    }
                }
            }
        }
        if (icol != 0) Vio_printf(sock, "\n");

        /* Create the field */
        Vio_printf(sock, "attribute \"dep\" string \"positions\"\n");
        Vio_printf(sock, "object \"regular positions regular connections\" \
class field\n");
        Vio_printf(sock, "component \"positions\" value 1\n");
        Vio_printf(sock, "component \"connections\" value 2\n");
        Vio_printf(sock, "component \"data\" value 3\n");
    }

    /* Close off the socket */
    Vio_connectFree(sock);
    Vio_dtor(&sock);
}

/* ////////////////////////////////////////////////////////////////////////
// Routine:  Vgrid_writeUHBD
// Author:   Nathan Baker
VPUBLIC void Vgrid_writeUHBD(Vgrid *thee, const char *iodev, const char *iofmt,
  const char *thost, const char *fname, char *title, double *pvec) {

    int icol, i, j, k, u, nx, ny, nz, gotit;
    double xmin, ymin, zmin, hzed, hy, hx;
    Vio *sock;

    if (thee == VNULL) {
        Vnm_print(2, "Vgrid_writeUHBD:  Error -- got VNULL thee!\n");
        VASSERT(0);
    }
    if (!(thee->ctordata || thee->readdata)) {
        Vnm_print(2, "Vgrid_writeUHBD:  Error -- no data available!\n");
        VASSERT(0);
    }

    if ((thee->hx!=thee->hy) || (thee->hy!=thee->hzed)
      || (thee->hx!=thee->hzed)) {
        Vnm_print(2, "Vgrid_writeUHBD: can't write UHBD mesh with non-uniform \
spacing\n");
        return;
    }

    /* Set up the virtual socket */
    sock = Vio_ctor(iodev,iofmt,thost,fname,"w");
    if (sock == VNULL) {
```

```
        Vnm_print(2, "Vgrid_writeUHBD: Problem opening virtual socket %s\n",
          fname);
        return;
    }
    if (Vio_connect(sock, 0) < 0) {
        Vnm_print(2, "Vgrid_writeUHBD: Problem connecting virtual socket %s\n",
          fname);
        return;
    }

    /* Get the lower corner and number of grid points for the local
     * partition */
    hx = thee->hx;
    hy = thee->hy;
    hzed = thee->hzed;
    nx = thee->nx;
    ny = thee->ny;
    nz = thee->nz;
    xmin = thee->xmin;
    ymin = thee->ymin;
    zmin = thee->zmin;

    /* Let interested folks know that partition information is ignored */
    if (pvec != VNULL) {
        gotit = 0;
        for (i=0; i<(nx*ny*nz); i++) {
            if (pvec[i] == 0) {
                gotit = 1;
                break;
            }
        }
        if (gotit) {
            Vnm_print(2, "Vgrid_writeUHBD:  IGNORING PARTITION INFORMATION!\n")
;
            Vnm_print(2, "Vgrid_writeUHBD:  This means I/O from parallel runs \
will have significant overlap.\n");
        }
    }

    /* Write out the header */
    Vio_printf(sock, "%72s\n", title);
    Vio_printf(sock, "%12.5e%12.5e%7d%7d%7d%7d%7d\n", 1.0, 0.0, -1, 0,
      nz, 1, nz);
    Vio_printf(sock, "%7d%7d%7d%12.5e%12.5e%12.5e%12.5e\n", nx, ny, nz,
      hx, (xmin-hx), (ymin-hx), (zmin-hx));
    Vio_printf(sock, "%12.5e%12.5e%12.5e%12.5e\n", 0.0, 0.0, 0.0, 0.0);
    Vio_printf(sock, "%12.5e%12.5e%7d%7d", 0.0, 0.0, 0, 0);

    /* Write out the entries */
    icol = 0;
    for (k=0; k<nz; k++) {
        Vio_printf(sock, "\n%7d%7d%7d\n", k+1, thee->nx, thee->ny);
        icol = 0;
        for (j=0; j<ny; j++) {
            for (i=0; i<nx; i++) {
                u = k*(nx)*(ny)+j*(nx)+i;
                icol++;
                Vio_printf(sock, " %12.5e", thee->data[u]);
                if (icol == 6) {
                    icol = 0;
                    Vio_printf(sock, "\n");
                }
            }
        }
    }
    if (icol != 0) Vio_printf(sock, "\n");

    /* Close off the socket */
    Vio_connectFree(sock);
    Vio_dtor(&sock);
}

VPUBLIC double Vgrid_integrate(Vgrid *thee) {

    int i, j, k, nx, ny, nz;
    double sum, w;

    if (thee == VNULL) {
        Vnm_print(2, "Vgrid_integrate:  Got VNULL thee!\n");
        VASSERT(0);
    }
```

```c

    nx = thee->nx;
    ny = thee->ny;
    nz = thee->nz;

    sum = 0.0;

    for (k=0; k<nz; k++) {
  w = 1.0;
  if ((k==0) || (k==(nz-1))) w = w * 0.5;
        for (j=0; j<ny; j++) {
   w = 1.0;
   if ((j==0) || (j==(ny-1))) w = w * 0.5;
            for (i=0; i<nx; i++) {
    w = 1.0;
    if ((i==0) || (i==(nx-1))) w = w * 0.5;
                sum = sum + w*(thee->data[IJK(i,j,k)]);
            }
        }
    }

    sum = sum*(thee->hx)*(thee->hy)*(thee->hzed);

    return sum;

}


VPUBLIC double Vgrid_normL1(Vgrid *thee) {

    int i, j, k, nx, ny, nz;
    double sum;

    if (thee == VNULL) {
        Vnm_print(2, "Vgrid_normL1:  Got VNULL thee!\n");
        VASSERT(0);
    }

    nx = thee->nx;
    ny = thee->ny;
    nz = thee->nz;

    sum = 0.0;
    for (k=0; k<nz; k++) {
        for (j=0; j<ny; j++) {
            for (i=0; i<nx; i++) {
                sum = sum + VABS(thee->data[IJK(i,j,k)]);
            }
        }
    }

    sum = sum*(thee->hx)*(thee->hy)*(thee->hzed);

    return sum;

}

VPUBLIC double Vgrid_normL2(Vgrid *thee) {

    int i, j, k, nx, ny, nz;
    double sum;

    if (thee == VNULL) {
        Vnm_print(2, "Vgrid_normL2:  Got VNULL thee!\n");
        VASSERT(0);
    }

    nx = thee->nx;
    ny = thee->ny;
    nz = thee->nz;

    sum = 0.0;
    for (k=0; k<nz; k++) {
        for (j=0; j<ny; j++) {
            for (i=0; i<nx; i++) {
                sum = sum + VSQR(thee->data[IJK(i,j,k)]);
            }
        }
    }

    sum = sum*(thee->hx)*(thee->hy)*(thee->hzed);
```

```
    return VSQRT(sum);

}

VPUBLIC double Vgrid_seminormH1(Vgrid *thee) {

    int i, j, k, d, nx, ny, nz;
    double pt[3], grad[3], sum, hx, hy, hzed, xmin, ymin, zmin;

    if (thee == VNULL) {
        Vnm_print(2, "Vgrid_seminormH1:  Got VNULL thee!\n");
        VASSERT(0);
    }

    nx = thee->nx;
    ny = thee->ny;
    nz = thee->nz;
    hx = thee->hx;
    hy = thee->hy;
    hzed = thee->hzed;
    xmin = thee->xmin;
    ymin = thee->ymin;
    zmin = thee->zmin;

    sum = 0.0;
    for (k=0; k<nz; k++) {
        pt[2] = k*hzed + zmin;
        for (j=0; j<ny; j++) {
            pt[1] = j*hy + ymin;
            for (i=0; i<nx; i++) {
                pt[0] = i*hx + xmin;
                VASSERT(Vgrid_gradient(thee, pt, grad));
                for (d=0; d<3; d++) sum = sum + VSQR(grad[d]);
            }
        }
    }

    sum = sum*(hx)*(hy)*(hzed);

    if (VABS(sum) < VSMALL) sum = 0.0;
    else sum = VSQRT(sum);

    return sum;

}

VPUBLIC double Vgrid_normH1(Vgrid *thee) {

    double sum = 0.0;

    if (thee == VNULL) {
        Vnm_print(2, "Vgrid_normH1:  Got VNULL thee!\n");
        VASSERT(0);
    }

    sum = VSQR(Vgrid_seminormH1(thee)) + VSQR(Vgrid_normL2(thee));

    return VSQRT(sum);

}

VPUBLIC double Vgrid_normLinf(Vgrid *thee) {

    int i, j, k, nx, ny, nz, gotval;
    double sum, val;

    if (thee == VNULL) {
        Vnm_print(2, "Vgrid_normLinf:  Got VNULL thee!\n");
        VASSERT(0);
    }

    nx = thee->nx;
    ny = thee->ny;
    nz = thee->nz;

    sum = 0.0;
    gotval = 0;
    for (k=0; k<nz; k++) {
```

```
        for (j=0; j<ny; j++) {
            for (i=0; i<nx; i++) {
                val = VABS(thee->data[IJK(i,j,k)]);
                if (!gotval) {
                    gotval = 1;
                    sum = val;
                }
                if (val > sum) sum = val;
            }
        }
    }

    return sum;

}
```

9.93 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.

9.93.1 Detailed Description

Class Vopot methods.

Author

Nathan Baker

Version

Id:

vopot.c 1774 2012-07-19 18:46:14Z sobolevnrm

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.

9.94 vopot.c

```
00001
00057 #include "apbscfg.h"
00058 #include "apbs/vopot.h"
00059 #include "apbs/pbeparm.h"
00060
00061 VEMBED(rcsid="$Id: vopot.c 1774 2012-07-19 18:46:14Z sobolevnrm $")
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
```

```
#define IJK(i,j,k)  (((k)*(nx)*(ny))+((j)*(nx))+(i))
VPUBLIC int Vopot_pot(Vopot *thee, double pt[3], double *value) {

    Vatom *atom;
    int i, iatom;
    double u, T, charge, eps_w, xkappa, dist, size, val, *position;
    Valist *alist;

    VASSERT(thee != VNULL);

    eps_w = Vpbe_getSolventDiel(thee->pbe);
    xkappa = (1.0e10)*Vpbe_getXkappa(thee->pbe);
    T = Vpbe_getTemperature(thee->pbe);
    alist = Vpbe_getValist(thee->pbe);

    u = 0;

    /* See if we're on the mesh */
    if (Vmgrid_value(thee->mgrid, pt, &u)) {

        *value = u;

    } else {

        switch (thee->bcfl) {

            case BCFL_ZERO:
                u = 0;
                break;

            case BCFL_SDH:
                size = (1.0e-10)*Vpbe_getSoluteRadius(thee->pbe);
                position = Vpbe_getSoluteCenter(thee->pbe);
                charge = Vunit_ec*Vpbe_getSoluteCharge(thee->pbe);
                dist = 0;
                for (i=0; i<3; i++)
                  dist += VSQR(position[i] - pt[i]);
                dist = (1.0e-10)*VSQRT(dist);
                val = (charge)/(4*VPI*Vunit_eps0*eps_w*dist);
                if (xkappa != 0.0)
                  val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
                val = val*Vunit_ec/(Vunit_kb*T);
                u = val;
                break;

            case BCFL_MDH:
                u = 0;
                for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
                    atom = Valist_getAtom(alist, iatom);
                    position = Vatom_getPosition(atom);
                    charge = Vunit_ec*Vatom_getCharge(atom);
                    size = (1e-10)*Vatom_getRadius(atom);
                    dist = 0;
                    for (i=0; i<3; i++)
                      dist += VSQR(position[i] - pt[i]);
                    dist = (1.0e-10)*VSQRT(dist);
                    val = (charge)/(4*VPI*Vunit_eps0*eps_w*dist);
                    if (xkappa != 0.0)
                      val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
                    val = val*Vunit_ec/(Vunit_kb*T);
                    u = u + val;
                }
                break;

            case BCFL_UNUSED:
                Vnm_print(2, "Vopot_pot:  Invalid bcfl flag (%d)!\n",
                  thee->bcfl);
                return 0;

            case BCFL_FOCUS:
                Vnm_print(2, "Vopot_pot:  Invalid bcfl flag (%d)!\n",
                  thee->bcfl);
                return 0;

            default:
                Vnm_print(2, "Vopot_pot:  Bogus thee->bcfl flag (%d)!\n",
                  thee->bcfl);
```

```
                return 0;
                break;
        }

        *value = u;

    }

    return 1;

}

/* ////////////////////////////////////////////////////////////////////////
// Routine:  Vopot_curvature
//
//   Notes:  cflag=0 ==> Reduced Maximal Curvature
//           cflag=1 ==> Mean Curvature (Laplace)
//           cflag=2 ==> Gauss Curvature
//           cflag=3 ==> True Maximal Curvature
//   If we are off the grid, we can still evaluate the Laplacian; assuming, we
//   are away from the molecular surface, it is simply equal to the DH factor.
//
// Authors:  Nathan Baker
VPUBLIC int Vopot_curvature(Vopot *thee, double pt[3], int cflag,
  double *value) {

    Vatom *atom;
    int i, iatom;
    double u, T, charge, eps_w, xkappa, dist, size, val, *position, zkappa2;
    Valist *alist;

    VASSERT(thee != VNULL);

    eps_w = Vpbe_getSolventDiel(thee->pbe);
    xkappa = (1.0e10)*Vpbe_getXkappa(thee->pbe);
    zkappa2 = Vpbe_getZkappa2(thee->pbe);
    T = Vpbe_getTemperature(thee->pbe);
    alist = Vpbe_getValist(thee->pbe);

    u = 0;

    if (Vmgrid_curvature(thee->mgrid, pt, cflag, value)) return 1;
    else if (cflag != 1) {
        Vnm_print(2, "Vopot_curvature:  Off mesh!\n");
        return 1;
    } else {

        switch (thee->bcfl) {

            case BCFL_ZERO:
                u = 0;
                break;

            case BCFL_SDH:
                size = (1.0e-10)*Vpbe_getSoluteRadius(thee->pbe);
                position = Vpbe_getSoluteCenter(thee->pbe);
                charge = Vunit_ec*Vpbe_getSoluteCharge(thee->pbe);
                dist = 0;
                for (i=0; i<3; i++)
                  dist += VSQR(position[i] - pt[i]);
                dist = (1.0e-10)*VSQRT(dist);
                if (xkappa != 0.0)
                  u = zkappa2*(exp(-xkappa*(dist-size))/(1+xkappa*size));
                break;

            case BCFL_MDH:
                u = 0;
                for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
                    atom = Valist_getAtom(alist, iatom);
                    position = Vatom_getPosition(atom);
                    charge = Vunit_ec*Vatom_getCharge(atom);
                    size = (1e-10)*Vatom_getRadius(atom);
                    dist = 0;
                    for (i=0; i<3; i++)
                      dist += VSQR(position[i] - pt[i]);
                    dist = (1.0e-10)*VSQRT(dist);
                    if (xkappa != 0.0)
                      val = zkappa2*(exp(-xkappa*(dist-size))/(1+xkappa*size));
                    u = u + val;
                }
                break;
```

```

            case BCFL_UNUSED:
                Vnm_print(2, "Vopot_pot:  Invlid bcfl (%d)!\n", thee->bcfl);
                return 0;

            case BCFL_FOCUS:
                Vnm_print(2, "Vopot_pot:  Invlid bcfl (%d)!\n", thee->bcfl);
                return 0;

            default:
                Vnm_print(2, "Vopot_pot:  Bogus thee->bcfl flag (%d)!\n",
                  thee->bcfl);
                return 0;
                break;
        }

        *value = u;
    }

    return 1;

}

/* ///////////////////////////////////////////////////////////////////////////
// Routine:  Vopot_gradient
//
// Authors:  Nathan Baker
VPUBLIC int Vopot_gradient(Vopot *thee, double pt[3], double grad[3]) {

    Vatom *atom;
    int iatom;
    double T, charge, eps_w, xkappa, size, val, *position;
    double dx, dy, dz, dist;
    Valist *alist;

    VASSERT(thee != VNULL);

    eps_w = Vpbe_getSolventDiel(thee->pbe);
    xkappa = (1.0e10)*Vpbe_getXkappa(thee->pbe);
    T = Vpbe_getTemperature(thee->pbe);
    alist = Vpbe_getValist(thee->pbe);


    if (!Vmgrid_gradient(thee->mgrid, pt, grad)) {

        switch (thee->bcfl) {

            case BCFL_ZERO:
                grad[0] = 0.0;
                grad[1] = 0.0;
                grad[2] = 0.0;
                break;

            case BCFL_SDH:
                grad[0] = 0.0;
                grad[1] = 0.0;
                grad[2] = 0.0;
                size = (1.0e-10)*Vpbe_getSoluteRadius(thee->pbe);
                position = Vpbe_getSoluteCenter(thee->pbe);
                charge = Vunit_ec*Vpbe_getSoluteCharge(thee->pbe);
                dx = position[0] - pt[0];
                dy = position[1] - pt[1];
                dz = position[2] - pt[2];
                dist = VSQR(dx) + VSQR(dy) + VSQR(dz);
                dist = (1.0e-10)*VSQRT(dist);
                val = (charge)/(4*VPI*Vunit_eps0*eps_w);
                if (xkappa != 0.0)
                  val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
                val = val*Vunit_ec/(Vunit_kb*T);
                grad[0] = val*dx/dist*(-1.0/dist/dist + xkappa/dist);
                grad[1] = val*dy/dist*(-1.0/dist/dist + xkappa/dist);
                grad[2] = val*dz/dist*(-1.0/dist/dist + xkappa/dist);
                break;

            case BCFL_MDH:
                grad[0] = 0.0;
                grad[1] = 0.0;
                grad[2] = 0.0;
                for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
                    atom = Valist_getAtom(alist, iatom);
                    position = Vatom_getPosition(atom);
```

```
                    charge = Vunit_ec*Vatom_getCharge(atom);
                    size = (1e-10)*Vatom_getRadius(atom);
                    dx = position[0] - pt[0];
                    dy = position[1] - pt[1];
                    dz = position[2] - pt[2];
                    dist = VSQR(dx) + VSQR(dy) + VSQR(dz);
                    dist = (1.0e-10)*VSQRT(dist);
                    val = (charge)/(4*VPI*Vunit_eps0*eps_w);
                    if (xkappa != 0.0)
                      val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
                    val = val*Vunit_ec/(Vunit_kb*T);
                    grad[0] += (val*dx/dist*(-1.0/dist/dist + xkappa/dist));
                    grad[1] += (val*dy/dist*(-1.0/dist/dist + xkappa/dist));
                    grad[2] += (val*dz/dist*(-1.0/dist/dist + xkappa/dist));
                }
                break;

            case BCFL_UNUSED:
                Vnm_print(2, "Vopot:  Invalid bcfl (%d)!\n", thee->bcfl);
                return 0;

            case BCFL_FOCUS:
                Vnm_print(2, "Vopot:  Invalid bcfl (%d)!\n", thee->bcfl);
                return 0;

            default:
                Vnm_print(2, "Vopot_pot:  Bogus thee->bcfl flag (%d)!\n",
                  thee->bcfl);
                return 0;
                break;
        }

        return 1;
    }

    return 1;

}
```

9.95 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, PBEparm *pbeparm)

  Fill the specified array with accessibility values.
- VPRIVATE double Vpmg_polarizEnergy (Vpmg *thee, int extFlag)

  Determines energy from polarizeable charge and interaction with fixed charges according to Rocchia et al.
- VPUBLIC double Vpmg_energy (Vpmg *thee, int extFlag)

  Get the total electrostatic energy.
- VPUBLIC double Vpmg_dielEnergy (Vpmg *thee, int extFlag)

  Get the "polarization" contribution to the electrostatic energy.
- VPUBLIC double Vpmg_dielGradNorm (Vpmg *thee)

  Get the integral of the gradient of the dielectric function.
- VPUBLIC double Vpmg_qmEnergy (Vpmg *thee, int extFlag)

  Get the "mobile charge" contribution to the electrostatic energy.
- VPRIVATE double **Vpmg_qmEnergyNONLIN** (Vpmg *thee, int extFlag)
- VPUBLIC double Vpmg_qmEnergySMPBE (Vpmg *thee, int extFlag)

  Vpmg_qmEnergy for SMPBE.
- VPUBLIC double Vpmg_qfEnergy (Vpmg *thee, int extFlag)

  Get the "fixed charge" contribution to the electrostatic energy.
- VPRIVATE double Vpmg_qfEnergyPoint (Vpmg *thee, int extFlag)

  Calculates charge-potential energy using summation over delta function positions (i.e. something like an Linf norm)
- VPUBLIC double Vpmg_qfAtomEnergy (Vpmg *thee, Vatom *atom)

  Get the per-atom "fixed charge" contribution to the electrostatic energy.
- VPRIVATE double Vpmg_qfEnergyVolume (Vpmg *thee, int extFlag)

  Calculates charge-potential energy as integral over a volume.
- VPRIVATE void Vpmg_splineSelect (int 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.
- VPRIVATE void bcfl1 (double size, double *apos, double charge, double xkappa, double pre1, double *gxcf, double *gycf, double *gzcf, double *xf, double *yf, double *zf, int nx, int ny, int nz)

  Increment all boundary points by pre1(charge/d)*(exp(-xkappa*(d-size))/(1+xkappa*size) to add the effect of the Debye-Huckel potential due to a single charge.*
- VPRIVATE void **bcCalcOrig** (Vpmg *thee)
- VPRIVATE int **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-Huckel potential tensor for a spherical ion with a central charge, dipole and quadrupole. See the code for an in depth description.
- VPRIVATE void **bcfl_sdh** (Vpmg ∗thee)
- VPRIVATE void **bcfl_mdh** (Vpmg ∗thee)
- VPRIVATE void **bcfl_mem** (double zmem, double L, double eps_m, double eps_w, double V, double xkappa, double ∗gxcf, double ∗gycf, double ∗gzcf, double ∗xf, double ∗yf, double ∗zf, int nx, int ny, int nz)
- VPRIVATE void **bcfl_map** (Vpmg ∗thee)
- VPRIVATE void bcCalc (Vpmg ∗thee)

  Fill boundary condition arrays.
- VPRIVATE void 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)
- VPRIVATE void markSphere (double rtot, double ∗tpos, int nx, int ny, int nz, double hx, double hy, double hz, double xmin, double ymin, double zmin, double ∗array, double markVal)

  Mark the grid points inside a sphere with a particular value. This marks by resetting the the grid points inside the sphere to the specified value.
- VPRIVATE void zlapSolve (Vpmg ∗thee, double ∗∗solution, double ∗∗source, double ∗∗work1)

  Calculate the solution to Poisson's equation with a simple Laplacian operator and zero-valued Dirichlet boundary conditions. Store the solution in thee->u.
- VPUBLIC int Vpmg_solveLaplace (Vpmg ∗thee)

  Solve Poisson's equation with a homogeneous Laplacian operator using the solvent dielectric constant. This solution is performed by a sine wave decomposition.
- VPRIVATE double VFCHI4 (int i, double f)

  Return 2.5 plus difference of i - f.
- VPRIVATE double bspline4 (double x)

  Evaluate a 5th Order B-Spline (4th order polynomial)
- VPUBLIC double 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 fillcoCoefSpline3 (Vpmg ∗thee)

  Fill operator coefficient arrays from a 5th order polynomial based surface calculation.
- VPRIVATE void bcolcomp (int ∗iparm, double ∗rparm, int ∗iwork, double ∗rwork, double ∗values, int ∗rowind, int ∗colptr, int ∗flag)

  Build a column-compressed matrix in Harwell-Boeing format.
- VPRIVATE void bcolcomp2 (int ∗iparm, double ∗rparm, int ∗nx, int ∗ny, int ∗nz, int ∗iz, int ∗ipc, double ∗rpc, double ∗ac, double ∗cc, double ∗values, int ∗rowind, int ∗colptr, int ∗flag)

  Build a column-compressed matrix in Harwell-Boeing format.
- VPRIVATE void bcolcomp3 (int ∗nx, int ∗ny, int ∗nz, int ∗ipc, double ∗rpc, double ∗ac, double ∗cc, double ∗values, int ∗rowind, int ∗colptr, int ∗flag)

Build a column-compressed matrix in Harwell-Boeing format.

- VPRIVATE void bcolcomp4 (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *oC, double *cc, double *oE, double *oN, double *uC, double *values, int *rowind, int *colptr, int *flag)

  Build a column-compressed matrix in Harwell-Boeing format.

- VPRIVATE void pcolcomp (int *nrow, int *ncol, int *nnzero, double *values, int *rowind, int *colptr, char *path, char *title, char *mxtype)

  Print a column-compressed matrix in Harwell-Boeing format.

9.95.1 Detailed Description

Class Vpmg methods.

Author

Nathan Baker

Version

Id:

vpmg.c 1793 2012-07-28 00:54:24Z sobolevnrm

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.

9.95.2 Function Documentation

9.95.2.1 VPRIVATE void bcCalc (Vpmg * *thee*)

Fill boundary condition arrays.

Author

Nathan Baker

Definition at line 4373 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.2 VPRIVATE void bcfl1 (double *size*, double * *apos*, double *charge*, double *xkappa*, double *pre1*, double * *gxcf*, double * *gycf*, double * *gzcf*, double * *xf*, double * *yf*, double * *zf*, int *nx*, int *ny*, int *nz*)

Increment all boundary points by pre1∗(charge/d)∗(exp(-xkappa∗(d-size))/(1+xkappa∗size) to add the effect of the Debye-Huckel potential due to a single charge.

Author

Nathan Baker

Parameters

apos	Size of the ion
charge	Position of the ion
xkappa	Charge of the ion
pre1	Exponential screening factor
gxcf	Unit- and dielectric-dependent prefactor
gycf	Set to x-boundary values
gzcf	Set to y-boundary values
xf	Set to z-boundary values
yf	Boundary point x-coordinates
zf	Boundary point y-coordinates
nx	Boundary point z-coordinates
ny	Number of grid points in x-direction
nz	Number of grid points in y-direction Number of grid points in y-direction

Definition at line 2570 of file vpmg.c.

9.95.2.3 VPRIVATE double bspline2 (double *x*)

Evaluate a cubic B-spline.

Author

Nathan Baker

Returns

Cubic B-spline value

Parameters

x	Position

Definition at line 5487 of file vpmg.c.

Here is the caller graph for this function:

9.95.2.4 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 7127 of file vpmg.c.

Here is the caller graph for this function:

9.95.2.5 VPUBLIC double d2bspline4 (double *x*)

Evaluate the 2nd derivative of a 5th Order B-Spline.

Author

: Michael Schnieders

Returns

2nd derivative of a 5th Order B-Spline

Parameters

x	Position

Definition at line 7193 of file vpmg.c.

Here is the caller graph for this function:

9.95.2.6 VPUBLIC double d3bspline4 (double *x*)

Evaluate the 3rd derivative of a 5th Order B-Spline.

Author

: Michael Schnieders

Returns

3rd derivative of a 5th Order B-Spline

Parameters

x	Position

Definition at line 7220 of file vpmg.c.

9.95.2.7 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 5503 of file vpmg.c.

Here is the caller graph for this function:

9.95.2.8 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 7161 of file vpmg.c.

Here is the caller graph for this function:

9.95.2.9 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 5278 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.10 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 5334 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.11 VPRIVATE void fillcoChargeSpline1 (Vpmg * *thee*)

Fill source term charge array from linear interpolation.

Author

Nathan Baker

Definition at line 5382 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.12 VPRIVATE void fillcoChargeSpline2 (Vpmg * *thee*)

Fill source term charge array from cubic spline interpolation.

Author

Nathan Baker

Definition at line 5519 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.13 VPRIVATE void fillcoCoef (Vpmg * *thee*)

Top-level driver to fill all operator coefficient arrays.

Author

Nathan Baker

Definition at line 5238 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.14 VPRIVATE void fillcoCoefMap (Vpmg * *thee*)

Fill operator coefficient arrays from pre-calculated maps.

Author

Nathan Baker

Definition at line 4480 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.15 VPRIVATE void fillcoCoefMol (Vpmg * *thee*)

Fill operator coefficient arrays from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4603 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.16 VPRIVATE void fillcoCoefMolDiel (Vpmg * *thee*)

Fill differential operator coefficient arrays from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4717 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.17 VPRIVATE void fillcoCoefMolDielNoSmooth (Vpmg * *thee*)

Fill differential operator coefficient arrays from a molecular surface calculation without smoothing.

Author

Nathan Baker

Definition at line 4728 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.18 VPRIVATE void fillcoCoefMolDielSmooth (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 4882 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.19 VPRIVATE void fillcoCoefMolIon (Vpmg * *thee*)

Fill ion (nonlinear) operator coefficient array from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4619 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.20 VPRIVATE void fillcoCoefSpline (Vpmg * *thee*)

Fill operator coefficient arrays from a spline-based surface calculation.

Author

Nathan Baker

Definition at line 5013 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.21 VPRIVATE void fillcoCoefSpline3 (Vpmg * *thee*)

Fill operator coefficient arrays from a 5th order polynomial based surface calculation.

Author

Michael Schnieders

Definition at line 10421 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.22 VPRIVATE void fillcoCoefSpline4 (Vpmg * *thee*)

Fill operator coefficient arrays from a 7th order polynomial based surface calculation.

Author

Michael Schnieders

Definition at line 9930 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.23 VPUBLIC void fillcoPermanentMultipole (Vpmg * *thee*)

Fill source term charge array for the use of permanent multipoles.

Author

Michael Schnieders

Definition at line 7231 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.24 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

tpos	Sphere radius
nx	Sphere position
ny	Number of grid points
nz	Number of grid points
hx	Number of grid points
hy	Grid spacing
hz	Grid spacing
xmin	Grid spacing
ymin	Grid lower corner
zmin	Grid lower corner
array	Grid lower corner
markVal	Grid values Value to mark with

Definition at line 6840 of file vpmg.c.

Here is the caller graph for this function:

9.95.2.25 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

kappa	Distance to the boundary
eps_p	Exponential screening factor
eps_w	Solute dielectric
rad	Solvent dielectric
tsr	Radius of the sphere Contraction-independent portion of each tensor

Definition at line 3483 of file vpmg.c.

9.95.2.26 VPRIVATE void qfForceSpline1 (Vpmg * *thee*, double * *force*, int *atomID*)

Charge-field force due to a linear spline charge function.

Author

Nathan Baker

Parameters

atomID	Set to force Valist atom ID

Definition at line 6302 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.27 VPRIVATE void qfForceSpline2 (Vpmg * *thee*, double * *force*, int *atomID*)

Charge-field force due to a cubic spline charge function.

Author

Nathan Baker

Parameters

atomID	Set to force Valist atom ID

Definition at line 6439 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.28 VPRIVATE void qfForceSpline4 (Vpmg * *thee*, double * *force*, int *atomID*)

Charge-field force due to a quintic spline charge function.

Author

Michael Schnieders

Parameters

atomID	Set to force Valist atom ID

Definition at line 6552 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.29 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 7123 of file vpmg.c.

Here is the caller graph for this function:

9.95.2.30 VPRIVATE double Vpmg_polarizEnergy (Vpmg * *thee*, int *extFlag*)

Determines energy from polarizeable charge and interaction with fixed charges according to Rocchia et al.

Author

Nathan Baker

Returns

Energy in kT

Parameters

extFlag	If 1, add external energy contributions to result

Definition at line 1154 of file vpmg.c.

Here is the call graph for this function:

9.95.2.31 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

extFlag	If 1, add external energy contributions to result

Definition at line 1710 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.32 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

extFlag	If 1, add external energy contributions to result

Definition at line 1867 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.33 VPUBLIC double Vpmg_qmEnergySMPBE (Vpmg * *thee*, int *extFlag*)

Vpmg_qmEnergy for SMPBE.

Author

Vincent Chu

Definition at line 1496 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.34 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

acc	Surface method, currently VSM_SPLINE, VSM_SPLINE5, or VSM_SPLINE7
gpos	Accessibility object
win	Position array -> array[3]
infrad	Spline window
atom	Inflation radius
force	Atom object Force array -> array[3]

Definition at line 1899 of file vpmg.c.

Here is the call graph for this function:

Here is the caller graph for this function:

9.95.2.35 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

source	Solution term vector
work1	Source term vector Work vector

Definition at line 6889 of file vpmg.c.

Here is the caller graph for this function:

9.96 vpmg.c

```
#include "apbs/vpmg.h"
#include "apbs/vhal.h"

VEMBED(rcsid="$Id: vpmg.c 1793 2012-07-28 00:54:24Z sobolevnrm $")

#if !defined(VINLINE_VPMG)

VPUBLIC unsigned long int Vpmg_memChk(Vpmg *thee) {
    if (thee == VNULL) return 0;
    return Vmem_bytes(thee->vmem);
}

#endif /* if !defined(VINLINE_VPMG) */


VPUBLIC void Vpmg_printColComp(Vpmg *thee, char path[72], char title[72],
  char mxtype[3], int flag) {

    int nn, nxm2, nym2, nzm2, ncol, nrow, nonz;
    double *nzval;
    int *colptr, *rowind;

    /* Calculate the total number of unknowns */
    nxm2 = thee->pmgp->nx - 2;
    nym2 = thee->pmgp->ny - 2;
    nzm2 = thee->pmgp->nz - 2;
    nn = nxm2*nym2*nzm2;
    ncol = nn;
    nrow = nn;

    /* Calculate the number of non-zero matrix entries:
     *    nn       nonzeros on diagonal
     *    nn-1     nonzeros on first off-diagonal
     *    nn-nx    nonzeros on second off-diagonal
     *    nn-nx*ny nonzeros on third off-diagonal
     *
     *    7*nn-2*nx*ny-2*nx-2 TOTAL non-zeros
     */
    nonz = 7*nn - 2*nxm2*nym2 - 2*nxm2 - 2;
    nzval  = (double*)Vmem_malloc(thee->vmem, nonz, sizeof(double));
    rowind = (int*)Vmem_malloc(thee->vmem, nonz, sizeof(int));
    colptr = (int*)Vmem_malloc(thee->vmem, (ncol+1), sizeof(int));

#ifndef VAPBSQUIET
    Vnm_print(1, "Vpmg_printColComp:  Allocated space for %d nonzeros\n",
      nonz);
#endif

    bcolcomp(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
      nzval, rowind, colptr, &flag);


#if 0
    for (i=0; i<nn; i++) {
```

```
        Vnm_print(1, "nnz(%d) = %g\n", i, nzval[i]);
    }
#endif

    /* I do not understand why I need to pass nzval in this way, but it
     * works... */
    pcolcomp(&nrow, &ncol, &nonz, &(nzval[0]), rowind, colptr, path, title,
      mxtype);

    Vmem_free(thee->vmem, (ncol+1), sizeof(int), (void **)&colptr);
    Vmem_free(thee->vmem, nonz, sizeof(int), (void **)&rowind);
    Vmem_free(thee->vmem, nonz, sizeof(double), (void **)&nzval);

}

VPUBLIC Vpmg* Vpmg_ctor(Vpmgp *pmgp, Vpbe *pbe, int focusFlag,
        Vpmg *pmgOLD, MGparm *mgparm, PBEparm_calcEnergy energyFlag) {

    Vpmg *thee = VNULL;

    thee = (Vpmg*)Vmem_malloc(VNULL, 1, sizeof(Vpmg) );
    VASSERT(thee != VNULL);
    VASSERT( Vpmg_ctor2(thee, pmgp, pbe, focusFlag, pmgOLD, mgparm,
                energyFlag) );
    return thee;
}

VPUBLIC int Vpmg_ctor2(Vpmg *thee, Vpmgp *pmgp, Vpbe *pbe, int focusFlag,
        Vpmg *pmgOLD, MGparm *mgparm, PBEparm_calcEnergy energyFlag) {

    int i, j, nion;
    double ionConc[MAXION], ionQ[MAXION], ionRadii[MAXION], zkappa2, zks2;
    double ionstr, partMin[3], partMax[3];

    /* Get the parameters */
    VASSERT(pmgp != VNULL);
    VASSERT(pbe != VNULL);
    thee->pmgp = pmgp;
    thee->pbe = pbe;

    /* Set up the memory */
    thee->vmem = Vmem_ctor("APBS:VPMG");



 /* Initialize ion concentrations and valencies in PMG routines */
 zkappa2 = Vpbe_getZkappa2(thee->pbe);
 ionstr = Vpbe_getBulkIonicStrength(thee->pbe);
 if (ionstr > 0.0) zks2 = 0.5/ionstr;
 else zks2 = 0.0;
 Vpbe_getIons(thee->pbe, &nion, ionConc, ionRadii, ionQ);

 /* TEMPORARY USEAQUA */
        /* Calculate storage requirements */
 if(mgparm->useAqua == 0){
            Vpmgp_size(thee->pmgp);
 }else{
     VABORT_MSG0("Aqua is currently disabled");
 }

    /* We need some additional storage if: nonlinear & newton OR cgmg */
 /* SMPBE Added - nonlin = 2 added since it mimics NPBE */
    if ( ( ((thee->pmgp->nonlin == NONLIN_NPBE) || (thee->pmgp->nonlin == NONLIN_SMPBE))
     && (thee->pmgp->meth == VSOL_Newton) ) || (thee->pmgp->meth == VSOL_CGMG) )
 {
  thee->pmgp->nrwk += (2*(thee->pmgp->nf));
    }


        if (thee->pmgp->iinfo > 1) {
            Vnm_print(2, "Vpmg_ctor2:  PMG chose nx = %d, ny = %d, nz = %d\n",
                              thee->pmgp->nx, thee->pmgp->ny, thee->pmgp->nz);
```

```
            Vnm_print(2, "Vpmg_ctor2:  PMG chose nlev = %d\n",
                            thee->pmgp->nlev);
            Vnm_print(2, "Vpmg_ctor2:  PMG chose nxc = %d, nyc = %d, nzc = %d\n",
                            thee->pmgp->nxc, thee->pmgp->nyc, thee->pmgp->nzc);
            Vnm_print(2, "Vpmg_ctor2:  PMG chose nf = %d, nc = %d\n",
                            thee->pmgp->nf, thee->pmgp->nc);
            Vnm_print(2, "Vpmg_ctor2:  PMG chose narr = %d, narrc = %d\n",
                            thee->pmgp->narr, thee->pmgp->narrc);
            Vnm_print(2, "Vpmg_ctor2:  PMG chose n_rpc = %d, n_iz = %d, n_ipc = %d\n",
                            thee->pmgp->n_rpc, thee->pmgp->n_iz, thee->pmgp->n_ipc);
            Vnm_print(2, "Vpmg_ctor2:  PMG chose nrwk = %d, niwk = %d\n",
                            thee->pmgp->nrwk, thee->pmgp->niwk);
        }



 /* Allocate boundary storage */
 thee->gxcf = (double *)Vmem_malloc(
        thee->vmem,
        10*(thee->pmgp->ny)*(thee->pmgp->nz),
        sizeof(double)
        );

 thee->gycf = (double *)Vmem_malloc(
        thee->vmem,
        10*(thee->pmgp->nx)*(thee->pmgp->nz),
        sizeof(double)
        );

 thee->gzcf = (double *)Vmem_malloc(
        thee->vmem,
        10*(thee->pmgp->nx)*(thee->pmgp->ny),
        sizeof(double)
        );



 /* Warn users if they are using BCFL_MAP that
    we do not include external energies */
 if (thee->pmgp->bcfl == BCFL_MAP)
  Vnm_print(2,"Vpmg_ctor2: \nWarning: External energies are not used in BCFL_MAP calculations!\n");

 if (focusFlag) {

  /* Overwrite any default or user-specified boundary condition
  * arguments; we are now committed to a calculation via focusing */
  if (thee->pmgp->bcfl != BCFL_FOCUS) {
   Vnm_print(2,
       "Vpmg_ctor2: reset boundary condition flag to BCFL_FOCUS!\n");
   thee->pmgp->bcfl = BCFL_FOCUS;
  }

  /* Fill boundaries */
  Vnm_print(0, "Vpmg_ctor2:  Filling boundary with old solution!\n");
  focusFillBound(thee, pmgOLD);

  /* Calculate energetic contributions from region outside focusing
   * domain */
  if (energyFlag != PCE_NO) {

   if (mgparm->type == MCT_PARALLEL) {

    for (j=0; j<3; j++) {
     partMin[j] = mgparm->partDisjCenter[j]
     - 0.5*mgparm->partDisjLength[j];
     partMax[j] = mgparm->partDisjCenter[j]
      + 0.5*mgparm->partDisjLength[j];
    }

   } else {
    for (j=0; j<3; j++) {
     partMin[j] = mgparm->center[j] - 0.5*mgparm->glen[j];
     partMax[j] = mgparm->center[j] + 0.5*mgparm->glen[j];
    }
```

```
    }
    extEnergy(thee, pmgOLD, energyFlag, partMin, partMax,
        mgparm->partDisjOwnSide);
   }

  } else {

   /* Ignore external energy contributions */
   thee->extQmEnergy = 0;
   thee->extDiEnergy = 0;
   thee->extQfEnergy = 0;
  }

  /* Allocate partition vector storage */
  thee->pvec = (double *)Vmem_malloc(
         thee->vmem,
         (thee->pmgp->nx)*(thee->pmgp->ny)*(thee->pmgp->nz),
         sizeof(double)
         );

  /* Allocate remaining storage */
  thee->iparm  = (   int *)Vmem_malloc(thee->vmem,                100, sizeof(   int));
  thee->rparm  = (double *)Vmem_malloc(thee->vmem,                100, sizeof(double));
  thee->iwork  = (   int *)Vmem_malloc(thee->vmem,   thee->pmgp->niwk, sizeof(   int));
  thee->rwork  = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->nrwk, sizeof(double));
  thee->charge = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
  thee->kappa  = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
  thee->pot    = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
  thee->epsx   = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
  thee->epsy   = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
  thee->epsz   = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
  thee->a1cf   = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
  thee->a2cf   = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
  thee->a3cf   = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
  thee->ccf    = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
  thee->fcf    = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
  thee->tcf    = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
  thee->u      = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
  thee->xf     = (double *)Vmem_malloc(thee->vmem, 5*(thee->pmgp->nx), sizeof(double));
  thee->yf     = (double *)Vmem_malloc(thee->vmem, 5*(thee->pmgp->ny), sizeof(double));
  thee->zf     = (double *)Vmem_malloc(thee->vmem, 5*(thee->pmgp->nz), sizeof(double));



  /* Packs parameters into the iparm and rparm arrays */
     Vpackmg(thee->iparm, thee->rparm, &(thee->pmgp->nrwk), &(thee->pmgp->niwk),
        &(thee->pmgp->nx), &(thee->pmgp->ny), &(thee->pmgp->nz),
        &(thee->pmgp->nlev), &(thee->pmgp->nu1), &(thee->pmgp->nu2),
        &(thee->pmgp->mgkey), &(thee->pmgp->itmax), &(thee->pmgp->istop),
        &(thee->pmgp->ipcon), &(thee->pmgp->nonlin), &(thee->pmgp->mgsmoo),
        &(thee->pmgp->mgprol), &(thee->pmgp->mgcoar), &(thee->pmgp->mgsolv),
        &(thee->pmgp->mgdisc), &(thee->pmgp->iinfo), &(thee->pmgp->errtol),
        &(thee->pmgp->ipkey), &(thee->pmgp->omegal), &(thee->pmgp->omegan),
```

```
      &(thee->pmgp->irite), &(thee->pmgp->iperf));



 /* Currently for SMPBE type calculations we do not want to apply a scale
  factor to the ionConc */
 switch(pmgp->ipkey){

  case IPKEY_SMPBE:

                        Vmypdefinitsmpbe(&nion, ionQ, ionConc, &pbe->smvolume, &pbe->smsize);
   break;



  case IPKEY_NPBE:

   /* Else adjust the inoConc by scaling factor zks2 */
   for (i=0; i<nion; i++)
       ionConc[i] = zks2 * ionConc[i];

                        Vmypdefinitnpbe(&nion, ionQ, ionConc);
   break;



  case IPKEY_LPBE:

   /* Else adjust the inoConc by scaling factor zks2 */
   for (i=0; i<nion; i++)
                            ionConc[i] = zks2 * ionConc[i];

   Vmypdefinitlpbe(&nion, ionQ, ionConc);
   break;



  default:
   Vnm_print(2, "PMG: Warning: PBE structure not initialized!\n");
   /* Else adjust the inoConc by scaling factor zks2 */
   for (i=0; i<nion; i++)
                ionConc[i] = zks2 * ionConc[i];
   break;
 }

 /* Set the default chargeSrc for 5th order splines */
 thee->chargeSrc = mgparm->chgs;

 /* Turn off restriction of observable calculations to a specific
 * partition */
 Vpmg_unsetPart(thee);

 /* The coefficient arrays have not been filled */
 thee->filled = 0;


 /*
  * TODO: Move the dtor out of here. The current ctor is done in routines.c,
  *       This was originally moved out to kill a memory leak. The dtor has
  *       has been removed from initMG and placed back here to keep memory
  *       usage low. killMG has been modified accordingly.
  */
 Vpmg_dtor(&pmgOLD);

 return 1;
}

VPUBLIC int Vpmg_solve(Vpmg *thee) {

    int i,
        nx,
        ny,
        nz,
        n;
    double zkappa2;

    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    n = nx*ny*nz;
```

```

    if (!(thee->filled)) {
        Vnm_print(2, "Vpmg_solve:  Need to call Vpmg_fillco()!\n");
        return 0;
    }

    /* Fill the "true solution" array */
    for (i=0; i<n; i++) {
        thee->tcf[i] = 0.0;
    }

    /* Fill the RHS array */
    for (i=0; i<n; i++) {
        thee->fcf[i] = thee->charge[i];
    }

    /* Fill the operator coefficient array. */
    for (i=0; i<n; i++) {
        thee->a1cf[i] = thee->epsx[i];
        thee->a2cf[i] = thee->epsy[i];
        thee->a3cf[i] = thee->epsz[i];
    }

    /* Fill the nonlinear coefficient array by multiplying the kappa
     * accessibility array (containing values between 0 and 1) by zkappa2. */
    zkappa2 = Vpbe_getZkappa2(thee->pbe);
    if (zkappa2 > VPMGSMALL) {
        for (i=0; i<n; i++) {
            thee->ccf[i] = zkappa2*thee->kappa[i];
        }
    } else {
        for (i=0; i<n; i++) {
            thee->ccf[i] = 0.0;
        }
    }

    switch(thee->pmgp->meth) {
        /* CGMG (linear) */
        case VSOL_CGMG:

            if (thee->pmgp->iinfo > 1)
                Vnm_print(2, "Driving with CGMGDRIV\n");

            VABORT_MSG0("CGMGDRIV is not currently supported");
            break;

        /* Newton (nonlinear) */
        case VSOL_Newton:

            if (thee->pmgp->iinfo > 1)
                Vnm_print(2, "Driving with NEWDRIV\n");

            Vnewdriv
                       (thee->iparm, thee->rparm, thee->iwork, thee->rwork,
                        thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
                        thee->gzcf, thee->a1cf, thee->a2cf, thee->a3cf, thee->ccf,
                        thee->fcf, thee->tcf);
            break;

        /* MG (linear/nonlinear) */
        case VSOL_MG:

            if (thee->pmgp->iinfo > 1)
                Vnm_print(2, "Driving with MGDRIV\n");

            Vmgdriv(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
                                    thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
                                    thee->gzcf, thee->a1cf, thee->a2cf, thee->a3cf, thee->ccf,
                                    thee->fcf, thee->tcf);
            break;

        /* CGHS (linear/nonlinear) */
        case VSOL_CG:

            if (thee->pmgp->iinfo > 1)
```

```
                Vnm_print(2, "Driving with NCGHSDRIV\n");

            VABORT_MSG0("NCGHSDRIV is not currently supported");
            break;

        /* SOR (linear/nonlinear) */
        case VSOL_SOR:

            if (thee->pmgp->iinfo > 1)
                Vnm_print(2, "Driving with NSORDRIV\n");

            VABORT_MSG0("NSORDRIV is not currently supported");
            break;

        /* GSRB (linear/nonlinear) */
        case VSOL_RBGS:

            if (thee->pmgp->iinfo > 1)
                Vnm_print(2, "Driving with NGSRBDRIV\n");

            VABORT_MSG0("NGSRBDRIV is not currently supported");
            break;

        /* WJAC (linear/nonlinear) */
        case VSOL_WJ:

            if (thee->pmgp->iinfo > 1)
                Vnm_print(2, "Driving with NWJACDRIV\n");

            VABORT_MSG0("NWJACDRIV is not currently supported");
            break;

        /* RICH (linear/nonlinear) */
        case VSOL_Richardson:

            if (thee->pmgp->iinfo > 1)
                Vnm_print(2, "Driving with NRICHDRIV\n");

            VABORT_MSG0("NRICHDRIV is not currently supported");
            break;

        /* CGMG (linear) TEMPORARY USEAQUA */
        case VSOL_CGMGAqua:

            if (thee->pmgp->iinfo > 1)
                Vnm_print(2, "Driving with CGMGDRIVAQUA\n");

            VABORT_MSG0("CGMGDRIVAQUA is not currently supported");
            break;

        /* Newton (nonlinear) TEMPORARY USEAQUA */
        case VSOL_NewtonAqua:

            if (thee->pmgp->iinfo > 1)
                Vnm_print(2, "Driving with NEWDRIVAQUA\n");

            VABORT_MSG0("NEWDRIVAQUA is not currently supported");
            break;

        /* Error handling */
        default:
            Vnm_print(2, "Vpmg_solve: invalid solver method key (%d)\n",
              thee->pmgp->key);
            return 0;
            break;
    }

    return 1;

}


VPUBLIC void Vpmg_dtor(Vpmg **thee) {

    if ((*thee) != VNULL) {
        Vpmg_dtor2(*thee);
        Vmem_free(VNULL, 1, sizeof(Vpmg), (void **)thee);
        (*thee) = VNULL;
    }

}
```

```
VPUBLIC void Vpmg_dtor2(Vpmg *thee) {

    /* Clean up the storage */

    Vmem_free(thee->vmem,              100, sizeof(int),
      (void **)&(thee->iparm));
    Vmem_free(thee->vmem,              100, sizeof(double),
        (void **)&(thee->rparm));
    Vmem_free(thee->vmem, thee->pmgp->niwk, sizeof(int),
      (void **)&(thee->iwork));
    Vmem_free(thee->vmem, thee->pmgp->nrwk, sizeof(double),
      (void **)&(thee->rwork));
    Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
      (void **)&(thee->charge));
    Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
      (void **)&(thee->kappa));
 Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
     (void **)&(thee->pot));
    Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
      (void **)&(thee->epsx));
    Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
      (void **)&(thee->epsy));
    Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
      (void **)&(thee->epsz));
    Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
      (void **)&(thee->a1cf));
    Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
      (void **)&(thee->a2cf));
    Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
      (void **)&(thee->a3cf));
    Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
      (void **)&(thee->ccf));
    Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
      (void **)&(thee->fcf));
    Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
      (void **)&(thee->tcf));
    Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
      (void **)&(thee->u));
    Vmem_free(thee->vmem, 5*(thee->pmgp->nx), sizeof(double),
      (void **)&(thee->xf));
    Vmem_free(thee->vmem, 5*(thee->pmgp->ny), sizeof(double),
      (void **)&(thee->yf));
    Vmem_free(thee->vmem, 5*(thee->pmgp->nz), sizeof(double),
      (void **)&(thee->zf));
    Vmem_free(thee->vmem, 10*(thee->pmgp->ny)*(thee->pmgp->nz),
     sizeof(double),
      (void **)&(thee->gxcf));
    Vmem_free(thee->vmem, 10*(thee->pmgp->nx)*(thee->pmgp->nz),
     sizeof(double),
      (void **)&(thee->gycf));
    Vmem_free(thee->vmem, 10*(thee->pmgp->nx)*(thee->pmgp->ny),
     sizeof(double),
      (void **)&(thee->gzcf));
    Vmem_free(thee->vmem, (thee->pmgp->nx)*(thee->pmgp->ny)*(
    thee->pmgp->nz),
      sizeof(double), (void **)&(thee->pvec));

    Vmem_dtor(&(thee->vmem));
}

VPUBLIC void Vpmg_setPart(Vpmg *thee, double lowerCorner[3],
        double upperCorner[3], int bflags[6]) {

    Valist *alist;
    Vatom *atom;
    int i, j, k, nx, ny, nz;
    double xmin, ymin, zmin, x, y, z, hx, hy, hzed, xok,
    yok, zok;
    double x0,x1,y0,y1,z0,z1;

    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    xmin = thee->pmgp->xcent - 0.5*hx*(nx-1);
    ymin = thee->pmgp->ycent - 0.5*hy*(ny-1);
    zmin = thee->pmgp->zcent - 0.5*hzed*(nz-1);

```

```
    xok = 0;
    yok = 0;
    zok = 0;

    /* We need have called Vpmg_fillco first */

    alist = thee->pbe->alist;

    Vnm_print(0, "Vpmg_setPart:  lower corner = (%g, %g, %g)\n",
      lowerCorner[0], lowerCorner[1], lowerCorner[2]);
    Vnm_print(0, "Vpmg_setPart:  upper corner = (%g, %g, %g)\n",
      upperCorner[0], upperCorner[1], upperCorner[2]);
    Vnm_print(0, "Vpmg_setPart:  actual minima = (%g, %g, %g)\n",
      xmin, ymin, zmin);
    Vnm_print(0, "Vpmg_setPart:  actual maxima = (%g, %g, %g)\n",
      xmin+hx*(nx-1), ymin+hy*(ny-1), zmin+hzed*(nz-1));
    Vnm_print(0, "Vpmg_setPart:  bflag[FRONT] = %d\n",
      bflags[VAPBS_FRONT]);
    Vnm_print(0, "Vpmg_setPart:  bflag[BACK] = %d\n",
      bflags[VAPBS_BACK]);
    Vnm_print(0, "Vpmg_setPart:  bflag[LEFT] = %d\n",
      bflags[VAPBS_LEFT]);
    Vnm_print(0, "Vpmg_setPart:  bflag[RIGHT] = %d\n",
      bflags[VAPBS_RIGHT]);
    Vnm_print(0, "Vpmg_setPart:  bflag[UP] = %d\n",
      bflags[VAPBS_UP]);
    Vnm_print(0, "Vpmg_setPart:  bflag[DOWN] = %d\n",
      bflags[VAPBS_DOWN]);

    /* Identify atoms as inside, outside, or on the border
       If on the border, use the bflags to determine if there
       is an adjacent processor - if so, this atom should be equally
       shared. */

    for (i=0; i<Valist_getNumberAtoms(alist); i++) {
        atom = Valist_getAtom(alist, i);

        if ((atom->position[0] < upperCorner[0]) &&
            (atom->position[0] > lowerCorner[0])) xok = 1;
        else {
            if ((VABS(atom->position[0] - lowerCorner[0]) < VPMGSMALL) &&
                (bflags[VAPBS_LEFT] == 0)) xok = 1;
            else if ((VABS(atom->position[0] - lowerCorner[0]) < VPMGSMALL) &&
                (bflags[VAPBS_LEFT] == 1)) xok = 0.5;
            else if ((VABS(atom->position[0] - upperCorner[0]) < VPMGSMALL) &&
                (bflags[VAPBS_RIGHT] == 0)) xok = 1;
            else if ((VABS(atom->position[0] - upperCorner[0]) < VPMGSMALL) &&
                (bflags[VAPBS_RIGHT] == 1)) xok = 0.5;
            else xok = 0;
        }
        if ((atom->position[1] < upperCorner[1]) &&
            (atom->position[1] > lowerCorner[1])) yok = 1;
        else {
            if ((VABS(atom->position[1] - lowerCorner[1]) < VPMGSMALL) &&
                (bflags[VAPBS_BACK] == 0)) yok = 1;
            else if ((VABS(atom->position[1] - lowerCorner[1]) < VPMGSMALL) &&
                (bflags[VAPBS_BACK] == 1)) yok = 0.5;
            else if ((VABS(atom->position[1] - upperCorner[1]) < VPMGSMALL) &&
                (bflags[VAPBS_FRONT] == 0)) yok = 1;
            else if ((VABS(atom->position[1] - upperCorner[1]) < VPMGSMALL) &&
                (bflags[VAPBS_FRONT] == 1)) yok = 0.5;
            else yok = 0;
        }
        if ((atom->position[2] < upperCorner[2]) &&
            (atom->position[2] > lowerCorner[2])) zok = 1;
        else {
            if ((VABS(atom->position[2] - lowerCorner[2]) < VPMGSMALL) &&
                (bflags[VAPBS_DOWN] == 0)) zok = 1;
            else if ((VABS(atom->position[2] - lowerCorner[2]) < VPMGSMALL) &&
                (bflags[VAPBS_DOWN] == 1)) zok = 0.5;
            else if ((VABS(atom->position[2] - upperCorner[2]) <
```

```
VPMGSMALL) &&
                (bflags[VAPBS_UP] == 0)) zok = 1;
            else if ((VABS(atom->position[2] - upperCorner[2]) <
VPMGSMALL) &&
                (bflags[VAPBS_UP] == 1)) zok = 0.5;
            else zok = 0;
        }

        atom->partID = xok*yok*zok;
  /*
  Vnm_print(1, "DEBUG (%s, %d):  atom->position[0] - upperCorner[0] = %g\n",
      __FILE__, __LINE__, atom->position[0] - upperCorner[0]);
  Vnm_print(1, "DEBUG (%s, %d):  atom->position[0] - lowerCorner[0] = %g\n",
      __FILE__, __LINE__, atom->position[0] - lowerCorner[0]);
  Vnm_print(1, "DEBUG (%s, %d):  atom->position[1] - upperCorner[1] = %g\n",
      __FILE__, __LINE__, atom->position[1] - upperCorner[1]);
  Vnm_print(1, "DEBUG (%s, %d):  atom->position[1] - lowerCorner[1] = %g\n",
      __FILE__, __LINE__, atom->position[1] - lowerCorner[1]);
  Vnm_print(1, "DEBUG (%s, %d):  atom->position[2] - upperCorner[2] = %g\n",
      __FILE__, __LINE__, atom->position[2] - upperCorner[2]);
  Vnm_print(1, "DEBUG (%s, %d):  atom->position[2] - lowerCorner[0] = %g\n",
      __FILE__, __LINE__, atom->position[2] - lowerCorner[2]);
  Vnm_print(1, "DEBUG (%s, %d):  xok = %g, yok = %g, zok = %g\n",
      __FILE__, __LINE__, xok, yok, zok);
   */

 }

    /* Load up pvec -
       For all points within h{axis}/2 of a border - use a gradient
       to determine the pvec weight.
       Points on the boundary depend on the presence of an adjacent
       processor. */

    for (i=0; i<(nx*ny*nz); i++) thee->pvec[i] = 0.0;

    for (i=0; i<nx; i++) {
        xok = 0.0;
        x = i*hx + xmin;
        if ( (x < (upperCorner[0]-hx/2)) &&
             (x > (lowerCorner[0]+hx/2))
           ) xok = 1.0;
        else if ( (VABS(x - lowerCorner[0]) < VPMGSMALL) &&
                  (bflags[VAPBS_LEFT] == 0)) xok = 1.0;
        else if ((VABS(x - lowerCorner[0]) < VPMGSMALL) &&
                 (bflags[VAPBS_LEFT] == 1)) xok = 0.5;
        else if ((VABS(x - upperCorner[0]) < VPMGSMALL) &&
                 (bflags[VAPBS_RIGHT] == 0)) xok = 1.0;
        else if ((VABS(x - upperCorner[0]) < VPMGSMALL) &&
                 (bflags[VAPBS_RIGHT] == 1)) xok = 0.5;
        else if ((x > (upperCorner[0] + hx/2)) || (x < (lowerCorner[0] - hx/2))
) xok = 0.0;
        else if ((x < (upperCorner[0] + hx/2)) || (x > (lowerCorner[0] - hx/2))
) {
            x0 = VMAX2(x - hx/2, lowerCorner[0]);
            x1 = VMIN2(x + hx/2, upperCorner[0]);
            xok = VABS(x1-x0)/hx;

            if (xok < 0.0) {
                if (VABS(xok) < VPMGSMALL) xok = 0.0;
                else {
                    Vnm_print(2, "Vpmg_setPart:  fell off x-interval (%1.12E)!
\n",
                              xok);
                    VASSERT(0);
                }
            }
            if (xok > 1.0) {
                if (VABS(xok - 1.0) < VPMGSMALL) xok = 1.0;
                else {
                    Vnm_print(2, "Vpmg_setPart:  fell off x-interval (%1.12E)!
\n",
                              xok);
                    VASSERT(0);
                }
            }

        } else xok = 0.0;

        for (j=0; j<ny; j++) {
            yok = 0.0;
```

```
            y = j*hy + ymin;
            if ((y < (upperCorner[1]-hy/2)) && (y > (lowerCorner[1]+hy/2))) yok = 1.0;
            else if ((VABS(y - lowerCorner[1]) < VPMGSMALL) &&
                     (bflags[VAPBS_BACK] == 0)) yok = 1.0;
            else if ((VABS(y - lowerCorner[1]) < VPMGSMALL) &&
                     (bflags[VAPBS_BACK] == 1)) yok = 0.5;
            else if ((VABS(y - upperCorner[1]) < VPMGSMALL) &&
                     (bflags[VAPBS_FRONT] == 0)) yok = 1.0;
            else if ((VABS(y - upperCorner[1]) < VPMGSMALL) &&
                     (bflags[VAPBS_FRONT] == 1)) yok = 0.5;
            else if ((y > (upperCorner[1] + hy/2)) || (y < (lowerCorner[1] - hy/2))) yok=0.0;
            else if ((y < (upperCorner[1] + hy/2)) || (y > (lowerCorner[1] - hy/2))){
                y0 = VMAX2(y - hy/2, lowerCorner[1]);
                y1 = VMIN2(y + hy/2, upperCorner[1]);
                yok = VABS(y1-y0)/hy;

                if (yok < 0.0) {
                    if (VABS(yok) < VPMGSMALL) yok = 0.0;
                    else {
                        Vnm_print(2, "Vpmg_setPart:  fell off y-interval (%1.12E)!\n",
                                 yok);
                        VASSERT(0);
                    }
                }
                if (yok > 1.0) {
                    if (VABS(yok - 1.0) < VPMGSMALL) yok = 1.0;
                    else {
                        Vnm_print(2, "Vpmg_setPart:  fell off y-interval (%1.12E)!\n",
                                 yok);
                        VASSERT(0);
                    }
                }
            }
            else yok=0.0;

            for (k=0; k<nz; k++) {
                zok = 0.0;
                z = k*hzed + zmin;
                if ((z < (upperCorner[2]-hzed/2)) && (z > (lowerCorner[2]+hzed/2))) zok = 1.0;
                else if ((VABS(z - lowerCorner[2]) < VPMGSMALL) &&
                         (bflags[VAPBS_DOWN] == 0)) zok = 1.0;
                else if ((VABS(z - lowerCorner[2]) < VPMGSMALL) &&
                         (bflags[VAPBS_DOWN] == 1)) zok = 0.5;
                else if ((VABS(z - upperCorner[2]) < VPMGSMALL) &&
                         (bflags[VAPBS_UP] == 0)) zok = 1.0;
                else if ((VABS(z - upperCorner[2]) < VPMGSMALL) &&
                         (bflags[VAPBS_UP] == 1)) zok = 0.5;
                else if ((z > (upperCorner[2] + hzed/2)) || (z < (lowerCorner[2] - hzed/2))) zok=0.0;
                else if ((z < (upperCorner[2] + hzed/2)) || (z > (lowerCorner[2] - hzed/2))){
                    z0 = VMAX2(z - hzed/2, lowerCorner[2]);
                    z1 = VMIN2(z + hzed/2, upperCorner[2]);
                    zok = VABS(z1-z0)/hzed;

                    if (zok < 0.0) {
                        if (VABS(zok) < VPMGSMALL) zok = 0.0;
                        else {
                            Vnm_print(2, "Vpmg_setPart:  fell off z-interval (%1.12E)!\n",
                                     zok);
                            VASSERT(0);
                        }
                    }
                    if (zok > 1.0) {
                        if (VABS(zok - 1.0) < VPMGSMALL) zok = 1.0;
                        else {
                            Vnm_print(2, "Vpmg_setPart:  fell off z-interval (%1.12E)!\n",
                                     zok);
                            VASSERT(0);
                        }
                    }
                }
                else zok = 0.0;
```

```
                if (VABS(xok*yok*zok) < VPMGSMALL) thee->pvec[IJK(i,j,k)] = 0.0;
                else thee->pvec[IJK(i,j,k)] = xok*yok*zok;

            }
        }
    }
}

VPUBLIC void Vpmg_unsetPart(Vpmg *thee) {

    int i, nx, ny, nz;
    Vatom *atom;
    Valist *alist;

    VASSERT(thee != VNULL);

    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    alist = thee->pbe->alist;

    for (i=0; i<(nx*ny*nz); i++) thee->pvec[i] = 1;
    for (i=0; i<Valist_getNumberAtoms(alist); i++) {
        atom = Valist_getAtom(alist, i);
        atom->partID = 1;
    }
}

VPUBLIC int Vpmg_fillArray(Vpmg *thee, double *vec, Vdata_Type type,
  double parm, Vhal_PBEType pbetype, PBEparm *pbeparm) {

    Vacc *acc = VNULL;
    Vpbe *pbe = VNULL;
    Vgrid *grid = VNULL;
 Vatom *atoms = VNULL;
 Valist *alist = VNULL;
    double position[3], hx, hy, hzed, xmin, ymin, zmin;
    double grad[3], eps, epsp, epss, zmagic;
    int i, j, k, l, nx, ny, nz, ichop;

    pbe = thee->pbe;
    acc = Vpbe_getVacc(pbe);
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    xmin = thee->pmgp->xmin;
    ymin = thee->pmgp->ymin;
    zmin = thee->pmgp->zmin;
    epsp = Vpbe_getSoluteDiel(pbe);
    epss = Vpbe_getSolventDiel(pbe);
    zmagic = Vpbe_getZmagic(pbe);

    if (!(thee->filled)) {
        Vnm_print(2, "Vpmg_fillArray:  need to call Vpmg_fillco first!\n");
        return 0;
    }

    switch (type) {

        case VDT_CHARGE:

            for (i=0; i<nx*ny*nz; i++) vec[i] = thee->charge[i]/zmagic;
            break;

        case VDT_DIELX:

            for (i=0; i<nx*ny*nz; i++) vec[i] = thee->epsx[i];
            break;

        case VDT_DIELY:

            for (i=0; i<nx*ny*nz; i++) vec[i] = thee->epsy[i];
            break;

        case VDT_DIELZ:
```

```
            for (i=0; i<nx*ny*nz; i++) vec[i] = thee->epsz[i];
            break;

        case VDT_KAPPA:

            for (i=0; i<nx*ny*nz; i++) vec[i] = thee->kappa[i];
            break;

        case VDT_POT:

            for (i=0; i<nx*ny*nz; i++) vec[i] = thee->u[i];
            break;

  case VDT_ATOMPOT:
   alist = thee->pbe->alist;
   atoms = alist[pbeparm->molid-1].atoms;
   grid = Vgrid_ctor(nx, ny, nz, hx, hy,
         hzed, xmin, ymin, zmin,thee->u);
   for (i=0; i<alist[pbeparm->molid-1].number;i++) {
    position[0] = atoms[i].position[0];
    position[1] = atoms[i].position[1];
    position[2] = atoms[i].position[2];

    Vgrid_value(grid, position, &vec[i]);
   }
   Vgrid_dtor(&grid);
   break;

        case VDT_SMOL:

            for (k=0; k<nz; k++) {
                for (j=0; j<ny; j++) {
                    for (i=0; i<nx; i++) {

                        position[0] = i*hx + xmin;
                        position[1] = j*hy + ymin;
                        position[2] = k*hzed + zmin;

                        vec[IJK(i,j,k)] = (Vacc_molAcc(acc,position,
parm));
                    }
                }
            }
            break;

        case VDT_SSPL:

            for (k=0; k<nz; k++) {
                for (j=0; j<ny; j++) {
                    for (i=0; i<nx; i++) {

                        position[0] = i*hx + xmin;
                        position[1] = j*hy + ymin;
                        position[2] = k*hzed + zmin;

                        vec[IJK(i,j,k)] = Vacc_splineAcc(acc,
position,parm,0);
                    }
                }
            }
            break;

        case VDT_VDW:

            for (k=0; k<nz; k++) {
                for (j=0; j<ny; j++) {
                    for (i=0; i<nx; i++) {

                        position[0] = i*hx + xmin;
                        position[1] = j*hy + ymin;
                        position[2] = k*hzed + zmin;

                        vec[IJK(i,j,k)] = Vacc_vdwAcc(acc,position);
                    }
                }
            }
            break;

        case VDT_IVDW:

```

```
            for (k=0; k<nz; k++) {
                for (j=0; j<ny; j++) {
                    for (i=0; i<nx; i++) {

                        position[0] = i*hx + xmin;
                        position[1] = j*hy + ymin;
                        position[2] = k*hzed + zmin;

                        vec[IJK(i,j,k)] = Vacc_ivdwAcc(acc,position,parm);
                    }
                }
            }
            break;

        case VDT_LAP:

            grid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin,
              thee->u);
            for (k=0; k<nz; k++) {
                for (j=0; j<ny; j++) {
                    for (i=0; i<nx; i++) {

                        if ((k==0) || (k==(nz-1)) ||
                            (j==0) || (j==(ny-1)) ||
                            (i==0) || (i==(nx-1))) {

                            vec[IJK(i,j,k)] = 0;

                        } else {
                                position[0] = i*hx + xmin;
                                position[1] = j*hy + ymin;
                                position[2] = k*hzed + zmin;
                                VASSERT(Vgrid_curvature(grid, position, 1,
                                  &(vec[IJK(i,j,k)])));
                        }
                    }
                }
            }
            Vgrid_dtor(&grid);
            break;

        case VDT_EDENS:

            grid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin,
              thee->u);
            for (k=0; k<nz; k++) {
                for (j=0; j<ny; j++) {
                    for (i=0; i<nx; i++) {

                        position[0] = i*hx + xmin;
                        position[1] = j*hy + ymin;
                        position[2] = k*hzed + zmin;
                        VASSERT(Vgrid_gradient(grid, position, grad));
                        eps = epsp + (epss-epsp)*Vacc_molAcc(acc, position,
                          pbe->solventRadius);
                        vec[IJK(i,j,k)] = 0.0;
                        for (l=0; l<3; l++)
                          vec[IJK(i,j,k)] += eps*VSQR(grad[l]);
                    }
                }
            }
            Vgrid_dtor(&grid);
            break;

        case VDT_NDENS:

            for (k=0; k<nz; k++) {
                for (j=0; j<ny; j++) {
                    for (i=0; i<nx; i++) {

                        position[0] = i*hx + xmin;
                        position[1] = j*hy + ymin;
                        position[2] = k*hzed + zmin;
                        vec[IJK(i,j,k)] = 0.0;
                        if ( VABS(Vacc_ivdwAcc(acc,
```

```
                                position, pbe->maxIonRadius) - 1.0) < VSMALL) {
                            for (l=0; l<pbe->numIon; l++) {
                                if (pbetype == PBE_NPBE || pbetype == PBE_SMPBE /*  SMPBE Added */) {
                                    vec[IJK(i,j,k)] += (pbe->ionConc[l]
                                        * Vcap_exp(-pbe->ionQ[l]*thee->u[IJK(i,j,k)],
                                        &ichop));
                                } else if (pbetype == PBE_LPBE){
                                    vec[IJK(i,j,k)] += (pbe->ionConc[l]
                                        * (1 - pbe->ionQ[l]*thee->u[IJK(i,j,k)]));
                                }
                            }
                        }
                    }
                }
            }
            break;

        case VDT_QDENS:

            for (k=0; k<nz; k++) {
                for (j=0; j<ny; j++) {
                    for (i=0; i<nx; i++) {

                        position[0] = i*hx + xmin;
                        position[1] = j*hy + ymin;
                        position[2] = k*hzed + zmin;
                        vec[IJK(i,j,k)] = 0.0;
                        if ( VABS(Vacc_ivdwAcc(acc,
                                position, pbe->maxIonRadius) - 1.0) < VSMALL) {
                            for (l=0; l<pbe->numIon; l++) {
                                if (pbetype == PBE_NPBE || pbetype == PBE_SMPBE /*  SMPBE Added */) {
                                    vec[IJK(i,j,k)] += (pbe->ionConc[l]
                                        * pbe->ionQ[l]
                                        * Vcap_exp(-pbe->ionQ[l]*thee->u[IJK(i,j,k)],
                                        &ichop));
                                } else if (pbetype == PBE_LPBE) {
                                    vec[IJK(i,j,k)] += (pbe->ionConc[l]
                                        * pbe->ionQ[l]
                                        * (1 - pbe->ionQ[l]*thee->u[IJK(i,j,k)]));
                                }
                            }
                        }
                    }
                }
            }
            break;

        default:

            Vnm_print(2, "main:  Bogus data type (%d)!\n", type);
            return 0;
            break;

    }

    return 1;

}

VPRIVATE double Vpmg_polarizEnergy(Vpmg *thee,
                                   int extFlag
                                  ) {

    int i,
        j,
        k,
        ijk,
        nx,
        ny,
        nz,
        iatom;
    double xmin,
           ymin,
```

```
            zmin,
            //x, // gcc: not used
            //y,
            //z,
            hx,
            hy,
            hzed,
            epsp,
            lap,
            pt[3],
            T,
            pre,
            polq,
            dist2,
            dist,
            energy,
            q,
            *charge,
            *pos,
            eps_w;
    Vgrid *potgrid;
    Vpbe *pbe;
    Valist *alist;
    Vatom *atom;

    xmin = thee->pmgp->xmin;
    ymin = thee->pmgp->ymin;
    zmin = thee->pmgp->ymin;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    pbe = thee->pbe;
    epsp = Vpbe_getSoluteDiel(pbe);
    eps_w = Vpbe_getSolventDiel(pbe);
    alist = pbe->alist;
    charge = thee->charge;

    /* Calculate the prefactor for Coulombic calculations */
    T = Vpbe_getTemperature(pbe);
    pre = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T);
    pre = pre*(1.0e10);

    /* Set up Vgrid object with solution */
    potgrid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin, thee->u);

    /* Calculate polarization charge */
    energy = 0.0;
    for (i=1; i<(nx-1); i++) {
        pt[0] = xmin + hx*i;
        for (j=1; j<(ny-1); j++) {
            pt[1] = ymin + hy*j;
            for (k=1; k<(nz-1); k++) {
                pt[2] = zmin + hzed*k;

                /* Calculate polarization charge */
                VASSERT(Vgrid_curvature(potgrid, pt, 1, &lap));
                ijk = IJK(i,j,k);
                polq = charge[ijk] + epsp*lap*3.0;

                /* Calculate interaction energy with atoms */
                if (VABS(polq) > VSMALL) {
                    for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
                        atom = Valist_getAtom(alist, iatom);
                        q = Vatom_getCharge(atom);
                        pos = Vatom_getPosition(atom);
                        dist2 = VSQR(pos[0]-pt[0]) + VSQR(pos[1]-pt[1]) \
                                + VSQR(pos[2]-pt[2]);
                        dist = VSQRT(dist2);

                        if (dist < VSMALL) {
                            Vnm_print(2, "Vpmg_polarizEnergy:  atom on grid point; ignoring!\n");
                        } else {
                            energy = energy + polq*q/dist;
                        }
```

```
                    }
                }
            }
        }
    }

    return pre*energy;
}

VPUBLIC double Vpmg_energy(Vpmg *thee,
                           int extFlag
                          ) {

    double totEnergy = 0.0,
           dielEnergy = 0.0,
           qmEnergy = 0.0,
           qfEnergy = 0.0;

    VASSERT(thee != VNULL);

    if ((thee->pmgp->nonlin) && (Vpbe_getBulkIonicStrength(thee->pbe) > 0.)) {
        Vnm_print(0, "Vpmg_energy:  calculating full PBE energy\n");
        qmEnergy = Vpmg_qmEnergy(thee, extFlag);
        Vnm_print(0, "Vpmg_energy:  qmEnergy = %1.12E kT\n", qmEnergy);
        qfEnergy = Vpmg_qfEnergy(thee, extFlag);
        Vnm_print(0, "Vpmg_energy:  qfEnergy = %1.12E kT\n", qfEnergy);
        dielEnergy = Vpmg_dielEnergy(thee, extFlag);
        Vnm_print(0, "Vpmg_energy:  dielEnergy = %1.12E kT\n", dielEnergy);
        totEnergy = qfEnergy - dielEnergy - qmEnergy;
    } else {
        Vnm_print(0, "Vpmg_energy:  calculating only q-phi energy\n");
        qfEnergy = Vpmg_qfEnergy(thee, extFlag);
        Vnm_print(0, "Vpmg_energy:  qfEnergy = %1.12E kT\n", qfEnergy);
        totEnergy = 0.5*qfEnergy;
    }

    return totEnergy;

}

VPUBLIC double Vpmg_dielEnergy(Vpmg *thee,
                               int extFlag
                              ) {

    double hx,
           hy,
           hzed,
           energy,
           nrgx,
           nrgy,
           nrgz,
           pvecx,
           pvecy,
           pvecz;
    int i,
        j,
        k,
        nx,
        ny,
        nz;

    VASSERT(thee != VNULL);

    /* Get the mesh information */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;

    energy = 0.0;

    if (!thee->filled) {
        Vnm_print(2, "Vpmg_dielEnergy:  Need to call Vpmg_fillco!\n");
        VASSERT(0);
    }

    for (k=0; k<(nz-1); k++) {
        for (j=0; j<(ny-1); j++) {
```

```
            for (i=0; i<(nx-1); i++) {
                pvecx = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i+1,j,k)]);
                pvecy = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j+1,k)]);
                pvecz = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j,k+1)]);
                nrgx = thee->epsx[IJK(i,j,k)]*pvecx
                  * VSQR((thee->u[IJK(i,j,k)]-thee->u[IJK(i+1,j,k)])/hx);
                nrgy = thee->epsy[IJK(i,j,k)]*pvecy
                  * VSQR((thee->u[IJK(i,j,k)]-thee->u[IJK(i,j+1,k)])/hy);
                nrgz = thee->epsz[IJK(i,j,k)]*pvecz
                  * VSQR((thee->u[IJK(i,j,k)]-thee->u[IJK(i,j,k+1)])/hzed);
                energy += (nrgx + nrgy + nrgz);
            }
        }
    }

    energy = 0.5*energy*hx*hy*hzed;
    energy = energy/Vpbe_getZmagic(thee->pbe);

    if (extFlag == 1) energy += (thee->extDiEnergy);

    return energy;
}

VPUBLIC double Vpmg_dielGradNorm(Vpmg *thee) {

    double hx, hy, hzed, energy, nrgx, nrgy, nrgz, pvecx, pvecy, pvecz;
    int i, j, k, nx, ny, nz;

    VASSERT(thee != VNULL);

    /* Get the mesh information */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;

    energy = 0.0;

    if (!thee->filled) {
        Vnm_print(2, "Vpmg_dielGradNorm:  Need to call Vpmg_fillco!\n");
        VASSERT(0);
    }

    for (k=1; k<nz; k++) {
        for (j=1; j<ny; j++) {
            for (i=1; i<nx; i++) {
                pvecx = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i-1,j,k)]);
                pvecy = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j-1,k)]);
                pvecz = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j,k-1)]);
                nrgx = pvecx
                 * VSQR((thee->epsx[IJK(i,j,k)]-thee->epsx[IJK(i-1,j,k)])/hx);
                nrgy = pvecy
                 * VSQR((thee->epsy[IJK(i,j,k)]-thee->epsy[IJK(i,j-1,k)])/hy);
                nrgz = pvecz
                 * VSQR((thee->epsz[IJK(i,j,k)]-thee->epsz[IJK(i,j,k-1)])/hzed);
                energy += VSQRT(nrgx + nrgy + nrgz);
            }
        }
    }

    energy = energy*hx*hy*hzed;

    return energy;
}

VPUBLIC double Vpmg_qmEnergy(Vpmg *thee,
                             int extFlag
                            ) {

 double energy;
```

```
 if(thee->pbe->ipkey == IPKEY_SMPBE){
  energy = Vpmg_qmEnergySMPBE(thee,extFlag);
 }else{
  energy = Vpmg_qmEnergyNONLIN(thee,extFlag);
 }

 return energy;
}

VPRIVATE double Vpmg_qmEnergyNONLIN(Vpmg *thee,
                                    int extFlag
                                   ) {

    double hx,
           hy,
           hzed,
           energy,
           ionConc[MAXION],
           ionRadii[MAXION],
           ionQ[MAXION],
           zkappa2,
           ionstr,
           zks2;
    int i, /* Loop variable */
        j,
        nx,
        ny,
        nz,
        nion,
        ichop,
        nchop,
        len; /* Stores number of iterations for loops to avoid multiple recalculations */

    VASSERT(thee != VNULL);

    /* Get the mesh information */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    zkappa2 = Vpbe_getZkappa2(thee->pbe);
    ionstr = Vpbe_getBulkIonicStrength(thee->pbe);

    /* Bail if we're at zero ionic strength */
    if (zkappa2 < VSMALL) {

#ifndef VAPBSQUIET
        Vnm_print(0, "Vpmg_qmEnergy:  Zero energy for zero ionic strength!\n");
#endif

        return 0.0;
    }
    zks2 = 0.5*zkappa2/ionstr;

    if (!thee->filled) {
        Vnm_print(2, "Vpmg_qmEnergy:  Need to call Vpmg_fillco()!\n");
        VASSERT(0);
    }

    energy = 0.0;
    nchop = 0;
    Vpbe_getIons(thee->pbe, &nion, ionConc, ionRadii, ionQ);
    if (thee->pmgp->nonlin) {
        Vnm_print(0, "Vpmg_qmEnergy:  Calculating nonlinear energy\n");
        for (i=0, len=nx*ny*nz; i<len; i++) {
            if (thee->pvec[i]*thee->kappa[i] > VSMALL) {
                for (j=0; j<nion; j++) {
                    energy += (thee->pvec[i]*thee->kappa[i]*zks2
                      * ionConc[j]
                      * (Vcap_exp(-ionQ[j]*thee->u[i], &ichop)-1.0));
                    nchop += ichop;
                }
            }
        }
        if (nchop > 0){
   Vnm_print(2, "Vpmg_qmEnergy:  Chopped EXP %d times!\n",nchop);
   Vnm_print(2, "\nERROR!  Detected large potential values in energy
```

```
 evaluation! \nERROR!  This calculation failed -- please report to the APBS developers!\n\n
");
   VASSERT(0);
  }
    } else {
        /* Zkappa2 OK here b/c LPBE approx */
        Vnm_print(0, "Vpmg_qmEnergy:  Calculating linear energy\n");
        for (i=0, len=nx*ny*nz; i<len; i++) {
            if (thee->pvec[i]*thee->kappa[i] > VSMALL)
              energy += (thee->pvec[i]*zkappa2*thee->kappa[i]*VSQR(
thee->u[i]));
        }
        energy = 0.5*energy;
    }
    energy = energy*hx*hy*hzed;
    energy = energy/Vpbe_getZmagic(thee->pbe);

    if (extFlag == 1) energy += thee->extQmEnergy;

    return energy;
}

VPUBLIC double Vpmg_qmEnergySMPBE(Vpmg *thee,
                                  int extFlag
                                 ) {

    double hx,
           hy,
           hzed,
           energy,
           ionConc[MAXION],
           ionRadii[MAXION],
           ionQ[MAXION],
           zkappa2,
           ionstr,
           zks2;
    int i,
        //j, // gcc: not used
        nx,
        ny,
        nz,
        nion,
        //ichop, // gcc: not used
        nchop,
        len; /* Loop variable */

    /* SMPB Modification (vchu, 09/21/06)*/
    /* variable declarations for SMPB energy terms */
    double a,
           k,
           z1,
           z2,
           z3,
           cb1,
           cb2,
           cb3,
           a1,
           a2,
           a3,
           c1,
           c2,
           c3,
           currEnergy,
           fracOccA,
           fracOccB,
           fracOccC,
           phi,
           gpark,
           denom;
           // Na; /**< @todo remove if no conflicts are caused - This constant
 is already defined in vpde.h.  no need to redefine. */
    int ichop1,
        ichop2,
        ichop3;

    VASSERT(thee != VNULL);

    /* Get the mesh information */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
```

```
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    zkappa2 = Vpbe_getZkappa2(thee->pbe);
    ionstr = Vpbe_getBulkIonicStrength(thee->pbe);

    /* Bail if we're at zero ionic strength */
    if (zkappa2 < VSMALL) {

#ifndef VAPBSQUIET
        Vnm_print(0, "Vpmg_qmEnergySMPBE:  Zero energy for zero ionic strength!\n");
#endif

        return 0.0;
    }
    zks2 = 0.5*zkappa2/ionstr;

    if (!thee->filled) {
        Vnm_print(2, "Vpmg_qmEnergySMPBE:  Need to call Vpmg_fillco()!\n");
        VASSERT(0);
    }

    energy = 0.0;
    nchop = 0;
    Vpbe_getIons(thee->pbe, &nion, ionConc, ionRadii, ionQ);

    /* SMPB Modification (vchu, 09/21/06) */
    /* Extensive modification to the first part of the if statement
  where that handles the thee->pmgp->nonlin part. Basically, I've
  deleted all of the original code and written my own code that computes
  the electrostatic free energy in the SMPB framework. Definitely really hacky
  at this stage of the game, but gets the job done. The second part of the
  if statement (the part that handles linear poisson-boltzmann) has been deleted
  because there will be no linearized SMPB energy.. */

    z1 = ionQ[0];
    z2 = ionQ[1];
    z3 = ionQ[2];
    cb1 = ionConc[0];
    cb2 = ionConc[1];
    cb3 = ionConc[2];
    a  = thee->pbe->smvolume;
    k  = thee->pbe->smsize;

    // This constant is defined in vpde.h  Do not need to redefine
    //Na = 6.022045000e-04; /* Converts from Molar to N/A^3 */

    fracOccA = Na*cb1*VCUB(a);
    fracOccB = Na*cb2*VCUB(a);
    fracOccC = Na*cb3*VCUB(a);

    phi = (fracOccA/k) + fracOccB + fracOccC;

    if (thee->pmgp->nonlin) {
        Vnm_print(0, "Vpmg_qmEnergySMPBE:  Calculating nonlinear energy using SMPB functional!\n");
        for (i=0, len=nx*ny*nz; i<len; i++) {
   if (((k-1) > VSMALL) && (thee->pvec[i]*thee->kappa[i] > VSMALL)) {

    a1 = Vcap_exp(-1.0*z1*thee->u[i], &ichop1);
    a2 = Vcap_exp(-1.0*z2*thee->u[i], &ichop2);
    a3 = Vcap_exp(-1.0*z3*thee->u[i], &ichop3);

    nchop += ichop1 + ichop2 + ichop3;

    gpark = (1 - phi + (fracOccA/k)*a1);
    denom = VPOW(gpark, k) + VPOW(1-fracOccB-fracOccC, k-1)*(fracOccB*a2+fracOccC*a3);

    if (cb1 > VSMALL) {
     c1 = Na*cb1*VPOW(gpark, k-1)*a1/denom;
     if(c1 != c1) c1 = 0.;
    } else c1 = 0.;

    if (cb2 > VSMALL) {
     c2 = Na*cb2*VPOW(1-fracOccB-fracOccC,k-1)*a2/denom;
     if(c2 != c2) c2 = 0.;
    } else c2 = 0.;

```

```
    if (cb3 > VSMALL) {
     c3 = Na*cb3*VPOW(1-fracOccB-fracOccC,k-1)*a3/denom;
     if(c3 != c3) c3 = 0.;
    } else c3 = 0.;

    currEnergy = k*VLOG((1-(c1*VCUB(a)/k)-c2*VCUB(a)-c3*VCUB(a))/(1-phi))
     -(k-1)*VLOG((1-c2*VCUB(a)-c3*VCUB(a))/(1-phi+(fracOccA/k)));

    energy += thee->pvec[i]*thee->kappa[i]*currEnergy;

   } else if (thee->pvec[i]*thee->kappa[i] > VSMALL){

    a1 = Vcap_exp(-1.0*z1*thee->u[i], &ichop1);
    a2 = Vcap_exp(-1.0*z2*thee->u[i], &ichop2);
    a3 = Vcap_exp(-1.0*z3*thee->u[i], &ichop3);

    nchop += ichop1 + ichop2 + ichop3;

    gpark = (1 - phi + (fracOccA)*a1);
    denom = gpark + (fracOccB*a2+fracOccC*a3);

    if (cb1 > VSMALL) {
     c1 = Na*cb1*a1/denom;
     if(c1 != c1) c1 = 0.;
    } else c1 = 0.;

    if (cb2 > VSMALL) {
     c2 = Na*cb2*a2/denom;
     if(c2 != c2) c2 = 0.;
    } else c2 = 0.;

    if (cb3 > VSMALL) {
     c3 = Na*cb3*a3/denom;
     if(c3 != c3) c3 = 0.;
    } else c3 = 0.;

    currEnergy = VLOG((1-c1*VCUB(a)-c2*VCUB(a)-c3*VCUB(a))/(1-fracOccA-fracOccB-fracOccC));

    energy += thee->pvec[i]*thee->kappa[i]*currEnergy;
   }
        }

  energy = -energy/VCUB(a);

        if (nchop > 0) Vnm_print(2, "Vpmg_qmEnergySMPBE:  Chopped EXP %d times!\n",
         nchop);

    } else {
        /* Zkappa2 OK here b/c LPBE approx */
        Vnm_print(0, "Vpmg_qmEnergySMPBE:  ERROR: NO LINEAR ENERGY!! Returning 0!\n");

  energy = 0.0;

    }
    energy = energy*hx*hy*hzed;

    if (extFlag == 1) energy += thee->extQmEnergy;

    return energy;
}

VPUBLIC double Vpmg_qfEnergy(Vpmg *thee,
                             int extFlag
                            ) {

    double energy = 0.0;

    VASSERT(thee != VNULL);

    if ((thee->useChargeMap) || (thee->chargeMeth == VCM_BSPL2)) {
        energy = Vpmg_qfEnergyVolume(thee, extFlag);
    } else {
        energy = Vpmg_qfEnergyPoint(thee, extFlag);
    }

    return energy;
}
```

```c

VPRIVATE double Vpmg_qfEnergyPoint(Vpmg *thee,
                                   int extFlag
                                  ) {

    int iatom, nx, ny, nz, ihi, ilo, jhi, jlo, khi, klo;
    double xmax, ymax, zmax, xmin, ymin, zmin, hx, hy, hzed, ifloat, jfloat;
    double charge, kfloat, dx, dy, dz, energy, uval, *position;
    double *u;
    double *pvec;
    Valist *alist;
    Vatom *atom;
    Vpbe *pbe;

    pbe = thee->pbe;
    alist = pbe->alist;
    VASSERT(alist != VNULL);

    /* Get the mesh information */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    xmax = thee->pmgp->xmax;
    ymax = thee->pmgp->ymax;
    zmax = thee->pmgp->zmax;
    xmin = thee->pmgp->xmin;
    ymin = thee->pmgp->ymin;
    zmin = thee->pmgp->zmin;

    u = thee->u;
    pvec = thee->pvec;

    energy = 0.0;

    for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {

        /* Get atomic information */
        atom = Valist_getAtom(alist, iatom);

        position = Vatom_getPosition(atom);
        charge = Vatom_getCharge(atom);

        /* Figure out which vertices we're next to */
        ifloat = (position[0] - xmin)/hx;
        jfloat = (position[1] - ymin)/hy;
        kfloat = (position[2] - zmin)/hzed;
        ihi = (int)ceil(ifloat);
        ilo = (int)floor(ifloat);
        jhi = (int)ceil(jfloat);
        jlo = (int)floor(jfloat);
        khi = (int)ceil(kfloat);
        klo = (int)floor(kfloat);

        if (atom->partID > 0) {

            if ((ihi<nx) && (jhi<ny) && (khi<nz) &&
                (ilo>=0) && (jlo>=0) && (klo>=0)) {

                /* Now get trilinear interpolation constants */
                dx = ifloat - (double)(ilo);
                dy = jfloat - (double)(jlo);
                dz = kfloat - (double)(klo);
                uval =
                  dx*dy*dz*u[IJK(ihi,jhi,khi)]
                + dx*(1.0-dy)*dz*u[IJK(ihi,jlo,khi)]
                + dx*dy*(1.0-dz)*u[IJK(ihi,jhi,klo)]
                + dx*(1.0-dy)*(1.0-dz)*u[IJK(ihi,jlo,klo)]
                + (1.0-dx)*dy*dz*u[IJK(ilo,jhi,khi)]
                + (1.0-dx)*(1.0-dy)*dz*u[IJK(ilo,jlo,khi)]
                + (1.0-dx)*dy*(1.0-dz)*u[IJK(ilo,jhi,klo)]
                + (1.0-dx)*(1.0-dy)*(1.0-dz)*u[IJK(ilo,jlo,klo)];
                energy += (uval*charge*atom->partID);
            } else if (thee->pmgp->bcfl != BCFL_FOCUS) {
                Vnm_print(2, "Vpmg_qfEnergy:  Atom #%d at (%4.3f, %4.3f, \
%4.3f) is off the mesh (ignoring)!\n",
                iatom, position[0], position[1], position[2]);
```

```
            }
        }
    }

    if (extFlag) energy += thee->extQfEnergy;

    return energy;
}

VPUBLIC double Vpmg_qfAtomEnergy(Vpmg *thee, Vatom *atom) {

    int nx, ny, nz, ihi, ilo, jhi, jlo, khi, klo;
    double xmax, xmin, ymax, ymin, zmax, zmin, hx, hy, hzed, ifloat, jfloat;
    double charge, kfloat, dx, dy, dz, energy, uval, *position;
    double *u;


    /* Get the mesh information */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    xmax = thee->xf[nx-1];
    ymax = thee->yf[ny-1];
    zmax = thee->zf[nz-1];
    xmin = thee->xf[0];
    ymin = thee->yf[0];
    zmin = thee->zf[0];

    u = thee->u;

    energy = 0.0;


    position = Vatom_getPosition(atom);
    charge = Vatom_getCharge(atom);

    /* Figure out which vertices we're next to */
    ifloat = (position[0] - xmin)/hx;
    jfloat = (position[1] - ymin)/hy;
    kfloat = (position[2] - zmin)/hzed;
    ihi = (int)ceil(ifloat);
    ilo = (int)floor(ifloat);
    jhi = (int)ceil(jfloat);
    jlo = (int)floor(jfloat);
    khi = (int)ceil(kfloat);
    klo = (int)floor(kfloat);

    if (atom->partID > 0) {

        if ((ihi<nx) && (jhi<ny) && (khi<nz) &&
            (ilo>=0) && (jlo>=0) && (klo>=0)) {

            /* Now get trilinear interpolation constants */
            dx = ifloat - (double)(ilo);
            dy = jfloat - (double)(jlo);
            dz = kfloat - (double)(klo);
            uval =
              dx*dy*dz*u[IJK(ihi,jhi,khi)]
            + dx*(1.0-dy)*dz*u[IJK(ihi,jlo,khi)]
            + dx*dy*(1.0-dz)*u[IJK(ihi,jhi,klo)]
            + dx*(1.0-dy)*(1.0-dz)*u[IJK(ihi,jlo,klo)]
            + (1.0-dx)*dy*dz*u[IJK(ilo,jhi,khi)]
            + (1.0-dx)*(1.0-dy)*dz*u[IJK(ilo,jlo,khi)]
            + (1.0-dx)*dy*(1.0-dz)*u[IJK(ilo,jhi,klo)]
            + (1.0-dx)*(1.0-dy)*(1.0-dz)*u[IJK(ilo,jlo,klo)];
            energy += (uval*charge*atom->partID);
        } else if (thee->pmgp->bcfl != BCFL_FOCUS) {
            Vnm_print(2, "Vpmg_qfAtomEnergy:  Atom at (%4.3f, %4.3f, \
%4.3f) is off the mesh (ignoring)!\n",
            position[0], position[1], position[2]);
        }
    }

    return energy;
}

```

```
VPRIVATE double Vpmg_qfEnergyVolume(Vpmg *thee, int extFlag) {

    double hx, hy, hzed, energy;
    int i, nx, ny, nz;

    VASSERT(thee != VNULL);

    /* Get the mesh information */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;

    if (!thee->filled) {
        Vnm_print(2, "Vpmg_qfEnergyVolume:  need to call Vpmg_fillco!\n");
        VASSERT(0);
    }

    energy = 0.0;
    Vnm_print(0, "Vpmg_qfEnergyVolume:  Calculating energy\n");
    for (i=0; i<(nx*ny*nz); i++) {
        energy += (thee->pvec[i]*thee->u[i]*thee->charge[i]);
    }
    energy = energy*hx*hy*hzed/Vpbe_getZmagic(thee->pbe);

    if (extFlag == 1) energy += thee->extQfEnergy;

    return energy;
}

VPRIVATE void Vpmg_splineSelect(int srfm,Vacc *acc,double *gpos,double win,
           double infrad,Vatom *atom,double *force){

 switch (srfm) {
  case VSM_SPLINE :
   Vacc_splineAccGradAtomNorm(acc, gpos, win, infrad, atom, force);
   break;
  case VSM_SPLINE3:
   Vacc_splineAccGradAtomNorm3(acc, gpos, win, infrad, atom, force);
   break;
  case VSM_SPLINE4 :
   Vacc_splineAccGradAtomNorm4(acc, gpos, win, infrad, atom, force);
   break;
  default:
   Vnm_print(2, "Vpmg_dbnbForce: Unknown surface method.\n");
   return;
 }

 return;
}

VPRIVATE void focusFillBound(Vpmg *thee,
                             Vpmg *pmgOLD
                            ) {

    Vpbe *pbe;
    double hxOLD,
           hyOLD,
           hzOLD,
           xminOLD,
           yminOLD,
           zminOLD,
           xmaxOLD,
           ymaxOLD,
           zmaxOLD,
           hxNEW,
           hyNEW,
           hzNEW,
           xminNEW,
           yminNEW,
           zminNEW,
           xmaxNEW,
           ymaxNEW,
           zmaxNEW,
```

```
            x,
            y,
            z,
            dx,
            dy,
            dz,
            ifloat,
            jfloat,
            kfloat,
            uval,
            eps_w,
            T,
            pre1,
            xkappa,
            size,
            *apos,
            charge,
            //pos[3], // gcc: not used
            uvalMin,
            uvalMax,
            *data;
    int nxOLD,
        nyOLD,
        nzOLD,
        nxNEW,
        nyNEW,
        nzNEW,
        i,
        j,
        k,
        ihi,
        ilo,
        jhi,
        jlo,
        khi,
        klo,
        nx,
        ny,
        nz;

    /* Calculate new problem dimensions */
    hxNEW = thee->pmgp->hx;
    hyNEW = thee->pmgp->hy;
    hzNEW = thee->pmgp->hzed;
    nx =  thee->pmgp->nx;
    ny =  thee->pmgp->ny;
    nz =  thee->pmgp->nz;
    nxNEW = thee->pmgp->nx;
    nyNEW = thee->pmgp->ny;
    nzNEW = thee->pmgp->nz;
    xminNEW = thee->pmgp->xcent - ((double)(nxNEW-1)*hxNEW)/2.0;
    xmaxNEW = thee->pmgp->xcent + ((double)(nxNEW-1)*hxNEW)/2.0;
    yminNEW = thee->pmgp->ycent - ((double)(nyNEW-1)*hyNEW)/2.0;
    ymaxNEW = thee->pmgp->ycent + ((double)(nyNEW-1)*hyNEW)/2.0;
    zminNEW = thee->pmgp->zcent - ((double)(nzNEW-1)*hzNEW)/2.0;
    zmaxNEW = thee->pmgp->zcent + ((double)(nzNEW-1)*hzNEW)/2.0;

  if(pmgOLD != VNULL){
   /* Relevant old problem parameters */
   hxOLD = pmgOLD->pmgp->hx;
   hyOLD = pmgOLD->pmgp->hy;
   hzOLD = pmgOLD->pmgp->hzed;
   nxOLD = pmgOLD->pmgp->nx;
   nyOLD = pmgOLD->pmgp->ny;
   nzOLD = pmgOLD->pmgp->nz;
   xminOLD = pmgOLD->pmgp->xcent - ((double)(nxOLD-1)*hxOLD)/2.0;
   xmaxOLD = pmgOLD->pmgp->xcent + ((double)(nxOLD-1)*hxOLD)/2.0;
   yminOLD = pmgOLD->pmgp->ycent - ((double)(nyOLD-1)*hyOLD)/2.0;
   ymaxOLD = pmgOLD->pmgp->ycent + ((double)(nyOLD-1)*hyOLD)/2.0;
   zminOLD = pmgOLD->pmgp->zcent - ((double)(nzOLD-1)*hzOLD)/2.0;
   zmaxOLD = pmgOLD->pmgp->zcent + ((double)(nzOLD-1)*hzOLD)/2.0;

   data = pmgOLD->u;
  }else{
   /* Relevant old problem parameters */
   hxOLD = thee->potMap->hx;
   hyOLD = thee->potMap->hy;
   hzOLD = thee->potMap->hzed;
   nxOLD = thee->potMap->nx;
   nyOLD = thee->potMap->ny;
   nzOLD = thee->potMap->nz;
```

```
  xminOLD = thee->potMap->xmin;
  xmaxOLD = thee->potMap->xmax;
  yminOLD = thee->potMap->ymin;
  ymaxOLD = thee->potMap->ymax;
  zminOLD = thee->potMap->zmin;
  zmaxOLD = thee->potMap->zmax;

  data = thee->potMap->data;
 }
    /* BOUNDARY CONDITION SETUP FOR POINTS OFF OLD MESH:
     * For each "atom" (only one for bcfl=1), we use the following formula to
     * calculate the boundary conditions:
     *    g(x) = \frac{q e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
     *          * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
     *          * 1/d
     * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
     * We only need to evaluate some of these prefactors once:
     *    pre1 = \frac{e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
     * which gives the potential as
     *    g(x) = pre1 * q/d * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
     */
    pbe = thee->pbe;
    eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
    T = Vpbe_getTemperature(pbe);               /* K */
    pre1 = (Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T);

    /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
     * m/A, then we will only need to deal with distances and sizes in
     * Angstroms rather than meters.                                    */
    xkappa = Vpbe_getXkappa(pbe);              /* A^{-1} */
    pre1 = pre1*(1.0e10);
    size = Vpbe_getSoluteRadius(pbe);
    apos = Vpbe_getSoluteCenter(pbe);
    charge = Vunit_ec*Vpbe_getSoluteCharge(pbe);

    /* Check for rounding error */
    if (VABS(xminOLD-xminNEW) < VSMALL) xminNEW = xminOLD;
    if (VABS(xmaxOLD-xmaxNEW) < VSMALL) xmaxNEW = xmaxOLD;
    if (VABS(yminOLD-yminNEW) < VSMALL) yminNEW = yminOLD;
    if (VABS(ymaxOLD-ymaxNEW) < VSMALL) ymaxNEW = ymaxOLD;
    if (VABS(zminOLD-zminNEW) < VSMALL) zminNEW = zminOLD;
    if (VABS(zmaxOLD-zmaxNEW) < VSMALL) zmaxNEW = zmaxOLD;


    /* Sanity check: make sure we're within the old mesh */
    Vnm_print(0, "VPMG::focusFillBound -- New mesh mins = %g, %g, %g\n",
     xminNEW, yminNEW, zminNEW);
    Vnm_print(0, "VPMG::focusFillBound -- New mesh maxs = %g, %g, %g\n",
     xmaxNEW, ymaxNEW, zmaxNEW);
    Vnm_print(0, "VPMG::focusFillBound -- Old mesh mins = %g, %g, %g\n",
     xminOLD, yminOLD, zminOLD);
    Vnm_print(0, "VPMG::focusFillBound -- Old mesh maxs = %g, %g, %g\n",
     xmaxOLD, ymaxOLD, zmaxOLD);

    /* The following is obsolete; we'll substitute analytical boundary
     * condition values when the new mesh falls outside the old */
    if ((xmaxNEW>xmaxOLD) || (ymaxNEW>ymaxOLD) || (zmaxNEW>zmaxOLD) ||
        (xminOLD>xminNEW) || (yminOLD>yminNEW) || (zminOLD>zminNEW)) {

        Vnm_print(2, "Vpmg::focusFillBound -- new mesh not contained in old!\n");
  Vnm_print(2, "Vpmg::focusFillBound -- old mesh min = (%g, %g, %g)\n",
      xminOLD, yminOLD, zminOLD);
        Vnm_print(2, "Vpmg::focusFillBound -- old mesh max = (%g, %g, %g)\n",
      xmaxOLD, ymaxOLD, zmaxOLD);
  Vnm_print(2, "Vpmg::focusFillBound -- new mesh min = (%g, %g, %g)\n",
      xminNEW, yminNEW, zminNEW);
        Vnm_print(2, "Vpmg::focusFillBound -- new mesh max = (%g, %g, %g)\n",
      xmaxNEW, ymaxNEW, zmaxNEW);
  fflush(stderr);
        VASSERT(0);
    }

 uvalMin = VPMGSMALL;
 uvalMax = -VPMGSMALL;

    /* Fill the "i" boundaries (dirichlet) */
```

```
    for (k=0; k<nzNEW; k++) {
        for (j=0; j<nyNEW; j++) {
            /* Low X face */
            x = xminNEW;
            y = yminNEW + j*hyNEW;
            z = zminNEW + k*hzNEW;
            if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&
                (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
                ifloat = (x - xminOLD)/hxOLD;
                jfloat = (y - yminOLD)/hyOLD;
                kfloat = (z - zminOLD)/hzOLD;
                ihi = (int)ceil(ifloat);
                if (ihi > (nxOLD-1)) ihi = nxOLD-1;
                ilo = (int)floor(ifloat);
                if (ilo < 0) ilo = 0;
                jhi = (int)ceil(jfloat);
                if (jhi > (nyOLD-1)) jhi = nyOLD-1;
                jlo = (int)floor(jfloat);
                if (jlo < 0) jlo = 0;
                khi = (int)ceil(kfloat);
                if (khi > (nzOLD-1)) khi = nzOLD-1;
                klo = (int)floor(kfloat);
                if (klo < 0) klo = 0;
                dx = ifloat - (double)(ilo);
                dy = jfloat - (double)(jlo);
                dz = kfloat - (double)(klo);
                nx = nxOLD; ny = nyOLD; nz = nzOLD;
                uval =  dx*dy*dz*(data[IJK(ihi,jhi,khi)])
    + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
    + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
    + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
    + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
    + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
    + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
    + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
                nx = nxNEW; ny = nyNEW; nz = nzNEW;
            } else {
    Vnm_print(2, "focusFillBound (%s, %d):  Off old mesh at %g, %g \
        %g!\n", __FILE__, __LINE__, x, y, z);
    Vnm_print(2, "focusFillBound (%s, %d):  old mesh lower corner at \
        %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
                Vnm_print(2, "focusFillBound (%s, %d):  old mesh upper corner at \
        %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
    VASSERT(0);
            }
            nx = nxNEW; ny = nyNEW; nz = nzNEW;
            thee->gxcf[IJKx(j,k,0)] = uval;
   if(uval < uvalMin) uvalMin = uval;
   if(uval > uvalMax) uvalMax = uval;

            /* High X face */
            x = xmaxNEW;
            if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&
                (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
                ifloat = (x - xminOLD)/hxOLD;
                jfloat = (y - yminOLD)/hyOLD;
                kfloat = (z - zminOLD)/hzOLD;
                ihi = (int)ceil(ifloat);
                if (ihi > (nxOLD-1)) ihi = nxOLD-1;
                ilo = (int)floor(ifloat);
                if (ilo < 0) ilo = 0;
                jhi = (int)ceil(jfloat);
                if (jhi > (nyOLD-1)) jhi = nyOLD-1;
                jlo = (int)floor(jfloat);
                if (jlo < 0) jlo = 0;
                khi = (int)ceil(kfloat);
                if (khi > (nzOLD-1)) khi = nzOLD-1;
                klo = (int)floor(kfloat);
                if (klo < 0) klo = 0;
                dx = ifloat - (double)(ilo);
                dy = jfloat - (double)(jlo);
                dz = kfloat - (double)(klo);
                nx = nxOLD; ny = nyOLD; nz = nzOLD;
                uval =  dx*dy*dz*(data[IJK(ihi,jhi,khi)])
    + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
    + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
```

```
    + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
    + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
    + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
    + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
    + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
                nx = nxNEW; ny = nyNEW; nz = nzNEW;
            } else {
    Vnm_print(2, "focusFillBound (%s, %d):  Off old mesh at %g, %g \
        %g!\n", __FILE__, __LINE__, x, y, z);
    Vnm_print(2, "focusFillBound (%s, %d):  old mesh lower corner at \
        %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
                Vnm_print(2, "focusFillBound (%s, %d):  old mesh upper corner at \
        %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
    VASSERT(0);
            }
            nx = nxNEW; ny = nyNEW; nz = nzNEW;
            thee->gxcf[IJKx(j,k,1)] = uval;
   if(uval < uvalMin) uvalMin = uval;
   if(uval > uvalMax) uvalMax = uval;

            /* Zero Neumann conditions */
            nx = nxNEW; ny = nyNEW; nz = nzNEW;
            thee->gxcf[IJKx(j,k,2)] = 0.0;
            nx = nxNEW; ny = nyNEW; nz = nzNEW;
            thee->gxcf[IJKx(j,k,3)] = 0.0;
        }
    }

    /* Fill the "j" boundaries (dirichlet) */
    for (k=0; k<nzNEW; k++) {
        for (i=0; i<nxNEW; i++) {
            /* Low Y face */
            x = xminNEW + i*hxNEW;
            y = yminNEW;
            z = zminNEW + k*hzNEW;
            if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&
                (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
                ifloat = (x - xminOLD)/hxOLD;
                jfloat = (y - yminOLD)/hyOLD;
                kfloat = (z - zminOLD)/hzOLD;
                ihi = (int)ceil(ifloat);
                if (ihi > (nxOLD-1)) ihi = nxOLD-1;
                ilo = (int)floor(ifloat);
                if (ilo < 0) ilo = 0;
                jhi = (int)ceil(jfloat);
                if (jhi > (nyOLD-1)) jhi = nyOLD-1;
                jlo = (int)floor(jfloat);
                if (jlo < 0) jlo = 0;
                khi = (int)ceil(kfloat);
                if (khi > (nzOLD-1)) khi = nzOLD-1;
                klo = (int)floor(kfloat);
                if (klo < 0) klo = 0;
                dx = ifloat - (double)(ilo);
                dy = jfloat - (double)(jlo);
                dz = kfloat - (double)(klo);
                nx = nxOLD; ny = nyOLD; nz = nzOLD;
                uval =  dx*dy*dz*(data[IJK(ihi,jhi,khi)])
    + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
    + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
    + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
    + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
    + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
    + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
    + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
                nx = nxNEW; ny = nyNEW; nz = nzNEW;
            } else {
    Vnm_print(2, "focusFillBound (%s, %d):  Off old mesh at %g, %g \
        %g!\n", __FILE__, __LINE__, x, y, z);
    Vnm_print(2, "focusFillBound (%s, %d):  old mesh lower corner at \
        %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
                Vnm_print(2, "focusFillBound (%s, %d):  old mesh upper corner at \
        %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
    VASSERT(0);
            }
            nx = nxNEW; ny = nyNEW; nz = nzNEW;
            thee->gycf[IJKy(i,k,0)] = uval;
   if(uval < uvalMin) uvalMin = uval;
```

```
  if(uval > uvalMax) uvalMax = uval;

            /* High Y face */
            y = ymaxNEW;
            if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&
                (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
                ifloat = (x - xminOLD)/hxOLD;
                jfloat = (y - yminOLD)/hyOLD;
                kfloat = (z - zminOLD)/hzOLD;
                ihi = (int)ceil(ifloat);
                if (ihi > (nxOLD-1)) ihi = nxOLD-1;
                ilo = (int)floor(ifloat);
                if (ilo < 0) ilo = 0;
                jhi = (int)ceil(jfloat);
                if (jhi > (nyOLD-1)) jhi = nyOLD-1;
                jlo = (int)floor(jfloat);
                if (jlo < 0) jlo = 0;
                khi = (int)ceil(kfloat);
                if (khi > (nzOLD-1)) khi = nzOLD-1;
                klo = (int)floor(kfloat);
                if (klo < 0) klo = 0;
                dx = ifloat - (double)(ilo);
                dy = jfloat - (double)(jlo);
                dz = kfloat - (double)(klo);
                nx = nxOLD; ny = nyOLD; nz = nzOLD;
                uval =  dx*dy*dz*(data[IJK(ihi,jhi,khi)])
    + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
    + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
    + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
    + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
    + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
    + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
    + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
                nx = nxNEW; ny = nyNEW; nz = nzNEW;
            } else {
    Vnm_print(2, "focusFillBound (%s, %d):  Off old mesh at %g, %g \
        %g!\n", __FILE__, __LINE__, x, y, z);
    Vnm_print(2, "focusFillBound (%s, %d):  old mesh lower corner at \
        %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
                Vnm_print(2, "focusFillBound (%s, %d):  old mesh upper corner at \
        %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
    VASSERT(0);
            }
            nx = nxNEW; ny = nyNEW; nz = nzNEW;
            thee->gycf[IJKy(i,k,1)] = uval;
  if(uval < uvalMin) uvalMin = uval;
  if(uval > uvalMax) uvalMax = uval;

            /* Zero Neumann conditions */
            nx = nxNEW; ny = nyNEW; nz = nzNEW;
            thee->gycf[IJKy(i,k,2)] = 0.0;
            nx = nxNEW; ny = nyNEW; nz = nzNEW;
            thee->gycf[IJKy(i,k,3)] = 0.0;
        }
    }

    /* Fill the "k" boundaries (dirichlet) */
    for (j=0; j<nyNEW; j++) {
        for (i=0; i<nxNEW; i++) {
            /* Low Z face */
            x = xminNEW + i*hxNEW;
            y = yminNEW + j*hyNEW;
            z = zminNEW;
            if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&
                (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
                ifloat = (x - xminOLD)/hxOLD;
                jfloat = (y - yminOLD)/hyOLD;
                kfloat = (z - zminOLD)/hzOLD;
                ihi = (int)ceil(ifloat);
                if (ihi > (nxOLD-1)) ihi = nxOLD-1;
                ilo = (int)floor(ifloat);
                if (ilo < 0) ilo = 0;
                jhi = (int)ceil(jfloat);
                if (jhi > (nyOLD-1)) jhi = nyOLD-1;
                jlo = (int)floor(jfloat);
                if (jlo < 0) jlo = 0;
```

```
                khi = (int)ceil(kfloat);
                if (khi > (nzOLD-1)) khi = nzOLD-1;
                klo = (int)floor(kfloat);
                if (klo < 0) klo = 0;
                dx = ifloat - (double)(ilo);
                dy = jfloat - (double)(jlo);
                dz = kfloat - (double)(klo);
                nx = nxOLD; ny = nyOLD; nz = nzOLD;
                uval =  dx*dy*dz*(data[IJK(ihi,jhi,khi)])
    + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
    + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
    + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
    + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
    + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
    + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
    + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
                nx = nxNEW; ny = nyNEW; nz = nzNEW;
            } else {
    Vnm_print(2, "focusFillBound (%s, %d):  Off old mesh at %g, %g \
        %g!\n", __FILE__, __LINE__, x, y, z);
    Vnm_print(2, "focusFillBound (%s, %d):  old mesh lower corner at \
        %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
                Vnm_print(2, "focusFillBound (%s, %d):  old mesh upper corner
 at \
        %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
    VASSERT(0);
            }
            nx = nxNEW; ny = nyNEW; nz = nzNEW;
            thee->gzcf[IJKz(i,j,0)] = uval;
   if(uval < uvalMin) uvalMin = uval;
   if(uval > uvalMax) uvalMax = uval;

            /* High Z face */
            z = zmaxNEW;
            if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (
zminOLD-VSMALL)) &&
                (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (
zmaxOLD+VSMALL))) {
                ifloat = (x - xminOLD)/hxOLD;
                jfloat = (y - yminOLD)/hyOLD;
                kfloat = (z - zminOLD)/hzOLD;
                ihi = (int)ceil(ifloat);
                if (ihi > (nxOLD-1)) ihi = nxOLD-1;
                ilo = (int)floor(ifloat);
                if (ilo < 0) ilo = 0;
                jhi = (int)ceil(jfloat);
                if (jhi > (nyOLD-1)) jhi = nyOLD-1;
                jlo = (int)floor(jfloat);
                if (jlo < 0) jlo = 0;
                khi = (int)ceil(kfloat);
                if (khi > (nzOLD-1)) khi = nzOLD-1;
                klo = (int)floor(kfloat);
                if (klo < 0) klo = 0;
                dx = ifloat - (double)(ilo);
                dy = jfloat - (double)(jlo);
                dz = kfloat - (double)(klo);
                nx = nxOLD; ny = nyOLD; nz = nzOLD;
                uval =  dx*dy*dz*(data[IJK(ihi,jhi,khi)])
    + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
    + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
    + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
    + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
    + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
    + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
    + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
                nx = nxNEW; ny = nyNEW; nz = nzNEW;
            } else {
    Vnm_print(2, "focusFillBound (%s, %d):  Off old mesh at %g, %g \
        %g!\n", __FILE__, __LINE__, x, y, z);
    Vnm_print(2, "focusFillBound (%s, %d):  old mesh lower corner at \
        %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
                Vnm_print(2, "focusFillBound (%s, %d):  old mesh upper corner
 at \
        %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
    VASSERT(0);
            }
            nx = nxNEW; ny = nyNEW; nz = nzNEW;
            thee->gzcf[IJKz(i,j,1)] = uval;
   if(uval < uvalMin) uvalMin = uval;
   if(uval > uvalMax) uvalMax = uval;

```

```
            /* Zero Neumann conditions */
            nx = nxNEW; ny = nyNEW; nz = nzNEW;
            thee->gzcf[IJKz(i,j,2)] = 0.0;
            nx = nxNEW; ny = nyNEW; nz = nzNEW;
            thee->gzcf[IJKz(i,j,3)] = 0.0;
        }
    }

    VWARN_MSG0(
        uvalMin >= SINH_MIN && uvalMax <= SINH_MAX,
        "Unusually large potential values\n"
        "    detected on the focusing boundary!\n"
        "    Convergence not guaranteed for NPBE/NRPBE calculations!"
        );
}

VPRIVATE void extEnergy(Vpmg *thee, Vpmg *pmgOLD, PBEparm_calcEnergy extFlag,
      double partMin[3], double partMax[3], int bflags[6]) {

    Vatom *atom;
    double hxNEW, hyNEW, hzNEW;
    double lowerCorner[3], upperCorner[3];
    int nxNEW, nyNEW, nzNEW;
    int nxOLD, nyOLD, nzOLD;
    int i,j,k;
    double xmin, xmax, ymin, ymax, zmin, zmax;
    double hxOLD, hyOLD, hzOLD;
    double xval, yval, zval;
    double x,y,z;
    int nx, ny, nz;

    /* Set the new external energy contribution to zero.  Any external
     * contributions from higher levels will be included in the appropriate
     * energy function call. */
    thee->extQmEnergy = 0;
    thee->extQfEnergy = 0;
    thee->extDiEnergy = 0;

    /* New problem dimensions */
    hxNEW = thee->pmgp->hx;
    hyNEW = thee->pmgp->hy;
    hzNEW = thee->pmgp->hzed;
    nxNEW = thee->pmgp->nx;
    nyNEW = thee->pmgp->ny;
    nzNEW = thee->pmgp->nz;
    lowerCorner[0] = thee->pmgp->xcent - ((double)(nxNEW-1)*hxNEW)/2.0;
    upperCorner[0] = thee->pmgp->xcent + ((double)(nxNEW-1)*hxNEW)/2.0;
    lowerCorner[1] = thee->pmgp->ycent - ((double)(nyNEW-1)*hyNEW)/2.0;
    upperCorner[1] = thee->pmgp->ycent + ((double)(nyNEW-1)*hyNEW)/2.0;
    lowerCorner[2] = thee->pmgp->zcent - ((double)(nzNEW-1)*hzNEW)/2.0;
    upperCorner[2] = thee->pmgp->zcent + ((double)(nzNEW-1)*hzNEW)/2.0;

    Vnm_print(0, "VPMG::extEnergy:  energy flag = %d\n", extFlag);

    /* Old problem dimensions */
    nxOLD = pmgOLD->pmgp->nx;
    nyOLD = pmgOLD->pmgp->ny;
    nzOLD = pmgOLD->pmgp->nz;

    /* Create a partition based on the new problem dimensions */
    /* Vnm_print(1, "DEBUG (%s, %d):  extEnergy calling Vpmg_setPart for old PMG.\n",
     __FILE__, __LINE__); */
    Vpmg_setPart(pmgOLD, lowerCorner, upperCorner, bflags);


    Vnm_print(0,"VPMG::extEnergy:   Finding extEnergy dimensions...\n");
    Vnm_print(0,"VPMG::extEnergy    Disj part lower corner = (%g, %g, %g)\n",
     partMin[0], partMin[1], partMin[2]);
    Vnm_print(0,"VPMG::extEnergy    Disj part upper corner = (%g, %g, %g)\n",
     partMax[0], partMax[1], partMax[2]);

    /* Find the old dimensions */

```

```
    hxOLD = pmgOLD->pmgp->hx;
    hyOLD = pmgOLD->pmgp->hy;
    hzOLD = pmgOLD->pmgp->hzed;
    xmin =  pmgOLD->pmgp->xcent - 0.5*hxOLD*(nxOLD-1);
    ymin =  pmgOLD->pmgp->ycent - 0.5*hyOLD*(nyOLD-1);
    zmin =  pmgOLD->pmgp->zcent - 0.5*hzOLD*(nzOLD-1);
    xmax =  xmin+hxOLD*(nxOLD-1);
    ymax =  ymin+hyOLD*(nyOLD-1);
    zmax =  zmin+hzOLD*(nzOLD-1);

    Vnm_print(0,"VPMG::extEnergy    Old lower corner = (%g, %g, %g)\n",
     xmin, ymin, zmin);
    Vnm_print(0,"VPMG::extEnergy    Old upper corner = (%g, %g, %g)\n",
     xmax, ymax, zmax);

    /* Flip the partition, but do not include any points that will
  be included by another processor */

    nx = nxOLD;
    ny = nyOLD;
    nz = nzOLD;

    for(i=0; i<nx; i++) {
        xval = 1;
        x = i*hxOLD + xmin;
        if (x < partMin[0] && bflags[VAPBS_LEFT] == 1) xval = 0;
        else if (x > partMax[0] && bflags[VAPBS_RIGHT] == 1) xval = 0;

        for(j=0; j<ny; j++) {
            yval = 1;
            y = j*hyOLD + ymin;
            if (y < partMin[1] && bflags[VAPBS_BACK] == 1) yval = 0;
            else if (y > partMax[1] && bflags[VAPBS_FRONT] == 1) yval = 0;

            for(k=0; k<nz; k++) {
                zval = 1;
                z = k*hzOLD + zmin;
                if (z < partMin[2] && bflags[VAPBS_DOWN] == 1) zval = 0;
                else if (z > partMax[2] && bflags[VAPBS_UP] == 1) zval = 0;

                if (pmgOLD->pvec[IJK(i,j,k)] > VSMALL) pmgOLD->pvec[IJK(i,j,k)] = 1.0;
                pmgOLD->pvec[IJK(i,j,k)] = (1 - (pmgOLD->pvec[IJK(i,j,k)])) * (xval*yval*zval);
            }
        }
    }

    for (i=0; i<Valist_getNumberAtoms(thee->pbe->alist); i++) {
        xval=1;
        yval=1;
        zval=1;
        atom = Valist_getAtom(thee->pbe->alist, i);
        x = atom->position[0];
        y = atom->position[1];
        z = atom->position[2];
        if (x < partMin[0] && bflags[VAPBS_LEFT] == 1) xval = 0;
        else if (x > partMax[0] && bflags[VAPBS_RIGHT] == 1) xval = 0;
        if (y < partMin[1] && bflags[VAPBS_BACK] == 1) yval = 0;
        else if (y > partMax[1] && bflags[VAPBS_FRONT] == 1) yval = 0;
        if (z < partMin[2] && bflags[VAPBS_DOWN] == 1) zval = 0;
        else if (z > partMax[2] && bflags[VAPBS_UP] == 1) zval = 0;
        if (atom->partID > VSMALL) atom->partID = 1.0;
        atom->partID = (1 - atom->partID) * (xval*yval*zval);
    }

    /* Now calculate the energy on inverted subset of the domain */
    thee->extQmEnergy = Vpmg_qmEnergy(pmgOLD, 1);
    Vnm_print(0, "VPMG::extEnergy: extQmEnergy = %g kT\n", thee->extQmEnergy);
    thee->extQfEnergy = Vpmg_qfEnergy(pmgOLD, 1);
    Vnm_print(0, "VPMG::extEnergy: extQfEnergy = %g kT\n", thee->extQfEnergy);
    thee->extDiEnergy = Vpmg_dielEnergy(pmgOLD, 1);
```

```
    Vnm_print(0, "VPMG::extEnergy: extDiEnergy = %g kT\n", thee->extDiEnergy
);
    Vpmg_unsetPart(pmgOLD);
}

VPRIVATE double bcfl1sp(double size, double *apos, double charge,
      double xkappa, double pre1, double *pos) {

    double dist, val;

    dist = VSQRT(VSQR(pos[0]-apos[0]) + VSQR(pos[1]-apos[1])
     + VSQR(pos[2]-apos[2]));
    if (xkappa > VSMALL) {
        val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
  / (1+xkappa*size);
    } else {
        val = pre1*(charge/dist);
    }

    return val;
}

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) {

    int i, j, k;
    double dist, val;
    double gpos[3];

    /* the "i" boundaries (dirichlet) */
    for (k=0; k<nz; k++) {
        gpos[2] = zf[k];
        for (j=0; j<ny; j++) {
            gpos[1] = yf[j];
            gpos[0] = xf[0];
            dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
       + VSQR(gpos[2]-apos[2]));
            if (xkappa > VSMALL) {
                val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
    / (1+xkappa*size);
            } else {
                val = pre1*(charge/dist);
            }
            gxcf[IJKx(j,k,0)] += val;
            gpos[0] = xf[nx-1];
            dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
       + VSQR(gpos[2]-apos[2]));
            if (xkappa > VSMALL) {
                val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
    / (1+xkappa*size);
            } else {
                val = pre1*(charge/dist);
            }
            gxcf[IJKx(j,k,1)] += val;
        }
    }

    /* the "j" boundaries (dirichlet) */
    for (k=0; k<nz; k++) {
        gpos[2] = zf[k];
        for (i=0; i<nx; i++) {
            gpos[0] = xf[i];
            gpos[1] = yf[0];
            dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
       + VSQR(gpos[2]-apos[2]));
            if (xkappa > VSMALL) {
                val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
    / (1+xkappa*size);
            } else {
                val = pre1*(charge/dist);
            }
            gycf[IJKy(i,k,0)] += val;
            gpos[1] = yf[ny-1];
            dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
       + VSQR(gpos[2]-apos[2]));
            if (xkappa > VSMALL) {
                val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
    / (1+xkappa*size);
            } else {
                val = pre1*(charge/dist);
```

```
            }
            gycf[IJKy(i,k,1)] += val;
        }
    }

    /* the "k" boundaries (dirichlet) */
    for (j=0; j<ny; j++) {
        gpos[1] = yf[j];
        for (i=0; i<nx; i++) {
            gpos[0] = xf[i];
            gpos[2] = zf[0];
            dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
       + VSQR(gpos[2]-apos[2]));
            if (xkappa > VSMALL) {
                val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
    / (1+xkappa*size);
            } else {
                val = pre1*(charge/dist);
            }
            gzcf[IJKz(i,j,0)] += val;
            gpos[2] = zf[nz-1];
            dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
       + VSQR(gpos[2]-apos[2]));
            if (xkappa > VSMALL) {
                val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
    / (1+xkappa*size);
            } else {
                val = pre1*(charge/dist);
            }
            gzcf[IJKz(i,j,1)] += val;
        }
    }
}

VPRIVATE void bcfl2(double size, double *apos,
                    double charge, double *dipole, double *quad,
                    double xkappa, double eps_p, double eps_w, double T,
                    double *gxcf, double *gycf, double *gzcf,
                    double *xf, double *yf, double *zf,
                    int nx, int ny, int nz) {

    int i, j, k;
    double val;
    double gpos[3],tensor[3];
    double ux,uy,uz,xr,yr,zr;
    double qxx,qxy,qxz,qyx,qyy,qyz,qzx,qzy,qzz;
    double dist, pre;

    VASSERT(dipole != VNULL);
    ux = dipole[0];
    uy = dipole[1];
    uz = dipole[2];
    if (quad != VNULL) {
  /* The factor of 1/3 results from using a
   traceless quadrupole definition. See, for example,
   "The Theory of Intermolecular Forces" by A.J. Stone,
   Chapter 3. */
  qxx = quad[0] / 3.0;
  qxy = quad[1] / 3.0;
  qxz = quad[2] / 3.0;
  qyx = quad[3] / 3.0;
  qyy = quad[4] / 3.0;
  qyz = quad[5] / 3.0;
  qzx = quad[6] / 3.0;
  qzy = quad[7] / 3.0;
  qzz = quad[8] / 3.0;
    } else {
  qxx = 0.0;
  qxy = 0.0;
  qxz = 0.0;
  qyx = 0.0;
  qyy = 0.0;
  qyz = 0.0;
  qzx = 0.0;
  qzy = 0.0;
  qzz = 0.0;
    }

    pre = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*
Vunit_kb*T);
    pre = pre*(1.0e10);
```

```
    /* the "i" boundaries (dirichlet) */
    for (k=0; k<nz; k++) {
        gpos[2] = zf[k];
        for (j=0; j<ny; j++) {
            gpos[1] = yf[j];
            gpos[0] = xf[0];
            xr = gpos[0] - apos[0];
            yr = gpos[1] - apos[1];
            zr = gpos[2] - apos[2];
            dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
            multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
            val = pre*charge*tensor[0];
            val -= pre*ux*xr*tensor[1];
            val -= pre*uy*yr*tensor[1];
            val -= pre*uz*zr*tensor[1];
            val += pre*qxx*xr*xr*tensor[2];
            val += pre*qyy*yr*yr*tensor[2];
            val += pre*qzz*zr*zr*tensor[2];
            val += pre*2.0*qxy*xr*yr*tensor[2];
            val += pre*2.0*qxz*xr*zr*tensor[2];
            val += pre*2.0*qyz*yr*zr*tensor[2];
            gxcf[IJKx(j,k,0)] += val;

            gpos[0] = xf[nx-1];
            xr = gpos[0] - apos[0];
            dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
            multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
            val = pre*charge*tensor[0];
            val -= pre*ux*xr*tensor[1];
            val -= pre*uy*yr*tensor[1];
            val -= pre*uz*zr*tensor[1];
            val += pre*qxx*xr*xr*tensor[2];
            val += pre*qyy*yr*yr*tensor[2];
            val += pre*qzz*zr*zr*tensor[2];
            val += pre*2.0*qxy*xr*yr*tensor[2];
            val += pre*2.0*qxz*xr*zr*tensor[2];
            val += pre*2.0*qyz*yr*zr*tensor[2];
            gxcf[IJKx(j,k,1)] += val;
        }
    }

    /* the "j" boundaries (dirichlet) */
    for (k=0; k<nz; k++) {
        gpos[2] = zf[k];
        for (i=0; i<nx; i++) {
            gpos[0] = xf[i];
            gpos[1] = yf[0];
            xr = gpos[0] - apos[0];
            yr = gpos[1] - apos[1];
            zr = gpos[2] - apos[2];
            dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
            multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
            val = pre*charge*tensor[0];
            val -= pre*ux*xr*tensor[1];
            val -= pre*uy*yr*tensor[1];
            val -= pre*uz*zr*tensor[1];
            val += pre*qxx*xr*xr*tensor[2];
            val += pre*qyy*yr*yr*tensor[2];
            val += pre*qzz*zr*zr*tensor[2];
            val += pre*2.0*qxy*xr*yr*tensor[2];
            val += pre*2.0*qxz*xr*zr*tensor[2];
            val += pre*2.0*qyz*yr*zr*tensor[2];
            gycf[IJKy(i,k,0)] += val;

            gpos[1] = yf[ny-1];
            yr = gpos[1] - apos[1];
            dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
            multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
            val = pre*charge*tensor[0];
            val -= pre*ux*xr*tensor[1];
            val -= pre*uy*yr*tensor[1];
            val -= pre*uz*zr*tensor[1];
            val += pre*qxx*xr*xr*tensor[2];
            val += pre*qyy*yr*yr*tensor[2];
            val += pre*qzz*zr*zr*tensor[2];
            val += pre*2.0*qxy*xr*yr*tensor[2];
            val += pre*2.0*qxz*xr*zr*tensor[2];
            val += pre*2.0*qyz*yr*zr*tensor[2];
            gycf[IJKy(i,k,1)] += val;
        }
```

```
    }

    /* the "k" boundaries (dirichlet) */
    for (j=0; j<ny; j++) {
        gpos[1] = yf[j];
        for (i=0; i<nx; i++) {
            gpos[0] = xf[i];
            gpos[2] = zf[0];
            xr = gpos[0] - apos[0];
            yr = gpos[1] - apos[1];
            zr = gpos[2] - apos[2];
            dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
            multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
            val = pre*charge*tensor[0];
            val -= pre*ux*xr*tensor[1];
            val -= pre*uy*yr*tensor[1];
            val -= pre*uz*zr*tensor[1];
            val += pre*qxx*xr*xr*tensor[2];
            val += pre*qyy*yr*yr*tensor[2];
            val += pre*qzz*zr*zr*tensor[2];
            val += pre*2.0*qxy*xr*yr*tensor[2];
            val += pre*2.0*qxz*xr*zr*tensor[2];
            val += pre*2.0*qyz*yr*zr*tensor[2];
            gzcf[IJKz(i,j,0)] += val;

            gpos[2] = zf[nz-1];
            zr = gpos[2] - apos[2];
            dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
            multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
            val = pre*charge*tensor[0];
            val -= pre*ux*xr*tensor[1];
            val -= pre*uy*yr*tensor[1];
            val -= pre*uz*zr*tensor[1];
            val += pre*qxx*xr*xr*tensor[2];
            val += pre*qyy*yr*yr*tensor[2];
            val += pre*qzz*zr*zr*tensor[2];
            val += pre*2.0*qxy*xr*yr*tensor[2];
            val += pre*2.0*qxz*xr*zr*tensor[2];
            val += pre*2.0*qyz*yr*zr*tensor[2];
            gzcf[IJKz(i,j,1)] += val;
        }
    }
}

VPRIVATE void bcCalcOrig(Vpmg *thee) {

    int nx, ny, nz;
    double size, *position, charge, xkappa, eps_w, T, pre1;
    double *dipole, *quadrupole, debye, eps_p;
    double xr,yr,zr,qave,*apos;
    double sdhcharge, sdhdipole[3], traced[9], sdhquadrupole[9];
    int i, j, k, iatom;
    Vpbe *pbe;
    Vatom *atom;
    Valist *alist;

    pbe = thee->pbe;
    alist = thee->pbe->alist;
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;

    /* Zero out the boundaries */
    /* the "i" boundaries (dirichlet) */
    for (k=0; k<nz; k++) {
        for (j=0; j<ny; j++) {
            thee->gxcf[IJKx(j,k,0)] = 0.0;
            thee->gxcf[IJKx(j,k,1)] = 0.0;
            thee->gxcf[IJKx(j,k,2)] = 0.0;
            thee->gxcf[IJKx(j,k,3)] = 0.0;
        }
    }

    /* the "j" boundaries (dirichlet) */
    for (k=0; k<nz; k++) {
        for (i=0; i<nx; i++) {
            thee->gycf[IJKy(i,k,0)] = 0.0;
            thee->gycf[IJKy(i,k,1)] = 0.0;
            thee->gycf[IJKy(i,k,2)] = 0.0;
            thee->gycf[IJKy(i,k,3)] = 0.0;
        }
```

```
    }

    /* the "k" boundaries (dirichlet) */
    for (j=0; j<ny; j++) {
        for (i=0; i<nx; i++) {
            thee->gzcf[IJKz(i,j,0)] = 0.0;
            thee->gzcf[IJKz(i,j,1)] = 0.0;
            thee->gzcf[IJKz(i,j,2)] = 0.0;
            thee->gzcf[IJKz(i,j,3)] = 0.0;
        }
    }

    /* For each "atom" (only one for bcfl=1), we use the following formula to
 * calculate the boundary conditions:
 *    g(x) = \frac{q e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
 *          * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
 *          * 1/d
 * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
 * We only need to evaluate some of these prefactors once:
 *    pre1 = \frac{e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
 * which gives the potential as
 *    g(x) = pre1 * q/d * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
 */
    eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
    eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
    T = Vpbe_getTemperature(pbe);               /* K */
    pre1 = (Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T);

    /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
  * m/A, then we will only need to deal with distances and sizes in
  * Angstroms rather than meters.                                    */
    xkappa = Vpbe_getXkappa(pbe);              /* A^{-1} */
    pre1 = pre1*(1.0e10);

    switch (thee->pmgp->bcfl) {
        /*  If we have zero boundary conditions, we're done */
        case BCFL_ZERO:
            return;

   /*  For single DH sphere BC's, we only have one "atom" to deal with;
   *  get its information and */
        case BCFL_SDH:
            size = Vpbe_getSoluteRadius(pbe);
            position = Vpbe_getSoluteCenter(pbe);

            /*
    For AMOEBA SDH boundary conditions, we need to find the
    total monopole, dipole and traceless quadrupole moments
    of either the permanent multipoles, induced dipoles or
    non-local induced dipoles.
    */

            sdhcharge = 0.0;
            for (i=0; i<3; i++) sdhdipole[i] = 0.0;
   for (i=0; i<9; i++) sdhquadrupole[i] = 0.0;

   for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
    atom = Valist_getAtom(alist, iatom);
    apos = Vatom_getPosition(atom);
    xr = apos[0] - position[0];
    yr = apos[1] - position[1];
    zr = apos[2] - position[2];
    switch (thee->chargeSrc) {
     case VCM_CHARGE:
      charge = Vatom_getCharge(atom);
      sdhcharge += charge;
      sdhdipole[0] += xr * charge;
      sdhdipole[1] += yr * charge;
      sdhdipole[2] += zr * charge;
      traced[0] = xr*xr*charge;
      traced[1] = xr*yr*charge;
      traced[2] = xr*zr*charge;
      traced[3] = yr*xr*charge;
      traced[4] = yr*yr*charge;
      traced[5] = yr*zr*charge;
```

```
      traced[6] = zr*xr*charge;
      traced[7] = zr*yr*charge;
      traced[8] = zr*zr*charge;
      qave = (traced[0] + traced[4] + traced[8]) / 3.0;
      sdhquadrupole[0] += 1.5*(traced[0] - qave);
      sdhquadrupole[1] += 1.5*(traced[1]);
      sdhquadrupole[2] += 1.5*(traced[2]);
      sdhquadrupole[3] += 1.5*(traced[3]);
      sdhquadrupole[4] += 1.5*(traced[4] - qave);
      sdhquadrupole[5] += 1.5*(traced[5]);
      sdhquadrupole[6] += 1.5*(traced[6]);
      sdhquadrupole[7] += 1.5*(traced[7]);
      sdhquadrupole[8] += 1.5*(traced[8] - qave);
#if defined(WITH_TINKER)
     case VCM_PERMANENT:
      charge = Vatom_getCharge(atom);
      dipole = Vatom_getDipole(atom);
      quadrupole = Vatom_getQuadrupole(atom);
      sdhcharge += charge;
      sdhdipole[0] += xr * charge;
      sdhdipole[1] += yr * charge;
      sdhdipole[2] += zr * charge;
      traced[0] = xr*xr*charge;
      traced[1] = xr*yr*charge;
      traced[2] = xr*zr*charge;
      traced[3] = yr*xr*charge;
      traced[4] = yr*yr*charge;
      traced[5] = yr*zr*charge;
      traced[6] = zr*xr*charge;
      traced[7] = zr*yr*charge;
      traced[8] = zr*zr*charge;
      sdhdipole[0] += dipole[0];
      sdhdipole[1] += dipole[1];
      sdhdipole[2] += dipole[2];
      traced[0] += 2.0*xr*dipole[0];
      traced[1] += xr*dipole[1] + yr*dipole[0];
      traced[2] += xr*dipole[2] + zr*dipole[0];
      traced[3] += yr*dipole[0] + xr*dipole[1];
      traced[4] += 2.0*yr*dipole[1];
      traced[5] += yr*dipole[2] + zr*dipole[1];
      traced[6] += zr*dipole[0] + xr*dipole[2];
      traced[7] += zr*dipole[1] + yr*dipole[2];
      traced[8] += 2.0*zr*dipole[2];
      qave = (traced[0] + traced[4] + traced[8]) / 3.0;
      sdhquadrupole[0] += 1.5*(traced[0] - qave);
      sdhquadrupole[1] += 1.5*(traced[1]);
      sdhquadrupole[2] += 1.5*(traced[2]);
      sdhquadrupole[3] += 1.5*(traced[3]);
      sdhquadrupole[4] += 1.5*(traced[4] - qave);
      sdhquadrupole[5] += 1.5*(traced[5]);
      sdhquadrupole[6] += 1.5*(traced[6]);
      sdhquadrupole[7] += 1.5*(traced[7]);
      sdhquadrupole[8] += 1.5*(traced[8] - qave);
      sdhquadrupole[0] += quadrupole[0];
      sdhquadrupole[1] += quadrupole[1];
      sdhquadrupole[2] += quadrupole[2];
      sdhquadrupole[3] += quadrupole[3];
      sdhquadrupole[4] += quadrupole[4];
      sdhquadrupole[5] += quadrupole[5];
      sdhquadrupole[6] += quadrupole[6];
      sdhquadrupole[7] += quadrupole[7];
      sdhquadrupole[8] += quadrupole[8];
     case VCM_INDUCED:
      dipole = Vatom_getInducedDipole(atom);
      sdhdipole[0] += dipole[0];
      sdhdipole[1] += dipole[1];
      sdhdipole[2] += dipole[2];
      traced[0] = 2.0*xr*dipole[0];
      traced[1] = xr*dipole[1] + yr*dipole[0];
      traced[2] = xr*dipole[2] + zr*dipole[0];
      traced[3] = yr*dipole[0] + xr*dipole[1];
      traced[4] = 2.0*yr*dipole[1];
      traced[5] = yr*dipole[2] + zr*dipole[1];
      traced[6] = zr*dipole[0] + xr*dipole[2];
      traced[7] = zr*dipole[1] + yr*dipole[2];
      traced[8] = 2.0*zr*dipole[2];
      qave = (traced[0] + traced[4] + traced[8]) / 3.0;
      sdhquadrupole[0] += 1.5*(traced[0] - qave);
      sdhquadrupole[1] += 1.5*(traced[1]);
      sdhquadrupole[2] += 1.5*(traced[2]);
      sdhquadrupole[3] += 1.5*(traced[3]);
```

```
      sdhquadrupole[4] += 1.5*(traced[4] - qave);
      sdhquadrupole[5] += 1.5*(traced[5]);
      sdhquadrupole[6] += 1.5*(traced[6]);
      sdhquadrupole[7] += 1.5*(traced[7]);
      sdhquadrupole[8] += 1.5*(traced[8] - qave);
     case VCM_NLINDUCED:
      dipole = Vatom_getNLInducedDipole(atom);
      sdhdipole[0] += dipole[0];
      sdhdipole[1] += dipole[1];
      sdhdipole[2] += dipole[2];
      traced[0] = 2.0*xr*dipole[0];
      traced[1] = xr*dipole[1] + yr*dipole[0];
      traced[2] = xr*dipole[2] + zr*dipole[0];
      traced[3] = yr*dipole[0] + xr*dipole[1];
      traced[4] = 2.0*yr*dipole[1];
      traced[5] = yr*dipole[2] + zr*dipole[1];
      traced[6] = zr*dipole[0] + xr*dipole[2];
      traced[7] = zr*dipole[1] + yr*dipole[2];
      traced[8] = 2.0*zr*dipole[2];
      qave = (traced[0] + traced[4] + traced[8]) / 3.0;
      sdhquadrupole[0] += 1.5*(traced[0] - qave);
      sdhquadrupole[1] += 1.5*(traced[1]);
      sdhquadrupole[2] += 1.5*(traced[2]);
      sdhquadrupole[3] += 1.5*(traced[3]);
      sdhquadrupole[4] += 1.5*(traced[4] - qave);
      sdhquadrupole[5] += 1.5*(traced[5]);
      sdhquadrupole[6] += 1.5*(traced[6]);
      sdhquadrupole[7] += 1.5*(traced[7]);
      sdhquadrupole[8] += 1.5*(traced[8] - qave);
#endif /* if defined(WITH_TINKER) */
    }
   }
   /* SDH dipole and traceless quadrupole values
    were checked against similar routines in TINKER
    for large proteins.

    debye=4.8033324;
    printf("%6.3f, %6.3f, %6.3f\n", sdhdipole[0]*debye,
    sdhdipole[1]*debye, sdhdipole[2]*debye);
    printf("%6.3f\n", sdhquadrupole[0]*debye);
    printf("%6.3f %6.3f\n", sdhquadrupole[3]*debye,
    sdhquadrupole[4]*debye);
    printf("%6.3f %6.3f %6.3f\n", sdhquadrupole[6]*debye,
    sdhquadrupole[7]*debye, sdhquadrupole[8]*debye);
    */

   bcfl2(size, position, sdhcharge, sdhdipole, sdhquadrupole,
      xkappa, eps_p, eps_w, T, thee->gxcf, thee->gycf,
      thee->gzcf, thee->xf, thee->yf, thee->zf, nx, ny, nz);
            break;

        case BCFL_MDH:
   for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom
++) {
    atom = Valist_getAtom(alist, iatom);
    position = Vatom_getPosition(atom);
    charge = Vunit_ec*Vatom_getCharge(atom);
    dipole = VNULL;
    quadrupole = VNULL;
    size = Vatom_getRadius(atom);
    switch (thee->chargeSrc)
    {
     case VCM_CHARGE:
      ;
#if  defined(WITH_TINKER)
     case VCM_PERMANENT:
      dipole = Vatom_getDipole(atom);
      quadrupole = Vatom_getQuadrupole(atom);

     case VCM_INDUCED:
      dipole = Vatom_getInducedDipole(atom);

     case VCM_NLINDUCED:
      dipole = Vatom_getNLInducedDipole(atom);
#endif
    }
    bcfl1(size, position, charge, xkappa, pre1,
       thee->gxcf, thee->gycf, thee->gzcf,
       thee->xf, thee->yf, thee->zf, nx, ny, nz);
   }
   break;
```

```

        case BCFL_UNUSED:
            Vnm_print(2, "bcCalc:  Invalid bcfl (%d)!\n", thee->pmgp->bcfl
);
            VASSERT(0);

        case BCFL_FOCUS:
            Vnm_print(2, "VPMG::bcCalc -- not appropriate for focusing!\n");
            VASSERT(0);

        default:
            Vnm_print(2, "VPMG::bcCalc -- invalid boundary condition \
flag (%d)!\n", thee->pmgp->bcfl);
            VASSERT(0);
    }
}

/*
 Used by bcflnew
 */
VPRIVATE int gridPointIsValid(int i, int j, int k, int nx, int ny, int nz){

 int isValid = 0;

 if((k==0) || (k==nz-1)){
  isValid = 1;
 }else if((j==0) || (j==ny-1)){
  isValid = 1;
 }else if((i==0) || (i==nx-1)){
  isValid = 1;
 }

 return isValid;
}

/*
 Used by bcflnew
 */
#ifdef DEBUG_MAC_OSX_OCL
#include "mach_chud.h"
VPRIVATE void packAtomsOpenCL(float *ax, float *ay, float *az,
      float *charge, float *size, Vpmg *thee){

 int i;
 int natoms;

 Vatom *atom = VNULL;
 Valist *alist = VNULL;

 alist = thee->pbe->alist;
 natoms = Valist_getNumberAtoms(alist);

 for(i=0;i<natoms;i++){
  atom = &(alist->atoms[i]);
  charge[i] = Vunit_ec*atom->charge;
  ax[i] = atom->position[0];
  ay[i] = atom->position[1];
  az[i] = atom->position[2];
  size[i] = atom->radius;
 }
}

/*
 Used by bcflnew
 */
VPRIVATE void packUnpackOpenCL(int nx, int ny, int nz, int ngrid,
       float *gx, float *gy, float *gz, float *value,
       Vpmg *thee, int pack){

 int i,j,k,igrid;
 int x0,x1,y0,y1,z0,z1;

 float gpos[3];
 double *xf, *yf, *zf;
 double *gxcf, *gycf, *gzcf;

 xf = thee->xf;
 yf = thee->yf;
 zf = thee->zf;

 gxcf = thee->gxcf;
```

```
gycf = thee->gycf;
gzcf = thee->gzcf;

igrid = 0;
for(k=0;k<nz;k++){
 gpos[2] = zf[k];
 for(j=0;j<ny;j++){
  gpos[1] = yf[j];
  for(i=0;i<nx;i++){
   gpos[0] = xf[i];
   if(gridPointIsValid(i, j, k, nx, ny, nz)){
    if(pack != 0){
     gx[igrid] = gpos[0];
     gy[igrid] = gpos[1];
     gz[igrid] = gpos[2];

     value[igrid] = 0.0;
    }else{
     x0 = IJKx(j,k,0);
     x1 = IJKx(j,k,1);
     y0 = IJKy(i,k,0);
     y1 = IJKy(i,k,1);
     z0 = IJKz(i,j,0);
     z1 = IJKz(i,j,1);

     if(i==0){
      gxcf[x0] += value[igrid];
     }
     if(i==nx-1){
      gxcf[x1] += value[igrid];
     }
     if(j==0){
      gycf[y0] += value[igrid];
     }
     if(j==ny-1){
      gycf[y1] += value[igrid];
     }
     if(k==0){
      gzcf[z0] += value[igrid];
     }
     if(k==nz-1){
      gzcf[z1] += value[igrid];
     }
    }

    igrid++;
   } //end is valid point
  } //end i
 } //end j
} //end k

}

/*
 bcflnew is an optimized replacement for bcfl1. bcfl1 is still used when TINKER
 support is compiled in.
 bcflnew uses: packUnpack, packAtoms, gridPointIsValid
 */
VPRIVATE void bcflnewOpenCL(Vpmg *thee){

 int i,j,k, iatom, igrid;
 int x0, x1, y0, y1, z0, z1;

 int nx, ny, nz;
 int natoms, ngrid, ngadj;

 float dist, pre1, eps_w, eps_p, T, xkappa;

 float *ax, *ay, *az;
 float *charge, *size, *val;

 float *gx, *gy, *gz;

 Vpbe *pbe = thee->pbe;

 nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;

 eps_w = Vpbe_getSolventDiel(pbe);           /*
 Dimensionless */
```

```
    eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
    T = Vpbe_getTemperature(pbe);               /* K */
    pre1 = ((Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T))*(1.0e10);
 xkappa = Vpbe_getXkappa(pbe);

 natoms = Valist_getNumberAtoms(thee->pbe->alist);
 ngrid = 2*((nx*ny) + (ny*nz) + (nx*nz));
 ngadj = ngrid + (512 - (ngrid & 511));

 ax = (float*)malloc(natoms * sizeof(float));
 ay = (float*)malloc(natoms * sizeof(float));
 az = (float*)malloc(natoms * sizeof(float));

 charge = (float*)malloc(natoms * sizeof(float));
 size = (float*)malloc(natoms * sizeof(float));

 gx = (float*)malloc(ngrid * sizeof(float));
 gy = (float*)malloc(ngrid * sizeof(float));
 gz = (float*)malloc(ngrid * sizeof(float));

 val = (float*)malloc(ngrid * sizeof(float));

 packAtomsOpenCL(ax,ay,az,charge,size,thee);
 packUnpackOpenCL(nx,ny,nz,ngrid,gx,gy,gz,val,thee,1);

 runMDHCL(ngrid,natoms,ngadj,ax,ay,az,gx,gy,gz,charge,size,xkappa,pre1,val);

 packUnpackOpenCL(nx,ny,nz,ngrid,gx,gy,gz,val,thee,0);

 free(ax);
 free(ay);
 free(az);
 free(charge);
 free(size);

 free(gx);
 free(gy);
 free(gz);
 free(val);
}
#endif

VPRIVATE void packAtoms(double *ax, double *ay, double *az,
      double *charge, double *size, Vpmg *thee){

 int i;
 int natoms;

 Vatom *atom = VNULL;
 Valist *alist = VNULL;

 alist = thee->pbe->alist;
 natoms = Valist_getNumberAtoms(alist);

 for(i=0;i<natoms;i++){
  atom = &(alist->atoms[i]);
  charge[i] = Vunit_ec*atom->charge;
  ax[i] = atom->position[0];
  ay[i] = atom->position[1];
  az[i] = atom->position[2];
  size[i] = atom->radius;
 }
}

/*
 Used by bcflnew
 */
VPRIVATE void packUnpack(int nx, int ny, int nz, int ngrid,
       double *gx, double *gy, double *gz, double *value,
       Vpmg *thee, int pack){

 int i,j,k,igrid;
 int x0,x1,y0,y1,z0,z1;

 double gpos[3];
 double *xf, *yf, *zf;
 double *gxcf, *gycf, *gzcf;

```

```
xf = thee->xf;
yf = thee->yf;
zf = thee->zf;

gxcf = thee->gxcf;
gycf = thee->gycf;
gzcf = thee->gzcf;

igrid = 0;
for(k=0;k<nz;k++){
 gpos[2] = zf[k];
 for(j=0;j<ny;j++){
  gpos[1] = yf[j];
  for(i=0;i<nx;i++){
   gpos[0] = xf[i];
   if(gridPointIsValid(i, j, k, nx, ny, nz)){
    if(pack != 0){
     gx[igrid] = gpos[0];
     gy[igrid] = gpos[1];
     gz[igrid] = gpos[2];

     value[igrid] = 0.0;
    }else{
     x0 = IJKx(j,k,0);
     x1 = IJKx(j,k,1);
     y0 = IJKy(i,k,0);
     y1 = IJKy(i,k,1);
     z0 = IJKz(i,j,0);
     z1 = IJKz(i,j,1);

     if(i==0){
      gxcf[x0] += value[igrid];
     }
     if(i==nx-1){
      gxcf[x1] += value[igrid];
     }
     if(j==0){
      gycf[y0] += value[igrid];
     }
     if(j==ny-1){
      gycf[y1] += value[igrid];
     }
     if(k==0){
      gzcf[z0] += value[igrid];
     }
     if(k==nz-1){
      gzcf[z1] += value[igrid];
     }
    }

    igrid++;
   } //end is valid point
  } //end i
 } //end j
} //end k

}

VPRIVATE void bcflnew(Vpmg *thee){

int i,j,k, iatom, igrid;
int x0, x1, y0, y1, z0, z1;

int nx, ny, nz;
int natoms, ngrid;

double dist, pre1, eps_w, eps_p, T, xkappa;

double *ax, *ay, *az;
double *charge, *size, *val;

double *gx, *gy, *gz;

Vpbe *pbe = thee->pbe;

nx = thee->pmgp->nx;
   ny = thee->pmgp->ny;
   nz = thee->pmgp->nz;

eps_w = Vpbe_getSolventDiel(pbe);          /* Dimensionless */
```

```
    eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
    T = Vpbe_getTemperature(pbe);               /* K */
    pre1 = ((Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T))*(1.0e10);
 xkappa = Vpbe_getXkappa(pbe);

 natoms = Valist_getNumberAtoms(thee->pbe->alist);
 ngrid = 2*((nx*ny) + (ny*nz) + (nx*nz));

 ax = (double*)malloc(natoms * sizeof(double));
 ay = (double*)malloc(natoms * sizeof(double));
 az = (double*)malloc(natoms * sizeof(double));

 charge = (double*)malloc(natoms * sizeof(double));
 size = (double*)malloc(natoms * sizeof(double));

 gx = (double*)malloc(ngrid * sizeof(double));
 gy = (double*)malloc(ngrid * sizeof(double));
 gz = (double*)malloc(ngrid * sizeof(double));

 val = (double*)malloc(ngrid * sizeof(double));

 packAtoms(ax,ay,az,charge,size,thee);
 packUnpack(nx,ny,nz,ngrid,gx,gy,gz,val,thee,1);

 if(xkappa > VSMALL){
#pragma omp parallel for default(shared) private(igrid,iatom,dist)
  for(igrid=0;igrid<ngrid;igrid++){
   for(iatom=0; iatom<natoms; iatom++){
    dist = VSQRT(VSQR(gx[igrid]-ax[iatom]) + VSQR(gy[igrid]-ay[iatom])
        + VSQR(gz[igrid]-az[iatom]));
    val[igrid] += pre1*(charge[iatom]/dist)*VEXP(-xkappa*(dist-size[iatom]))
    / (1+xkappa*size[iatom]);
   }
  }
 }else{
#pragma omp parallel for default(shared) private(igrid,iatom,dist)
  for(igrid=0;igrid<ngrid;igrid++){
   for(iatom=0; iatom<natoms; iatom++){
    dist = VSQRT(VSQR(gx[igrid]-ax[iatom]) + VSQR(gy[igrid]-ay[iatom])
        + VSQR(gz[igrid]-az[iatom]));
    val[igrid] += pre1*(charge[iatom]/dist);
   }
  }
 }
 packUnpack(nx,ny,nz,ngrid,gx,gy,gz,val,thee,0);

 free(ax);
 free(ay);
 free(az);
 free(charge);
 free(size);

 free(gx);
 free(gy);
 free(gz);
 free(val);
}

VPRIVATE void multipolebc(double r, double kappa, double eps_p,
                          double eps_w, double rad, double tsr[3]) {
    double r2,r3,r5;
    double eps_r;
    double ka,ka2,ka3;
    double kr,kr2,kr3;

    /*
  Below an attempt is made to explain the potential outside of a
  multipole located at the center of spherical cavity of dieletric
  eps_p, with dielectric eps_w outside (and possibly kappa > 0).


  First, eps_p = 1.0
  eps_w = 1.0
  kappa = 0.0

  The general form for the potential of a traceless multipole tensor
  of rank n in vacuum is:

```

```
V(r) = (-1)^n * u . n . Del^n (1/r)

where
u                     is a multipole of order n (3^n components)
u . n. Del^n (1/r)    is the contraction of u with the nth
derivative of 1/r

for example, if n = 1, the dipole potential is
V_vac(r) = (-1)*[ux*x + uy*y + uz*z]/r^3

This function returns the parts of V(r) for multipoles of
order 0, 1 and 2 that are independent of the contraction.

For the vacuum example, this would be 1/r, -1/r^3 and 3/r^5
respectively.

*** Note that this requires that the quadrupole is
traceless. If not, the diagonal quadrupole potential changes
from
qaa *  3*a^2/r^5
to
qaa * (3*a^2/r^5 - 1/r^3a )
where we sum over the trace; a = x, y and z.

(In other words, the -1/r^3 term cancels for a traceless quadrupole.
qxx + qyy + qzz = 0
such that
-(qxx + qyy + qzz)/r^3 = 0

For quadrupole with trace:
qxx + qyy + qzz != 0
such that
-(qxx + qyy + qzz)/r^3 != 0
)

=========================================================================

eps_p != 1 or eps_w != 1
kappa = 0.0

If the multipole is placed at the center of a sphere with
dieletric eps_p in a solvent of dielectric eps_w, the potential
outside the sphere is the solution to the Laplace equation:

V(r) = 1/eps_w * (2*n+1)*eps_r/(n+(n+1)*eps_r)
* (-1)^n * u . n . Del^n (1/r)
where
eps_r = eps_w / eps_p
is the ratio of solvent to solute dielectric

=========================================================================

kappa > 0

Finally, if the region outside the sphere is treated by the linearized
PB equation with Debye-Huckel parameter kappa, the solution is:

V(r) = kappa/eps_w * alpha_n(kappa*a) * K_n(kappa*r) * r^(n+1)/a^n
* (-1)^n * u . n . Del^n (1/r)
where
alpha_n(x) is [(2n + 1) / x] / [(n*K_n(x)/eps_r) - x*K_n'(x)]
K_n(x) are modified spherical Bessel functions of the third kind.
K_n'(x) is the derivative of K_n(x)
   */

  eps_r = eps_w/eps_p;
  r2 = r*r;
  r3 = r2*r;
  r5 = r3*r2;
  tsr[0] = (1.0/eps_w)/r;
  tsr[1] = (1.0/eps_w)*(-1.0)/r3;
  tsr[2] = (1.0/eps_w)*(3.0)/r5;
  if (kappa < VSMALL) {
      tsr[1] = (3.0*eps_r)/(1.0 + 2.0*eps_r)*tsr[1];
      tsr[2] = (5.0*eps_r)/(2.0 + 3.0*eps_r)*tsr[2];
  } else {
      ka = kappa*rad;
      ka2 = ka*ka;
      ka3 = ka2*ka;
      kr = kappa*r;
      kr2 = kr*kr;
```

```
        kr3 = kr2*kr;
        tsr[0] = exp(ka-kr) / (1.0 + ka) * tsr[0];
        tsr[1] = 3.0*eps_r*exp(ka-kr)*(1.0 + kr) * tsr[1];
        tsr[1] = tsr[1] / (1.0 + ka + eps_r*(2.0 + 2.0*ka + ka2));
        tsr[2] = 5.0*eps_r*exp(ka-kr)*(3.0 + 3.0*kr + kr2) * tsr[2];
        tsr[2] = tsr[2]/(6.0+6.0*ka+2.0*ka2+eps_r*(9.0+9.0*ka+4.0*ka2+ka3));
    }
}

VPRIVATE void bcfl_sdh(Vpmg *thee){

 int i,j,k,iatom;
 int nx, ny, nz;

 double size, *position, charge, xkappa, eps_w, eps_p, T, pre, dist;
 double sdhcharge, sdhdipole[3], traced[9], sdhquadrupole[9];
 double *dipole, *quadrupole;

 double val, *apos, gpos[3], tensor[3], qave;
 double ux, uy, uz, xr, yr, zr;
 double qxx,qxy,qxz,qyx,qyy,qyz,qzx,qzy,qzz;

 double *xf, *yf, *zf;
 double *gxcf, *gycf, *gzcf;

 Vpbe *pbe;
 Vatom *atom;
    Valist *alist;

 pbe = thee->pbe;
 alist = thee->pbe->alist;
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;

 xf = thee->xf;
 yf = thee->yf;
 zf = thee->zf;

 gxcf = thee->gxcf;
 gycf = thee->gycf;
 gzcf = thee->gzcf;

 /* For each "atom" (only one for bcfl=1), we use the following formula to
  * calculate the boundary conditions:
  *    g(x) = \frac{q e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
  *          * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
  *          * 1/d
  * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
  * We only need to evaluate some of these prefactors once:
  *    pre1 = \frac{e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
  * which gives the potential as
  *    g(x) = pre1 * q/d * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
  */
    eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
    eps_p = Vpbe_getSoluteDiel(pbe);            /* Dimensionless */
    T = Vpbe_getTemperature(pbe);               /* K */

    pre = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*Vunit_kb*T);
    pre = pre*(1.0e10);

 /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
  * m/A, then we will only need to deal with distances and sizes in
  * Angstroms rather than meters.                                    */
    xkappa = Vpbe_getXkappa(pbe);               /* A^{-1} */

 /* Solute size and position */
 size = Vpbe_getSoluteRadius(pbe);
 position = Vpbe_getSoluteCenter(pbe);

 sdhcharge = 0.0;
 for (i=0; i<3; i++) sdhdipole[i] = 0.0;
 for (i=0; i<9; i++) sdhquadrupole[i] = 0.0;

 for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
```

```
  atom = Valist_getAtom(alist, iatom);
  apos = Vatom_getPosition(atom);
  xr = apos[0] - position[0];
  yr = apos[1] - position[1];
  zr = apos[2] - position[2];
  switch (thee->chargeSrc) {
   case VCM_CHARGE:
    charge = Vatom_getCharge(atom);
    sdhcharge += charge;
    sdhdipole[0] += xr * charge;
    sdhdipole[1] += yr * charge;
    sdhdipole[2] += zr * charge;
    traced[0] = xr*xr*charge;
    traced[1] = xr*yr*charge;
    traced[2] = xr*zr*charge;
    traced[3] = yr*xr*charge;
    traced[4] = yr*yr*charge;
    traced[5] = yr*zr*charge;
    traced[6] = zr*xr*charge;
    traced[7] = zr*yr*charge;
    traced[8] = zr*zr*charge;
    qave = (traced[0] + traced[4] + traced[8]) / 3.0;
    sdhquadrupole[0] += 1.5*(traced[0] - qave);
    sdhquadrupole[1] += 1.5*(traced[1]);
    sdhquadrupole[2] += 1.5*(traced[2]);
    sdhquadrupole[3] += 1.5*(traced[3]);
    sdhquadrupole[4] += 1.5*(traced[4] - qave);
    sdhquadrupole[5] += 1.5*(traced[5]);
    sdhquadrupole[6] += 1.5*(traced[6]);
    sdhquadrupole[7] += 1.5*(traced[7]);
    sdhquadrupole[8] += 1.5*(traced[8] - qave);
#if defined(WITH_TINKER)
   case VCM_PERMANENT:
    charge = Vatom_getCharge(atom);
    dipole = Vatom_getDipole(atom);
    quadrupole = Vatom_getQuadrupole(atom);
    sdhcharge += charge;
    sdhdipole[0] += xr * charge;
    sdhdipole[1] += yr * charge;
    sdhdipole[2] += zr * charge;
    traced[0] = xr*xr*charge;
    traced[1] = xr*yr*charge;
    traced[2] = xr*zr*charge;
    traced[3] = yr*xr*charge;
    traced[4] = yr*yr*charge;
    traced[5] = yr*zr*charge;
    traced[6] = zr*xr*charge;
    traced[7] = zr*yr*charge;
    traced[8] = zr*zr*charge;
    sdhdipole[0] += dipole[0];
    sdhdipole[1] += dipole[1];
    sdhdipole[2] += dipole[2];
    traced[0] += 2.0*xr*dipole[0];
    traced[1] += xr*dipole[1] + yr*dipole[0];
    traced[2] += xr*dipole[2] + zr*dipole[0];
    traced[3] += yr*dipole[0] + xr*dipole[1];
    traced[4] += 2.0*yr*dipole[1];
    traced[5] += yr*dipole[2] + zr*dipole[1];
    traced[6] += zr*dipole[0] + xr*dipole[2];
    traced[7] += zr*dipole[1] + yr*dipole[2];
    traced[8] += 2.0*zr*dipole[2];
    qave = (traced[0] + traced[4] + traced[8]) / 3.0;
    sdhquadrupole[0] += 1.5*(traced[0] - qave);
    sdhquadrupole[1] += 1.5*(traced[1]);
    sdhquadrupole[2] += 1.5*(traced[2]);
    sdhquadrupole[3] += 1.5*(traced[3]);
    sdhquadrupole[4] += 1.5*(traced[4] - qave);
    sdhquadrupole[5] += 1.5*(traced[5]);
    sdhquadrupole[6] += 1.5*(traced[6]);
    sdhquadrupole[7] += 1.5*(traced[7]);
    sdhquadrupole[8] += 1.5*(traced[8] - qave);
    sdhquadrupole[0] += quadrupole[0];
    sdhquadrupole[1] += quadrupole[1];
    sdhquadrupole[2] += quadrupole[2];
    sdhquadrupole[3] += quadrupole[3];
    sdhquadrupole[4] += quadrupole[4];
    sdhquadrupole[5] += quadrupole[5];
    sdhquadrupole[6] += quadrupole[6];
    sdhquadrupole[7] += quadrupole[7];
    sdhquadrupole[8] += quadrupole[8];
   case VCM_INDUCED:
```

```
    dipole = Vatom_getInducedDipole(atom);
    sdhdipole[0] += dipole[0];
    sdhdipole[1] += dipole[1];
    sdhdipole[2] += dipole[2];
    traced[0] = 2.0*xr*dipole[0];
    traced[1] = xr*dipole[1] + yr*dipole[0];
    traced[2] = xr*dipole[2] + zr*dipole[0];
    traced[3] = yr*dipole[0] + xr*dipole[1];
    traced[4] = 2.0*yr*dipole[1];
    traced[5] = yr*dipole[2] + zr*dipole[1];
    traced[6] = zr*dipole[0] + xr*dipole[2];
    traced[7] = zr*dipole[1] + yr*dipole[2];
    traced[8] = 2.0*zr*dipole[2];
    qave = (traced[0] + traced[4] + traced[8]) / 3.0;
    sdhquadrupole[0] += 1.5*(traced[0] - qave);
    sdhquadrupole[1] += 1.5*(traced[1]);
    sdhquadrupole[2] += 1.5*(traced[2]);
    sdhquadrupole[3] += 1.5*(traced[3]);
    sdhquadrupole[4] += 1.5*(traced[4] - qave);
    sdhquadrupole[5] += 1.5*(traced[5]);
    sdhquadrupole[6] += 1.5*(traced[6]);
    sdhquadrupole[7] += 1.5*(traced[7]);
    sdhquadrupole[8] += 1.5*(traced[8] - qave);
   case VCM_NLINDUCED:
    dipole = Vatom_getNLInducedDipole(atom);
    sdhdipole[0] += dipole[0];
    sdhdipole[1] += dipole[1];
    sdhdipole[2] += dipole[2];
    traced[0] = 2.0*xr*dipole[0];
    traced[1] = xr*dipole[1] + yr*dipole[0];
    traced[2] = xr*dipole[2] + zr*dipole[0];
    traced[3] = yr*dipole[0] + xr*dipole[1];
    traced[4] = 2.0*yr*dipole[1];
    traced[5] = yr*dipole[2] + zr*dipole[1];
    traced[6] = zr*dipole[0] + xr*dipole[2];
    traced[7] = zr*dipole[1] + yr*dipole[2];
    traced[8] = 2.0*zr*dipole[2];
    qave = (traced[0] + traced[4] + traced[8]) / 3.0;
    sdhquadrupole[0] += 1.5*(traced[0] - qave);
    sdhquadrupole[1] += 1.5*(traced[1]);
    sdhquadrupole[2] += 1.5*(traced[2]);
    sdhquadrupole[3] += 1.5*(traced[3]);
    sdhquadrupole[4] += 1.5*(traced[4] - qave);
    sdhquadrupole[5] += 1.5*(traced[5]);
    sdhquadrupole[6] += 1.5*(traced[6]);
    sdhquadrupole[7] += 1.5*(traced[7]);
    sdhquadrupole[8] += 1.5*(traced[8] - qave);
#endif /* if defined(WITH_TINKER) */
  }
 }

 ux = sdhdipole[0];
 uy = sdhdipole[1];
    uz = sdhdipole[2];

 /* The factor of 1/3 results from using a
  traceless quadrupole definition. See, for example,
  "The Theory of Intermolecular Forces" by A.J. Stone,
  Chapter 3. */
 qxx = sdhquadrupole[0] / 3.0;
 qxy = sdhquadrupole[1] / 3.0;
 qxz = sdhquadrupole[2] / 3.0;
 qyx = sdhquadrupole[3] / 3.0;
 qyy = sdhquadrupole[4] / 3.0;
 qyz = sdhquadrupole[5] / 3.0;
 qzx = sdhquadrupole[6] / 3.0;
 qzy = sdhquadrupole[7] / 3.0;
 qzz = sdhquadrupole[8] / 3.0;

 for(k=0;k<nz;k++){
  gpos[2] = zf[k];
  for(j=0;j<ny;j++){
   gpos[1] = yf[j];
   for(i=0;i<nx;i++){
    gpos[0] = xf[i];
    if(gridPointIsValid(i, j, k, nx, ny, nz)){
     xr = gpos[0] - position[0];
     yr = gpos[1] - position[1];
     zr = gpos[2] - position[2];

     dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
```

```
      multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);

      val = pre*sdhcharge*tensor[0];
      val -= pre*ux*xr*tensor[1];
      val -= pre*uy*yr*tensor[1];
      val -= pre*uz*zr*tensor[1];
      val += pre*qxx*xr*xr*tensor[2];
      val += pre*qyy*yr*yr*tensor[2];
      val += pre*qzz*zr*zr*tensor[2];
      val += pre*2.0*qxy*xr*yr*tensor[2];
      val += pre*2.0*qxz*xr*zr*tensor[2];
      val += pre*2.0*qyz*yr*zr*tensor[2];

      if(i==0){
       gxcf[IJKx(j,k,0)] = val;
      }
      if(i==nx-1){
       gxcf[IJKx(j,k,1)] = val;
      }
      if(j==0){
       gycf[IJKy(i,k,0)] = val;
      }
      if(j==ny-1){
       gycf[IJKy(i,k,1)] = val;
      }
      if(k==0){
       gzcf[IJKz(i,j,0)] = val;
      }
      if(k==nz-1){
       gzcf[IJKz(i,j,1)] = val;
      }
     } /* End grid point is valid */
    } /* End i loop */
   } /* End j loop */
  } /* End k loop */

}

VPRIVATE void bcfl_mdh(Vpmg *thee){

 int i,j,k,iatom;
 int nx, ny, nz;

 double val, *apos, gpos[3];
 double *dipole, *quadrupole;
 double size, charge, xkappa, eps_w, eps_p, T, pre1, dist;

 double *xf, *yf, *zf;
 double *gxcf, *gycf, *gzcf;

 Vpbe *pbe;
 Vatom *atom;
    Valist *alist;

 pbe = thee->pbe;
 alist = thee->pbe->alist;
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;

 xf = thee->xf;
 yf = thee->yf;
 zf = thee->zf;

 gxcf = thee->gxcf;
 gycf = thee->gycf;
 gzcf = thee->gzcf;

 /* For each "atom" (only one for bcfl=1), we use the following formula to
  * calculate the boundary conditions:
  *    g(x) = \frac{q e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
  *          * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
  *          * 1/d
  * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
  * We only need to evaluate some of these prefactors once:
  *    pre1 = \frac{e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
  * which gives the potential as
  *    g(x) = pre1 * q/d * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
  */
    eps_w = Vpbe_getSolventDiel(pbe);           /*
 Dimensionless */
```

```
    eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
    T = Vpbe_getTemperature(pbe);               /* K */
    pre1 = (Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T);

    /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
   * m/A, then we will only need to deal with distances and sizes in
   * Angstroms rather than meters.                                    */
    xkappa = Vpbe_getXkappa(pbe);              /* A^{-1} */
    pre1 = pre1*(1.0e10);

  /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
   * m/A, then we will only need to deal with distances and sizes in
   * Angstroms rather than meters.                                    */
    xkappa = Vpbe_getXkappa(pbe);              /* A^{-1} */

  for(k=0;k<nz;k++){
   gpos[2] = zf[k];
   for(j=0;j<ny;j++){
    gpos[1] = yf[j];
    for(i=0;i<nx;i++){
     gpos[0] = xf[i];
     if(gridPointIsValid(i, j, k, nx, ny, nz)){

      val = 0.0;

      for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
       atom = Valist_getAtom(alist, iatom);
       apos = Vatom_getPosition(atom);
       charge = Vunit_ec*Vatom_getCharge(atom);
       size = Vatom_getRadius(atom);

       dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
           + VSQR(gpos[2]-apos[2]));
       if (xkappa > VSMALL) {
        val += pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
        / (1+xkappa*size);
       } else {
        val += pre1*(charge/dist);
       }

      }

      if(i==0){
       gxcf[IJKx(j,k,0)] = val;
      }
      if(i==nx-1){
       gxcf[IJKx(j,k,1)] = val;
      }
      if(j==0){
       gycf[IJKy(i,k,0)] = val;
      }
      if(j==ny-1){
       gycf[IJKy(i,k,1)] = val;
      }
      if(k==0){
       gzcf[IJKz(i,j,0)] = val;
      }
      if(k==nz-1){
       gzcf[IJKz(i,j,1)] = val;
      }
     } /* End grid point is valid */
    } /* End i loop */
   } /* End j loop */
  } /* End k loop */

}

/* ///////////////////////////////////////////////////////////////////////////
 // Routine:  bcfl_mem
 //
 // Purpose:  Increment all the boundary points by the
 //           analytic expression for a membrane system in
 //           the presence of a membrane potential. This
 //           Boundary flag should only be used for systems
 //           that explicitly have membranes in the dielectric
```

```
//           and solvent maps.
//
//           There should be several input variables add to this
//           function such as membrane potential, membrane thickness
//           and height.
//
// Args:     apos is a 3-vector
//
// Author: Michael Grabe
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) {

    /* some definitions                              */
    /* L = total length of the membrane              */
    /* xkappa = inverse Debeye length                */
    /* zmem = z value of membrane bottom (Cytoplasm) */
    /* V = electrical potential inside the cell      */
    int i, j, k;
    double dist, val, z_low, z_high, z_shift;
    double A, B, C, D, edge_L, l;
    double G, z_0, z_rel;
    double gpos[3];

 Vnm_print(0, "Here is the value of kappa: %f\n",xkappa);
 Vnm_print(0, "Here is the value of L: %f\n",L);
 Vnm_print(0, "Here is the value of zmem: %f\n",zmem);
 Vnm_print(0, "Here is the value of mdie: %f\n",eps_m);
 Vnm_print(0, "Here is the value of memv: %f\n",V);

 /* no salt symmetric BC's at +/- infinity */
 // B=V/(edge_L - l*(1-eps_w/eps_m));
 // A=V + B*edge_L;
 // D=eps_w/eps_m*B;
 z_low = zmem;     /* this defines the bottom of the membrane */
 z_high = zmem + L;  /* this is the top of the membrane */

 /*****************************************************/
 /* proper boundary conditions for V = 0 extracellular */
 /* and psi=-V cytoplasm.                              */
 /* Implicit in this formulation is that the membrane  */
 /* center be at z = 0                                 */
 /*****************************************************/

 l=L/2;                     /* half of the membrane length */
 z_0 = z_low + l;           /* center of the membrane      */
 G=l*eps_w/eps_m*xkappa;
 A=-V/2*(1/(G+l))*exp(xkappa*l);
 B=V/2;
 C=-V/2*eps_w/eps_m*xkappa*(1/(G+l));
 D=-A;
 /* The analytic expression for the boundary conditions      */
 /* had the cytoplasmic surface of the membrane set to zero. */
 /* This requires an off-set of the BC equations.            */

    /* the "i" boundaries (dirichlet) */
    for (k=0; k<nz; k++) {
        gpos[2] = zf[k];
        z_rel = gpos[2] - z_0;    /* relative position for BCs */

        for (j=0; j<ny; j++) {

   if (gpos[2] <= z_low) {                       /* cytoplasmic */

    val = A*exp(xkappa*z_rel) + V;
    gxcf[IJKx(j,k,0)] += val;    /* assign low side BC */
    gxcf[IJKx(j,k,1)] += val;    /* assign high side BC */

   }

   else if (gpos[2] > z_low && gpos[2] <= z_high) {  /* in membrane */

    val = B + C*z_rel;
    gxcf[IJKx(j,k,0)] += val;    /* assign low side BC */
    gxcf[IJKx(j,k,1)] += val;    /* assign high side BC */

   }

   else if (gpos[2] > z_high)  {                  /* extracellular */

    val = D*exp(-xkappa*z_rel);
```

```
 gxcf[IJKx(j,k,0)] += val;    /* assign low side BC */
 gxcf[IJKx(j,k,1)] += val;    /* assign high side BC */

}

    }
 }

 /* the "j" boundaries (dirichlet) */
 for (k=0; k<nz; k++) {
     gpos[2] = zf[k];
     z_rel = gpos[2] - z_0;
     for (i=0; i<nx; i++) {

if (gpos[2] <= z_low) {                       /* cytoplasmic */

 val = A*exp(xkappa*z_rel) + V;
 gycf[IJKy(i,k,0)] += val;    /* assign low side BC */
 gycf[IJKy(i,k,1)] += val;    /* assign high side BC */
 //printf("%f \n",val);

}

else if (gpos[2] > z_low && gpos[2] <= z_high) {  /* in membrane */

 val = B + C*z_rel;
 gycf[IJKy(i,k,0)] += val;    /* assign low side BC */
 gycf[IJKy(i,k,1)] += val;    /* assign high side BC */
 //printf("%f \n",val);

}
else if (gpos[2] > z_high)  {                  /* extracellular */

 val = D*exp(-xkappa*z_rel);
 gycf[IJKy(i,k,0)] += val;    /* assign low side BC */
 gycf[IJKy(i,k,1)] += val;    /* assign high side BC */
 //printf("%f \n",val);

}

    }
 }

 /* the "k" boundaries (dirichlet) */
 for (j=0; j<ny; j++) {
     for (i=0; i<nx; i++) {

/* first assign the bottom boundary */

gpos[2] = zf[0];
z_rel = gpos[2] - z_0;

if (gpos[2] <= z_low) {                       /* cytoplasmic */

 val = A*exp(xkappa*z_rel) + V;
 gzcf[IJKz(i,j,0)] += val;    /* assign low side BC */
 //printf("%f \n",val);

}

else if (gpos[2] > z_low && gpos[2] <= z_high) {  /* in membrane */

 val = B + C*z_rel;
 gzcf[IJKz(i,j,0)] += val;    /* assign low side BC */

}

else if (gpos[2] > z_high)  {                  /* extracellular */

 val = D*exp(-xkappa*z_rel);
 gzcf[IJKz(i,j,0)] += val;    /* assign low side BC */

}

/* now assign the top boundary */

gpos[2] = zf[nz-1];
z_rel = gpos[2] - z_0;

if (gpos[2] <= z_low) {                       /* cytoplasmic */

```

```
    val = A*exp(xkappa*z_rel) + V;
    gzcf[IJKz(i,j,1)] += val;    /* assign high side BC */

   }

   else if (gpos[2] > z_low && gpos[2] <= z_high) {  /* in membrane */

    val = B + C*z_rel;
    gzcf[IJKz(i,j,1)] += val;    /* assign high side BC */

   }

   else if (gpos[2] > z_high)  {                  /* extracellular */

    val = D*exp(-xkappa*z_rel);
    gzcf[IJKz(i,j,1)] += val;    /* assign high side BC */
    //printf("%f \n",val);

   }

        }
    }
}

VPRIVATE void bcfl_map(Vpmg *thee){

 Vpbe *pbe;
    double position[3], pot, hx, hy, hzed;
    int i, j, k, nx, ny, nz, rc;


    VASSERT(thee != VNULL);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;

    /* Reset the potential array */
    for (i=0; i<(nx*ny*nz); i++) thee->pot[i] = 0.0;

    /* Fill in the source term (atomic potentials) */
    Vnm_print(0, "Vpmg_fillco:  filling in source term.\n");
    for (k=0; k<nz; k++) {
        for (j=0; j<ny; j++) {
            for (i=0; i<nx; i++) {
                position[0] = thee->xf[i];
                position[1] = thee->yf[j];
                position[2] = thee->zf[k];
    rc = Vgrid_value(thee->potMap, position, &pot);
    if (!rc) {
     Vnm_print(2, "fillcoChargeMap:  Error -- fell off of potential map at (%g,
 %g, %g)!\n",
         position[0], position[1], position[2]);
     VASSERT(0);
    }
                thee->pot[IJK(i,j,k)] = pot;
            }
        }
    }

}

#if  defined(WITH_TINKER)
VPRIVATE void bcfl_mdh_tinker(Vpmg *thee){

 int i,j,k,iatom;
 int nx, ny, nz;

 double val, *apos, gpos[3], tensor[9];
 double *dipole, *quadrupole;
 double size, charge, xkappa, eps_w, eps_p, T, pre1, dist;

 double ux,uy,uz,xr,yr,zr;
 double qxx,qxy,qxz,qyx,qyy,qyz,qzx,qzy,qzz;

 double *xf, *yf, *zf;
 double *gxcf, *gycf, *gzcf;
```

```

 Vpbe *pbe;
 Vatom *atom;
    Valist *alist;

 pbe = thee->pbe;
 alist = thee->pbe->alist;
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;

 xf = thee->xf;
 yf = thee->yf;
 zf = thee->zf;

 gxcf = thee->gxcf;
 gycf = thee->gycf;
 gzcf = thee->gzcf;

 /* For each "atom" (only one for bcfl=1), we use the following formula to
  * calculate the boundary conditions:
  *    g(x) = \frac{q e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
  *          * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
  *          * 1/d
  * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
  * We only need to evaluate some of these prefactors once:
  *    pre1 = \frac{e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
  * which gives the potential as
  *    g(x) = pre1 * q/d * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
  */
    eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
    eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
    T = Vpbe_getTemperature(pbe);               /* K */
    pre1 = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*Vunit_kb*T);

    /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
  * m/A, then we will only need to deal with distances and sizes in
  * Angstroms rather than meters.                                        */
    xkappa = Vpbe_getXkappa(pbe);              /* A^{-1} */
    pre1 = pre1*(1.0e10);

 /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
  * m/A, then we will only need to deal with distances and sizes in
  * Angstroms rather than meters.                                        */
    xkappa = Vpbe_getXkappa(pbe);              /* A^{-1} */

 for(k=0;k<nz;k++){
  gpos[2] = zf[k];
  for(j=0;j<ny;j++){
   gpos[1] = yf[j];
   for(i=0;i<nx;i++){
    gpos[0] = xf[i];
    if(gridPointIsValid(i, j, k, nx, ny, nz)){

     val = 0.0;

     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
      atom = Valist_getAtom(alist, iatom);
      apos = Vatom_getPosition(atom);
      size = Vatom_getRadius(atom);

      charge = 0.0;

      dipole = VNULL;
      quadrupole = VNULL;

      if (thee->chargeSrc == VCM_PERMANENT) {
       charge = Vatom_getCharge(atom);
       dipole = Vatom_getDipole(atom);
       quadrupole = Vatom_getQuadrupole(atom);
      } else if (thee->chargeSrc == VCM_INDUCED) {
       dipole = Vatom_getInducedDipole(atom);
      } else {
       dipole = Vatom_getNLInducedDipole(atom);
```

```
      }

      ux = dipole[0];
      uy = dipole[1];
      uz = dipole[2];

      if (quadrupole != VNULL) {
       /* The factor of 1/3 results from using a
        traceless quadrupole definition. See, for example,
        "The Theory of Intermolecular Forces" by A.J. Stone,
        Chapter 3. */
       qxx = quadrupole[0] / 3.0;
       qxy = quadrupole[1] / 3.0;
       qxz = quadrupole[2] / 3.0;
       qyx = quadrupole[3] / 3.0;
       qyy = quadrupole[4] / 3.0;
       qyz = quadrupole[5] / 3.0;
       qzx = quadrupole[6] / 3.0;
       qzy = quadrupole[7] / 3.0;
       qzz = quadrupole[8] / 3.0;
      } else {
       qxx = 0.0;
       qxy = 0.0;
       qxz = 0.0;
       qyx = 0.0;
       qyy = 0.0;
       qyz = 0.0;
       qzx = 0.0;
       qzy = 0.0;
       qzz = 0.0;
      }

      xr = gpos[0] - apos[0];
      yr = gpos[1] - apos[1];
      zr = gpos[2] - apos[2];

      dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
      multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);

      val += pre1*charge*tensor[0];
      val -= pre1*ux*xr*tensor[1];
      val -= pre1*uy*yr*tensor[1];
      val -= pre1*uz*zr*tensor[1];
      val += pre1*qxx*xr*xr*tensor[2];
      val += pre1*qyy*yr*yr*tensor[2];
      val += pre1*qzz*zr*zr*tensor[2];
      val += pre1*2.0*qxy*xr*yr*tensor[2];
      val += pre1*2.0*qxz*xr*zr*tensor[2];
      val += pre1*2.0*qyz*yr*zr*tensor[2];

     }

     if(i==0){
      gxcf[IJKx(j,k,0)] = val;
     }
     if(i==nx-1){
      gxcf[IJKx(j,k,1)] = val;
     }
     if(j==0){
      gycf[IJKy(i,k,0)] = val;
     }
     if(j==ny-1){
      gycf[IJKy(i,k,1)] = val;
     }
     if(k==0){
      gzcf[IJKz(i,j,0)] = val;
     }
     if(k==nz-1){
      gzcf[IJKz(i,j,1)] = val;
     }
    } /* End grid point is valid */
   } /* End i loop */
  } /* End j loop */
 } /* End k loop */

}
#endif

VPRIVATE void bcCalc(Vpmg *thee){

 int i, j, k;
```

```
int nx, ny, nz;

double zmem, eps_m, Lmem, memv, eps_w, xkappa;

   nx = thee->pmgp->nx;
   ny = thee->pmgp->ny;
   nz = thee->pmgp->nz;

   /* Zero out the boundaries */
   /* the "i" boundaries (dirichlet) */
   for (k=0; k<nz; k++) {
       for (j=0; j<ny; j++) {
           thee->gxcf[IJKx(j,k,0)] = 0.0;
           thee->gxcf[IJKx(j,k,1)] = 0.0;
           thee->gxcf[IJKx(j,k,2)] = 0.0;
           thee->gxcf[IJKx(j,k,3)] = 0.0;
       }
   }

   /* the "j" boundaries (dirichlet) */
   for (k=0; k<nz; k++) {
       for (i=0; i<nx; i++) {
           thee->gycf[IJKy(i,k,0)] = 0.0;
           thee->gycf[IJKy(i,k,1)] = 0.0;
           thee->gycf[IJKy(i,k,2)] = 0.0;
           thee->gycf[IJKy(i,k,3)] = 0.0;
       }
   }

   /* the "k" boundaries (dirichlet) */
   for (j=0; j<ny; j++) {
       for (i=0; i<nx; i++) {
           thee->gzcf[IJKz(i,j,0)] = 0.0;
           thee->gzcf[IJKz(i,j,1)] = 0.0;
           thee->gzcf[IJKz(i,j,2)] = 0.0;
           thee->gzcf[IJKz(i,j,3)] = 0.0;
       }
   }

 switch (thee->pmgp->bcfl) {
   /*  If we have zero boundary conditions, we're done */
       case BCFL_ZERO:
           return;
  case BCFL_SDH:
   bcfl_sdh(thee);
   break;
  case BCFL_MDH:
#if defined(WITH_TINKER)
   bcfl_mdh_tinker(thee);
#else

#ifdef DEBUG_MAC_OSX_OCL
#include "mach_chud.h"
   uint64_t mbeg = mach_absolute_time();

   /*
    * If OpenCL is available we use it, otherwise fall back to
    * normal route (CPU multithreaded w/ OpenMP)
    */
   if (kOpenCLAvailable == 1) bcflnewOpenCL(thee);
   else bcflnew(thee);

   mets_(&mbeg, "MDH");
#else
   /* bcfl_mdh(thee); */
   bcflnew(thee);
#endif /* DEBUG_MAC_OSX_OCL */

#endif /* WITH_TINKER */
   break;
  case BCFL_MEM:

   zmem  = Vpbe_getzmem(thee->pbe);
   Lmem  = Vpbe_getLmem(thee->pbe);
   eps_m = Vpbe_getmembraneDiel(thee->pbe);
   memv =  Vpbe_getmemv(thee->pbe);

   eps_w = Vpbe_getSolventDiel(thee->pbe);
   xkappa = Vpbe_getXkappa(thee->pbe);

   bcfl_mem(zmem, Lmem, eps_m, eps_w, memv, xkappa,
```

```
      thee->gxcf, thee->gycf, thee->gzcf,
      thee->xf, thee->yf, thee->zf, nx, ny, nz);
   break;
        case BCFL_UNUSED:
            Vnm_print(2, "bcCalc:  Invalid bcfl (%d)!\n", thee->pmgp->bcfl);
            VASSERT(0);
   break;
        case BCFL_FOCUS:
            Vnm_print(2, "VPMG::bcCalc -- not appropriate for focusing!\n");
            VASSERT(0);
   break;
  case BCFL_MAP:
   bcfl_map(thee);
   focusFillBound(thee,VNULL);
   break;
        default:
            Vnm_print(2, "VPMG::bcCalc -- invalid boundary condition \
       flag (%d)!\n", thee->pmgp->bcfl);
            VASSERT(0);
   break;
    }
}

VPRIVATE void fillcoCoefMap(Vpmg *thee) {

    Vpbe *pbe;
    double ionstr, position[3], tkappa, eps, pot, hx, hy, hzed;
    int i, j, k, nx, ny, nz;
    double kappamax;
    VASSERT(thee != VNULL);

    /* Get PBE info */
    pbe = thee->pbe;
    ionstr = Vpbe_getBulkIonicStrength(pbe);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;

    if ((!thee->useDielXMap) || (!thee->useDielYMap)
  || (!thee->useDielZMap) || ((!thee->useKappaMap) && (ionstr>VPMGSMALL))) {

        Vnm_print(2, "fillcoCoefMap:  You need to use all coefficient maps!\n");
        VASSERT(0);

    }

    /* Scale the kappa map to values between 0 and 1
       Thus get the maximum value in the map - this
       is theoretically unnecessary, but a good check.*/
    kappamax = -1.00;
    for (k=0; k<nz; k++) {
        for (j=0; j<ny; j++) {
            for (i=0; i<nx; i++) {
                if (ionstr > VPMGSMALL) {
                     position[0] = thee->xf[i];
                     position[1] = thee->yf[j];
                     position[2] = thee->zf[k];
                     if (!Vgrid_value(thee->kappaMap, position, &tkappa)) {
                         Vnm_print(2, "Vpmg_fillco:  Off kappaMap at:\n");
                         Vnm_print(2, "Vpmg_fillco:  (x,y,z) = (%g,%g %g)\n",
                                   position[0], position[1], position[2]);
                         VASSERT(0);
                     }
                     if (tkappa > kappamax) {
                         kappamax = tkappa;
                     }
                     if (tkappa < 0.0){
                       Vnm_print(2, "Vpmg_fillcoCoefMap: Kappa map less than 0\n");
                       Vnm_print(2, "Vpmg_fillcoCoefMap: at (x,y,z) = (%g,%g %g)\n",
                                 position[0], position[1], position[2]);
```

```
                        VASSERT(0);
                      }
                }
            }
        }
    }

    if (kappamax > 1.0){
      Vnm_print(2, "Vpmg_fillcoCoefMap:  Maximum Kappa value\n");
      Vnm_print(2, "%g is greater than 1 - will scale appropriately!\n",
                kappamax);
    }
    else {
      kappamax = 1.0;
    }

    for (k=0; k<nz; k++) {
        for (j=0; j<ny; j++) {
            for (i=0; i<nx; i++) {

                if (ionstr > VPMGSMALL) {
                     position[0] = thee->xf[i];
                     position[1] = thee->yf[j];
                     position[2] = thee->zf[k];
                     if (!Vgrid_value(thee->kappaMap, position, &tkappa)) {
                         Vnm_print(2, "Vpmg_fillco:  Off kappaMap at:\n");
                         Vnm_print(2, "Vpmg_fillco:  (x,y,z) = (%g,%g %g)\n",
                           position[0], position[1], position[2]);
                         VASSERT(0);
                     }
                     if (tkappa < VPMGSMALL) tkappa = 0.0;
                     thee->kappa[IJK(i,j,k)] = (tkappa / kappamax);
                }

                position[0] = thee->xf[i] + 0.5*hx;
                position[1] = thee->yf[j];
                position[2] = thee->zf[k];
                if (!Vgrid_value(thee->dielXMap, position, &eps)) {
                    Vnm_print(2, "Vpmg_fillco:  Off dielXMap at:\n");
                    Vnm_print(2, "Vpmg_fillco:  (x,y,z) = (%g,%g %g)\n",
                      position[0], position[1], position[2]);
                    VASSERT(0);
                 }
                 thee->epsx[IJK(i,j,k)] = eps;

                 position[0] = thee->xf[i];
                 position[1] = thee->yf[j] + 0.5*hy;
                 position[2] = thee->zf[k];
                 if (!Vgrid_value(thee->dielYMap, position, &eps)) {
                    Vnm_print(2, "Vpmg_fillco:  Off dielYMap at:\n");
                    Vnm_print(2, "Vpmg_fillco:  (x,y,z) = (%g,%g %g)\n",
                      position[0], position[1], position[2]);
                    VASSERT(0);
                 }
                 thee->epsy[IJK(i,j,k)] = eps;

                 position[0] = thee->xf[i];
                 position[1] = thee->yf[j];
                 position[2] = thee->zf[k] + 0.5*hzed;
                 if (!Vgrid_value(thee->dielZMap, position, &eps)) {
                    Vnm_print(2, "Vpmg_fillco:  Off dielZMap at:\n");
                    Vnm_print(2, "Vpmg_fillco:  (x,y,z) = (%g,%g %g)\n",
                      position[0], position[1], position[2]);
                    VASSERT(0);
                 }
                 thee->epsz[IJK(i,j,k)] = eps;
            }
        }
    }
}

VPRIVATE void fillcoCoefMol(Vpmg *thee) {

    if (thee->useDielXMap || thee->useDielYMap || thee->useDielZMap ||
      thee->useKappaMap)  {

```

```
        fillcoCoefMap(thee);

    } else {

        fillcoCoefMolDiel(thee);
        fillcoCoefMolIon(thee);

    }

}

VPRIVATE void fillcoCoefMolIon(Vpmg *thee) {

    Vacc *acc;
    Valist *alist;
    Vpbe *pbe;
    Vatom *atom;
    double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr;
    double xlen, ylen, zlen, irad;
    double hx, hy, hzed, *apos, arad;
    int i, nx, ny, nz, iatom;
    Vsurf_Meth surfMeth;

    VASSERT(thee != VNULL);
    surfMeth = thee->surfMeth;

    /* Get PBE info */
    pbe = thee->pbe;
    acc = pbe->acc;
    alist = pbe->alist;
    irad = Vpbe_getMaxIonRadius(pbe);
    ionstr = Vpbe_getBulkIonicStrength(pbe);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;

    /* Define the total domain size */
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;

    /* Define the min/max dimensions */
    xmin = thee->pmgp->xcent - (xlen/2.0);
    ymin = thee->pmgp->ycent - (ylen/2.0);
    zmin = thee->pmgp->zcent - (zlen/2.0);
    xmax = thee->pmgp->xcent + (xlen/2.0);
    ymax = thee->pmgp->ycent + (ylen/2.0);
    zmax = thee->pmgp->zcent + (zlen/2.0);

    /* This is a floating point parameter related to the non-zero nature of the
     * bulk ionic strength.  If the ionic strength is greater than zero; this
     * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
     * Otherwise, this parameter is set to 0.0 */
    if (ionstr > VPMGSMALL) ionmask = 1.0;
    else ionmask = 0.0;

    /* Reset the kappa array, marking everything accessible */
    for (i=0; i<(nx*ny*nz); i++) thee->kappa[i] = ionmask;

    if (ionstr < VPMGSMALL) return;

    /* Loop through the atoms and set kappa = 0.0 (inaccessible) if a point
     * is inside the ion-inflated van der Waals radii */
    for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {

        atom = Valist_getAtom(alist, iatom);
        apos = Vatom_getPosition(atom);
        arad = Vatom_getRadius(atom);

        if (arad > VSMALL) {

            /* Make sure we're on the grid */
            if ((apos[0]<(xmin-irad-arad)) || (apos[0]>(xmax+irad+arad))  || \
                (apos[1]<(ymin-irad-arad)) || (apos[1]>(ymax+irad+arad))  || \
```

```
                (apos[2]<(zmin-irad-arad)) || (apos[2]>(zmax+irad+arad))) {
                if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
     (thee->pmgp->bcfl != BCFL_MAP)) {
                    Vnm_print(2,
    "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f, %4.3f) is off the mesh
   (ignoring):\n",
                      iatom, apos[0], apos[1], apos[2]);
                    Vnm_print(2, "Vpmg_fillco:  xmin = %g, xmax = %g\n",
                      xmin, xmax);
                    Vnm_print(2, "Vpmg_fillco:  ymin = %g, ymax = %g\n",
                      ymin, ymax);
                    Vnm_print(2, "Vpmg_fillco:  zmin = %g, zmax = %g\n",
                      zmin, zmax);
                }
                fflush(stderr);

            } else { /* if we're on the mesh */

                /* Mark ions */
                markSphere((irad+arad), apos,
                        nx, ny, nz,
                        hx, hy, hzed,
                        xmin, ymin, zmin,
                        thee->kappa, 0.0);

            } /* endif (on the mesh) */
        }
    } /* endfor (over all atoms) */

}

VPRIVATE void fillcoCoefMolDiel(Vpmg *thee) {

    /* Always call NoSmooth to fill the epsilon arrays */
    fillcoCoefMolDielNoSmooth(thee);

    /* Call the smoothing algorithm as needed */
    if (thee->surfMeth == VSM_MOLSMOOTH) {
        fillcoCoefMolDielSmooth(thee);
    }
}

VPRIVATE void fillcoCoefMolDielNoSmooth(Vpmg *thee
   ) {

    Vacc *acc;
    VaccSurf *asurf;
    Valist *alist;
    Vpbe *pbe;
    Vatom *atom;
    double xmin, xmax, ymin, ymax, zmin, zmax;
    double xlen, ylen, zlen, position[3];
    double srad, epsw, epsp, deps, area;
    double hx, hy, hzed, *apos, arad;
    int i, nx, ny, nz, ntot, iatom, ipt;

    /* Get PBE info */
    pbe = thee->pbe;
    acc = pbe->acc;
    alist = pbe->alist;
    srad = Vpbe_getSolventRadius(pbe);
    epsw = Vpbe_getSolventDiel(pbe);
    epsp = Vpbe_getSoluteDiel(pbe);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;

    /* Define the total domain size */
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;

    /* Define the min/max dimensions */
    xmin = thee->pmgp->xcent - (xlen/2.0);
    ymin = thee->pmgp->ycent - (ylen/2.0);
    zmin = thee->pmgp->zcent - (zlen/2.0);
```

```
    xmax = thee->pmgp->xcent + (xlen/2.0);
    ymax = thee->pmgp->ycent + (ylen/2.0);
    zmax = thee->pmgp->zcent + (zlen/2.0);

    /* Reset the arrays */
 ntot = nx*ny*nz;
    for (i=0; i<ntot; i++) {
        thee->epsx[i] = epsw;
        thee->epsy[i] = epsw;
        thee->epsz[i] = epsw;
    }

    /* Loop through the atoms and set a{123}cf = 0.0 (inaccessible)
     * if a point is inside the solvent-inflated van der Waals radii */
#pragma omp parallel for default(shared) private(iatom,atom,apos,arad)
    for (iatom=0; iatom<Valist_getNumberAtoms(alist);
iatom++) {

        atom = Valist_getAtom(alist, iatom);
        apos = Vatom_getPosition(atom);
        arad = Vatom_getRadius(atom);

        /* Make sure we're on the grid */
        if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
            (apos[1]<=ymin) || (apos[1]>=ymax)  || \
            (apos[2]<=zmin) || (apos[2]>=zmax)) {
   if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
    (thee->pmgp->bcfl != BCFL_MAP)) {
                Vnm_print(2, "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f,\
 %4.3f) is off the mesh (ignoring):\n",
                  iatom, apos[0], apos[1], apos[2]);
                Vnm_print(2, "Vpmg_fillco:  xmin = %g, xmax = %g\n",
                  xmin, xmax);
                Vnm_print(2, "Vpmg_fillco:  ymin = %g, ymax = %g\n",
                  ymin, ymax);
                Vnm_print(2, "Vpmg_fillco:  zmin = %g, zmax = %g\n",
                  zmin, zmax);
            }
            fflush(stderr);

        } else { /* if we're on the mesh */

            if (arad > VSMALL) {
                /* Mark x-shifted dielectric */
                markSphere((arad+srad), apos,
                        nx, ny, nz,
                        hx, hy, hzed,
                        (xmin+0.5*hx), ymin, zmin,
                        thee->epsx, epsp);

                /* Mark y-shifted dielectric */
                markSphere((arad+srad), apos,
                        nx, ny, nz,
                        hx, hy, hzed,
                        xmin, (ymin+0.5*hy), zmin,
                        thee->epsy, epsp);

                /* Mark z-shifted dielectric */
                markSphere((arad+srad), apos,
                        nx, ny, nz,
                        hx, hy, hzed,
                        xmin, ymin, (zmin+0.5*hzed),
                        thee->epsz, epsp);
            }

        } /* endif (on the mesh) */
    } /* endfor (over all atoms) */

 area = Vacc_SASA(acc, srad);

    /* We only need to do the next step for non-zero solvent radii */
    if (srad > VSMALL) {

        /* Now loop over the solvent accessible surface points */

#pragma omp parallel for default(shared)
 private(iatom,atom,area,asurf,ipt,position)
        for (iatom=0; iatom<Valist_getNumberAtoms(alist);
iatom++) {
            atom = Valist_getAtom(alist, iatom);
   area = Vacc_atomSASA(acc, srad, atom);
```

```
    if (area > 0.0 ) {
     asurf = Vacc_atomSASPoints(acc, srad, atom);

     /* Use each point on the SAS to reset the solvent accessibility */
     /* TODO:  Make sure we're not still wasting time here. */
     for (ipt=0; ipt<(asurf->npts); ipt++) {

      position[0] = asurf->xpts[ipt];
      position[1] = asurf->ypts[ipt];
      position[2] = asurf->zpts[ipt];

      /* Mark x-shifted dielectric */
      markSphere(srad, position,
           nx, ny, nz,
           hx, hy, hzed,
           (xmin+0.5*hx), ymin, zmin,
           thee->epsx, epsw);

      /* Mark y-shifted dielectric */
      markSphere(srad, position,
           nx, ny, nz,
           hx, hy, hzed,
           xmin, (ymin+0.5*hy), zmin,
           thee->epsy, epsw);

      /* Mark z-shifted dielectric */
      markSphere(srad, position,
           nx, ny, nz,
           hx, hy, hzed,
           xmin, ymin, (zmin+0.5*hzed),
           thee->epsz, epsw);

     }
    }
         }
     }
}

VPRIVATE void fillcoCoefMolDielSmooth(Vpmg *thee) {

   /* This function smoothes using a 9 point method based on
      Bruccoleri, et al. J Comput Chem 18 268-276 (1997).  The nine points
      used are the shifted grid point and the 8 points that are 1/sqrt(2)
      grid spacings away.  The harmonic mean of the 9 points is then used to
      find the overall dielectric value for the point in question. The use of
      this function assumes that the non-smoothed values were placed in the
      dielectric arrays by the fillcoCoefMolDielNoSmooth function.*/

     Vpbe *pbe;
     double frac, epsw;
     int i, j, k, nx, ny, nz, numpts;

     /* Mesh info */
     nx = thee->pmgp->nx;
     ny = thee->pmgp->ny;
     nz = thee->pmgp->nz;

     pbe = thee->pbe;
     epsw = Vpbe_getSolventDiel(pbe);

     /* Copy the existing diel arrays to work arrays */
     for (i=0; i<(nx*ny*nz); i++) {
         thee->a1cf[i] = thee->epsx[i];
         thee->a2cf[i] = thee->epsy[i];
         thee->a3cf[i] = thee->epsz[i];
         thee->epsx[i] = epsw;
         thee->epsy[i] = epsw;
         thee->epsz[i] = epsw;
     }

     /* Smooth the dielectric values */
     for (i=0; i<nx; i++) {
         for (j=0; j<ny; j++) {
             for (k=0; k<nz; k++) {

                 /* Get the 8 points that are 1/sqrt(2) grid spacings away */

                 /* Points for the X-shifted array */
                 frac = 1.0/thee->a1cf[IJK(i,j,k)];
                 frac += 1.0/thee->a2cf[IJK(i,j,k)];
                 frac += 1.0/thee->a3cf[IJK(i,j,k)];
```

```
                numpts = 3;

                if (j > 0) {
                    frac += 1.0/thee->a2cf[IJK(i,j-1,k)];
                    numpts += 1;
                }
                if (k > 0) {
                    frac += 1.0/thee->a3cf[IJK(i,j,k-1)];
                    numpts += 1;
                }
                if (i < (nx-1)){
                    frac += 1.0/thee->a2cf[IJK(i+1,j,k)];
                    frac += 1.0/thee->a3cf[IJK(i+1,j,k)];
                    numpts += 2;
                    if (j > 0) {
                        frac += 1.0/thee->a2cf[IJK(i+1,j-1,k)];
                        numpts += 1;
                    }
                    if (k > 0) {
                        frac += 1.0/thee->a3cf[IJK(i+1,j,k-1)];
                        numpts += 1;
                    }
                }
                thee->epsx[IJK(i,j,k)] = numpts/frac;

                /* Points for the Y-shifted array */
                frac = 1.0/thee->a2cf[IJK(i,j,k)];
                frac += 1.0/thee->a1cf[IJK(i,j,k)];
                frac += 1.0/thee->a3cf[IJK(i,j,k)];
                numpts = 3;

                if (i > 0) {
                    frac += 1.0/thee->a1cf[IJK(i-1,j,k)];
                    numpts += 1;
                }
                if (k > 0) {
                    frac += 1.0/thee->a3cf[IJK(i,j,k-1)];
                    numpts += 1;
                }
                if (j < (ny-1)){
                    frac += 1.0/thee->a1cf[IJK(i,j+1,k)];
                    frac += 1.0/thee->a3cf[IJK(i,j+1,k)];
                    numpts += 2;
                    if (i > 0) {
                        frac += 1.0/thee->a1cf[IJK(i-1,j+1,k)];
                        numpts += 1;
                    }
                    if (k > 0) {
                        frac += 1.0/thee->a3cf[IJK(i,j+1,k-1)];
                        numpts += 1;
                    }
                }
                thee->epsy[IJK(i,j,k)] = numpts/frac;

                /* Points for the Z-shifted array */
                frac = 1.0/thee->a3cf[IJK(i,j,k)];
                frac += 1.0/thee->a1cf[IJK(i,j,k)];
                frac += 1.0/thee->a2cf[IJK(i,j,k)];
                numpts = 3;

                if (i > 0) {
                    frac += 1.0/thee->a1cf[IJK(i-1,j,k)];
                    numpts += 1;
                }
                if (j > 0) {
                    frac += 1.0/thee->a2cf[IJK(i,j-1,k)];
                    numpts += 1;
                }
                if (k < (nz-1)){
                    frac += 1.0/thee->a1cf[IJK(i,j,k+1)];
                    frac += 1.0/thee->a2cf[IJK(i,j,k+1)];
                    numpts += 2;
                    if (i > 0) {
                        frac += 1.0/thee->a1cf[IJK(i-1,j,k+1)];
                        numpts += 1;
                    }
                    if (j > 0) {
                        frac += 1.0/thee->a2cf[IJK(i,j-1,k+1)];
                        numpts += 1;
                    }
                }
```

```
                thee->epsz[IJK(i,j,k)] = numpts/frac;
            }
        }
    }
}


VPRIVATE void fillcoCoefSpline(Vpmg *thee) {

    Valist *alist;
    Vpbe *pbe;
    Vatom *atom;
    double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr, dist2;
    double xlen, ylen, zlen, position[3], itot, stot, ictot, ictot2, sctot;
    double irad, dx, dy, dz, epsw, epsp, w2i;
    double hx, hy, hzed, *apos, arad, sctot2;
    double dx2, dy2, dz2, stot2, itot2, rtot, rtot2, splineWin, w3i;
    double dist, value, sm, sm2;
    int i, j, k, nx, ny, nz, iatom;
    int imin, imax, jmin, jmax, kmin, kmax;

    VASSERT(thee != VNULL);
    splineWin = thee->splineWin;
    w2i = 1.0/(splineWin*splineWin);
    w3i = 1.0/(splineWin*splineWin*splineWin);

    /* Get PBE info */
    pbe = thee->pbe;
    alist = pbe->alist;
    irad = Vpbe_getMaxIonRadius(pbe);
    ionstr = Vpbe_getBulkIonicStrength(pbe);
    epsw = Vpbe_getSolventDiel(pbe);
    epsp = Vpbe_getSoluteDiel(pbe);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;

    /* Define the total domain size */
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;

    /* Define the min/max dimensions */
    xmin = thee->pmgp->xcent - (xlen/2.0);
    ymin = thee->pmgp->ycent - (ylen/2.0);
    zmin = thee->pmgp->zcent - (zlen/2.0);
    xmax = thee->pmgp->xcent + (xlen/2.0);
    ymax = thee->pmgp->ycent + (ylen/2.0);
    zmax = thee->pmgp->zcent + (zlen/2.0);

    /* This is a floating point parameter related to the non-zero nature of the
     * bulk ionic strength.  If the ionic strength is greater than zero; this
     * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
     * Otherwise, this parameter is set to 0.0 */
    if (ionstr > VPMGSMALL) ionmask = 1.0;
    else ionmask = 0.0;

    /* Reset the kappa, epsx, epsy, and epsz arrays */
    for (i=0; i<(nx*ny*nz); i++) {
        thee->kappa[i] = 1.0;
        thee->epsx[i] = 1.0;
        thee->epsy[i] = 1.0;
        thee->epsz[i] = 1.0;
    }

    /* Loop through the atoms and do assign the dielectric */
    for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {

        atom = Valist_getAtom(alist, iatom);
        apos = Vatom_getPosition(atom);
        arad = Vatom_getRadius(atom);

        /* Make sure we're on the grid */
```

```
        if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
            (apos[1]<=ymin) || (apos[1]>=ymax)  || \
            (apos[2]<=zmin) || (apos[2]>=zmax)) {
   if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
    (thee->pmgp->bcfl != BCFL_MAP)) {
                Vnm_print(2, "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f,\
 %4.3f) is off the mesh (ignoring):\n",
                  iatom, apos[0], apos[1], apos[2]);
                Vnm_print(2, "Vpmg_fillco:    xmin = %g, xmax = %g\n",
                  xmin, xmax);
                Vnm_print(2, "Vpmg_fillco:    ymin = %g, ymax = %g\n",
                  ymin, ymax);
                Vnm_print(2, "Vpmg_fillco:    zmin = %g, zmax = %g\n",
                  zmin, zmax);
            }
            fflush(stderr);

        } else if (arad > VPMGSMALL ) { /* if we're on the mesh */

            /* Convert the atom position to grid reference frame */
            position[0] = apos[0] - xmin;
            position[1] = apos[1] - ymin;
            position[2] = apos[2] - zmin;

            /* MARK ION ACCESSIBILITY AND DIELECTRIC VALUES FOR LATER
             * ASSIGNMENT (Steps #1-3) */
            itot = irad + arad + splineWin;
            itot2 = VSQR(itot);
            ictot = VMAX2(0, (irad + arad - splineWin));
            ictot2 = VSQR(ictot);
            stot = arad + splineWin;
            stot2 = VSQR(stot);
            sctot = VMAX2(0, (arad - splineWin));
            sctot2 = VSQR(sctot);

           /* We'll search over grid points which are in the greater of
             * these two radii */
            rtot = VMAX2(itot, stot);
            rtot2 = VMAX2(itot2, stot2);
            dx = rtot + 0.5*hx;
            dy = rtot + 0.5*hy;
            dz = rtot + 0.5*hzed;
            imin = VMAX2(0,(int)floor((position[0] - dx)/hx));
            imax = VMIN2(nx-1,(int)ceil((position[0] + dx)/hx));
            jmin = VMAX2(0,(int)floor((position[1] - dy)/hy));
            jmax = VMIN2(ny-1,(int)ceil((position[1] + dy)/hy));
            kmin = VMAX2(0,(int)floor((position[2] - dz)/hzed));
            kmax = VMIN2(nz-1,(int)ceil((position[2] + dz)/hzed));
            for (i=imin; i<=imax; i++) {
                dx2 = VSQR(position[0] - hx*i);
                for (j=jmin; j<=jmax; j++) {
                    dy2 = VSQR(position[1] - hy*j);
                    for (k=kmin; k<=kmax; k++) {
                        dz2 = VSQR(position[2] - k*hzed);

                        /* ASSIGN CCF */
                        if (thee->kappa[IJK(i,j,k)] > VPMGSMALL)
 {
                            dist2 = dz2 + dy2 + dx2;
                            if (dist2 >= itot2) {
                                ;
                            }
                            if (dist2 <= ictot2) {
                                thee->kappa[IJK(i,j,k)] = 0.0;
                            }
                            if ((dist2 < itot2) && (dist2 > ictot2)) {
                                dist = VSQRT(dist2);
                                sm = dist - (arad + irad) + splineWin;
                                sm2 = VSQR(sm);
                                value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
                                thee->kappa[IJK(i,j,k)] *= value;
                            }
                        }

                        /* ASSIGN A1CF */
                        if (thee->epsx[IJK(i,j,k)] > VPMGSMALL) {
                            dist2 = dz2+dy2+VSQR(position[0]-(i+0.5)*hx);
                            if (dist2 >= stot2) {
                                thee->epsx[IJK(i,j,k)] *= 1.0;
                            }
                            if (dist2 <= sctot2) {
```

```
                                thee->epsx[IJK(i,j,k)] = 0.0;
                            }
                            if ((dist2 > sctot2) && (dist2 < stot2)) {
                                dist = VSQRT(dist2);
                                sm = dist - arad + splineWin;
                                sm2 = VSQR(sm);
                                value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
                                thee->epsx[IJK(i,j,k)] *= value;
                            }
                        }

                        /* ASSIGN A2CF */
                        if (thee->epsy[IJK(i,j,k)] > VPMGSMALL) {
                            dist2 = dz2+dx2+VSQR(position[1]-(j+0.5)*hy);
                            if (dist2 >= stot2) {
                                thee->epsy[IJK(i,j,k)] *= 1.0;
                            }
                            if (dist2 <= sctot2) {
                                thee->epsy[IJK(i,j,k)] = 0.0;
                            }
                            if ((dist2 > sctot2) && (dist2 < stot2)) {
                                dist = VSQRT(dist2);
                                sm = dist - arad + splineWin;
                                sm2 = VSQR(sm);
                                value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
                                thee->epsy[IJK(i,j,k)] *= value;
                            }
                        }

                        /* ASSIGN A3CF */
                        if (thee->epsz[IJK(i,j,k)] > VPMGSMALL) {
                            dist2 = dy2+dx2+VSQR(position[2]-(k+0.5)*hzed);
                            if (dist2 >= stot2) {
                                thee->epsz[IJK(i,j,k)] *= 1.0;
                            }
                            if (dist2 <= sctot2) {
                                thee->epsz[IJK(i,j,k)] = 0.0;
                            }
                            if ((dist2 > sctot2) && (dist2 < stot2)) {
                                dist = VSQRT(dist2);
                                sm = dist - arad + splineWin;
                                sm2 = VSQR(sm);
                                value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
                                thee->epsz[IJK(i,j,k)] *= value;
                            }
                        }


                    } /* k loop */
                } /* j loop */
            } /* i loop */
        } /* endif (on the mesh) */
    } /* endfor (over all atoms) */

    Vnm_print(0, "Vpmg_fillco:  filling coefficient arrays\n");
    /* Interpret markings and fill the coefficient arrays */
    for (k=0; k<nz; k++) {
        for (j=0; j<ny; j++) {
            for (i=0; i<nx; i++) {

                thee->kappa[IJK(i,j,k)] = ionmask*thee->kappa[IJK(i,j,k)];
                thee->epsx[IJK(i,j,k)] = (epsw-epsp)*thee->epsx[IJK(i,j,k)]
                  + epsp;
                thee->epsy[IJK(i,j,k)] = (epsw-epsp)*thee->epsy[IJK(i,j,k)]
                  + epsp;
                thee->epsz[IJK(i,j,k)] = (epsw-epsp)*thee->epsz[IJK(i,j,k)]
                  + epsp;

            } /* i loop */
        } /* j loop */
    } /* k loop */

}

VPRIVATE void fillcoCoef(Vpmg *thee) {

    VASSERT(thee != VNULL);
```

```

    if (thee->useDielXMap || thee->useDielYMap ||
  thee->useDielZMap || thee->useKappaMap) {
        fillcoCoefMap(thee);
        return;
    }

    switch(thee->surfMeth) {
        case VSM_MOL:
            Vnm_print(0, "fillcoCoef:  Calling fillcoCoefMol...\n");
            fillcoCoefMol(thee);
            break;
        case VSM_MOLSMOOTH:
            Vnm_print(0, "fillcoCoef:  Calling fillcoCoefMol...\n");
            fillcoCoefMol(thee);
            break;
        case VSM_SPLINE:
            Vnm_print(0, "fillcoCoef:  Calling fillcoCoefSpline...\n");
            fillcoCoefSpline(thee);
            break;
  case VSM_SPLINE3:
            Vnm_print(0, "fillcoCoef:  Calling fillcoCoefSpline3...\n");
            fillcoCoefSpline3(thee);
            break;
        case VSM_SPLINE4:
            Vnm_print(0, "fillcoCoef:  Calling fillcoCoefSpline4...\n");
            fillcoCoefSpline4(thee);
            break;
        default:
            Vnm_print(2, "fillcoCoef:  Invalid surfMeth (%d)!\n",
              thee->surfMeth);
            VASSERT(0);
            break;
    }
}


VPRIVATE Vrc_Codes fillcoCharge(Vpmg *thee) {

 Vrc_Codes rc;

    VASSERT(thee != VNULL);

    if (thee->useChargeMap) {
        return fillcoChargeMap(thee);
    }

    switch(thee->chargeMeth) {
        case VCM_TRIL:
            Vnm_print(0, "fillcoCharge:  Calling fillcoChargeSpline1...\n");
            fillcoChargeSpline1(thee);
            break;
        case VCM_BSPL2:
            Vnm_print(0, "fillcoCharge:  Calling fillcoChargeSpline2...\n");
            fillcoChargeSpline2(thee);
            break;
        case VCM_BSPL4:
            switch (thee->chargeSrc) {
                case VCM_CHARGE:
                    Vnm_print(0, "fillcoCharge: Calling
    fillcoPermanentMultipole...\n");
                    fillcoPermanentMultipole(thee);
                    break;
#if defined(WITH_TINKER)
                case VCM_PERMANENT:
                    Vnm_print(0, "fillcoCharge: Calling
    fillcoPermanentMultipole...\n");
                    fillcoPermanentMultipole(thee);
                    break;
                case VCM_INDUCED:
                    Vnm_print(0, "fillcoCharge: Calling fillcoInducedDipole...
    \n");
                    fillcoInducedDipole(thee);
                    break;
                case VCM_NLINDUCED:
                     Vnm_print(0, "fillcoCharge: Calling
    fillcoNLInducedDipole...\n");
                     fillcoNLInducedDipole(thee);
                     break;
#endif /* if defined(WITH_TINKER) */
                default:
```

```
                    Vnm_print(2, "fillcoCharge:  Invalid chargeSource (%d)!\n",
                      thee->chargeSrc);
                    return VRC_FAILURE;
                    break;
            }
            break;
        default:
            Vnm_print(2, "fillcoCharge:  Invalid chargeMeth (%d)!\n",
              thee->chargeMeth);
            return VRC_FAILURE;
            break;
    }

 return VRC_SUCCESS;
}

VPRIVATE Vrc_Codes fillcoChargeMap(Vpmg *thee) {

    Vpbe *pbe;
    double position[3], charge, zmagic, hx, hy, hzed;
    int i, j, k, nx, ny, nz, rc;


    VASSERT(thee != VNULL);

    /* Get PBE info */
    pbe = thee->pbe;
    zmagic = Vpbe_getZmagic(pbe);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;

    /* Reset the charge array */
    for (i=0; i<(nx*ny*nz); i++) thee->charge[i] = 0.0;

    /* Fill in the source term (atomic charges) */
    Vnm_print(0, "Vpmg_fillco:  filling in source term.\n");
    for (k=0; k<nz; k++) {
        for (j=0; j<ny; j++) {
            for (i=0; i<nx; i++) {
                position[0] = thee->xf[i];
                position[1] = thee->yf[j];
                position[2] = thee->zf[k];
    rc = Vgrid_value(thee->chargeMap, position, &charge);
    if (!rc) {
     Vnm_print(2, "fillcoChargeMap:  Error -- fell off of charge map at (%g,
  %g, %g)!\n",
        position[0], position[1], position[2]);
     return VRC_FAILURE;
    }
    /* Scale the charge to internal units */
                charge = charge*zmagic;
                thee->charge[IJK(i,j,k)] = charge;
            }
        }
    }

 return VRC_SUCCESS;
}

VPRIVATE void fillcoChargeSpline1(Vpmg *thee) {

    Valist *alist;
    Vpbe *pbe;
    Vatom *atom;
    double xmin, xmax, ymin, ymax, zmin, zmax;
    double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;
    double charge, dx, dy, dz, zmagic, hx, hy, hzed, *apos;
    int i, nx, ny, nz, iatom, ihi, ilo, jhi, jlo, khi, klo;


    VASSERT(thee != VNULL);

    /* Get PBE info */
    pbe = thee->pbe;
    alist = pbe->alist;
```

```
    zmagic = Vpbe_getZmagic(pbe);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;

    /* Define the total domain size */
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;

    /* Define the min/max dimensions */
    xmin = thee->pmgp->xcent - (xlen/2.0);
    ymin = thee->pmgp->ycent - (ylen/2.0);
    zmin = thee->pmgp->zcent - (zlen/2.0);
    xmax = thee->pmgp->xcent + (xlen/2.0);
    ymax = thee->pmgp->ycent + (ylen/2.0);
    zmax = thee->pmgp->zcent + (zlen/2.0);

    /* Reset the charge array */
    for (i=0; i<(nx*ny*nz); i++) thee->charge[i] = 0.0;

    /* Fill in the source term (atomic charges) */
    Vnm_print(0, "Vpmg_fillco:  filling in source term.\n");
    for (iatom=0; iatom<Valist_getNumberAtoms(alist);
iatom++) {

        atom = Valist_getAtom(alist, iatom);
        apos = Vatom_getPosition(atom);
        charge = Vatom_getCharge(atom);

        /* Make sure we're on the grid */
        if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
            (apos[1]<=ymin) || (apos[1]>=ymax)  || \
            (apos[2]<=zmin) || (apos[2]>=zmax)) {
   if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
    (thee->pmgp->bcfl != BCFL_MAP)) {
                Vnm_print(2, "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f, \
%4.3f) is off the mesh (ignoring):\n",
                  iatom, apos[0], apos[1], apos[2]);
                Vnm_print(2, "Vpmg_fillco:    xmin = %g, xmax = %g\n",
                  xmin, xmax);
                Vnm_print(2, "Vpmg_fillco:    ymin = %g, ymax = %g\n",
                  ymin, ymax);
                Vnm_print(2, "Vpmg_fillco:    zmin = %g, zmax = %g\n",
                  zmin, zmax);
            }
            fflush(stderr);
        } else {

            /* Convert the atom position to grid reference frame */
            position[0] = apos[0] - xmin;
            position[1] = apos[1] - ymin;
            position[2] = apos[2] - zmin;

            /* Scale the charge to be a delta function */
            charge = charge*zmagic/(hx*hy*hzed);

            /* Figure out which vertices we're next to */
            ifloat = position[0]/hx;
            jfloat = position[1]/hy;
            kfloat = position[2]/hzed;

            ihi = (int)ceil(ifloat);
            ilo = (int)floor(ifloat);
            jhi = (int)ceil(jfloat);
            jlo = (int)floor(jfloat);
            khi = (int)ceil(kfloat);
            klo = (int)floor(kfloat);

            /* Now assign fractions of the charge to the nearby verts */
            dx = ifloat - (double)(ilo);
            dy = jfloat - (double)(jlo);
            dz = kfloat - (double)(klo);
            thee->charge[IJK(ihi,jhi,khi)] += (dx*dy*dz*charge);
            thee->charge[IJK(ihi,jlo,khi)] += (dx*(1.0-dy)*dz*charge);
            thee->charge[IJK(ihi,jhi,klo)] += (dx*dy*(1.0-dz)*charge);
```

```
            thee->charge[IJK(ihi,jlo,klo)] += (dx*(1.0-dy)*(1.0-dz)*charge);
            thee->charge[IJK(ilo,jhi,khi)] += ((1.0-dx)*dy*dz *charge);
            thee->charge[IJK(ilo,jlo,khi)] += ((1.0-dx)*(1.0-dy)*dz *charge);
            thee->charge[IJK(ilo,jhi,klo)] += ((1.0-dx)*dy*(1.0-dz)*charge);
            thee->charge[IJK(ilo,jlo,klo)] += ((1.0-dx)*(1.0-dy)*(1.0-dz)*charge);
        } /* endif (on the mesh) */
    } /* endfor (each atom) */
}

VPRIVATE double bspline2(double x) {

    double m2m, m2, m3;

    if ((x >= 0.0) && (x <= 2.0)) m2m = 1.0 - VABS(x - 1.0);
    else m2m = 0.0;
    if ((x >= 1.0) && (x <= 3.0)) m2 = 1.0 - VABS(x - 2.0);
    else m2 = 0.0;

    if ((x >= 0.0) && (x <= 3.0)) m3 = 0.5*x*m2m + 0.5*(3.0-x)*m2;
    else m3 = 0.0;

    return m3;

}

VPRIVATE double dbspline2(double x) {

    double m2m, m2, dm3;

    if ((x >= 0.0) && (x <= 2.0)) m2m = 1.0 - VABS(x - 1.0);
    else m2m = 0.0;
    if ((x >= 1.0) && (x <= 3.0)) m2 = 1.0 - VABS(x - 2.0);
    else m2 = 0.0;

    dm3 = m2m - m2;

    return dm3;

}


VPRIVATE void fillcoChargeSpline2(Vpmg *thee) {

    Valist *alist;
    Vpbe *pbe;
    Vatom *atom;
    double xmin, xmax, ymin, ymax, zmin, zmax, zmagic;
    double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;
    double charge, hx, hy, hzed, *apos, mx, my, mz;
    int i, ii, jj, kk, nx, ny, nz, iatom;
    int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;


    VASSERT(thee != VNULL);

    /* Get PBE info */
    pbe = thee->pbe;
    alist = pbe->alist;
    zmagic = Vpbe_getZmagic(pbe);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;

    /* Define the total domain size */
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;

    /* Define the min/max dimensions */
    xmin = thee->pmgp->xcent - (xlen/2.0);
    ymin = thee->pmgp->ycent - (ylen/2.0);
    zmin = thee->pmgp->zcent - (zlen/2.0);
```

```
    xmax = thee->pmgp->xcent + (xlen/2.0);
    ymax = thee->pmgp->ycent + (ylen/2.0);
    zmax = thee->pmgp->zcent + (zlen/2.0);

    /* Reset the charge array */
    for (i=0; i<(nx*ny*nz); i++) thee->charge[i] = 0.0;

    /* Fill in the source term (atomic charges) */
    Vnm_print(0, "Vpmg_fillco:  filling in source term.\n");
    for (iatom=0; iatom<Valist_getNumberAtoms(alist);
iatom++) {

        atom = Valist_getAtom(alist, iatom);
        apos = Vatom_getPosition(atom);
        charge = Vatom_getCharge(atom);

        /* Make sure we're on the grid */
        if ((apos[0]<=(xmin-hx)) || (apos[0]>=(xmax+hx))  || \
            (apos[1]<=(ymin-hy)) || (apos[1]>=(ymax+hy))  || \
            (apos[2]<=(zmin-hzed)) || (apos[2]>=(zmax+hzed))) {
   if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
    (thee->pmgp->bcfl != BCFL_MAP)) {
                Vnm_print(2, "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f, \
%4.3f) is off the mesh (for cubic splines!!) (ignoring this atom):\n",
                  iatom, apos[0], apos[1], apos[2]);
                Vnm_print(2, "Vpmg_fillco:    xmin = %g, xmax = %g\n",
                  xmin, xmax);
                Vnm_print(2, "Vpmg_fillco:    ymin = %g, ymax = %g\n",
                  ymin, ymax);
                Vnm_print(2, "Vpmg_fillco:    zmin = %g, zmax = %g\n",
                  zmin, zmax);
            }
            fflush(stderr);
        } else {

            /* Convert the atom position to grid reference frame */
            position[0] = apos[0] - xmin;
            position[1] = apos[1] - ymin;
            position[2] = apos[2] - zmin;

            /* Scale the charge to be a delta function */
            charge = charge*zmagic/(hx*hy*hzed);

            /* Figure out which vertices we're next to */
            ifloat = position[0]/hx;
            jfloat = position[1]/hy;
            kfloat = position[2]/hzed;

            ip1   = (int)ceil(ifloat);
            ip2   = ip1 + 1;
            im1   = (int)floor(ifloat);
            im2   = im1 - 1;
            jp1   = (int)ceil(jfloat);
            jp2   = jp1 + 1;
            jm1   = (int)floor(jfloat);
            jm2   = jm1 - 1;
            kp1   = (int)ceil(kfloat);
            kp2   = kp1 + 1;
            km1   = (int)floor(kfloat);
            km2   = km1 - 1;

            /* This step shouldn't be necessary, but it saves nasty debugging
             * later on if something goes wrong */
            ip2 = VMIN2(ip2,nx-1);
            ip1 = VMIN2(ip1,nx-1);
            im1 = VMAX2(im1,0);
            im2 = VMAX2(im2,0);
            jp2 = VMIN2(jp2,ny-1);
            jp1 = VMIN2(jp1,ny-1);
            jm1 = VMAX2(jm1,0);
            jm2 = VMAX2(jm2,0);
            kp2 = VMIN2(kp2,nz-1);
            kp1 = VMIN2(kp1,nz-1);
            km1 = VMAX2(km1,0);
            km2 = VMAX2(km2,0);

            /* Now assign fractions of the charge to the nearby verts */
            for (ii=im2; ii<=ip2; ii++) {
                mx = bspline2(VFCHI(ii,ifloat));
                for (jj=jm2; jj<=jp2; jj++) {
                    my = bspline2(VFCHI(jj,jfloat));
```

```
                    for (kk=km2; kk<=kp2; kk++) {
                        mz = bspline2(VFCHI(kk,kfloat));
                        thee->charge[IJK(ii,jj,kk)] += (charge*mx*my*mz);
                    }
                }
            }

        } /* endif (on the mesh) */
    } /* endfor (each atom) */
}

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
                       ) {

    Vpbe *pbe;
    double xmin,
           xmax,
           ymin,
           ymax,
           zmin,
           zmax,
           xlen,
           ylen,
           zlen,
           hx,
           hy,
           hzed,
           epsw,
           epsp,
           ionstr;
    int i,
        nx,
        ny,
        nz,
        islap;
 Vrc_Codes rc;

    if (thee == VNULL) {
        Vnm_print(2, "Vpmg_fillco:  got NULL thee!\n");
        return 0;
    }

    thee->surfMeth = surfMeth;
    thee->splineWin = splineWin;
    thee->chargeMeth = chargeMeth;
    thee->useDielXMap = useDielXMap;
    if (thee->useDielXMap) thee->dielXMap = dielXMap;
    thee->useDielYMap = useDielYMap;
    if (thee->useDielYMap) thee->dielYMap = dielYMap;
    thee->useDielZMap = useDielZMap;
    if (thee->useDielZMap) thee->dielZMap = dielZMap;
    thee->useKappaMap = useKappaMap;
    if (thee->useKappaMap) thee->kappaMap = kappaMap;
 thee->usePotMap = usePotMap;
    if (thee->usePotMap) thee->potMap = potMap;
    thee->useChargeMap = useChargeMap;
    if (thee->useChargeMap) thee->chargeMap = chargeMap;

    /* Get PBE info */
    pbe = thee->pbe;
    ionstr = Vpbe_getBulkIonicStrength(pbe);
    epsw = Vpbe_getSolventDiel(pbe);
    epsp = Vpbe_getSoluteDiel(pbe);

    /* Mesh info */
    nx = thee->pmgp->nx;
```

```
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;

    /* Define the total domain size */
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;

    /* Define the min/max dimensions */
    xmin = thee->pmgp->xcent - (xlen/2.0);
    thee->pmgp->xmin = xmin;
    ymin = thee->pmgp->ycent - (ylen/2.0);
    thee->pmgp->ymin = ymin;
    zmin = thee->pmgp->zcent - (zlen/2.0);
    thee->pmgp->zmin = zmin;
    xmax = thee->pmgp->xcent + (xlen/2.0);
    thee->pmgp->xmax = xmax;
    ymax = thee->pmgp->ycent + (ylen/2.0);
    thee->pmgp->ymax = ymax;
    zmax = thee->pmgp->zcent + (zlen/2.0);
    thee->pmgp->zmax = zmax;
    thee->rparm[2] = xmin;
    thee->rparm[3] = xmax;
    thee->rparm[4] = ymin;
    thee->rparm[5] = ymax;
    thee->rparm[6] = zmin;
    thee->rparm[7] = zmax;

    /* This is a flag that gets set if the operator is a simple Laplacian;
     * i.e., in the case of a homogenous dielectric and zero ionic strength
  * The operator cannot be a simple Laplacian if maps are read in. */
 if(thee->useDielXMap || thee->useDielYMap || thee->useDielZMap ||
    thee->useKappaMap || thee->usePotMap){
  islap = 0;
 }else if ( (ionstr < VPMGSMALL) && (VABS(epsp-epsw) < VPMGSMALL) ){
  islap = 1;
 }else{
  islap = 0;
 }

    /* Fill the mesh point coordinate arrays */
    for (i=0; i<nx; i++) thee->xf[i] = xmin + i*hx;
    for (i=0; i<ny; i++) thee->yf[i] = ymin + i*hy;
    for (i=0; i<nz; i++) thee->zf[i] = zmin + i*hzed;

    /* Reset the tcf array */
    for (i=0; i<(nx*ny*nz); i++) thee->tcf[i] = 0.0;

    /* Fill in the source term (atomic charges) */
    Vnm_print(0, "Vpmg_fillco:  filling in source term.\n");
    rc = fillcoCharge(thee);
 switch(rc) {
  case VRC_SUCCESS:
   break;
  case VRC_WARNING:
   Vnm_print(2, "Vpmg_fillco:  non-fatal errors while filling charge map!\n");
   break;
  case VRC_FAILURE:
   Vnm_print(2, "Vpmg_fillco:  fatal errors while filling charge map!\n");
   return 0;
   break;
 }

    /* THE FOLLOWING NEEDS TO BE DONE IF WE'RE NOT USING A SIMPLE LAPLACIAN
     * OPERATOR */
    if (!islap) {
        Vnm_print(0, "Vpmg_fillco:  marking ion and solvent accessibility.\n");
        fillcoCoef(thee);
        Vnm_print(0, "Vpmg_fillco:  done filling coefficient arrays\n");

    } else { /* else (!islap) ==> It's a Laplacian operator! */

        for (i=0; i<(nx*ny*nz); i++) {
            thee->kappa[i] = 0.0;
            thee->epsx[i] = epsp;
            thee->epsy[i] = epsp;
```

```
            thee->epsz[i] = epsp;
        }

    } /* endif (!islap) */

    /* Fill the boundary arrays (except when focusing, bcfl = 4) */
    if (thee->pmgp->bcfl != BCFL_FOCUS) {
        Vnm_print(0, "Vpmg_fillco:  filling boundary arrays\n");
        bcCalc(thee);
        Vnm_print(0, "Vpmg_fillco:  done filling boundary arrays\n");
    }

    thee->filled = 1;

    return 1;
}


VPUBLIC int Vpmg_force(Vpmg *thee, double *force, int atomID,
  Vsurf_Meth srfm, Vchrg_Meth chgm) {

    int rc = 1;
    double qfF[3];                  /* Charge-field force */
    double dbF[3];                  /* Dielectric boundary force */
    double ibF[3];                  /* Ion boundary force */
    double npF[3];                  /* Non-polar boundary force */

    VASSERT(thee != VNULL);

    rc = rc && Vpmg_dbForce(thee, qfF, atomID, srfm);
    rc = rc && Vpmg_ibForce(thee, dbF, atomID, srfm);
    rc = rc && Vpmg_qfForce(thee, ibF, atomID, chgm);

    force[0] = qfF[0] + dbF[0] + ibF[0];
    force[1] = qfF[1] + dbF[1] + ibF[1];
    force[2] = qfF[2] + dbF[2] + ibF[2];

    return rc;

}

VPUBLIC int Vpmg_ibForce(Vpmg *thee, double *force, int atomID,
  Vsurf_Meth srfm) {

    Valist *alist;
    Vacc *acc;
    Vpbe *pbe;
    Vatom *atom;

    double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
    double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;
    double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
    double izmagic;
    int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;

 /* For nonlinear forces */
 int ichop, nchop, nion, m;
    double ionConc[MAXION], ionRadii[MAXION], ionQ[MAXION], ionstr;

    VASSERT(thee != VNULL);

    acc = thee->pbe->acc;
    atom = Valist_getAtom(thee->pbe->alist, atomID);
    apos = Vatom_getPosition(atom);
    arad = Vatom_getRadius(atom);

    /* Reset force */
    force[0] = 0.0;
    force[1] = 0.0;
    force[2] = 0.0;

    /* Check surface definition */
    if ((srfm != VSM_SPLINE) && (srfm!=VSM_SPLINE3) && (srfm!=VSM_SPLINE4)) {
        Vnm_print(2, "Vpmg_ibForce:  Forces *must* be calculated with \
spline-based surfaces!\n");
        Vnm_print(2, "Vpmg_ibForce:  Skipping ionic boundary force \
calculation!\n");
        return 0;
```

```
    }

    /* If we aren't in the current position, then we're done */
    if (atom->partID == 0) return 1;

    /* Get PBE info */
    pbe = thee->pbe;
    acc = pbe->acc;
    alist = pbe->alist;
    irad = Vpbe_getMaxIonRadius(pbe);
    zkappa2 = Vpbe_getZkappa2(pbe);
    izmagic = 1.0/Vpbe_getZmagic(pbe);

 ionstr = Vpbe_getBulkIonicStrength(pbe);
    Vpbe_getIons(pbe, &nion, ionConc, ionRadii, ionQ);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;
    xmin = thee->pmgp->xmin;
    ymin = thee->pmgp->ymin;
    zmin = thee->pmgp->zmin;
    xmax = thee->pmgp->xmax;
    ymax = thee->pmgp->ymax;
    zmax = thee->pmgp->zmax;

    /* Sanity check: there is no force if there is zero ionic strength */
    if (zkappa2 < VPMGSMALL) {
#ifndef VAPBSQUIET
        Vnm_print(2, "Vpmg_ibForce:  No force for zero ionic strength!\n");
#endif
        return 1;
    }

    /* Make sure we're on the grid */
    if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
      (apos[1]<=ymin) || (apos[1]>=ymax)  || \
      (apos[2]<=zmin) || (apos[2]>=zmax)) {
  if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
   (thee->pmgp->bcfl != BCFL_MAP)) {
            Vnm_print(2, "Vpmg_ibForce:  Atom #%d at (%4.3f, %4.3f, %4.3f) is
 off the mesh (ignoring):\n",
                  atom, apos[0], apos[1], apos[2]);
            Vnm_print(2, "Vpmg_ibForce:    xmin = %g, xmax = %g\n",
              xmin, xmax);
            Vnm_print(2, "Vpmg_ibForce:    ymin = %g, ymax = %g\n",
              ymin, ymax);
            Vnm_print(2, "Vpmg_ibForce:    zmin = %g, zmax = %g\n",
              zmin, zmax);
        }
        fflush(stderr);
    } else {

        /* Convert the atom position to grid reference frame */
        position[0] = apos[0] - xmin;
        position[1] = apos[1] - ymin;
        position[2] = apos[2] - zmin;

        /* Integrate over points within this atom's (inflated) radius */
        rtot = (irad + arad + thee->splineWin);
        rtot2 = VSQR(rtot);
        dx = rtot + 0.5*hx;
        imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
        imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
        for (i=imin; i<=imax; i++) {
            dx2 = VSQR(position[0] - hx*i);
            if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
            else dy = 0.5*hy;
            jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
            jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
            for (j=jmin; j<=jmax; j++) {
                dy2 = VSQR(position[1] - hy*j);
                if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
                else dz = 0.5*hzed;
```

```
                kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
                kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
                for (k=kmin; k<=kmax; k++) {
                    dz2 = VSQR(k*hzed - position[2]);
                    /* See if grid point is inside ivdw radius and set kappa
                     * accordingly (do spline assignment here) */
                    if ((dz2 + dy2 + dx2) <= rtot2) {
                        gpos[0] = i*hx + xmin;
                        gpos[1] = j*hy + ymin;
                        gpos[2] = k*hzed + zmin;

      /* Select the correct function based on the surface definition
       * (now including the 7th order polynomial) */
      Vpmg_splineSelect(srfm,acc, gpos,thee->splineWin, irad, atom, tgrad);

                        if (thee->pmgp->nonlin) {
                            /* Nonlinear forces */
                            fmag = 0.0;
                            nchop = 0;
                            for (m=0; m<nion; m++) {
                                fmag += (thee->kappa[IJK(i,j,k)])*ionConc[m]*(Vcap_exp(-ionQ[m]*thee->u[IJK(i,j,k)], &ichop)-1.0)/ionstr;
                                nchop += ichop;
                            }
      /*          if (nchop > 0) Vnm_print(2, "Vpmg_ibForece:  Chopped EXP %d times!\n", nchop);*/
                            force[0] += (zkappa2*fmag*tgrad[0]);
                            force[1] += (zkappa2*fmag*tgrad[1]);
                            force[2] += (zkappa2*fmag*tgrad[2]);
                        } else {
                            /* Use of bulk factor (zkappa2) OK here becuase
                             * LPBE force approximation */
                            /* NAB -- did we forget a kappa factor here??? */
                            fmag = VSQR(thee->u[IJK(i,j,k)])*(thee->kappa[IJK(i,j,k)]);
                            force[0] += (zkappa2*fmag*tgrad[0]);
                            force[1] += (zkappa2*fmag*tgrad[1]);
                            force[2] += (zkappa2*fmag*tgrad[2]);
                        }
                    }
                } /* k loop */
            } /* j loop */
        } /* i loop */
    }
    force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
    force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
    force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;

    return 1;
}

VPUBLIC int Vpmg_dbForce(Vpmg *thee, double *dbForce, int atomID,
        Vsurf_Meth srfm) {

    Vacc *acc;
    Vpbe *pbe;
    Vatom *atom;

    double *apos, position[3], arad, srad, hx, hy, hzed, izmagic, deps, depsi;
    double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2, epsp;
    double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
    double *u, Hxijk, Hyijk, Hzijk, Hxim1jk, Hyijm1k, Hzijkm1;
    double dHxijk[3], dHyijk[3], dHzijk[3], dHxim1jk[3], dHyijm1k[3];
    double dHzijkm1[3];
    int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;

    VASSERT(thee != VNULL);
    if (!thee->filled) {
        Vnm_print(2, "Vpmg_dbForce:  Need to callVpmg_fillco!\n");
        return 0;
    }

    acc = thee->pbe->acc;
    atom = Valist_getAtom(thee->pbe->alist, atomID);
    apos = Vatom_getPosition(atom);
    arad = Vatom_getRadius(atom);
    srad = Vpbe_getSolventRadius(thee->pbe);
```

```
    /* Reset force */
    dbForce[0] = 0.0;
    dbForce[1] = 0.0;
    dbForce[2] = 0.0;

    /* Check surface definition */
    if ((srfm != VSM_SPLINE) && (srfm!=VSM_SPLINE3) && (srfm!=VSM_SPLINE4)) {
        Vnm_print(2, "Vpmg_dbForce:  Forces *must* be calculated with \spline-based surfaces!\n");
        Vnm_print(2, "Vpmg_dbForce:  Skipping dielectric/apolar boundary \force calculation!\n");
        return 0;
    }


    /* If we aren't in the current position, then we're done */
    if (atom->partID == 0) return 1;

    /* Get PBE info */
    pbe = thee->pbe;
    acc = pbe->acc;
    epsp = Vpbe_getSoluteDiel(pbe);
    epsw = Vpbe_getSolventDiel(pbe);
    kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*Vunit_kb;
    izmagic = 1.0/Vpbe_getZmagic(pbe);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;
    xmin = thee->pmgp->xmin;
    ymin = thee->pmgp->ymin;
    zmin = thee->pmgp->zmin;
    xmax = thee->pmgp->xmax;
    ymax = thee->pmgp->ymax;
    zmax = thee->pmgp->zmax;
    u = thee->u;

    /* Sanity check: there is no force if there is zero ionic strength */
    if (VABS(epsp-epsw) < VPMGSMALL) {
  Vnm_print(0, "Vpmg_dbForce: No force for uniform dielectric!\n");
  return 1;
    }
    deps = (epsw - epsp);
    depsi = 1.0/deps;
    rtot = (arad + thee->splineWin + srad);

    /* Make sure we're on the grid */
    /* Grid checking modified by Matteo Rotter */
    if ((apos[0]<=xmin + rtot) || (apos[0]>=xmax - rtot)  || \
   (apos[1]<=ymin + rtot) || (apos[1]>=ymax - rtot)  || \
   (apos[2]<=zmin + rtot) || (apos[2]>=zmax - rtot)) {
   if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
    (thee->pmgp->bcfl != BCFL_MAP)) {
            Vnm_print(2, "Vpmg_dbForce:  Atom #%d at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):\n",
        atomID, apos[0], apos[1], apos[2]);
            Vnm_print(2, "Vpmg_dbForce:    xmin = %g, xmax = %g\n",
        xmin, xmax);
            Vnm_print(2, "Vpmg_dbForce:    ymin = %g, ymax = %g\n",
        ymin, ymax);
            Vnm_print(2, "Vpmg_dbForce:    zmin = %g, zmax = %g\n",
        zmin, zmax);
         }
         fflush(stderr);
    } else {

         /* Convert the atom position to grid reference frame */
         position[0] = apos[0] - xmin;
         position[1] = apos[1] - ymin;
         position[2] = apos[2] - zmin;

```

```
        /* Integrate over points within this atom's (inflated) radius */
        rtot2 = VSQR(rtot);
        dx = rtot/hx;
        imin = (int)floor((position[0]-rtot)/hx);
        if (imin < 1) {
            Vnm_print(2, "Vpmg_dbForce:  Atom %d off grid!\n", atomID);
            return 0;
        }
        imax = (int)ceil((position[0]+rtot)/hx);
        if (imax > (nx-2)) {
            Vnm_print(2, "Vpmg_dbForce:  Atom %d off grid!\n", atomID);
            return 0;
        }
        jmin = (int)floor((position[1]-rtot)/hy);
        if (jmin < 1) {
            Vnm_print(2, "Vpmg_dbForce:  Atom %d off grid!\n", atomID);
            return 0;
        }
        jmax = (int)ceil((position[1]+rtot)/hy);
        if (jmax > (ny-2)) {
            Vnm_print(2, "Vpmg_dbForce:  Atom %d off grid!\n", atomID);
            return 0;
        }
        kmin = (int)floor((position[2]-rtot)/hzed);
        if (kmin < 1) {
            Vnm_print(2, "Vpmg_dbForce:  Atom %d off grid!\n", atomID);
            return 0;
        }
        kmax = (int)ceil((position[2]+rtot)/hzed);
        if (kmax > (nz-2)) {
            Vnm_print(2, "Vpmg_dbForce:  Atom %d off grid!\n", atomID);
            return 0;
        }
        for (i=imin; i<=imax; i++) {
            for (j=jmin; j<=jmax; j++) {
                for (k=kmin; k<=kmax; k++) {
                    /* i,j,k */
                    gpos[0] = (i+0.5)*hx + xmin;
                    gpos[1] = j*hy + ymin;
                    gpos[2] = k*hzed + zmin;
                    Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;

     /* Select the correct function based on the surface definition
      * (now including the 7th order polynomial) */
     Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin
, 0.,atom, dHxijk);
     /*
      switch (srfm) {
       case VSM_SPLINE :
        Vacc_splineAccGradAtomNorm(acc, gpos, thee->splineWin, 0.,
              atom, dHxijk);
        break;
       case VSM_SPLINE4 :
        Vacc_splineAccGradAtomNorm4(acc, gpos, thee->splineWin, 0.,
              atom, dHxijk);
        break;
       default:
        Vnm_print(2, "Vpmg_dbnbForce: Unknown surface method.\n");
        return;
      }
      */
                    for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
                    gpos[0] = i*hx + xmin;
                    gpos[1] = (j+0.5)*hy + ymin;
                    gpos[2] = k*hzed + zmin;
                    Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;

     /* Select the correct function based on the surface definition
      * (now including the 7th order polynomial) */
     Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin
, 0.,atom, dHyijk);

     for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
                    gpos[0] = i*hx + xmin;
                    gpos[1] = j*hy + ymin;
                    gpos[2] = (k+0.5)*hzed + zmin;
                    Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;

     /* Select the correct function based on the surface definition
      * (now including the 7th order polynomial) */
     Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin
```

```
      , 0.,atom, dHzijk);

      for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
                    /* i-1,j,k */
                    gpos[0] = (i-0.5)*hx + xmin;
                    gpos[1] = j*hy + ymin;
                    gpos[2] = k*hzed + zmin;
                    Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;

      /* Select the correct function based on the surface definition
       * (now including the 7th order polynomial) */
      Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin
      , 0.,atom, dHxim1jk);

      for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
                    /* i,j-1,k */
                    gpos[0] = i*hx + xmin;
                    gpos[1] = (j-0.5)*hy + ymin;
                    gpos[2] = k*hzed + zmin;
                    Hyijm1k = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;

      /* Select the correct function based on the surface definition
       * (now including the 7th order polynomial) */
      Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin
      , 0.,atom, dHyijm1k);

      for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;
                    /* i,j,k-1 */
                    gpos[0] = i*hx + xmin;
                    gpos[1] = j*hy + ymin;
                    gpos[2] = (k-0.5)*hzed + zmin;
                    Hzijkm1 = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;

      /* Select the correct function based on the surface definition
       * (now including the 7th order polynomial) */
      Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin
      , 0.,atom, dHzijkm1);

      for (l=0; l<3; l++) dHzijkm1[l] *= Hzijkm1;
                    /* *** CALCULATE DIELECTRIC BOUNDARY FORCES *** */
                    dbFmag = u[IJK(i,j,k)];
                    tgrad[0] =
       (dHxijk[0]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
        +  dHxim1jk[0]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
       + (dHyijk[0]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
          +  dHyijm1k[0]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
       + (dHzijk[0]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
          + dHzijkm1[0]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
                    tgrad[1] =
       (dHxijk[1]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
        +  dHxim1jk[1]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
       + (dHyijk[1]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
          +  dHyijm1k[1]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
       + (dHzijk[1]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
          + dHzijkm1[1]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
                    tgrad[2] =
       (dHxijk[2]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
        +  dHxim1jk[2]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
       + (dHyijk[2]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
          +  dHyijm1k[2]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
       + (dHzijk[2]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
          + dHzijkm1[2]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
      dbForce[0] += (dbFmag*tgrad[0]);
      dbForce[1] += (dbFmag*tgrad[1]);
      dbForce[2] += (dbFmag*tgrad[2]);

     } /* k loop */
    } /* j loop */
   } /* i loop */

         dbForce[0] = -dbForce[0]*hx*hy*hzed*deps*0.5*izmagic;
         dbForce[1] = -dbForce[1]*hx*hy*hzed*deps*0.5*izmagic;
         dbForce[2] = -dbForce[2]*hx*hy*hzed*deps*0.5*izmagic;
  }

  return 1;
}

VPUBLIC int Vpmg_qfForce(Vpmg *thee, double *force, int atomID,
   Vchrg_Meth chgm) {

```

```
    double tforce[3];

    /* Reset force */
    force[0] = 0.0;
    force[1] = 0.0;
    force[2] = 0.0;

    /* Check surface definition */
    if (chgm != VCM_BSPL2) {
        Vnm_print(2, "Vpmg_qfForce:  It is recommended that forces be \
calculated with the\n");
        Vnm_print(2, "Vpmg_qfForce:  cubic spline charge discretization \
scheme\n");
 }

    switch (chgm) {
        case VCM_TRIL:
            qfForceSpline1(thee, tforce, atomID);
            break;
        case VCM_BSPL2:
            qfForceSpline2(thee, tforce, atomID);
            break;
  case VCM_BSPL4:
   qfForceSpline4(thee, tforce, atomID);
   break;
        default:
            Vnm_print(2, "Vpmg_qfForce:  Undefined charge discretization \
method (%d)!\n", chgm);
            Vnm_print(2, "Vpmg_qfForce:  Forces not calculated!\n");
            return 0;
    }

    /* Assign forces */
    force[0] = tforce[0];
    force[1] = tforce[1];
    force[2] = tforce[2];

    return 1;
}


VPRIVATE void qfForceSpline1(Vpmg *thee, double *force, int
atomID) {

    Vatom *atom;

    double *apos, position[3], hx, hy, hzed;
    double xmin, ymin, zmin, xmax, ymax, zmax;
    double dx, dy, dz;
    double *u, charge, ifloat, jfloat, kfloat;
    int nx, ny, nz, ihi, ilo, jhi, jlo, khi, klo;

    VASSERT(thee != VNULL);

    atom = Valist_getAtom(thee->pbe->alist, atomID);
    apos = Vatom_getPosition(atom);
    charge = Vatom_getCharge(atom);

    /* Reset force */
    force[0] = 0.0;
    force[1] = 0.0;
    force[2] = 0.0;

    /* If we aren't in the current position, then we're done */
    if (atom->partID == 0) return;

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    xmin = thee->pmgp->xmin;
    ymin = thee->pmgp->ymin;
    zmin = thee->pmgp->zmin;
    xmax = thee->pmgp->xmax;
    ymax = thee->pmgp->ymax;
    zmax = thee->pmgp->zmax;
    u = thee->u;

```

```
    /* Make sure we're on the grid */
    if ((apos[0]<=xmin) || (apos[0]>=xmax) || (apos[1]<=ymin) || \
        (apos[1]>=ymax) || (apos[2]<=zmin) || (apos[2]>=zmax)) {
  if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
   (thee->pmgp->bcfl != BCFL_MAP)) {
            Vnm_print(2, "Vpmg_qfForce:  Atom #%d at (%4.3f, %4.3f, %4.3f) is
 off the mesh (ignoring):\n", atomID, apos[0], apos[1], apos[2]);
            Vnm_print(2, "Vpmg_qfForce:    xmin = %g, xmax = %g\n", xmin, xmax)
;
            Vnm_print(2, "Vpmg_qfForce:    ymin = %g, ymax = %g\n", ymin, ymax)
;
            Vnm_print(2, "Vpmg_qfForce:    zmin = %g, zmax = %g\n", zmin, zmax)
;
        }
        fflush(stderr);
    } else {

        /* Convert the atom position to grid coordinates */
        position[0] = apos[0] - xmin;
        position[1] = apos[1] - ymin;
        position[2] = apos[2] - zmin;
        ifloat = position[0]/hx;
        jfloat = position[1]/hy;
        kfloat = position[2]/hzed;
        ihi = (int)ceil(ifloat);
        ilo = (int)floor(ifloat);
        jhi = (int)ceil(jfloat);
        jlo = (int)floor(jfloat);
        khi = (int)ceil(kfloat);
        klo = (int)floor(kfloat);
        VASSERT((ihi < nx) && (ihi >=0));
        VASSERT((ilo < nx) && (ilo >=0));
        VASSERT((jhi < ny) && (jhi >=0));
        VASSERT((jlo < ny) && (jlo >=0));
        VASSERT((khi < nz) && (khi >=0));
        VASSERT((klo < nz) && (klo >=0));
        dx = ifloat - (double)(ilo);
        dy = jfloat - (double)(jlo);
        dz = kfloat - (double)(klo);


#if 0
        Vnm_print(1, "Vpmg_qfForce: (DEBUG) u ~ %g\n",
          dx    *dy    *dz    *u[IJK(ihi,jhi,khi)]
         +dx    *dy    *(1-dz)*u[IJK(ihi,jhi,klo)]
         +dx    *(1-dy)*dz    *u[IJK(ihi,jlo,khi)]
         +dx    *(1-dy)*(1-dz)*u[IJK(ihi,jlo,klo)]
         +(1-dx)*dy    *dz    *u[IJK(ilo,jhi,khi)]
         +(1-dx)*dy    *(1-dz)*u[IJK(ilo,jhi,klo)]
         +(1-dx)*(1-dy)*dz    *u[IJK(ilo,jlo,khi)]
         +(1-dx)*(1-dy)*(1-dz)*u[IJK(ilo,jlo,klo)]);
#endif


        if ((dx > VPMGSMALL) && (VABS(1.0-dx) > VPMGSMALL)) {
            force[0] =
              -charge*(dy    *dz    *u[IJK(ihi,jhi,khi)]
                     + dy    *(1-dz)*u[IJK(ihi,jhi,klo)]
                     + (1-dy)*dz    *u[IJK(ihi,jlo,khi)]
                     + (1-dy)*(1-dz)*u[IJK(ihi,jlo,klo)]
                     - dy    *dz    *u[IJK(ilo,jhi,khi)]
                     - dy    *(1-dz)*u[IJK(ilo,jhi,klo)]
                     - (1-dy)*dz    *u[IJK(ilo,jlo,khi)]
                     - (1-dy)*(1-dz)*u[IJK(ilo,jlo,klo)])/hx;
        } else {
            force[0] = 0;
            Vnm_print(0,
              "Vpmg_qfForce:  Atom %d on x gridline; zero x-force\n", atomID);
        }
        if ((dy > VPMGSMALL) && (VABS(1.0-dy) > VPMGSMALL)) {
            force[1] =
              -charge*(dx    *dz    *u[IJK(ihi,jhi,khi)]
                     + dx    *(1-dz)*u[IJK(ihi,jhi,klo)]
                     - dx    *dz    *u[IJK(ihi,jlo,khi)]
                     - dx    *(1-dz)*u[IJK(ihi,jlo,klo)]
                     + (1-dx)*dz    *u[IJK(ilo,jhi,khi)]
                     + (1-dx)*(1-dz)*u[IJK(ilo,jhi,klo)]
                     - (1-dx)*dz    *u[IJK(ilo,jlo,khi)]
                     - (1-dx)*(1-dz)*u[IJK(ilo,jlo,klo)])/hy;
        } else {
            force[1] = 0;
```

```
            Vnm_print(0,
              "Vpmg_qfForce:  Atom %d on y gridline; zero y-force\n", atomID);
        }
        if ((dz > VPMGSMALL) && (VABS(1.0-dz) > VPMGSMALL)) {
            force[2] =
              -charge*(dy    *dx    *u[IJK(ihi,jhi,khi)]
                     - dy    *dx    *u[IJK(ihi,jhi,klo)]
                     + (1-dy)*dx    *u[IJK(ihi,jlo,khi)]
                     - (1-dy)*dx    *u[IJK(ihi,jlo,klo)]
                     + dy    *(1-dx)*u[IJK(ilo,jhi,khi)]
                     - dy    *(1-dx)*u[IJK(ilo,jhi,klo)]
                     + (1-dy)*(1-dx)*u[IJK(ilo,jlo,khi)]
                     - (1-dy)*(1-dx)*u[IJK(ilo,jlo,klo)])/hzed;
        } else {
            force[2] = 0;
            Vnm_print(0,
              "Vpmg_qfForce:  Atom %d on z gridline; zero z-force\n", atomID);
        }
    }
}

VPRIVATE void qfForceSpline2(Vpmg *thee, double *force, int atomID) {

    Vatom *atom;

    double *apos, position[3], hx, hy, hzed;
    double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
    double mx, my, mz, dmx, dmy, dmz;
    double *u, charge, ifloat, jfloat, kfloat;
    int nx, ny, nz, im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1;
    int kp1, kp2, ii, jj, kk;

    VASSERT(thee != VNULL);

    atom = Valist_getAtom(thee->pbe->alist, atomID);
    apos = Vatom_getPosition(atom);
    charge = Vatom_getCharge(atom);

    /* Reset force */
    force[0] = 0.0;
    force[1] = 0.0;
    force[2] = 0.0;

    /* If we aren't in the current position, then we're done */
    if (atom->partID == 0) return;

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;
    xmin = thee->pmgp->xmin;
    ymin = thee->pmgp->ymin;
    zmin = thee->pmgp->zmin;
    xmax = thee->pmgp->xmax;
    ymax = thee->pmgp->ymax;
    zmax = thee->pmgp->zmax;
    u = thee->u;

    /* Make sure we're on the grid */
    if ((apos[0]<=(xmin+hx))   || (apos[0]>=(xmax-hx)) \
     || (apos[1]<=(ymin+hy))   || (apos[1]>=(ymax-hy)) \
     || (apos[2]<=(zmin+hzed)) || (apos[2]>=(zmax-hzed))) {
  if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
   (thee->pmgp->bcfl != BCFL_MAP)) {
            Vnm_print(2, "qfForceSpline2:  Atom #%d off the mesh \
    (ignoring)\n", atomID);
        }
        fflush(stderr);

    } else {

        /* Convert the atom position to grid coordinates */
        position[0] = apos[0] - xmin;
```

```
        position[1] = apos[1] - ymin;
        position[2] = apos[2] - zmin;
        ifloat = position[0]/hx;
        jfloat = position[1]/hy;
        kfloat = position[2]/hzed;
        ip1 = (int)ceil(ifloat);
        ip2 = ip1 + 1;
        im1 = (int)floor(ifloat);
        im2 = im1 - 1;
        jp1 = (int)ceil(jfloat);
        jp2 = jp1 + 1;
        jm1 = (int)floor(jfloat);
        jm2 = jm1 - 1;
        kp1 = (int)ceil(kfloat);
        kp2 = kp1 + 1;
        km1 = (int)floor(kfloat);
        km2 = km1 - 1;

        /* This step shouldn't be necessary, but it saves nasty debugging
         * later on if something goes wrong */
        ip2 = VMIN2(ip2,nx-1);
        ip1 = VMIN2(ip1,nx-1);
        im1 = VMAX2(im1,0);
        im2 = VMAX2(im2,0);
        jp2 = VMIN2(jp2,ny-1);
        jp1 = VMIN2(jp1,ny-1);
        jm1 = VMAX2(jm1,0);
        jm2 = VMAX2(jm2,0);
        kp2 = VMIN2(kp2,nz-1);
        kp1 = VMIN2(kp1,nz-1);
        km1 = VMAX2(km1,0);
        km2 = VMAX2(km2,0);


        for (ii=im2; ii<=ip2; ii++) {
            mx = bspline2(VFCHI(ii,ifloat));
            dmx = dbspline2(VFCHI(ii,ifloat));
            for (jj=jm2; jj<=jp2; jj++) {
                my = bspline2(VFCHI(jj,jfloat));
                dmy = dbspline2(VFCHI(jj,jfloat));
                for (kk=km2; kk<=kp2; kk++) {
                    mz = bspline2(VFCHI(kk,kfloat));
                    dmz = dbspline2(VFCHI(kk,kfloat));

                    force[0] += (charge*dmx*my*mz*u[IJK(ii,jj,kk)])/hx;
                    force[1] += (charge*mx*dmy*mz*u[IJK(ii,jj,kk)])/hy;
                    force[2] += (charge*mx*my*dmz*u[IJK(ii,jj,kk)])/hzed;

                }
            }
        }

    }
}

VPRIVATE void qfForceSpline4(Vpmg *thee, double *force, int atomID) {

    Vatom *atom;
    double f, c, *u, *apos, position[3];

    /* Grid variables */
    int nx,ny,nz;
    double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
    double hx, hy, hzed, ifloat, jfloat, kfloat;

    /* B-spline weights */
    double mx, my, mz, dmx, dmy, dmz;
    double mi, mj, mk;

    /* Loop indeces */
    int i, j, k, ii, jj, kk;
    int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;

    /* field */
    double e[3];

    VASSERT(thee != VNULL);
    VASSERT(thee->filled);

```

```
    atom = Valist_getAtom(thee->pbe->alist, atomID);
    apos = Vatom_getPosition(atom);
    c = Vatom_getCharge(atom);

    for (i=0;i<3;i++){
  e[i] = 0.0;
    }

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;
    xmin = thee->pmgp->xmin;
    ymin = thee->pmgp->ymin;
    zmin = thee->pmgp->zmin;
    xmax = thee->pmgp->xmax;
    ymax = thee->pmgp->ymax;
    zmax = thee->pmgp->zmax;
    u = thee->u;

    /* Make sure we're on the grid */
    if ((apos[0]<=(xmin+2*hx))   || (apos[0]>=(xmax-2*hx)) \
  || (apos[1]<=(ymin+2*hy))   || (apos[1]>=(ymax-2*hy)) \
  || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
        Vnm_print(2, "qfForceSpline4:  Atom off the mesh \
   (ignoring) %6.3f %6.3f %6.3f\n", apos[0], apos[1], apos[2]);
        fflush(stderr);
    } else {

        /* Convert the atom position to grid coordinates */
        position[0] = apos[0] - xmin;
        position[1] = apos[1] - ymin;
        position[2] = apos[2] - zmin;
        ifloat = position[0]/hx;
        jfloat = position[1]/hy;
        kfloat = position[2]/hzed;
        ip1 = (int)ceil(ifloat);
        ip2 = ip1 + 2;
        im1 = (int)floor(ifloat);
        im2 = im1 - 2;
        jp1 = (int)ceil(jfloat);
        jp2 = jp1 + 2;
        jm1 = (int)floor(jfloat);
        jm2 = jm1 - 2;
        kp1 = (int)ceil(kfloat);
        kp2 = kp1 + 2;
        km1 = (int)floor(kfloat);
        km2 = km1 - 2;

        /* This step shouldn't be necessary, but it saves nasty debugging
    * later on if something goes wrong */
        ip2 = VMIN2(ip2,nx-1);
        ip1 = VMIN2(ip1,nx-1);
        im1 = VMAX2(im1,0);
        im2 = VMAX2(im2,0);
        jp2 = VMIN2(jp2,ny-1);
        jp1 = VMIN2(jp1,ny-1);
        jm1 = VMAX2(jm1,0);
        jm2 = VMAX2(jm2,0);
        kp2 = VMIN2(kp2,nz-1);
        kp1 = VMIN2(kp1,nz-1);
        km1 = VMAX2(km1,0);
        km2 = VMAX2(km2,0);

        for (ii=im2; ii<=ip2; ii++) {
            mi = VFCHI4(ii,ifloat);
            mx = bspline4(mi);
            dmx = dbspline4(mi);
            for (jj=jm2; jj<=jp2; jj++) {
                mj = VFCHI4(jj,jfloat);
                my = bspline4(mj);
                dmy = dbspline4(mj);
                for (kk=km2; kk<=kp2; kk++) {
                    mk = VFCHI4(kk,kfloat);
                    mz = bspline4(mk);
```

```
                    dmz = dbspline4(mk);
                    f = u[IJK(ii,jj,kk)];
                    /* Field */
                    e[0] += f*dmx*my*mz/hx;
                    e[1] += f*mx*dmy*mz/hy;
                    e[2] += f*mx*my*dmz/hzed;
                }
            }
        }
    }

    /* Monopole Force */
    force[0] = e[0]*c;
    force[1] = e[1]*c;
    force[2] = e[2]*c;

}

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) {

    int i, j, k, imin, imax, jmin, jmax, kmin, kmax;
    double dx, dx2, dy, dy2, dz, dz2, a000, a001, a010, a100, r2;
    double x, xp, xm, y, yp, ym, zp, z, zm, xspan, yspan, zspan;
    double rtot2, pos[3];

    /* Convert to grid reference frame */
    pos[0] = tpos[0] - xmin;
    pos[1] = tpos[1] - ymin;
    pos[2] = tpos[2] - zmin;

    rtot2 = VSQR(rtot);

    xspan = rtot + 2*hx;
    imin = VMAX2(0, (int)ceil((pos[0] - xspan)/hx));
    imax = VMIN2(nx-1, (int)floor((pos[0] + xspan)/hx));
    for (i=imin; i<=imax; i++) {
        x = hx*i;
        dx2 = VSQR(pos[0] - x);
        if (rtot2 > dx2) {
            yspan = VSQRT(rtot2 - dx2) + 2*hy;
        } else {
            yspan = 2*hy;
        }
        jmin = VMAX2(0,(int)ceil((pos[1] - yspan)/hy));
        jmax = VMIN2(ny-1,(int)floor((pos[1] + yspan)/hy));
        for (j=jmin; j<=jmax; j++) {
            y = hy*j;
            dy2 = VSQR(pos[1] - y);
            if (rtot2 > (dx2+dy2)) {
                zspan = VSQRT(rtot2-dx2-dy2) + 2*hzed;
            } else {
                zspan = 2*hzed;
            }
            kmin = VMAX2(0,(int)ceil((pos[2] - zspan)/hzed));
            kmax = VMIN2(nz-1,(int)floor((pos[2] + zspan)/hzed));
            for (k=kmin; k<=kmax; k++) {
                z = hzed*k;
                dz2 = VSQR(pos[2] - z);

                r2 = dx2 + dy2 + dz2;

                /* We need to determine the inclusion value a000 at (i,j,k) */
                if (r2 < rtot2) a000 = 1.0;
                else a000 = 0.0;

                /* We need to evaluate the values of x which intersect the
                 * sphere and determine if these are in the interval
                 * [(i,j,k), (i+1,j,k)] */
                if (r2 < (rtot2 - hx*hx)) a100 = 1.0;
                else if (r2 > (rtot2 + hx*hx)) a100 = 0.0;
                else if (rtot2 > (dy2 + dz2)) {
                    dx = VSQRT(rtot2 - dy2 - dz2);
                    xm = pos[0] - dx;
                    xp = pos[0] + dx;
                    if ((xm < x+hx) && (xm > x)) {
                        a100 = (xm - x)/hx;
```

```
                    } else if ((xp < x+hx) && (xp > x)) {
                        a100 = (xp - x)/hx;
                    }
                } else a100 = 0.0;

                /* We need to evaluate the values of y which intersect the
                 * sphere and determine if these are in the interval
                 * [(i,j,k), (i,j+1,k)] */
                if (r2 < (rtot2 - hy*hy)) a010 = 1.0;
                else if (r2 > (rtot2 + hy*hy)) a010 = 0.0;
                else if (rtot2 > (dx2 + dz2)) {
                    dy = VSQRT(rtot2 - dx2 - dz2);
                    ym = pos[1] - dy;
                    yp = pos[1] + dy;
                    if ((ym < y+hy) && (ym > y)) {
                        a010 = (ym - y)/hy;
                    } else if ((yp < y+hy) && (yp > y)) {
                        a010 = (yp - y)/hy;
                    }
                } else a010 = 0.0;

                /* We need to evaluate the values of y which intersect the
                 * sphere and determine if these are in the interval
                 * [(i,j,k), (i,j,k+1)] */
                if (r2 < (rtot2 - hzed*hzed)) a001 = 1.0;
                else if (r2 > (rtot2 + hzed*hzed)) a001 = 0.0;
                else if (rtot2 > (dx2 + dy2)) {
                    dz = VSQRT(rtot2 - dx2 - dy2);
                    zm = pos[2] - dz;
                    zp = pos[2] + dz;
                    if ((zm < z+hzed) && (zm > z)) {
                        a001 = (zm - z)/hzed;
                    } else if ((zp < z+hzed) && (zp > z)) {
                        a001 = (zp - z)/hzed;
                    }
                } else a001 = 0.0;

                if (a100 < xarray[IJK(i,j,k)]) xarray[IJK(i,j,k)] = a100;
                if (a010 < yarray[IJK(i,j,k)]) yarray[IJK(i,j,k)] = a010;
                if (a001 < zarray[IJK(i,j,k)]) zarray[IJK(i,j,k)] = a001;

            } /* k loop */
        } /* j loop */
    } /* i loop */
}

/*

 NOTE: This is the original version of the markSphere function. It's in here
 for reference and in case a reversion to the original code is needed.
 D. Gohara (2/14/08)
 */
/*
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) {

 int i, j, k, imin, imax, jmin, jmax, kmin, kmax;
 double dx, dx2, dy, dy2, dz, dz2;
 double rtot2, pos[3];

 // Convert to grid reference frame
 pos[0] = tpos[0] - xmin;
 pos[1] = tpos[1] - ymin;
 pos[2] = tpos[2] - zmin;

 rtot2 = VSQR(rtot);

 dx = rtot + 0.5*hx;
 imin = VMAX2(0,(int)ceil((pos[0] - dx)/hx));
 imax = VMIN2(nx-1,(int)floor((pos[0] + dx)/hx));
 for (i=imin; i<=imax; i++) {
  dx2 = VSQR(pos[0] - hx*i);
  if (rtot2 > dx2) {
   dy = VSQRT(rtot2 - dx2) + 0.5*hy;
  } else {
   dy = 0.5*hy;
  }
```

```
  jmin = VMAX2(0,(int)ceil((pos[1] - dy)/hy));
  jmax = VMIN2(ny-1,(int)floor((pos[1] + dy)/hy));
  for (j=jmin; j<=jmax; j++) {
   dy2 = VSQR(pos[1] - hy*j);
   if (rtot2 > (dx2+dy2)) {
    dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
   } else {
    dz = 0.5*hzed;
   }
   kmin = VMAX2(0,(int)ceil((pos[2] - dz)/hzed));
   kmax = VMIN2(nz-1,(int)floor((pos[2] + dz)/hzed));
   for (k=kmin; k<=kmax; k++) {
    dz2 = VSQR(k*hzed - pos[2]);
    if ((dz2 + dy2 + dx2) <= rtot2) {
     array[IJK(i,j,k)] = markVal;
    }
   } // k loop
  } // j loop
 } // i loop
}
*/
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) {

    int i, j, k;
 double fi,fj,fk;
 int imin, imax;
 int jmin, jmax;
 int kmin, kmax;
    double dx2, dy2, dz2;
 double xrange, yrange, zrange;
    double rtot2, posx, posy, posz;

    /* Convert to grid reference frame */
    posx = tpos[0] - xmin;
    posy = tpos[1] - ymin;
    posz = tpos[2] - zmin;

    rtot2 = VSQR(rtot);

 xrange = rtot + 0.5 * hx;
 yrange = rtot + 0.5 * hy;
 zrange = rtot + 0.5 * hz;

 imin = VMAX2(0, (int)ceil((posx - xrange)/hx));
 jmin = VMAX2(0, (int)ceil((posy - yrange)/hy));
 kmin = VMAX2(0, (int)ceil((posz - zrange)/hz));

 imax = VMIN2(nx-1, (int)floor((posx + xrange)/hx));
 jmax = VMIN2(ny-1, (int)floor((posy + yrange)/hy));
 kmax = VMIN2(nz-1, (int)floor((posz + zrange)/hz));

 for (i=imin,fi=imin; i<=imax; i++, fi+=1.) {
  dx2 = VSQR(posx - hx*fi);
  for (j=jmin,fj=jmin; j<=jmax; j++, fj+=1.) {
   dy2 = VSQR(posy - hy*fj);
   if((dx2 + dy2) > rtot2) continue;
   for (k=kmin, fk=kmin; k<=kmax; k++, fk+=1.) {
    dz2 = VSQR(posz - hz*fk);
    if ((dz2 + dy2 + dx2) <= rtot2) {
     array[IJK(i,j,k)] = markVal;
                }
   }
  }
 }
}

VPRIVATE void zlapSolve(
        Vpmg *thee,
        double **solution,
        double **source,
        double **work1
        ) {

    /* NOTE:  this is an incredibly inefficient algorithm.  The next
     * improvement is to focus on only non-zero entries in the source term.
     * The best improvement is to use a fast sine transform */

```

```
    int n, nx, ny, nz, i, j, k, kx, ky, kz;
    double hx, hy, hzed, wx, wy, wz, xlen, ylen, zlen;
    double phix, phixp1, phixm1, phiy, phiym1, phiyp1, phiz, phizm1, phizp1;
    double norm, coef, proj, eigx, eigy, eigz;
    double ihx2, ihy2, ihzed2;
    double *u, *f, *phi;

    /* Snarf grid parameters */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    n = nx*ny*nz;
    hx = thee->pmgp->hx;
    ihx2 = 1.0/hx/hx;
    hy = thee->pmgp->hy;
    ihy2 = 1.0/hy/hy;
    hzed = thee->pmgp->hzed;
    ihzed2 = 1.0/hzed/hzed;
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;

    /* Set solution and source array pointers */
    u = *solution;
    f = *source;
    phi = *work1;

    /* Zero out the solution vector */
    for (i=0; i<n; i++) thee->u[i] = 0.0;

    /* Iterate through the wavenumbers */
    for (kx=1; kx<(nx-1); kx++) {

        wx = (VPI*(double)kx)/((double)nx - 1.0);
        eigx = 2.0*ihx2*(1.0 - cos(wx));

        for (ky=1; ky<(ny-1); ky++) {

            wy = (VPI*(double)ky)/((double)ny - 1.0);
            eigy = 2.0*ihy2*(1.0 - cos(wy));

            for (kz=1; kz<(nz-1); kz++) {

                wz = (VPI*(double)kz)/((double)nz - 1.0);
                eigz = 2.0*ihzed2*(1.0 - cos(wz));

                /* Calculate the basis function.
                 * We could calculate each basis function as
                 *   phix(i) = sin(wx*i)
                 *   phiy(j) = sin(wy*j)
                 *   phiz(k) = sin(wz*k)
                 * However, this is likely to be very expensive.
                 * Therefore, we can use the fact that
                 *   phix(i+1) = (2-hx*hx*eigx)*phix(i) - phix(i-1)
                 * */
                for (i=1; i<(nx-1); i++) {
                    if (i == 1) {
                        phix = sin(wx*(double)i);
                        phixm1 = 0.0;
                    } else {
                        phixp1 = (2.0-hx*hx*eigx)*phix - phixm1;
                        phixm1 = phix;
                        phix = phixp1;
                    }
                    /* phix = sin(wx*(double)i); */
                    for (j=1; j<(ny-1); j++) {
                        if (j == 1) {
                            phiy = sin(wy*(double)j);
                            phiym1 = 0.0;
                        } else {
                            phiyp1 = (2.0-hy*hy*eigy)*phiy - phiym1;
                            phiym1 = phiy;
                            phiy = phiyp1;
                        }
                        /* phiy = sin(wy*(double)j); */
                        for (k=1; k<(nz-1); k++) {
                            if (k == 1) {
                                phiz = sin(wz*(double)k);
                                phizm1 = 0.0;
                            } else {
                                phizp1 = (2.0-hzed*hzed*eigz)*phiz - phizm1;
```

```
                                phizm1 = phiz;
                                phiz = phizp1;
                            }
                            /* phiz = sin(wz*(double)k); */

                            phi[IJK(i,j,k)] = phix*phiy*phiz;

                        }
                    }
                }

                /* Calculate the projection of the source function on this
                 * basis function */
                proj = 0.0;
                for (i=1; i<(nx-1); i++) {
                    for (j=1; j<(ny-1); j++) {
                        for (k=1; k<(nz-1); k++) {

                            proj += f[IJK(i,j,k)]*phi[IJK(i,j,k)];

                        } /* k loop */
                    } /* j loop */
                } /* i loop */

                /* Assemble the coefficient to weight the contribution of this
                 * basis function to the solution */
                /* The first contribution is the projection */
                coef = proj;
                /* The second contribution is the eigenvalue */
                coef = coef/(eigx + eigy + eigz);
                /* The third contribution is the normalization factor */
                coef = (8.0/xlen/ylen/zlen)*coef;
                /* The fourth contribution is from scaling the diagonal */
                /* coef = hx*hy*hzed*coef; */

                /* Evaluate the basis function at each grid point */
                for (i=1; i<(nx-1); i++) {
                    for (j=1; j<(ny-1); j++) {
                        for (k=1; k<(nz-1); k++) {

                            u[IJK(i,j,k)] += coef*phi[IJK(i,j,k)];

                        } /* k loop */
                    } /* j loop */
                } /* i loop */

            } /* kz loop */
        } /* ky loop */
    } /* kx loop */

}

VPUBLIC int Vpmg_solveLaplace(Vpmg *thee) {

    int i, j, k, ijk, nx, ny, nz, n, dilo, dihi, djlo, djhi, dklo, dkhi;
    double hx, hy, hzed, epsw, iepsw, scal, scalx, scaly, scalz;

    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    n = nx*ny*nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    epsw = Vpbe_getSolventDiel(thee->pbe);
    iepsw = 1.0/epsw;
    scal = hx*hy*hzed;
    scalx = hx*hy/hzed;
    scaly = hx*hzed/hy;
    scalz = hx*hy/hzed;

    if (!(thee->filled)) {
        Vnm_print(2, "Vpmg_solve:  Need to call Vpmg_fillco()!\n");
        return 0;
    }

    /* Load boundary conditions into the RHS array */
    for (i=1; i<(nx-1); i++) {

        if (i == 1) dilo = 1;
        else dilo = 0;
```

```
        if (i == nx-2) dihi = 1;
        else dihi = 0;

        for (j=1; j<(ny-1); j++) {

            if (j == 1) djlo = 1;
            else djlo = 0;
            if (j == ny-2) djhi = 1;
            else djhi = 0;

            for (k=1; k<(nz-1); k++) {

                if (k == 1) dklo = 1;
                else dklo = 0;
                if (k == nz-2) dkhi = 1;
                else dkhi = 0;

                thee->fcf[IJK(i,j,k)] = \
                      iepsw*scal*thee->charge[IJK(i,j,k)] \
                    + dilo*scalx*thee->gxcf[IJKx(j,k,0)] \
                    + dihi*scalx*thee->gxcf[IJKx(j,k,1)] \
                    + djlo*scaly*thee->gycf[IJKy(i,k,0)] \
                    + djhi*scaly*thee->gycf[IJKy(i,k,1)] \
                    + dklo*scalz*thee->gzcf[IJKz(i,j,0)] \
                    + dkhi*scalz*thee->gzcf[IJKz(i,j,1)] ;

            }
        }
    }

    /* Solve */
    zlapSolve( thee, &(thee->u), &(thee->fcf), &(thee->tcf) );

    /* Add boundary conditions to solution */
    /* i faces */
    for (j=0; j<ny; j++) {
        for (k=0; k<nz; k++) {
            thee->u[IJK(0,j,k)] = thee->gxcf[IJKx(j,k,0)];
            thee->u[IJK(nx-1,j,k)] = thee->gycf[IJKx(j,k,1)];
        }
    }
    /* j faces */
    for (i=0; i<nx; i++) {
        for (k=0; k<nz; k++) {
            thee->u[IJK(i,0,k)] = thee->gycf[IJKy(i,k,0)];
            thee->u[IJK(i,ny-1,k)] = thee->gycf[IJKy(i,k,1)];
        }
    }
    /* k faces */
    for (i=0; i<nx; i++) {
        for (j=0; j<ny; j++) {
            thee->u[IJK(i,j,0)] = thee->gzcf[IJKz(i,j,0)];
            thee->u[IJK(i,j,nz-1)] = thee->gzcf[IJKz(i,j,1)];
        }
    }

    return 1;

}

VPRIVATE double VFCHI4(int i, double f) {
  return (2.5+((double)(i)-(f)));
}

VPRIVATE double bspline4(double x) {

    double m, m2;
    static double one6 = 1.0/6.0;
    static double one8 = 1.0/8.0;
    static double one24 = 1.0/24.0;
    static double thirteen24 = 13.0/24.0;
    static double fourtyseven24 = 47.0/24.0;
    static double seventeen24 = 17.0/24.0;

    if ((x > 0.0) && (x <= 1.0)){
      m = x*x;
      return one24*m*m;
    } else if ((x > 1.0) && (x <= 2.0)){
      m = x - 1.0;
      m2 = m*m;
      return -one8 + one6*x + m2*(0.25 + one6*m - one6*m2);
```

```
    } else if ((x > 2.0) && (x <= 3.0)){
      m = x - 2.0;
      m2 = m*m;
      return -thirteen24 + 0.5*x + m2*(-0.25 - 0.5*m + 0.25*m2);
    } else if ((x > 3.0) && (x <= 4.0)){
      m = x - 3.0;
      m2 = m*m;
      return fourtyseven24 - 0.5*x + m2*(-0.25 + 0.5*m - one6*m2);
    } else if ((x > 4.0) && (x <= 5.0)){
      m = x - 4.0;
      m2 = m*m;
      return seventeen24 - one6*x + m2*(0.25 - one6*m + one24*m2);
    } else {
      return 0.0;
    }
}

VPUBLIC double dbspline4(double x) {

    double m, m2;
    static double one6 = 1.0/6.0;
    static double one3 = 1.0/3.0;
    static double two3 = 2.0/3.0;
    static double thirteen6 = 13.0/6.0;

    if ((x > 0.0) && (x <= 1.0)){
      m2 = x*x;
      return one6*x*m2;
    } else if ((x > 1.0) && (x <= 2.0)){
      m = x - 1.0;
      m2 = m*m;
      return -one3 + 0.5*x + m2*(0.5 - two3*m);
    } else if ((x > 2.0) && (x <= 3.0)){
      m = x - 2.0;
      m2 = m*m;
      return 1.5 - 0.5*x + m2*(-1.5 + m);
    } else if ((x > 3.0) && (x <= 4.0)){
      m = x - 3.0;
      m2 = m*m;
      return 1.0 - 0.5*x + m2*(1.5 - two3*m);
    } else if ((x > 4.0) && (x <= 5.0)){
      m = x - 4.0;
      m2 = m*m;
      return -thirteen6 + 0.5*x + m2*(-0.5 + one6*m);
    } else {
      return 0.0;
    }
}

VPUBLIC double d2bspline4(double x) {

    double m, m2;

    if ((x > 0.0) && (x <= 1.0)){
      return 0.5*x*x;
    } else if ((x > 1.0) && (x <= 2.0)){
      m = x - 1.0;
      m2 = m*m;
      return -0.5 + x - 2.0*m2;
    } else if ((x > 2.0) && (x <= 3.0)){
      m = x - 2.0;
      m2 = m*m;
      return 5.5 - 3.0*x + 3.0*m2;
    } else if ((x > 3.0) && (x <= 4.0)){
      m = x - 3.0;
      m2 = m*m;
      return -9.5 + 3.0*x - 2.0*m2;
    } else if ((x > 4.0) && (x <= 5.0)){
      m = x - 4.0;
      m2 = m*m;
      return 4.5 - x + 0.5*m2;
    } else {
      return 0.0;
    }
}

VPUBLIC double d3bspline4(double x) {

    if      ((x > 0.0) && (x <= 1.0)) return x;
    else if ((x > 1.0) && (x <= 2.0)) return 5.0 - 4.0 * x;
    else if ((x > 2.0) && (x <= 3.0)) return -15.0 + 6.0 * x;
```

```
    else if ((x > 3.0) && (x <= 4.0)) return 15.0 - 4.0 * x;
    else if ((x > 4.0) && (x <= 5.0)) return x - 5.0;
    else                              return 0.0;

}

VPUBLIC void fillcoPermanentMultipole(Vpmg *thee) {

    Valist *alist;
    Vpbe *pbe;
    Vatom *atom;
    /* Coversions */
    double zmagic, f;
    /* Grid */
    double xmin, xmax, ymin, ymax, zmin, zmax;
    double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;
    double hx, hy, hzed, *apos;
    /* Multipole */
    double charge, *dipole,*quad;
    double c,ux,uy,uz,qxx,qyx,qyy,qzx,qzy,qzz,qave;
    /* B-spline weights */
    double mx,my,mz,dmx,dmy,dmz,d2mx,d2my,d2mz;
    double mi,mj,mk;
    /* Loop variables */
    int i, ii, jj, kk, nx, ny, nz, iatom;
    int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;

    /* sanity check */
    double mir,mjr,mkr,mr2;
    double debye,mc,mux,muy,muz,mqxx,mqyx,mqyy,mqzx,mqzy,mqzz;

    VASSERT(thee != VNULL);

    /* Get PBE info */
    pbe = thee->pbe;
    alist = pbe->alist;
    zmagic = Vpbe_getZmagic(pbe);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;

    /* Conversion */
    f = zmagic/(hx*hy*hzed);

    /* Define the total domain size */
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;

    /* Define the min/max dimensions */
    xmin = thee->pmgp->xcent - (xlen/2.0);
    ymin = thee->pmgp->ycent - (ylen/2.0);
    zmin = thee->pmgp->zcent - (zlen/2.0);
    xmax = thee->pmgp->xcent + (xlen/2.0);
    ymax = thee->pmgp->ycent + (ylen/2.0);
    zmax = thee->pmgp->zcent + (zlen/2.0);

    /* Fill in the source term (permanent atomic multipoles) */
    Vnm_print(0, "fillcoPermanentMultipole:  filling in source term.\n");
    for (iatom=0; iatom<Valist_getNumberAtoms(alist);
iatom++) {

        atom = Valist_getAtom(alist, iatom);
        apos = Vatom_getPosition(atom);

        c = Vatom_getCharge(atom)*f;

#if defined(WITH_TINKER)
        dipole = Vatom_getDipole(atom);
        ux = dipole[0]/hx*f;
        uy = dipole[1]/hy*f;
        uz = dipole[2]/hzed*f;
        quad = Vatom_getQuadrupole(atom);
        qxx = (1.0/3.0)*quad[0]/(hx*hx)*f;
        qyx = (2.0/3.0)*quad[3]/(hx*hy)*f;
        qyy = (1.0/3.0)*quad[4]/(hy*hy)*f;
```

```
        qzx = (2.0/3.0)*quad[6]/(hzed*hx)*f;
        qzy = (2.0/3.0)*quad[7]/(hzed*hy)*f;
        qzz = (1.0/3.0)*quad[8]/(hzed*hzed)*f;
#else
        ux = 0.0;
        uy = 0.0;
        uz = 0.0;
        qxx = 0.0;
        qyx = 0.0;
        qyy = 0.0;
        qzx = 0.0;
        qzy = 0.0;
        qzz = 0.0;
#endif /* if defined(WITH_TINKER) */

        /* check
        mc = 0.0;
        mux = 0.0;
        muy = 0.0;
        muz = 0.0;
        mqxx = 0.0;
        mqyx = 0.0;
        mqyy = 0.0;
        mqzx = 0.0;
        mqzy = 0.0;
        mqzz = 0.0; */

        /* Make sure we're on the grid */
        if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx))  || \
            (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy))  || \
            (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
            Vnm_print(2, "fillcoPermanentMultipole: Atom #%d at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring this atom):\n", iatom, apos[0], apos[1], apos[2]);
            Vnm_print(2, "fillcoPermanentMultipole: xmin = %g, xmax = %g\n", xmin, xmax);
            Vnm_print(2, "fillcoPermanentMultipole: ymin = %g, ymax = %g\n", ymin, ymax);
            Vnm_print(2, "fillcoPermanentMultipole: zmin = %g, zmax = %g\n", zmin, zmax);
            fflush(stderr);
        } else {

            /* Convert the atom position to grid reference frame */
            position[0] = apos[0] - xmin;
            position[1] = apos[1] - ymin;
            position[2] = apos[2] - zmin;

            /* Figure out which vertices we're next to */
            ifloat = position[0]/hx;
            jfloat = position[1]/hy;
            kfloat = position[2]/hzed;

            ip1   = (int)ceil(ifloat);
            ip2   = ip1 + 2;
            im1   = (int)floor(ifloat);
            im2   = im1 - 2;
            jp1   = (int)ceil(jfloat);
            jp2   = jp1 + 2;
            jm1   = (int)floor(jfloat);
            jm2   = jm1 - 2;
            kp1   = (int)ceil(kfloat);
            kp2   = kp1 + 2;
            km1   = (int)floor(kfloat);
            km2   = km1 - 2;

            /* This step shouldn't be necessary, but it saves nasty debugging
             * later on if something goes wrong */
            ip2 = VMIN2(ip2,nx-1);
            ip1 = VMIN2(ip1,nx-1);
            im1 = VMAX2(im1,0);
            im2 = VMAX2(im2,0);
            jp2 = VMIN2(jp2,ny-1);
            jp1 = VMIN2(jp1,ny-1);
            jm1 = VMAX2(jm1,0);
            jm2 = VMAX2(jm2,0);
            kp2 = VMIN2(kp2,nz-1);
            kp1 = VMIN2(kp1,nz-1);
            km1 = VMAX2(km1,0);
            km2 = VMAX2(km2,0);

```

```
            /* Now assign fractions of the charge to the nearby verts */
            for (ii=im2; ii<=ip2; ii++) {
                mi = VFCHI4(ii,ifloat);
                mx = bspline4(mi);
                dmx = dbspline4(mi);
                d2mx = d2bspline4(mi);
                for (jj=jm2; jj<=jp2; jj++) {
                    mj = VFCHI4(jj,jfloat);
                    my = bspline4(mj);
                    dmy = dbspline4(mj);
                    d2my = d2bspline4(mj);
                    for (kk=km2; kk<=kp2; kk++) {
                        mk = VFCHI4(kk,kfloat);
                        mz = bspline4(mk);
                        dmz = dbspline4(mk);
                        d2mz = d2bspline4(mk);
                        charge = mx*my*mz*c -
                         dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz +
                         d2mx*my*mz*qxx +
                         dmx*dmy*mz*qyx + mx*d2my*mz*qyy +
                         dmx*my*dmz*qzx + mx*dmy*dmz*qzy + mx*my*d2mz*qzz;
                        thee->charge[IJK(ii,jj,kk)] += charge;

                        /* sanity check - recalculate traceless multipoles
                           from the grid charge distribution for this
                           site.

                        mir = (mi - 2.5) * hx;
                        mjr = (mj - 2.5) * hy;
                        mkr = (mk - 2.5) * hzed;
                        mr2 = mir*mir+mjr*mjr+mkr*mkr;
                        mc += charge;
                        mux += mir * charge;
                        muy += mjr * charge;
                        muz += mkr * charge;
                        mqxx += (1.5*mir*mir - 0.5*mr2) * charge;
                        mqyx += 1.5*mjr*mir * charge;
                        mqyy += (1.5*mjr*mjr - 0.5*mr2) * charge;
                        mqzx += 1.5*mkr*mir * charge;
                        mqzy += 1.5*mkr*mjr * charge;
                        mqzz += (1.5*mkr*mkr - 0.5*mr2) * charge;
                         */
                    }
                }
            }
        } /* endif (on the mesh) */

        /* print out the Grid vs. Ideal Point Multipole. */

        /*
        debye = 4.8033324;
        mc = mc/f;
        mux = mux/f*debye;
        muy = muy/f*debye;
        muz = muz/f*debye;
        mqxx = mqxx/f*debye;
        mqyy = mqyy/f*debye;
        mqzz = mqzz/f*debye;
        mqyx = mqyx/f*debye;
        mqzx = mqzx/f*debye;
        mqzy = mqzy/f*debye;

        printf(" Grid v. Actual Permanent Multipole for Site %i\n",iatom);
        printf(" G: %10.6f\n",mc);
        printf(" A: %10.6f\n\n",c/f);
        printf(" G: %10.6f %10.6f %10.6f\n",mux,muy,muz);
        printf(" A: %10.6f %10.6f %10.6f\n\n",
                 (ux * hx / f) * debye,
                 (uy * hy / f) * debye,
                 (uz * hzed /f) * debye);
        printf(" G: %10.6f\n",mqxx);
        printf(" A: %10.6f\n",quad[0]*debye);
        printf(" G: %10.6f %10.6f\n",mqyx,mqyy);
        printf(" A: %10.6f %10.6f\n",quad[3]*debye,quad[4]*debye);
        printf(" G: %10.6f %10.6f %10.6f\n",mqzx,mqzy,mqzz);
        printf(" A: %10.6f %10.6f %10.6f\n\n",
                quad[6]*debye,quad[7]*debye,quad[8]*debye);  */

    } /* endfor (each atom) */
}
```

```
#if defined(WITH_TINKER)

VPUBLIC void fillcoInducedDipole(Vpmg *thee) {

    Valist *alist;
    Vpbe *pbe;
    Vatom *atom;
    /* Conversions */
    double zmagic, f;
    /* Grid */
    double xmin, xmax, ymin, ymax, zmin, zmax;
    double xlen, ylen, zlen, ifloat, jfloat, kfloat;
    double hx, hy, hzed, *apos, position[3];
    /* B-spline weights */
    double mx, my, mz, dmx, dmy, dmz;
    /* Dipole */
    double charge, *dipole, ux,uy,uz;
    double mi,mj,mk;
    /* Loop indeces */
    int i, ii, jj, kk, nx, ny, nz, iatom;
    int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;

    double debye;
    double mux,muy,muz;
    double mir,mjr,mkr;

    VASSERT(thee != VNULL);

    /* Get PBE info */
    pbe = thee->pbe;
    alist = pbe->alist;
    zmagic = Vpbe_getZmagic(pbe);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;

    /* Conversion */
    f = zmagic/(hx*hy*hzed);

    /* Define the total domain size */
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;

    /* Define the min/max dimensions */
    xmin = thee->pmgp->xcent - (xlen/2.0);
    ymin = thee->pmgp->ycent - (ylen/2.0);
    zmin = thee->pmgp->zcent - (zlen/2.0);
    xmax = thee->pmgp->xcent + (xlen/2.0);
    ymax = thee->pmgp->ycent + (ylen/2.0);
    zmax = thee->pmgp->zcent + (zlen/2.0);

    /* Fill in the source term (induced dipoles) */
    Vnm_print(0, "fillcoInducedDipole:  filling in the source term.\n");
    for (iatom=0; iatom<Valist_getNumberAtoms(alist);
iatom++) {

        atom = Valist_getAtom(alist, iatom);
        apos = Vatom_getPosition(atom);

        dipole = Vatom_getInducedDipole(atom);
        ux = dipole[0]/hx*f;
        uy = dipole[1]/hy*f;
        uz = dipole[2]/hzed*f;

        mux = 0.0;
        muy = 0.0;
        muz = 0.0;

        /* Make sure we're on the grid */
        if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx))  || \
            (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy))  || \
            (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
            Vnm_print(2, "fillcoInducedDipole: Atom #%d at (%4.3f, %4.3f,
%4.3f) is off the mesh (ignoring this atom):\n", iatom, apos[0], apos[1], apos[2]);
            Vnm_print(2, "fillcoInducedDipole: xmin = %g, xmax = %g\n", xmin,
```

```
xmax);
            Vnm_print(2, "fillcoInducedDipole: ymin = %g, ymax = %g\n", ymin, ymax);
            Vnm_print(2, "fillcoInducedDipole: zmin = %g, zmax = %g\n", zmin, zmax);
            fflush(stderr);
        } else {

            /* Convert the atom position to grid reference frame */
            position[0] = apos[0] - xmin;
            position[1] = apos[1] - ymin;
            position[2] = apos[2] - zmin;

            /* Figure out which vertices we're next to */
            ifloat = position[0]/hx;
            jfloat = position[1]/hy;
            kfloat = position[2]/hzed;

            ip1   = (int)ceil(ifloat);
            ip2   = ip1 + 2;
            im1   = (int)floor(ifloat);
            im2   = im1 - 2;
            jp1   = (int)ceil(jfloat);
            jp2   = jp1 + 2;
            jm1   = (int)floor(jfloat);
            jm2   = jm1 - 2;
            kp1   = (int)ceil(kfloat);
            kp2   = kp1 + 2;
            km1   = (int)floor(kfloat);
            km2   = km1 - 2;

            /* This step shouldn't be necessary, but it saves nasty debugging
             * later on if something goes wrong */
            ip2 = VMIN2(ip2,nx-1);
            ip1 = VMIN2(ip1,nx-1);
            im1 = VMAX2(im1,0);
            im2 = VMAX2(im2,0);
            jp2 = VMIN2(jp2,ny-1);
            jp1 = VMIN2(jp1,ny-1);
            jm1 = VMAX2(jm1,0);
            jm2 = VMAX2(jm2,0);
            kp2 = VMIN2(kp2,nz-1);
            kp1 = VMIN2(kp1,nz-1);
            km1 = VMAX2(km1,0);
            km2 = VMAX2(km2,0);

            /* Now assign fractions of the dipole to the nearby verts */
            for (ii=im2; ii<=ip2; ii++) {
                mi = VFCHI4(ii,ifloat);
                mx = bspline4(mi);
                dmx = dbspline4(mi);
                for (jj=jm2; jj<=jp2; jj++) {
                    mj = VFCHI4(jj,jfloat);
                    my = bspline4(mj);
                    dmy = dbspline4(mj);
                    for (kk=km2; kk<=kp2; kk++) {
                        mk = VFCHI4(kk,kfloat);
                        mz = bspline4(mk);
                        dmz = dbspline4(mk);
                        charge = -dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz;
                        thee->charge[IJK(ii,jj,kk)] += charge;

                        /*
                        mir = (mi - 2.5) * hx;
                        mjr = (mj - 2.5) * hy;
                        mkr = (mk - 2.5) * hzed;
                        mux += mir * charge;
                        muy += mjr * charge;
                        muz += mkr * charge;
                        */
                    }
                }
            }
        } /* endif (on the mesh) */

        /* check
        debye = 4.8033324;
        mux = mux/f*debye;
        muy = muy/f*debye;
        muz = muz/f*debye;

```

```
        printf(" Grid v. Actual Induced Dipole for Site %i\n",iatom);
        printf(" G: %10.6f %10.6f %10.6f\n",mux,muy,muz);
        printf(" A: %10.6f %10.6f %10.6f\n\n",
                (ux * hx / f) * debye,
                (uy * hy / f) * debye,
                (uz * hzed /f) * debye);
         */

    } /* endfor (each atom) */
}

VPUBLIC void fillcoNLInducedDipole(Vpmg *thee) {

    Valist *alist;
    Vpbe *pbe;
    Vatom *atom;
    /* Conversions */
    double zmagic, f;
    /* Grid */
    double xmin, xmax, ymin, ymax, zmin, zmax;
    double xlen, ylen, zlen, ifloat, jfloat, kfloat;
    double hx, hy, hzed, *apos, position[3];
    /* B-spline weights */
    double mx, my, mz, dmx, dmy, dmz;
    /* Dipole */
    double charge, *dipole, ux,uy,uz;
    double mi,mj,mk;
    /* Loop indeces */
    int i, ii, jj, kk, nx, ny, nz, iatom;
    int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;

    /* sanity check
    double debye;
    double mux,muy,muz;
    double mir,mjr,mkr;
     */

    VASSERT(thee != VNULL);

    /* Get PBE info */
    pbe = thee->pbe;
    alist = pbe->alist;
    zmagic = Vpbe_getZmagic(pbe);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;

    /* Conversion */
    f = zmagic/(hx*hy*hzed);

    /* Define the total domain size */
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;

    /* Define the min/max dimensions */
    xmin = thee->pmgp->xcent - (xlen/2.0);
    ymin = thee->pmgp->ycent - (ylen/2.0);
    zmin = thee->pmgp->zcent - (zlen/2.0);
    xmax = thee->pmgp->xcent + (xlen/2.0);
    ymax = thee->pmgp->ycent + (ylen/2.0);
    zmax = thee->pmgp->zcent + (zlen/2.0);

    /* Fill in the source term (non-local induced dipoles) */
    Vnm_print(0, "fillcoNLInducedDipole:  filling in source term.\n");
    for (iatom=0; iatom<Valist_getNumberAtoms(alist);
iatom++) {

        atom = Valist_getAtom(alist, iatom);
        apos = Vatom_getPosition(atom);

        dipole = Vatom_getNLInducedDipole(atom);
        ux = dipole[0]/hx*f;
        uy = dipole[1]/hy*f;
        uz = dipole[2]/hzed*f;

```

```
        /*
        mux = 0.0;
        muy = 0.0;
        muz = 0.0;
         */

        /* Make sure we're on the grid */
        if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx))  || \
            (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy))  || \
            (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
            Vnm_print(2, "fillcoNLInducedDipole: Atom #%d at (%4.3f,%4.3f) is off the mesh (ignoring this atom):\n", iatom, apos[0], apos[1], apos[2]);
            Vnm_print(2, "fillcoNLInducedDipole: xmin = %g, xmax = %g\n", xmin, xmax);
            Vnm_print(2, "fillcoNLInducedDipole: ymin = %g, ymax = %g\n", ymin, ymax);
            Vnm_print(2, "fillcoNLInducedDipole: zmin = %g, zmax = %g\n", zmin, zmax);
            fflush(stderr);
        } else {

            /* Convert the atom position to grid reference frame */
            position[0] = apos[0] - xmin;
            position[1] = apos[1] - ymin;
            position[2] = apos[2] - zmin;

            /* Figure out which vertices we're next to */
            ifloat = position[0]/hx;
            jfloat = position[1]/hy;
            kfloat = position[2]/hzed;

            ip1   = (int)ceil(ifloat);
            ip2   = ip1 + 2;
            im1   = (int)floor(ifloat);
            im2   = im1 - 2;
            jp1   = (int)ceil(jfloat);
            jp2   = jp1 + 2;
            jm1   = (int)floor(jfloat);
            jm2   = jm1 - 2;
            kp1   = (int)ceil(kfloat);
            kp2   = kp1 + 2;
            km1   = (int)floor(kfloat);
            km2   = km1 - 2;

            /* This step shouldn't be necessary, but it saves nasty debugging
             * later on if something goes wrong */
            ip2 = VMIN2(ip2,nx-1);
            ip1 = VMIN2(ip1,nx-1);
            im1 = VMAX2(im1,0);
            im2 = VMAX2(im2,0);
            jp2 = VMIN2(jp2,ny-1);
            jp1 = VMIN2(jp1,ny-1);
            jm1 = VMAX2(jm1,0);
            jm2 = VMAX2(jm2,0);
            kp2 = VMIN2(kp2,nz-1);
            kp1 = VMIN2(kp1,nz-1);
            km1 = VMAX2(km1,0);
            km2 = VMAX2(km2,0);

            /* Now assign fractions of the non local induced dipole
               to the nearby verts */
            for (ii=im2; ii<=ip2; ii++) {
                mi = VFCHI4(ii,ifloat);
                mx = bspline4(mi);
                dmx = dbspline4(mi);
                for (jj=jm2; jj<=jp2; jj++) {
                    mj = VFCHI4(jj,jfloat);
                    my = bspline4(mj);
                    dmy = dbspline4(mj);
                    for (kk=km2; kk<=kp2; kk++) {
                        mk = VFCHI4(kk,kfloat);
                        mz = bspline4(mk);
                        dmz = dbspline4(mk);
                        charge = -dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz;
                        thee->charge[IJK(ii,jj,kk)] += charge;

                        /*
                        mir = (mi - 2.5) * hx;
                        mjr = (mj - 2.5) * hy;
                        mkr = (mk - 2.5) * hzed;
                        mux += mir * charge;
```

```
                        muy += mjr * charge;
                        muz += mkr * charge;
                         */
                    }
                }
            }
        } /* endif (on the mesh) */

        /*
        debye = 4.8033324;
        mux = mux/f*debye;
        muy = muy/f*debye;
        muz = muz/f*debye;

        printf(" Grid v. Actual Non-Local Induced Dipole for Site %i\n",iatom);
        printf(" G: %10.6f %10.6f %10.6f\n",mux,muy,muz);
        printf(" A: %10.6f %10.6f %10.6f\n\n",
                 (ux * hx / f) * debye,
                 (uy * hy / f) * debye,
                 (uz * hzed /f) * debye); */

    } /* endfor (each atom) */
}

VPUBLIC double Vpmg_qfPermanentMultipoleEnergy(
    Vpmg *thee, int atomID) {

    double *u;
    Vatom *atom;
    /* Grid variables */
    int nx, ny, nz;
    double xmax, xmin, ymax, ymin, zmax, zmin;
    double hx, hy, hzed, ifloat, jfloat, kfloat;
    double mi, mj, mk;
    double *position;
    /* B-spline weights */
    double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz;
    /* Loop indeces */
    int ip1,ip2,im1,im2,jp1,jp2,jm1,jm2,kp1,kp2,km1,km2;
    int i,j,ii,jj,kk;
    /* Potential, field, field gradient and multipole components */
    double pot, rfe[3], rfde[3][3], energy;
    double f, charge, *dipole, *quad;
    double qxx, qyx, qyy, qzx, qzy, qzz;


    VASSERT(thee != VNULL);
    VASSERT(thee->filled);

    /* Get the mesh information */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    xmax = thee->xf[nx-1];
    ymax = thee->yf[ny-1];
    zmax = thee->zf[nz-1];
    xmin = thee->xf[0];
    ymin = thee->yf[0];
    zmin = thee->zf[0];

    u = thee->u;

    atom = Valist_getAtom(thee->pbe->alist, atomID);

    /* Currently all atoms must be in the same partition. */

    VASSERT(atom->partID != 0);

    /* Convert the atom position to grid coordinates */

    position = Vatom_getPosition(atom);
    ifloat = (position[0] - xmin)/hx;
    jfloat = (position[1] - ymin)/hy;
    kfloat = (position[2] - zmin)/hzed;
    ip1 = (int)ceil(ifloat);
    ip2 = ip1 + 2;
    im1 = (int)floor(ifloat);
    im2 = im1 - 2;
```

```
    jp1 = (int)ceil(jfloat);
    jp2 = jp1 + 2;
    jm1 = (int)floor(jfloat);
    jm2 = jm1 - 2;
    kp1 = (int)ceil(kfloat);
    kp2 = kp1 + 2;
    km1 = (int)floor(kfloat);
    km2 = km1 - 2;

    /* This step shouldn't be necessary, but it saves nasty debugging
     * later on if something goes wrong */
    ip2 = VMIN2(ip2,nx-1);
    ip1 = VMIN2(ip1,nx-1);
    im1 = VMAX2(im1,0);
    im2 = VMAX2(im2,0);
    jp2 = VMIN2(jp2,ny-1);
    jp1 = VMIN2(jp1,ny-1);
    jm1 = VMAX2(jm1,0);
    jm2 = VMAX2(jm2,0);
    kp2 = VMIN2(kp2,nz-1);
    kp1 = VMIN2(kp1,nz-1);
    km1 = VMAX2(km1,0);
    km2 = VMAX2(km2,0);

    /* Initialize observables to zero */
    energy = 0.0;
    pot = 0.0;
    for (i=0;i<3;i++){
       rfe[i] = 0.0;
       for (j=0;j<3;j++){
         rfde[i][j] = 0.0;
       }
    }

    for (ii=im2; ii<=ip2; ii++) {
      mi = VFCHI4(ii,ifloat);
      mx = bspline4(mi);
      dmx = dbspline4(mi);
      d2mx = d2bspline4(mi);
      for (jj=jm2; jj<=jp2; jj++) {
        mj = VFCHI4(jj,jfloat);
        my = bspline4(mj);
        dmy = dbspline4(mj);
        d2my = d2bspline4(mj);
        for (kk=km2; kk<=kp2; kk++) {
          mk = VFCHI4(kk,kfloat);
          mz = bspline4(mk);
          dmz = dbspline4(mk);
          d2mz = d2bspline4(mk);
          f = u[IJK(ii,jj,kk)];
          /* potential */
          pot  += f*mx*my*mz;
          /* field */
          rfe[0] += f*dmx*my*mz/hx;
          rfe[1] += f*mx*dmy*mz/hy;
          rfe[2] += f*mx*my*dmz/hzed;
          /* field gradient */
          rfde[0][0] += f*d2mx*my*mz/(hx*hx);
          rfde[1][0] += f*dmx*dmy*mz/(hy*hx);
          rfde[1][1] += f*mx*d2my*mz/(hy*hy);
          rfde[2][0] += f*dmx*my*dmz/(hx*hzed);
          rfde[2][1] += f*mx*dmy*dmz/(hy*hzed);
          rfde[2][2] += f*mx*my*d2mz/(hzed*hzed);
        }
      }
    }

    charge = Vatom_getCharge(atom);
    dipole = Vatom_getDipole(atom);
    quad = Vatom_getQuadrupole(atom);
    qxx = quad[0]/3.0;
    qyx = quad[3]/3.0;
    qyy = quad[4]/3.0;
    qzx = quad[6]/3.0;
    qzy = quad[7]/3.0;
    qzz = quad[8]/3.0;

    energy =   pot * charge
             - rfe[0] * dipole[0]
             - rfe[1] * dipole[1]
             - rfe[2] * dipole[2]
```

```
            +     rfde[0][0]*qxx
            + 2.0*rfde[1][0]*qyx +     rfde[1][1]*qyy
            + 2.0*rfde[2][0]*qzx + 2.0*rfde[2][1]*qzy + rfde[2][2]*qzz;

    return energy;
}

VPUBLIC void Vpmg_fieldSpline4(Vpmg *thee, int atomID,
double field[3]) {

    Vatom *atom;
    double *u, f;
    /* Grid variables */
    int nx, ny, nz;
    double xmax, xmin, ymax, ymin, zmax, zmin;
    double hx, hy, hzed, ifloat, jfloat, kfloat;
    double *apos, position[3];
    /* B-Spline weights */
    double mx, my, mz, dmx, dmy, dmz;
    double mi, mj, mk;
    /* Loop indeces */
    int ip1,ip2,im1,im2,jp1,jp2,jm1,jm2,kp1,kp2,km1,km2;
    int i,j,ii,jj,kk;


    VASSERT (thee != VNULL);

    /* Get the mesh information */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    xmax = thee->xf[nx-1];
    ymax = thee->yf[ny-1];
    zmax = thee->zf[nz-1];
    xmin = thee->xf[0];
    ymin = thee->yf[0];
    zmin = thee->zf[0];

    u = thee->u;

    atom = Valist_getAtom(thee->pbe->alist, atomID);

    /* Currently all atoms must be in the same partition. */

    VASSERT (atom->partID != 0);

    /* Convert the atom position to grid coordinates */

    apos = Vatom_getPosition(atom);
    position[0] = apos[0] - xmin;
    position[1] = apos[1] - ymin;
    position[2] = apos[2] - zmin;
    ifloat = position[0]/hx;
    jfloat = position[1]/hy;
    kfloat = position[2]/hzed;
    ip1 = (int)ceil(ifloat);
    ip2 = ip1 + 2;
    im1 = (int)floor(ifloat);
    im2 = im1 - 2;
    jp1 = (int)ceil(jfloat);
    jp2 = jp1 + 2;
    jm1 = (int)floor(jfloat);
    jm2 = jm1 - 2;
    kp1 = (int)ceil(kfloat);
    kp2 = kp1 + 2;
    km1 = (int)floor(kfloat);
    km2 = km1 - 2;

    /* This step shouldn't be necessary, but it saves nasty debugging
     * later on if something goes wrong */
    ip2 = VMIN2(ip2,nx-1);
    ip1 = VMIN2(ip1,nx-1);
    im1 = VMAX2(im1,0);
    im2 = VMAX2(im2,0);
    jp2 = VMIN2(jp2,ny-1);
    jp1 = VMIN2(jp1,ny-1);
    jm1 = VMAX2(jm1,0);
    jm2 = VMAX2(jm2,0);
```

```
    kp2 = VMIN2(kp2,nz-1);
    kp1 = VMIN2(kp1,nz-1);
    km1 = VMAX2(km1,0);
    km2 = VMAX2(km2,0);

    for (i=0;i<3;i++){
       field[i] = 0.0;
    }

    for (ii=im2; ii<=ip2; ii++) {
      mi = VFCHI4(ii,ifloat);
      mx = bspline4(mi);
      dmx = dbspline4(mi);
      for (jj=jm2; jj<=jp2; jj++) {
        mj = VFCHI4(jj,jfloat);
        my = bspline4(mj);
        dmy = dbspline4(mj);
        for (kk=km2; kk<=kp2; kk++) {
          mk = VFCHI4(kk,kfloat);
          mz = bspline4(mk);
          dmz = dbspline4(mk);
          f = u[IJK(ii,jj,kk)];

          field[0] += f*dmx*my*mz/hx;
          field[1] += f*mx*dmy*mz/hy;
          field[2] += f*mx*my*dmz/hzed;
        }
      }
    }
}

VPUBLIC void Vpmg_qfPermanentMultipoleForce(Vpmg *thee, int atomID,
                                            double force[3], double torque[3]) {

    Vatom *atom;
    double f, *u, *apos, position[3];

    /* Grid variables */
    int nx,ny,nz;
    double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
    double hx, hy, hzed, ifloat, jfloat, kfloat;

    /* B-spline weights */
    double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz, d3mx, d3my, d3mz;
    double mi, mj, mk;

    /* Loop indeces */
    int i, j, k, ii, jj, kk;
    int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;

    /* Potential, field, field gradient and 2nd field gradient */
    double pot, e[3], de[3][3], d2e[3][3][3];

    /* Permanent multipole components */
    double *dipole, *quad;
    double c, ux, uy, uz, qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;

    VASSERT(thee != VNULL);
    VASSERT(thee->filled);

    atom = Valist_getAtom(thee->pbe->alist, atomID);

    /* Currently all atoms must be in the same partition. */

    VASSERT(atom->partID != 0);

    apos = Vatom_getPosition(atom);

    c = Vatom_getCharge(atom);
    dipole = Vatom_getDipole(atom);
    ux = dipole[0];
    uy = dipole[1];
    uz = dipole[2];
    quad = Vatom_getQuadrupole(atom);
    qxx = quad[0]/3.0;
    qxy = quad[1]/3.0;
    qxz = quad[2]/3.0;
    qyx = quad[3]/3.0;
    qyy = quad[4]/3.0;
```

```
    qyz = quad[5]/3.0;
    qzx = quad[6]/3.0;
    qzy = quad[7]/3.0;
    qzz = quad[8]/3.0;

    /* Initialize observables */
    pot = 0.0;
    for (i=0;i<3;i++){
       e[i] = 0.0;
       for (j=0;j<3;j++){
          de[i][j] = 0.0;
          for (k=0;k<3;k++){
             d2e[i][j][k] = 0.0;
          }
       }
    }

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;
    xmin = thee->pmgp->xmin;
    ymin = thee->pmgp->ymin;
    zmin = thee->pmgp->zmin;
    xmax = thee->pmgp->xmax;
    ymax = thee->pmgp->ymax;
    zmax = thee->pmgp->zmax;
    u = thee->u;

    /* Make sure we're on the grid */
    if ((apos[0]<=(xmin+2*hx))   || (apos[0]>=(xmax-2*hx)) \
     || (apos[1]<=(ymin+2*hy))   || (apos[1]>=(ymax-2*hy)) \
     || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
        Vnm_print(2, "qfPermanentMultipoleForce:  Atom off the mesh (ignoring)
%6.3f %6.3f %6.3f\n", apos[0], apos[1], apos[2]);
        fflush(stderr);
    } else {

        /* Convert the atom position to grid coordinates */
        position[0] = apos[0] - xmin;
        position[1] = apos[1] - ymin;
        position[2] = apos[2] - zmin;
        ifloat = position[0]/hx;
        jfloat = position[1]/hy;
        kfloat = position[2]/hzed;
        ip1 = (int)ceil(ifloat);
        ip2 = ip1 + 2;
        im1 = (int)floor(ifloat);
        im2 = im1 - 2;
        jp1 = (int)ceil(jfloat);
        jp2 = jp1 + 2;
        jm1 = (int)floor(jfloat);
        jm2 = jm1 - 2;
        kp1 = (int)ceil(kfloat);
        kp2 = kp1 + 2;
        km1 = (int)floor(kfloat);
        km2 = km1 - 2;

        /* This step shouldn't be necessary, but it saves nasty debugging
         * later on if something goes wrong */
        ip2 = VMIN2(ip2,nx-1);
        ip1 = VMIN2(ip1,nx-1);
        im1 = VMAX2(im1,0);
        im2 = VMAX2(im2,0);
        jp2 = VMIN2(jp2,ny-1);
        jp1 = VMIN2(jp1,ny-1);
        jm1 = VMAX2(jm1,0);
        jm2 = VMAX2(jm2,0);
        kp2 = VMIN2(kp2,nz-1);
        kp1 = VMIN2(kp1,nz-1);
        km1 = VMAX2(km1,0);
        km2 = VMAX2(km2,0);

        for (ii=im2; ii<=ip2; ii++) {
            mi = VFCHI4(ii,ifloat);
```

```
            mx = bspline4(mi);
            dmx = dbspline4(mi);
            d2mx = d2bspline4(mi);
            d3mx = d3bspline4(mi);
            for (jj=jm2; jj<=jp2; jj++) {
                mj = VFCHI4(jj,jfloat);
                my = bspline4(mj);
                dmy = dbspline4(mj);
                d2my = d2bspline4(mj);
                d3my = d3bspline4(mj);
                for (kk=km2; kk<=kp2; kk++) {
                    mk = VFCHI4(kk,kfloat);
                    mz = bspline4(mk);
                    dmz = dbspline4(mk);
                    d2mz = d2bspline4(mk);
                    d3mz = d3bspline4(mk);
                    f = u[IJK(ii,jj,kk)];
                    /* Potential */
                    pot  += f*mx*my*mz;
                    /* Field */
                    e[0] += f*dmx*my*mz/hx;
                    e[1] += f*mx*dmy*mz/hy;
                    e[2] += f*mx*my*dmz/hzed;
                    /* Field gradient */
                    de[0][0] += f*d2mx*my*mz/(hx*hx);
                    de[1][0] += f*dmx*dmy*mz/(hy*hx);
                    de[1][1] += f*mx*d2my*mz/(hy*hy);
                    de[2][0] += f*dmx*my*dmz/(hx*hzed);
                    de[2][1] += f*mx*dmy*dmz/(hy*hzed);
                    de[2][2] += f*mx*my*d2mz/(hzed*hzed);
                    /* 2nd Field Gradient
                       VxVxVa */
                    d2e[0][0][0] += f*d3mx*my*mz /(hx*hx*hx);
                    d2e[0][0][1] += f*d2mx*dmy*mz/(hx*hy*hx);
                    d2e[0][0][2] += f*d2mx*my*dmz/(hx*hx*hzed);
                    /* VyVxVa */
                    d2e[1][0][0] += f*d2mx*dmy*mz/(hx*hx*hy);
                    d2e[1][0][1] += f*dmx*d2my*mz/(hx*hy*hy);
                    d2e[1][0][2] += f*dmx*dmy*dmz/(hx*hy*hzed);
                    /* VyVyVa */
                    d2e[1][1][0] += f*dmx*d2my*mz/(hx*hy*hy);
                    d2e[1][1][1] += f*mx*d3my*mz /(hy*hy*hy);
                    d2e[1][1][2] += f*mx*d2my*dmz/(hy*hy*hzed);
                    /* VzVxVa */
                    d2e[2][0][0] += f*d2mx*my*dmz/(hx*hx*hzed);
                    d2e[2][0][1] += f*dmx*dmy*dmz/(hx*hy*hzed);
                    d2e[2][0][2] += f*dmx*my*d2mz/(hx*hzed*hzed);
                    /* VzVyVa */
                    d2e[2][1][0] += f*dmx*dmy*dmz/(hx*hy*hzed);
                    d2e[2][1][1] += f*mx*d2my*dmz/(hy*hy*hzed);
                    d2e[2][1][2] += f*mx*dmy*d2mz/(hy*hzed*hzed);
                    /* VzVzVa */
                    d2e[2][2][0] += f*dmx*my*d2mz/(hx*hzed*hzed);
                    d2e[2][2][1] += f*mx*dmy*d2mz/(hy*hzed*hzed);
                    d2e[2][2][2] += f*mx*my*d3mz /(hzed*hzed*hzed);
                }
            }
        }
    }

    /* Monopole Force */
    force[0] = e[0]*c;
    force[1] = e[1]*c;
    force[2] = e[2]*c;

    /* Dipole Force */
    force[0] -= de[0][0]*ux+de[1][0]*uy+de[2][0]*uz;
    force[1] -= de[1][0]*ux+de[1][1]*uy+de[2][1]*uz;
    force[2] -= de[2][0]*ux+de[2][1]*uy+de[2][2]*uz;

    /* Quadrupole Force */
    force[0] += d2e[0][0][0]*qxx
             +  d2e[1][0][0]*qyx*2.0+d2e[1][1][0]*qyy
             +  d2e[2][0][0]*qzx*2.0+d2e[2][1][0]*qzy*2.0+d2e[2][2][0]*qzz;
    force[1] += d2e[0][0][1]*qxx
             +  d2e[1][0][1]*qyx*2.0+d2e[1][1][1]*qyy
             +  d2e[2][0][1]*qzx*2.0+d2e[2][1][1]*qzy*2.0+d2e[2][2][1]*qzz;
    force[2] += d2e[0][0][2]*qxx
             +  d2e[1][0][2]*qyx*2.0+d2e[1][1][2]*qyy
             +  d2e[2][0][2]*qzx*2.0+d2e[2][1][2]*qzy*2.0+d2e[2][2][2]*qzz;

```

```
    /* Dipole Torque */
    torque[0] = uy * e[2] - uz * e[1];
    torque[1] = uz * e[0] - ux * e[2];
    torque[2] = ux * e[1] - uy * e[0];
    /* Quadrupole Torque */
    de[0][1] = de[1][0];
    de[0][2] = de[2][0];
    de[1][2] = de[2][1];
    torque[0] -= 2.0*(qyx*de[0][2] + qyy*de[1][2] + qyz*de[2][2]
                    - qzx*de[0][1] - qzy*de[1][1] - qzz*de[2][1]);
    torque[1] -= 2.0*(qzx*de[0][0] + qzy*de[1][0] + qzz*de[2][0]
                    - qxx*de[0][2] - qxy*de[1][2] - qxz*de[2][2]);
    torque[2] -= 2.0*(qxx*de[0][1] + qxy*de[1][1] + qxz*de[2][1]
                    - qyx*de[0][0] - qyy*de[1][0] - qyz*de[2][0]);


    /* printf(" qPhi Force %f %f %f\n", force[0], force[1], force[2]);
       printf(" qPhi Torque %f %f %f\n", torque[0], torque[1], torque[2]); */
}

VPUBLIC void Vpmg_ibPermanentMultipoleForce(Vpmg *thee, int atomID,
                                            double force[3]) {

    Valist *alist;
    Vacc *acc;
    Vpbe *pbe;
    Vatom *atom;
    Vsurf_Meth srfm;

    /* Grid variables */
    double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
    double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;
    double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
    double izmagic;
    int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;

    VASSERT(thee != VNULL);

    /* Nonlinear PBE is not implemented for AMOEBA */
    VASSERT(!thee->pmgp->nonlin);

    acc = thee->pbe->acc;
    srfm = thee->surfMeth;
    atom = Valist_getAtom(thee->pbe->alist, atomID);

    /* Currently all atoms must be in the same partition. */

    VASSERT(atom->partID != 0);
    apos = Vatom_getPosition(atom);
    arad = Vatom_getRadius(atom);

    /* Reset force */
    force[0] = 0.0;
    force[1] = 0.0;
    force[2] = 0.0;

    /* Get PBE info */
    pbe = thee->pbe;
    acc = pbe->acc;
    alist = pbe->alist;
    irad = Vpbe_getMaxIonRadius(pbe);
    zkappa2 = Vpbe_getZkappa2(pbe);
    izmagic = 1.0/Vpbe_getZmagic(pbe);

    /* Should be a check for this further up. */
    VASSERT (zkappa2 > VPMGSMALL);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;
    xmin = thee->pmgp->xmin;
    ymin = thee->pmgp->ymin;
```

```
    zmin = thee->pmgp->zmin;
    xmax = thee->pmgp->xmax;
    ymax = thee->pmgp->ymax;
    zmax = thee->pmgp->zmax;

    /* Make sure we're on the grid */
    if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
      (apos[1]<=ymin) || (apos[1]>=ymax)  || \
      (apos[2]<=zmin) || (apos[2]>=zmax)) {
        Vnm_print(2, "ibPermanentMultipoleForce:  Atom %d at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):\n", atomID, apos[0], apos[1], apos[2]);
        Vnm_print(2, "ibPermanentMultipoleForce:  xmin = %g, xmax = %g\n", xmin, xmax);
        Vnm_print(2, "ibPermanentMultipoleForce:  ymin = %g, ymax = %g\n", ymin, ymax);
        Vnm_print(2, "ibPermanentMultipoleForce:  zmin = %g, zmax = %g\n", zmin, zmax);
        fflush(stderr);
    } else {

        /* Convert the atom position to grid reference frame */
        position[0] = apos[0] - xmin;
        position[1] = apos[1] - ymin;
        position[2] = apos[2] - zmin;

        /* Integrate over points within this atom's (inflated) radius */
        rtot = (irad + arad + thee->splineWin);
        rtot2 = VSQR(rtot);
        dx = rtot + 0.5*hx;
        imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
        imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
        for (i=imin; i<=imax; i++) {
            dx2 = VSQR(position[0] - hx*i);
            if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
            else dy = 0.5*hy;
            jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
            jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
            for (j=jmin; j<=jmax; j++) {
                dy2 = VSQR(position[1] - hy*j);
                if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
                else dz = 0.5*hzed;
                kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
                kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
                for (k=kmin; k<=kmax; k++) {
                    dz2 = VSQR(k*hzed - position[2]);
                    /* See if grid point is inside ivdw radius and set ccf
                     * accordingly (do spline assignment here) */
                    if ((dz2 + dy2 + dx2) <= rtot2) {
                        gpos[0] = i*hx + xmin;
                        gpos[1] = j*hy + ymin;
                        gpos[2] = k*hzed + zmin;
                        Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, irad, atom, tgrad);
                        fmag = VSQR(thee->u[IJK(i,j,k)])*thee->kappa[IJK(i,j,k)];
                        force[0] += (zkappa2*fmag*tgrad[0]);
                        force[1] += (zkappa2*fmag*tgrad[1]);
                        force[2] += (zkappa2*fmag*tgrad[2]);
                    }
                } /* k loop */
            } /* j loop */
        } /* i loop */
    }

    force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
    force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
    force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;

}

VPUBLIC void Vpmg_dbPermanentMultipoleForce(Vpmg *thee, int atomID,
                                            double force[3]) {

    Vacc *acc;
    Vpbe *pbe;
    Vatom *atom;
    Vsurf_Meth srfm;

    double *apos, position[3], arad, hx, hy, hzed, izmagic, deps, depsi;
```

```
    double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2, epsp;
    double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
    double *u, Hxijk, Hyijk, Hzijk, Hxim1jk, Hyijm1k, Hzijkm1;
    double dHxijk[3], dHyijk[3], dHzijk[3], dHxim1jk[3], dHyijm1k[3];
    double dHzijkm1[3];
    int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;

    VASSERT(thee != VNULL);

    acc = thee->pbe->acc;
    srfm = thee->surfMeth;
    atom = Valist_getAtom(thee->pbe->alist, atomID);

    /* Currently all atoms must be in the same partition. */

    VASSERT(atom->partID != 0);
    arad = Vatom_getRadius(atom);
    apos = Vatom_getPosition(atom);

    /* Reset force */
    force[0] = 0.0;
    force[1] = 0.0;
    force[2] = 0.0;

    /* Get PBE info */
    pbe = thee->pbe;
    acc = pbe->acc;
    epsp = Vpbe_getSoluteDiel(pbe);
    epsw = Vpbe_getSolventDiel(pbe);
    kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*Vunit_kb;
    izmagic = 1.0/Vpbe_getZmagic(pbe);


    deps = (epsw - epsp);
    depsi = 1.0/deps;

    VASSERT(VABS(deps) > VPMGSMALL);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;
    xmin = thee->pmgp->xmin;
    ymin = thee->pmgp->ymin;
    zmin = thee->pmgp->zmin;
    xmax = thee->pmgp->xmax;
    ymax = thee->pmgp->ymax;
    zmax = thee->pmgp->zmax;
    u = thee->u;

    /* Make sure we're on the grid */
    if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
      (apos[1]<=ymin) || (apos[1]>=ymax)  || \
      (apos[2]<=zmin) || (apos[2]>=zmax)) {
        Vnm_print(2, "dbPermanentMultipoleForce:  Atom at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):\n", apos[0], apos[1], apos[2]);
        Vnm_print(2, "dbPermanentMultipoleForce:  xmin = %g, xmax = %g\n", xmin, xmax);
        Vnm_print(2, "dbPermanentMultipoleForce:  ymin = %g, ymax = %g\n", ymin, ymax);
        Vnm_print(2, "dbPermanentMultipoleForce:  zmin = %g, zmax = %g\n", zmin, zmax);
        fflush(stderr);
    } else {

        /* Convert the atom position to grid reference frame */
        position[0] = apos[0] - xmin;
        position[1] = apos[1] - ymin;
        position[2] = apos[2] - zmin;

        /* Integrate over points within this atom's (inflated) radius */
        rtot = (arad + thee->splineWin);
        rtot2 = VSQR(rtot);
```

```
        dx = rtot/hx;
        imin = (int)floor((position[0]-rtot)/hx);
        if (imin < 1) {
            Vnm_print(2, "dbPermanentMultipoleForce:  Atom off grid!\n");
            return;
        }
        imax = (int)ceil((position[0]+rtot)/hx);
        if (imax > (nx-2)) {
            Vnm_print(2, "dbPermanentMultipoleForce:  Atom off grid!\n");
            return;
        }
        jmin = (int)floor((position[1]-rtot)/hy);
        if (jmin < 1) {
            Vnm_print(2, "dbPermanentMultipoleForce:  Atom off grid!\n");
            return;
        }
        jmax = (int)ceil((position[1]+rtot)/hy);
        if (jmax > (ny-2)) {
            Vnm_print(2, "dbPermanentMultipoleForce:  Atom off grid!\n");
            return;
        }
        kmin = (int)floor((position[2]-rtot)/hzed);
        if (kmin < 1) {
            Vnm_print(2, "dbPermanentMultipoleForce:  Atom off grid!\n");
            return;
        }
        kmax = (int)ceil((position[2]+rtot)/hzed);
        if (kmax > (nz-2)) {
            Vnm_print(2, "dbPermanentMultipoleForce:  Atom off grid!\n");
            return;
        }
        for (i=imin; i<=imax; i++) {
            for (j=jmin; j<=jmax; j++) {
                for (k=kmin; k<=kmax; k++) {
                    /* i,j,k */
                    gpos[0] = (i+0.5)*hx + xmin;
                    gpos[1] = j*hy + ymin;
                    gpos[2] = k*hzed + zmin;
                    Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
                    Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
                            atom, dHxijk);
                    for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
                    gpos[0] = i*hx + xmin;
                    gpos[1] = (j+0.5)*hy + ymin;
                    gpos[2] = k*hzed + zmin;
                    Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
                    Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
                            atom, dHyijk);
                    for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
                    gpos[0] = i*hx + xmin;
                    gpos[1] = j*hy + ymin;
                    gpos[2] = (k+0.5)*hzed + zmin;
                    Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
                    Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
                            atom, dHzijk);
                    for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
                    /* i-1,j,k */
                    gpos[0] = (i-0.5)*hx + xmin;
                    gpos[1] = j*hy + ymin;
                    gpos[2] = k*hzed + zmin;
                    Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
                    Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
                            atom, dHxim1jk);
                    for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
                    /* i,j-1,k */
                    gpos[0] = i*hx + xmin;
                    gpos[1] = (j-0.5)*hy + ymin;
                    gpos[2] = k*hzed + zmin;
                    Hyijm1k = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
                    Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
                            atom, dHyijm1k);
                    for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;
                    /* i,j,k-1 */
                    gpos[0] = i*hx + xmin;
                    gpos[1] = j*hy + ymin;
                    gpos[2] = (k-0.5)*hzed + zmin;
```

```
                    Hzijkm1 = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
                    Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
                            atom, dHzijkm1);
                    for (l=0; l<3; l++) dHzijkm1[l] *= Hzijkm1;
                    dbFmag = u[IJK(i,j,k)];
                    tgrad[0] =
                       (dHxijk[0]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
                     +  dHxim1jk[0]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
                     + (dHyijk[0]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
                     +  dHyijm1k[0]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
                     + (dHzijk[0]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
                     + dHzijkm1[0]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
                    tgrad[1] =
                       (dHxijk[1]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
                     +  dHxim1jk[1]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
                     + (dHyijk[1]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
                     +  dHyijm1k[1]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
                     + (dHzijk[1]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
                     + dHzijkm1[1]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
                    tgrad[2] =
                       (dHxijk[2]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
                     +  dHxim1jk[2]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
                     + (dHyijk[2]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
                     +  dHyijm1k[2]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
                     + (dHzijk[2]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
                     + dHzijkm1[2]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
                     force[0] += (dbFmag*tgrad[0]);
                     force[1] += (dbFmag*tgrad[1]);
                     force[2] += (dbFmag*tgrad[2]);
                 } /* k loop */
             } /* j loop */
         } /* i loop */
         force[0] = -force[0]*hx*hy*hzed*deps*0.5*izmagic;
         force[1] = -force[1]*hx*hy*hzed*deps*0.5*izmagic;
         force[2] = -force[2]*hx*hy*hzed*deps*0.5*izmagic;
     }
}

VPUBLIC void Vpmg_qfDirectPolForce(Vpmg *thee, Vgrid * perm, Vgrid *induced,
                                   int atomID, double force[3], double torque[3]) {

    Vatom *atom;
    Vpbe *pbe;
    double f, fp, *u, *up, *apos, position[3];

    /* Grid variables */
    int nx,ny,nz;
    double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
    double hx, hy, hzed, ifloat, jfloat, kfloat;

    /* B-spline weights */
    double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz, d3mx, d3my, d3mz;
    double mi, mj, mk;

    /* Loop indeces */
    int i, j, k, ii, jj, kk;
    int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;

    /* Permanent potential, field, field gradient and 2nd field gradient */
    double pot, e[3], de[3][3], d2e[3][3][3];
    /* Induced dipole field */
    double dep[3][3];

    /* Permanent multipole components */
    double *dipole, *quad;
    double c, ux, uy, uz, qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
    double uix, uiy, uiz;

    VASSERT(thee != VNULL);
    VASSERT(induced != VNULL); /* the potential due to permanent multipoles.*/
    VASSERT(induced != VNULL); /* the potential due to local induced dipoles.*/
    VASSERT(thee->pbe != VNULL);
    VASSERT(thee->pbe->alist != VNULL);

    atom = Valist_getAtom(thee->pbe->alist, atomID);
    VASSERT(atom->partID != 0); /* all atoms must be in the same partition.*/
```

```
    apos = Vatom_getPosition(atom);

    c = Vatom_getCharge(atom);
    dipole = Vatom_getDipole(atom);
    ux = dipole[0];
    uy = dipole[1];
    uz = dipole[2];
    quad = Vatom_getQuadrupole(atom);
    qxx = quad[0]/3.0;
    qxy = quad[1]/3.0;
    qxz = quad[2]/3.0;
    qyx = quad[3]/3.0;
    qyy = quad[4]/3.0;
    qyz = quad[5]/3.0;
    qzx = quad[6]/3.0;
    qzy = quad[7]/3.0;
    qzz = quad[8]/3.0;

    dipole = Vatom_getInducedDipole(atom);
    uix = dipole[0];
    uiy = dipole[1];
    uiz = dipole[2];

    /* Reset Field Gradients */
    pot = 0.0;
    for (i=0;i<3;i++){
       e[i] = 0.0;
       for (j=0;j<3;j++){
          de[i][j] = 0.0;
          dep[i][j] = 0.0;
          for (k=0;k<3;k++){
             d2e[i][j][k] = 0.0;
          }
       }
    }

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;
    xmin = thee->pmgp->xmin;
    ymin = thee->pmgp->ymin;
    zmin = thee->pmgp->zmin;
    xmax = thee->pmgp->xmax;
    ymax = thee->pmgp->ymax;
    zmax = thee->pmgp->zmax;
    u = induced->data;
    up = perm->data;

    /* Make sure we're on the grid */
    if ((apos[0]<=(xmin+2*hx))   || (apos[0]>=(xmax-2*hx)) \
     || (apos[1]<=(ymin+2*hy))   || (apos[1]>=(ymax-2*hy)) \
     || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
        Vnm_print(2, "qfDirectPolForce:  Atom off the mesh (ignoring) %6.3f
%6.3f %6.3f\n", apos[0], apos[1], apos[2]);
        fflush(stderr);

    } else {

        /* Convert the atom position to grid coordinates */
        position[0] = apos[0] - xmin;
        position[1] = apos[1] - ymin;
        position[2] = apos[2] - zmin;
        ifloat = position[0]/hx;
        jfloat = position[1]/hy;
        kfloat = position[2]/hzed;
        ip1 = (int)ceil(ifloat);
        ip2 = ip1 + 2;
        im1 = (int)floor(ifloat);
        im2 = im1 - 2;
        jp1 = (int)ceil(jfloat);
        jp2 = jp1 + 2;
        jm1 = (int)floor(jfloat);
        jm2 = jm1 - 2;
        kp1 = (int)ceil(kfloat);
```

```
        kp2 = kp1 + 2;
        km1 = (int)floor(kfloat);
        km2 = km1 - 2;

        /* This step shouldn't be necessary, but it saves nasty debugging
         * later on if something goes wrong */
        ip2 = VMIN2(ip2,nx-1);
        ip1 = VMIN2(ip1,nx-1);
        im1 = VMAX2(im1,0);
        im2 = VMAX2(im2,0);
        jp2 = VMIN2(jp2,ny-1);
        jp1 = VMIN2(jp1,ny-1);
        jm1 = VMAX2(jm1,0);
        jm2 = VMAX2(jm2,0);
        kp2 = VMIN2(kp2,nz-1);
        kp1 = VMIN2(kp1,nz-1);
        km1 = VMAX2(km1,0);
        km2 = VMAX2(km2,0);

        for (ii=im2; ii<=ip2; ii++) {
            mi = VFCHI4(ii,ifloat);
            mx = bspline4(mi);
            dmx = dbspline4(mi);
            d2mx = d2bspline4(mi);
            d3mx = d3bspline4(mi);
            for (jj=jm2; jj<=jp2; jj++) {
                mj = VFCHI4(jj,jfloat);
                my = bspline4(mj);
                dmy = dbspline4(mj);
                d2my = d2bspline4(mj);
                d3my = d3bspline4(mj);
                for (kk=km2; kk<=kp2; kk++) {
                    mk = VFCHI4(kk,kfloat);
                    mz = bspline4(mk);
                    dmz = dbspline4(mk);
                    d2mz = d2bspline4(mk);
                    d3mz = d3bspline4(mk);
                    f = u[IJK(ii,jj,kk)];
                    fp = up[IJK(ii,jj,kk)];
                    /* The potential */
                    pot  += f*mx*my*mz;
                    /* The field */
                    e[0] += f*dmx*my*mz/hx;
                    e[1] += f*mx*dmy*mz/hy;
                    e[2] += f*mx*my*dmz/hzed;
                    /* The gradient of the field */
                    de[0][0] += f*d2mx*my*mz/(hx*hx);
                    de[1][0] += f*dmx*dmy*mz/(hy*hx);
                    de[1][1] += f*mx*d2my*mz/(hy*hy);
                    de[2][0] += f*dmx*my*dmz/(hx*hzed);
                    de[2][1] += f*mx*dmy*dmz/(hy*hzed);
                    de[2][2] += f*mx*my*d2mz/(hzed*hzed);
                    /* The gradient of the (permanent) field */
                    dep[0][0] += fp*d2mx*my*mz/(hx*hx);
                    dep[1][0] += fp*dmx*dmy*mz/(hy*hx);
                    dep[1][1] += fp*mx*d2my*mz/(hy*hy);
                    dep[2][0] += fp*dmx*my*dmz/(hx*hzed);
                    dep[2][1] += fp*mx*dmy*dmz/(hy*hzed);
                    dep[2][2] += fp*mx*my*d2mz/(hzed*hzed);
                    /* The 2nd gradient of the field
                       VxVxVa */
                    d2e[0][0][0] += f*d3mx*my*mz /(hx*hx*hx);
                    d2e[0][0][1] += f*d2mx*dmy*mz/(hx*hy*hx);
                    d2e[0][0][2] += f*d2mx*my*dmz/(hx*hx*hzed);
                    /* VyVxVa */
                    d2e[1][0][0] += f*d2mx*dmy*mz/(hx*hx*hy);
                    d2e[1][0][1] += f*dmx*d2my*mz/(hx*hy*hy);
                    d2e[1][0][2] += f*dmx*dmy*dmz/(hx*hy*hzed);
                    /* VyVyVa */
                    d2e[1][1][0] += f*dmx*d2my*mz/(hx*hy*hy);
                    d2e[1][1][1] += f*mx*d3my*mz /(hy*hy*hy);
                    d2e[1][1][2] += f*mx*d2my*dmz/(hy*hy*hzed);
                    /* VzVxVa */
                    d2e[2][0][0] += f*d2mx*my*dmz/(hx*hx*hzed);
                    d2e[2][0][1] += f*dmx*dmy*dmz/(hx*hy*hzed);
                    d2e[2][0][2] += f*dmx*my*d2mz/(hx*hzed*hzed);
                    /* VzVyVa */
                    d2e[2][1][0] += f*dmx*dmy*dmz/(hx*hy*hzed);
                    d2e[2][1][1] += f*mx*d2my*dmz/(hy*hy*hzed);
                    d2e[2][1][2] += f*mx*dmy*d2mz/(hy*hzed*hzed);
                    /* VzVzVa */
```

```
                    d2e[2][2][0] += f*dmx*my*d2mz/(hx*hzed*hzed);
                    d2e[2][2][1] += f*mx*dmy*d2mz/(hy*hzed*hzed);
                    d2e[2][2][2] += f*mx*my*d3mz /(hzed*hzed*hzed);
                }
            }
        }
    }

    /* force on permanent multipole due to induced reaction field */

    /* Monopole Force */
    force[0] = e[0]*c;
    force[1] = e[1]*c;
    force[2] = e[2]*c;

    /* Dipole Force */
    force[0] -= de[0][0]*ux+de[1][0]*uy+de[2][0]*uz;
    force[1] -= de[1][0]*ux+de[1][1]*uy+de[2][1]*uz;
    force[2] -= de[2][0]*ux+de[2][1]*uy+de[2][2]*uz;

    /* Quadrupole Force */
    force[0] += d2e[0][0][0]*qxx
             +  d2e[1][0][0]*qyx*2.0+d2e[1][1][0]*qyy
             +  d2e[2][0][0]*qzx*2.0+d2e[2][1][0]*qzy*2.0+d2e[2][2][0]*qzz;
    force[1] += d2e[0][0][1]*qxx
             +  d2e[1][0][1]*qyx*2.0+d2e[1][1][1]*qyy
             +  d2e[2][0][1]*qzx*2.0+d2e[2][1][1]*qzy*2.0+d2e[2][2][1]*qzz;
    force[2] += d2e[0][0][2]*qxx
             +  d2e[1][0][2]*qyx*2.0+d2e[1][1][2]*qyy
             +  d2e[2][0][2]*qzx*2.0+d2e[2][1][2]*qzy*2.0+d2e[2][2][2]*qzz;

    /* torque on permanent mulitpole due to induced reaction field */

    /* Dipole Torque */
    torque[0] = uy * e[2] - uz * e[1];
    torque[1] = uz * e[0] - ux * e[2];
    torque[2] = ux * e[1] - uy * e[0];

    /* Quadrupole Torque */
    /* Tx = -2.0*(Sum_a (Qya*dEaz) + Sum_b (Qzb*dEby))
       Ty = -2.0*(Sum_a (Qza*dEax) + Sum_b (Qxb*dEbz))
       Tz = -2.0*(Sum_a (Qxa*dEay) + Sum_b (Qyb*dEbx))  */
    de[0][1] = de[1][0];
    de[0][2] = de[2][0];
    de[1][2] = de[2][1];
    torque[0] -= 2.0*(qyx*de[0][2] + qyy*de[1][2] + qyz*de[2][2]
                    - qzx*de[0][1] - qzy*de[1][1] - qzz*de[2][1]);
    torque[1] -= 2.0*(qzx*de[0][0] + qzy*de[1][0] + qzz*de[2][0]
                    - qxx*de[0][2] - qxy*de[1][2] - qxz*de[2][2]);
    torque[2] -= 2.0*(qxx*de[0][1] + qxy*de[1][1] + qxz*de[2][1]
                    - qyx*de[0][0] - qyy*de[1][0] - qyz*de[2][0]);

    /* force on induced dipole due to permanent reaction field */

    force[0] -= dep[0][0]*uix+dep[1][0]*uiy+dep[2][0]*uiz;
    force[1] -= dep[1][0]*uix+dep[1][1]*uiy+dep[2][1]*uiz;
    force[2] -= dep[2][0]*uix+dep[2][1]*uiy+dep[2][2]*uiz;

    force[0] = 0.5 * force[0];
    force[1] = 0.5 * force[1];
    force[2] = 0.5 * force[2];
    torque[0] = 0.5 * torque[0];
    torque[1] = 0.5 * torque[1];
    torque[2] = 0.5 * torque[2];

    /* printf(" qPhi Force %f %f %f\n", force[0], force[1], force[2]);
      printf(" qPhi Torque %f %f %f\n", torque[0], torque[1], torque[2]); */
}

VPUBLIC void Vpmg_qfNLDirectPolForce(Vpmg *thee,
Vgrid *perm, Vgrid *nlInduced,
                                     int atomID, double force[3], double torque
[3]) {

    Vatom *atom;
    double *apos, *dipole, *quad, position[3], hx, hy, hzed;
    double xlen, ylen, zlen, xmin, ymin, zmin, xmax
, ymax, zmax;
    double pot, e[3],de[3][3],dep[3][3],d2e[3][3][3];
    double mx, my, mz, dmx, dmy, dmz, mi, mj, mk;
    double d2mx, d2my, d2mz, d3mx, d3my, d3mz;
```

```
    double *u, *up, charge, ifloat, jfloat, kfloat;
    double f, fp, c, ux, uy, uz, qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
    double uix, uiy, uiz;
    int i,j,k,nx, ny, nz, im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1;
    int kp1, kp2, ii, jj, kk;

    VASSERT(thee != VNULL);
    VASSERT(perm != VNULL);      /* potential due to permanent multipoles. */
    VASSERT(nlInduced != VNULL); /* potential due to non-local induced dipoles */
    VASSERT(!thee->pmgp->nonlin); /* Nonlinear PBE is not implemented for AMOEBA */

    atom = Valist_getAtom(thee->pbe->alist, atomID);
    VASSERT(atom->partID != 0);   /* Currently all atoms must be in the same partition. */
    apos = Vatom_getPosition(atom);

    c = Vatom_getCharge(atom);
    dipole = Vatom_getDipole(atom);
    ux = dipole[0];
    uy = dipole[1];
    uz = dipole[2];
    quad = Vatom_getQuadrupole(atom);
    qxx = quad[0]/3.0;
    qxy = quad[1]/3.0;
    qxz = quad[2]/3.0;
    qyx = quad[3]/3.0;
    qyy = quad[4]/3.0;
    qyz = quad[5]/3.0;
    qzx = quad[6]/3.0;
    qzy = quad[7]/3.0;
    qzz = quad[8]/3.0;

    dipole = Vatom_getNLInducedDipole(atom);
    uix = dipole[0];
    uiy = dipole[1];
    uiz = dipole[2];

    /* Reset Field Gradients */
    pot = 0.0;
    for (i=0;i<3;i++){
       e[i] = 0.0;
       for (j=0;j<3;j++){
          de[i][j] = 0.0;
          dep[i][j] = 0.0;
          for (k=0;k<3;k++){
             d2e[i][j][k] = 0.0;
          }
       }
    }

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;
    xmin = thee->pmgp->xmin;
    ymin = thee->pmgp->ymin;
    zmin = thee->pmgp->zmin;
    xmax = thee->pmgp->xmax;
    ymax = thee->pmgp->ymax;
    zmax = thee->pmgp->zmax;
    u = nlInduced->data;
    up = perm->data;


    /* Make sure we're on the grid */
    if ((apos[0]<=(xmin+2*hx))   || (apos[0]>=(xmax-2*hx)) \
     || (apos[1]<=(ymin+2*hy))   || (apos[1]>=(ymax-2*hy)) \
     || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
        Vnm_print(2, "qfNLDirectMultipoleForce:  Atom off the mesh (ignoring) %6.3f %6.3f %6.3f\n", apos[0], apos[1], apos[2]);
    } else {

```

```
        /* Convert the atom position to grid coordinates */
        position[0] = apos[0] - xmin;
        position[1] = apos[1] - ymin;
        position[2] = apos[2] - zmin;
        ifloat = position[0]/hx;
        jfloat = position[1]/hy;
        kfloat = position[2]/hzed;
        ip1 = (int)ceil(ifloat);
        ip2 = ip1 + 2;
        im1 = (int)floor(ifloat);
        im2 = im1 - 2;
        jp1 = (int)ceil(jfloat);
        jp2 = jp1 + 2;
        jm1 = (int)floor(jfloat);
        jm2 = jm1 - 2;
        kp1 = (int)ceil(kfloat);
        kp2 = kp1 + 2;
        km1 = (int)floor(kfloat);
        km2 = km1 - 2;

        /* This step shouldn't be necessary, but it saves nasty debugging
         * later on if something goes wrong */
        ip2 = VMIN2(ip2,nx-1);
        ip1 = VMIN2(ip1,nx-1);
        im1 = VMAX2(im1,0);
        im2 = VMAX2(im2,0);
        jp2 = VMIN2(jp2,ny-1);
        jp1 = VMIN2(jp1,ny-1);
        jm1 = VMAX2(jm1,0);
        jm2 = VMAX2(jm2,0);
        kp2 = VMIN2(kp2,nz-1);
        kp1 = VMIN2(kp1,nz-1);
        km1 = VMAX2(km1,0);
        km2 = VMAX2(km2,0);

        for (ii=im2; ii<=ip2; ii++) {
            mi = VFCHI4(ii,ifloat);
            mx = bspline4(mi);
            dmx = dbspline4(mi);
            d2mx = d2bspline4(mi);
            d3mx = d3bspline4(mi);
            for (jj=jm2; jj<=jp2; jj++) {
                mj = VFCHI4(jj,jfloat);
                my = bspline4(mj);
                dmy = dbspline4(mj);
                d2my = d2bspline4(mj);
                d3my = d3bspline4(mj);
                for (kk=km2; kk<=kp2; kk++) {
                    mk = VFCHI4(kk,kfloat);
                    mz = bspline4(mk);
                    dmz = dbspline4(mk);
                    d2mz = d2bspline4(mk);
                    d3mz = d3bspline4(mk);
                    f = u[IJK(ii,jj,kk)];
                    fp = up[IJK(ii,jj,kk)];
                    /* The potential */
                    pot  += f*mx*my*mz;
                    /* The field */
                    e[0] += f*dmx*my*mz/hx;
                    e[1] += f*mx*dmy*mz/hy;
                    e[2] += f*mx*my*dmz/hzed;
                    /* The gradient of the field */
                    de[0][0] += f*d2mx*my*mz/(hx*hx);
                    de[1][0] += f*dmx*dmy*mz/(hy*hx);
                    de[1][1] += f*mx*d2my*mz/(hy*hy);
                    de[2][0] += f*dmx*my*dmz/(hx*hzed);
                    de[2][1] += f*mx*dmy*dmz/(hy*hzed);
                    de[2][2] += f*mx*my*d2mz/(hzed*hzed);
                    /* The gradient of the (permanent) field */
                    dep[0][0] += fp*d2mx*my*mz/(hx*hx);
                    dep[1][0] += fp*dmx*dmy*mz/(hy*hx);
                    dep[1][1] += fp*mx*d2my*mz/(hy*hy);
                    dep[2][0] += fp*dmx*my*dmz/(hx*hzed);
                    dep[2][1] += fp*mx*dmy*dmz/(hy*hzed);
                    dep[2][2] += fp*mx*my*d2mz/(hzed*hzed);
                    /* The 2nd gradient of the field */
                    /* VxVxVa */
                    d2e[0][0][0] += f*d3mx*my*mz /(hx*hx*hx);
                    d2e[0][0][1] += f*d2mx*dmy*mz/(hx*hy*hx);
                    d2e[0][0][2] += f*d2mx*my*dmz/(hx*hx*hzed);
                    /* VyVxVa */
```

```
                    d2e[1][0][0] += f*d2mx*dmy*mz/(hx*hx*hy);
                    d2e[1][0][1] += f*dmx*d2my*mz/(hx*hy*hy);
                    d2e[1][0][2] += f*dmx*dmy*dmz/(hx*hy*hzed);
                    /* VyVyVa */
                    d2e[1][1][0] += f*dmx*d2my*mz/(hx*hy*hy);
                    d2e[1][1][1] += f*mx*d3my*mz /(hy*hy*hy);
                    d2e[1][1][2] += f*mx*d2my*dmz/(hy*hy*hzed);
                    /* VzVxVa */
                    d2e[2][0][0] += f*d2mx*my*dmz/(hx*hx*hzed);
                    d2e[2][0][1] += f*dmx*dmy*dmz/(hx*hy*hzed);
                    d2e[2][0][2] += f*dmx*my*d2mz/(hx*hzed*hzed);
                    /* VzVyVa */
                    d2e[2][1][0] += f*dmx*dmy*dmz/(hx*hy*hzed);
                    d2e[2][1][1] += f*mx*d2my*dmz/(hy*hy*hzed);
                    d2e[2][1][2] += f*mx*dmy*d2mz/(hy*hzed*hzed);
                    /* VzVzVa */
                    d2e[2][2][0] += f*dmx*my*d2mz/(hx*hzed*hzed);
                    d2e[2][2][1] += f*mx*dmy*d2mz/(hy*hzed*hzed);
                    d2e[2][2][2] += f*mx*my*d3mz /(hzed*hzed*hzed);
                }
            }
        }
    }

    /* force on permanent multipole due to non-local induced reaction field */

    /* Monopole Force */
    force[0] = e[0]*c;
    force[1] = e[1]*c;
    force[2] = e[2]*c;

    /* Dipole Force */
    force[0] -= de[0][0]*ux+de[1][0]*uy+de[2][0]*uz;
    force[1] -= de[1][0]*ux+de[1][1]*uy+de[2][1]*uz;
    force[2] -= de[2][0]*ux+de[2][1]*uy+de[2][2]*uz;

    /* Quadrupole Force */
    force[0] += d2e[0][0][0]*qxx
             +  d2e[1][0][0]*qyx*2.0+d2e[1][1][0]*qyy
             +  d2e[2][0][0]*qzx*2.0+d2e[2][1][0]*qzy*2.0+d2e[2][2][0]*qzz;
    force[1] += d2e[0][0][1]*qxx
             +  d2e[1][0][1]*qyx*2.0+d2e[1][1][1]*qyy
             +  d2e[2][0][1]*qzx*2.0+d2e[2][1][1]*qzy*2.0+d2e[2][2][1]*qzz;
    force[2] += d2e[0][0][2]*qxx
             +  d2e[1][0][2]*qyx*2.0+d2e[1][1][2]*qyy
             +  d2e[2][0][2]*qzx*2.0+d2e[2][1][2]*qzy*2.0+d2e[2][2][2]*qzz;

    /* torque on permanent mulitpole due to non-local induced reaction field */

    /* Dipole Torque */
    torque[0] = uy * e[2] - uz * e[1];
    torque[1] = uz * e[0] - ux * e[2];
    torque[2] = ux * e[1] - uy * e[0];

    /* Quadrupole Torque */
    /* Tx = -2.0*(Sum_a (Qya*dEaz) + Sum_b (Qzb*dEby))
       Ty = -2.0*(Sum_a (Qza*dEax) + Sum_b (Qxb*dEbz))
       Tz = -2.0*(Sum_a (Qxa*dEay) + Sum_b (Qyb*dEbx))  */
    de[0][1] = de[1][0];
    de[0][2] = de[2][0];
    de[1][2] = de[2][1];
    torque[0] -= 2.0*(qyx*de[0][2] + qyy*de[1][2] + qyz*de[2][2]
                    - qzx*de[0][1] - qzy*de[1][1] - qzz*de[2][1]);
    torque[1] -= 2.0*(qzx*de[0][0] + qzy*de[1][0] + qzz*de[2][0]
                    - qxx*de[0][2] - qxy*de[1][2] - qxz*de[2][2]);
    torque[2] -= 2.0*(qxx*de[0][1] + qxy*de[1][1] + qxz*de[2][1]
                    - qyx*de[0][0] - qyy*de[1][0] - qyz*de[2][0]);

    /* force on non-local induced dipole due to permanent reaction field */

    force[0] -= dep[0][0]*uix+dep[1][0]*uiy+dep[2][0]*uiz;
    force[1] -= dep[1][0]*uix+dep[1][1]*uiy+dep[2][1]*uiz;
    force[2] -= dep[2][0]*uix+dep[2][1]*uiy+dep[2][2]*uiz;

    force[0] = 0.5 * force[0];
    force[1] = 0.5 * force[1];
    force[2] = 0.5 * force[2];
    torque[0] = 0.5 * torque[0];
    torque[1] = 0.5 * torque[1];
    torque[2] = 0.5 * torque[2];

```

```
    /* printf(" qPhi Force %f %f %f\n", force[0], force[1], force[2]);
       printf(" qPhi Torque %f %f %f\n", torque[0], torque[1], torque[2]); */
}

VPUBLIC void Vpmg_ibDirectPolForce(Vpmg *thee, Vgrid *perm, Vgrid *induced,
                                   int atomID, double force[3]) {

    Vatom *atom;
    Valist *alist;
    Vacc *acc;
    Vpbe *pbe;
    Vsurf_Meth srfm;

    double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
    double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;
    double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
    double izmagic;
    int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;

    VASSERT(thee != VNULL);
    VASSERT(perm != VNULL);        /* potential due to permanent multipoles.*/
    VASSERT(induced != VNULL);     /* potential due to induced dipoles. */
    VASSERT (!thee->pmgp->nonlin); /* Nonlinear PBE is not implemented for AMOEBA */

    acc = thee->pbe->acc;
    srfm = thee->surfMeth;
    atom = Valist_getAtom(thee->pbe->alist, atomID);
    VASSERT(atom->partID != 0);   /* Currently all atoms must be in the same partition. */
    apos = Vatom_getPosition(atom);
    arad = Vatom_getRadius(atom);

    /* Reset force */
    force[0] = 0.0;
    force[1] = 0.0;
    force[2] = 0.0;

    /* Get PBE info */
    pbe = thee->pbe;
    acc = pbe->acc;
    alist = pbe->alist;
    irad = Vpbe_getMaxIonRadius(pbe);
    zkappa2 = Vpbe_getZkappa2(pbe);
    izmagic = 1.0/Vpbe_getZmagic(pbe);

    VASSERT (zkappa2 > VPMGSMALL); /* It is ok to run AMOEBA with no ions, but this is checked for higher up in the driver. */

    /* Mesh info */
    nx = induced->nx;
    ny = induced->ny;
    nz = induced->nz;
    hx = induced->hx;
    hy = induced->hy;
    hzed = induced->hzed;
    xmin = induced->xmin;
    ymin = induced->ymin;
    zmin = induced->zmin;
    xmax = induced->xmax;
    ymax = induced->ymax;
    zmax = induced->zmax;
    xlen = xmax-xmin;
    ylen = ymax-ymin;
    zlen = zmax-zmin;

    /* Make sure we're on the grid */
    if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
      (apos[1]<=ymin) || (apos[1]>=ymax)  || \
      (apos[2]<=zmin) || (apos[2]>=zmax)) {
        Vnm_print(2, "Vpmg_ibForce:  Atom at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):\n",
            apos[0], apos[1], apos[2]);
        Vnm_print(2, "Vpmg_ibForce:    xmin = %g, xmax = %g\n", xmin, xmax);
        Vnm_print(2, "Vpmg_ibForce:    ymin = %g, ymax = %g\n", ymin, ymax);
        Vnm_print(2, "Vpmg_ibForce:    zmin = %g, zmax = %g\n", zmin, zmax);
        fflush(stderr);
    } else {

```

```
        /* Convert the atom position to grid reference frame */
        position[0] = apos[0] - xmin;
        position[1] = apos[1] - ymin;
        position[2] = apos[2] - zmin;

        /* Integrate over points within this atom's (inflated) radius */
        rtot = (irad + arad + thee->splineWin);
        rtot2 = VSQR(rtot);
        dx = rtot + 0.5*hx;
        imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
        imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
        for (i=imin; i<=imax; i++) {
            dx2 = VSQR(position[0] - hx*i);
            if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
            else dy = 0.5*hy;
            jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
            jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
            for (j=jmin; j<=jmax; j++) {
                dy2 = VSQR(position[1] - hy*j);
                if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
                else dz = 0.5*hzed;
                kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
                kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
                for (k=kmin; k<=kmax; k++) {
                    dz2 = VSQR(k*hzed - position[2]);
                    /* See if grid point is inside ivdw radius and set ccf
                     * accordingly (do spline assignment here) */
                    if ((dz2 + dy2 + dx2) <= rtot2) {
                        gpos[0] = i*hx + xmin;
                        gpos[1] = j*hy + ymin;
                        gpos[2] = k*hzed + zmin;
                        Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, irad,
                          atom, tgrad);
                        fmag = induced->data[IJK(i,j,k)];
                        fmag *= perm->data[IJK(i,j,k)];
                        fmag *= thee->kappa[IJK(i,j,k)];
                        force[0] += (zkappa2*fmag*tgrad[0]);
                        force[1] += (zkappa2*fmag*tgrad[1]);
                        force[2] += (zkappa2*fmag*tgrad[2]);
                    }
                } /* k loop */
            } /* j loop */
        } /* i loop */
    }

    force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
    force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
    force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;

}

VPUBLIC void Vpmg_ibNLDirectPolForce(Vpmg *thee, Vgrid *perm, Vgrid *nlInduced,
                                     int atomID, double force[3]) {
     Vpmg_ibDirectPolForce(thee, perm, nlInduced, atomID, force);
}

VPUBLIC void Vpmg_dbDirectPolForce(Vpmg *thee, Vgrid *perm, Vgrid *induced,
                                   int atomID, double force[3]) {

    Vatom *atom;
    Vacc *acc;
    Vpbe *pbe;
    Vsurf_Meth srfm;

    double *apos, position[3], arad, hx, hy, hzed, izmagic, deps, depsi;
    double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2, epsp;
    double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
    double *u, *up, Hxijk, Hyijk, Hzijk, Hxim1jk, Hyijm1k, Hzijkm1;
    double dHxijk[3], dHyijk[3], dHzijk[3], dHxim1jk[3], dHyijm1k[3];
    double dHzijkm1[3];
    int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;

    VASSERT(thee != VNULL);
    VASSERT(perm != VNULL);    /* permanent multipole PMG solution. */
    VASSERT(induced != VNULL); /* potential due to induced dipoles. */
```

```

    acc = thee->pbe->acc;
    atom = Valist_getAtom(thee->pbe->alist, atomID);
    VASSERT (atom->partID != 0);   /* Currently all atoms must be in the same partition. */
    apos = Vatom_getPosition(atom);
    arad = Vatom_getRadius(atom);

    /* Reset force */
    force[0] = 0.0;
    force[1] = 0.0;
    force[2] = 0.0;

    /* Get PBE info */
    pbe = thee->pbe;
    acc = pbe->acc;
    srfm = thee->surfMeth;
    epsp = Vpbe_getSoluteDiel(pbe);
    epsw = Vpbe_getSolventDiel(pbe);
    kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*Vunit_kb;
    izmagic = 1.0/Vpbe_getZmagic(pbe);

    deps = (epsw - epsp);
    depsi = 1.0/deps;
    VASSERT(VABS(deps) > VPMGSMALL);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;
    xmin = thee->pmgp->xmin;
    ymin = thee->pmgp->ymin;
    zmin = thee->pmgp->zmin;
    xmax = thee->pmgp->xmax;
    ymax = thee->pmgp->ymax;
    zmax = thee->pmgp->zmax;
    /* If the permanent and induced potentials are flipped the
       results are exactly the same. */
    u = induced->data;
    up = perm->data;

    /* Make sure we're on the grid */
    if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
      (apos[1]<=ymin) || (apos[1]>=ymax)  || \
      (apos[2]<=zmin) || (apos[2]>=zmax)) {
         Vnm_print(2, "Vpmg_dbDirectPolForce:  Atom at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):\n", apos[0], apos[1], apos[2]);
         Vnm_print(2, "Vpmg_dbDirectPolForce:    xmin = %g, xmax = %g\n", xmin, xmax);
         Vnm_print(2, "Vpmg_dbDirectPolForce:    ymin = %g, ymax = %g\n", ymin, ymax);
         Vnm_print(2, "Vpmg_dbDirectPolForce:    zmin = %g, zmax = %g\n", zmin, zmax);
         fflush(stderr);
    } else {

        /* Convert the atom position to grid reference frame */
        position[0] = apos[0] - xmin;
        position[1] = apos[1] - ymin;
        position[2] = apos[2] - zmin;

        /* Integrate over points within this atom's (inflated) radius */
        rtot = (arad + thee->splineWin);
        rtot2 = VSQR(rtot);
        dx = rtot/hx;
        imin = (int)floor((position[0]-rtot)/hx);
        if (imin < 1) {
            Vnm_print(2, "Vpmg_dbDirectPolForce:  Atom %d off grid!\n", atomID);
            return;
        }
        imax = (int)ceil((position[0]+rtot)/hx);
        if (imax > (nx-2)) {
            Vnm_print(2, "Vpmg_dbDirectPolForce:  Atom %d off grid!\n", atomID)
```

```
;
            return;
        }
        jmin = (int)floor((position[1]-rtot)/hy);
        if (jmin < 1) {
            Vnm_print(2, "Vpmg_dbDirectPolForce:  Atom %d off grid!\n", atomID);
            return;
        }
        jmax = (int)ceil((position[1]+rtot)/hy);
        if (jmax > (ny-2)) {
            Vnm_print(2, "Vpmg_dbDirectPolForce:  Atom %d off grid!\n", atomID);
            return;
        }
        kmin = (int)floor((position[2]-rtot)/hzed);
        if (kmin < 1) {
            Vnm_print(2, "Vpmg_dbDirectPolForce:  Atom %d off grid!\n", atomID);
            return;
        }
        kmax = (int)ceil((position[2]+rtot)/hzed);
        if (kmax > (nz-2)) {
            Vnm_print(2, "Vpmg_dbDirectPolForce:  Atom %d off grid!\n", atomID);
            return;
        }
        for (i=imin; i<=imax; i++) {
            for (j=jmin; j<=jmax; j++) {
                for (k=kmin; k<=kmax; k++) {
                    /* i,j,k */
                    gpos[0] = (i+0.5)*hx + xmin;
                    gpos[1] = j*hy + ymin;
                    gpos[2] = k*hzed + zmin;
                    Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
                    Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
                            atom, dHxijk);
                    for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
                    gpos[0] = i*hx + xmin;
                    gpos[1] = (j+0.5)*hy + ymin;
                    gpos[2] = k*hzed + zmin;
                    Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
                    Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
                            atom, dHyijk);
                    for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
                    gpos[0] = i*hx + xmin;
                    gpos[1] = j*hy + ymin;
                    gpos[2] = (k+0.5)*hzed + zmin;
                    Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
                    Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
                            atom, dHzijk);
                    for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
                    /* i-1,j,k */
                    gpos[0] = (i-0.5)*hx + xmin;
                    gpos[1] = j*hy + ymin;
                    gpos[2] = k*hzed + zmin;
                    Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
                    Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
                            atom, dHxim1jk);
                    for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
                    /* i,j-1,k */
                    gpos[0] = i*hx + xmin;
                    gpos[1] = (j-0.5)*hy + ymin;
                    gpos[2] = k*hzed + zmin;
                    Hyijm1k = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
                    Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
                            atom, dHyijm1k);
                    for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;
                    /* i,j,k-1 */
                    gpos[0] = i*hx + xmin;
                    gpos[1] = j*hy + ymin;
                    gpos[2] = (k-0.5)*hzed + zmin;
                    Hzijkm1 = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
                    Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
                            atom, dHzijkm1);
```

```
                    for (l=0; l<3; l++) dHzijkm1[l] *= Hzijkm1;

                    dbFmag = up[IJK(i,j,k)];
                    tgrad[0] =
                       (dHxijk[0]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
                     +  dHxim1jk[0]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
                     + (dHyijk[0]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
                     +  dHyijm1k[0]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
                     + (dHzijk[0]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
                     + dHzijkm1[0]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
                    tgrad[1] =
                       (dHxijk[1]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
                     +  dHxim1jk[1]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
                     + (dHyijk[1]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
                     +  dHyijm1k[1]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
                     + (dHzijk[1]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
                     + dHzijkm1[1]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
                    tgrad[2] =
                       (dHxijk[2]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
                     +  dHxim1jk[2]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
                     + (dHyijk[2]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
                     +  dHyijm1k[2]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
                     + (dHzijk[2]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
                     + dHzijkm1[2]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
                     force[0] += (dbFmag*tgrad[0]);
                     force[1] += (dbFmag*tgrad[1]);
                     force[2] += (dbFmag*tgrad[2]);

                } /* k loop */
            } /* j loop */
        } /* i loop */

        force[0] = -force[0]*hx*hy*hzed*deps*0.5*izmagic;
        force[1] = -force[1]*hx*hy*hzed*deps*0.5*izmagic;
        force[2] = -force[2]*hx*hy*hzed*deps*0.5*izmagic;

    }
}

VPUBLIC void Vpmg_dbNLDirectPolForce(Vpmg *thee, Vgrid *perm, Vgrid *nlInduced,
                                     int atomID, double force[3]) {
     Vpmg_dbDirectPolForce(thee, perm, nlInduced, atomID, force);
}

VPUBLIC void Vpmg_qfMutualPolForce(Vpmg *thee, Vgrid *induced,
                                   Vgrid *nlinduced, int atomID, double force[3]) {

    Vatom *atom;
    double *apos, *dipole, position[3], hx, hy, hzed;
    double *u, *unl;
    double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
    double de[3][3], denl[3][3];
    double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz, mi, mj, mk;
    double ifloat, jfloat, kfloat;
    double f, fnl, uix, uiy, uiz, uixnl, uiynl, uiznl;
    int i,j,k,nx, ny, nz, im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1;
    int kp1, kp2, ii, jj, kk;

    VASSERT(thee != VNULL);   /* PMG object with PBE info. */
    VASSERT(induced != VNULL); /* potential due to induced dipoles. */
    VASSERT(nlinduced != VNULL); /* potential due to non-local induced dipoles. */
    atom = Valist_getAtom(thee->pbe->alist, atomID);
    VASSERT(atom->partID != 0);    /* all atoms must be in the same partition. */
    apos = Vatom_getPosition(atom);
    dipole = Vatom_getInducedDipole(atom);
    uix = dipole[0];
    uiy = dipole[1];
    uiz = dipole[2];
    dipole = Vatom_getNLInducedDipole(atom);
    uixnl = dipole[0];
    uiynl = dipole[1];
    uiznl = dipole[2];
    u = induced->data;
```

```
    unl = nlinduced->data;

    for (i=0;i<3;i++){
       for (j=0;j<3;j++){
          de[i][j] = 0.0;
          denl[i][j] = 0.0;
       }
    }

    /* Mesh info */
    nx = induced->nx;
    ny = induced->ny;
    nz = induced->nz;
    hx = induced->hx;
    hy = induced->hy;
    hzed = induced->hzed;
    xmin = induced->xmin;
    ymin = induced->ymin;
    zmin = induced->zmin;
    xmax = induced->xmax;
    ymax = induced->ymax;
    zmax = induced->zmax;
    xlen = xmax-xmin;
    ylen = ymax-ymin;
    zlen = zmax-zmin;

    /* If we aren't in the current position, then we're done */
    if (atom->partID == 0) return;

    /* Make sure we're on the grid */
    if ((apos[0]<=(xmin+2*hx))   || (apos[0]>=(xmax-2*hx)) \
     || (apos[1]<=(ymin+2*hy))   || (apos[1]>=(ymax-2*hy)) \
     || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
        Vnm_print(2, "qfMutualPolForce:  Atom off the mesh (ignoring) %6.3f
%6.3f %6.3f\n", apos[0], apos[1], apos[2]);
        fflush(stderr);
    } else {

        /* Convert the atom position to grid coordinates */
        position[0] = apos[0] - xmin;
        position[1] = apos[1] - ymin;
        position[2] = apos[2] - zmin;
        ifloat = position[0]/hx;
        jfloat = position[1]/hy;
        kfloat = position[2]/hzed;
        ip1 = (int)ceil(ifloat);
        ip2 = ip1 + 2;
        im1 = (int)floor(ifloat);
        im2 = im1 - 2;
        jp1 = (int)ceil(jfloat);
        jp2 = jp1 + 2;
        jm1 = (int)floor(jfloat);
        jm2 = jm1 - 2;
        kp1 = (int)ceil(kfloat);
        kp2 = kp1 + 2;
        km1 = (int)floor(kfloat);
        km2 = km1 - 2;

        /* This step shouldn't be necessary, but it saves nasty debugging
         * later on if something goes wrong */
        ip2 = VMIN2(ip2,nx-1);
        ip1 = VMIN2(ip1,nx-1);
        im1 = VMAX2(im1,0);
        im2 = VMAX2(im2,0);
        jp2 = VMIN2(jp2,ny-1);
        jp1 = VMIN2(jp1,ny-1);
        jm1 = VMAX2(jm1,0);
        jm2 = VMAX2(jm2,0);
        kp2 = VMIN2(kp2,nz-1);
        kp1 = VMIN2(kp1,nz-1);
        km1 = VMAX2(km1,0);
        km2 = VMAX2(km2,0);

        for (ii=im2; ii<=ip2; ii++) {
            mi = VFCHI4(ii,ifloat);
            mx = bspline4(mi);
            dmx = dbspline4(mi);
            d2mx = d2bspline4(mi);
            for (jj=jm2; jj<=jp2; jj++) {
                mj = VFCHI4(jj,jfloat);
                my = bspline4(mj);
```

```
                dmy = dbspline4(mj);
                d2my = d2bspline4(mj);
                for (kk=km2; kk<=kp2; kk++) {
                    mk = VFCHI4(kk,kfloat);
                    mz = bspline4(mk);
                    dmz = dbspline4(mk);
                    d2mz = d2bspline4(mk);
                    f = u[IJK(ii,jj,kk)];
                    fnl = unl[IJK(ii,jj,kk)];

                    /* The gradient of the reaction field
                       due to induced dipoles */
                    de[0][0] += f*d2mx*my*mz/(hx*hx);
                    de[1][0] += f*dmx*dmy*mz/(hy*hx);
                    de[1][1] += f*mx*d2my*mz/(hy*hy);
                    de[2][0] += f*dmx*my*dmz/(hx*hzed);
                    de[2][1] += f*mx*dmy*dmz/(hy*hzed);
                    de[2][2] += f*mx*my*d2mz/(hzed*hzed);

                    /* The gradient of the reaction field
                       due to non-local induced dipoles */
                    denl[0][0] += fnl*d2mx*my*mz/(hx*hx);
                    denl[1][0] += fnl*dmx*dmy*mz/(hy*hx);
                    denl[1][1] += fnl*mx*d2my*mz/(hy*hy);
                    denl[2][0] += fnl*dmx*my*dmz/(hx*hzed);
                    denl[2][1] += fnl*mx*dmy*dmz/(hy*hzed);
                    denl[2][2] += fnl*mx*my*d2mz/(hzed*hzed);
                }
            }
        }
    }

    /* mutual polarization force */
    force[0] = -(de[0][0]*uixnl + de[1][0]*uiynl + de[2][0]*uiznl);
    force[1] = -(de[1][0]*uixnl + de[1][1]*uiynl + de[2][1]*uiznl);
    force[2] = -(de[2][0]*uixnl + de[2][1]*uiynl + de[2][2]*uiznl);
    force[0] -=  denl[0][0]*uix + denl[1][0]*uiy + denl[2][0]*uiz;
    force[1] -=  denl[1][0]*uix + denl[1][1]*uiy + denl[2][1]*uiz;
    force[2] -=  denl[2][0]*uix + denl[2][1]*uiy + denl[2][2]*uiz;

    force[0] = 0.5 * force[0];
    force[1] = 0.5 * force[1];
    force[2] = 0.5 * force[2];

}

VPUBLIC void Vpmg_ibMutualPolForce(Vpmg *thee, Vgrid *induced, Vgrid *nlinduced,
                                   int atomID, double force[3]) {

    Vatom *atom;
    Valist *alist;
    Vacc *acc;
    Vpbe *pbe;
    Vsurf_Meth srfm;

    double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
    double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;
    double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
    double izmagic;
    int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;

    VASSERT(thee != VNULL);        /* We need a PMG object with PBE info. */
    VASSERT(induced != VNULL);     /* We need the potential due to induced dipoles. */
    VASSERT(nlinduced != VNULL);   /* We need the potential due to non-local induced dipoles. */
    VASSERT (!thee->pmgp->nonlin); /* Nonlinear PBE is not implemented for AMOEBA */

    atom = Valist_getAtom(thee->pbe->alist, atomID);
VASSERT (atom->partID != 0);   /* Currently all atoms must be in the same partition. */

    acc = thee->pbe->acc;
    srfm = thee->surfMeth;
    apos = Vatom_getPosition(atom);
    arad = Vatom_getRadius(atom);

    /* Reset force */
```

```
    force[0] = 0.0;
    force[1] = 0.0;
    force[2] = 0.0;

    /* If we aren't in the current position, then we're done */
    if (atom->partID == 0) return;

    /* Get PBE info */
    pbe = thee->pbe;
    acc = pbe->acc;
    alist = pbe->alist;
    irad = Vpbe_getMaxIonRadius(pbe);
    zkappa2 = Vpbe_getZkappa2(pbe);
    izmagic = 1.0/Vpbe_getZmagic(pbe);

    VASSERT (zkappa2 > VPMGSMALL); /* Should be a check for this further up.*/

    /* Mesh info */
    nx = induced->nx;
    ny = induced->ny;
    nz = induced->nz;
    hx = induced->hx;
    hy = induced->hy;
    hzed = induced->hzed;
    xmin = induced->xmin;
    ymin = induced->ymin;
    zmin = induced->zmin;
    xmax = induced->xmax;
    ymax = induced->ymax;
    zmax = induced->zmax;
    xlen = xmax-xmin;
    ylen = ymax-ymin;
    zlen = zmax-zmin;

    /* Make sure we're on the grid */
    if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
      (apos[1]<=ymin) || (apos[1]>=ymax)  || \
      (apos[2]<=zmin) || (apos[2]>=zmax)) {
        Vnm_print(2, "Vpmg_ibMutalPolForce:  Atom at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):\n", apos[0], apos[1], apos[2]);
        Vnm_print(2, "Vpmg_ibMutalPolForce:    xmin = %g, xmax = %g\n", xmin, xmax);
        Vnm_print(2, "Vpmg_ibMutalPolForce:    ymin = %g, ymax = %g\n", ymin, ymax);
        Vnm_print(2, "Vpmg_ibMutalPolForce:    zmin = %g, zmax = %g\n", zmin, zmax);
        fflush(stderr);
    } else {

        /* Convert the atom position to grid reference frame */
        position[0] = apos[0] - xmin;
        position[1] = apos[1] - ymin;
        position[2] = apos[2] - zmin;

        /* Integrate over points within this atom's (inflated) radius */
        rtot = (irad + arad + thee->splineWin);
        rtot2 = VSQR(rtot);
        dx = rtot + 0.5*hx;
        imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
        imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
        for (i=imin; i<=imax; i++) {
            dx2 = VSQR(position[0] - hx*i);
            if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
            else dy = 0.5*hy;
            jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
            jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
            for (j=jmin; j<=jmax; j++) {
                dy2 = VSQR(position[1] - hy*j);
                if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
                else dz = 0.5*hzed;
                kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
                kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
                for (k=kmin; k<=kmax; k++) {
                    dz2 = VSQR(k*hzed - position[2]);
                    /* See if grid point is inside ivdw radius and set ccf
                     * accordingly (do spline assignment here) */
                    if ((dz2 + dy2 + dx2) <= rtot2) {
                        gpos[0] = i*hx + xmin;
                        gpos[1] = j*hy + ymin;
                        gpos[2] = k*hzed + zmin;
```

```
                        Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, irad,
                          atom, tgrad);
                        fmag = induced->data[IJK(i,j,k)];
                        fmag *= nlinduced->data[IJK(i,j,k)];
                        fmag *= thee->kappa[IJK(i,j,k)];
                        force[0] += (zkappa2*fmag*tgrad[0]);
                        force[1] += (zkappa2*fmag*tgrad[1]);
                        force[2] += (zkappa2*fmag*tgrad[2]);
                    }
                } /* k loop */
            } /* j loop */
        } /* i loop */
    }

    force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
    force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
    force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
}

VPUBLIC void Vpmg_dbMutualPolForce(Vpmg *thee, Vgrid *induced,
                                   Vgrid *nlinduced, int atomID,
                                   double force[3]) {

    Vatom *atom;
    Vacc *acc;
    Vpbe *pbe;
    Vsurf_Meth srfm;

    double *apos, position[3], arad, hx, hy, hzed, izmagic, deps, depsi;
    double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2, epsp;
    double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
    double *u, *unl, Hxijk, Hyijk, Hzijk, Hxim1jk, Hyijm1k, Hzijkm1;
    double dHxijk[3], dHyijk[3], dHzijk[3], dHxim1jk[3], dHyijm1k[3];
    double dHzijkm1[3];
    int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;

    VASSERT(thee != VNULL); /* PMG object with PBE info. */
    VASSERT(induced != VNULL); /* potential due to induced dipoles.*/
    VASSERT(nlinduced != VNULL); /* potential due to non-local induced dipoles.*/

    acc = thee->pbe->acc;
    srfm = thee->surfMeth;
    atom = Valist_getAtom(thee->pbe->alist, atomID);
    VASSERT (atom->partID != 0); /* all atoms must be in the same partition.*/
    apos = Vatom_getPosition(atom);
    arad = Vatom_getRadius(atom);

    /* Reset force */
    force[0] = 0.0;
    force[1] = 0.0;
    force[2] = 0.0;

    /* Get PBE info */
    pbe = thee->pbe;
    acc = pbe->acc;
    epsp = Vpbe_getSoluteDiel(pbe);
    epsw = Vpbe_getSolventDiel(pbe);
    kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*Vunit_kb;
    izmagic = 1.0/Vpbe_getZmagic(pbe);

    deps = (epsw - epsp);
    depsi = 1.0/deps;
    VASSERT(VABS(deps) > VPMGSMALL);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;
```

```
    xmin = thee->pmgp->xmin;
    ymin = thee->pmgp->ymin;
    zmin = thee->pmgp->zmin;
    xmax = thee->pmgp->xmax;
    ymax = thee->pmgp->ymax;
    zmax = thee->pmgp->zmax;
    u = induced->data;
    unl = nlinduced->data;

    /* Make sure we're on the grid */
    if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
      (apos[1]<=ymin) || (apos[1]>=ymax)  || \
      (apos[2]<=zmin) || (apos[2]>=zmax)) {
        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]);
        Vnm_print(2, "Vpmg_dbMutualPolForce:    xmin = %g, xmax = %g\n", xmin, xmax);
        Vnm_print(2, "Vpmg_dbMutualPolForce:    ymin = %g, ymax = %g\n", ymin, ymax);
        Vnm_print(2, "Vpmg_dbMutualPolForce:    zmin = %g, zmax = %g\n", zmin, zmax);
        fflush(stderr);
    } else {

        /* Convert the atom position to grid reference frame */
        position[0] = apos[0] - xmin;
        position[1] = apos[1] - ymin;
        position[2] = apos[2] - zmin;

        /* Integrate over points within this atom's (inflated) radius */
        rtot = (arad + thee->splineWin);
        rtot2 = VSQR(rtot);
        dx = rtot/hx;
        imin = (int)floor((position[0]-rtot)/hx);
        if (imin < 1) {
            Vnm_print(2, "Vpmg_dbMutualPolForce:  Atom %d off grid!\n", atomID);
            return;
        }
        imax = (int)ceil((position[0]+rtot)/hx);
        if (imax > (nx-2)) {
            Vnm_print(2, "Vpmg_dbMutualPolForce:  Atom %d off grid!\n", atomID);
            return;
        }
        jmin = (int)floor((position[1]-rtot)/hy);
        if (jmin < 1) {
            Vnm_print(2, "Vpmg_dbMutualPolForce:  Atom %d off grid!\n", atomID);
            return;
        }
        jmax = (int)ceil((position[1]+rtot)/hy);
        if (jmax > (ny-2)) {
            Vnm_print(2, "Vpmg_dbMutualPolForce:  Atom %d off grid!\n", atomID);
            return;
        }
        kmin = (int)floor((position[2]-rtot)/hzed);
        if (kmin < 1) {
            Vnm_print(2, "Vpmg_dbMutualPolForce:  Atom %d off grid!\n", atomID);
            return;
        }
        kmax = (int)ceil((position[2]+rtot)/hzed);
        if (kmax > (nz-2)) {
            Vnm_print(2, "Vpmg_dbMutualPolForce:  Atom %d off grid!\n", atomID);
            return;
        }
        for (i=imin; i<=imax; i++) {
            for (j=jmin; j<=jmax; j++) {
                for (k=kmin; k<=kmax; k++) {
                    /* i,j,k */
                    gpos[0] = (i+0.5)*hx + xmin;
                    gpos[1] = j*hy + ymin;
                    gpos[2] = k*hzed + zmin;
                    Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
                    Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
                            atom, dHxijk);
                    for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
```

```
                    gpos[0] = i*hx + xmin;
                    gpos[1] = (j+0.5)*hy + ymin;
                    gpos[2] = k*hzed + zmin;
                    Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
                    Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
                            atom, dHyijk);
                    for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
                    gpos[0] = i*hx + xmin;
                    gpos[1] = j*hy + ymin;
                    gpos[2] = (k+0.5)*hzed + zmin;
                    Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
                    Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
                            atom, dHzijk);
                    for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
                    /* i-1,j,k */
                    gpos[0] = (i-0.5)*hx + xmin;
                    gpos[1] = j*hy + ymin;
                    gpos[2] = k*hzed + zmin;
                    Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
                    Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
                            atom, dHxim1jk);
                    for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
                    /* i,j-1,k */
                    gpos[0] = i*hx + xmin;
                    gpos[1] = (j-0.5)*hy + ymin;
                    gpos[2] = k*hzed + zmin;
                    Hyijm1k = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
                    Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
                            atom, dHyijm1k);
                    for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;
                    /* i,j,k-1 */
                    gpos[0] = i*hx + xmin;
                    gpos[1] = j*hy + ymin;
                    gpos[2] = (k-0.5)*hzed + zmin;
                    Hzijkm1 = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
                    Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
                            atom, dHzijkm1);
                    for (l=0; l<3; l++) dHzijkm1[l] *= Hzijkm1;
                    dbFmag = unl[IJK(i,j,k)];
                    tgrad[0] =
                       (dHxijk[0]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
                     +  dHxim1jk[0]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
                     + (dHyijk[0]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
                     +  dHyijm1k[0]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
                     + (dHzijk[0]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
                     + dHzijkm1[0]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
                    tgrad[1] =
                       (dHxijk[1]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
                     +  dHxim1jk[1]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
                     + (dHyijk[1]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
                     +  dHyijm1k[1]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
                     + (dHzijk[1]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
                     + dHzijkm1[1]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
                    tgrad[2] =
                       (dHxijk[2]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
                     +  dHxim1jk[2]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
                     + (dHyijk[2]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
                     +  dHyijm1k[2]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
                     + (dHzijk[2]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
                     + dHzijkm1[2]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
                     force[0] += (dbFmag*tgrad[0]);
                     force[1] += (dbFmag*tgrad[1]);
                     force[2] += (dbFmag*tgrad[2]);
                } /* k loop */
            } /* j loop */
        } /* i loop */

        force[0] = -force[0]*hx*hy*hzed*deps*0.5*izmagic;
        force[1] = -force[1]*hx*hy*hzed*deps*0.5*izmagic;
        force[2] = -force[2]*hx*hy*hzed*deps*0.5*izmagic;
    }
}

#endif /* if defined(WITH_TINKER) */

VPRIVATE void fillcoCoefSpline4(Vpmg *thee) {
```

```
    Valist *alist;
    Vpbe *pbe;
    Vatom *atom;
    double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr, dist2;
    double xlen, ylen, zlen, position[3], itot, stot, ictot, ictot2, sctot;
    double irad, dx, dy, dz, epsw, epsp, w2i;
    double hx, hy, hzed, *apos, arad, sctot2;
    double dx2, dy2, dz2, stot2, itot2, rtot, rtot2, splineWin;
    double dist, value, denom, sm, sm2, sm3, sm4, sm5, sm6, sm7;
    double e, e2, e3, e4, e5, e6, e7;
    double b, b2, b3, b4, b5, b6, b7;
    double c0, c1, c2, c3, c4, c5, c6, c7;
    double ic0, ic1, ic2, ic3, ic4, ic5, ic6, ic7;
    int i, j, k, nx, ny, nz, iatom;
    int imin, imax, jmin, jmax, kmin, kmax;

    VASSERT(thee != VNULL);
    splineWin = thee->splineWin;

    /* Get PBE info */
    pbe = thee->pbe;
    alist = pbe->alist;
    irad = Vpbe_getMaxIonRadius(pbe);
    ionstr = Vpbe_getBulkIonicStrength(pbe);
    epsw = Vpbe_getSolventDiel(pbe);
    epsp = Vpbe_getSoluteDiel(pbe);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;

    /* Define the total domain size */
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;

    /* Define the min/max dimensions */
    xmin = thee->pmgp->xcent - (xlen/2.0);
    ymin = thee->pmgp->ycent - (ylen/2.0);
    zmin = thee->pmgp->zcent - (zlen/2.0);
    xmax = thee->pmgp->xcent + (xlen/2.0);
    ymax = thee->pmgp->ycent + (ylen/2.0);
    zmax = thee->pmgp->zcent + (zlen/2.0);

    /* This is a floating point parameter related to the non-zero nature of the
     * bulk ionic strength.  If the ionic strength is greater than zero; this
     * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
     * Otherwise, this parameter is set to 0.0 */
    if (ionstr > VPMGSMALL) ionmask = 1.0;
    else ionmask = 0.0;

    /* Reset the kappa, epsx, epsy, and epsz arrays */
    for (i=0; i<(nx*ny*nz); i++) {
        thee->kappa[i] = 1.0;
        thee->epsx[i] = 1.0;
        thee->epsy[i] = 1.0;
        thee->epsz[i] = 1.0;
    }

    /* Loop through the atoms and do assign the dielectric */
    for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {

        atom = Valist_getAtom(alist, iatom);
        apos = Vatom_getPosition(atom);
        arad = Vatom_getRadius(atom);

        b = arad - splineWin;
        e = arad + splineWin;
        e2 = e * e;
        e3 = e2 * e;
        e4 = e3 * e;
        e5 = e4 * e;
        e6 = e5 * e;
```

```
        e7 = e6 * e;
        b2 = b * b;
        b3 = b2 * b;
        b4 = b3 * b;
        b5 = b4 * b;
        b6 = b5 * b;
        b7 = b6 * b;
        denom = e7  - 7.0*b*e6 + 21.0*b2*e5 - 35.0*e4*b3
              + 35.0*e3*b4 - 21.0*b5*e2  + 7.0*e*b6 - b7;
        c0 = b4*(35.0*e3 - 21.0*b*e2 + 7*e*b2 - b3)/denom;
        c1 = -140.0*b3*e3/denom;
        c2 = 210.0*e2*b2*(e + b)/denom;
        c3 = -140.0*e*b*(e2 + 3.0*b*e + b2)/denom;
        c4 =  35.0*(e3 + 9.0*b*e2 + + 9.0*e*b2 + b3)/denom;
        c5 = -84.0*(e2 + 3.0*b*e + b2)/denom;
        c6 =  70.0*(e + b)/denom;
        c7 = -20.0/denom;

        b = irad + arad - splineWin;
        e = irad + arad + splineWin;
        e2 = e * e;
        e3 = e2 * e;
        e4 = e3 * e;
        e5 = e4 * e;
        e6 = e5 * e;
        e7 = e6 * e;
        b2 = b * b;
        b3 = b2 * b;
        b4 = b3 * b;
        b5 = b4 * b;
        b6 = b5 * b;
        b7 = b6 * b;
        denom = e7  - 7.0*b*e6 + 21.0*b2*e5 - 35.0*e4*b3
              + 35.0*e3*b4 - 21.0*b5*e2  + 7.0*e*b6 - b7;
        ic0 = b4*(35.0*e3 - 21.0*b*e2 + 7*e*b2 - b3)/denom;
        ic1 = -140.0*b3*e3/denom;
        ic2 = 210.0*e2*b2*(e + b)/denom;
        ic3 = -140.0*e*b*(e2 + 3.0*b*e + b2)/denom;
        ic4 =  35.0*(e3 + 9.0*b*e2 + + 9.0*e*b2 + b3)/denom;
        ic5 = -84.0*(e2 + 3.0*b*e + b2)/denom;
        ic6 =  70.0*(e + b)/denom;
        ic7 = -20.0/denom;

        /* Make sure we're on the grid */
        if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
            (apos[1]<=ymin) || (apos[1]>=ymax)  || \
            (apos[2]<=zmin) || (apos[2]>=zmax)) {
            if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
    (thee->pmgp->bcfl != BCFL_MAP)) {
                Vnm_print(2, "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f,\
%4.3f) is off the mesh (ignoring):\n",
                  iatom, apos[0], apos[1], apos[2]);
                Vnm_print(2, "Vpmg_fillco:    xmin = %g, xmax = %g\n",
                  xmin, xmax);
                Vnm_print(2, "Vpmg_fillco:    ymin = %g, ymax = %g\n",
                  ymin, ymax);
                Vnm_print(2, "Vpmg_fillco:    zmin = %g, zmax = %g\n",
                  zmin, zmax);
            }
            fflush(stderr);

        } else if (arad > VPMGSMALL ) { /* if we're on the mesh */

            /* Convert the atom position to grid reference frame */
            position[0] = apos[0] - xmin;
            position[1] = apos[1] - ymin;
            position[2] = apos[2] - zmin;

            /* MARK ION ACCESSIBILITY AND DIELECTRIC VALUES FOR LATER
             * ASSIGNMENT (Steps #1-3) */
            itot = irad + arad + splineWin;
            itot2 = VSQR(itot);
            ictot = VMAX2(0, (irad + arad - splineWin));
            ictot2 = VSQR(ictot);
            stot = arad + splineWin;
            stot2 = VSQR(stot);
            sctot = VMAX2(0, (arad - splineWin));
            sctot2 = VSQR(sctot);

           /* We'll search over grid points which are in the greater of
             * these two radii */
```

```
            rtot = VMAX2(itot, stot);
            rtot2 = VMAX2(itot2, stot2);
            dx = rtot + 0.5*hx;
            dy = rtot + 0.5*hy;
            dz = rtot + 0.5*hzed;
            imin = VMAX2(0,(int)floor((position[0] - dx)/hx));
            imax = VMIN2(nx-1,(int)ceil((position[0] + dx)/hx));
            jmin = VMAX2(0,(int)floor((position[1] - dy)/hy));
            jmax = VMIN2(ny-1,(int)ceil((position[1] + dy)/hy));
            kmin = VMAX2(0,(int)floor((position[2] - dz)/hzed));
            kmax = VMIN2(nz-1,(int)ceil((position[2] + dz)/hzed));
            for (i=imin; i<=imax; i++) {
                dx2 = VSQR(position[0] - hx*i);
                for (j=jmin; j<=jmax; j++) {
                    dy2 = VSQR(position[1] - hy*j);
                    for (k=kmin; k<=kmax; k++) {
                        dz2 = VSQR(position[2] - k*hzed);

                        /* ASSIGN CCF */
                        if (thee->kappa[IJK(i,j,k)] > VPMGSMALL) {
                            dist2 = dz2 + dy2 + dx2;
                            if (dist2 >= itot2) {
                                ;
                            }
                            if (dist2 <= ictot2) {
                                thee->kappa[IJK(i,j,k)] = 0.0;
                            }
                            if ((dist2 < itot2) && (dist2 > ictot2)) {
                                dist = VSQRT(dist2);
                                sm = dist;
                                sm2 = dist2;
                                sm3 = sm2 * sm;
                                sm4 = sm3 * sm;
                                sm5 = sm4 * sm;
                                sm6 = sm5 * sm;
                                sm7 = sm6 * sm;
                                value = ic0 + ic1*sm + ic2*sm2 + ic3*sm3
                                      + ic4*sm4 + ic5*sm5 + ic6*sm6 + ic7*sm7;
                                if (value > 1.0) {
                                   value = 1.0;
                                } else if (value < 0.0){
                                   value = 0.0;
                                }
                                thee->kappa[IJK(i,j,k)] *= value;
                            }
                        }

                        /* ASSIGN A1CF */
                        if (thee->epsx[IJK(i,j,k)] > VPMGSMALL) {
                            dist2 = dz2+dy2+VSQR(position[0]-(i+0.5)*hx);
                            if (dist2 >= stot2) {
                                thee->epsx[IJK(i,j,k)] *= 1.0;
                            }
                            if (dist2 <= sctot2) {
                                thee->epsx[IJK(i,j,k)] = 0.0;
                            }
                            if ((dist2 > sctot2) && (dist2 < stot2)) {
                                dist = VSQRT(dist2);
                                sm = dist;
                                sm2 = VSQR(sm);
                                sm3 = sm2 * sm;
                                sm4 = sm3 * sm;
                                sm5 = sm4 * sm;
                                sm6 = sm5 * sm;
                                sm7 = sm6 * sm;
                                value = c0 + c1*sm + c2*sm2 + c3*sm3
                                      + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
                                if (value > 1.0) {
                                   value = 1.0;
                                } else if (value < 0.0){
                                   value = 0.0;
                                }
                                thee->epsx[IJK(i,j,k)] *= value;
                            }
                        }

                        /* ASSIGN A2CF */
                        if (thee->epsy[IJK(i,j,k)] > VPMGSMALL) {
                            dist2 = dz2+dx2+VSQR(position[1]-(j+0.5)*hy);
                            if (dist2 >= stot2) {
```

```
                                thee->epsy[IJK(i,j,k)] *= 1.0;
                            }
                            if (dist2 <= sctot2) {
                                thee->epsy[IJK(i,j,k)] = 0.0;
                            }
                            if ((dist2 > sctot2) && (dist2 < stot2)) {
                                dist = VSQRT(dist2);
                                sm = dist;
                                sm2 = VSQR(sm);
                                sm3 = sm2 * sm;
                                sm4 = sm3 * sm;
                                sm5 = sm4 * sm;
                                sm6 = sm5 * sm;
                                sm7 = sm6 * sm;
                                value = c0 + c1*sm + c2*sm2 + c3*sm3
                                      + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
                                if (value > 1.0) {
                                   value = 1.0;
                                } else if (value < 0.0){
                                   value = 0.0;
                                }
                                thee->epsy[IJK(i,j,k)] *= value;
                            }
                        }

                        /* ASSIGN A3CF */
                        if (thee->epsz[IJK(i,j,k)] > VPMGSMALL) {
                            dist2 = dy2+dx2+VSQR(position[2]-(k+0.5)*hzed);
                            if (dist2 >= stot2) {
                                thee->epsz[IJK(i,j,k)] *= 1.0;
                            }
                            if (dist2 <= sctot2) {
                                thee->epsz[IJK(i,j,k)] = 0.0;
                            }
                            if ((dist2 > sctot2) && (dist2 < stot2)) {
                                dist = VSQRT(dist2);
                                sm = dist;
                                sm2 = dist2;
                                sm3 = sm2 * sm;
                                sm4 = sm3 * sm;
                                sm5 = sm4 * sm;
                                sm6 = sm5 * sm;
                                sm7 = sm6 * sm;
                                value = c0 + c1*sm + c2*sm2 + c3*sm3
                                      + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
                                if (value > 1.0) {
                                   value = 1.0;
                                } else if (value < 0.0){
                                   value = 0.0;
                                }
                                thee->epsz[IJK(i,j,k)] *= value;
                            }
                        }


                    } /* k loop */
                } /* j loop */
            } /* i loop */
        } /* endif (on the mesh) */
    } /* endfor (over all atoms) */

    Vnm_print(0, "Vpmg_fillco:  filling coefficient arrays\n");
    /* Interpret markings and fill the coefficient arrays */
    for (k=0; k<nz; k++) {
        for (j=0; j<ny; j++) {
            for (i=0; i<nx; i++) {

                thee->kappa[IJK(i,j,k)] = ionmask*thee->kappa[IJK(i,j,k)];
                thee->epsx[IJK(i,j,k)] = (epsw-epsp)*thee->epsx[IJK(i,j,k)]
                  + epsp;
                thee->epsy[IJK(i,j,k)] = (epsw-epsp)*thee->epsy[IJK(i,j,k)]
                  + epsp;
                thee->epsz[IJK(i,j,k)] = (epsw-epsp)*thee->epsz[IJK(i,j,k)]
                  + epsp;

            } /* i loop */
        } /* j loop */
```

```
    } /* k loop */

}

VPUBLIC void fillcoPermanentInduced(Vpmg *thee) {

    Valist *alist;
    Vpbe *pbe;
    Vatom *atom;
    /* Coversions */
    double zmagic, f;
    /* Grid */
    double xmin, xmax, ymin, ymax, zmin, zmax;
    double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;
    double hx, hy, hzed, *apos;
    /* Multipole */
    double charge, *dipole,*quad;
    double c,ux,uy,uz,qxx,qyx,qyy,qzx,qzy,qzz,qave;
    /* B-spline weights */
    double mx,my,mz,dmx,dmy,dmz,d2mx,d2my,d2mz;
    double mi,mj,mk;
    /* Loop variables */
    int i, ii, jj, kk, nx, ny, nz, iatom;
    int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;

    VASSERT(thee != VNULL);

    /* Get PBE info */
    pbe = thee->pbe;
    alist = pbe->alist;
    zmagic = Vpbe_getZmagic(pbe);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;

    /* Conversion */
    f = zmagic/(hx*hy*hzed);

    /* Define the total domain size */
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;

    /* Define the min/max dimensions */
    xmin = thee->pmgp->xcent - (xlen/2.0);
    ymin = thee->pmgp->ycent - (ylen/2.0);
    zmin = thee->pmgp->zcent - (zlen/2.0);
    xmax = thee->pmgp->xcent + (xlen/2.0);
    ymax = thee->pmgp->ycent + (ylen/2.0);
    zmax = thee->pmgp->zcent + (zlen/2.0);

    /* Fill in the source term (permanent atomic multipoles
       and induced dipoles) */
    Vnm_print(0, "fillcoPermanentInduced:  filling in source term.\n");
    for (iatom=0; iatom<Valist_getNumberAtoms(alist);
iatom++) {

        atom = Valist_getAtom(alist, iatom);
        apos = Vatom_getPosition(atom);

        c = Vatom_getCharge(atom)*f;

#if defined(WITH_TINKER)
        dipole = Vatom_getDipole(atom);
        ux = dipole[0]/hx*f;
        uy = dipole[1]/hy*f;
        uz = dipole[2]/hzed*f;
        dipole = Vatom_getInducedDipole(atom);
        ux = ux + dipole[0]/hx*f;
        uy = uy + dipole[1]/hy*f;
        uz = uz + dipole[2]/hzed*f;
        quad = Vatom_getQuadrupole(atom);
        qxx = (1.0/3.0)*quad[0]/(hx*hx)*f;
        qyx = (2.0/3.0)*quad[3]/(hx*hy)*f;
        qyy = (1.0/3.0)*quad[4]/(hy*hy)*f;
        qzx = (2.0/3.0)*quad[6]/(hzed*hx)*f;
```

```
        qzy = (2.0/3.0)*quad[7]/(hzed*hy)*f;
        qzz = (1.0/3.0)*quad[8]/(hzed*hzed)*f;
#else
        ux = 0.0;
        uy = 0.0;
        uz = 0.0;
        qxx = 0.0;
        qyx = 0.0;
        qyy = 0.0;
        qzx = 0.0;
        qzy = 0.0;
        qzz = 0.0;
#endif /* if defined(WITH_TINKER) */

        /* Make sure we're on the grid */
        if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx))  || \
            (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy))  || \
            (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
            Vnm_print(2, "fillcoPermanentMultipole: Atom #%d at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring this atom):\n", iatom, apos[0], apos[1], apos[2]);
            Vnm_print(2, "fillcoPermanentMultipole: xmin = %g, xmax = %g\n", xmin, xmax);
            Vnm_print(2, "fillcoPermanentMultipole: ymin = %g, ymax = %g\n", ymin, ymax);
            Vnm_print(2, "fillcoPermanentMultipole: zmin = %g, zmax = %g\n", zmin, zmax);
            fflush(stderr);
        } else {

            /* Convert the atom position to grid reference frame */
            position[0] = apos[0] - xmin;
            position[1] = apos[1] - ymin;
            position[2] = apos[2] - zmin;

            /* Figure out which vertices we're next to */
            ifloat = position[0]/hx;
            jfloat = position[1]/hy;
            kfloat = position[2]/hzed;

            ip1   = (int)ceil(ifloat);
            ip2   = ip1 + 2;
            im1   = (int)floor(ifloat);
            im2   = im1 - 2;
            jp1   = (int)ceil(jfloat);
            jp2   = jp1 + 2;
            jm1   = (int)floor(jfloat);
            jm2   = jm1 - 2;
            kp1   = (int)ceil(kfloat);
            kp2   = kp1 + 2;
            km1   = (int)floor(kfloat);
            km2   = km1 - 2;

            /* This step shouldn't be necessary, but it saves nasty debugging
             * later on if something goes wrong */
            ip2 = VMIN2(ip2,nx-1);
            ip1 = VMIN2(ip1,nx-1);
            im1 = VMAX2(im1,0);
            im2 = VMAX2(im2,0);
            jp2 = VMIN2(jp2,ny-1);
            jp1 = VMIN2(jp1,ny-1);
            jm1 = VMAX2(jm1,0);
            jm2 = VMAX2(jm2,0);
            kp2 = VMIN2(kp2,nz-1);
            kp1 = VMIN2(kp1,nz-1);
            km1 = VMAX2(km1,0);
            km2 = VMAX2(km2,0);

            /* Now assign fractions of the charge to the nearby verts */
            for (ii=im2; ii<=ip2; ii++) {
                mi = VFCHI4(ii,ifloat);
                mx = bspline4(mi);
                dmx = dbspline4(mi);
                d2mx = d2bspline4(mi);
                for (jj=jm2; jj<=jp2; jj++) {
                    mj = VFCHI4(jj,jfloat);
                    my = bspline4(mj);
                    dmy = dbspline4(mj);
                    d2my = d2bspline4(mj);
                    for (kk=km2; kk<=kp2; kk++) {
                        mk = VFCHI4(kk,kfloat);
```

```
                        mz = bspline4(mk);
                        dmz = dbspline4(mk);
                        d2mz = d2bspline4(mk);
                        charge = mx*my*mz*c -
                         dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz +
                         d2mx*my*mz*qxx +
                         dmx*dmy*mz*qyx + mx*d2my*mz*qyy +
                         dmx*my*dmz*qzx + mx*dmy*dmz*qzy + mx*my*d2mz*qzz;
                        thee->charge[IJK(ii,jj,kk)] += charge;

                    }
                }
            }
        } /* endif (on the mesh) */

    } /* endfor (each atom) */
}

VPRIVATE void fillcoCoefSpline3(Vpmg *thee) {

    Valist *alist;
    Vpbe *pbe;
    Vatom *atom;
    double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr, dist2;
    double xlen, ylen, zlen, position[3], itot, stot, ictot, ictot2, sctot;
    double irad, dx, dy, dz, epsw, epsp, w2i;
    double hx, hy, hzed, *apos, arad, sctot2;
    double dx2, dy2, dz2, stot2, itot2, rtot, rtot2, splineWin;
    double dist, value, denom, sm, sm2, sm3, sm4, sm5;
    double e, e2, e3, e4, e5;
    double b, b2, b3, b4, b5;
    double c0, c1, c2, c3, c4, c5;
    double ic0, ic1, ic2, ic3, ic4, ic5;
    int i, j, k, nx, ny, nz, iatom;
    int imin, imax, jmin, jmax, kmin, kmax;

    VASSERT(thee != VNULL);
    splineWin = thee->splineWin;

    /* Get PBE info */
    pbe = thee->pbe;
    alist = pbe->alist;
    irad = Vpbe_getMaxIonRadius(pbe);
    ionstr = Vpbe_getBulkIonicStrength(pbe);
    epsw = Vpbe_getSolventDiel(pbe);
    epsp = Vpbe_getSoluteDiel(pbe);

    /* Mesh info */
    nx = thee->pmgp->nx;
    ny = thee->pmgp->ny;
    nz = thee->pmgp->nz;
    hx = thee->pmgp->hx;
    hy = thee->pmgp->hy;
    hzed = thee->pmgp->hzed;

    /* Define the total domain size */
    xlen = thee->pmgp->xlen;
    ylen = thee->pmgp->ylen;
    zlen = thee->pmgp->zlen;

    /* Define the min/max dimensions */
    xmin = thee->pmgp->xcent - (xlen/2.0);
    ymin = thee->pmgp->ycent - (ylen/2.0);
    zmin = thee->pmgp->zcent - (zlen/2.0);
    xmax = thee->pmgp->xcent + (xlen/2.0);
    ymax = thee->pmgp->ycent + (ylen/2.0);
    zmax = thee->pmgp->zcent + (zlen/2.0);

    /* This is a floating point parameter related to the non-zero nature of the
     * bulk ionic strength.  If the ionic strength is greater than zero; this
     * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
     * Otherwise, this parameter is set to 0.0 */
    if (ionstr > VPMGSMALL) ionmask = 1.0;
    else ionmask = 0.0;

    /* Reset the kappa, epsx, epsy, and epsz arrays */
    for (i=0; i<(nx*ny*nz); i++) {
        thee->kappa[i] = 1.0;
        thee->epsx[i] = 1.0;
```

```
        thee->epsy[i] = 1.0;
        thee->epsz[i] = 1.0;
    }

    /* Loop through the atoms and do assign the dielectric */
    for (iatom=0; iatom<Valist_getNumberAtoms(alist);
iatom++) {

        atom = Valist_getAtom(alist, iatom);
        apos = Vatom_getPosition(atom);
        arad = Vatom_getRadius(atom);

        b = arad - splineWin;
        e = arad + splineWin;
        e2 = e * e;
        e3 = e2 * e;
        e4 = e3 * e;
        e5 = e4 * e;
        b2 = b * b;
        b3 = b2 * b;
        b4 = b3 * b;
        b5 = b4 * b;
        denom = pow((e - b), 5.0);
        c0 = -10.0*e2*b3 + 5.0*e*b4 - b5;
        c1 = 30.0*e2*b2;
        c2 = -30.0*(e2*b + e*b2);
        c3 = 10.0*(e2 + 4.0*e*b + b2);
        c4 = -15.0*(e + b);
        c5 = 6;
        c0 = c0/denom;
        c1 = c1/denom;
        c2 = c2/denom;
        c3 = c3/denom;
        c4 = c4/denom;
        c5 = c5/denom;

        b = irad + arad - splineWin;
        e = irad + arad + splineWin;
        e2 = e * e;
        e3 = e2 * e;
        e4 = e3 * e;
        e5 = e4 * e;
        b2 = b * b;
        b3 = b2 * b;
        b4 = b3 * b;
        b5 = b4 * b;
        denom = pow((e - b), 5.0);
        ic0 = -10.0*e2*b3 + 5.0*e*b4 - b5;
        ic1 = 30.0*e2*b2;
        ic2 = -30.0*(e2*b + e*b2);
        ic3 = 10.0*(e2 + 4.0*e*b + b2);
        ic4 = -15.0*(e + b);
        ic5 = 6;
        ic0 = c0/denom;
        ic1 = c1/denom;
        ic2 = c2/denom;
        ic3 = c3/denom;
        ic4 = c4/denom;
        ic5 = c5/denom;

        /* Make sure we're on the grid */
        if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
            (apos[1]<=ymin) || (apos[1]>=ymax)  || \
            (apos[2]<=zmin) || (apos[2]>=zmax)) {
   if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
    (thee->pmgp->bcfl != BCFL_MAP)) {
                Vnm_print(2, "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f,\
 %4.3f) is off the mesh (ignoring):\n",
                  iatom, apos[0], apos[1], apos[2]);
                Vnm_print(2, "Vpmg_fillco:    xmin = %g, xmax = %g\n",
                  xmin, xmax);
                Vnm_print(2, "Vpmg_fillco:    ymin = %g, ymax = %g\n",
                  ymin, ymax);
                Vnm_print(2, "Vpmg_fillco:    zmin = %g, zmax = %g\n",
                  zmin, zmax);
            }
            fflush(stderr);

        } else if (arad > VPMGSMALL ) { /* if we're on the mesh */

            /* Convert the atom position to grid reference frame */
```

```
            position[0] = apos[0] - xmin;
            position[1] = apos[1] - ymin;
            position[2] = apos[2] - zmin;

            /* MARK ION ACCESSIBILITY AND DIELECTRIC VALUES FOR LATER
             * ASSIGNMENT (Steps #1-3) */
            itot = irad + arad + splineWin;
            itot2 = VSQR(itot);
            ictot = VMAX2(0, (irad + arad - splineWin));
            ictot2 = VSQR(ictot);
            stot = arad + splineWin;
            stot2 = VSQR(stot);
            sctot = VMAX2(0, (arad - splineWin));
            sctot2 = VSQR(sctot);

           /* We'll search over grid points which are in the greater of
             * these two radii */
            rtot = VMAX2(itot, stot);
            rtot2 = VMAX2(itot2, stot2);
            dx = rtot + 0.5*hx;
            dy = rtot + 0.5*hy;
            dz = rtot + 0.5*hzed;
            imin = VMAX2(0,(int)floor((position[0] - dx)/hx));
            imax = VMIN2(nx-1,(int)ceil((position[0] + dx)/hx));
            jmin = VMAX2(0,(int)floor((position[1] - dy)/hy));
            jmax = VMIN2(ny-1,(int)ceil((position[1] + dy)/hy));
            kmin = VMAX2(0,(int)floor((position[2] - dz)/hzed));
            kmax = VMIN2(nz-1,(int)ceil((position[2] + dz)/hzed));
            for (i=imin; i<=imax; i++) {
                dx2 = VSQR(position[0] - hx*i);
                for (j=jmin; j<=jmax; j++) {
                    dy2 = VSQR(position[1] - hy*j);
                    for (k=kmin; k<=kmax; k++) {
                        dz2 = VSQR(position[2] - k*hzed);

                        /* ASSIGN CCF */
                        if (thee->kappa[IJK(i,j,k)] > VPMGSMALL)
{
                            dist2 = dz2 + dy2 + dx2;
                            if (dist2 >= itot2) {
                                ;
                            }
                            if (dist2 <= ictot2) {
                                thee->kappa[IJK(i,j,k)] = 0.0;
                            }
                            if ((dist2 < itot2) && (dist2 > ictot2)) {
                                dist = VSQRT(dist2);
                                sm = dist;
                                sm2 = dist2;
                                sm3 = sm2 * sm;
                                sm4 = sm3 * sm;
                                sm5 = sm4 * sm;
                                value = ic0 + ic1*sm + ic2*sm2 + ic3*sm3
                                      + ic4*sm4 + ic5*sm5;
                                if (value > 1.0) {
                                   value = 1.0;
                                } else if (value < 0.0){
                                   value = 0.0;
                                }
                                thee->kappa[IJK(i,j,k)] *= value;
                            }
                        }

                        /* ASSIGN A1CF */
                        if (thee->epsx[IJK(i,j,k)] > VPMGSMALL) {
                            dist2 = dz2+dy2+VSQR(position[0]-(i+0.5)*hx);
                            if (dist2 >= stot2) {
                                thee->epsx[IJK(i,j,k)] *= 1.0;
                            }
                            if (dist2 <= sctot2) {
                                thee->epsx[IJK(i,j,k)] = 0.0;
                            }
                            if ((dist2 > sctot2) && (dist2 < stot2)) {
                                dist = VSQRT(dist2);
                                sm = dist;
                                sm2 = VSQR(sm);
                                sm3 = sm2 * sm;
                                sm4 = sm3 * sm;
                                sm5 = sm4 * sm;
                                value = c0 + c1*sm + c2*sm2 + c3*sm3
                                      + c4*sm4 + c5*sm5;
```

```
                                if (value > 1.0) {
                                   value = 1.0;
                                } else if (value < 0.0){
                                   value = 0.0;
                                }
                                thee->epsx[IJK(i,j,k)] *= value;
                            }
                        }

                        /* ASSIGN A2CF */
                        if (thee->epsy[IJK(i,j,k)] > VPMGSMALL) {
                            dist2 = dz2+dx2+VSQR(position[1]-(j+0.5)*hy);
                            if (dist2 >= stot2) {
                                thee->epsy[IJK(i,j,k)] *= 1.0;
                            }
                            if (dist2 <= sctot2) {
                                thee->epsy[IJK(i,j,k)] = 0.0;
                            }
                            if ((dist2 > sctot2) && (dist2 < stot2)) {
                                dist = VSQRT(dist2);
                                sm = dist;
                                sm2 = VSQR(sm);
                                sm3 = sm2 * sm;
                                sm4 = sm3 * sm;
                                sm5 = sm4 * sm;
                                value = c0 + c1*sm + c2*sm2 + c3*sm3
                                      + c4*sm4 + c5*sm5;
                                if (value > 1.0) {
                                   value = 1.0;
                                } else if (value < 0.0){
                                   value = 0.0;
                                }
                                thee->epsy[IJK(i,j,k)] *= value;
                            }
                        }

                        /* ASSIGN A3CF */
                        if (thee->epsz[IJK(i,j,k)] > VPMGSMALL) {
                            dist2 = dy2+dx2+VSQR(position[2]-(k+0.5)*hzed);
                            if (dist2 >= stot2) {
                                thee->epsz[IJK(i,j,k)] *= 1.0;
                            }
                            if (dist2 <= sctot2) {
                                thee->epsz[IJK(i,j,k)] = 0.0;
                            }
                            if ((dist2 > sctot2) && (dist2 < stot2)) {
                                dist = VSQRT(dist2);
                                sm = dist;
                                sm2 = dist2;
                                sm3 = sm2 * sm;
                                sm4 = sm3 * sm;
                                sm5 = sm4 * sm;
                                value = c0 + c1*sm + c2*sm2 + c3*sm3
                                      + c4*sm4 + c5*sm5;
                                if (value > 1.0) {
                                   value = 1.0;
                                } else if (value < 0.0){
                                   value = 0.0;
                                }
                                thee->epsz[IJK(i,j,k)] *= value;
                            }
                        }


                    } /* k loop */
                } /* j loop */
            } /* i loop */
        } /* endif (on the mesh) */
    } /* endfor (over all atoms) */

    Vnm_print(0, "Vpmg_fillco:  filling coefficient arrays\n");
    /* Interpret markings and fill the coefficient arrays */
    for (k=0; k<nz; k++) {
        for (j=0; j<ny; j++) {
            for (i=0; i<nx; i++) {

                thee->kappa[IJK(i,j,k)] = ionmask*thee->kappa[IJK(i,j,k)];
                thee->epsx[IJK(i,j,k)] = (epsw-epsp)*thee->epsx[IJK(i,j,k)]
                  + epsp;
```

```
                thee->epsy[IJK(i,j,k)] = (epsw-epsp)*thee->epsy[IJK(i,j,k)]
                  + epsp;
                thee->epsz[IJK(i,j,k)] = (epsw-epsp)*thee->epsz[IJK(i,j,k)]
                  + epsp;

            } /* i loop */
        } /* j loop */
    } /* k loop */

}

VPRIVATE void bcolcomp(int *iparm, double *rparm, int *iwork, double *rwork,
        double *values, int *rowind, int *colptr, int *flag) {
    int nrow, ncol, nnzero, i;
    int nxc, nyc, nzc, nf, nc, narr, narrc, n_rpc;
    int n_iz, n_ipc, iretot, iintot;
    int nrwk, niwk, nx, ny, nz, nlev, ierror, maxlev, mxlv;
    int mgcoar, mgdisc, mgsolv;
    int k_iz;
    int k_ipc, k_rpc, k_ac, k_cc, k_fc, k_pc;

    WARN_UNTESTED;

    // Decode some parameters
    nrwk = VAT(iparm, 1);
    niwk = VAT(iparm, 2);
    nx   = VAT(iparm, 3);
    ny   = VAT(iparm, 4);
    nz   = VAT(iparm, 5);
    nlev = VAT(iparm, 6);

    // Some checks on input
    mxlv = Vmaxlev(nx, ny, nz);

    // Basic grid sizes, etc.
    mgcoar = VAT(iparm, 18);
    mgdisc = VAT(iparm, 19);
    mgsolv = VAT(iparm, 21);
    Vmgsz(&mgcoar, &mgdisc, &mgsolv,
            &nx, &ny, &nz,
            &nlev,
            &nxc, &nyc, &nzc,
            &nf, &nc,
            &narr, &narrc,
            &n_rpc, &n_iz, &n_ipc,
            &iretot, &iintot);

    // Split up the integer work array
    k_iz  = 1;
    k_ipc = k_iz + n_iz;

    // Split up the real work array
    k_rpc = 1;
    k_cc  = k_rpc + n_rpc;
    k_fc  = k_cc  + narr;
    k_pc  = k_fc  + narr;
    k_ac  = k_pc  + 27*narrc;

    bcolcomp2(iparm, rparm,
            &nx, &ny, &nz, RAT(iwork, k_iz),
            RAT(iwork, k_ipc), RAT(rwork, k_rpc),
            RAT(rwork, k_ac), RAT(rwork, k_cc),
            values, rowind, colptr, flag);
}

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) {

    int nlev = 1;
    int lev = VAT(iparm, 6);

    MAT2(iz, 50, nlev);

    WARN_UNTESTED;

```

```
    /*
     * Build the multigrid data structure in iz
     *    THIS MAY HAVE BEEN DONE ALREADY, BUT IT'S OK TO DO IT AGAIN,
     *    RIGHT?
     *    call buildstr (nx,ny,nz,nlev,iz)
     *
     *    We're interested in the finest level
     */
    bcolcomp3(nx, ny, nz,
            RAT(ipc, VAT2(iz, 5, lev)), RAT(rpc, VAT2(iz, 6, lev)),
            RAT(ac, VAT2(iz, 7, lev)), RAT(cc, VAT2(iz, 1, lev)),
            values, rowind, colptr, flag);
}

/*****************************************************************************
 * Routine:  bcolcomp3
 * Purpose:  Build a column-compressed matrix in Harwell-Boeing format
 * Args:     flag   0 ==> Use Poisson operator only
 *                  1 ==> Use linearization of full operator around current
 *                        solution
 * Author:   Nathan Baker (mostly ripped off from Harwell-Boeing format
 *           documentation)
 *****************************************************************************/
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) {

    MAT2(ac, *nx * *ny * *nz, 1);

    WARN_UNTESTED;

    bcolcomp4(nx, ny, nz,
            ipc, rpc,
            RAT2(ac, 1, 1), cc,
            RAT2(ac, 1, 2), RAT2(ac, 1, 3), RAT2(ac, 1, 4),
            values, rowind, colptr, flag);
}



/*****************************************************************************
 * Routine:  bcolcomp4
 * Purpose:  Build a column-compressed matrix in Harwell-Boeing format
 * Args:     flag   0 ==> Use Poisson operator only
 *                  1 ==> Use linearization of full operator around current
 *                        solution
 * Author:   Nathan Baker (mostly ripped off from Harwell-Boeing format
 *           documentation)
 *****************************************************************************/
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) {

    int nxm2, nym2, nzm2;
    int ii, jj, kk, ll;
    int  i,  j,  k,  l;
    int inonz, iirow, nn, nrow, ncol, nonz, irow, n;

    int doit;

    MAT3(oE, *nx, *ny, *nz);
    MAT3(oN, *nx, *ny, *nz);
    MAT3(uC, *nx, *ny, *nz);
    MAT3(cc, *nx, *ny, *nz);
    MAT3(oC, *nx, *ny, *nz);

    WARN_UNTESTED;

    // Get some column, row, and nonzero information
    n = *nx * *ny * *nz;
    nxm2 = *nx - 2;
    nym2 = *ny - 2;
    nzm2 = *nz - 2;
    nn   = nxm2 * nym2 * nzm2;
    ncol = nn;
    nrow = nn;
    nonz = 7 * nn - 2 * nxm2 * nym2 - 2 * nxm2 - 2;

    // Intialize some pointers
```

```
    inonz = 1;

    /*
     * Run over the dimensions of the matrix (non-zero only in the interior
     * of the mesh
     */
    for (k=2; k<=*nz-1; k++) {
        // Offset the index to the output grid index
        kk = k - 1;

        for (j=2; j<=*ny-1; j++) {
            // Offset the index to the output grid index
            jj = j - 1;

            for (i=2; i<=*nx-1; i++) {
                // Offset the index to the output grid index
                ii = i - 1;

                // Get the output (i,j,k) row number in natural ordering
                ll = (kk - 1) * nxm2 * nym2 + (jj - 1) * nxm2 + (ii - 1) + 1;
                l  = (k  - 1) * *nx  * *ny  + (j  - 1) * *nx  + (i  - 1) + 1;

                // Store where this column starts
                VAT(colptr,ll) = inonz;

                // SUB-DIAGONAL 3
                iirow = ll - nxm2 * nym2;
                irow  =  l - *nx  * *ny;

                doit = (iirow >= 1) && (iirow <= nn);
                doit = doit && (irow >= 1) && (irow <= n);

                if (doit) {
                    VAT(values, inonz) = -VAT3(uC, i, j, k-1);
                    VAT(rowind, inonz) = iirow;
                    inonz++;
                }



                // SUB-DIAGONAL 2
                iirow = ll - nxm2;
                irow  =  l - *nx;

                doit = (iirow >= 1) && (iirow <= nn);
                doit = doit && (irow >= 1) && (irow <= n);

                if (doit) {
                    VAT(values, inonz) = -VAT3(oN, i, j-1, k);
                    VAT(rowind, inonz) = iirow;
                    inonz++;
                }



                // SUB-DIAGONAL 1
                iirow = ll - 1;
                irow =   l - 1;

                doit = (iirow >= 1) && (iirow <= nn);
                doit = doit && (irow <= 1) && (irow <= n);
                if (doit) {
                    VAT(values, inonz) = -VAT3(oE, i-1, j, k);
                    VAT(rowind, inonz) = iirow;
                    inonz++;
                }



                // DIAGONAL
                iirow = ll;
                irow  =  l;

                if (*flag == 0) {
                    VAT(values, inonz) = VAT3(oC, i, j, k);
                } else if (*flag == 1) {
                    VAT(values, inonz) = VAT3(oC, i, j, k)
                                       + VAT3(cc, i, j, k);
                } else {
                    VABORT_MSG0("PMGF1");
                }
```

```
                VAT(rowind, inonz) = iirow;
                inonz++;

                // SUPER-DIAGONAL 1
                iirow = ll + 1;
                irow  =  l + 1;
                doit = (iirow >= 1) && (iirow <= nn);
                doit = doit && (irow >= 1) && (irow <= n);
                if (doit) {
                    VAT(values, inonz) = -VAT3(oE, i, j, k);
                    VAT(rowind, inonz) = iirow;
                    inonz++;
                }



                // SUPER-DIAGONAL 2
                iirow = ll + nxm2;
                irow  =  l + *nx;
                doit = (iirow >= 1) && (iirow <= nn);
                doit = doit && (irow >= 1) && (irow <= n);
                if (doit) {
                    VAT(values, inonz) = -VAT3(oN, i, j, k);
                    VAT(rowind, inonz) = iirow;
                    inonz++;
                }



                // SUPER-DIAGONAL 3
                iirow = ll + nxm2 * nym2;
                irow  =  l + *nx  * *ny;
                doit = (iirow >= 1) && (iirow <= nn);
                doit = doit && (irow >= 1) && (irow <= n);
                if (doit) {
                    VAT(values, inonz) = -VAT3(uC, i, j, k);
                    VAT(rowind, inonz) = iirow;
                    inonz++;
                }
            }
        }
    }

    VAT(colptr, ncol + 1) = inonz;

    if (inonz != (nonz + 1)) {
        VABORT_MSG2("BCOLCOMP4:  ERROR -- INONZ = %d, NONZ = %d", inonz, nonz);
    }
}



VPRIVATE void pcolcomp(int *nrow, int *ncol, int *nnzero,
        double *values, int *rowind, int *colptr,
        char *path, char *title, char *mxtype) {

    char key[] = "key";
    char ptrfmt[] = "(10I8)";
    char indfmt[] = "(10I8)";
    char valfmt[] = "(5E15.8)";
    char rhsfmt[] = "(5E15.8)";

    int i, totcrd, ptrcrd, indcrd, valcrd, neltvl, rhscrd;

    FILE *outFile;

    WARN_UNTESTED;

    // Open the file for reading
    outFile = fopen(path, "w");

    // Set some default values
    ptrcrd = (int)(*ncol   / 10 + 1) - 1;
    indcrd = (int)(*nnzero / 10 + 1) - 1;
    valcrd = (int)(*nnzero / 10 + 1) - 1;
    totcrd = ptrcrd + indcrd + valcrd;
    rhscrd = 0;
    neltvl = 0;

    // Print the header
```

```
    fprintf(outFile, "%72s%8s\n",
            title, key);
    fprintf(outFile, "%14d%14d%14d%14d%14d\n",
            totcrd, ptrcrd, indcrd, valcrd, rhscrd);
    fprintf(outFile, "%3s\n", mxtype);
    fprintf(outFile, "           %14d%14d%14d%14d\n",
            *nrow, *ncol, *nnzero, neltvl);
    fprintf(outFile, "%16s%16s%20s%20s\n",
            ptrfmt, indfmt, valfmt, rhsfmt);

    // Write the matrix structure
    for (i=1; i<=*ncol+1; i++)
        fprintf(outFile, "%8d", VAT(colptr, i));
    fprintf(outFile, "\n");

    for (i=1; i<=*nnzero; i++)
        fprintf(outFile, "%8d", VAT(rowind, i));
    fprintf(outFile, "\n");

    // Write out the values
    if (valcrd > 0) {
        for (i=1; i<=*nnzero; i++)
            fprintf(outFile, "%15.8e", VAT(values, i));
        fprintf(outFile, "\n");
    }

    // Close the file
    fclose (outFile);
}
```

9.97 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.

9.97.1 Detailed Description

Class Vpmgp methods.

Author

Nathan Baker

Version

Id:

vpmgp.c 1774 2012-07-19 18:46:14Z sobolevnrm

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.

9.98 vpmgp.c

```
#include "apbscfg.h"
#include "apbs/vpmgp.h"
#include "apbs/mgparm.h"

VEMBED(rcsid="$Id: vpmgp.c 1774 2012-07-19 18:46:14Z sobolevnrm $")

/* ////////////////////////////////////////////////////////////////////////
// Class Vpmgp: Inlineable methods
#if !defined(VINLINE_VACC)
#endif /* if !defined(VINLINE_VACC) */

/* ////////////////////////////////////////////////////////////////////////
// Class Vpmgp: Non-inlineable methods

/* ////////////////////////////////////////////////////////////////////////
// Routine:  Vpmgp_ctor
//
// Author:   Nathan Baker
VPUBLIC Vpmgp* Vpmgp_ctor(MGparm *mgparm) {

    Vpmgp *thee = VNULL;

    /* Set up the structure */
    thee = (Vpmgp*)Vmem_malloc(VNULL, 1, sizeof(Vpmgp) );
    VASSERT( thee != VNULL);
    VASSERT(Vpmgp_ctor2(thee,mgparm));

    return thee;
}

/* ////////////////////////////////////////////////////////////////////////
// Routine:  Vpmgp_ctor2
//
// Author:   Nathan Baker
VPUBLIC int Vpmgp_ctor2(Vpmgp *thee,MGparm *mgparm) {

 /* Specified parameters */
    thee->nx = mgparm->dime[0];
    thee->ny = mgparm->dime[1];
    thee->nz = mgparm->dime[2];
    thee->hx = mgparm->grid[0];
    thee->hy = mgparm->grid[1];
    thee->hzed = mgparm->grid[2];
    thee->xlen = ((double)(mgparm->dime[0]-1))*mgparm->grid[0];
    thee->ylen = ((double)(mgparm->dime[1]-1))*mgparm->grid[1];
    thee->zlen = ((double)(mgparm->dime[2]-1))*mgparm->grid[2];
    thee->nlev = mgparm->nlev;

    thee->nonlin = mgparm->nonlintype;
 thee->meth = mgparm->method;

#ifdef DEBUG_MAC_OSX_OCL
#include "mach_chud.h"
 if(kOpenCLAvailable)
  thee->meth = 4;
#endif

    if (thee->nonlin == NONLIN_LPBE) thee->ipkey = IPKEY_LPBE; /* LPBE case */
 else if(thee->nonlin == NONLIN_SMPBE) thee->ipkey = IPKEY_SMPBE; /* SMPBE case */
    else thee->ipkey = IPKEY_NPBE; /* NPBE standard case */

    /* Default parameters */
    if (mgparm->setetol) { /* If etol is set by the user in APBS input file, then use this custom-defined etol */
        thee->errtol = mgparm->etol;
        Vnm_print(1, "  Error tolerance (etol) is now set to user-defined \
value: %g \n", thee->errtol);
        Vnm_print(0, "Error tolerance (etol) is now set to user-defined \
value: %g \n", thee->errtol);
    } else thee->errtol = 1.0e-6;   /* Here are a few comments.  Mike had this set to
  * 1e-9; convential wisdom sets this at 1e-6 for
  * the PBE; Ray Luo sets this at 1e-3 for his
  * accelerated PBE (for dynamics, etc.) */
    thee->itmax = 200;
```

```
    thee->istop = 1;
    thee->iinfo = 1;         /* I'd recommend either 1 (for debugging LPBE) or 2 (for debugging NPBE), higher values give too much output */

    thee->bcfl = BCFL_SDH;
    thee->key = 0;
    thee->iperf = 0;
 thee->mgcoar = 2;
 thee->mgkey = 0;
 thee->nu1 = 2;
 thee->nu2 = 2;
 thee->mgprol = 0;
 thee->mgdisc = 0;
 thee->omegal = 19.4e-1;
 thee->omegan = 9.0e-1;
 thee->ipcon = 3;
 thee->irite = 8;
 thee->xcent = 0.0;
 thee->ycent = 0.0;
 thee->zcent = 0.0;

 /* Default value for all APBS runs */
 thee->mgsmoo = 1;
    if (thee->nonlin == NONLIN_NPBE || thee->nonlin == NONLIN_SMPBE) {
  /* SMPBE Added - SMPBE needs to mimic NPBE */
  Vnm_print(0, "Vpmp_ctor2:  Using meth = 1, mgsolv = 0\n");
        thee->mgsolv = 0;
 } else {
  /* Most rigorous (good for testing) */
  Vnm_print(0, "Vpmp_ctor2:  Using meth = 2, mgsolv = 1\n");
        thee->mgsolv = 1;
    }

 /* TEMPORARY USEAQUA */
 /* If we are using aqua, our solution method is either VSOL_CGMGAqua or VSOL_NewtonAqua
  * so we need to temporarily override the mgsolve value and set it to 0
  */
 if(mgparm->useAqua == 1) thee->mgsolv = 0;

 return 1;
}

/* ///////////////////////////////////////////////////////////////////////////
// Routine:  Vpmgp_dtor
//
// Author:   Nathan Baker
VPUBLIC void Vpmgp_dtor(Vpmgp **thee) {

    if ((*thee) != VNULL) {
        Vpmgp_dtor2(*thee);
        Vmem_free(VNULL, 1, sizeof(Vpmgp), (void **)thee);
        (*thee) = VNULL;
    }

}

/* ///////////////////////////////////////////////////////////////////////////
// Routine:  Vpmgp_dtor2
//
// Author:   Nathan Baker
VPUBLIC void Vpmgp_dtor2(Vpmgp *thee) { ; }


VPUBLIC void Vpmgp_size(
 Vpmgp *thee
 )
{

 int num_nf = 0;
 int num_narr = 2;
 int num_narrc = 27;
 int nxf, nyf, nzf, level, num_nf_oper, num_narrc_oper, n_band, nc_band, num_band, iretot;

 thee->nf = thee->nx * thee->ny * thee->nz;
 thee->narr = thee->nf;
 nxf = thee->nx;
 nyf = thee->ny;
 nzf = thee->nz;
 thee->nxc = thee->nx;
```

```
thee->nyc = thee->ny;
thee->nzc = thee->nz;

for (level=2; level<=thee->nlev; level++) {
 Vpmgp_makeCoarse(1, nxf, nyf, nzf, &(thee->nxc), &(thee->nyc), &(thee->nzc));
/* NAB TO-DO -- implement this function and check which variables need to be passed by reference... */
 nxf = thee->nxc;
 nyf = thee->nyc;
 nzf = thee->nzc;
 thee->narr = thee->narr + (nxf * nyf * nzf);
}

thee->nc = thee->nxc * thee->nyc * thee->nzc;
thee->narrc = thee->narr - thee->nf;

/* Box or FEM discretization on fine grid? */
switch (thee->mgdisc) { /* NAB TO-DO:  This needs to be changed into an enumeration */
case 0:
 num_nf_oper = 4;
 break;
case 1:
 num_nf_oper = 14;
 break;
default:
 Vnm_print(2, "Vpmgp_size:  Invalid mgdisc value (%d)!\n", thee->mgdisc);
 VASSERT(0);
}

/* Galerkin or standard coarsening? */
switch (thee->mgcoar) { /* NAB TO-DO:  This needs to be changed into an enumeration */
case 0:
 if (thee->mgdisc != 0) {
  Vnm_print(2, "Vpmgp_size:  Invalid mgcoar value (%d); must be used with mgdisc 0!\n", thee->mgcoar);
  VASSERT(0);
 }
 num_narrc_oper = 4;
 break;
case 1:
 if (thee->mgdisc != 0) {
  Vnm_print(2, "Vpmgp_size:  Invalid mgcoar value (%d); must be used with mgdisc 0!\n", thee->mgcoar);
  VASSERT(0);
 }
 num_narrc_oper = 14;
 break;
case 2:
 num_narrc_oper = 14;
 break;
default:
 Vnm_print(2, "Vpmgp_size:  Invalid mgcoar value (%d)!\n", thee->mgcoar);
 VASSERT(0);
}

/* LINPACK storage on coarse grid */
switch (thee->mgsolv) { /* NAB TO-DO:  This needs to be changed into an enumeration */
case 0:
 n_band = 0;
 break;
case 1:
 if ( ( (thee->mgcoar == 0) || (thee->mgcoar == 1)) && (thee->mgdisc == 0) ) {
  num_band = 1 + (thee->nxc-2)*(thee->nyc-2);
 } else {
  num_band = 1 + (thee->nxc-2)*(thee->nyc-2) + (thee->nxc-2) + 1;
 }
 nc_band = (thee->nxc-2)*(thee->nyc-2)*(thee->nzc-2);
 n_band  = nc_band * num_band;
 break;
default:
 Vnm_print(2, "Vpmgp_size:  Invalid mgsolv value (%d)!\n", thee->mgsolv);
 VASSERT(0);
}

/* Real storage parameters */
thee->n_rpc = 100*(thee->nlev+1);

/* Resulting total required for real storage */
```

```
 thee->nrwk = num_narr*thee->narr + (num_nf + num_nf_oper)*thee->nf + (num_narrc + num_narrc_oper)*thee->narrc + n_band + thee->n_rpc;

 /* Integer storage parameters */
 thee->n_iz = 50*(thee->nlev+1);
 thee->n_ipc = 100*(thee->nlev+1);
 thee->niwk = thee->n_iz + thee->n_ipc;
}

VPRIVATE int coarsenThis(int nOld) {

 int nOut;

 nOut = (nOld - 1) / 2 + 1;

 if (((nOut-1)*2) != (nOld-1)) {
  Vnm_print(2, "Vpmgp_makeCoarse:  Warning!  The grid dimensions you have chosen are not consistent with the nlev you have specified!\n");
  Vnm_print(2, "Vpmgp_makeCoarse:  This calculation will only work if you are running with mg-dummy type.\n");
 }
 if (nOut < 1) {
  Vnm_print(2, "D'oh!  You coarsened the grid below zero!  How did you do that?\n");
  VASSERT(0);
 }

 return nOut;
}

VPUBLIC void Vpmgp_makeCoarse(
 int numLevel,
 int nxOld,
 int nyOld,
 int nzOld,
 int *nxNew,
 int *nyNew,
 int *nzNew
 )
{
 int nxtmp, nytmp, nztmp, iLevel;

 for (iLevel=0; iLevel<numLevel; iLevel++) {
  nxtmp = *nxNew;
  nytmp = *nyNew;
  nztmp = *nzNew;
  *nxNew = coarsenThis(nxtmp);
  *nyNew = coarsenThis(nytmp);
  *nzNew = coarsenThis(nztmp);
 }


}
```