APBS

3.0.0

Generated by Doxygen 1.9.1

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 Bug List 5

3 Deprecated List 9

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 dependencies . . . 19
7.2 Vcsm class . . . 19
7.2.1 Detailed Description . . . 20
7.2.2 Function Documentation . . . 20
7.2.2.1 Gem_setExternalUpdateFunction() . . . 20
7.2.2.2 Vcsm_ctor() . . . 20
7.2.2.3 Vcsm_ctor2() . . . 21
7.2.2.4 Vcsm_dtor() . . . 22
7.2.2.5 Vcsm_dtor2() . . . 22
7.2.2.6 Vcsm_getAtom() . . . 22
7.2.2.7 Vcsm_getAtomIndex() . . . 23
7.2.2.8 Vcsm_getNumberAtoms() . . . 23
7.2.2.9 Vcsm_getNumberSimplices() . . . 23
7.2.2.10 Vcsm_getSimplex() . . . 24
7.2.2.11 Vcsm_getSimplexIndex() . . . 24
7.2.2.12 Vcsm_getValist() . . . 25
7.2.2.13 Vcsm_init() . . . 25
7.2.2.14 Vcsm_memChk() . . . 26
7.2.2.15 Vcsm_update() . . . 26
7.3 Vfetk class . . . 26
7.3.1 Detailed Description . . . 30

7.3.2 Enumeration Type Documentation . . . 30
7.3.2.1 eVfetk_GuessType . . . 30
7.3.2.2 eVfetk_LsolvType . . . 30
7.3.2.3 eVfetk_MeshLoad . . . 31
7.3.2.4 eVfetk_NsolvType . . . 31
7.3.2.5 eVfetk_PrecType . . . 31
7.3.3 Function Documentation . . . 32
7.3.3.1 Bmat_printHB() . . . 32
7.3.3.2 Vfetk_ctor() . . . 32
7.3.3.3 Vfetk_ctor2() . . . 33
7.3.3.4 Vfetk_dqmEnergy() . . . 33
7.3.3.5 Vfetk_dtor() . . . 34
7.3.3.6 Vfetk_dtor2() . . . 34
7.3.3.7 Vfetk_dumpLocalVar() . . . 35
7.3.3.8 Vfetk_energy() . . . 35
7.3.3.9 Vfetk_externalUpdateFunction() . . . 35
7.3.3.10 Vfetk_fillArray() . . . 36
7.3.3.11 Vfetk_genCube() . . . 36
7.3.3.12 Vfetk_getAM() . . . 37
7.3.3.13 Vfetk_getAtomColor() . . . 37
7.3.3.14 Vfetk_getGem() . . . 38
7.3.3.15 Vfetk_getSolution() . . . 38
7.3.3.16 Vfetk_getVcsm() . . . 39
7.3.3.17 Vfetk_getVpbe() . . . 39
7.3.3.18 Vfetk_loadGem() . . . 39
7.3.3.19 Vfetk_loadMesh() . . . 40
7.3.3.20 Vfetk_memChk() . . . 40
7.3.3.21 Vfetk_PDE_bisectEdge() . . . 41
7.3.3.22 Vfetk_PDE_ctor() . . . 41
7.3.3.23 Vfetk_PDE_ctor2() . . . 42
7.3.3.24 Vfetk_PDE_delta() . . . 42
7.3.3.25 Vfetk_PDE_DFu_wv() . . . 43
7.3.3.26 Vfetk_PDE_dtor() . . . 43
7.3.3.27 Vfetk_PDE_dtor2() . . . 44
7.3.3.28 Vfetk_PDE_Fu() . . . 44
7.3.3.29 Vfetk_PDE_Fu_v() . . . 45
7.3.3.30 Vfetk_PDE_initAssemble() . . . 45
7.3.3.31 Vfetk_PDE_initElement() . . . 46
7.3.3.32 Vfetk_PDE_initFace() . . . 46

7.3.3.33 Vfetk_PDE_initPoint() . . . 47
7.3.3.34 Vfetk_PDE_Ju() . . . 47
7.3.3.35 Vfetk_PDE_mapBoundary() . . . 48
7.3.3.36 Vfetk_PDE_markSimplex() . . . 48
7.3.3.37 Vfetk_PDE_oneChart() . . . 49
7.3.3.38 Vfetk_PDE_simplexBasisForm() . . . 50
7.3.3.39 Vfetk_PDE_simplexBasisInit() . . . 50
7.3.3.40 Vfetk_PDE_u_D() . . . 52
7.3.3.41 Vfetk_PDE_u_T() . . . 52
7.3.3.42 Vfetk_qfEnergy() . . . 53
7.3.3.43 Vfetk_readMesh() . . . 54
7.3.3.44 Vfetk_setAtomColors() . . . 54
7.3.3.45 Vfetk_setParameters() . . . 55
7.3.3.46 Vfetk_write() . . . 55
7.4 Vpee class . . . 56
7.4.1 Detailed Description . . . 57
7.4.2 Function Documentation . . . 57
7.4.2.1 Vpee_ctor() . . . 57
7.4.2.2 Vpee_ctor2() . . . 58
7.4.2.3 Vpee_dtor() . . . 58
7.4.2.4 Vpee_dtor2() . . . 59
7.4.2.5 Vpee_markRefine() . . . 59
7.4.2.6 Vpee_numSS() . . . 60
7.5 APOLparm class . . . 60
7.5.1 Detailed Description . . . 61
7.5.2 Enumeration Type Documentation . . . 62
7.5.2.1 eAPOLparm_calcEnergy . . . 62
7.5.2.2 eAPOLparm_calcForce . . . 62
7.5.2.3 eAPOLparm_doCalc . . . 62
7.5.3 Function Documentation . . . 62
7.5.3.1 APOLparm_check() . . . 62
7.5.3.2 APOLparm_copy() . . . 63
7.5.3.3 APOLparm_ctor() . . . 63
7.5.3.4 APOLparm_ctor2() . . . 63
7.5.3.5 APOLparm_dtor() . . . 64
7.5.3.6 APOLparm_dtor2() . . . 64
7.6 BEMparm class . . . 64
7.6.1 Detailed Description . . . 65
7.6.2 Typedef Documentation . . . 65

7.6.2.1 BEMparm . . . 65
7.6.3 Enumeration Type Documentation . . . 66
7.6.3.1 eBEMparm_CalcType . . . 66
7.6.4 Function Documentation . . . 66
7.6.4.1 BEMparm_check() . . . 66
7.6.4.2 BEMparm_ctor() . . . 66
7.6.4.3 BEMparm_ctor2() . . . 67
7.6.4.4 BEMparm_dtor() . . . 67
7.6.4.5 BEMparm_dtor2() . . . 68
7.6.4.6 BEMparm_parseToken() . . . 68
7.7 FEMparm class . . . 68
7.7.1 Detailed Description . . . 69
7.7.2 Typedef Documentation . . . 69
7.7.2.1 FEMparm_EtolType . . . 70
7.7.3 Enumeration Type Documentation . . . 70
7.7.3.1 eFEMparm_CalcType . . . 70
7.7.3.2 eFEMparm_EstType . . . 70
7.7.3.3 eFEMparm_EtolType . . . 70
7.7.4 Function Documentation . . . 71
7.7.4.1 FEMparm_check() . . . 71
7.7.4.2 FEMparm_copy() . . . 71
7.7.4.3 FEMparm_ctor() . . . 72
7.7.4.4 FEMparm_ctor2() . . . 72
7.7.4.5 FEMparm_dtor() . . . 72
7.7.4.6 FEMparm_dtor2() . . . 73
7.8 GEOFLOWparm class . . . 73
7.8.1 Detailed Description . . . 74
7.8.2 Typedef Documentation . . . 74
7.8.2.1 GEOFLOWparm . . . 74
7.8.3 Enumeration Type Documentation . . . 74
7.8.3.1 eGEOFLOWparm_CalcType . . . 74
7.8.4 Function Documentation . . . 75
7.8.4.1 GEOFLOWparm_check() . . . 75
7.8.4.2 GEOFLOWparm_copy() . . . 75
7.8.4.3 GEOFLOWparm_ctor() . . . 75
7.8.4.4 GEOFLOWparm_ctor2() . . . 76
7.8.4.5 GEOFLOWparm_dtor() . . . 76
7.8.4.6 GEOFLOWparm_dtor2() . . . 77
7.8.4.7 GEOFLOWparm_parseToken() . . . 77

7.9 MGparm class . . . 77
7.9.1 Detailed Description . . . 79
7.9.2 Enumeration Type Documentation . . . 79
7.9.2.1 eMGparm_CalcType . . . 79
7.9.2.2 eMGparm_CentMeth . . . 79
7.9.3 Function Documentation . . . 80
7.9.3.1 APOLparm_parseToken() . . . 80
7.9.3.2 FEMparm_parseToken() . . . 80
7.9.3.3 MGparm_check() . . . 81
7.9.3.4 MGparm_copy() . . . 81
7.9.3.5 MGparm_ctor() . . . 81
7.9.3.6 MGparm_ctor2() . . . 82
7.9.3.7 MGparm_dtor() . . . 82
7.9.3.8 MGparm_dtor2() . . . 83
7.9.3.9 MGparm_getCenterX() . . . 83
7.9.3.10 MGparm_getCenterY() . . . 83
7.9.3.11 MGparm_getCenterZ() . . . 84
7.9.3.12 MGparm_getHx() . . . 84
7.9.3.13 MGparm_getHy() . . . 84
7.9.3.14 MGparm_getHz() . . . 85
7.9.3.15 MGparm_getNx() . . . 85
7.9.3.16 MGparm_getNy() . . . 86
7.9.3.17 MGparm_getNz() . . . 86
7.9.3.18 MGparm_parseToken() . . . 86
7.9.3.19 MGparm_setCenterX() . . . 87
7.9.3.20 MGparm_setCenterY() . . . 87
7.9.3.21 MGparm_setCenterZ() . . . 88
7.10 NOsh class . . . 88
7.10.1 Detailed Description . . . 90
7.10.2 Enumeration Type Documentation . . . 91
7.10.2.1 eNOsh_CalcType . . . 91
7.10.2.2 eNOsh_MolFormat . . . 91
7.10.2.3 eNOsh_ParmFormat . . . 91
7.10.2.4 eNOsh_PrintType . . . 91
7.10.3 Function Documentation . . . 92
7.10.3.1 NOsh_apol2calc() . . . 92
7.10.3.2 NOsh_calc_copy() . . . 92
7.10.3.3 NOsh_calc_ctor() . . . 93
7.10.3.4 NOsh_calc_dtor() . . . 93

7.10.3.5 NOsh_ctor() . . . 93
7.10.3.6 NOsh_ctor2() . . . 94
7.10.3.7 NOsh_dtor() . . . 94
7.10.3.8 NOsh_dtor2() . . . 94
7.10.3.9 NOsh_elec2calc() . . . 95
7.10.3.10 NOsh_elecname() . . . 95
7.10.3.11 NOsh_getCalc() . . . 96
7.10.3.12 NOsh_getChargefmt() . . . 96
7.10.3.13 NOsh_getChargepath() . . . 97
7.10.3.14 NOsh_getDielfmt() . . . 97
7.10.3.15 NOsh_getDielXpath() . . . 97
7.10.3.16 NOsh_getDielYpath() . . . 98
7.10.3.17 NOsh_getDielZpath() . . . 98
7.10.3.18 NOsh_getKappafmt() . . . 99
7.10.3.19 NOsh_getKappapath() . . . 99
7.10.3.20 NOsh_getMolpath() . . . 99
7.10.3.21 NOsh_getPotfmt() . . . 100
7.10.3.22 NOsh_getPotpath() . . . 100
7.10.3.23 NOsh_parseInput() . . . 101
7.10.3.24 NOsh_parseInputFile() . . . 101
7.10.3.25 NOsh_printCalc() . . . 102
7.10.3.26 NOsh_printNarg() . . . 102
7.10.3.27 NOsh_printOp() . . . 103
7.10.3.28 NOsh_printWhat() . . . 104
7.10.3.29 NOsh_setupApolCalc() . . . 104
7.10.3.30 NOsh_setupElecCalc() . . . 105
7.11 PBAMparm class . . . 105
7.11.1 Detailed Description . . . 107
7.11.2 Typedef Documentation . . . 107
7.11.2.1 PBAMparm . . . 107
7.11.3 Enumeration Type Documentation . . . 107
7.11.3.1 ePBAMparm_CalcType . . . 107
7.11.4 Function Documentation . . . 108
7.11.4.1 PBAMparm_check() . . . 108
7.11.4.2 PBAMparm_copy() . . . 108
7.11.4.3 PBAMparm_ctor() . . . 108
7.11.4.4 PBAMparm_ctor2() . . . 109
7.11.4.5 PBAMparm_dtor() . . . 109
7.11.4.6 PBAMparm_dtor2() . . . 110

7.11.4.7 PBAMparm_parse3Dmap() . . . 110
7.11.4.8 PBAMparm_parseDiff() . . . 110
7.11.4.9 PBAMparm_parseDX() . . . 111
7.11.4.10 PBAMparm_parseGrid2D() . . . 111
7.11.4.11 PBAMparm_parseGridPts() . . . 111
7.11.4.12 PBAMparm_parsePBCS() . . . 112
7.11.4.13 PBAMparm_parseRandorient() . . . 112
7.11.4.14 PBAMparm_parseRunName() . . . 112
7.11.4.15 PBAMparm_parseRunType() . . . 113
7.11.4.16 PBAMparm_parseSalt() . . . 113
7.11.4.17 PBAMparm_parseTermcombine() . . . 113
7.11.4.18 PBAMparm_parseToken() . . . 114
7.11.4.19 PBAMparm_parseUnits() . . . 114
7.11.4.20 PBAMparm_parseXYZ() . . . 114
7.12 PBEparm class . . . 115
7.12.1 Detailed Description . . . 116
7.12.2 Enumeration Type Documentation . . . 116
7.12.2.1 ePBEparm_calcEnergy . . . 116
7.12.2.2 ePBEparm_calcForce . . . 116
7.12.3 Function Documentation . . . 117
7.12.3.1 PBEparm_check() . . . 117
7.12.3.2 PBEparm_copy() . . . 117
7.12.3.3 PBEparm_ctor() . . . 118
7.12.3.4 PBEparm_ctor2() . . . 118
7.12.3.5 PBEparm_dtor() . . . 118
7.12.3.6 PBEparm_dtor2() . . . 119
7.12.3.7 PBEparm_getIonCharge() . . . 119
7.12.3.8 PBEparm_getIonConc() . . . 119
7.12.3.9 PBEparm_getIonRadius() . . . 120
7.12.3.10 PBEparm_parseToken() . . . 120
7.13 PBSAMparm class . . . 121
7.13.1 Detailed Description . . . 122
7.13.2 Typedef Documentation . . . 122
7.13.2.1 PBSAMparm . . . 122
7.13.3 Enumeration Type Documentation . . . 122
7.13.3.1 ePBSAMparm_CalcType . . . 122
7.13.4 Function Documentation . . . 122
7.13.4.1 PBSAMparm_check() . . . 123
7.13.4.2 PBSAMparm_copy() . . . 123

7.13.4.3 PBSAMparm_ctor() . . . 123
7.13.4.4 PBSAMparm_ctor2() . . . 124
7.13.4.5 PBSAMparm_dtor() . . . 124
7.13.4.6 PBSAMparm_dtor2() . . . 124
7.13.4.7 PBSAMparm_parseExp() . . . 125
7.13.4.8 PBSAMparm_parseImat() . . . 125
7.13.4.9 PBSAMparm_parseMSMS() . . . 125
7.13.4.10 PBSAMparm_parseSurf() . . . 126
7.13.4.11 PBSAMparm_parseToken() . . . 126
7.13.4.12 PBSAMparm_parseTolsp() . . . 127
7.14 Vacc class . . . 127
7.14.1 Detailed Description . . . 129
7.14.2 Function Documentation . . . 129
7.14.2.1 Vacc_atomdSASA() . . . 129
7.14.2.2 Vacc_atomdSAV() . . . 130
7.14.2.3 Vacc_atomSASA() . . . 130
7.14.2.4 Vacc_atomSASPoints() . . . 131
7.14.2.5 Vacc_atomSurf() . . . 131
7.14.2.6 Vacc_ctor() . . . 132
7.14.2.7 Vacc_ctor2() . . . 132
7.14.2.8 Vacc_dtor() . . . 133
7.14.2.9 Vacc_dtor2() . . . 133
7.14.2.10 Vacc_fastMolAcc() . . . 134
7.14.2.11 Vacc_ivdwAcc() . . . 134
7.14.2.12 Vacc_memChk() . . . 135
7.14.2.13 Vacc_molAcc() . . . 135
7.14.2.14 Vacc_SASA() . . . 136
7.14.2.15 Vacc_splineAcc() . . . 136
7.14.2.16 Vacc_splineAccAtom() . . . 137
7.14.2.17 Vacc_splineAccGrad() . . . 137
7.14.2.18 Vacc_splineAccGradAtomNorm() . . . 138
7.14.2.19 Vacc_splineAccGradAtomNorm3() . . . 138
7.14.2.20 Vacc_splineAccGradAtomNorm4() . . . 139
7.14.2.21 Vacc_splineAccGradAtomUnnorm() . . . 139
7.14.2.22 Vacc_totalAtomdSASA() . . . 140
7.14.2.23 Vacc_totalAtomdSAV() . . . 140
7.14.2.24 Vacc_totalSASA() . . . 141
7.14.2.25 Vacc_totalSAV() . . . 141
7.14.2.26 Vacc_vdwAcc() . . . 142

7.14.2.27 Vacc_wcaEnergy() . . . 142
7.14.2.28 Vacc_wcaEnergyAtom() . . . 143
7.14.2.29 Vacc_wcaForceAtom() . . . 143
7.14.2.30 VaccSurf_ctor() . . . 144
7.14.2.31 VaccSurf_ctor2() . . . 144
7.14.2.32 VaccSurf_dtor() . . . 145
7.14.2.33 VaccSurf_dtor2() . . . 145
7.14.2.34 VaccSurf_refSphere() . . . 146
7.15 Valist class . . . 146
7.15.1 Detailed Description . . . 147
7.15.2 Function Documentation . . . 147
7.15.2.1 Valist_ctor() . . . 147
7.15.2.2 Valist_ctor2() . . . 148
7.15.2.3 Valist_dtor() . . . 148
7.15.2.4 Valist_dtor2() . . . 148
7.15.2.5 Valist_getAtom() . . . 149
7.15.2.6 Valist_getAtomList() . . . 149
7.15.2.7 Valist_getCenterX() . . . 149
7.15.2.8 Valist_getCenterY() . . . 150
7.15.2.9 Valist_getCenterZ() . . . 150
7.15.2.10 Valist_getNumberAtoms() . . . 151
7.15.2.11 Valist_getStatistics() . . . 151
7.15.2.12 Valist_memChk() . . . 151
7.15.2.13 Valist_readPDB() . . . 152
7.15.2.14 Valist_readPQR() . . . 152
7.15.2.15 Valist_readXML() . . . 153
7.16 Vatom class . . . 153
7.16.1 Detailed Description . . . 155
7.16.2 Macro Definition Documentation . . . 155
7.16.2.1 VMAX_RECLEN . . . 155
7.16.3 Function Documentation . . . 155
7.16.3.1 Vatom_copyFrom() . . . 155
7.16.3.2 Vatom_copyTo() . . . 156
7.16.3.3 Vatom_ctor() . . . 156
7.16.3.4 Vatom_ctor2() . . . 156
7.16.3.5 Vatom_dtor() . . . 157
7.16.3.6 Vatom_dtor2() . . . 157
7.16.3.7 Vatom_getAtomID() . . . 157
7.16.3.8 Vatom_getAtomName() . . . 158

7.16.3.9 Vatom_getCharge() . . . 158
7.16.3.10 Vatom_getEpsilon() . . . 158
7.16.3.11 Vatom_getPartID() . . . 159
7.16.3.12 Vatom_getPosition() . . . 159
7.16.3.13 Vatom_getRadius() . . . 159
7.16.3.14 Vatom_getResName() . . . 160
7.16.3.15 Vatom_memChk() . . . 160
7.16.3.16 Vatom_setAtomID() . . . 161
7.16.3.17 Vatom_setAtomName() . . . 161
7.16.3.18 Vatom_setCharge() . . . 161
7.16.3.19 Vatom_setEpsilon() . . . 162
7.16.3.20 Vatom_setPartID() . . . 162
7.16.3.21 Vatom_setPosition() . . . 162
7.16.3.22 Vatom_setRadius() . . . 163
7.16.3.23 Vatom_setResName() . . . 163
7.17 Vcap class . . . 163
7.17.1 Detailed Description . . . 164
7.17.2 Function Documentation . . . 164
7.17.2.1 Vcap_cosh() . . . 164
7.17.2.2 Vcap_exp() . . . 165
7.17.2.3 Vcap_sinh() . . . 165
7.18 Vclist class . . . 166
7.18.1 Detailed Description . . . 167
7.18.2 Enumeration Type Documentation . . . 167
7.18.2.1 eVclist_DomainMode . . . 167
7.18.3 Function Documentation . . . 168
7.18.3.1 Vclist_ctor() . . . 168
7.18.3.2 Vclist_ctor2() . . . 168
7.18.3.3 Vclist_dtor() . . . 169
7.18.3.4 Vclist_dtor2() . . . 169
7.18.3.5 Vclist_getCell() . . . 170
7.18.3.6 Vclist_maxRadius() . . . 170
7.18.3.7 Vclist_memChk() . . . 170
7.18.3.8 VclistCell_ctor() . . . 171
7.18.3.9 VclistCell_ctor2() . . . 171
7.18.3.10 VclistCell_dtor() . . . 172
7.18.3.11 VclistCell_dtor2() . . . 172
7.19 Vgreen class . . . 172
7.19.1 Detailed Description . . . 173

7.19.2 Function Documentation . . . 174
7.19.2.1 Vgreen_coulomb() . . . 174
7.19.2.2 Vgreen_coulomb_direct() . . . 175
7.19.2.3 Vgreen_coulombD() . . . 176
7.19.2.4 Vgreen_coulombD_direct() . . . 176
7.19.2.5 Vgreen_ctor() . . . 177
7.19.2.6 Vgreen_ctor2() . . . 178
7.19.2.7 Vgreen_dtor() . . . 178
7.19.2.8 Vgreen_dtor2() . . . 178
7.19.2.9 Vgreen_getValist() . . . 179
7.19.2.10 Vgreen_helmholtz() . . . 179
7.19.2.11 Vgreen_helmholtzD() . . . 180
7.19.2.12 Vgreen_memChk() . . . 181
7.20 Vhal class . . . 181
7.20.1 Detailed Description . . . 184
7.20.2 Macro Definition Documentation . . . 184
7.20.2.1 VAPBS_BACK . . . 184
7.20.2.2 VAPBS_DOWN . . . 184
7.20.2.3 VAPBS_FRONT . . . 184
7.20.2.4 VAPBS_LEFT . . . 185
7.20.2.5 VAPBS_RIGHT . . . 185
7.20.2.6 VAPBS_UP . . . 185
7.20.2.7 VEMBED . . . 185
7.20.2.8 VFLOOR . . . 186
7.20.3 Enumeration Type Documentation . . . 186
7.20.3.1 eVbcfl . . . 186
7.20.3.2 eVchrg_Meth . . . 186
7.20.3.3 eVchrg_Src . . . 187
7.20.3.4 eVdata_Format . . . 187
7.20.3.5 eVdata_Type . . . 187
7.20.3.6 eVhal_IPKEYType . . . 188
7.20.3.7 eVhal_PBEType . . . 188
7.20.3.8 eVoutput_Format . . . 189
7.20.3.9 eVrc_Codes . . . 189
7.20.3.10 eVsol_Meth . . . 189
7.20.3.11 eVsurf_Meth . . . 189
7.21 Matrix wrapper class . . . 190
7.21.1 Detailed Description . . . 190
7.22 Vparam class . . . 190

7.22.1 Detailed Description . . . 192
7.22.2 Function Documentation . . . 192
7.22.2.1 readFlatFileLine() . . . 192
7.22.2.2 readXMLFileAtom() . . . 193
7.22.2.3 Vparam_AtomData_copyFrom() . . . 193
7.22.2.4 Vparam_AtomData_copyTo() . . . 193
7.22.2.5 Vparam_AtomData_ctor() . . . 194
7.22.2.6 Vparam_AtomData_ctor2() . . . 194
7.22.2.7 Vparam_AtomData_dtor() . . . 194
7.22.2.8 Vparam_AtomData_dtor2() . . . 195
7.22.2.9 Vparam_ctor() . . . 195
7.22.2.10 Vparam_ctor2() . . . 195
7.22.2.11 Vparam_dtor() . . . 196
7.22.2.12 Vparam_dtor2() . . . 196
7.22.2.13 Vparam_getAtomData() . . . 196
7.22.2.14 Vparam_getResData() . . . 197
7.22.2.15 Vparam_memChk() . . . 198
7.22.2.16 Vparam_readFlatFile() . . . 198
7.22.2.17 Vparam_readXMLFile() . . . 199
7.22.2.18 Vparam_ResData_copyTo() . . . 199
7.22.2.19 Vparam_ResData_ctor() . . . 200
7.22.2.20 Vparam_ResData_ctor2() . . . 200
7.22.2.21 Vparam_ResData_dtor() . . . 200
7.22.2.22 Vparam_ResData_dtor2() . . . 201
7.23 Vpbe class . . . 201
7.23.1 Detailed Description . . . 203
7.23.2 Function Documentation . . . 203
7.23.2.1 Vpbe_ctor() . . . 203
7.23.2.2 Vpbe_ctor2() . . . 204
7.23.2.3 Vpbe_dtor() . . . 206
7.23.2.4 Vpbe_dtor2() . . . 206
7.23.2.5 Vpbe_getBulkIonicStrength() . . . 206
7.23.2.6 Vpbe_getCoulombEnergy1() . . . 207
7.23.2.7 Vpbe_getDeblen() . . . 207
7.23.2.8 Vpbe_getGamma() . . . 208
7.23.2.9 Vpbe_getIons() . . . 208
7.23.2.10 Vpbe_getLmem() . . . 208
7.23.2.11 Vpbe_getMaxIonRadius() . . . 209
7.23.2.12 Vpbe_getmembraneDiel() . . . 209

7.23.2.13 Vpbe_getmemv() . . . 209
7.23.2.14 Vpbe_getSoluteCenter() . . . 210
7.23.2.15 Vpbe_getSoluteCharge() . . . 210
7.23.2.16 Vpbe_getSoluteDiel() . . . 211
7.23.2.17 Vpbe_getSoluteRadius() . . . 211
7.23.2.18 Vpbe_getSoluteXlen() . . . 211
7.23.2.19 Vpbe_getSoluteYlen() . . . 212
7.23.2.20 Vpbe_getSoluteZlen() . . . 212
7.23.2.21 Vpbe_getSolventDiel() . . . 212
7.23.2.22 Vpbe_getSolventRadius() . . . 213
7.23.2.23 Vpbe_getTemperature() . . . 213
7.23.2.24 Vpbe_getVacc() . . . 214
7.23.2.25 Vpbe_getValist() . . . 214
7.23.2.26 Vpbe_getXkappa() . . . 214
7.23.2.27 Vpbe_getZkappa2() . . . 215
7.23.2.28 Vpbe_getZmagic() . . . 215
7.23.2.29 Vpbe_getzmem() . . . 215
7.23.2.30 Vpbe_memChk() . . . 216
7.24 Vstring class . . . 216
7.24.1 Detailed Description . . . 217
7.24.2 Function Documentation . . . 217
7.24.2.1 Vstring_isdigit() . . . 217
7.24.2.2 Vstring_strcasecmp() . . . 217
7.24.2.3 Vstring_wrappedtext() . . . 218
7.25 Vunit class . . . 218
7.25.1 Detailed Description . . . 219
7.26 Vgrid class . . . 219
7.26.1 Detailed Description . . . 221
7.26.2 Function Documentation . . . 221
7.26.2.1 Vgrid_ctor() . . . 221
7.26.2.2 Vgrid_ctor2() . . . 222
7.26.2.3 Vgrid_curvature() . . . 222
7.26.2.4 Vgrid_dtor() . . . 223
7.26.2.5 Vgrid_dtor2() . . . 223
7.26.2.6 Vgrid_gradient() . . . 224
7.26.2.7 Vgrid_integrate() . . . 224
7.26.2.8 Vgrid_memChk() . . . 224
7.26.2.9 Vgrid_normH1() . . . 225
7.26.2.10 Vgrid_normL1() . . . 225

7.26.2.11 Vgrid_normL2() . . . 226
7.26.2.12 Vgrid_normLinf() . . . 226
7.26.2.13 Vgrid_readDX() . . . 226
7.26.2.14 Vgrid_readDXBIN() . . . 227
7.26.2.15 Vgrid_readGZ() . . . 228
7.26.2.16 Vgrid_seminormH1() . . . 228
7.26.2.17 Vgrid_value() . . . 229
7.26.2.18 Vgrid_writeDX() . . . 229
7.26.2.19 Vgrid_writeDXBIN() . . . 230
7.26.2.20 Vgrid_writeUHBD() . . . 230
7.27 Vmgrid class . . . 231
7.27.1 Detailed Description . . . 232
7.27.2 Function Documentation . . . 232
7.27.2.1 Vmgrid_addGrid() . . . 232
7.27.2.2 Vmgrid_ctor() . . . 232
7.27.2.3 Vmgrid_ctor2() . . . 233
7.27.2.4 Vmgrid_curvature() . . . 233
7.27.2.5 Vmgrid_dtor() . . . 234
7.27.2.6 Vmgrid_dtor2() . . . 234
7.27.2.7 Vmgrid_getGridByNum() . . . 234
7.27.2.8 Vmgrid_getGridByPoint() . . . 235
7.27.2.9 Vmgrid_gradient() . . . 235
7.27.2.10 Vmgrid_value() . . . 235
7.28 Vopot class . . . 236
7.28.1 Detailed Description . . . 237
7.28.2 Function Documentation . . . 237
7.28.2.1 Vopot_ctor() . . . 237
7.28.2.2 Vopot_ctor2() . . . 237
7.28.2.3 Vopot_curvature() . . . 238
7.28.2.4 Vopot_dtor() . . . 238
7.28.2.5 Vopot_dtor2() . . . 239
7.28.2.6 Vopot_gradient() . . . 239
7.28.2.7 Vopot_pot() . . . 240
7.29 Vpmg class . . . 240
7.29.1 Detailed Description . . . 243
7.29.2 Function Documentation . . . 243
7.29.2.1 bcolcomp() . . . 243
7.29.2.2 bcolcomp2() . . . 244
7.29.2.3 bcolcomp3() . . . 245

7.29.2.4 bcolcomp4() . . . 245
7.29.2.5 pcolcomp() . . . 246
7.29.2.6 Vpackmg() . . . 247
7.29.2.7 Vpmg_ctor() . . . 248
7.29.2.8 Vpmg_ctor2() . . . 248
7.29.2.9 Vpmg_dbDirectPolForce() . . . 249
7.29.2.10 Vpmg_dbForce() . . . 250
7.29.2.11 Vpmg_dbMutualPolForce() . . . 250
7.29.2.12 Vpmg_dbNLDirectPolForce() . . . 251
7.29.2.13 Vpmg_dbPermanentMultipoleForce() . . . 251
7.29.2.14 Vpmg_dielEnergy() . . . 252
7.29.2.15 Vpmg_dielGradNorm() . . . 252
7.29.2.16 Vpmg_dtor() . . . 253
7.29.2.17 Vpmg_dtor2() . . . 253
7.29.2.18 Vpmg_energy() . . . 253
7.29.2.19 Vpmg_fieldSpline4() . . . 254
7.29.2.20 Vpmg_fillArray() . . . 254
7.29.2.21 Vpmg_fillco() . . . 255
7.29.2.22 Vpmg_force() . . . 256
7.29.2.23 Vpmg_ibDirectPolForce() . . . 257
7.29.2.24 Vpmg_ibForce() . . . 257
7.29.2.25 Vpmg_ibMutualPolForce() . . . 258
7.29.2.26 Vpmg_ibNLDirectPolForce() . . . 258
7.29.2.27 Vpmg_ibPermanentMultipoleForce() . . . 259
7.29.2.28 Vpmg_memChk() . . . 259
7.29.2.29 Vpmg_printColComp() . . . 259
7.29.2.30 Vpmg_qfAtomEnergy() . . . 260
7.29.2.31 Vpmg_qfDirectPolForce() . . . 261
7.29.2.32 Vpmg_qfEnergy() . . . 261
7.29.2.33 Vpmg_qfForce() . . . 262
7.29.2.34 Vpmg_qfMutualPolForce() . . . 263
7.29.2.35 Vpmg_qfNLDirectPolForce() . . . 263
7.29.2.36 Vpmg_qfPermanentMultipoleEnergy() . . . 264
7.29.2.37 Vpmg_qfPermanentMultipoleForce() . . . 264
7.29.2.38 Vpmg_qmEnergy() . . . 264
7.29.2.39 Vpmg_setPart() . . . 265
7.29.2.40 Vpmg_solve() . . . 266
7.29.2.41 Vpmg_solveLaplace() . . . 266
7.29.2.42 Vpmg_unsetPart() . . . 266

7.30 Vpmgp class . . . 267
7.30.1 Detailed Description . . . 268
7.30.2 Function Documentation . . . 268
7.30.2.1 Vpmgp_ctor() . . . 268
7.30.2.2 Vpmgp_ctor2() . . . 268
7.30.2.3 Vpmgp_dtor() . . . 269
7.30.2.4 Vpmgp_dtor2() . . . 269
7.30.2.5 Vpmgp_makeCoarse() . . . 269
7.30.2.6 Vpmgp_size() . . . 270
7.31 C translation of Holst group PMG code . . . 270
7.31.1 Detailed Description . . . 274
7.31.2 Macro Definition Documentation . . . 274
7.31.2.1 HARMO2 . . . 274
7.31.2.2 MAXIONS . . . 275
7.31.3 Function Documentation . . . 276
7.31.3.1 Vaxrand() . . . 276
7.31.3.2 Vazeros() . . . 277
7.31.3.3 VbuildA() . . . 277
7.31.3.4 Vbuildband() . . . 280
7.31.3.5 Vbuildband1_27() . . . 282
7.31.3.6 Vbuildband1_7() . . . 284
7.31.3.7 VbuildG() . . . 284
7.31.3.8 VbuildG_1() . . . 287
7.31.3.9 VbuildG_27() . . . 289
7.31.3.10 VbuildG_7() . . . 292
7.31.3.11 Vbuildgaler0() . . . 295
7.31.3.12 Vbuildops() . . . 296
7.31.3.13 VbuildP() . . . 299
7.31.3.14 Vbuildstr() . . . 301
7.31.3.15 Vc_vec() . . . 302
7.31.3.16 Vc_vecpmg() . . . 302
7.31.3.17 Vc_vecsmpbe() . . . 303
7.31.3.18 Vcghs() . . . 304
7.31.3.19 Vdc_vec() . . . 306
7.31.3.20 Vdpbsl() . . . 307
7.31.3.21 Vextrac() . . . 309
7.31.3.22 VfboundPMG() . . . 310
7.31.3.23 VfboundPMG00() . . . 310
7.31.3.24 Vfmvfas() . . . 311

7.31.3.25 Vfnewton() . . . 314
7.31.3.26 Vgetjac() . . . 317
7.31.3.27 Vgsrb() . . . 318
7.31.3.28 VinterpPMG() . . . 321
7.31.3.29 Vipower() . . . 322
7.31.3.30 Vmatvec() . . . 323
7.31.3.31 Vmgdriv() . . . 326
7.31.3.32 Vmgdriv2() . . . 328
7.31.3.33 Vmgsz() . . . 330
7.31.3.34 Vmresid() . . . 332
7.31.3.35 Vmvcs() . . . 332
7.31.3.36 Vmvfas() . . . 336
7.31.3.37 Vmypdefinitlpbe() . . . 338
7.31.3.38 Vmypdefinitnpbe() . . . 339
7.31.3.39 Vmypdefinitsmpbe() . . . 340
7.31.3.40 Vnewdriv() . . . 340
7.31.3.41 Vnewdriv2() . . . 343
7.31.3.42 Vnewton() . . . 345
7.31.3.43 Vnmatvec() . . . 346
7.31.3.44 Vnmresid() . . . 347
7.31.3.45 Vnsmooth() . . . 348
7.31.3.46 Vpower() . . . 349
7.31.3.47 Vprtmatd() . . . 352
7.31.3.48 Vrestrc() . . . 352
7.31.3.49 Vsmooth() . . . 353
7.31.3.50 Vxaxpy() . . . 356
7.31.3.51 Vxcopy() . . . 356
7.31.3.52 Vxcopy_large() . . . 359
7.31.3.53 Vxcopy_small() . . . 359
7.31.3.54 Vxdot() . . . 360
7.31.3.55 Vxnrm1() . . . 360
7.31.3.56 Vxnrm2() . . . 361
7.31.3.57 Vxscal() . . . 361
7.32 High-level front-end routines . . . 362
7.32.1 Detailed Description . . . 365
7.32.2 Function Documentation . . . 365
7.32.2.1 energyAPOL() . . . 365
7.32.2.2 energyFE() . . . 366
7.32.2.3 energyMG() . . . 367

7.32.2.4 forceAPOL() . . . 367
7.32.2.5 forceMG() . . . 368
7.32.2.6 initAPOL() . . . 369
7.32.2.7 initFE() . . . 370
7.32.2.8 initMG() . . . 371
7.32.2.9 killChargeMaps() . . . 372
7.32.2.10 killDielMaps() . . . 372
7.32.2.11 killEnergy() . . . 373
7.32.2.12 killFE() . . . 373
7.32.2.13 killForce() . . . 373
7.32.2.14 killKappaMaps() . . . 374
7.32.2.15 killMeshes() . . . 374
7.32.2.16 killMG() . . . 374
7.32.2.17 killMolecules() . . . 375
7.32.2.18 killPotMaps() . . . 375
7.32.2.19 loadChargeMaps() . . . 375
7.32.2.20 loadDielMaps() . . . 376
7.32.2.21 loadKappaMaps() . . . 376
7.32.2.22 loadMeshes() . . . 377
7.32.2.23 loadMolecules() . . . 377
7.32.2.24 loadParameter() . . . 378
7.32.2.25 loadPotMaps() . . . 378
7.32.2.26 main() . . . 379
7.32.2.27 partFE() . . . 379
7.32.2.28 postRefineFE() . . . 379
7.32.2.29 preRefineFE() . . . 380
7.32.2.30 printApolEnergy() . . . 381
7.32.2.31 printApolForce() . . . 381
7.32.2.32 printElecEnergy() . . . 382
7.32.2.33 printElecForce() . . . 382
7.32.2.34 printEnergy() . . . 383
7.32.2.35 printFEPARM() . . . 383
7.32.2.36 printForce() . . . 384
7.32.2.37 printMGPARM() . . . 384
7.32.2.38 printPBEPARM() . . . 385
7.32.2.39 returnEnergy() . . . 385
7.32.2.40 setPartMG() . . . 385
7.32.2.41 solveFE() . . . 386
7.32.2.42 solveMG() . . . 387

7.32.2.43 startVio() . . . 387
7.32.2.44 storeAtomEnergy() . . . 387
7.32.2.45 writedataFE() . . . 388
7.32.2.46 writedataFlat() . . . 388
7.32.2.47 writedataMG() . . . 389
7.32.2.48 writedataXML() . . . 390
7.32.2.49 writematMG() . . . 391

8 Data Structure Documentation **393**

8.1 AtomForce Struct Reference . . . 393
8.1.1 Detailed Description . . . 393
8.1.2 Field Documentation . . . 393
8.1.2.1 dbForce . . . 393
8.1.2.2 ibForce . . . 393
8.1.2.3 qfForce . . . 394
8.1.2.4 sasaForce . . . 394
8.1.2.5 savForce . . . 394
8.1.2.6 wcaForce . . . 394
8.2 sAPOLparm Struct Reference . . . 394
8.2.1 Detailed Description . . . 395
8.2.2 Field Documentation . . . 395
8.2.2.1 bconc . . . 395
8.2.2.2 calcenergy . . . 395
8.2.2.3 calcforce . . . 395
8.2.2.4 dpos . . . 396
8.2.2.5 gamma . . . 396
8.2.2.6 grid . . . 396
8.2.2.7 molid . . . 396
8.2.2.8 parsed . . . 396
8.2.2.9 press . . . 396
8.2.2.10 sasa . . . 396
8.2.2.11 sav . . . 396
8.2.2.12 sdens . . . 397
8.2.2.13 setbconc . . . 397
8.2.2.14 setcalcenergy . . . 397
8.2.2.15 setcalcforce . . . 397
8.2.2.16 setdpos . . . 397
8.2.2.17 setgamma . . . 397
8.2.2.18 setgrid . . . 398

8.2.2.19 setmolid . . . 398
8.2.2.20 setpress . . . 398
8.2.2.21 setsdens . . . 398
8.2.2.22 setsrad . . . 398
8.2.2.23 setsrfm . . . 399
8.2.2.24 setswin . . . 399
8.2.2.25 settemp . . . 399
8.2.2.26 setwat . . . 399
8.2.2.27 srad . . . 399
8.2.2.28 srfm . . . 399
8.2.2.29 swin . . . 400
8.2.2.30 temp . . . 400
8.2.2.31 totForce . . . 400
8.2.2.32 watepsilon . . . 400
8.2.2.33 watsigma . . . 400
8.2.2.34 wcaEnergy . . . 400
8.3 sBEMparm Struct Reference . . . 400
8.3.1 Detailed Description . . . 401
8.3.2 Field Documentation . . . 401
8.3.2.1 chgs . . . 401
8.3.2.2 mac . . . 401
8.3.2.3 mesh . . . 401
8.3.2.4 nonlintype . . . 402
8.3.2.5 outdata . . . 402
8.3.2.6 parsed . . . 402
8.3.2.7 setmac . . . 402
8.3.2.8 setmesh . . . 402
8.3.2.9 setnonlintype . . . 402
8.3.2.10 setoutdata . . . 403
8.3.2.11 settree_n0 . . . 403
8.3.2.12 settree_order . . . 403
8.3.2.13 tree_n0 . . . 403
8.3.2.14 tree_order . . . 403
8.3.2.15 type . . . 403
8.4 sFEMparm Struct Reference . . . 404
8.4.1 Detailed Description . . . 404
8.4.2 Field Documentation . . . 404
8.4.2.1 akeyPRE . . . 404
8.4.2.2 akeySOLVE . . . 405

8.4.2.3 ekey . . . 405
8.4.2.4 etol . . . 405
8.4.2.5 glen . . . 405
8.4.2.6 maxsolve . . . 405
8.4.2.7 maxvert . . . 405
8.4.2.8 meshID . . . 405
8.4.2.9 parsed . . . 405
8.4.2.10 pkey . . . 406
8.4.2.11 setakeyPRE . . . 406
8.4.2.12 setakeySOLVE . . . 406
8.4.2.13 setekey . . . 406
8.4.2.14 setetol . . . 406
8.4.2.15 setglen . . . 406
8.4.2.16 setmaxsolve . . . 406
8.4.2.17 setmaxvert . . . 406
8.4.2.18 settargetNum . . . 407
8.4.2.19 settargetRes . . . 407
8.4.2.20 settype . . . 407
8.4.2.21 targetNum . . . 407
8.4.2.22 targetRes . . . 407
8.4.2.23 type . . . 407
8.4.2.24 useMesh . . . 407
8.5 sGEOFLOWparm Struct Reference . . . 408
8.5.1 Detailed Description . . . 408
8.5.2 Field Documentation . . . 408
8.5.2.1 etol . . . 408
8.5.2.2 parsed . . . 408
8.5.2.3 type . . . 408
8.6 sMGparm Struct Reference . . . 409
8.6.1 Detailed Description . . . 410
8.6.2 Field Documentation . . . 410
8.6.2.1 async . . . 410
8.6.2.2 ccenter . . . 410
8.6.2.3 ccentmol . . . 410
8.6.2.4 ccmeth . . . 410
8.6.2.5 center . . . 411
8.6.2.6 centmol . . . 411
8.6.2.7 cglen . . . 411
8.6.2.8 chgm . . . 411

8.6.2.9 chgs . . . 411
8.6.2.10 cmeth . . . 411
8.6.2.11 dime . . . 411
8.6.2.12 etol . . . 412
8.6.2.13 fcenter . . . 412
8.6.2.14 fcentmol . . . 412
8.6.2.15 fcmeth . . . 412
8.6.2.16 fglen . . . 412
8.6.2.17 glen . . . 412
8.6.2.18 grid . . . 412
8.6.2.19 method . . . 412
8.6.2.20 nlev . . . 413
8.6.2.21 nonlintype . . . 413
8.6.2.22 ofrac . . . 413
8.6.2.23 parsed . . . 413
8.6.2.24 partDisjCenter . . . 413
8.6.2.25 partDisjLength . . . 413
8.6.2.26 partDisjOwnSide . . . 413
8.6.2.27 pdime . . . 414
8.6.2.28 proc_rank . . . 414
8.6.2.29 proc_size . . . 414
8.6.2.30 setasync . . . 414
8.6.2.31 setcgcent . . . 414
8.6.2.32 setcglen . . . 414
8.6.2.33 setchgm . . . 415
8.6.2.34 setdime . . . 415
8.6.2.35 setetol . . . 415
8.6.2.36 setfgcent . . . 415
8.6.2.37 setfglen . . . 415
8.6.2.38 setgcent . . . 416
8.6.2.39 setglen . . . 416
8.6.2.40 setgrid . . . 416
8.6.2.41 setmethod . . . 416
8.6.2.42 setnlev . . . 416
8.6.2.43 setnonlintype . . . 417
8.6.2.44 setofrac . . . 417
8.6.2.45 setpdime . . . 417
8.6.2.46 setrank . . . 417
8.6.2.47 setsize . . . 417

8.6.2.48 setUseAqua . . . 418
8.6.2.49 type . . . 418
8.6.2.50 useAqua . . . 418
8.7 sNOsh Struct Reference . . . 418
8.7.1 Detailed Description . . . 419
8.7.2 Field Documentation . . . 419
8.7.2.1 alist . . . 419
8.7.2.2 apol . . . 419
8.7.2.3 apol2calc . . . 420
8.7.2.4 apolname . . . 420
8.7.2.5 bogus . . . 420
8.7.2.6 calc . . . 420
8.7.2.7 chargefmt . . . 420
8.7.2.8 chargepath . . . 420
8.7.2.9 dielfmt . . . 420
8.7.2.10 dielXpath . . . 420
8.7.2.11 dielYpath . . . 421
8.7.2.12 dielZpath . . . 421
8.7.2.13 elec . . . 421
8.7.2.14 elec2calc . . . 421
8.7.2.15 elecname . . . 421
8.7.2.16 gotparm . . . 421
8.7.2.17 ispara . . . 421
8.7.2.18 kappafmt . . . 421
8.7.2.19 kappapath . . . 422
8.7.2.20 meshfmt . . . 422
8.7.2.21 meshpath . . . 422
8.7.2.22 molfmt . . . 422
8.7.2.23 molpath . . . 422
8.7.2.24 napol . . . 422
8.7.2.25 ncalc . . . 422
8.7.2.26 ncharge . . . 423
8.7.2.27 ndiel . . . 423
8.7.2.28 nelec . . . 423
8.7.2.29 nkappa . . . 423
8.7.2.30 nmesh . . . 423
8.7.2.31 nmol . . . 423
8.7.2.32 npot . . . 423
8.7.2.33 nprint . . . 423

8.7.2.34 parmfmt . . . 424
8.7.2.35 parmpath . . . 424
8.7.2.36 parsed . . . 424
8.7.2.37 potfmt . . . 424
8.7.2.38 potpath . . . 424
8.7.2.39 printcalc . . . 424
8.7.2.40 printnarg . . . 424
8.7.2.41 printop . . . 424
8.7.2.42 printwhat . . . 425
8.7.2.43 proc_rank . . . 425
8.7.2.44 proc_size . . . 425
8.8 sNOsh_calc Struct Reference . . . 425
8.8.1 Detailed Description . . . 425
8.8.2 Field Documentation . . . 426
8.8.2.1 apolparm . . . 426
8.8.2.2 bemparm . . . 426
8.8.2.3 calctype . . . 426
8.8.2.4 femparm . . . 426
8.8.2.5 geoflowparm . . . 426
8.8.2.6 mgparm . . . 426
8.8.2.7 pbamparm . . . 426
8.8.2.8 pbeparm . . . 427
8.8.2.9 pbsamparm . . . 427
8.9 sPBAMparm Struct Reference . . . 427
8.9.1 Detailed Description . . . 428
8.9.2 Field Documentation . . . 428
8.9.2.1 parsed . . . 428
8.9.2.2 type . . . 428
8.10 sPBEparm Struct Reference . . . 428
8.10.1 Detailed Description . . . 430
8.10.2 Field Documentation . . . 430
8.10.2.1 bcfl . . . 430
8.10.2.2 calcenergy . . . 430
8.10.2.3 calcforce . . . 430
8.10.2.4 chargeMapID . . . 431
8.10.2.5 dielMapID . . . 431
8.10.2.6 ionc . . . 431
8.10.2.7 ionq . . . 431
8.10.2.8 ionr . . . 431

8.10.2.9 kappaMapID 431
8.10.2.10 Lmem 431
8.10.2.11 mdie 431
8.10.2.12 memv 432
8.10.2.13 molid 432
8.10.2.14 nion 432
8.10.2.15 numwrite 432
8.10.2.16 parsed 432
8.10.2.17 pbetype 432
8.10.2.18 pdie 432
8.10.2.19 potMapID 432
8.10.2.20 sdens 433
8.10.2.21 sdie 433
8.10.2.22 setbcfl 433
8.10.2.23 setcalcenergy 433
8.10.2.24 setcalcforce 433
8.10.2.25 setion 433
8.10.2.26 setLmem 434
8.10.2.27 setmdie 434
8.10.2.28 setmemv 434
8.10.2.29 setmolid 434
8.10.2.30 setnion 434
8.10.2.31 setpbetype 434
8.10.2.32 setpdie 435
8.10.2.33 setsdens 435
8.10.2.34 setsdie 435
8.10.2.35 setsmsize 435
8.10.2.36 setsmvolume 435
8.10.2.37 setsrad 436
8.10.2.38 setsrfm 436
8.10.2.39 setswin 436
8.10.2.40 settemp 436
8.10.2.41 setwritemat 436
8.10.2.42 setzmem 437
8.10.2.43 smsize 437
8.10.2.44 smvolume 437
8.10.2.45 srad 437
8.10.2.46 srfm 437
8.10.2.47 swin 437

8.10.2.48 temp . . . 437
8.10.2.49 useChargeMap . . . 437
8.10.2.50 useDielMap . . . 438
8.10.2.51 useKappaMap . . . 438
8.10.2.52 usePotMap . . . 438
8.10.2.53 writefmt . . . 438
8.10.2.54 writemat . . . 438
8.10.2.55 writematflag . . . 438
8.10.2.56 writematstem . . . 438
8.10.2.57 writestem . . . 439
8.10.2.58 writetype . . . 439
8.10.2.59 zmem . . . 439
8.11 sPBSAMparm Struct Reference . . . 439
8.11.1 Detailed Description . . . 439
8.11.2 Field Documentation . . . 440
8.11.2.1 parsed . . . 440
8.11.2.2 type . . . 440
8.12 sVacc Struct Reference . . . 440
8.12.1 Detailed Description . . . 440
8.12.2 Field Documentation . . . 440
8.12.2.1 acc . . . 441
8.12.2.2 alist . . . 441
8.12.2.3 atomFlags . . . 441
8.12.2.4 clist . . . 441
8.12.2.5 mem . . . 441
8.12.2.6 refSphere . . . 441
8.12.2.7 surf . . . 441
8.12.2.8 surf_density . . . 441
8.13 sVaccSurf Struct Reference . . . 442
8.13.1 Detailed Description . . . 442
8.13.2 Field Documentation . . . 442
8.13.2.1 area . . . 442
8.13.2.2 bpts . . . 442
8.13.2.3 mem . . . 442
8.13.2.4 npts . . . 443
8.13.2.5 probe_radius . . . 443
8.13.2.6 xpts . . . 443
8.13.2.7 ypts . . . 443
8.13.2.8 zpts . . . 443

8.14 sValist Struct Reference . . . 443
8.14.1 Detailed Description . . . 444
8.14.2 Field Documentation . . . 444
8.14.2.1 atoms . . . 444
8.14.2.2 center . . . 444
8.14.2.3 charge . . . 444
8.14.2.4 maxcrd . . . 444
8.14.2.5 maxrad . . . 444
8.14.2.6 mincrd . . . 444
8.14.2.7 number . . . 445
8.14.2.8 vmem . . . 445
8.15 sVatom Struct Reference . . . 445
8.15.1 Detailed Description . . . 445
8.15.2 Field Documentation . . . 445
8.15.2.1 atomName . . . 445
8.15.2.2 charge . . . 446
8.15.2.3 epsilon . . . 446
8.15.2.4 id . . . 446
8.15.2.5 partID . . . 446
8.15.2.6 position . . . 446
8.15.2.7 radius . . . 446
8.15.2.8 resName . . . 446
8.16 sVclist Struct Reference . . . 447
8.16.1 Detailed Description . . . 447
8.16.2 Field Documentation . . . 447
8.16.2.1 alist . . . 447
8.16.2.2 cells . . . 447
8.16.2.3 lower_corner . . . 447
8.16.2.4 max_radius . . . 448
8.16.2.5 mode . . . 448
8.16.2.6 n . . . 448
8.16.2.7 npts . . . 448
8.16.2.8 spacs . . . 448
8.16.2.9 upper_corner . . . 448
8.16.2.10 vmem . . . 448
8.17 sVclistCell Struct Reference . . . 448
8.17.1 Detailed Description . . . 449
8.17.2 Field Documentation . . . 449
8.17.2.1 atoms . . . 449

8.17.2.2 natoms . . . 449
8.18 sVcsm Struct Reference . . . 449
8.18.1 Detailed Description . . . 450
8.18.2 Field Documentation . . . 450
8.18.2.1 alist . . . 450
8.18.2.2 gm . . . 450
8.18.2.3 initFlag . . . 450
8.18.2.4 msimp . . . 450
8.18.2.5 natom . . . 450
8.18.2.6 nqsm . . . 450
8.18.2.7 nsimp . . . 451
8.18.2.8 nsqm . . . 451
8.18.2.9 qsm . . . 451
8.18.2.10 sqm . . . 451
8.18.2.11 vmem . . . 451
8.19 sVfetk Struct Reference . . . 451
8.19.1 Detailed Description . . . 452
8.19.2 Field Documentation . . . 452
8.19.2.1 am . . . 452
8.19.2.2 aprx . . . 452
8.19.2.3 csm . . . 452
8.19.2.4 feparm . . . 452
8.19.2.5 gm . . . 453
8.19.2.6 gues . . . 453
8.19.2.7 level . . . 453
8.19.2.8 lkey . . . 453
8.19.2.9 lmax . . . 453
8.19.2.10 lprec . . . 453
8.19.2.11 ltol . . . 453
8.19.2.12 nkey . . . 453
8.19.2.13 nmax . . . 454
8.19.2.14 ntol . . . 454
8.19.2.15 pbe . . . 454
8.19.2.16 pbeparm . . . 454
8.19.2.17 pde . . . 454
8.19.2.18 pjac . . . 454
8.19.2.19 type . . . 454
8.19.2.20 vmem . . . 454
8.20 sVfetk_LocalVar Struct Reference . . . 455

8.20.1 Detailed Description . . . 455
8.20.2 Field Documentation . . . 456
8.20.2.1 A . . . 456
8.20.2.2 B . . . 456
8.20.2.3 d2W . . . 456
8.20.2.4 DB . . . 456
8.20.2.5 delta . . . 456
8.20.2.6 DFu_wv . . . 456
8.20.2.7 dU . . . 456
8.20.2.8 dW . . . 457
8.20.2.9 F . . . 457
8.20.2.10 fetk . . . 457
8.20.2.11 fType . . . 457
8.20.2.12 Fu_v . . . 457
8.20.2.13 green . . . 457
8.20.2.14 initGreen . . . 457
8.20.2.15 ionConc . . . 457
8.20.2.16 ionQ . . . 458
8.20.2.17 ionRadii . . . 458
8.20.2.18 ionstr . . . 458
8.20.2.19 jumpDiel . . . 458
8.20.2.20 nion . . . 458
8.20.2.21 nvec . . . 458
8.20.2.22 nverts . . . 458
8.20.2.23 simp . . . 458
8.20.2.24 sType . . . 459
8.20.2.25 U . . . 459
8.20.2.26 u_D . . . 459
8.20.2.27 u_T . . . 459
8.20.2.28 verts . . . 459
8.20.2.29 vx . . . 459
8.20.2.30 W . . . 459
8.20.2.31 xq . . . 459
8.20.2.32 zkappa2 . . . 460
8.20.2.33 zks2 . . . 460
8.21 sVgreen Struct Reference . . . 460
8.21.1 Detailed Description . . . 460
8.21.2 Field Documentation . . . 460
8.21.2.1 alist . . . 460

8.21.2.2 np . . . 461
8.21.2.3 qp . . . 461
8.21.2.4 vmem . . . 461
8.21.2.5 xp . . . 461
8.21.2.6 yp . . . 461
8.21.2.7 zp . . . 461
8.22 sVgrid Struct Reference . . . 461
8.22.1 Detailed Description . . . 462
8.22.2 Field Documentation . . . 462
8.22.2.1 ctordata . . . 462
8.22.2.2 data . . . 462
8.22.2.3 hx . . . 462
8.22.2.4 hy . . . 462
8.22.2.5 hzed . . . 463
8.22.2.6 mem . . . 463
8.22.2.7 nx . . . 463
8.22.2.8 ny . . . 463
8.22.2.9 nz . . . 463
8.22.2.10 readdata . . . 463
8.22.2.11 xmax . . . 463
8.22.2.12 xmin . . . 463
8.22.2.13 ymax . . . 464
8.22.2.14 ymin . . . 464
8.22.2.15 zmax . . . 464
8.22.2.16 zmin . . . 464
8.23 sVmgrid Struct Reference . . . 464
8.23.1 Detailed Description . . . 464
8.23.2 Field Documentation . . . 464
8.23.2.1 grids . . . 465
8.23.2.2 ngrids . . . 465
8.24 sVopot Struct Reference . . . 465
8.24.1 Detailed Description . . . 465
8.24.2 Field Documentation . . . 465
8.24.2.1 bcfl . . . 465
8.24.2.2 mgrid . . . 465
8.24.2.3 pbe . . . 466
8.25 sVparam_AtomData Struct Reference . . . 466
8.25.1 Detailed Description . . . 466
8.25.2 Field Documentation . . . 466

8.25.2.1 atomName . . . 466
8.25.2.2 charge . . . 467
8.25.2.3 epsilon . . . 467
8.25.2.4 radius . . . 467
8.25.2.5 resName . . . 467
8.26 sVpbe Struct Reference . . . 467
8.26.1 Detailed Description . . . 468
8.26.2 Field Documentation . . . 468
8.26.2.1 acc . . . 468
8.26.2.2 alist . . . 468
8.26.2.3 bulkIonicStrength . . . 468
8.26.2.4 clist . . . 469
8.26.2.5 deblen . . . 469
8.26.2.6 ionConc . . . 469
8.26.2.7 ionQ . . . 469
8.26.2.8 ionRadii . . . 469
8.26.2.9 ipkey . . . 469
8.26.2.10 L . . . 469
8.26.2.11 maxIonRadius . . . 469
8.26.2.12 membraneDiel . . . 470
8.26.2.13 numIon . . . 470
8.26.2.14 param2Flag . . . 470
8.26.2.15 paramFlag . . . 470
8.26.2.16 smsize . . . 470
8.26.2.17 smvolume . . . 470
8.26.2.18 soluteCenter . . . 470
8.26.2.19 soluteCharge . . . 470
8.26.2.20 soluteDiel . . . 471
8.26.2.21 soluteRadius . . . 471
8.26.2.22 soluteXlen . . . 471
8.26.2.23 soluteYlen . . . 471
8.26.2.24 soluteZlen . . . 471
8.26.2.25 solventDiel . . . 471
8.26.2.26 solventRadius . . . 471
8.26.2.27 T . . . 471
8.26.2.28 V . . . 472
8.26.2.29 vmem . . . 472
8.26.2.30 xkappa . . . 472
8.26.2.31 z_mem . . . 472

8.26.2.32 zkappa2 . . . 472
8.26.2.33 zmagic . . . 472
8.27 sVpee Struct Reference . . . 472
8.27.1 Detailed Description . . . 473
8.27.2 Field Documentation . . . 473
8.27.2.1 gm . . . 473
8.27.2.2 killFlag . . . 473
8.27.2.3 killParam . . . 473
8.27.2.4 localPartCenter . . . 473
8.27.2.5 localPartID . . . 473
8.27.2.6 localPartRadius . . . 474
8.27.2.7 mem . . . 474
8.28 sVpmg Struct Reference . . . 474
8.28.1 Detailed Description . . . 475
8.28.2 Field Documentation . . . 475
8.28.2.1 a1cf . . . 475
8.28.2.2 a2cf . . . 475
8.28.2.3 a3cf . . . 475
8.28.2.4 ccf . . . 476
8.28.2.5 charge . . . 476
8.28.2.6 chargeMap . . . 476
8.28.2.7 chargeMeth . . . 476
8.28.2.8 chargeSrc . . . 476
8.28.2.9 dielXMap . . . 476
8.28.2.10 dielYMap . . . 476
8.28.2.11 dielZMap . . . 476
8.28.2.12 epsx . . . 477
8.28.2.13 epsy . . . 477
8.28.2.14 epsz . . . 477
8.28.2.15 extDiEnergy . . . 477
8.28.2.16 extNpEnergy . . . 477
8.28.2.17 extQfEnergy . . . 477
8.28.2.18 extQmEnergy . . . 477
8.28.2.19 fcf . . . 477
8.28.2.20 filled . . . 478
8.28.2.21 gxcf . . . 478
8.28.2.22 gycf . . . 478
8.28.2.23 gzcf . . . 478
8.28.2.24 iparm . . . 478

8.28.2.25 iwork . . . 478
8.28.2.26 kappa . . . 478
8.28.2.27 kappaMap . . . 478
8.28.2.28 pbe . . . 479
8.28.2.29 pmgp . . . 479
8.28.2.30 pot . . . 479
8.28.2.31 potMap . . . 479
8.28.2.32 pvec . . . 479
8.28.2.33 rparm . . . 479
8.28.2.34 rwork . . . 479
8.28.2.35 splineWin . . . 479
8.28.2.36 surfMeth . . . 480
8.28.2.37 tcf . . . 480
8.28.2.38 u . . . 480
8.28.2.39 useChargeMap . . . 480
8.28.2.40 useDielXMap . . . 480
8.28.2.41 useDielYMap . . . 480
8.28.2.42 useDielZMap . . . 480
8.28.2.43 useKappaMap . . . 480
8.28.2.44 usePotMap . . . 481
8.28.2.45 vmem . . . 481
8.28.2.46 xf . . . 481
8.28.2.47 yf . . . 481
8.28.2.48 zf . . . 481
8.29 sVpmgp Struct Reference . . . 481
8.29.1 Detailed Description . . . 482
8.29.2 Field Documentation . . . 483
8.29.2.1 bcfl . . . 483
8.29.2.2 errtol . . . 483
8.29.2.3 hx . . . 483
8.29.2.4 hy . . . 483
8.29.2.5 hzed . . . 483
8.29.2.6 iinfo . . . 483
8.29.2.7 ipcon . . . 484
8.29.2.8 iperf . . . 484
8.29.2.9 ipkey . . . 484
8.29.2.10 irite . . . 484
8.29.2.11 istop . . . 485
8.29.2.12 itmax . . . 485

8.29.2.13 key . . . 485
8.29.2.14 meth . . . 485
8.29.2.15 mgcoar . . . 486
8.29.2.16 mgdisc . . . 486
8.29.2.17 mgkey . . . 486
8.29.2.18 mgprol . . . 486
8.29.2.19 mgsmoo . . . 486
8.29.2.20 mgsolv . . . 487
8.29.2.21 n_ipc . . . 487
8.29.2.22 n_iz . . . 487
8.29.2.23 n_rpc . . . 487
8.29.2.24 narr . . . 487
8.29.2.25 narrc . . . 487
8.29.2.26 nc . . . 488
8.29.2.27 nf . . . 488
8.29.2.28 niwk . . . 488
8.29.2.29 nlev . . . 488
8.29.2.30 nonlin . . . 488
8.29.2.31 nrwk . . . 488
8.29.2.32 nu1 . . . 488
8.29.2.33 nu2 . . . 489
8.29.2.34 nx . . . 489
8.29.2.35 nxc . . . 489
8.29.2.36 ny . . . 489
8.29.2.37 nyc . . . 489
8.29.2.38 nz . . . 489
8.29.2.39 nzc . . . 489
8.29.2.40 omegal . . . 489
8.29.2.41 omegan . . . 490
8.29.2.42 xcent . . . 490
8.29.2.43 xlen . . . 490
8.29.2.44 xmax . . . 490
8.29.2.45 xmin . . . 490
8.29.2.46 ycent . . . 490
8.29.2.47 ylen . . . 490
8.29.2.48 ymax . . . 491
8.29.2.49 ymin . . . 491
8.29.2.50 zcent . . . 491
8.29.2.51 zlen . . . 491

8.29.2.52 zmax . . . 491
8.29.2.53 zmin . . . 491
8.30 Vparam Struct Reference . . . 491
8.30.1 Detailed Description . . . 492
8.30.2 Field Documentation . . . 492
8.30.2.1 nResData . . . 492
8.30.2.2 resData . . . 492
8.30.2.3 vmem . . . 492
8.31 Vparam_ResData Struct Reference . . . 492
8.31.1 Detailed Description . . . 492
8.31.2 Field Documentation . . . 493
8.31.2.1 atomData . . . 493
8.31.2.2 name . . . 493
8.31.2.3 nAtomData . . . 493
8.31.2.4 vmem . . . 493

9 File Documentation **495**

9.1 src/apbs.h File Reference . . . 495
9.1.1 Detailed Description . . . 496
9.2 apbs.h . . . 497
9.3 src/fem/vcsm.c File Reference . . . 498
9.3.1 Detailed Description . . . 499
9.4 vcsm.c . . . 500
9.5 src/fem/vcsm.h File Reference . . . 506
9.5.1 Detailed Description . . . 508
9.6 vcsm.h . . . 509
9.7 src/fem/vfetk.c File Reference . . . 510
9.7.1 Detailed Description . . . 513
9.7.2 Variable Documentation . . . 514
9.7.2.1 lgr_2DP1 . . . 514
9.7.2.2 lgr_2DP1x . . . 515
9.7.2.3 lgr_2DP1y . . . 515
9.7.2.4 lgr_2DP1z . . . 515
9.7.2.5 lgr_3DP1 . . . 515
9.7.2.6 lgr_3DP1x . . . 515
9.7.2.7 lgr_3DP1y . . . 515
9.7.2.8 lgr_3DP1z . . . 516
9.8 vfetk.c . . . 516
9.9 src/fem/vfetk.h File Reference . . . 546

9.9.1 Detailed Description . . . 550
9.10 vfetk.h . . . 551
9.11 src/fem/vpee.c File Reference . . . 556
9.11.1 Detailed Description . . . 557
9.12 vpee.c . . . 558
9.13 src/fem/vpee.h File Reference . . . 565
9.13.1 Detailed Description . . . 566
9.14 vpee.h . . . 567
9.15 src/generic/apolparm.c File Reference . . . 568
9.15.1 Detailed Description . . . 569
9.16 apolparm.c . . . 570
9.17 src/generic/bemparm.c File Reference . . . 577
9.17.1 Detailed Description . . . 579
9.17.2 Function Documentation . . . 579
9.17.2.1 BEMparm_copy() . . . 580
9.18 bemparm.c . . . 580
9.19 src/generic/femparm.c File Reference . . . 584
9.19.1 Detailed Description . . . 585
9.20 femparm.c . . . 586
9.21 src/generic/femparm.h File Reference . . . 591
9.21.1 Detailed Description . . . 593
9.22 femparm.h . . . 594
9.23 src/generic/geoflowparm.c File Reference . . . 595
9.23.1 Detailed Description . . . 597
9.24 geoflowparm.c . . . 597
9.25 src/generic/geoflowparm.h File Reference . . . 600
9.25.1 Detailed Description . . . 602
9.26 geoflowparm.h . . . 603
9.27 src/generic/mgparm.c File Reference . . . 603
9.27.1 Detailed Description . . . 605
9.28 mgparm.c . . . 606
9.29 src/generic/mgparm.h File Reference . . . 618
9.29.1 Detailed Description . . . 620
9.30 mgparm.h . . . 621
9.31 src/generic/nosh.c File Reference . . . 623
9.31.1 Detailed Description . . . 625
9.32 nosh.c . . . 626
9.33 src/generic/nosh.h File Reference . . . 666
9.33.1 Detailed Description . . . 670

9.34 nosh.h . . . 670
9.35 src/generic/pbamparm.c File Reference . . . 673
9.35.1 Detailed Description . . . 675
9.36 pbamparm.c . . . 676
9.37 src/generic/pbamparm.h File Reference . . . 684
9.37.1 Detailed Description . . . 687
9.38 pbamparm.h . . . 688
9.39 src/generic/pbeparm.c File Reference . . . 690
9.39.1 Detailed Description . . . 692
9.40 pbeparm.c . . . 692
9.41 src/generic/pbeparm.h File Reference . . . 708
9.41.1 Detailed Description . . . 710
9.42 pbeparm.h . . . 711
9.43 src/generic/pbsamparm.c File Reference . . . 713
9.43.1 Detailed Description . . . 714
9.44 pbsamparm.c . . . 715
9.45 src/generic/pbsamparm.h File Reference . . . 718
9.45.1 Detailed Description . . . 720
9.46 pbsamparm.h . . . 721
9.47 src/generic/vacc.c File Reference . . . 722
9.47.1 Detailed Description . . . 724
9.47.2 Function Documentation . . . 725
9.47.2.1 ivdwAccExclus() . . . 725
9.47.2.2 splineAcc() . . . 726
9.47.2.3 Vacc_allocate() . . . 726
9.47.2.4 Vacc_storeParms() . . . 727
9.48 vacc.c . . . 727
9.49 src/generic/vacc.h File Reference . . . 750
9.49.1 Detailed Description . . . 753
9.50 vacc.h . . . 754
9.51 src/generic/valist.c File Reference . . . 757
9.51.1 Detailed Description . . . 759
9.52 valist.c . . . 760
9.53 src/generic/valist.h File Reference . . . 771
9.53.1 Detailed Description . . . 773
9.54 valist.h . . . 774
9.55 src/generic/vatom.c File Reference . . . 775
9.55.1 Detailed Description . . . 777
9.56 vatom.c . . . 778

9.57 src/generic/vatom.h File Reference . . . 781
9.57.1 Detailed Description . . . 783
9.58 vatom.h . . . 784
9.59 src/generic/vcap.c File Reference . . . 786
9.59.1 Detailed Description . . . 787
9.60 vcap.c . . . 788
9.61 src/generic/vcap.h File Reference . . . 788
9.61.1 Detailed Description . . . 790
9.62 vcap.h . . . 791
9.63 src/generic/vclist.c File Reference . . . 791
9.63.1 Detailed Description . . . 792
9.64 vclist.c . . . 793
9.65 src/generic/vclist.h File Reference . . . 799
9.65.1 Detailed Description . . . 801
9.66 vclist.h . . . 802
9.67 src/generic/vgreen.c File Reference . . . 803
9.67.1 Detailed Description . . . 805
9.68 vgreen.c . . . 806
9.69 src/generic/vgreen.h File Reference . . . 812
9.69.1 Detailed Description . . . 813
9.70 vgreen.h . . . 814
9.71 src/generic/vhal.h File Reference . . . 815
9.71.1 Detailed Description . . . 819
9.71.2 Macro Definition Documentation . . . 819
9.71.2.1 PRINT_FUNC . . . 819
9.71.2.2 VABORT_MSG0 . . . 820
9.71.2.3 VABORT_MSG1 . . . 820
9.71.2.4 VABORT_MSG2 . . . 820
9.71.2.5 VASSERT_MSG0 . . . 820
9.71.2.6 VASSERT_MSG1 . . . 821
9.71.2.7 VASSERT_MSG2 . . . 821
9.71.2.8 VCHANNELEDMESSAGE0 . . . 821
9.71.2.9 VCHANNELEDMESSAGE1 . . . 821
9.71.2.10 VCHANNELEDMESSAGE2 . . . 822
9.71.2.11 VCHANNELEDMESSAGE3 . . . 822
9.71.2.12 VCOPY . . . 822
9.71.2.13 VFILL . . . 822
9.71.2.14 VWARN_MSG0 . . . 823
9.71.2.15 VWARN_MSG1 . . . 823

9.71.2.16 VWARN_MSG2 . . . 823
9.72 vhal.h . . . 824
9.73 src/generic/vmatrix.h File Reference . . . 832
9.73.1 Detailed Description . . . 832
9.73.2 Macro Definition Documentation . . . 833
9.73.2.1 MAT2 . . . 833
9.73.2.2 MAT3 . . . 834
9.73.2.3 VAT3 . . . 834
9.74 vmatrix.h . . . 834
9.75 src/generic/vparam.c File Reference . . . 834
9.75.1 Detailed Description . . . 836
9.76 vparam.c . . . 837
9.77 src/generic/vparam.h File Reference . . . 845
9.77.1 Detailed Description . . . 848
9.78 vparam.h . . . 849
9.79 src/generic/vpbe.c File Reference . . . 850
9.79.1 Detailed Description . . . 852
9.80 vpbe.c . . . 853
9.81 src/generic/vpbe.h File Reference . . . 859
9.81.1 Detailed Description . . . 862
9.82 vpbe.h . . . 862
9.83 src/generic/vstring.c File Reference . . . 865
9.83.1 Detailed Description . . . 866
9.84 vstring.c . . . 867
9.85 src/generic/vstring.h File Reference . . . 869
9.85.1 Detailed Description . . . 870
9.86 vstring.h . . . 872
9.87 src/generic/vunit.h File Reference . . . 872
9.87.1 Detailed Description . . . 873
9.88 vunit.h . . . 874
9.89 src/main.c File Reference . . . 874
9.89.1 Detailed Description . . . 875
9.90 main.c . . . 876
9.91 src/mg/vgrid.c File Reference . . . 887
9.91.1 Detailed Description . . . 889
9.91.2 Function Documentation . . . 889
9.91.2.1 Vgrid_writeGZ() . . . 890
9.92 vgrid.c . . . 890
9.93 src/mg/vgrid.h File Reference . . . 914

9.93.1 Detailed Description . . . 916
9.93.2 Function Documentation . . . 917
9.93.2.1 Vgrid_writeGZ() . . . 917
9.94 vgrid.h . . . 918
9.95 src/mg/vmgrid.c File Reference . . . 919
9.95.1 Detailed Description . . . 921
9.96 vmgrid.c . . . 921
9.97 src/mg/vmgrid.h File Reference . . . 924
9.97.1 Detailed Description . . . 926
9.98 vmgrid.h . . . 927
9.99 src/mg/vopot.c File Reference . . . 927
9.99.1 Detailed Description . . . 928
9.100 vopot.c . . . 929
9.101 src/mg/vopot.h File Reference . . . 933
9.101.1 Detailed Description . . . 935
9.102 vopot.h . . . 936
9.103 src/mg/vpmg.c File Reference . . . 936
9.103.1 Detailed Description . . . 940
9.103.2 Function Documentation . . . 941
9.103.2.1 bcCalc() . . . 941
9.103.2.2 bcfl1() . . . 942
9.103.2.3 bspline2() . . . 942
9.103.2.4 bspline4() . . . 943
9.103.2.5 d2bspline4() . . . 943
9.103.2.6 d3bspline4() . . . 944
9.103.2.7 dbspline2() . . . 944
9.103.2.8 dbspline4() . . . 944
9.103.2.9 fillcoCharge() . . . 945
9.103.2.10 fillcoChargeMap() . . . 945
9.103.2.11 fillcoChargeSpline1() . . . 945
9.103.2.12 fillcoChargeSpline2() . . . 946
9.103.2.13 fillcoCoef() . . . 946
9.103.2.14 fillcoCoefMap() . . . 946
9.103.2.15 fillcoCoefMol() . . . 946
9.103.2.16 fillcoCoefMolDiel() . . . 946
9.103.2.17 fillcoCoefMolDielNoSmooth() . . . 947
9.103.2.18 fillcoCoefMolDielSmooth() . . . 947
9.103.2.19 fillcoCoefMolIon() . . . 947
9.103.2.20 fillcoCoefSpline() . . . 947

9.103.2.21 fillcoCoefSpline3() . . . 948
9.103.2.22 fillcoCoefSpline4() . . . 948
9.103.2.23 fillcoPermanentMultipole() . . . 948
9.103.2.24 markSphere() . . . 948
9.103.2.25 multipolebc() . . . 949
9.103.2.26 qfForceSpline1() . . . 949
9.103.2.27 qfForceSpline2() . . . 950
9.103.2.28 qfForceSpline4() . . . 950
9.103.2.29 VFCHI4() . . . 951
9.103.2.30 Vpmg_polarizEnergy() . . . 951
9.103.2.31 Vpmg_qfEnergyPoint() . . . 951
9.103.2.32 Vpmg_qfEnergyVolume() . . . 952
9.103.2.33 Vpmg_qmEnergySMPBE() . . . 952
9.103.2.34 Vpmg_splineSelect() . . . 952
9.103.2.35 zlapSolve() . . . 953
9.104 vpmg.c . . . 953
9.105 src/mg/vpmg.h File Reference . . . 1089
9.105.1 Detailed Description . . . 1095
9.105.2 Function Documentation . . . 1096
9.105.2.1 bcCalc() . . . 1096
9.105.2.2 bcfl1() . . . 1096
9.105.2.3 bspline2() . . . 1097
9.105.2.4 bspline4() . . . 1098
9.105.2.5 d2bspline4() . . . 1098
9.105.2.6 d3bspline4() . . . 1098
9.105.2.7 dbspline2() . . . 1099
9.105.2.8 dbspline4() . . . 1099
9.105.2.9 fillcoCharge() . . . 1100
9.105.2.10 fillcoChargeMap() . . . 1100
9.105.2.11 fillcoChargeSpline1() . . . 1100
9.105.2.12 fillcoChargeSpline2() . . . 1100
9.105.2.13 fillcoCoef() . . . 1101
9.105.2.14 fillcoCoefMap() . . . 1101
9.105.2.15 fillcoCoefMol() . . . 1101
9.105.2.16 fillcoCoefMolDiel() . . . 1101
9.105.2.17 fillcoCoefMolDielNoSmooth() . . . 1101
9.105.2.18 fillcoCoefMolDielSmooth() . . . 1102
9.105.2.19 fillcoCoefMolIon() . . . 1102
9.105.2.20 fillcoCoefSpline() . . . 1102

9.105.2.21 fillcoCoefSpline3() . . . 1102
9.105.2.22 fillcoCoefSpline4() . . . 1103
9.105.2.23 fillcoInducedDipole() . . . 1103
9.105.2.24 fillcoNLInducedDipole() . . . 1103
9.105.2.25 fillcoPermanentMultipole() . . . 1103
9.105.2.26 markSphere() . . . 1103
9.105.2.27 multipolebc() . . . 1104
9.105.2.28 qfForceSpline1() . . . 1105
9.105.2.29 qfForceSpline2() . . . 1105
9.105.2.30 qfForceSpline4() . . . 1105
9.105.2.31 VFCHI4() . . . 1106
9.105.2.32 Vpmg_polarizEnergy() . . . 1106
9.105.2.33 Vpmg_qfEnergyPoint() . . . 1106
9.105.2.34 Vpmg_qfEnergyVolume() . . . 1107
9.105.2.35 Vpmg_qmEnergySMPBE() . . . 1107
9.105.2.36 Vpmg_splineSelect() . . . 1108
9.105.2.37 zlapSolve() . . . 1108
9.106 vpmg.h . . . 1109
9.107 src/mg/vpmgp.c File Reference . . . 1117
9.107.1 Detailed Description . . . 1119
9.108 vpmgp.c . . . 1119
9.109 src/mg/vpmgp.h File Reference . . . 1123
9.109.1 Detailed Description . . . 1126
9.110 vpmgp.h . . . 1127
9.111 src/routines.c File Reference . . . 1128
9.111.1 Detailed Description . . . 1131
9.112 routines.c . . . 1132
9.113 src/routines.h File Reference . . . 1200
9.113.1 Detailed Description . . . 1203
9.114 routines.h . . . 1204

Index **1209**

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.

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

Bug List

Global Bmat_printHB (Bmat *thee, char *fname)

Hardwired to only handle the single block symmetric case.

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

"calcenergy 2" does not work

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

THIS FUNCTION IS HARD-CODED TO SOLVE LRPBE

THIS FUNCTION IS HARD-CODED TO SOLVE LRPBE

Class sVpmgp

Value ipcon does not currently allow for preconditioning in PMG

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

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

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

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

Global Vfetk_dumpLocalVar ()

This function is not thread-safe

Global Vfetk_externalUpdateFunction (SS **simps, int num)

This function is not thread-safe.

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

Several values of type are not implemented

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 vertex↩ Type[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, size_t *nrwk, int *niwk, int *nx, int *ny, int *nz, int *nlev, int *nu1, int *nu2, int *mgkey, int *itmax, int *istop, int *ipcon, int *nonlin, int *mgsmoo, int *mgprol, int *mgcoar, int *mgsolv, int *mgdisc, int *iinfo, double *errtol, int *ipkey, double *omegal, double *omegan, int *irite, int *iperf)

Can this path variable be replaced with a Vio socket?

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

The focusing flag is currently not used!!

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 3

Deprecated List

Global sMGparm::nlev

Just ignored now

Chapter 4

Module Index

4.1 Modules

Here is a list of all modules:

dependencies . . . 19
Vcsm class . . . 19
Vfetk class . . . 26
Vpee class . . . 56
APOLparm class . . . 60
BEMparm class . . . 64
FEMparm class . . . 68
GEOFLOWparm class . . . 73
MGparm class . . . 77
NOsh class . . . 88
PBAMparm class . . . 105
PBEparm class . . . 115
PBSAMparm class . . . 121
Vacc class . . . 127
Valist class . . . 146
Vatom class . . . 153
Vcap class . . . 163
Vclist class . . . 166
Vgreen class . . . 172
Vhal class . . . 181
Matrix wrapper class . . . 190
Vparam class . . . 190
Vpbe class . . . 201
Vstring class . . . 216
Vunit class . . . 218
Vgrid class . . . 219
Vmgrid class . . . 231
Vopot class . . . 236
Vpmg class . . . 240
Vpmgp class . . . 267
C translation of Holst group PMG code . . . 270
High-level front-end routines . . . 362

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

AtomForce
: Structure to hold atomic forces . . . 393

sAPOLparm
: Parameter structure for APOL-specific variables from input files . . . 394

sBEMparm
: Parameter structure for BEM-specific variables from input files . . . 400

sFEMparm
: Parameter structure for FEM-specific variables from input files . . . 404

sGEOFLOWparm
: Parameter structure for GEOFLOW-specific variables from input files . . . 408

sMGparm
: Parameter structure for MG-specific variables from input files . . . 409

sNOsh
: Class for parsing fixed format input files . . . 418

sNOsh_calc
: Calculation class for use when parsing fixed format input files . . . 425

sPBAMparm
: Parameter structure for PBAM-specific variables from input files . . . 427

sPBEparm
: Parameter structure for PBE variables from input files . . . 428

sPBSAMparm
: Parameter structure for PBSAM-specific variables from input files . . . 439

sVacc
: Oracle for solvent- and ion-accessibility around a biomolecule . . . 440

sVaccSurf
: Surface object list of per-atom surface points . . . 442

sValist
: Container class for list of atom objects . . . 443

sVatom
: Contains public data members for Vatom class/module . . . 445

sVclist
: Atom cell list . . . 447

sVclistCell
: Atom cell list cell . . . 448

sVcsm
: Charge-simplex map class . . . 449

sVfetk
Contains public data members for Vfetk class/module . . . 451
sVfetk_LocalVar
Vfetk LocalVar subclass . . . 455
sVgreen
Contains public data members for Vgreen class/module . . . 460
sVgrid
Electrostatic potential oracle for Cartesian mesh data . . . 461
sVmgrid
Multiresoltion oracle for Cartesian mesh data . . . 464
sVopot
Electrostatic potential oracle for Cartesian mesh data . . . 465
sVparam_AtomData
AtomData sub-class; stores atom data . . . 466
sVpbe
Contains public data members for Vpbe class/module . . . 467
sVpee
Contains public data members for Vpee class/module . . . 472
sVpmg
Contains public data members for Vpmg class/module . . . 474
sVpmgp
Contains public data members for Vpmgp class/module . . . 481
Vparam
Reads and assigns charge/radii parameters . . . 491
Vparam_ResData
ResData sub-class; stores residue data . . . 492

Chapter 6

File Index

6.1 File List

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

doc/programmer/**mainpage.h** . . . . . **??**

src/apbs.h
Header file for header dependencies 495

src/main.c
APBS "front end" program using formatted input files 874

src/routines.c
Supporting routines for APBS front end 1128

src/routines.h
Header file for front end auxiliary routines 1200

src/fem/vcsm.c
Class Vcsm methods 498

src/fem/vcsm.h
Contains declarations for the Vcsm class 506

src/fem/vfetk.c
Class Vfetk methods 510

src/fem/vfetk.h
Contains declarations for class Vfetk 546

src/fem/vpee.c
Class Vpee methods 556

src/fem/vpee.h
Contains declarations for class Vpee 565

src/generic/apolparm.c
Class APOLparm methods 568

src/generic/**apolparm.h** . . . . . **??**

src/generic/bemparm.c
Class BEMparm methods 577

src/generic/**bemparm.h** . . . . . **??**

src/generic/femparm.c
Class FEMparm methods 584

src/generic/femparm.h
Contains declarations for class APOLparm 591

src/generic/geoflowparm.c
Class GEOFLOWparm methods 595

src/generic/geoflowparm.h
Contains declarations for class GEOFLOWparm 600

src/generic/mgparm.c
Class MGparm methods . . . 603
src/generic/mgparm.h
Contains declarations for class MGparm . . . 618
src/generic/nosh.c
Class NOsh methods . . . 623
src/generic/nosh.h
Contains declarations for class NOsh . . . 666
src/generic/pbamparm.c
Class PBAMparm methods . . . 673
src/generic/pbamparm.h
Contains declarations for class PBAMparm . . . 684
src/generic/pbeparm.c
Class PBEparm methods . . . 690
src/generic/pbeparm.h
Contains declarations for class PBEparm . . . 708
src/generic/pbsamparm.c
Class PBSAMparm methods . . . 713
src/generic/pbsamparm.h
Contains declarations for class PBSAMparm . . . 718
src/generic/vacc.c
Class Vacc methods . . . 722
src/generic/vacc.h
Contains declarations for class Vacc . . . 750
src/generic/valist.c
Class Valist methods . . . 757
src/generic/valist.h
Contains declarations for class Valist . . . 771
src/generic/vatom.c
Class Vatom methods . . . 775
src/generic/vatom.h
Contains declarations for class Vatom . . . 781
src/generic/vcap.c
Class Vcap methods . . . 786
src/generic/vcap.h
Contains declarations for class Vcap . . . 788
src/generic/vclist.c
Class Vclist methods . . . 791
src/generic/vclist.h
Contains declarations for class Vclist . . . 799
src/generic/vgreen.c
Class Vgreen methods . . . 803
src/generic/vgreen.h
Contains declarations for class Vgreen . . . 812
src/generic/vhal.h
Contains generic macro definitions for APBS . . . 815
src/generic/vmatrix.h
Contains inclusions for matrix data wrappers . . . 832
src/generic/vparam.c
Class Vparam methods . . . 834
src/generic/vparam.h
Contains declarations for class Vparam . . . 845
src/generic/vpbe.c
Class Vpbe methods . . . 850

src/generic/vpbe.h
Contains declarations for class Vpbe . . . 859
src/generic/vstring.c
Class Vstring methods . . . 865
src/generic/vstring.h
Contains declarations for class Vstring . . . 869
src/generic/vunit.h
Contains a collection of useful constants and conversion factors . . . 872
src/mg/vgrid.c
Class Vgrid methods . . . 887
src/mg/vgrid.h
Potential oracle for Cartesian mesh data . . . 914
src/mg/vmgrid.c
Class Vmgrid methods . . . 919
src/mg/vmgrid.h
Multiresolution oracle for Cartesian mesh data . . . 924
src/mg/vopot.c
Class Vopot methods . . . 927
src/mg/vopot.h
Potential oracle for Cartesian mesh data . . . 933
src/mg/vpmg.c
Class Vpmg methods . . . 936
src/mg/vpmg.h
Contains declarations for class Vpmg . . . 1089
src/mg/vpmgp.c
Class Vpmgp methods . . . 1117
src/mg/vpmgp.h
Contains declarations for class Vpmgp . . . 1123
src/pmgc/**buildAd.c** . . . **??**
src/pmgc/**buildAd.h** . . . **??**
src/pmgc/**buildBd.c** . . . **??**
src/pmgc/**buildBd.h** . . . **??**
src/pmgc/**buildGd.c** . . . **??**
src/pmgc/**buildGd.h** . . . **??**
src/pmgc/**buildPd.c** . . . **??**
src/pmgc/**buildPd.h** . . . **??**
src/pmgc/**cgd.c** . . . **??**
src/pmgc/**cgd.h** . . . **??**
src/pmgc/**gsd.c** . . . **??**
src/pmgc/**gsd.h** . . . **??**
src/pmgc/**matvecd.c** . . . **??**
src/pmgc/**matvecd.h** . . . **??**
src/pmgc/**mgcsd.c** . . . **??**
src/pmgc/**mgcsd.h** . . . **??**
src/pmgc/**mgdrvd.c** . . . **??**
src/pmgc/**mgdrvd.h** . . . **??**
src/pmgc/**mgfasd.c** . . . **??**
src/pmgc/**mgfasd.h** . . . **??**
src/pmgc/**mgsubd.c** . . . **??**
src/pmgc/**mgsubd.h** . . . **??**
src/pmgc/**mikpckd.c** . . . **??**
src/pmgc/**mikpckd.h** . . . **??**
src/pmgc/**mlinpckd.c** . . . **??**
src/pmgc/**mlinpckd.h** . . . **??**

src/pmgc/**mypdec.c** . . . . . **??**
src/pmgc/**mypdec.h** . . . . . **??**
src/pmgc/**newdrvd.c** . . . . . **??**
src/pmgc/**newdrvd.h** . . . . . **??**
src/pmgc/**newtond.c** . . . . . **??**
src/pmgc/**newtond.h** . . . . . **??**
src/pmgc/**powerd.c** . . . . . **??**
src/pmgc/**powerd.h** . . . . . **??**
src/pmgc/**smoothd.c** . . . . . **??**
src/pmgc/**smoothd.h** . . . . . **??**

Chapter 7

Module Documentation

7.1 dependencies

7.2 Vcsm class

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

Files

- file vcsm.c

  Class Vcsm methods.
- file vcsm.h

  Contains declarations for the Vcsm class.

Data Structures

- struct sVcsm

  Charge-simplex map class.

Typedefs

- typedef struct sVcsm Vcsm

  Declaration of the Vcsm class as the Vcsm structure.

Functions

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

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

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

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

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

  Get ID of particular atom in a simplex.
- VEXTERNC int Vcsm_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.

7.2.1 Detailed Description

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

7.2.2 Function Documentation

7.2.2.1 Gem_setExternalUpdateFunction()

```
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

7.2.2.2 Vcsm_ctor()

```
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 136 of file vcsm.c.

7.2.2.3 Vcsm_ctor2()

```
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 147 of file vcsm.c.

7.2.2.4 Vcsm_dtor()

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

Destroy Vcsm object.

Author

Nathan Baker

Parameters

thee	Pointer to memory location for Vcsm object

Definition at line 292 of file vcsm.c.

7.2.2.5 Vcsm_dtor2()

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

FORTRAN stub to destroy Vcsm object.

Author

Nathan Baker

Parameters

thee	Pointer to Vcsm object

Definition at line 300 of file vcsm.c.

7.2.2.6 Vcsm_getAtom()

```
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 77 of file vcsm.c.

7.2.2.7 Vcsm_getAtomIndex()

```
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 88 of file vcsm.c.

7.2.2.8 Vcsm_getNumberAtoms()

```
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 69 of file vcsm.c.

7.2.2.9 Vcsm_getNumberSimplices()

```
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 99 of file vcsm.c.

7.2.2.10 Vcsm_getSimplex()

```
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 109 of file vcsm.c.

7.2.2.11 Vcsm_getSimplexIndex()

```
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 119 of file vcsm.c.

7.2.2.12 Vcsm_getValist()

```
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 62 of file vcsm.c.

7.2.2.13 Vcsm_init()

```
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 170 of file vcsm.c.

7.2.2.14 Vcsm_memChk()

```
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 129 of file vcsm.c.

7.2.2.15 Vcsm_update()

```
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 326 of file vcsm.c.

7.3 Vfetk class

FEtk master class (interface between FEtk and APBS)

Files

- file vfetk.c

  Class Vfetk methods.
- file vfetk.h

  Contains declarations for class Vfetk.

Data Structures

- struct sVfetk

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

  Vfetk LocalVar subclass.

Macros

- #define VRINGMAX 1000

  Maximum number of simplices in a simplex ring.
- #define VATOMMAX 1000000

  Maximum number of atoms associated with a vertex.

Typedefs

- typedef enum eVfetk_LsolvType Vfetk_LsolvType

  Declare FEMparm_LsolvType type.
- typedef enum eVfetk_MeshLoad Vfetk_MeshLoad

  Declare FEMparm_GuessType type.
- typedef enum eVfetk_NsolvType Vfetk_NsolvType

  Declare FEMparm_NsolvType type.
- typedef enum eVfetk_GuessType Vfetk_GuessType

  Declare FEMparm_GuessType type.
- typedef enum eVfetk_PrecType Vfetk_PrecType

  Declare FEMparm_GuessType type.
- typedef struct sVfetk_LocalVar Vfetk_LocalVar

  Declaration of the Vfetk_LocalVar subclass as the Vfetk_LocalVar structure.
- typedef struct sVfetk Vfetk

  Declaration of the Vfetk class as the Vfetk structure.

Enumerations

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

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

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

  Non-linear solver type.
- enum eVfetk_GuessType { 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

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

  Return the total electrostatic energy.
- VEXTERNC Gem * Vfetk_getGem (Vfetk *thee)

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

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

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

  Get a pointer to the Vcsm (charge-simplex map) object.
- VEXTERNC int Vfetk_getAtomColor (Vfetk *thee, int iatom)

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

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

  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_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:
- 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:
- 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:
- 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:
- 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.

7.3.1 Detailed Description

FEtk master class (interface between FEtk and APBS)

7.3.2 Enumeration Type Documentation

7.3.2.1 eVfetk_GuessType

`enum eVfetk_GuessType`

Initial guess type.

Note

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

Enumerator

VGT_ZERO	Zero initial guess
VGT_DIRI	Dirichlet boundary condition initial guess
VGT_PREV	Previous level initial guess

Definition at line 138 of file vfetk.h.

7.3.2.2 eVfetk_LsolvType

`enum eVfetk_LsolvType`

Linear solver type.

Note

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

Enumerator

VLT_SLU	SuperLU direct solve
VLT_MG	Multigrid
VLT_CG	Conjugate gradient
VLT_BCG	BiCGStab

Definition at line 86 of file vfetk.h.

7.3.2.3 eVfetk_MeshLoad

enum eVfetk_MeshLoad
Mesh loading operation.

Enumerator

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

Definition at line 104 of file vfetk.h.

7.3.2.4 eVfetk_NsolvType

enum eVfetk_NsolvType
Non-linear solver type.

Note

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

Enumerator

VNT_NEW	Newton solver
VNT_INC	Incremental
VNT_ARC	Psuedo-arclength

Definition at line 121 of file vfetk.h.

7.3.2.5 eVfetk_PrecType

enum eVfetk_PrecType
Preconditioner type.

Note

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

Enumerator

VPT_IDEN	Identity matrix
VPT_DIAG	Diagonal scaling
VPT_MG	Multigrid

Definition at line 155 of file vfetk.h.

7.3.3 Function Documentation

7.3.3.1 Bmat_printHB()

```
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 1026 of file vfetk.c.

7.3.3.2 Vfetk_ctor()

```
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 532 of file vfetk.c.

7.3.3.3 Vfetk_ctor2()

```
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 545 of file vfetk.c.

7.3.3.4 Vfetk_dqmEnergy()

```
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:

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

for the NPBE and

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

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 842 of file vfetk.c.

7.3.3.5 Vfetk_dtor()

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

Object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of Vfetk object

Definition at line 625 of file vfetk.c.

7.3.3.6 Vfetk_dtor2()

```
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 633 of file vfetk.c.

7.3.3.7 Vfetk_dumpLocalVar()

```
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 2255 of file vfetk.c.

7.3.3.8 Vfetk_energy()

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

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

Author

Nathan Baker

Returns

Total electrostatic energy in units of k_BT.

$<$ 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 693 of file vfetk.c.

7.3.3.9 Vfetk_externalUpdateFunction()

```
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 2078 of file vfetk.c.

7.3.3.10 Vfetk_fillArray()

```
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 2299 of file vfetk.c.

7.3.3.11 Vfetk_genCube()

```
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 885 of file vfetk.c.

7.3.3.12 Vfetk_getAM()

```
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 497 of file vfetk.c.

7.3.3.13 Vfetk_getAtomColor()

```
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 517 of file vfetk.c.

7.3.3.14 Vfetk_getGem()

```
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 490 of file vfetk.c.

7.3.3.15 Vfetk_getSolution()

```
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 641 of file vfetk.c.

7.3.3.16 Vfetk_getVcsm()

```
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 510 of file vfetk.c.

7.3.3.17 Vfetk_getVpbe()

```
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 503 of file vfetk.c.

7.3.3.18 Vfetk_loadGem()

```
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

7.3.3.19 Vfetk_loadMesh()

```
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 980 of file vfetk.c.

7.3.3.20 Vfetk_memChk()

```
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 867 of file vfetk.c.

7.3.3.21 Vfetk_PDE_bisectEdge()

```
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

7.3.3.22 Vfetk_PDE_ctor()

```
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 1176 of file vfetk.c.

7.3.3.23 Vfetk_PDE_ctor2()

```
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 1187 of file vfetk.c.

7.3.3.24 Vfetk_PDE_delta()

```
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 1780 of file vfetk.c.

7.3.3.25 Vfetk_PDE_DFu_wv()

```
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:

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

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

7.3.3.26 Vfetk_PDE_dtor()

```
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 1231 of file vfetk.c.

7.3.3.27 Vfetk_PDE_dtor2()

```
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 1246 of file vfetk.c.

7.3.3.28 Vfetk_PDE_Fu()

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

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

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

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

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

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

Author

Nathan Baker

Bug This function is not thread-safe

This function is not implemented (sets error to zero)

Parameters

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

Definition at line 1695 of file vfetk.c.

7.3.3.29 Vfetk_PDE_Fu_v()

```
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:

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

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 1703 of file vfetk.c.

7.3.3.30 Vfetk_PDE_initAssemble()

```
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 1508 of file vfetk.c.

7.3.3.31 Vfetk_PDE_initElement()

```
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)

7.3.3.32 Vfetk_PDE_initFace()

```
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 1559 of file vfetk.c.

7.3.3.33 Vfetk_PDE_initPoint()

```
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

7.3.3.34 Vfetk_PDE_Ju()

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

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

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

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 2003 of file vfetk.c.

7.3.3.35 Vfetk_PDE_mapBoundary()

```
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

7.3.3.36 Vfetk_PDE_markSimplex()

```
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

7.3.3.37 Vfetk_PDE_oneChart()

```
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

7.3.3.38 Vfetk_PDE_simplexBasisForm()

```
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 2203 of file vfetk.c.

7.3.3.39 Vfetk_PDE_simplexBasisInit()

```
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?

Parameters

ndof	Set to the number of degrees of freedom
dof	Set to degree of freedom per v/e/f/s

Definition at line 2140 of file vfetk.c.

7.3.3.40 Vfetk_PDE_u_D()

```
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 1867 of file vfetk.c.

7.3.3.41 Vfetk_PDE_u_T()

```
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 1886 of file vfetk.c.

7.3.3.42 Vfetk_qfEnergy()

```
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: \f[ G = \sum_i q_i u(r_i) \f]
and return the result in units of \f$k_B T\f$.  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_BT.

Parameters

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

Definition at line 732 of file vfetk.c.

7.3.3.43 Vfetk_readMesh()

```
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

7.3.3.44 Vfetk_setAtomColors()

```
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 849 of file vfetk.c.

7.3.3.45 Vfetk_setParameters()

```
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 615 of file vfetk.c.

7.3.3.46 Vfetk_write()

```
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 2464 of file vfetk.c.

7.4 Vpee class

This class provides some functionality for error esimation in parallel.

Files

- file vpee.c

  Class Vpee methods.
- file vpee.h

  Contains declarations for class Vpee.

Data Structures

- struct sVpee

  Contains public data members for Vpee class/module.

Typedefs

- typedef struct sVpee Vpee

  Declaration of the Vpee class as the Vpee structure.

Functions

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

  Construct the Vpee object.
- VEXTERNC int Vpee_ctor2 (Vpee *thee, Gem *gm, int localPartID, int killFlag, double killParam)

  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.

7.4.1 Detailed Description

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

7.4.2 Function Documentation

7.4.2.1 Vpee_ctor()

```
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 93 of file vpee.c.

7.4.2.2 Vpee_ctor2()

```
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 114 of file vpee.c.

7.4.2.3 Vpee_dtor()

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

Object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of the Vpee object

Definition at line 225 of file vpee.c.

7.4.2.4 Vpee_dtor2()

```
VEXTERNC void Vpee_dtor2 (
            Vpee * thee )
```

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to object to be destroyed

Definition at line 240 of file vpee.c.

7.4.2.5 Vpee_markRefine()

```
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 250 of file vpee.c.

7.4.2.6 Vpee_numSS()

```
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 479 of file vpee.c.

7.5 APOLparm class

Parameter structure for APOL-specific variables from input files.

Files

- file apolparm.c

  Class APOLparm methods.
- file femparm.h

  Contains declarations for class APOLparm.

Data Structures

- struct sAPOLparm

  Parameter structure for APOL-specific variables from input files.

Typedefs

- typedef enum eAPOLparm_calcEnergy APOLparm_calcEnergy

  Define eAPOLparm_calcEnergy enumeration as APOLparm_calcEnergy.
- typedef enum eAPOLparm_calcForce APOLparm_calcForce

  Define eAPOLparm_calcForce enumeration as APOLparm_calcForce.
- typedef enum eAPOLparm_doCalc APOLparm_doCalc

  Define eAPOLparm_calcForce enumeration as APOLparm_calcForce.
- typedef struct sAPOLparm APOLparm

  Declaration of the APOLparm class as the APOLparm structure.

Enumerations

- enum eAPOLparm_calcEnergy { ACE_NO =0 , ACE_TOTAL =1 , ACE_COMPS =2 }

  Define energy calculation enumeration.
- enum eAPOLparm_calcForce { ACF_NO =0 , ACF_TOTAL =1 , ACF_COMPS =2 }

  Define force calculation enumeration.
- enum eAPOLparm_doCalc { ACD_NO =0 , ACD_YES =1 , ACD_ERROR =2 }

  Define force calculation enumeration.

Functions

- VEXTERNC APOLparm * APOLparm_ctor ()

  Construct APOLparm.
- VEXTERNC Vrc_Codes APOLparm_ctor2 (APOLparm *thee)

  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.

7.5.1 Detailed Description

Parameter structure for APOL-specific variables from input files.

7.5.2 Enumeration Type Documentation

7.5.2.1 eAPOLparm_calcEnergy

enum eAPOLparm_calcEnergy
Define energy calculation enumeration.

Enumerator

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

Definition at line 79 of file apolparm.h.

7.5.2.2 eAPOLparm_calcForce

enum eAPOLparm_calcForce
Define force calculation enumeration.

Enumerator

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

Definition at line 95 of file apolparm.h.

7.5.2.3 eAPOLparm_doCalc

enum eAPOLparm_doCalc
Define force calculation enumeration.

Enumerator

ACD_NO	Do not perform calculation
ACD_YES	Perform calculations
ACD_ERROR	Error setting up calculation

Definition at line 111 of file apolparm.h.

7.5.3 Function Documentation

7.5.3.1 APOLparm_check()

```
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 179 of file apolparm.c.

7.5.3.2 APOLparm_copy()

```
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 108 of file apolparm.c.

7.5.3.3 APOLparm_ctor()

```
VEXTERNC APOLparm* APOLparm_ctor ( )
```

Construct APOLparm.

Author

David Gohara

Returns

Newly allocated and initialized Vpmgp object

Definition at line 65 of file apolparm.c.

7.5.3.4 APOLparm_ctor2()

```
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 76 of file apolparm.c.

7.5.3.5 APOLparm_dtor()

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

Object destructor.

Author

David Gohara

Parameters

thee	Pointer to memory location of APOLparm object

Definition at line 167 of file apolparm.c.

7.5.3.6 APOLparm_dtor2()

```
VEXTERNC void APOLparm_dtor2 (
            APOLparm * thee )
```

FORTRAN stub for object destructor.

Author

David Gohara

Parameters

thee	Pointer to APOLparm object

Definition at line 177 of file apolparm.c.

7.6 BEMparm class

Parameter which holds useful parameters for generic multigrid calculations.

Files

- file bemparm.c

  Class BEMparm methods.

Data Structures

- struct sBEMparm

  Parameter structure for BEM-specific variables from input files.

Typedefs

- typedef enum eBEMparm_CalcType BEMparm_CalcType

  Declare BEMparm_CalcType type.
- typedef struct sBEMparm BEMparm

  Parameter structure for BEM-specific variables from input files.

Enumerations

- enum eBEMparm_CalcType { BCT_MANUAL =0 , BCT_NONE =1 }

  Calculation type.

Functions

- VEXTERNC BEMparm * BEMparm_ctor (BEMparm_CalcType type)

  Construct BEMparm object.
- VEXTERNC Vrc_Codes BEMparm_ctor2 (BEMparm *thee, BEMparm_CalcType type)

  FORTRAN stub to construct BEMparm object.
- VEXTERNC void BEMparm_dtor (BEMparm **thee)

  Object destructor.
- VEXTERNC void BEMparm_dtor2 (BEMparm *thee)

  FORTRAN stub for object destructor.
- VEXTERNC Vrc_Codes BEMparm_check (BEMparm *thee)

  Consistency check for parameter values stored in object.
- VEXTERNC Vrc_Codes BEMparm_parseToken (BEMparm *thee, char tok[VMAX_BUFSIZE], Vio *sock)

  Parse an MG keyword from an input file.

7.6.1 Detailed Description

Parameter which holds useful parameters for generic multigrid calculations.

7.6.2 Typedef Documentation

7.6.2.1 BEMparm

```
typedef struct sBEMparm BEMparm
```

Parameter structure for BEM-specific variables from input files.

Author

Nathan Baker and Todd Dolinsky and Weihua Geng

Note

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

7.6.3 Enumeration Type Documentation

7.6.3.1 eBEMparm_CalcType

```
enum eBEMparm_CalcType
```

Calculation type.

Enumerator

BCT_MANUAL	bem-manual
BCT_NONE	not defined

Definition at line 77 of file bemparm.h.

7.6.4 Function Documentation

7.6.4.1 BEMparm_check()

```
VEXTERNC Vrc_Codes BEMparm_check (
            BEMparm * thee )
```

Consistency check for parameter values stored in object.

Author

Nathan Baker

Parameters

thee	BEMparm object

Returns

Success enumeration

Definition at line 124 of file bemparm.c.

7.6.4.2 BEMparm_ctor()

```
VEXTERNC BEMparm* BEMparm_ctor (
            BEMparm_CalcType type )
```

Construct BEMparm object.

Author

Nathan Baker

Parameters

type	Type of BEM calculation

Returns

Newly allocated and initialized BEMparm object

Definition at line 66 of file bemparm.c.

7.6.4.3 BEMparm_ctor2()

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

FORTRAN stub to construct BEMparm object.

Author

Nathan Baker and Todd Dolinsky

Parameters

thee	Space for BEMparm object
type	Type of MG calculation

Returns

Success enumeration

Definition at line 77 of file bemparm.c.

7.6.4.4 BEMparm_dtor()

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

Object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of BEMparm object

Definition at line 114 of file bemparm.c.

7.6.4.5 BEMparm_dtor2()

```
VEXTERNC void BEMparm_dtor2 (
            BEMparm * thee )
```

FORTRAN stub for object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to BEMparm object

Definition at line 122 of file bemparm.c.

7.6.4.6 BEMparm_parseToken()

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

Parse an MG keyword from an input file.

Author

Nathan Baker and Todd Dolinsky

Parameters

thee	BEMparm 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 332 of file bemparm.c.

7.7 FEMparm class

Parameter structure for FEM-specific variables from input files.

Files

- file femparm.c

  Class FEMparm methods.
- file femparm.h

  Contains declarations for class APOLparm.

Data Structures

- struct sFEMparm

  Parameter structure for FEM-specific variables from input files.

Typedefs

- typedef enum eFEMparm_EtolType FEMparm_EtolType

  Declare FEparm_EtolType type.
- typedef enum eFEMparm_EstType FEMparm_EstType

  Declare FEMparm_EstType type.
- typedef enum eFEMparm_CalcType FEMparm_CalcType

  Declare FEMparm_CalcType type.
- typedef struct sFEMparm FEMparm

  Declaration of the FEMparm class as the FEMparm structure.

Enumerations

- enum eFEMparm_EtolType { FET_SIMP =0 , FET_GLOB =1 , FET_FRAC =2 }

  Adaptive 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.

7.7.1 Detailed Description

Parameter structure for FEM-specific variables from input files.

7.7.2 Typedef Documentation

7.7.2.1 FEMparm_EtolType

`typedef enum eFEMparm_EtolType FEMparm_EtolType`
Declare FEparm_EtolType type.

Author

Nathan Baker

Definition at line 1 of file femparm.h.

7.7.3 Enumeration Type Documentation

7.7.3.1 eFEMparm_CalcType

`enum eFEMparm_CalcType`
Calculation type.

Enumerator

FCT_MANUAL	fe-manual
FCT_NONE	unspecified

Definition at line 117 of file femparm.h.

7.7.3.2 eFEMparm_EstType

`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 98 of file femparm.h.

7.7.3.3 eFEMparm_EtolType

`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 79 of file femparm.h.

7.7.4 Function Documentation

7.7.4.1 FEMparm_check()

```
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 143 of file femparm.c.

7.7.4.2 FEMparm_copy()

```
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 100 of file femparm.c.

7.7.4.3 FEMparm_ctor()

```
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 65 of file femparm.c.

7.7.4.4 FEMparm_ctor2()

```
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 76 of file femparm.c.

7.7.4.5 FEMparm_dtor()

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

Object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of FEMparm object

Definition at line 133 of file femparm.c.

7.7.4.6 FEMparm_dtor2()

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

FORTRAN stub for object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to FEMparm object

Definition at line 141 of file femparm.c.

7.8 GEOFLOWparm class

Parameter which holds useful parameters for GEOFLOWeric multigrid calculations.

Files

- file geoflowparm.c

  Class GEOFLOWparm methods.
- file geoflowparm.h

  Contains declarations for class GEOFLOWparm.

Data Structures

- struct sGEOFLOWparm

  Parameter structure for GEOFLOW-specific variables from input files.

Typedefs

- typedef enum eGEOFLOWparm_CalcType GEOFLOWparm_CalcType

  Declare GEOFLOWparm_CalcType type.
- typedef struct sGEOFLOWparm GEOFLOWparm

  Parameter structure for GEOFLOW-specific variables from input files.

Enumerations

- enum eGEOFLOWparm_CalcType { GFCT_AUTO =1 }

  Calculation type.

Functions

- VEXTERNC GEOFLOWparm * GEOFLOWparm_ctor (GEOFLOWparm_CalcType type)

  Construct GEOFLOWparm object.
- VEXTERNC Vrc_Codes GEOFLOWparm_ctor2 (GEOFLOWparm *thee, GEOFLOWparm_CalcType type)

  FORTRAN stub to construct GEOFLOWparm object ?????????!!!!!!!
- VEXTERNC void GEOFLOWparm_dtor (GEOFLOWparm **thee)

  Object destructor.
- VEXTERNC void GEOFLOWparm_dtor2 (GEOFLOWparm *thee)

  FORTRAN stub for object destructor ?????????!!!!!!!!!!!!
- VEXTERNC Vrc_Codes GEOFLOWparm_check (GEOFLOWparm *thee)

  Consistency check for parameter values stored in object.
- VEXTERNC Vrc_Codes GEOFLOWparm_parseToken (GEOFLOWparm *thee, char tok[VMAX_BUFSIZE], Vio *sock)

  Parse an MG keyword from an input file.
- VEXTERNC void GEOFLOWparm_copy (GEOFLOWparm *thee, GEOFLOWparm *parm)

  copy GEOFLOWparm object int thee.

7.8.1 Detailed Description

Parameter which holds useful parameters for GEOFLOWeric multigrid calculations.

7.8.2 Typedef Documentation

7.8.2.1 GEOFLOWparm

`typedef struct sGEOFLOWparm GEOFLOWparm`

Parameter structure for GEOFLOW-specific variables from input files.

Author

Andrew Stevens, Kyle Monson

Note

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

7.8.3 Enumeration Type Documentation

7.8.3.1 eGEOFLOWparm_CalcType

`enum eGEOFLOWparm_CalcType`

Calculation type.

Enumerator

GFCT_AUTO	GEOFLOW-auto

Definition at line 77 of file geoflowparm.h.

7.8.4 Function Documentation

7.8.4.1 GEOFLOWparm_check()

```
VEXTERNC Vrc_Codes GEOFLOWparm_check (
            GEOFLOWparm * thee )
```

Consistency check for parameter values stored in object.

Author

Andrew Stevens, Kyle Monson

Parameters

thee	GEOFLOWparm object

Returns

Success enumeration

Definition at line 101 of file geoflowparm.c.

7.8.4.2 GEOFLOWparm_copy()

```
VEXTERNC void GEOFLOWparm_copy (
            GEOFLOWparm * thee,
            GEOFLOWparm * parm )
```

copy GEOFLOWparm object int thee.

Author

Parameters

thee	GEOFLOWparm object to be copied into
parm	GEOFLOWparm object.

Definition at line 127 of file geoflowparm.c.

7.8.4.3 GEOFLOWparm_ctor()

```
VEXTERNC GEOFLOWparm* GEOFLOWparm_ctor (
            GEOFLOWparm_CalcType type )
```

Construct GEOFLOWparm object.

Author

Andrew Stevens, Kyle Monson

Parameters

type	Type of GEOFLOW calculation

Returns

Newly allocated and initialized GEOFLOWparm object

Definition at line 66 of file geoflowparm.c.

7.8.4.4 GEOFLOWparm_ctor2()

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

FORTRAN stub to construct GEOFLOWparm object ?????????!!!!!!!

Author

Andrew Stevens, Kyle Monson

Parameters

thee	Space for GEOFLOWparm object
type	Type of MG calculation

Returns

Success enumeration

Definition at line 77 of file geoflowparm.c.

7.8.4.5 GEOFLOWparm_dtor()

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

Object destructor.

Author

Andrew Stevens, Kyle Monson

Parameters

thee	Pointer to memory location of GEOFLOWparm object

Definition at line 91 of file geoflowparm.c.

7.8.4.6 GEOFLOWparm_dtor2()

```
VEXTERNC void GEOFLOWparm_dtor2 (
            GEOFLOWparm * thee )
```

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

Author

Andrew Stevens, Kyle Monson

Parameters

thee	Pointer to GEOFLOWparm object

Definition at line 99 of file geoflowparm.c.

7.8.4.7 GEOFLOWparm_parseToken()

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

Parse an MG keyword from an input file.

Author

Andrew Stevens, Kyle Monson

Parameters

thee	GEOFLOWparm 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 212 of file geoflowparm.c.

7.9 MGparm class

Parameter which holds useful parameters for generic multigrid calculations.

Files

- file mgparm.c

  Class MGparm methods.

- file mgparm.h

  Contains declarations for class MGparm.

Data Structures

- struct sMGparm

  Parameter structure for MG-specific variables from input files.

Typedefs

- typedef enum eMGparm_CalcType MGparm_CalcType

  Declare MGparm_CalcType type.
- typedef enum eMGparm_CentMeth MGparm_CentMeth

  Declare MGparm_CentMeth type.
- typedef struct sMGparm MGparm

  Declaration of the MGparm class as the MGparm structure.

Enumerations

- enum eMGparm_CalcType {
  MCT_MANUAL =0 , MCT_AUTO =1 , MCT_PARALLEL =2 , MCT_DUMMY =3 ,
  MCT_NONE =4 }

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

  Centering method.

Functions

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

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

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

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

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

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

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

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

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

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

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

Set center z-coordinate.

- VEXTERNC double MGparm_getCenterX (MGparm *thee)

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

  Get center y-coordinate.
- VEXTERNC double MGparm_getCenterZ (MGparm *thee)

  Get center z-coordinate.
- VEXTERNC MGparm * MGparm_ctor (MGparm_CalcType type)

  Construct MGparm object.
- VEXTERNC Vrc_Codes MGparm_ctor2 (MGparm *thee, MGparm_CalcType type)

  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.

7.9.1 Detailed Description

Parameter which holds useful parameters for generic multigrid calculations.

7.9.2 Enumeration Type Documentation

7.9.2.1 eMGparm_CalcType

`enum eMGparm_CalcType`
Calculation type.

Enumerator

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

Definition at line 77 of file mgparm.h.

7.9.2.2 eMGparm_CentMeth

`enum eMGparm_CentMeth`
Centering method.

Enumerator

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

Definition at line 95 of file mgparm.h.

7.9.3 Function Documentation

7.9.3.1 APOLparm_parseToken()

```
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 577 of file apolparm.c.

7.9.3.2 FEMparm_parseToken()

```
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 431 of file femparm.c.

7.9.3.3 MGparm_check()

```
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.

7.9.3.4 MGparm_copy()

```
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.

7.9.3.5 MGparm_ctor()

```
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.

7.9.3.6 MGparm_ctor2()

```
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.

7.9.3.7 MGparm_dtor()

```
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.

7.9.3.8 MGparm_dtor2()

```
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.

7.9.3.9 MGparm_getCenterX()

```
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.

7.9.3.10 MGparm_getCenterY()

```
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.

7.9.3.11 MGparm_getCenterZ()

```
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.

7.9.3.12 MGparm_getHx()

```
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.

7.9.3.13 MGparm_getHy()

```
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.

7.9.3.14 MGparm_getHz()

```
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.

7.9.3.15 MGparm_getNx()

```
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.

7.9.3.16 MGparm_getNy()

```
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.

7.9.3.17 MGparm_getNz()

```
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.

7.9.3.18 MGparm_parseToken()

```
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.

7.9.3.19 MGparm_setCenterX()

```
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.

7.9.3.20 MGparm_setCenterY()

```
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.

7.9.3.21 MGparm_setCenterZ()

```
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.

7.10 NOsh class

Class for parsing for fixed format input files.

Files

- file nosh.c

  Class NOsh methods.
- file nosh.h

  Contains declarations for class NOsh.

Data Structures

- struct sNOsh_calc

  Calculation class for use when parsing fixed format input files.
- struct sNOsh

  Class for parsing fixed format input files.

Macros

- #define NOSH_MAXMOL 20

  Maximum number of molecules in a run.
- #define NOSH_MAXCALC 20

  Maximum number of calculations in a run.
- #define NOSH_MAXPRINT 20

  Maximum number of PRINT statements in a run.
- #define NOSH_MAXPOP 20

  Maximum number of operations in a PRINT statement.

Typedefs

- typedef enum eNOsh_MolFormat NOsh_MolFormat

  Declare NOsh_MolFormat type.
- typedef enum eNOsh_CalcType NOsh_CalcType

  Declare NOsh_CalcType type.
- typedef enum eNOsh_ParmFormat NOsh_ParmFormat

  Declare NOsh_ParmFormat type.
- typedef enum eNOsh_PrintType NOsh_PrintType

  Declare NOsh_PrintType type.
- typedef struct sNOsh NOsh

  Declaration of the NOsh class as the NOsh structure.
- typedef struct sNOsh_calc NOsh_calc

  Declaration of the NOsh_calc class as the NOsh_calc structure.

Enumerations

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

  Molecule file format types.
- enum eNOsh_CalcType {
  NCT_MG =0 , NCT_FEM =1 , NCT_APOL =2 , NCT_BEM =3 ,
  NCT_GEOFLOW =4 , NCT_PBAM =5 , NCT_PBSAM =6 }

  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.

7.10.1 Detailed Description

Class for parsing for fixed format input files.

7.10.2 Enumeration Type Documentation

7.10.2.1 eNOsh_CalcType

enum eNOsh_CalcType
NOsh calculation types.

Enumerator

NCT_MG	Multigrid
NCT_FEM	Finite element
NCT_APOL	non-polar
NCT_BEM	Boundary element (TABI)
NCT_GEOFLOW	Geometric flow
NCT_PBAM	Analytical Poisson-Boltzmann Solver
NCT_PBSAM	Semi-Analytical Poisson-Boltzmann Solver

Definition at line 117 of file nosh.h.

7.10.2.2 eNOsh_MolFormat

enum eNOsh_MolFormat
Molecule file format types.

Enumerator

NMF_PQR	PQR format
NMF_PDB	PDB format
NMF_XML	XML format

Definition at line 101 of file nosh.h.

7.10.2.3 eNOsh_ParmFormat

enum eNOsh_ParmFormat
Parameter file format types.

Enumerator

NPF_FLAT	Flat-file format
NPF_XML	XML format

Definition at line 137 of file nosh.h.

7.10.2.4 eNOsh_PrintType

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 152 of file nosh.h.

7.10.3 Function Documentation

7.10.3.1 NOsh_apol2calc()

```
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 282 of file nosh.c.

7.10.3.2 NOsh_calc_copy()

```
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 467 of file nosh.c.

7.10.3.3 NOsh_calc_ctor()

```
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 374 of file nosh.c.

7.10.3.4 NOsh_calc_dtor()

```
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 423 of file nosh.c.

7.10.3.5 NOsh_ctor()

```
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 308 of file nosh.c.

7.10.3.6 NOsh_ctor2()

```
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 319 of file nosh.c.

7.10.3.7 NOsh_dtor()

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

Object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of NOsh object

Definition at line 354 of file nosh.c.

7.10.3.8 NOsh_dtor2()

```
VEXTERNC void NOsh_dtor2 (
            NOsh * thee )
```

FORTRAN stub for object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to NOsh object

Definition at line 362 of file nosh.c.

7.10.3.9 NOsh_elec2calc()

```
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 276 of file nosh.c.

7.10.3.10 NOsh_elecname()

```
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 288 of file nosh.c.

7.10.3.11 NOsh_getCalc()

```
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 235 of file nosh.c.

7.10.3.12 NOsh_getChargefmt()

```
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 255 of file nosh.c.

7.10.3.13 NOsh_getChargepath()

```
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 230 of file nosh.c.

7.10.3.14 NOsh_getDielfmt()

```
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 240 of file nosh.c.

7.10.3.15 NOsh_getDielXpath()

```
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 205 of file nosh.c.

7.10.3.16 NOsh_getDielYpath()

```
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 210 of file nosh.c.

7.10.3.17 NOsh_getDielZpath()

```
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 215 of file nosh.c.

7.10.3.18 NOsh_getKappafmt()

```
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 245 of file nosh.c.

7.10.3.19 NOsh_getKappapath()

```
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 220 of file nosh.c.

7.10.3.20 NOsh_getMolpath()

```
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 200 of file nosh.c.

7.10.3.21 NOsh_getPotfmt()

```
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 250 of file nosh.c.

7.10.3.22 NOsh_getPotpath()

```
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 225 of file nosh.c.

7.10.3.23 NOsh_parseInput()

```
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 513 of file nosh.c.

7.10.3.24 NOsh_parseInputFile()

```
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 498 of file nosh.c.

7.10.3.25 NOsh_printCalc()

```
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 301 of file nosh.c.

7.10.3.26 NOsh_printNarg()

```
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 270 of file nosh.c.

7.10.3.27 NOsh_printOp()

```
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 294 of file nosh.c.

7.10.3.28 NOsh_printWhat()

```
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 264 of file nosh.c.

7.10.3.29 NOsh_setupApolCalc()

```
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 1469 of file nosh.c.

7.10.3.30 NOsh_setupElecCalc()

```
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 1374 of file nosh.c.

7.11 PBAMparm class

Parameter which holds useful parameters for Poisson-boltzmann analytical method calculations.

Files

- file pbamparm.c

  Class PBAMparm methods.

- file pbamparm.h

  Contains declarations for class PBAMparm.

Data Structures

- struct sPBAMparm

  Parameter structure for PBAM-specific variables from input files.

Macros

- #define CHR_MAXLEN 1000

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

Typedefs

- typedef enum ePBAMparm_CalcType PBAMparm_CalcType

  Declare PBAMparm_CalcType type.
- typedef struct sPBAMparm PBAMparm

  Parameter structure for PBAM-specific variables from input files.

Enumerations

- enum ePBAMparm_CalcType { PBAMCT_AUTO =1 }

  Calculation type.

Functions

- VEXTERNC PBAMparm * PBAMparm_ctor (PBAMparm_CalcType type)

  Construct PBAMparm object.
- VEXTERNC Vrc_Codes PBAMparm_ctor2 (PBAMparm *thee, PBAMparm_CalcType type)

  FORTRAN stub to construct PBAMparm object ?????????!!!!!!!
- VEXTERNC void PBAMparm_dtor (PBAMparm **thee)

  Object destructor.
- VEXTERNC void PBAMparm_dtor2 (PBAMparm *thee)

  FORTRAN stub for object destructor ?????????!!!!!!!!!!!!
- VEXTERNC Vrc_Codes PBAMparm_check (PBAMparm *thee)

  Consistency check for parameter values stored in object.
- VEXTERNC Vrc_Codes PBAMparm_parseToken (PBAMparm *thee, char tok[VMAX_BUFSIZE], Vio *sock)

  Parse an MG keyword from an input file.
- VEXTERNC void PBAMparm_copy (PBAMparm *thee, PBAMparm *parm)

  copy PBAMparm object int thee.
- VPRIVATE Vrc_Codes PBAMparm_parseSalt (PBAMparm *thee, Vio *sock)

  Find salt conc and save it as a structure variable.
- VPRIVATE Vrc_Codes PBAMparm_parseRunType (PBAMparm *thee, Vio *sock)

  Find runType and save it as a structure variable.
- VPRIVATE Vrc_Codes PBAMparm_parseRunName (PBAMparm *thee, Vio *sock)

  Find runName and save it as a structure variable.
- VPRIVATE Vrc_Codes PBAMparm_parseRandorient (PBAMparm *thee, Vio *sock)

  Find randomorientation flag and save it as a boolean.

- VPRIVATE Vrc_Codes PBAMparm_parsePBCS (PBAMparm *thee, Vio *sock)

  Find PBC flag and save the type and the boxlength.
- VPRIVATE Vrc_Codes PBAMparm_parseUnits (PBAMparm *thee, Vio *sock)

  Find units flag and save units.
- VPRIVATE Vrc_Codes PBAMparm_parse3Dmap (PBAMparm *thee, Vio *sock)

  Find 3D map filename and save it.
- VPRIVATE Vrc_Codes PBAMparm_parseGrid2D (PBAMparm *thee, Vio *sock)

  Find 2D grid filename and save it.
- VPRIVATE Vrc_Codes PBAMparm_parseDX (PBAMparm *thee, Vio *sock)

  Find DX filename and save it.
- VPRIVATE Vrc_Codes PBAMparm_parseGridPts (PBAMparm *thee, Vio *sock)

  Find Grid points and save them.
- VPRIVATE Vrc_Codes PBAMparm_parseTermcombine (PBAMparm *thee, Vio *sock)

  Find Termination logic and save it.
- VPRIVATE Vrc_Codes PBAMparm_parseDiff (PBAMparm *thee, Vio *sock)

  Find diffusion coeffs for each molecule and save them.
- VPRIVATE Vrc_Codes PBAMparm_parseXYZ (PBAMparm *thee, Vio *sock)

  Find xyz files for each molecule for each traj and save them.

7.11.1 Detailed Description

Parameter which holds useful parameters for Poisson-boltzmann analytical method calculations.

7.11.2 Typedef Documentation

7.11.2.1 PBAMparm

`typedef struct sPBAMparm PBAMparm`

Parameter structure for PBAM-specific variables from input files.

Author

Andrew Stevens, Kyle Monson

Note

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

7.11.3 Enumeration Type Documentation

7.11.3.1 ePBAMparm_CalcType

`enum ePBAMparm_CalcType`

Calculation type.

Enumerator

PBAMCT_AUTO	PBAM-auto

Definition at line 84 of file pbamparm.h.

7.11.4 Function Documentation

7.11.4.1 PBAMparm_check()

```
VEXTERNC Vrc_Codes PBAMparm_check (
            PBAMparm * thee )
```

Consistency check for parameter values stored in object.

Author

Andrew Stevens, Kyle Monson

Parameters

thee	PBAMparm object

Returns

Success enumeration

Definition at line 132 of file pbamparm.c.

7.11.4.2 PBAMparm_copy()

```
VEXTERNC void PBAMparm_copy (
            PBAMparm * thee,
            PBAMparm * parm )
```

copy PBAMparm object int thee.

Author

Parameters

thee	PBAMparm object to be copied into
parm	PBAMparm object.

Definition at line 157 of file pbamparm.c.

7.11.4.3 PBAMparm_ctor()

```
VEXTERNC PBAMparm* PBAMparm_ctor (
            PBAMparm_CalcType type )
```

Construct PBAMparm object.

Author

Andrew Stevens, Kyle Monson

Parameters

type	Type of PBAM calculation

Returns

Newly allocated and initialized PBAMparm object

Definition at line 66 of file pbamparm.c.

7.11.4.4 PBAMparm_ctor2()

```
VEXTERNC Vrc_Codes PBAMparm_ctor2 (
            PBAMparm * thee,
            PBAMparm_CalcType type )
```

FORTRAN stub to construct PBAMparm object ?????????!!!!!!!

Author

Andrew Stevens, Kyle Monson

Parameters

thee	Space for PBAMparm object
type	Type of MG calculation

Returns

Success enumeration

Definition at line 77 of file pbamparm.c.

7.11.4.5 PBAMparm_dtor()

```
VEXTERNC void PBAMparm_dtor (
            PBAMparm ** thee )
```

Object destructor.

Author

Andrew Stevens, Kyle Monson

Parameters

thee	Pointer to memory location of PBAMparm object

Definition at line 122 of file pbamparm.c.

7.11.4.6 PBAMparm_dtor2()

```
VEXTERNC void PBAMparm_dtor2 (
            PBAMparm * thee )
```

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

Author

Andrew Stevens, Kyle Monson

Parameters

thee	Pointer to PBAMparm object

Definition at line 130 of file pbamparm.c.

7.11.4.7 PBAMparm_parse3Dmap()

```
VPRIVATE Vrc_Codes PBAMparm_parse3Dmap (
            PBAMparm * thee,
            Vio * sock )
```

Find 3D map filename and save it.

Author

Parameters

thee	PBAMparm object to be copied into
sock	The stream from which parameter is taken

Definition at line 371 of file pbamparm.c.

7.11.4.8 PBAMparm_parseDiff()

```
VPRIVATE Vrc_Codes PBAMparm_parseDiff (
            PBAMparm * thee,
            Vio * sock )
```

Find diffusion coeffs for each molecule and save them.

Author

Parameters

thee	PBAMparm object to be copied into
sock	The stream from which parameter is taken

Definition at line 477 of file pbamparm.c.

7.11.4.9 PBAMparm_parseDX()

```
VPRIVATE Vrc_Codes PBAMparm_parseDX (
            PBAMparm * thee,
            Vio * sock )
```

Find DX filename and save it.

Author

Parameters

thee	PBAMparm object to be copied into
sock	The stream from which parameter is taken

Definition at line 426 of file pbamparm.c.

7.11.4.10 PBAMparm_parseGrid2D()

```
VPRIVATE Vrc_Codes PBAMparm_parseGrid2D (
            PBAMparm * thee,
            Vio * sock )
```

Find 2D grid filename and save it.

Author

Parameters

thee	PBAMparm object to be copied into
sock	The stream from which parameter is taken

Definition at line 391 of file pbamparm.c.

7.11.4.11 PBAMparm_parseGridPts()

```
VPRIVATE Vrc_Codes PBAMparm_parseGridPts (
            PBAMparm * thee,
            Vio * sock )
```

Find Grid points and save them.

Author

Parameters

thee	PBAMparm object to be copied into
sock	The stream from which parameter is taken

Definition at line 351 of file pbamparm.c.

7.11.4.12 PBAMparm_parsePBCS()

```
VPRIVATE Vrc_Codes PBAMparm_parsePBCS (
            PBAMparm * thee,
            Vio * sock )
```

Find PBC flag and save the type and the boxlength.

Author

Parameters

thee	PBAMparm object to be copied into
sock	The stream from which parameter is taken

Definition at line 310 of file pbamparm.c.

7.11.4.13 PBAMparm_parseRandorient()

```
VPRIVATE Vrc_Codes PBAMparm_parseRandorient (
            PBAMparm * thee,
            Vio * sock )
```

Find randomorientation flag and save it as a boolean.

Author

Parameters

thee	PBAMparm object to be copied into
sock	The stream from which parameter is taken

Definition at line 304 of file pbamparm.c.

7.11.4.14 PBAMparm_parseRunName()

```
VPRIVATE Vrc_Codes PBAMparm_parseRunName (
            PBAMparm * thee,
            Vio * sock )
```

Find runName and save it as a structure variable.

Author

Parameters

thee	PBAMparm object to be copied into
sock	The stream from which parameter is taken

Definition at line 289 of file pbamparm.c.

7.11.4.15 PBAMparm_parseRunType()

```
VPRIVATE Vrc_Codes PBAMparm_parseRunType (
            PBAMparm * thee,
            Vio * sock )
```

Find runType and save it as a structure variable.

Author

Parameters

thee	PBAMparm object to be copied into
sock	The stream from which parameter is taken

Definition at line 271 of file pbamparm.c.

7.11.4.16 PBAMparm_parseSalt()

```
VPRIVATE Vrc_Codes PBAMparm_parseSalt (
            PBAMparm * thee,
            Vio * sock )
```

Find salt conc and save it as a structure variable.

Author

Parameters

thee	PBAMparm object to be copied into
parm	The stream from which parameter is taken

Definition at line 252 of file pbamparm.c.

7.11.4.17 PBAMparm_parseTermcombine()

```
VPRIVATE Vrc_Codes PBAMparm_parseTermcombine (
            PBAMparm * thee,
            Vio * sock )
```

Find Termination logic and save it.

Author

Parameters

thee	PBAMparm object to be copied into
sock	The stream from which parameter is taken

Definition at line 444 of file pbamparm.c.

7.11.4.18 PBAMparm_parseToken()

```
VEXTERNC Vrc_Codes PBAMparm_parseToken (
            PBAMparm * thee,
            char tok[VMAX_BUFSIZE],
            Vio * sock )
```

Parse an MG keyword from an input file.

Author

Andrew Stevens, Kyle Monson

Parameters

thee	PBAMparm 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 662 of file pbamparm.c.

7.11.4.19 PBAMparm_parseUnits()

```
VPRIVATE Vrc_Codes PBAMparm_parseUnits (
            PBAMparm * thee,
            Vio * sock )
```

Find units flag and save units.

Author

Parameters

thee	PBAMparm object to be copied into
sock	The stream from which parameter is taken

Definition at line 336 of file pbamparm.c.

7.11.4.20 PBAMparm_parseXYZ()

```
VPRIVATE Vrc_Codes PBAMparm_parseXYZ (
            PBAMparm * thee,
            Vio * sock )
```

Find xyz files for each molecule for each traj and save them.

Author

Parameters

thee	PBAMparm object to be copied into
sock	The stream from which parameter is taken

Definition at line 632 of file pbamparm.c.

7.12 PBEparm class

Parameter structure for PBE variables independent of solver.

Files

- file pbeparm.c

  Class PBEparm methods.
- file pbeparm.h

  Contains declarations for class PBEparm.

Data Structures

- struct sPBEparm

  Parameter structure for PBE variables from input files.

Macros

- #define PBEPARM_MAXWRITE 20

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

Typedefs

- typedef enum ePBEparm_calcEnergy PBEparm_calcEnergy

  Define ePBEparm_calcEnergy enumeration as PBEparm_calcEnergy.
- typedef enum ePBEparm_calcForce PBEparm_calcForce

  Define ePBEparm_calcForce enumeration as PBEparm_calcForce.
- typedef struct sPBEparm PBEparm

  Declaration of the PBEparm class as the PBEparm structure.

Enumerations

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

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

  Define force calculation enumeration.

Functions

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

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

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

  Get radius (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.

7.12.1 Detailed Description

Parameter structure for PBE variables independent of solver.

7.12.2 Enumeration Type Documentation

7.12.2.1 ePBEparm_calcEnergy

`enum ePBEparm_calcEnergy`

Define energy calculation enumeration.

Enumerator

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

Definition at line 81 of file pbeparm.h.

7.12.2.2 ePBEparm_calcForce

`enum ePBEparm_calcForce`

Define force calculation enumeration.

Enumerator

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

Definition at line 97 of file pbeparm.h.

7.12.3 Function Documentation

7.12.3.1 PBEparm_check()

```
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 183 of file pbeparm.c.

7.12.3.2 PBEparm_copy()

```
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 283 of file pbeparm.c.

7.12.3.3 PBEparm_ctor()

```
VEXTERNC PBEparm* PBEparm_ctor ( )
```

Construct PBEparm object.

Author

Nathan Baker

Returns

Newly allocated and initialized PBEparm object

Definition at line 104 of file pbeparm.c.

7.12.3.4 PBEparm_ctor2()

```
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 115 of file pbeparm.c.

7.12.3.5 PBEparm_dtor()

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

Object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of object

Definition at line 173 of file pbeparm.c.

7.12.3.6 PBEparm_dtor2()

```
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 181 of file pbeparm.c.

7.12.3.7 PBEparm_getIonCharge()

```
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 65 of file pbeparm.c.

7.12.3.8 PBEparm_getIonConc()

```
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 71 of file pbeparm.c.

7.12.3.9 PBEparm_getIonRadius()

```
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 77 of file pbeparm.c.

7.12.3.10 PBEparm_parseToken()

```
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 1215 of file pbeparm.c.

7.13 PBSAMparm class

Parameter which holds useful parameters for Poisson-boltzmann analytical method calculations.

Files

- file pbsamparm.c

  Class PBSAMparm methods.
- file pbsamparm.h

  Contains declarations for class PBSAMparm.

Data Structures

- struct sPBSAMparm

  Parameter structure for PBSAM-specific variables from input files.

Macros

- #define CHR_MAXLEN 1000

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

Typedefs

- typedef enum ePBSAMparm_CalcType PBSAMparm_CalcType

  Declare PBSAMparm_CalcType type.
- typedef struct sPBSAMparm PBSAMparm

  Parameter structure for PBSAM-specific variables from input files.

Enumerations

- enum ePBSAMparm_CalcType { PBSAMCT_AUTO =1 }

  Calculation type.

Functions

- VEXTERNC PBSAMparm * PBSAMparm_ctor (PBSAMparm_CalcType type)

  Construct PBSAMparm object.
- VEXTERNC Vrc_Codes PBSAMparm_ctor2 (PBSAMparm *thee, PBSAMparm_CalcType type)

  FORTRAN stub to construct PBSAMparm object ?????????!!!!!!!
- VEXTERNC void PBSAMparm_dtor (PBSAMparm **thee)

  Object destructor.
- VEXTERNC void PBSAMparm_dtor2 (PBSAMparm *thee)

  FORTRAN stub for object destructor ?????????!!!!!!!!!!!
- VEXTERNC Vrc_Codes PBSAMparm_check (PBSAMparm *thee)

  Consistency check for parameter values stored in object.
- VEXTERNC Vrc_Codes PBSAMparm_parseToken (PBSAMparm *thee, char tok[VMAX_BUFSIZE], Vio *sock)

  Parse an MG keyword from an input file.
- VEXTERNC void PBSAMparm_copy (PBSAMparm *thee, PBSAMparm *parm)

copy PBSAMparm object int thee.

- VPRIVATE Vrc_Codes PBSAMparm_parseTolsp (PBSAMparm *thee, Vio *sock)

  Find sphere tolerance for coarse-graining.
- VPRIVATE Vrc_Codes PBSAMparm_parseSurf (PBSAMparm *thee, Vio *sock)

  Find vertex files for each molecule and save them.
- VPRIVATE Vrc_Codes PBSAMparm_parseImat (PBSAMparm *thee, Vio *sock)

  Find IMAT files for each molecule and save them.
- VPRIVATE Vrc_Codes PBSAMparm_parseExp (PBSAMparm *thee, Vio *sock)

  Find expansion files for each molecule and save them.
- VPRIVATE Vrc_Codes PBSAMparm_parseMSMS (PBSAMparm *thee, Vio *sock)

  Find msms flag for if MSMS is to be run.

7.13.1 Detailed Description

Parameter which holds useful parameters for Poisson-boltzmann analytical method calculations.

7.13.2 Typedef Documentation

7.13.2.1 PBSAMparm

`typedef struct sPBSAMparm PBSAMparm`

Parameter structure for PBSAM-specific variables from input files.

Author

Lisa Felberg

Note

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

7.13.3 Enumeration Type Documentation

7.13.3.1 ePBSAMparm_CalcType

`enum ePBSAMparm_CalcType`

Calculation type.

Enumerator

PBSAMCT_AUTO	PBSAM-auto

Definition at line 84 of file pbsamparm.h.

7.13.4 Function Documentation

7.13.4.1 PBSAMparm_check()

```
VEXTERNC Vrc_Codes PBSAMparm_check (
            PBSAMparm * thee )
```

Consistency check for parameter values stored in object.

Author

Lisa Felberg

Parameters

thee	PBSAMparm object

Returns

Success enumeration

Definition at line 110 of file pbsamparm.c.

7.13.4.2 PBSAMparm_copy()

```
VEXTERNC void PBSAMparm_copy (
            PBSAMparm * thee,
            PBSAMparm * parm )
```

copy PBSAMparm object int thee.

Author

Parameters

thee	PBSAMparm object to be copied into
parm	PBSAMparm object.

Definition at line 135 of file pbsamparm.c.

7.13.4.3 PBSAMparm_ctor()

```
VEXTERNC PBSAMparm* PBSAMparm_ctor (
            PBSAMparm_CalcType type )
```

Construct PBSAMparm object.

Author

Lisa Felberg

Parameters

type	Type of PBSAM calculation

Returns

Newly allocated and initialized PBSAMparm object

Definition at line 66 of file pbsamparm.c.

7.13.4.4 PBSAMparm_ctor2()

```
VEXTERNC Vrc_Codes PBSAMparm_ctor2 (
            PBSAMparm * thee,
            PBSAMparm_CalcType type )
```

FORTRAN stub to construct PBSAMparm object ?????????!!!!!!!

Author

Lisa Felberg

Parameters

thee	Space for PBSAMparm object
type	Type of MG calculation

Returns

Success enumeration

Definition at line 77 of file pbsamparm.c.

7.13.4.5 PBSAMparm_dtor()

```
VEXTERNC void PBSAMparm_dtor (
            PBSAMparm ** thee )
```

Object destructor.

Author

Lisa Felberg

Parameters

thee	Pointer to memory location of PBSAMparm object

Definition at line 100 of file pbsamparm.c.

7.13.4.6 PBSAMparm_dtor2()

```
VEXTERNC void PBSAMparm_dtor2 (
            PBSAMparm * thee )
```

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

Author

Lisa Felberg

Parameters

thee	Pointer to PBSAMparm object

Definition at line 108 of file pbsamparm.c.

7.13.4.7 PBSAMparm_parseExp()

```
VPRIVATE Vrc_Codes PBSAMparm_parseExp (
            PBSAMparm * thee,
            Vio * sock )
```

Find expansion files for each molecule and save them.

Author

Parameters

thee	PBSAMparm object to be copied into
sock	The stream from which parameter is taken

Definition at line 218 of file pbsamparm.c.

7.13.4.8 PBSAMparm_parseImat()

```
VPRIVATE Vrc_Codes PBSAMparm_parseImat (
            PBSAMparm * thee,
            Vio * sock )
```

Find IMAT files for each molecule and save them.

Author

Parameters

thee	PBSAMparm object to be copied into
sock	The stream from which parameter is taken

Definition at line 203 of file pbsamparm.c.

7.13.4.9 PBSAMparm_parseMSMS()

```
VPRIVATE Vrc_Codes PBSAMparm_parseMSMS (
            PBSAMparm * thee,
            Vio * sock )
```

Find msms flag for if MSMS is to be run.

Author

Parameters

thee	PBSAMparm object to be copied into
sock	The stream from which parameter is taken

Definition at line 182 of file pbsamparm.c.

7.13.4.10 PBSAMparm_parseSurf()

```
VPRIVATE Vrc_Codes PBSAMparm_parseSurf (
            PBSAMparm * thee,
            Vio * sock )
```

Find vertex files for each molecule and save them.

Author

Parameters

thee	PBSAMparm object to be copied into
sock	The stream from which parameter is taken

Definition at line 166 of file pbsamparm.c.

7.13.4.11 PBSAMparm_parseToken()

```
VEXTERNC Vrc_Codes PBSAMparm_parseToken (
            PBSAMparm * thee,
            char tok[VMAX_BUFSIZE],
            Vio * sock )
```

Parse an MG keyword from an input file.

Author

Lisa Felberg

Parameters

thee	PBSAMparm 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 252 of file pbsamparm.c.

7.13.4.12 PBSAMparm_parseTolsp()

```
VPRIVATE Vrc_Codes PBSAMparm_parseTolsp (
             PBSAMparm * thee,
             Vio * sock )
```

Find sphere tolerance for coarse-graining.

Author

Parameters

thee	PBSAMparm object to be copied into
sock	The stream from which parameter is taken

Definition at line 232 of file pbsamparm.c.

7.14 Vacc class

Solvent- and ion-accessibility oracle.

Files

- file vacc.c

  Class Vacc methods.
- file vacc.h

  Contains declarations for class Vacc.

Data Structures

- struct sVaccSurf

  Surface object list of per-atom surface points.
- struct sVacc

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

Typedefs

- typedef struct sVaccSurf VaccSurf

  Declaration of the VaccSurf class as the VaccSurf structure.
- typedef struct sVacc Vacc

  Declaration of the Vacc class as the Vacc structure.

Functions

- VEXTERNC unsigned long int Vacc_memChk (Vacc *thee)

  Get number of bytes in this object and its members.
- VEXTERNC VaccSurf * VaccSurf_ctor (Vmem *mem, double probe_radius, int nsphere)

  Allocate and construct the surface object; do not assign surface points to positions.
- VEXTERNC int VaccSurf_ctor2 (VaccSurf *thee, Vmem *mem, double probe_radius, int nsphere)

  Construct the surface object using previously allocated memory; do not assign surface points to positions.
- VEXTERNC void VaccSurf_dtor (VaccSurf **thee)

  Destroy the surface object and free its memory.
- VEXTERNC void VaccSurf_dtor2 (VaccSurf *thee)

  Destroy the surface object.
- VEXTERNC VaccSurf * VaccSurf_refSphere (Vmem *mem, int npts)

  Set up an array of points for a reference sphere of unit radius.
- VEXTERNC VaccSurf * Vacc_atomSurf (Vacc *thee, Vatom *atom, VaccSurf *ref, double probe_radius)

  Set up an array of points corresponding to the SAS due to a particular atom.
- VEXTERNC Vacc * Vacc_ctor (Valist *alist, Vclist *clist, double surf_density)

  Construct the accessibility object.
- VEXTERNC int Vacc_ctor2 (Vacc *thee, Valist *alist, Vclist *clist, double surf_density)

  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.

7.14.1 Detailed Description

Solvent- and ion-accessibility oracle.

7.14.2 Function Documentation

7.14.2.1 Vacc_atomdSASA()

```
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 1320 of file vacc.c.

7.14.2.2 Vacc_atomdSAV()

```
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 1200 of file vacc.c.

7.14.2.3 Vacc_atomSASA()

```
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 780 of file vacc.c.

7.14.2.4 Vacc_atomSASPoints()

```
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 982 of file vacc.c.

7.14.2.5 Vacc_atomSurf()

```
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 868 of file vacc.c.

7.14.2.6 Vacc_ctor()

```
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 132 of file vacc.c.

7.14.2.7 Vacc_ctor2()

```
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 213 of file vacc.c.

7.14.2.8 Vacc_dtor()

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

Destroy object.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of object

Definition at line 245 of file vacc.c.

7.14.2.9 Vacc_dtor2()

```
VEXTERNC void Vacc_dtor2 (
             Vacc * thee )
```

FORTRAN stub to destroy object.

Author

Nathan Baker

Parameters

thee	Pointer to object

Definition at line 255 of file vacc.c.

7.14.2.10 Vacc_fastMolAcc()

```
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 637 of file vacc.c.

7.14.2.11 Vacc_ivdwAcc()

```
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)

7.14.2.12 Vacc_memChk()

```
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 63 of file vacc.c.

7.14.2.13 Vacc_molAcc()

```
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 608 of file vacc.c.

7.14.2.14 Vacc_SASA()

```
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 713 of file vacc.c.

7.14.2.15 Vacc_splineAcc()

```
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 528 of file vacc.c.

7.14.2.16 Vacc_splineAccAtom()

```
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 438 of file vacc.c.

7.14.2.17 Vacc_splineAccGrad()

```
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)

Parameters

infrad	Inflation radius (Å) for ion access.
grad	3-vector set to gradient of accessibility

Definition at line 561 of file vacc.c.

7.14.2.18 Vacc_splineAccGradAtomNorm()

```
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 316 of file vacc.c.

7.14.2.19 Vacc_splineAccGradAtomNorm3()

```
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 1099 of file vacc.c.

7.14.2.20 Vacc_splineAccGradAtomNorm4()

```
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 1006 of file vacc.c.

7.14.2.21 Vacc_splineAccGradAtomUnnorm()

```
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 377 of file vacc.c.

7.14.2.22 Vacc_totalAtomdSASA()

```
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 1389 of file vacc.c.

7.14.2.23 Vacc_totalAtomdSAV()

```
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 1448 of file vacc.c.

7.14.2.24 Vacc_totalSASA()

```
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 774 of file vacc.c.

7.14.2.25 Vacc_totalSAV()

```
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 1503 of file vacc.c.

7.14.2.26 Vacc_vdwAcc()

```
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

7.14.2.27 Vacc_wcaEnergy()

```
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 1721 of file vacc.c.

7.14.2.28 Vacc_wcaEnergyAtom()

```
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 1580 of file vacc.c.

7.14.2.29 Vacc_wcaForceAtom()

```
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 1756 of file vacc.c.

7.14.2.30 VaccSurf_ctor()

```
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 803 of file vacc.c.

7.14.2.31 VaccSurf_ctor2()

```
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 813 of file vacc.c.

7.14.2.32 VaccSurf_dtor()

```
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 839 of file vacc.c.

7.14.2.33 VaccSurf_dtor2()

```
VEXTERNC void VaccSurf_dtor2 (
            VaccSurf * thee )
```

Destroy the surface object.

Author

Nathan Baker

Parameters

thee	Object to be destroyed

Definition at line 853 of file vacc.c.

7.14.2.34 VaccSurf_refSphere()

```
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 926 of file vacc.c.

7.15 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.

7.15.1 Detailed Description

Container class for list of atom objects.

7.15.2 Function Documentation

7.15.2.1 Valist_ctor()

```
VEXTERNC Valist* Valist_ctor ( )
```

Construct the atom list object.

Author

Nathan Baker

Returns

Pointer to newly allocated (empty) atom list

Definition at line 138 of file valist.c.

7.15.2.2 Valist_ctor2()

```
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 155 of file valist.c.

7.15.2.3 Valist_dtor()

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

Destroys atom list object.

Author

Nathan Baker

Parameters

thee	Pointer to storage for atom list

Definition at line 167 of file valist.c.

7.15.2.4 Valist_dtor2()

```
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 176 of file valist.c.

7.15.2.5 Valist_getAtom()

```
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 115 of file valist.c.

7.15.2.6 Valist_getAtomList()

```
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 95 of file valist.c.

7.15.2.7 Valist_getCenterX()

```
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 66 of file valist.c.

7.15.2.8 Valist_getCenterY()

```
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 76 of file valist.c.

7.15.2.9 Valist_getCenterZ()

```
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 85 of file valist.c.

7.15.2.10 Valist_getNumberAtoms()

```
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 105 of file valist.c.

7.15.2.11 Valist_getStatistics()

```
VEXTERNC Vrc_Codes Valist_getStatistics (
            Valist * thee )
```

Load up Valist with various statistics.

Author

Nathan Baker, Yong Huang

Returns

Success enumeration

Definition at line 869 of file valist.c.

7.15.2.12 Valist_memChk()

```
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 129 of file valist.c.

7.15.2.13 Valist_readPDB()

```
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 515 of file valist.c.

7.15.2.14 Valist_readPQR()

```
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 606 of file valist.c.

7.15.2.15 Valist_readXML()

```
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 725 of file valist.c.

7.16 Vatom class

Atom class for interfacing APBS with PDB files.

Files

- file vatom.c

  Class Vatom methods.

- file vatom.h

  Contains declarations for class Vatom.

Data Structures

- struct sVatom

  Contains public data members for Vatom class/module.

Macros

- #define VMAX_RECLEN 64

  Residue name length.

Typedefs

- typedef struct sVatom Vatom

  Declaration of the Vatom class as the Vatom structure.

Functions

- VEXTERNC double * Vatom_getPosition (Vatom *thee)

  Get atomic position.
- VEXTERNC void Vatom_setRadius (Vatom *thee, double radius)

  Set atomic radius.
- VEXTERNC double Vatom_getRadius (Vatom *thee)

  Get atomic position.
- VEXTERNC void Vatom_setPartID (Vatom *thee, int partID)

  Set partition ID.
- VEXTERNC double Vatom_getPartID (Vatom *thee)

  Get partition ID.
- VEXTERNC void Vatom_setAtomID (Vatom *thee, int id)

  Set atom ID.
- VEXTERNC double Vatom_getAtomID (Vatom *thee)

  Get atom ID.
- VEXTERNC void Vatom_setCharge (Vatom *thee, double charge)

  Set atomic charge.
- VEXTERNC double Vatom_getCharge (Vatom *thee)

  Get atomic charge.
- VEXTERNC void Vatom_setEpsilon (Vatom *thee, double epsilon)

  Set atomic epsilon.
- VEXTERNC double Vatom_getEpsilon (Vatom *thee)

  Get atomic epsilon.
- VEXTERNC unsigned long int Vatom_memChk (Vatom *thee)

  Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC void Vatom_setResName (Vatom *thee, char resName[VMAX_RECLEN])

  Set residue name.
- VEXTERNC void Vatom_setAtomName (Vatom *thee, char atomName[VMAX_RECLEN])

  Set atom name.
- VEXTERNC void Vatom_getResName (Vatom *thee, char resName[VMAX_RECLEN])

Retrieve residue name.

- VEXTERNC void Vatom_getAtomName (Vatom *thee, char atomName[VMAX_RECLEN])

  Retrieve atom name.
- VEXTERNC Vatom * Vatom_ctor ()

  Constructor for the Vatom class.
- VEXTERNC int Vatom_ctor2 (Vatom *thee)

  FORTRAN stub constructor for the Vatom class.
- VEXTERNC void Vatom_dtor (Vatom **thee)

  Object destructor.
- VEXTERNC void Vatom_dtor2 (Vatom *thee)

  FORTRAN stub object destructor.
- VEXTERNC void Vatom_setPosition (Vatom *thee, double position[3])

  Set the atomic position.
- VEXTERNC void Vatom_copyTo (Vatom *thee, Vatom *dest)

  Copy information to another atom.
- VEXTERNC void Vatom_copyFrom (Vatom *thee, Vatom *src)

  Copy information to another atom.

7.16.1 Detailed Description

Atom class for interfacing APBS with PDB files.

7.16.2 Macro Definition Documentation

7.16.2.1 VMAX_RECLEN

```
#define VMAX_RECLEN 64
```

Residue name length.

Author

Nathan Baker, David Gohara, Mike Schneiders

Definition at line 77 of file vatom.h.

7.16.3 Function Documentation

7.16.3.1 Vatom_copyFrom()

```
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 186 of file vatom.c.

7.16.3.2 Vatom_copyTo()

```
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 177 of file vatom.c.

7.16.3.3 Vatom_ctor()

```
VEXTERNC Vatom* Vatom_ctor ( )
```

Constructor for the Vatom class.

Author

Nathan Baker

Returns

Pointer to newly allocated Vatom object

Definition at line 142 of file vatom.c.

7.16.3.4 Vatom_ctor2()

```
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 153 of file vatom.c.

7.16.3.5 Vatom_dtor()

```
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 158 of file vatom.c.

7.16.3.6 Vatom_dtor2()

```
VEXTERNC void Vatom_dtor2 (
            Vatom * thee )
```

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to object to be destroyed

Definition at line 166 of file vatom.c.

7.16.3.7 Vatom_getAtomID()

```
VEXTERNC double Vatom_getAtomID (
            Vatom * thee )
```

Get atom ID.

Author

Nathan Baker

Parameters

thee	Vatom object

Returns

Unique non-negative number

Definition at line 84 of file vatom.c.

7.16.3.8 Vatom_getAtomName()

```
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 214 of file vatom.c.

7.16.3.9 Vatom_getCharge()

```
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 119 of file vatom.c.

7.16.3.10 Vatom_getEpsilon()

```
VEXTERNC double Vatom_getEpsilon (
            Vatom * thee )
```

Get atomic epsilon.

Author

David Gohara

Parameters

thee	Vatom object

Returns

Atomic epsilon (in Å)

Definition at line 132 of file vatom.c.

7.16.3.11 Vatom_getPartID()

```
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 70 of file vatom.c.

7.16.3.12 Vatom_getPosition()

```
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 63 of file vatom.c.

7.16.3.13 Vatom_getRadius()

```
VEXTERNC double Vatom_getRadius (
```

```
            Vatom * thee )
```

Get atomic position.

Author

Nathan Baker

Parameters

thee	Vatom object

Returns

Atomic radius (in Å)

Definition at line 105 of file vatom.c.

7.16.3.14 Vatom_getResName()

```
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 199 of file vatom.c.

7.16.3.15 Vatom_memChk()

```
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 138 of file vatom.c.

7.16.3.16 Vatom_setAtomID()

```
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 91 of file vatom.c.

7.16.3.17 Vatom_setAtomName()

```
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 207 of file vatom.c.

7.16.3.18 Vatom_setCharge()

```
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 112 of file vatom.c.

7.16.3.19 Vatom_setEpsilon()

```
VEXTERNC void Vatom_setEpsilon (
            Vatom * thee,
            double epsilon )
```

Set atomic epsilon.

Author

David Gohara

Parameters

thee	Vatom object
epsilon	Atomic epsilon (in Å)

Definition at line 126 of file vatom.c.

7.16.3.20 Vatom_setPartID()

```
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 77 of file vatom.c.

7.16.3.21 Vatom_setPosition()

```
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 168 of file vatom.c.

7.16.3.22 Vatom_setRadius()

```
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 98 of file vatom.c.

7.16.3.23 Vatom_setResName()

```
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 192 of file vatom.c.

7.17 Vcap class

Collection of routines which cap certain exponential and hyperbolic functions.

Files

- file vcap.c

  Class Vcap methods.
- file vcap.h

  Contains declarations for class Vcap.

Macros

- #define EXPMAX 85.00

  Maximum argument for exp(), sinh(), or cosh()
- #define EXPMIN -85.00

  Minimum argument for exp(), sinh(), or cosh()

Functions

- VEXTERNC double Vcap_exp (double x, int *ichop)

  Provide a capped exp() function.
- VEXTERNC double Vcap_sinh (double x, int *ichop)

  Provide a capped sinh() function.
- VEXTERNC double Vcap_cosh (double x, int *ichop)

  Provide a capped cosh() function.

7.17.1 Detailed Description

Collection of routines which cap certain exponential and hyperbolic functions.

Note

These routines are based on FORTRAN code by Mike Holst

7.17.2 Function Documentation

7.17.2.1 Vcap_cosh()

```
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 91 of file vcap.c.

7.17.2.2 Vcap_exp()

```
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 59 of file vcap.c.

7.17.2.3 Vcap_sinh()

```
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 75 of file vcap.c.

7.18 Vclist class

Atom cell list.

Files

- file vclist.c

  Class Vclist methods.
- file vclist.h

  Contains declarations for class Vclist.

Data Structures

- struct sVclistCell

  Atom cell list cell.
- struct sVclist

  Atom cell list.

Typedefs

- typedef struct sVclistCell VclistCell

  Declaration of the VclistCell class as the VclistCell structure.
- typedef struct sVclist Vclist

  Declaration of the Vclist class as the Vclist structure.

- typedef enum eVclist_DomainMode Vclist_DomainMode

  Declaration of Vclist_DomainMode enumeration type.

Enumerations

- enum eVclist_DomainMode { CLIST_AUTO_DOMAIN , CLIST_MANUAL_DOMAIN }

  Atom cell list domain setup mode.

Functions

- VEXTERNC unsigned long int Vclist_memChk (Vclist *thee)

  Get number of bytes in this object and its members.
- VEXTERNC double Vclist_maxRadius (Vclist *thee)

  Get the max probe radius value (in 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.

7.18.1 Detailed Description

Atom cell list.

7.18.2 Enumeration Type Documentation

7.18.2.1 eVclist_DomainMode

`enum eVclist_DomainMode`

Atom cell list domain setup mode.

Author

Nathan Baker

Enumerator

CLIST_AUTO_DOMAIN	Setup the cell list domain automatically to encompass the entire molecule
CLIST_MANUAL_DOMAIN	Specify the cell list domain manually through the constructor

Definition at line 82 of file vclist.h.

7.18.3 Function Documentation

7.18.3.1 Vclist_ctor()

```
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 75 of file vclist.c.

7.18.3.2 Vclist_ctor2()

```
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 343 of file vclist.c.

7.18.3.3 Vclist_dtor()

```
VEXTERNC void Vclist_dtor (
             Vclist ** thee )
```

Destroy object.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of object

Definition at line 397 of file vclist.c.

7.18.3.4 Vclist_dtor2()

```
VEXTERNC void Vclist_dtor2 (
             Vclist * thee )
```

FORTRAN stub to destroy object.

Author

Nathan Baker

Parameters

thee	Pointer to object

Definition at line 408 of file vclist.c.

7.18.3.5 Vclist_getCell()

```
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 423 of file vclist.c.

7.18.3.6 Vclist_maxRadius()

```
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 68 of file vclist.c.

7.18.3.7 Vclist_memChk()

```
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 63 of file vclist.c.

7.18.3.8 VclistCell_ctor()

```
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 452 of file vclist.c.

7.18.3.9 VclistCell_ctor2()

```
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 464 of file vclist.c.

7.18.3.10 VclistCell_dtor()

```
VEXTERNC void VclistCell_dtor (
            VclistCell ** thee )
```

Destroy object.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of object

Definition at line 486 of file vclist.c.

7.18.3.11 VclistCell_dtor2()

```
VEXTERNC void VclistCell_dtor2 (
            VclistCell * thee )
```

FORTRAN stub to destroy object.

Author

Nathan Baker

Parameters

thee	Pointer to object

Definition at line 497 of file vclist.c.

7.19 Vgreen class

Provides capabilities for pointwise evaluation of free space Green's function for point charges in a uniform dielectric.

Files

- file vgreen.c

  Class Vgreen methods.
- file vgreen.h

  Contains declarations for class Vgreen.

Data Structures

- struct sVgreen

  Contains public data members for Vgreen class/module.

Typedefs

- typedef struct sVgreen Vgreen

  Declaration of the Vgreen class as the Vgreen structure.

Functions

- VEXTERNC Valist * Vgreen_getValist (Vgreen *thee)

  Get the atom list associated with this Green's function object.
- VEXTERNC unsigned long int Vgreen_memChk (Vgreen *thee)

  Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC Vgreen * Vgreen_ctor (Valist *alist)

  Construct the Green's function oracle.
- VEXTERNC int Vgreen_ctor2 (Vgreen *thee, Valist *alist)

  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)

7.19.1 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

7.19.2 Function Documentation

7.19.2.1 Vgreen_coulomb()

```
VEXTERNC int Vgreen_coulomb (
            Vgreen * thee,
            int npos,
            double * x,
            double * y,
            double * z,
            double * val )
```

Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation or H. E. Johnston, R. Krasny FMM library (if available)

Returns the potential ϕ defined by

$$\phi(r) = \sum_i \frac{q_i}{r_i}$$

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

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 258 of file vgreen.c.

7.19.2.2 Vgreen_coulomb_direct()

```
VEXTERNC int Vgreen_coulomb_direct (
             Vgreen * thee,
             int npos,
             double * x,
             double * y,
             double * z,
             double * val )
```

Get the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.
Returns the potential ϕ defined by

$$\phi(r) = \sum_i \frac{q_i}{r_i}$$

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

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 224 of file vgreen.c.

7.19.2.3 Vgreen_coulombD()

```
VEXTERNC int Vgreen_coulombD (
            Vgreen * thee,
            int npos,
            double * x,
            double * y,
            double * z,
            double * pot,
            double * gradx,
            double * grady,
            double * gradz )
```

Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using either direct summation or H. E. Johnston/R. Krasny FMM library (if available)
Returns the field $\nabla\phi$ defined by

$$\nabla\phi(r) = \sum_i \frac{q_i}{r_i}$$

where q_i is the atomic charge (in e) and r_i is the distance to the observation point r. The field is scaled to units of V/Å.

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 362 of file vgreen.c.

7.19.2.4 Vgreen_coulombD_direct()

```
VEXTERNC int Vgreen_coulombD_direct (
            Vgreen * thee,
            int npos,
```

```
            double * x,
            double * y,
            double * z,
            double * pot,
            double * gradx,
            double * grady,
            double * gradz )
```

Get gradient of the Coulomb's Law Green's function (solution to Laplace's equation) integrated over the atomic point charges using direct summation.
Returns the field $\nabla\phi$ defined by

$$\nabla\phi(r) = \sum_i \frac{q_i}{r_i}$$

where q_i is the atomic charge (in e) and r_i is the distance to the observation point r. The field is scaled to units of V/Å.

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 310 of file vgreen.c.

7.19.2.5 Vgreen_ctor()

```
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 156 of file vgreen.c.

7.19.2.6 Vgreen_ctor2()

```
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 167 of file vgreen.c.

7.19.2.7 Vgreen_dtor()

```
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 192 of file vgreen.c.

7.19.2.8 Vgreen_dtor2()

```
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 200 of file vgreen.c.

7.19.2.9 Vgreen_getValist()

```
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 142 of file vgreen.c.

7.19.2.10 Vgreen_helmholtz()

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

Get the Green's function for Helmholtz's equation integrated over the atomic point charges.
Returns the potential ϕ defined by

$$\phi(r) = \sum_i \frac{q_i e^{-\kappa r_i}}{r_i}$$

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

Author

Nathan Baker

Bug Not implemented yet

Note

Not implemented yet

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 209 of file vgreen.c.

7.19.2.11 Vgreen_helmholtzD()

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

Get the gradient of Green's function for Helmholtz's equation integrated over the atomic point charges.
Returns the field $\nabla\phi$ defined by

$$\nabla\phi(r) = \nabla \sum_i \frac{q_i e^{-\kappa r_i}}{r_i}$$

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

Author

Nathan Baker

Bug Not implemented yet

Note

Not implemented yet

Parameters

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

Parameters

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 216 of file vgreen.c.

7.19.2.12 Vgreen_memChk()

```
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 149 of file vgreen.c.

7.20 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 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 , VDF_DXBIN =6 }

  Format of data for APBS I/O.

7.20.1 Detailed Description

A "class" which consists solely of macro definitions which are used by several other classes.

7.20.2 Macro Definition Documentation

7.20.2.1 VAPBS_BACK

```
#define VAPBS_BACK 4
```

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line 432 of file vhal.h.

7.20.2.2 VAPBS_DOWN

```
#define VAPBS_DOWN 5
```

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line 438 of file vhal.h.

7.20.2.3 VAPBS_FRONT

```
#define VAPBS_FRONT 1
```

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line 414 of file vhal.h.

7.20.2.4 VAPBS_LEFT

```
#define VAPBS_LEFT 3
```

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line 426 of file vhal.h.

7.20.2.5 VAPBS_RIGHT

```
#define VAPBS_RIGHT 0
```

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line 408 of file vhal.h.

7.20.2.6 VAPBS_UP

```
#define VAPBS_UP 2
```

Face definition for a volume.

Note

Consistent with PMG if RIGHT = EAST, BACK = SOUTH

Definition at line 420 of file vhal.h.

7.20.2.7 VEMBED

```
#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 556 of file vhal.h.

7.20.2.8 VFLOOR

```
#define VFLOOR(
              value ) floor(value)
```

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

Author

Todd Dolinksy

Definition at line 547 of file vhal.h.

7.20.3 Enumeration Type Documentation

7.20.3.1 eVbcfl

```
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 207 of file vhal.h.

7.20.3.2 eVchrg_Meth

```
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 230 of file vhal.h.

7.20.3.3 eVchrg_Src

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 251 of file vhal.h.

7.20.3.4 eVdata_Format

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
VDF_DXBIN	OpendDX (Data Explorer) binary format

Definition at line 309 of file vhal.h.

7.20.3.5 eVdata_Type

enum eVdata_Type
Types of (scalar) data that can be written out of APBS.

Author

Nathan Baker

Enumerator

VDT_CHARGE	Charge distribution (e)
VDT_POT	Potential (kT/e)
VDT_ATOMPOT	Atom potential (kT/e)
VDT_SMOL	Solvent accessibility defined by molecular/Connolly surface definition (1 = accessible, 0 = inaccessible)
VDT_SSPL	Spline-based solvent accessibility (1 = accessible, 0 = inaccessible)
VDT_VDW	van der Waals-based accessibility (1 = accessible, 0 = inaccessible)
VDT_IVDW	Ion accessibility/inflated van der Waals (1 = accessible, 0 = inaccessible)
VDT_LAP	Laplacian of potential (kT/e/A^2)
VDT_EDENS	Energy density $\epsilon(\nabla u)^2$, where u is potential (kT/e/A)^2
VDT_NDENS	Ion number density $\sum c_i \exp(-q_i u)^2$, where u is potential (output in M)
VDT_QDENS	Ion charge density $\sum q_i c_i \exp(-q_i u)^2$, where u is potential (output in $e_c M$)
VDT_DIELX	Dielectric x-shifted map as calculated with the currently specified scheme (dimensionless)
VDT_DIELY	Dielectric y-shifted map as calculated with the currently specified scheme (dimensionless)
VDT_DIELZ	Dielectric 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 269 of file vhal.h.

7.20.3.6 eVhal_IPKEYType

`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 157 of file vhal.h.

7.20.3.7 eVhal_PBEType

`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 139 of file vhal.h.

7.20.3.8 eVoutput_Format

enum eVoutput_Format
Output file format.

Enumerator

OUTPUT_NULL	No output
OUTPUT_FLAT	Output in flat-file format

Definition at line 191 of file vhal.h.

7.20.3.9 eVrc_Codes

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 66 of file vhal.h.

7.20.3.10 eVsol_Meth

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 81 of file vhal.h.

7.20.3.11 eVsurf_Meth

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 102 of file vhal.h.

7.21 Matrix wrapper class

A header for including data wrapping matrices.

Files

- file vmatrix.h

  Contains inclusions for matrix data wrappers.

7.21.1 Detailed Description

A header for including data wrapping matrices.

7.22 Vparam class

Reads and assigns charge/radii parameters.

Files

- file vparam.c

  Class Vparam methods.
- file vparam.h

  Contains declarations for class Vparam.

Data Structures

- struct sVparam_AtomData

  AtomData sub-class; stores atom data.

- struct Vparam_ResData

  ResData sub-class; stores residue data.
- struct Vparam

  Reads and assigns charge/radii parameters.

Typedefs

- typedef struct sVparam_AtomData Vparam_AtomData

  Declaration of the Vparam_AtomData class as the sVparam_AtomData structure.
- typedef struct Vparam_ResData Vparam_ResData

  Declaration of the Vparam_ResData class as the Vparam_ResData structure.
- typedef struct Vparam Vparam

  Declaration of the Vparam class as the Vparam structure.

Functions

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

  Read a single line of the flat file database.
- VPRIVATE int readXMLFileAtom (Vio ∗sock, Vparam_AtomData ∗atom)

  Read atom information from an XML file.
- VEXTERNC unsigned long int Vparam_memChk (Vparam ∗thee)

  Get number of bytes in this object and its members.
- VEXTERNC Vparam_AtomData ∗ Vparam_AtomData_ctor ()

  Construct the object.
- VEXTERNC int Vparam_AtomData_ctor2 (Vparam_AtomData ∗thee)

  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.

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.

7.22.1 Detailed Description

Reads and assigns charge/radii parameters.

7.22.2 Function Documentation

7.22.2.1 readFlatFileLine()

```
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 691 of file vparam.c.

7.22.2.2 readXMLFileAtom()

```
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 610 of file vparam.c.

7.22.2.3 Vparam_AtomData_copyFrom()

```
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 607 of file vparam.c.

7.22.2.4 Vparam_AtomData_copyTo()

```
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 571 of file vparam.c.

7.22.2.5 Vparam_AtomData_ctor()

```
VEXTERNC Vparam_AtomData* Vparam_AtomData_ctor ( )
```

Construct the object.

Author

Nathan Baker

Returns

Newly allocated object

Definition at line 109 of file vparam.c.

7.22.2.6 Vparam_AtomData_ctor2()

```
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 121 of file vparam.c.

7.22.2.7 Vparam_AtomData_dtor()

```
VEXTERNC void Vparam_AtomData_dtor (
            Vparam_AtomData ** thee )
```

Destroy object.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of object

Definition at line 123 of file vparam.c.

7.22.2.8 Vparam_AtomData_dtor2()

```
VEXTERNC void Vparam_AtomData_dtor2 (
            Vparam_AtomData * thee )
```

FORTRAN stub to destroy object.

Author

Nathan Baker

Parameters

thee	Pointer to object

Definition at line 133 of file vparam.c.

7.22.2.9 Vparam_ctor()

```
VEXTERNC Vparam* Vparam_ctor ( )
```

Construct the object.

Author

Nathan Baker

Returns

Newly allocated Vparam object

Definition at line 181 of file vparam.c.

7.22.2.10 Vparam_ctor2()

```
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 193 of file vparam.c.

7.22.2.11 Vparam_dtor()

```
VEXTERNC void Vparam_dtor (
            Vparam ** thee )
```

Destroy object.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of object

Definition at line 213 of file vparam.c.

7.22.2.12 Vparam_dtor2()

```
VEXTERNC void Vparam_dtor2 (
            Vparam * thee )
```

FORTRAN stub to destroy object.

Author

Nathan Baker

Parameters

thee	Pointer to object

Definition at line 223 of file vparam.c.

7.22.2.13 Vparam_getAtomData()

```
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 267 of file vparam.c.

7.22.2.14 Vparam_getResData()

```
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 241 of file vparam.c.

7.22.2.15 Vparam_memChk()

```
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 102 of file vparam.c.

7.22.2.16 Vparam_readFlatFile()

```
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 445 of file vparam.c.

7.22.2.17 Vparam_readXMLFile()

```
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 306 of file vparam.c.

7.22.2.18 Vparam_ResData_copyTo()

```
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 585 of file vparam.c.

7.22.2.19 Vparam_ResData_ctor()

```
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 135 of file vparam.c.

7.22.2.20 Vparam_ResData_ctor2()

```
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 147 of file vparam.c.

7.22.2.21 Vparam_ResData_dtor()

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

Destroy object.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of object

Definition at line 160 of file vparam.c.

7.22.2.22 Vparam_ResData_dtor2()

```
VEXTERNC void Vparam_ResData_dtor2 (
            Vparam_ResData * thee )
```

FORTRAN stub to destroy object.

Author

Nathan Baker

Parameters

thee	Pointer to object

Definition at line 170 of file vparam.c.

7.23 Vpbe class

The Poisson-Boltzmann master class.

Files

- file vpbe.c

  Class Vpbe methods.
- file vpbe.h

  Contains declarations for class Vpbe.

Data Structures

- struct sVpbe

  Contains public data members for Vpbe class/module.

Typedefs

- typedef struct sVpbe Vpbe

  Declaration of the Vpbe class as the Vpbe structure.

Functions

- VEXTERNC Valist * Vpbe_getValist (Vpbe *thee)

  Get atom list.

- VEXTERNC Vacc * Vpbe_getVacc (Vpbe *thee)

  Get accessibility oracle.
- VEXTERNC double Vpbe_getBulkIonicStrength (Vpbe *thee)

  Get bulk ionic strength.
- VEXTERNC double Vpbe_getMaxIonRadius (Vpbe *thee)

  Get maximum radius of ion species.
- VEXTERNC double Vpbe_getTemperature (Vpbe *thee)

  Get temperature.
- VEXTERNC double Vpbe_getSoluteDiel (Vpbe *thee)

  Get solute dielectric constant.
- VEXTERNC double Vpbe_getGamma (Vpbe *thee)

  Get apolar coefficient.
- VEXTERNC double Vpbe_getSoluteRadius (Vpbe *thee)

  Get sphere radius which bounds biomolecule.
- VEXTERNC double Vpbe_getSoluteXlen (Vpbe *thee)

  Get length of solute in x dimension.
- VEXTERNC double Vpbe_getSoluteYlen (Vpbe *thee)

  Get length of solute in y dimension.
- VEXTERNC double Vpbe_getSoluteZlen (Vpbe *thee)

  Get length of solute in z dimension.
- VEXTERNC double * Vpbe_getSoluteCenter (Vpbe *thee)

  Get coordinates of solute center.
- VEXTERNC double Vpbe_getSoluteCharge (Vpbe *thee)

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

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

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

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

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

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

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

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

  Get length of the membrane (A)
  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.

7.23.1 Detailed Description

The Poisson-Boltzmann master class.

```
Contains objects and parameters used in every PBE calculation,
regardless of method.
```

7.23.2 Function Documentation

7.23.2.1 Vpbe_ctor()

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

Construct Vpbe object.

Author

Nathan Baker and Mike Holst and Michael Grabe

Note

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

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

where the units are esu∗esu/erg/mol. To obtain $^{-2}$, 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:

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

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

Returns

Pointer to newly allocated Vpbe object

Parameters

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 246 of file vpbe.c.

7.23.2.2 Vpbe_ctor2()

```
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:

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

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 264 of file vpbe.c.

7.23.2.3 Vpbe_dtor()

```
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 467 of file vpbe.c.

7.23.2.4 Vpbe_dtor2()

```
VEXTERNC void Vpbe_dtor2 (
            Vpbe * thee )
```

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to object to be destroyed

Definition at line 475 of file vpbe.c.

7.23.2.5 Vpbe_getBulkIonicStrength()

```
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 84 of file vpbe.c.

7.23.2.6 Vpbe_getCoulombEnergy1()

```
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 interio
    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 481 of file vpbe.c.

7.23.2.7 Vpbe_getDeblen()

```
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 141 of file vpbe.c.

7.23.2.8 Vpbe_getGamma()

```
VEXTERNC double Vpbe_getGamma (
            Vpbe * thee )
```

Get apolar coefficient.

Author

Nathan Baker

Parameters

thee	Vpbe object

Returns

Apolar coefficent (kJ/mol/A^2)

7.23.2.9 Vpbe_getIons()

```
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 535 of file vpbe.c.

7.23.2.10 Vpbe_getLmem()

```
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 209 of file vpbe.c.

7.23.2.11 Vpbe_getMaxIonRadius()

```
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 127 of file vpbe.c.

7.23.2.12 Vpbe_getmembraneDiel()

```
VEXTERNC double Vpbe_getmembraneDiel (
            Vpbe * thee )
```

Get membrane dielectric constant.

Author

Michael Grabe

Parameters

thee	Vpbe object

Returns

Membrane dielectric constant

Definition at line 221 of file vpbe.c.

7.23.2.13 Vpbe_getmemv()

```
VEXTERNC double Vpbe_getmemv (
```

```
            Vpbe * thee )
```

Get membrane potential (kT)

Author

Michael Grabe

Parameters

thee	Vpbe object

Definition at line 233 of file vpbe.c.

7.23.2.14 Vpbe_getSoluteCenter()

```
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 107 of file vpbe.c.

7.23.2.15 Vpbe_getSoluteCharge()

```
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 186 of file vpbe.c.

7.23.2.16 Vpbe_getSoluteDiel()

```
VEXTERNC double Vpbe_getSoluteDiel (
            Vpbe * thee )
```

Get solute dielectric constant.

Author

Nathan Baker

Parameters

thee	Vpbe object

Returns

Solute dielectric constant

Definition at line 99 of file vpbe.c.

7.23.2.17 Vpbe_getSoluteRadius()

```
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 162 of file vpbe.c.

7.23.2.18 Vpbe_getSoluteXlen()

```
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 168 of file vpbe.c.

7.23.2.19 Vpbe_getSoluteYlen()

```
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 174 of file vpbe.c.

7.23.2.20 Vpbe_getSoluteZlen()

```
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 180 of file vpbe.c.

7.23.2.21 Vpbe_getSolventDiel()

```
VEXTERNC double Vpbe_getSolventDiel (
            Vpbe * thee )
```

Get solvent dielectric constant.

Author

Nathan Baker

Parameters

thee	Vpbe object

Returns

Solvent dielectric constant

Definition at line 113 of file vpbe.c.

7.23.2.22 Vpbe_getSolventRadius()

```
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 120 of file vpbe.c.

7.23.2.23 Vpbe_getTemperature()

```
VEXTERNC double Vpbe_getTemperature (
            Vpbe * thee )
```

Get temperature.

Author

Nathan Baker

Parameters

thee	Vpbe object

Returns

Temperature (K)

Definition at line 91 of file vpbe.c.

7.23.2.24 Vpbe_getVacc()

```
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 76 of file vpbe.c.

7.23.2.25 Vpbe_getValist()

```
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 69 of file vpbe.c.

7.23.2.26 Vpbe_getXkappa()

```
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 134 of file vpbe.c.

7.23.2.27 Vpbe_getZkappa2()

```
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 148 of file vpbe.c.

7.23.2.28 Vpbe_getZmagic()

```
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 155 of file vpbe.c.

7.23.2.29 Vpbe_getzmem()

```
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 197 of file vpbe.c.

7.23.2.30 Vpbe_memChk()

```
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 523 of file vpbe.c.

7.24 Vstring class

Provides a collection of useful non-ANSI string functions.

Files

- file vstring.c

  Class Vstring methods.
- file vstring.h

  Contains declarations for class Vstring.

Functions

- char * Vstring_wrappedtext (const char *str, int right_margin, int left_padding)
- VEXTERNC int Vstring_strcasecmp (const char *s1, const char *s2)

  Case-insensitive string comparison (BSD standard)
- VEXTERNC int Vstring_isdigit (const char *tok)

  A modified sscanf that examines the complete string.

7.24.1 Detailed Description

Provides a collection of useful non-ANSI string functions.

7.24.2 Function Documentation

7.24.2.1 Vstring_isdigit()

```
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.

7.24.2.2 Vstring_strcasecmp()

```
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.

7.24.2.3 Vstring_wrappedtext()

```
VEXTERNC char * Vstring_wrappedtext (
            const char * str,
            int right_margin,
            int left_padding )
```

Creates a wrapped and indented string from an input string

Author

Tucker Beck

Note

This funciton allocates a new string, so be sure to free it!

Creates a wrapped and indented string from an input string

Author

Tucker Beck

Note

This function allocates a new string, so be sure to free it!

Note

: The +2 is for the newline character and the null-terminating character;

Parameters

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.

7.25 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*

7.25.1 Detailed Description

Collection of constants and conversion factors.

7.26 Vgrid class

Oracle for Cartesian mesh data.

Files

- file vgrid.c

  Class Vgrid methods.
- file vgrid.h

  Potential oracle for Cartesian mesh data.

Data Structures

- struct sVgrid

  Electrostatic potential oracle for Cartesian mesh data.

Macros

- #define VGRID_DIGITS 6

  Number of decimal places for comparisons and formatting.

Typedefs

- typedef struct sVgrid Vgrid

  Declaration of the Vgrid class as the sVgrid structure.

Functions

- VEXTERNC unsigned long int Vgrid_memChk (Vgrid *thee)

  Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC Vgrid * Vgrid_ctor (int nx, int ny, int nz, double hx, double hy, double hzed, double xmin, double ymin, double zmin, double *data)

  Construct Vgrid object with values obtained from Vpmg_readDX (for example)
- VEXTERNC int Vgrid_ctor2 (Vgrid *thee, int nx, int ny, int nz, double hx, double hy, double hzed, double xmin, double ymin, double zmin, double *data)

  Initialize Vgrid object with values obtained from Vpmg_readDX (for example)
- VEXTERNC int Vgrid_value (Vgrid *thee, double x[3], double *value)

  Get potential value (from mesh or approximation) at a point.
- VEXTERNC void Vgrid_dtor (Vgrid **thee)

  Object destructor.
- VEXTERNC void Vgrid_dtor2 (Vgrid *thee)

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

  Write out the binary data in OpenDX grid format.
- VEXTERNC int Vgrid_readDXBIN (Vgrid *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname)

  Read in binary 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:
- VEXTERNC double Vgrid_normL2 (Vgrid *thee)

  Get the L_2 norm of the data. This returns the integral:
- VEXTERNC double Vgrid_normLinf (Vgrid *thee)

Get the L_∞ norm of the data. This returns the integral:

- VEXTERNC double Vgrid_seminormH1 (Vgrid *thee)

  Get the H_1 semi-norm of the data. This returns the integral:

- VEXTERNC double Vgrid_normH1 (Vgrid *thee)

  Get the H_1 norm (or energy norm) of the data. This returns the integral:

7.26.1 Detailed Description

Oracle for Cartesian mesh data.

7.26.2 Function Documentation

7.26.2.1 Vgrid_ctor()

```
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 86 of file vgrid.c.

7.26.2.2 Vgrid_ctor2()

```
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 112 of file vgrid.c.

7.26.2.3 Vgrid_curvature()

```
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 299 of file vgrid.c.

7.26.2.4 Vgrid_dtor()

```
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 152 of file vgrid.c.

7.26.2.5 Vgrid_dtor2()

```
VEXTERNC void Vgrid_dtor2 (
            Vgrid * thee )
```

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to object to be destroyed

Definition at line 165 of file vgrid.c.

7.26.2.6 Vgrid_gradient()

```
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 379 of file vgrid.c.

7.26.2.7 Vgrid_integrate()

```
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 1790 of file vgrid.c.

7.26.2.8 Vgrid_memChk()

```
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.

7.26.2.9 Vgrid_normH1()

```
VEXTERNC double Vgrid_normH1 (
            Vgrid * thee )
```

Get the H_1 norm (or energy norm) of the data. This returns the integral:

Author

Nathan Baker

Parameters

thee	Vgrid object

Returns

Integral of data

$$\|u\|_{H_1} = \left(\int_\Omega |\nabla u(x)|^2 dx + \int_\Omega |u(x)|^2 dx \right)^{1/2}$$

Definition at line 1931 of file vgrid.c.

7.26.2.10 Vgrid_normL1()

```
VEXTERNC double Vgrid_normL1 (
            Vgrid * thee )
```

Get the L_1 norm of the data. This returns the integral:

Author

Nathan Baker

Parameters

thee	Vgrid object

Returns

L_1 norm of data

$$\|u\|_{L_1} = \int_\Omega |u(x)| dx$$

Definition at line 1828 of file vgrid.c.

7.26.2.11 Vgrid_normL2()

```
VEXTERNC double Vgrid_normL2 (
            Vgrid * thee )
```

Get the L_2 norm of the data. This returns the integral:

Author

Nathan Baker

Parameters

thee	Vgrid object

Returns

L_2 norm of data

$$\|u\|_{L_2} = \left(\int_\Omega |u(x)|^2 dx \right)^{1/2}$$

Definition at line 1858 of file vgrid.c.

7.26.2.12 Vgrid_normLinf()

```
VEXTERNC double Vgrid_normLinf (
            Vgrid * thee )
```

Get the L_∞ norm of the data. This returns the integral:

Author

Nathan Baker

Parameters

thee	Vgrid object

Returns

$L\infty$ norm of data

$$\|u\|_{L_\infty} = \sup_{x \in \Omega} |u(x)|$$

Definition at line 1946 of file vgrid.c.

7.26.2.13 Vgrid_readDX()

```
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 586 of file vgrid.c.

7.26.2.14 Vgrid_readDXBIN()

```
VEXTERNC int Vgrid_readDXBIN (
             Vgrid * thee,
             const char * iodev,
             const char * iofmt,
             const char * thost,
             const char * fname )
```

Read in binary data in OpenDX grid format.

Note

All dimension information is given in order: z, y, x

Author

Juan Brandi

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 dx binary file.

Author

Juan Brandi

Definition at line 810 of file vgrid.c.

7.26.2.15 Vgrid_readGZ()

```
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 462 of file vgrid.c.

7.26.2.16 Vgrid_seminormH1()

```
VEXTERNC double Vgrid_seminormH1 (
            Vgrid * thee )
```

Get the H_1 semi-norm of the data. This returns the integral:

Author

Nathan Baker

Parameters

thee	Vgrid object

Returns

Integral of data

$$|u|_{H_1} = \left(\int_\Omega |\nabla u(x)|^2 dx \right)^{1/2}$$

Definition at line 1888 of file vgrid.c.

7.26.2.17 Vgrid_value()

```
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 179 of file vgrid.c.

7.26.2.18 Vgrid_writeDX()

```
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 1206 of file vgrid.c.

7.26.2.19 Vgrid_writeDXBIN()

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

Write out the binary 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 1458 of file vgrid.c.

7.26.2.20 Vgrid_writeUHBD()

```
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 1692 of file vgrid.c.

7.27 Vmgrid class

Oracle for Cartesian mesh data.

Files

- file vmgrid.c

  Class Vmgrid methods.
- file vmgrid.h

  Multiresolution oracle for Cartesian mesh data.

Data Structures

- struct sVmgrid

  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.

7.27.1 Detailed Description

Oracle for Cartesian mesh data.

7.27.2 Function Documentation

7.27.2.1 Vmgrid_addGrid()

```
VEXTERNC int Vmgrid_addGrid (
            Vmgrid * thee,
            Vgrid * grid )
```

Add a grid to the hierarchy.

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 otherwise

Definition at line 195 of file vmgrid.c.

7.27.2.2 Vmgrid_ctor()

```
VEXTERNC Vmgrid* Vmgrid_ctor ( )
```

Construct Vmgrid object.

Author

Nathan Baker

Returns

Newly allocated and initialized Vmgrid object

Definition at line 57 of file vmgrid.c.

7.27.2.3 Vmgrid_ctor2()

```
VEXTERNC int Vmgrid_ctor2 (
            Vmgrid * thee )
```

Initialize Vmgrid object.

Author

Nathan Baker

Parameters

thee	Newly allocated Vmgrid object

Returns

Newly allocated and initialized Vmgrid object

Definition at line 72 of file vmgrid.c.

7.27.2.4 Vmgrid_curvature()

```
VEXTERNC int Vmgrid_curvature (
            Vmgrid * thee,
            double pt[3],
            int cflag,
            double * curv )
```

Get second derivative values at a point.

Author

Nathan Baker (wrapper for Vgrid routine by Steve Bond)

Parameters

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

Returns

1 if successful, 0 if off grid

Definition at line 138 of file vmgrid.c.

7.27.2.5 Vmgrid_dtor()

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

Object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to memory location of object to be destroyed

Definition at line 88 of file vmgrid.c.

7.27.2.6 Vmgrid_dtor2()

```
VEXTERNC void Vmgrid_dtor2 (
            Vmgrid * thee )
```

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to object to be destroyed

Definition at line 101 of file vmgrid.c.

7.27.2.7 Vmgrid_getGridByNum()

```
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

7.27.2.8 Vmgrid_getGridByPoint()

```
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

7.27.2.9 Vmgrid_gradient()

```
VEXTERNC int Vmgrid_gradient (
            Vmgrid * thee,
            double pt[3],
            double grad[3] )
```

Get first derivative values at a point.

Author

Nathan Baker and Steve Bond

Parameters

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

Returns

1 if successful, 0 if off grid

Definition at line 167 of file vmgrid.c.

7.27.2.10 Vmgrid_value()

```
VEXTERNC int Vmgrid_value (
```

```
Vmgrid * thee,
double x[3],
double * value )
```

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

Author

Nathan Baker

Parameters

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

Returns

1 if successful, 0 if off grid

Definition at line 107 of file vmgrid.c.

7.28 Vopot class

Potential oracle for Cartesian mesh data.

Files

- file vopot.c

  Class Vopot methods.
- file vopot.h

  Potential oracle for Cartesian mesh data.

Data Structures

- struct sVopot

  Electrostatic potential oracle for Cartesian mesh data.

Typedefs

- typedef struct sVopot Vopot

  Declaration of the Vopot class as the Vopot structure.

Functions

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

  Construct Vopot object with values obtained from Vpmg_readDX (for example)
- VEXTERNC int Vopot_ctor2 (Vopot *thee, Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl)

  Initialize Vopot object with values obtained from Vpmg_readDX (for example)
- VEXTERNC int Vopot_pot (Vopot *thee, double x[3], double *pot)

  Get potential value (from mesh or approximation) at a point.
- VEXTERNC void Vopot_dtor (Vopot **thee)

Object destructor.

- VEXTERNC void Vopot_dtor2 (Vopot *thee)

  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.

7.28.1 Detailed Description

Potential oracle for Cartesian mesh data.

7.28.2 Function Documentation

7.28.2.1 Vopot_ctor()

```
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 65 of file vopot.c.

7.28.2.2 Vopot_ctor2()

```
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 80 of file vopot.c.

7.28.2.3 Vopot_curvature()

```
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 214 of file vopot.c.

7.28.2.4 Vopot_dtor()

```
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 94 of file vopot.c.

7.28.2.5 Vopot_dtor2()

```
VEXTERNC void Vopot_dtor2 (
            Vopot * thee )
```

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to object to be destroyed

Definition at line 107 of file vopot.c.

7.28.2.6 Vopot_gradient()

```
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 300 of file vopot.c.

7.28.2.7 Vopot_pot()

```
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 114 of file vopot.c.

7.29 Vpmg class

A wrapper for Mike Holst's PMG multigrid code.

Files

- file vpmg.c

  Class Vpmg methods.
- file vpmg.h

  Contains declarations for class Vpmg.

Data Structures

- struct sVpmg

  Contains public data members for Vpmg class/module.

Typedefs

- typedef struct sVpmg Vpmg

  Declaration of the Vpmg class as the Vpmg structure.

Functions

- VEXTERNC unsigned long int Vpmg_memChk (Vpmg *thee)

  Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC Vpmg * Vpmg_ctor (Vpmgp *parms, Vpbe *pbe, int focusFlag, Vpmg *pmgOLD, MGparm *mgparm, PBEparm_calcEnergy energyFlag)

  Constructor for the Vpmg class (allocates new memory)
- VEXTERNC int Vpmg_ctor2 (Vpmg *thee, Vpmgp *parms, Vpbe *pbe, int focusFlag, Vpmg *pmgOLD, MGparm *mgparm, PBEparm_calcEnergy energyFlag)

  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 use↩ KappaMap, 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 pbe↩ type, PBEparm *pbeparm)

Fill the specified array with accessibility values.

- VPUBLIC void Vpmg_fieldSpline4 (Vpmg *thee, int atomID, double field[3])

  Computes the field at an atomic center using a stencil based on the first derivative of a 5th order B-spline.

- VEXTERNC double Vpmg_qfPermanentMultipoleEnergy (Vpmg *thee, int atomID)

  Computes the permanent multipole electrostatic hydration energy (the polarization component of the hydration energy currently computed in TINKER).

- VEXTERNC void Vpmg_qfPermanentMultipoleForce (Vpmg *thee, int atomID, double force[3], double torque[3])

  Computes the q-Phi Force for permanent multipoles based on 5th order B-splines.

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

  Compute the ionic boundary force for permanent multipoles.

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

  Compute the dielectric boundary force for permanent multipoles.

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

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

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

  q-Phi direct polarization force between permanent multipoles and non-local induced dipoles based on 5th Order B-Splines. Keep in mind that the "non-local" induced 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, size_t *nrwk, int *niwk, int *nx, int *ny, int *nz, int *nlev, int *nu1, int *nu2, int *mgkey, int *itmax, int *istop, int *ipcon, int *nonlin, int *mgsmoo, int *mgprol, int *mgcoar, int *mgsolv, int *mgdisc, int *iinfo, double *errtol, int *ipkey, double *omegal, double *omegan, int *irite, int *iperf)

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

7.29.1 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.

7.29.2 Function Documentation

7.29.2.1 bcolcomp()

```
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 10741 of file vpmg.c.

7.29.2.2 bcolcomp2()

```
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	

Parameters

colptr	
flag	Operation selection parameter 0 = Use Poisson operator only 1 = Use linearization of full operation around current solution.

Definition at line 10795 of file vpmg.c.

7.29.2.3 bcolcomp3()

```
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 10831 of file vpmg.c.

7.29.2.4 bcolcomp4()

```
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 10858 of file vpmg.c.

7.29.2.5 pcolcomp()

```
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 11023 of file vpmg.c.

7.29.2.6 Vpackmg()

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

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

Author

Nathan Baker

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

Definition at line 555 of file mgsubd.c.

7.29.2.7 Vpmg_ctor()

```
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 141 of file vpmg.c.

7.29.2.8 Vpmg_ctor2()

```
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 153 of file vpmg.c.

7.29.2.9 Vpmg_dbDirectPolForce()

```
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

7.29.2.10 Vpmg_dbForce()

```
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 6010 of file vpmg.c.

7.29.2.11 Vpmg_dbMutualPolForce()

```
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

Parameters

atomID	Atom index
force	(returned) force

7.29.2.12 Vpmg_dbNLDirectPolForce()

```
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

7.29.2.13 Vpmg_dbPermanentMultipoleForce()

```
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

7.29.2.14 Vpmg_dielEnergy()

```
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:
   \f[ G = \frac{1}{2} \int \epsilon (\nabla u)^2 dx \f]
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 1279 of file vpmg.c.

7.29.2.15 Vpmg_dielGradNorm()

```
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):
   \f[ \int \| \nabla \epsilon \| dx \f]
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 1342 of file vpmg.c.

7.29.2.16 Vpmg_dtor()

```
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 561 of file vpmg.c.

7.29.2.17 Vpmg_dtor2()

```
VEXTERNC void Vpmg_dtor2 (
             Vpmg * thee )
```

FORTRAN stub object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to object to be destroyed

Definition at line 571 of file vpmg.c.

7.29.2.18 Vpmg_energy()

```
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 1248 of file vpmg.c.

7.29.2.19 Vpmg_fieldSpline4()

```
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

7.29.2.20 Vpmg_fillArray()

```
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 892 of file vpmg.c.

7.29.2.21 Vpmg_fillco()

```
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

Parameters

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 5655 of file vpmg.c.

7.29.2.22 Vpmg_force()

```
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 5822 of file vpmg.c.

7.29.2.23 Vpmg_ibDirectPolForce()

```
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

7.29.2.24 Vpmg_ibForce()

```
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 5845 of file vpmg.c.

7.29.2.25 Vpmg_ibMutualPolForce()

```
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

7.29.2.26 Vpmg_ibNLDirectPolForce()

```
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

Parameters

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

7.29.2.27 Vpmg_ibPermanentMultipoleForce()

```
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

7.29.2.28 Vpmg_memChk()

```
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 79 of file vpmg.c.

7.29.2.29 Vpmg_printColComp()

```
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 87 of file vpmg.c.

7.29.2.30 Vpmg_qfAtomEnergy()

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

Get the per-atom "fixed charge" contribution to the electrostatic energy.
Using the solution at the finest mesh level, get the electrostatic energy due to the interaction of the fixed charges with the potential:

$$G = qu(r),$$

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

Author

Nathan Baker

Note

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

Returns

The fixed charge electrostatic energy in units of k_B T.

Parameters

thee	The Vpmg object
atom	The atom for energy calculations

Definition at line 1791 of file vpmg.c.

7.29.2.31 Vpmg_qfDirectPolForce()

```
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

7.29.2.32 Vpmg_qfEnergy()

```
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: \f[ G = \sum_i q_i u(r_i) \f]
```

```
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 1687 of file vpmg.c.

7.29.2.33 Vpmg_qfForce()

```
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 6267 of file vpmg.c.

7.29.2.34 Vpmg_qfMutualPolForce()

```
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

7.29.2.35 Vpmg_qfNLDirectPolForce()

```
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

7.29.2.36 Vpmg_qfPermanentMultipoleEnergy()

```
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

7.29.2.37 Vpmg_qfPermanentMultipoleForce()

```
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

7.29.2.38 Vpmg_qmEnergy()

```
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:
   \f[ G = \frac{1}{4 I_s} \sum_i c_i q_i^2 \int
   \kappa^2(x) e^{-q_i u(x)} dx \f]
for the NPBE and
   \f[ G = \frac{1}{2} \int \overline{\kappa}^2(x) u^2(x) dx \f]
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 1386 of file vpmg.c.

7.29.2.39 Vpmg_setPart()

```
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 627 of file vpmg.c.

7.29.2.40 Vpmg_solve()

```
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 401 of file vpmg.c.

7.29.2.41 Vpmg_solveLaplace()

```
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 7042 of file vpmg.c.

7.29.2.42 Vpmg_unsetPart()

```
VEXTERNC void Vpmg_unsetPart (
            Vpmg * thee )
```

Remove partition restrictions.

Author

Nathan Baker

Parameters

thee	Vpmg object

Definition at line 872 of file vpmg.c.

7.30 Vpmgp class

Parameter structure for Mike Holst's PMGP code.

Files

- file vpmgp.c

  Class Vpmgp methods.
- file vpmgp.h

  Contains declarations for class Vpmgp.

Data Structures

- struct sVpmgp

  Contains public data members for Vpmgp class/module.

Typedefs

- typedef struct sVpmgp Vpmgp

  Declaration of the Vpmgp class as the sVpmgp structure.

Functions

- VEXTERNC Vpmgp * Vpmgp_ctor (MGparm *mgparm)

  Construct PMG parameter object and initialize to default values.
- VEXTERNC int Vpmgp_ctor2 (Vpmgp *thee, MGparm *mgparm)

  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.

7.30.1 Detailed Description

Parameter structure for Mike Holst's PMGP code.

Note

Variables and many default values taken directly from PMG

7.30.2 Function Documentation

7.30.2.1 Vpmgp_ctor()

```
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 76 of file vpmgp.c.

7.30.2.2 Vpmgp_ctor2()

```
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 93 of file vpmgp.c.

7.30.2.3 Vpmgp_dtor()

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

Object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to memory location for Vpmgp object

Definition at line 178 of file vpmgp.c.

7.30.2.4 Vpmgp_dtor2()

```
VEXTERNC void Vpmgp_dtor2 (
            Vpmgp * thee )
```

FORTRAN stub for object destructor.

Author

Nathan Baker

Parameters

thee	Pointer to Vpmgp object

Definition at line 193 of file vpmgp.c.

7.30.2.5 Vpmgp_makeCoarse()

```
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

Parameters

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 312 of file vpmgp.c.

7.30.2.6 Vpmgp_size()

```
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 196 of file vpmgp.c.

7.31 C translation of Holst group PMG code

C translation of Holst group PMG code.

Macros

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

  Multigrid subroutines.
- #define MAXIONS 50

  Specifies the PDE definition for PMG to solve.

Functions

- VPUBLIC void VbuildA (int ∗nx, int ∗ny, int ∗nz, int ∗ipkey, int ∗mgdisc, int ∗numdia, int ∗ipc, double ∗rpc, double ∗ac, double ∗cc, double ∗fc, double ∗xf, double ∗yf, double ∗zf, double ∗gxcf, double ∗gycf, double ∗gzcf, double ∗a1cf, double ∗a2cf, double ∗a3cf, double ∗ccf, double ∗fcf)

  Build the Laplacian.
- VPUBLIC void Vbuildband (int ∗key, int ∗nx, int ∗ny, int ∗nz, int ∗ipc, double ∗rpc, double ∗ac, int ∗ipcB, double ∗rpcB, double ∗acB)

  Banded matrix builder.
- 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.
- 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.
- 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 *o↩ PN, double *oPS, double *oPE, double *oPW, double *oPNE, double *oPNW, double *oPSE, double *oPSW, double *uPC, double *uPN, double *uPS, double *uPE, double *uPW, double *uPNE, double *uPNW, double *uPSE, double *uPSW, double *dPC, double *dPN, double *dPS, double *dPE, double *dPW, double *dPNE, double *dPNW, double *dPSE, double *dPSW, double *oC, double *oE, double *oN, double *uC, double *oNE, double *oNW, double *uE, double *uW, double *uN, double *uS, double *uNE, double *uNW, double *uSE, double *uSW, double *XoC, double *XoE, double *XoN, double *XuC, double *XoNE, double *XoNW, double *XuE, double *XuW, double *XuN, double *XuS, double *XuNE, double *XuNW, double *XuSE, double *XuSW)

  Compute a 27-point galerkin coarse grid matrix from a 27-point fine grid matrix.
- VPUBLIC void VbuildP (int *nxf, int *nyf, int *nzf, int *nxc, int *nyc, int *nzc, int *mgprol, int *ipc, double *rpc, double *pc, double *ac, double *xf, double *yf, double *zf)

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

  A collection of useful low-level routines (timing, etc).
- VPUBLIC void Vgsrb (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *ac, double *cc, double *fc, double *x, double *w1, double *w2, double *r, int *itmax, int *iters, double *errtol, double *omega, int *iresid, int *iadjoint)

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

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

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

  Break the matrix data-structure into diagonals and then call the residual routine.

- VEXTERNC void Vnmresid (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *ac, double *cc, double *fc, double *x, double *r, double *w1)

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

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

  Apply the prolongation operator.
- VEXTERNC void Vextrac (int *nxf, int *nyf, int *nzf, int *nxc, int *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.
- VPUBLIC void Vmgdriv (int *iparm, double *rparm, int *iwork, double *rwork, double *u, double *xf, double *yf, double *zf, double *gxcf, double *gycf, double *gzcf, double *a1cf, double *a2cf, double *a3cf, double *ccf, double *fcf, double *tcf)

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

  Solves the pde using the multi-grid method.
- VEXTERNC void Vmgsz (int *mgcoar, int *mgdisc, int *mgsolv, int *nx, int *ny, int *nz, int *nlev, int *nxc, int *nyc, int *nzc, int *nf, int *nc, int *narr, int *narrc, int *n_rpc, int *n_iz, int *n_ipc, int *iretot, int *iintot)

  This routine computes the required sizes of the real and integer work arrays for the multigrid code. these two sizes are a (complicated) function of input parameters.
- VPUBLIC void 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.
- 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.
- VEXTERNC void Vbuildstr (int *nx, int *ny, int *nz, int *nlev, int *iz)

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

  Form the Galerkin coarse grid system.

- VPUBLIC void Vxcopy (int *nx, int *ny, int *nz, double *x, double *y)

  A collection of useful low-level routines (timing, etc).
- VEXTERNC void Vxcopy_small (int *nx, int *ny, int *nz, double *x, double *y)

  Copy operation for a grid function with boundary values. Quite simply copies one 3d matrix to another.
- VEXTERNC void Vxcopy_large (int *nx, int *ny, int *nz, double *x, double *y)

  Copy operation for a grid function with boundary values. Quite simply copies one 3d matrix to another.
- VEXTERNC void Vxaxpy (int *nx, int *ny, int *nz, double *alpha, double *x, double *y)

  saxpy operation for a grid function with boundary values.
- VEXTERNC double 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)
- VPUBLIC void Vdpbsl (double *abd, int *lda, int *n, int *m, double *b)

  LINPACK interface.
- 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.
- 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.
- VPUBLIC void Vpower (int *nx, int *ny, int *nz, int *iz, int *ilev, int *ipc, double *rpc, double *ac, double *cc, double *w1, double *w2, double *w3, double *w4, double *eigmax, double *eigmax_model, double *tol, int *itmax, int *iters, int *iinfo)

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

  Standard inverse power method for minimum eigenvalue estimation.
- VEXTERNC void Vsmooth (int *nx, int *ny, int *nz, int *ipc, double *rpc, double *ac, double *cc, double *fc, double *x, double *w1, double *w2, double *r, int *itmax, int *iters, double *errtol, double *omega, int *iresid, int *iadjoint, int *meth)

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

  call the appropriate non-linear smoothing routine.

7.31.1 Detailed Description

C translation of Holst group PMG code.

7.31.2 Macro Definition Documentation

7.31.2.1 HARMO2

```
#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) 2010-2020 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition at line 65 of file mgsubd.h.

7.31.2.2 MAXIONS

```
#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) 2010-2020 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition at line 62 of file mypdec.h.

7.31.3 Function Documentation

7.31.3.1 Vaxrand()

```
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 291 of file mikpckd.c.

7.31.3.2 Vazeros()

```
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 195 of file mikpckd.c.

7.31.3.3 VbuildA()

```
VEXTERNC void VbuildA (
            int * nx,
            int * ny,
            int * nz,
            int * ipkey,
            int * mgdisc,
            int * numdia,
            int * ipc,
            double * rpc,
            double * ac,
            double * cc,
            double * fc,
            double * xf,
            double * yf,
```

```
            double * zf,
            double * gxcf,
            double * gycf,
            double * gzcf,
            double * a1cf,
            double * a2cf,
            double * a3cf,
            double * ccf,
            double * fcf )
```

Build the Laplacian.

Author

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

Version

$Id:

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 Battelle Memorial Institute.
* Developed at the Pacific Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific No
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Author

Mike Holst [original], Tucker Beck [translation]

Attention

Break the matrix data-structure into diagonals and then call the matrix build routine

Author

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

Parameters

nx	
ny	
nz	
ipkey	
mgdisc	
numdia	
ipc	
rpc	
ac	

Parameters

cc	
fc	
xf	
yf	
zf	
gxcf	
gycf	
gzcf	
a1cf	
a2cf	
a3cf	
ccf	
fcf	

Definition at line 57 of file buildAd.c.

7.31.3.4 Vbuildband()

```
VEXTERNC void Vbuildband (
              int * key,
              int * nx,
              int * ny,
              int * nz,
              int * ipc,
              double * rpc,
              double * ac,
              int * ipcB,
              double * rpcB,
              double * acB )
```

Banded matrix builder.

Author

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

Version

$Id:

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Developed at the Pacific Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific No
* Copyright (c) 2010-2020 Battelle Memorial Institute.
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2020, Nathan A. Baker.
```

```
* Portions Copyright (c) 1999-2002, The Regents of the University of California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Author

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) 2010-2020 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
```

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

Build and factor a banded matrix given a matrix in diagonal form.

Author

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

Note

Replaces buildband from buildBd.f

Parameters

key	
nx	
ny	
nz	
ipc	
rpc	
ac	
ipcB	
rpcB	
acB	

Definition at line 57 of file buildBd.c.

7.31.3.5 Vbuildband1_27()

```
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 183 of file buildBd.c.

7.31.3.6 Vbuildband1_7()

```
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 119 of file buildBd.c.

7.31.3.7 VbuildG()

```
VEXTERNC void VbuildG (
            int * nxf,
```

```
            int * nyf,
            int * nzf,
            int * nxc,
            int * nyc,
            int * nzc,
            int * numdia,
            double * pcFF,
            double * acFF,
            double * ac )
```

Build Galerkin matrix structures.

Author

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

Version

$Id:

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 Battelle Memorial Institute.
* Developed at the Pacific Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific No
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Author

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

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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 57 of file buildGd.c.

7.31.3.8 VbuildG_1()

```
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), -Xd↩
SE(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), -Xo↩
SE(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), -Xu↩
SE(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	

Parameters

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 145 of file buildGd.c.

7.31.3.9 VbuildG_27()

```
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), -Xd↩
SE(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), -Xo↩
SE(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), -Xu↩
SE(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	

Parameters

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 1252 of file buildGd.c.

7.31.3.10 VbuildG_7()

```
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), -Xd↩
SE(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), -Xo↩
SE(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), -Xu↩
SE(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) RE-
TURN(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	

Parameters

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 450 of file buildGd.c.

7.31.3.11 Vbuildgaler0()

```
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 341 of file mgsubd.c.

7.31.3.12 Vbuildops()

```
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.

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) 2010-2020 Battelle Memorial Institute.
* Developed at the Pacific Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific No
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2020, 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.
*
*
```

Note

The fine level must be build before any coarse levels.

Author

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

Replaces buildops from mgsubd.f

Parameters

nx	
ny	
nz	
nlev	
ipkey	
iinfo	
ido	
iz	
mgprol	
mgcoar	
mgsolv	
mgdisc	
ipc	
rpc	
pc	
ac	
cc	
fc	
xf	
yf	
zf	
gxcf	
gycf	

Parameters

gzcf	
a1cf	
a2cf	
a3cf	
ccf	
fcf	
tcf	

Definition at line 57 of file mgsubd.c.

7.31.3.13 VbuildP()

```
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.

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) 2010-2020 Battelle Memorial Institute.
* Developed at the Pacific Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific No
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
```

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

Author

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

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) 2010-2020 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 57 of file buildPd.c.

7.31.3.14 Vbuildstr()

```
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 257 of file mgsubd.c.

7.31.3.15 Vc_vec()

```
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 121 of file mypdec.c.

7.31.3.16 Vc_vecpmg()

```
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 133 of file mypdec.c.

7.31.3.17 Vc_vecsmpbe()

```
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 213 of file mypdec.c.

7.31.3.18 Vcghs()

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

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

Author

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

Version

$Id:

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 Battelle Memorial Institute.
* Developed at the Pacific Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific No
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
```

```
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Author

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

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) 2010-2020 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 57 of file cgd.c.

7.31.3.19 Vdc_vec()

```
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	

Parameters

nx	
ny	
nz	
ipkey	

Definition at line 341 of file mypdec.c.

7.31.3.20 Vdpbsl()

```
VEXTERNC void Vdpbsl (
             double * abd,
             int * lda,
             int * n,
             int * m,
             double * b )
```

LINPACK interface.

Author

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

Version

$Id:

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 Battelle Memorial Institute.
* Developed at the Pacific Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific No
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
```

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

Author

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

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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

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 57 of file mlinpckd.c.

7.31.3.21 Vextrac()

```
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 1078 of file matvecd.c.

7.31.3.22 VfboundPMG()

```
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 209 of file mikpckd.c.

7.31.3.23 VfboundPMG00()

```
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

Parameters

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 258 of file mikpckd.c.

7.31.3.24 Vfmvfas()

```
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.

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) 2010-2020 Battelle Memorial Institute.
* Developed at the Pacific Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific No
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Author

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

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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:
```

Nested iteration for a nonlinear multilevel method. Algorithm: nonlinear multigrid iteration (fas)
this routine is the full multigrid front-end for a multigrid v-cycle solver. in other words, at repeatedly calls the v-cycle multigrid solver on successively finer and finer grids.

Note

Author

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

Replaces fmvfas from mgfasd.f

Parameters

nx	
ny	
nz	
x	
iz	
w0	
w1	
w2	
w3	
w4	
istop	
itmax	
iters	
ierror	
nlev	
ilev	
nlev_real	
mgsolv	

Parameters

iok	
iinfo	
epsiln	
errtol	
omega	
nu1	
nu2	
mgsmoo	
ipc	
rpc	
pc	
ac	
cc	
fc	
tru	

Definition at line 57 of file mgfasd.c.

7.31.3.25 Vfnewton()

```
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]

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) 2010-2020 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) 2002-2020, Nathan A. Baker.  Portions Copyright (c) 1999-2002,
* The Regents of the University of California. Portions Copyright (c) 1995,
* Michael Holst.
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Author

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

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Nested iteration for an inexact-newton-multilevel method.

Author

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

Note

Replaces fnewton from newtond.f

Parameters

nx	
ny	
nz	
x	
iz	
w0	

Parameters

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.

7.31.3.26 Vgetjac()

```
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.

7.31.3.27 Vgsrb()

```
VEXTERNC void Vgsrb (
             int * nx,
             int * ny,
             int * nz,
             int * ipc,
             double * rpc,
             double * ac,
             double * cc,
             double * fc,
             double * x,
             double * w1,
             double * w2,
             double * r,
             int * itmax,
             int * iters,
             double * errtol,
             double * omega,
             int * iresid,
             int * iadjoint )
```

Guass-Seidel solver.

Author

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

Version

$Id:

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 Battelle Memorial Institute.
* Developed at the Pacific Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific No
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Author

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

Attention

Call the fast diagonal iterative method.

Author

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

Note

Replaces gsrb from gsd.f

Parameters

nx	
ny	
nz	
ipc	
rpc	
ac	
cc	
fc	
x	

Parameters

w1	
w2	
r	
itmax	
iters	
errtol	
omega	
iresid	
iadjoint	

Definition at line 57 of file gsd.c.

7.31.3.28 VinterpPMG()

```
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 915 of file matvecd.c.

7.31.3.29 Vipower()

```
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 165 of file powerd.c.

7.31.3.30 Vmatvec()

```
VEXTERNC void Vmatvec (
             int * nx,
             int * ny,
             int * nz,
             int * ipc,
             double * rpc,
             double * ac,
             double * cc,
             double * x,
             double * y )
```

Matrix-vector multiplication routines.

Author

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

Version

$Id:

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 Battelle Memorial Institute.
* Developed at the Pacific Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific No
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Author

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

Attention

Break the matrix data-structure into diagonals and then call the matrix-vector routine.

Author

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

Note

Replaces matvec from matvecd.f

Parameters

nx	
ny	
nz	
ipc	
rpc	
ac	
cc	
x	
y	

Definition at line 57 of file matvecd.c.

7.31.3.31 Vmgdriv()

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

Multilevel solver driver.

Author

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

Version

$Id:

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 Battelle Memorial Institute.
* Developed at the Pacific Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific No
* Portions Copyright (c) 2002-2012, Washington University in St. Louis.
* Portions Copyright (c) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
```

```
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Author

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

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) 2010-2020 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Author

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

Replaces mgdriv from mgdrvd.f

Parameters

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

Definition at line 57 of file mgdrvd.c.

7.31.3.32 Vmgdriv2()

```
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	

Parameters

gzcf	
a1cf	
a2cf	
a3cf	
ccf	
fcf	
tcf	

Definition at line 190 of file mgdrvd.c.

7.31.3.33 Vmgsz()

```
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 562 of file mgdrvd.c.

7.31.3.34 Vmresid()

```
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 426 of file matvecd.c.

7.31.3.35 Vmvcs()

```
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) 2010-2020 Battelle Memorial Institute.
* Developed at the Pacific Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific No
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
```

```
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Author

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

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) 2010-2020 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

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	

Parameters

ipc	
rpc	
pc	
ac	
cc	
fc	
tru	

Definition at line 57 of file mgcsd.c.

7.31.3.36 Vmvfas()

```
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	

Parameters

fc	
tru	

Definition at line 157 of file mgfasd.c.

7.31.3.37 Vmypdefinitlpbe()

```
VEXTERNC void Vmypdefinitlpbe (
            int * tnion,
            double * tcharge,
            double * tsconc )
```

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

Author

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

Version

$Id:

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 Battelle Memorial Institute.
* Developed at the Pacific Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific No
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
```

```
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Author

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

Note

Replaces mypdefinitlpbe 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 57 of file mypdec.c.

7.31.3.38 Vmypdefinitnpbe()

```
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 75 of file mypdec.c.

7.31.3.39 Vmypdefinitsmpbe()

```
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 93 of file mypdec.c.

7.31.3.40 Vnewdriv()

```
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) 2010-2020 Battelle Memorial Institute.
* Developed at the Pacific Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific No
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Author

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

Version

$Id:

Attention

Driver for a screaming inexact-newton-multilevel solver.

Author

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

Note

Replaces newdriv from newdrvd.f

Parameters

iparm	
rparm	
iwork	
rwork	
u	
xf	
yf	
zf	
gxcf	

Parameters

gycf	
gzcf	
a1cf	
a2cf	
a3cf	
ccf	
fcf	
tcf	

Definition at line 57 of file newdrvd.c.

7.31.3.41 Vnewdriv2()

```
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 169 of file newdrvd.c.

7.31.3.42 Vnewton()

```
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	

Parameters

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.

7.31.3.43 Vnmatvec()

```
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 232 of file matvecd.c.

7.31.3.44 Vnmresid()

```
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 598 of file matvecd.c.

7.31.3.45 Vnsmooth()

```
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	

Parameters

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

Definition at line 98 of file smoothd.c.

7.31.3.46 Vpower()

```
VEXTERNC void Vpower (
            int * nx,
            int * ny,
            int * nz,
            int * iz,
            int * ilev,
            int * ipc,
            double * rpc,
            double * ac,
            double * cc,
            double * w1,
            double * w2,
            double * w3,
            double * w4,
            double * eigmax,
            double * eigmax_model,
            double * tol,
            int * itmax,
            int * iters,
            int * iinfo )
```

Power methods for eigenvalue estimation.

Author

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

Version

$Id:

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 Battelle Memorial Institute.
* Developed at the Pacific Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific No
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Author

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

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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:
```

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

Vpower is yet untested as a call stack including it hasn't been found

Parameters

nx	
ny	
nz	
iz	
ilev	
ipc	
rpc	
ac	
cc	
w1	
w2	
w3	
w4	
eigmax	

Parameters

eigmax_model	
tol	
itmax	
iters	
iinfo	

Definition at line 57 of file powerd.c.

7.31.3.47 Vprtmatd()

```
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 332 of file mikpckd.c.

7.31.3.48 Vrestrc()

```
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 782 of file matvecd.c.

7.31.3.49 Vsmooth()

```
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]

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) 2010-2020 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) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Author

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

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 Battelle Memorial Institute. Developed at the Pacific Northwest National Laboratory
```

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

call the appropriate linear smoothing routine.

Author

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

Note

Replaces smooth from smoothd.f

Parameters

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

Parameters

iadjoint	
meth	

Definition at line 58 of file smoothd.c.

7.31.3.50 Vxaxpy()

```
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 112 of file mikpckd.c.

7.31.3.51 Vxcopy()

```
VEXTERNC void Vxcopy (
            int * nx,
            int * ny,
            int * nz,
            double * x,
            double * y )
```

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

Author

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

Version

$Id:

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 Battelle Memorial Institute.
* Developed at the Pacific Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific No
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Author

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

Version

$Id:

Attention

Copy operation for a grid function with boundary values. Quite simply copies one 3d matrix to another

Author

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

Note

Replaces xcopy from mikpckd.f

Parameters

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 57 of file mikpckd.c.

7.31.3.52 Vxcopy_large()

```
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 91 of file mikpckd.c.

7.31.3.53 Vxcopy_small()

```
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

Parameters

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 75 of file mikpckd.c.

7.31.3.54 Vxdot()

```
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 173 of file mikpckd.c.

7.31.3.55 Vxnrm1()

```
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 131 of file mikpckd.c.

7.31.3.56 Vxnrm2()

```
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 152 of file mikpckd.c.

7.31.3.57 Vxscal()

```
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 318 of file mikpckd.c.

7.32 High-level front-end routines

Files

- file apbs.h

  Header file for header dependencies.
- file main.c

  APBS "front end" program using formatted input files.
- file routines.h

  Header file for front end auxiliary routines.

Data Structures

- struct AtomForce

  Structure to hold atomic forces.

Macros

- #define APBSRC 13

  Return code for APBS during failure.

Typedefs

- typedef struct AtomForce AtomForce

  Define AtomForce type.

Functions

- int main (int argc, char **argv)

  The main APBS function.
- VPUBLIC Vrc_Codes initFE (int icalc, NOsh *nosh, FEMparm *feparm, PBEparm *pbeparm, Vpbe *pbe[NOSH_MAXCALC], Valist *alist[NOSH_MAXMOL], Vfetk *fetk[NOSH_MAXCALC])

  Initialize FE solver objects.

- VEXTERNC Vparam * loadParameter (NOsh *nosh)

  Loads and returns parameter object.
- VEXTERNC int loadMolecules (NOsh *nosh, Vparam *param, Valist *alist[NOSH_MAXMOL])

  Load the molecules given in NOsh into atom lists.
- VEXTERNC void killMolecules (NOsh *nosh, Valist *alist[NOSH_MAXMOL])

  Destroy the loaded molecules.
- VEXTERNC int loadDielMaps (NOsh *nosh, Vgrid *dielXMap[NOSH_MAXMOL], Vgrid *dielYMap[NOSH_MAXMOL], Vgrid *dielZMap[NOSH_MAXMOL])

  Load the dielectric maps given in NOsh into grid objects.
- VEXTERNC void killDielMaps (NOsh *nosh, Vgrid *dielXMap[NOSH_MAXMOL], Vgrid *dielYMap[NOSH_MAXMOL], Vgrid *dielZMap[NOSH_MAXMOL])

  Destroy the loaded dielectric.
- VEXTERNC int loadKappaMaps (NOsh *nosh, Vgrid *kappa[NOSH_MAXMOL])

  Load the kappa maps given in NOsh into grid objects.
- VEXTERNC void killKappaMaps (NOsh *nosh, Vgrid *kappa[NOSH_MAXMOL])

  Destroy the loaded kappa maps.
- VEXTERNC int loadPotMaps (NOsh *nosh, Vgrid *pot[NOSH_MAXMOL])

  Load the potential maps given in NOsh into grid objects.
- VEXTERNC void killPotMaps (NOsh *nosh, Vgrid *pot[NOSH_MAXMOL])

  Destroy the loaded potential maps.
- VEXTERNC int loadChargeMaps (NOsh *nosh, Vgrid *charge[NOSH_MAXMOL])

  Load the charge maps given in NOsh into grid objects.
- VEXTERNC void killChargeMaps (NOsh *nosh, Vgrid *charge[NOSH_MAXMOL])

  Destroy the loaded charge maps.
- VEXTERNC void printPBEPARM (PBEparm *pbeparm)

  Print out generic PBE params loaded from input.
- VEXTERNC void printMGPARM (MGparm *mgparm, double realCenter[3])

  Print out MG-specific params loaded from input.
- VEXTERNC int initMG (int icalc, NOsh *nosh, MGparm *mgparm, PBEparm *pbeparm, double realCenter[3], Vpbe *pbe[NOSH_MAXCALC], Valist *alist[NOSH_MAXMOL], Vgrid *dielXMap[NOSH_MAXMOL], Vgrid *dielYMap[NOSH_MAXMOL], Vgrid *dielZMap[NOSH_MAXMOL], Vgrid *kappaMap[NOSH_MAXMOL], Vgrid *chargeMap[NOSH_MAXMOL], Vpmgp *pmgp[NOSH_MAXCALC], Vpmg *pmg[NOSH_MAXCALC], Vgrid *potMap[NOSH_MAXMOL])

  Initialize an MG calculation.
- VEXTERNC void killMG (NOsh *nosh, Vpbe *pbe[NOSH_MAXCALC], Vpmgp *pmgp[NOSH_MAXCALC], Vpmg *pmg[NOSH_MAXCALC])

  Kill structures initialized during an MG calculation.
- VEXTERNC int solveMG (NOsh *nosh, Vpmg *pmg, MGparm_CalcType type)

  Solve the PBE with MG.
- VEXTERNC int setPartMG (NOsh *nosh, MGparm *mgparm, Vpmg *pmg)

  Set MG partitions for calculating observables and performing I/O.
- VEXTERNC int energyMG (NOsh *nosh, int icalc, Vpmg *pmg, int *nenergy, double *totEnergy, double *qf↩ Energy, double *qmEnergy, double *dielEnergy)

  Calculate electrostatic energies from MG solution.
- VEXTERNC void killEnergy ()

  Kill arrays allocated for energies.
- VEXTERNC int forceMG (Vmem *mem, NOsh *nosh, PBEparm *pbeparm, MGparm *mgparm, Vpmg *pmg, int *nforce, AtomForce **atomForce, Valist *alist[NOSH_MAXMOL])

Calculate forces from MG solution.

- VEXTERNC void killForce (Vmem *mem, NOsh *nosh, int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC])

  Free memory from MG force calculation.

- VEXTERNC void storeAtomEnergy (Vpmg *pmg, int icalc, double **atomEnergy, int *nenergy)

  Store energy in arrays for future use.

- VEXTERNC int writedataFlat (NOsh *nosh, Vcom *com, const char *fname, double totEnergy[NOSH_MAXCALC], double qfEnergy[NOSH_MAXCALC], double qmEnergy[NOSH_MAXCALC], double dielEnergy[NOSH_MAXCALC], int nenergy[NOSH_MAXCALC], double *atomEnergy[NOSH_MAXCALC], int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC])

  Write out information to a flat file.

- VEXTERNC int writedataXML (NOsh *nosh, Vcom *com, const char *fname, double totEnergy[NOSH_MAXCALC], double qfEnergy[NOSH_MAXCALC], double qmEnergy[NOSH_MAXCALC], double dielEnergy[NOSH_MAXCALC], int nenergy[NOSH_MAXCALC], double *atomEnergy[NOSH_MAXCALC], int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC])

  Write out information to an XML file.

- VEXTERNC int writedataMG (int rank, NOsh *nosh, PBEparm *pbeparm, Vpmg *pmg)

  Write out observables from MG calculation to file.

- VEXTERNC int writematMG (int rank, NOsh *nosh, PBEparm *pbeparm, Vpmg *pmg)

  Write out operator matrix from MG calculation to file.

- VEXTERNC double returnEnergy (Vcom *com, NOsh *nosh, double totEnergy[NOSH_MAXCALC], int iprint)

  Access net local energy.

- VEXTERNC int printEnergy (Vcom *com, NOsh *nosh, double totEnergy[NOSH_MAXCALC], int iprint)

  Combine and pretty-print energy data (deprecated...see printElecEnergy)

- VEXTERNC int printElecEnergy (Vcom *com, NOsh *nosh, double totEnergy[NOSH_MAXCALC], int iprint)

  Combine and pretty-print energy data.

- VEXTERNC int printApolEnergy (NOsh *nosh, int iprint)

  Combine and pretty-print energy data.

- VEXTERNC int printForce (Vcom *com, NOsh *nosh, int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC], int i)

  Combine and pretty-print force data (deprecated...see printElecForce)

- VEXTERNC int printElecForce (Vcom *com, NOsh *nosh, int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC], int i)

  Combine and pretty-print force data.

- VEXTERNC int printApolForce (Vcom *com, NOsh *nosh, int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC], int i)

  Combine and pretty-print force data.

- VEXTERNC void startVio ()

  Wrapper to start MALOC Vio layer.

- VEXTERNC int energyAPOL (APOLparm *apolparm, double sasa, double sav, double atomsasa[ ], double atomwcaEnergy[ ], int numatoms)

  Calculate non-polar energies.

- VEXTERNC int forceAPOL (Vacc *acc, Vmem *mem, APOLparm *apolparm, int *nforce, AtomForce **atomForce, Valist *alist, Vclist *clist)

  Calculate non-polar forces.

- VEXTERNC int initAPOL (NOsh *nosh, Vmem *mem, Vparam *param, APOLparm *apolparm, int *nforce, AtomForce **atomForce, Valist *alist)

  Upperlevel routine to the non-polar energy and force routines.

- VEXTERNC void printFEPARM (int icalc, NOsh *nosh, FEMparm *feparm, Vfetk *fetk[NOSH_MAXCALC])

Print out FE-specific params loaded from input.

- VEXTERNC int energyFE (NOsh ∗nosh, int icalc, Vfetk ∗fetk[NOSH_MAXCALC], int ∗nenergy, double ∗totEnergy, double ∗qfEnergy, double ∗qmEnergy, double ∗dielEnergy)

  Calculate electrostatic energies from FE solution.
- VEXTERNC void killFE (NOsh ∗nosh, Vpbe ∗pbe[NOSH_MAXCALC], Vfetk ∗fetk[NOSH_MAXCALC], Gem ∗gem[NOSH_MAXMOL])

  Kill structures initialized during an FE calculation.
- VEXTERNC int preRefineFE (int i, FEMparm ∗feparm, Vfetk ∗fetk[NOSH_MAXCALC])

  Pre-refine mesh before solve.
- VEXTERNC int partFE (int i, NOsh ∗nosh, FEMparm ∗feparm, Vfetk ∗fetk[NOSH_MAXCALC])

  Partition mesh (if applicable)
- VEXTERNC int solveFE (int i, PBEparm ∗pbeparm, FEMparm ∗feparm, Vfetk ∗fetk[NOSH_MAXCALC])

  Solve-estimate-refine.
- VEXTERNC int postRefineFE (int icalc, FEMparm ∗feparm, Vfetk ∗fetk[NOSH_MAXCALC])

  Estimate error, mark mesh, and refine mesh after solve.
- VEXTERNC int writedataFE (int rank, NOsh ∗nosh, PBEparm ∗pbeparm, Vfetk ∗fetk)

  Write FEM data to files.
- VEXTERNC Vrc_Codes loadMeshes (NOsh ∗nosh, Gem ∗gm[NOSH_MAXMOL])

  Load the meshes given in NOsh into geometry objects.
- VEXTERNC void killMeshes (NOsh ∗nosh, Gem ∗alist[NOSH_MAXMOL])

  Destroy the loaded meshes.

7.32.1 Detailed Description

7.32.2 Function Documentation

7.32.2.1 energyAPOL()

```
VEXTERNC int energyAPOL (
            APOLparm * apolparm,
            double sasa,
            double sav,
            double atomsasa[],
            double atomwcaEnergy[],
            int numatoms )
```

Calculate non-polar energies.

Author

David Gohara

Returns

1 if successful, 0 otherwise

Parameters

sasa	APOLparm object
sav	Solvent accessible surface area
atomsasa	Solvent accessible volume
atomwcaEnergy	Array for SASA per atom ∗
numatoms	Array for WCA energy per atom ∗ Number of atoms (or size of the above arrays) ∗

Definition at line 4674 of file routines.c.

7.32.2.2 energyFE()

```
VEXTERNC int energyFE (
            NOsh * nosh,
            int icalc,
            Vfetk * fetk[NOSH_MAXCALC],
            int * nenergy,
            double * totEnergy,
            double * qfEnergy,
            double * qmEnergy,
            double * dielEnergy )
```

Calculate electrostatic energies from FE solution.

Author

Nathan Baker

Parameters

nosh	Object with parsed input file parameters
icalc	Index of calculation
fetk	FE object array
nenergy	Set to number of entries in energy arrays
totEnergy	Set to total energy (in kT)
qfEnergy	Set to charge-potential energy (in kT)
qmEnergy	Set to mobile ion energy (in kT)
dielEnergy	Set to polarization energy (in kT)

Bug "calcenergy 2" does not work

Returns

1 if successful, 0 otherwise

Calculates the electrostatic energies from an FE calculation. < FE-specific parameters
< PBE-specific parameters
If we're not ignoring this particular NOsh object because it has been rendered invalid, call the Vfetk object's energy calculation function. The flag differences specified have to do with setting specific calculation restrictions (see color variable documentation in function code).

Parameters

nosh	Object with parsed input file parameters
icalc	Calculation index
fetk	FE object array
nenergy	Set to number of entries in energy arrays
totEnergy	Set to total energy (in kT)
qfEnergy	Set to charge-potential energy (in kT)
qmEnergy	Set to mobile ion energy (in kT)
dielEnergy	Set to polarization energy (in kT)

Definition at line 4179 of file routines.c.

7.32.2.3 energyMG()

```
VEXTERNC int energyMG (
            NOsh * nosh,
            int icalc,
            Vpmg * pmg,
            int * nenergy,
            double * totEnergy,
            double * qfEnergy,
            double * qmEnergy,
            double * dielEnergy )
```

Calculate electrostatic energies from MG solution.

Author

Nathan Baker

Parameters

nosh	Object with parsed input file parameters
icalc	Index of calculation
pmg	MG object
nenergy	Set to number of entries in energy arrays
totEnergy	Set to total energy (in kT)
qfEnergy	Set to charge-potential energy (in kT)
qmEnergy	Set to mobile ion energy (in kT)
dielEnergy	Set to polarization energy (in kT)

Returns

1 if successful, 0 otherwise

Definition at line 1569 of file routines.c.

7.32.2.4 forceAPOL()

```
VEXTERNC int forceAPOL (
            Vacc * acc,
            Vmem * mem,
            APOLparm * apolparm,
            int * nforce,
            AtomForce ** atomForce,
            Valist * alist,
            Vclist * clist )
```

Calculate non-polar forces.

Author

David Gohara

Returns

1 if successful, 0 otherwise

Parameters

acc	Accessiblity object
mem	Memory manager
apolparm	Apolar calculation parameter object
nforce	Number of atomic forces to calculate statements for
atomForce	Object for storing atom forces
alist	Atom list
clist	Cell list for accessibility object

Definition at line 4729 of file routines.c.

7.32.2.5 forceMG()

```
VEXTERNC int forceMG (
            Vmem * mem,
            NOsh * nosh,
            PBEparm * pbeparm,
            MGparm * mgparm,
            Vpmg * pmg,
            int * nforce,
            AtomForce ** atomForce,
            Valist * alist[NOSH_MAXMOL] )
```

Calculate forces from MG solution.

Author

Nathan Baker

Parameters

mem	Memory management object
nosh	Parameters from input file
pbeparm	Generic PBE parameters
mgparm	MG-specific parmaeters
pmg	MG object
nforce	Set to number of forces in arrays
atomForce	List of atom forces
alist	List of atom lists

Returns

1 if successful, 0 otherwise

Definition at line 1636 of file routines.c.

7.32.2.6 initAPOL()

```
VEXTERNC int initAPOL (
            NOsh * nosh,
            Vmem * mem,
            Vparam * param,
            APOLparm * apolparm,
            int * nforce,
            AtomForce ** atomForce,
            Valist * alist )
```

Upperlevel routine to the non-polar energy and force routines.

Author

David Gohara

Returns

1 if successful, 0 otherwise

<
< Number of atoms
< Used to capture length of loops to prevent multiple calls in counters
<
<
< Temporary timing variable for debugging (PCE)
<
<
<
<
<
<
<
<
<
<
<
<
<
<
<
< WCA energy per atom
<
<
<
<
<
<
<
<
<
<
<
<
<
<
<
<

Parameters

nosh	Input parameter object
mem	Memory manager
param	Atom parameters
apolparm	Apolar calculation parameters
nforce	Number of force calculations
atomForce	Atom force storage object
alist	Atom list

Definition at line 4475 of file routines.c.

7.32.2.7 initFE()

```
VEXTERNC Vrc_Codes initFE (
            int icalc,
            NOsh * nosh,
            FEMparm * feparm,
            PBEparm * pbeparm,
            Vpbe * pbe[NOSH_MAXCALC],
            Valist * alist[NOSH_MAXMOL],
            Vfetk * fetk[NOSH_MAXCALC] )
```

Initialize FE solver objects.

Author

Nathan Baker

Bug THIS FUNCTION IS HARD-CODED TO SOLVE LRPBE

Author

Nathan Baker

Bug THIS FUNCTION IS HARD-CODED TO SOLVE LRPBE

< Loop counter
< Mesh ID
< Loop counter
< Loop counter
< Molecule ID
<
< I/O socket for reading MCSF mesh data
< Total bytes used by this operation
< High-water memory usage for this operation
< The type of mesh being used (see struct for enum values)
<
<
<
<
<
< Return codes for function calls (see struct for enum value)
< List of atoms being operated on
< Atom/molecule being operated on

Parameters

icalc	Index in pb, fetk to initialize (calculation index)
nosh	Master parmaeter object
feparm	FE-specific parameters
pbeparm	Generic PBE parameters
pbe	Array of PBE objects
alist	Array of atom lists
fetk	Array of finite element objects

Definition at line 3711 of file routines.c.

7.32.2.8 initMG()

```
VEXTERNC int initMG (
             int icalc,
             NOsh * nosh,
             MGparm * mgparm,
             PBEparm * pbeparm,
             double realCenter[3],
             Vpbe * pbe[NOSH_MAXCALC],
             Valist * alist[NOSH_MAXMOL],
             Vgrid * dielXMap[NOSH_MAXMOL],
             Vgrid * dielYMap[NOSH_MAXMOL],
             Vgrid * dielZMap[NOSH_MAXMOL],
             Vgrid * kappaMap[NOSH_MAXMOL],
             Vgrid * chargeMap[NOSH_MAXMOL],
             Vpmgp * pmgp[NOSH_MAXCALC],
             Vpmg * pmg[NOSH_MAXCALC],
             Vgrid * potMap[NOSH_MAXMOL] )
```

Initialize an MG calculation.

Author

Nathan Baker

Returns

1 if succesful, 0 otherwise

Initialize a multigrid calculation.

Parameters

icalc	Index of calculation in pmg/pmpg arrays
nosh	Object with parsed input file parameters
mgparm	Object with MG-specific parameters
pbeparm	Object with generic PBE parameters
realCenter	The actual center of the current mesh
pbe	Array of Vpbe objects (one for each calc)
alist	Array of atom lists
dielXMap	Array of x-shifted dielectric maps

Parameters

dielYMap	Array of y-shifted dielectric maps
dielZMap	Array of z-shifted dielectric maps
kappaMap	Array of kappa maps
chargeMap	Array of charge maps
pmgp	Array of MG parameter objects (one for each calc)
pmg	Array of MG objects (one for each calc)
potMap	Array of potential maps

Definition at line 1212 of file routines.c.

7.32.2.9 killChargeMaps()

```
VEXTERNC void killChargeMaps (
            NOsh * nosh,
            Vgrid * charge[NOSH_MAXMOL] )
```

Destroy the loaded charge maps.

Author

Nathan Baker

Parameters

nosh	NOsh object with input file information
charge	List of charge maps

Definition at line 984 of file routines.c.

7.32.2.10 killDielMaps()

```
VEXTERNC void killDielMaps (
            NOsh * nosh,
            Vgrid * dielXMap[NOSH_MAXMOL],
            Vgrid * dielYMap[NOSH_MAXMOL],
            Vgrid * dielZMap[NOSH_MAXMOL] )
```

Destroy the loaded dielectric.

Author

Nathan Baker

Parameters

nosh	NOsh object with input file information
dielXMap	List of x-shifted dielectric maps
dielYMap	List of y-shifted dielectric maps
dielZMap	List of x-shifted dielectric maps

Definition at line 639 of file routines.c.

7.32.2.11 killEnergy()

```
VEXTERNC void killEnergy ( )
```

Kill arrays allocated for energies.

Author

Nathan Baker

Definition at line 1774 of file routines.c.

7.32.2.12 killFE()

```
VEXTERNC void killFE (
            NOsh * nosh,
            Vpbe * pbe[NOSH_MAXCALC],
            Vfetk * fetk[NOSH_MAXCALC],
            Gem * gem[NOSH_MAXMOL] )
```

Kill structures initialized during an FE calculation.

Author

Nathan Baker

Parameters

pbe	Object with parsed input file parameters
fetk	Array of Vpbe objects for each calc
gem	Array of FEtk objects for each calc Array of geometry manager objects for each calc

Definition at line 3683 of file routines.c.

7.32.2.13 killForce()

```
VEXTERNC void killForce (
            Vmem * mem,
            NOsh * nosh,
            int nforce[NOSH_MAXCALC],
            AtomForce * atomForce[NOSH_MAXCALC] )
```

Free memory from MG force calculation.

Author

Nathan Baker

Parameters

mem	Memory management object
nosh	Parameters from input file
nforce	Number of forces in arrays
atomForce	List of atom forces

Definition at line 1782 of file routines.c.

7.32.2.14 killKappaMaps()

```
VEXTERNC void killKappaMaps (
            NOsh * nosh,
            Vgrid * kappa[NOSH_MAXMOL] )
```

Destroy the loaded kappa maps.

Author

Nathan Baker

Parameters

nosh	NOsh object with input file information
kappa	List of kappa maps

Definition at line 776 of file routines.c.

7.32.2.15 killMeshes()

```
VEXTERNC void killMeshes (
            NOsh * nosh,
            Gem * alist[NOSH_MAXMOL] )
```

Destroy the loaded meshes.

Author

Nathan Baker

Parameters

nosh	NOsh object with input file information
alist	Populated list of geometry objects to be destroyed

7.32.2.16 killMG()

```
VEXTERNC void killMG (
            NOsh * nosh,
            Vpbe * pbe[NOSH_MAXCALC],
            Vpmgp * pmgp[NOSH_MAXCALC],
            Vpmg * pmg[NOSH_MAXCALC] )
```

Kill structures initialized during an MG calculation.

Author

Nathan Baker

Parameters

pbe	Object with parsed input file parameters
pmgp	Array of Vpbe objects for each calc
pmg	Array of MG parameter objects for each calc Array of MG objects for each calc

Definition at line 1461 of file routines.c.

7.32.2.17 killMolecules()

```
VEXTERNC void killMolecules (
            NOsh * nosh,
            Valist * alist[NOSH_MAXMOL] )
```

Destroy the loaded molecules.

Author

Nathan Baker

Parameters

nosh	NOsh object with input file information
alist	List of atom list objects

Definition at line 233 of file routines.c.

7.32.2.18 killPotMaps()

```
VEXTERNC void killPotMaps (
            NOsh * nosh,
            Vgrid * pot[NOSH_MAXMOL] )
```

Destroy the loaded potential maps.

Author

David Gohara

Parameters

nosh	NOsh object with input file information
pot	List of potential maps

Definition at line 865 of file routines.c.

7.32.2.19 loadChargeMaps()

```
VEXTERNC int loadChargeMaps (
            NOsh * nosh,
            Vgrid * map[NOSH_MAXMOL] )
```

Load the charge maps given in NOsh into grid objects.

Author

Nathan Baker

Parameters

nosh	NOsh object with input file information
charge	List of kappa maps

Returns

1 if successful, 0 otherwise

0 on failure, 1 on success

Definition at line 884 of file routines.c.

7.32.2.20 loadDielMaps()

```
VEXTERNC int loadDielMaps (
            NOsh * nosh,
            Vgrid * dielXMap[NOSH_MAXMOL],
            Vgrid * dielYMap[NOSH_MAXMOL],
            Vgrid * dielZMap[NOSH_MAXMOL] )
```

Load the dielectric maps given in NOsh into grid objects.

Author

Nathan Baker

Parameters

nosh	NOsh object with input file information
dielXMap	List of x-shifted dielectric maps
dielYMap	List of y-shifted dielectric maps
dielZMap	List of x-shifted dielectric maps

Returns

1 if successful, 0 otherwise

Loads dielectric map path data into NOsh object

Returns

1 on success, 0 on error

Definition at line 250 of file routines.c.

7.32.2.21 loadKappaMaps()

```
VEXTERNC int loadKappaMaps (
            NOsh * nosh,
            Vgrid * map[NOSH_MAXMOL] )
```

Load the kappa maps given in NOsh into grid objects.

Author

Nathan Baker

Parameters

nosh	NOsh object with input file information
kappa	List of kappa maps

Returns

1 if successful, 0 otherwise

0 on failure, 1 on success

Definition at line 664 of file routines.c.

7.32.2.22 loadMeshes()

```
VEXTERNC Vrc_Codes loadMeshes (
             NOsh * nosh,
             Gem * gm[NOSH_MAXMOL] )
```

Load the meshes given in NOsh into geometry objects.

Author

Nathan Baker

Returns

Error code on success/failure

Parameters

nosh	NOsh object with input file information
gm	List of geometry objects (to be populated)

7.32.2.23 loadMolecules()

```
VEXTERNC int loadMolecules (
             NOsh * nosh,
             Vparam * param,
             Valist * alist[NOSH_MAXMOL] )
```

Load the molecules given in NOsh into atom lists.

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Parameters

nosh	NOsh object with input file information
param	NULL (if PQR files only) or pointer to parameter object
alist	List of atom list objects (to be populated)

Definition at line 95 of file routines.c.

7.32.2.24 loadParameter()

```
VEXTERNC Vparam* loadParameter (
             NOsh * nosh )
```

Loads and returns parameter object.

Author

Nathan Baker

Returns

Pointer to parameter object or NULL

Parameters

nosh	Pointer to NOsh object with input file information

Definition at line 60 of file routines.c.

7.32.2.25 loadPotMaps()

```
VEXTERNC int loadPotMaps (
             NOsh * nosh,
             Vgrid * map[NOSH_MAXMOL] )
```

Load the potential maps given in NOsh into grid objects.

Author

David Gohara

Parameters

nosh	NOsh object with input file information
pot	List of potential maps

Returns

1 if successful, 0 otherwise

0 on failure, 1 on success

Definition at line 793 of file routines.c.

7.32.2.26 main()

```
int main (
            int argc,
            char ** argv )
```

The main APBS function.

Author

Nathan Baker, Dave Gohara, Todd Dolinsky

Returns

Status code (0 for success)

Parameters

argc	Number of arguments
argv	Argument strings

Definition at line 80 of file main.c.

7.32.2.27 partFE()

```
VEXTERNC int partFE (
            int i,
            NOsh * nosh,
            FEMparm * feparm,
            Vfetk * fetk[NOSH_MAXCALC] )
```

Partition mesh (if applicable)

Author

Nathan Baker

Parameters

i	Calculation index
nosh	Master parameter object
feparm	FE-specific parameters
fetk	Array of FE solver objects

Returns

1 if successful, 0 otherwise

Definition at line 4039 of file routines.c.

7.32.2.28 postRefineFE()

```
VEXTERNC int postRefineFE (
            int icalc,
```

```
            FEMparm * feparm,
            Vfetk * fetk[NOSH_MAXCALC] )
```

Estimate error, mark mesh, and refine mesh after solve.

Author

Nathan Baker

Parameters

icalc	Calculation index
feparm	FE-specific parameters
fetk	Array of FE solver objects

Returns

1 if successful, 0 otherwise – note that a 0 will likely imply that either the max number of vertices have been met or no vertices were marked for refinement. In either case, this should not be treated as a fatal error.

Estimates the error, marks the mesh, and refines the mesh after solving.

Returns

1 if successful, 0 otherwise – note that a 0 will likely imply that either the max number of vertices have been met or no vertices were marked for refinement. In either case, this should not be treated as a fatal error.

< Number of vertices in the molecular geometry
< Whether vertices are marked for refinement

Parameters

icalc	Calculation index
feparm	FE-specific parameters
fetk	Array of FE solver objects

Definition at line 4239 of file routines.c.

7.32.2.29 preRefineFE()

```
VEXTERNC int preRefineFE (
            int i,
            FEMparm * feparm,
            Vfetk * fetk[NOSH_MAXCALC] )
```

Pre-refine mesh before solve.

Author

Nathan Baker

Parameters

i	Calculation index
nosh	Master parameter object

Parameters

feparm	FE-specific parameters
fetk	Array of FE solver objects

Returns

1 if successful, 0 otherwise

< Number of vertices in the mesh geometry
< Essentially a boolean; indicates whether further refinement is required after running MC's refinement algorithm.
TODO: could this be optimized by moving nverts out of the loop to just above this initial printout? This depends heavily on whether the number of vertices can change during the calculation. - PCE
Definition at line 4046 of file routines.c.

7.32.2.30 printApolEnergy()

```
VEXTERNC int printApolEnergy (
            NOsh * nosh,
            int iprint )
```

Combine and pretty-print energy data.

Author

David Gohara

Returns

1 if successful, 0 otherwise

Parameters

nosh	Parameters from input file
iprint	Index of energy statement to print

Definition at line 2918 of file routines.c.

7.32.2.31 printApolForce()

```
VEXTERNC int printApolForce (
            Vcom * com,
            NOsh * nosh,
            int nforce[NOSH_MAXCALC],
            AtomForce * atomForce[NOSH_MAXCALC],
            int i )
```

Combine and pretty-print force data.

Author

David Gohara

Returns

1 if successful, 0 otherwise

Parameters

nosh	Communications object
nforce	Parameters from input file
atomForce	Number of forces calculated
i	Array of force structures Index of force statement to print

Definition at line 3471 of file routines.c.

7.32.2.32 printElecEnergy()

```
VEXTERNC int printElecEnergy (
            Vcom * com,
            NOsh * nosh,
            double totEnergy[NOSH_MAXCALC],
            int iprint )
```

Combine and pretty-print energy data.

Author

David Gohara

Returns

1 if successful, 0 otherwise

Parameters

nosh	Communications object
totEnergy	Parameters from input file
iprint	Array of energies from different calculations Index of energy statement to print

Definition at line 2853 of file routines.c.

7.32.2.33 printElecForce()

```
VEXTERNC int printElecForce (
            Vcom * com,
            NOsh * nosh,
            int nforce[NOSH_MAXCALC],
            AtomForce * atomForce[NOSH_MAXCALC],
            int i )
```

Combine and pretty-print force data.

Author

David Gohara

Returns

1 if successful, 0 otherwise

Parameters

nosh	Communications object
nforce	Parameters from input file
atomForce	Number of forces calculated
i	Array of force structures Index of force statement to print

Definition at line 3228 of file routines.c.

7.32.2.34 printEnergy()

```
VEXTERNC int printEnergy (
            Vcom * com,
            NOsh * nosh,
            double totEnergy[NOSH_MAXCALC],
            int iprint )
```

Combine and pretty-print energy data (deprecated...see printElecEnergy)

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Parameters

nosh	Communications object
totEnergy	Parameters from input file
iprint	Array of energies from different calculations Index of energy statement to print

Definition at line 2785 of file routines.c.

7.32.2.35 printFEPARM()

```
VEXTERNC void printFEPARM (
            int icalc,
            NOsh * nosh,
            FEMparm * feparm,
            Vfetk * fetk[NOSH_MAXCALC] )
```

Print out FE-specific params loaded from input.

Author

Nathan Baker

Parameters

icalc	Calculation index
nosh	Master parameter object
feparm	FE-specific parameters
fetk	Array of FE solver objects

Definition at line 3903 of file routines.c.

7.32.2.36 printForce()

```
VEXTERNC int printForce (
            Vcom * com,
            NOsh * nosh,
            int nforce[NOSH_MAXCALC],
            AtomForce * atomForce[NOSH_MAXCALC],
            int i )
```

Combine and pretty-print force data (deprecated...see printElecForce)

Author

Nathan Baker

Returns

1 if successful, 0 otherwise

Parameters

nosh	Communications object
nforce	Parameters from input file
atomForce	Number of forces calculated
i	Array of force structures Index of force statement to print

Definition at line 2980 of file routines.c.

7.32.2.37 printMGPARM()

```
VEXTERNC void printMGPARM (
            MGparm * mgparm,
            double realCenter[3] )
```

Print out MG-specific params loaded from input.

Author

Nathan Baker

Parameters

realCenter	Center of mesh for actual calculation
mgparm	MGparm object

Definition at line 1179 of file routines.c.

7.32.2.38 printPBEPARM()

```
VEXTERNC void printPBEPARM (
            PBEparm * pbeparm )
```

Print out generic PBE params loaded from input.

Author

Nathan Baker

Parameters

pbeparm	PBEparm object

Definition at line 1002 of file routines.c.

7.32.2.39 returnEnergy()

```
VEXTERNC double returnEnergy (
            Vcom * com,
            NOsh * nosh,
            double totEnergy[NOSH_MAXCALC],
            int iprint )
```

Access net local energy.

Author

Justin Xiang

Parameters

com	Communications object
nosh	Parameters from input file
totEnergy	Array of energies from different calculations
iprint	Index of energy statement to print

Returns

Net local energy

Definition at line 2753 of file routines.c.

7.32.2.40 setPartMG()

```
VEXTERNC int setPartMG (
            NOsh * nosh,
            MGparm * mgparm,
            Vpmg * pmg )
```

Set MG partitions for calculating observables and performing I/O.

Author

Nathan Baker

Parameters

nosh	Object with parsed input file parameters
mgparm	MG parameters from input file
pmg	MG object

Returns

1 if successful, 0 otherwise

Definition at line 1523 of file routines.c.

7.32.2.41 solveFE()

```
VEXTERNC int solveFE (
            int icalc,
            PBEparm * pbeparm,
            FEMparm * feparm,
            Vfetk * fetk[NOSH_MAXCALC] )
```

Solve-estimate-refine.

Author

Nathan Baker

Parameters

i	Calculation index
feparm	FE-specific parameters
pbeparm	Generic PBE parameters
fetk	Array of FE solver objects

Returns

1 if successful, 0 otherwise

Call MC's mesh solving equations depending upon the type of PBE we're dealing with. < AM_hPcg
< Coarse-grid solver; 0 = SLU, 1 = MG, 2 = CG, 3 = BCG, 4 = PCG, 5 = PBCG
< Primal problem
< Preconditioner; 0 = identity.

Parameters

icalc	Calculation index
pbeparm	PBE-specific parameters
feparm	FE-specific parameters
fetk	Array of FE solver objects

Definition at line 4116 of file routines.c.

7.32.2.42 solveMG()

```
VEXTERNC int solveMG (
            NOsh * nosh,
            Vpmg * pmg,
            MGparm_CalcType type )
```

Solve the PBE with MG.

Author

Nathan Baker

Parameters

nosh	Object with parsed input file parameters
pmg	MG objects for this calculation
type	Type of MG calculation

Returns

1 if successful, 0 otherwise

Definition at line 1487 of file routines.c.

7.32.2.43 startVio()

```
VEXTERNC void startVio ( )
```

Wrapper to start MALOC Vio layer.

Author

Nathan Baker and Robert Konecny

Definition at line 58 of file routines.c.

7.32.2.44 storeAtomEnergy()

```
VEXTERNC void storeAtomEnergy (
            Vpmg * pmg,
            int icalc,
            double ** atomEnergy,
            int * nenergy )
```

Store energy in arrays for future use.

Author

Todd Dolinsky

Parameters

pmg	MG object

Parameters

icalc	Calculation number
atomEnergy	Pointer to storage array of doubles
nenergy	Stores number of atoms per calc

Definition at line 1870 of file routines.c.

7.32.2.45 writedataFE()

```
VEXTERNC int writedataFE (
            int rank,
            NOsh * nosh,
            PBEparm * pbeparm,
            Vfetk * fetk )
```

Write FEM data to files.

Author

Nathan Baker

Parameters

rank	Rank of processor (for parallel runs)
nosh	NOsh object
pbeparm	PBEparm object
fetk	FEtk object (with solution)

Returns

1 if successful, 0 otherwise

Write FEM data to file. <
<
< Loop counter
< Flag indicating whether data can be written to output
<
<

Parameters

rank	Rank of processor (for parallel runs)
nosh	NOsh object
pbeparm	PBE-specific parameters
fetk	FEtk object (with solution)

Definition at line 4300 of file routines.c.

7.32.2.46 writedataFlat()

```
VEXTERNC int writedataFlat (
```

```
            NOsh * nosh,
            Vcom * com,
            const char * fname,
            double totEnergy[NOSH_MAXCALC],
            double qfEnergy[NOSH_MAXCALC],
            double qmEnergy[NOSH_MAXCALC],
            double dielEnergy[NOSH_MAXCALC],
            int nenergy[NOSH_MAXCALC],
            double * atomEnergy[NOSH_MAXCALC],
            int nforce[NOSH_MAXCALC],
            AtomForce * atomForce[NOSH_MAXCALC] )
```

Write out information to a flat file.

Author

Todd Dolinsky

Parameters

nosh	Parameters from input file
com	The communications object
fname	The target XML file name
totEnergy	An array with per-calc total energies (in kT)
qfEnergy	An array with per-calc charge-potential energies (in kT)
qmEnergy	An array with per-calc mobile energies (in kT)
dielEnergy	An array with per-calc polarization energies (in kT)
nenergy	An array containing the number of atoms per-calc
atomEnergy	An array containing per-atom energies (in KT) per calc
nforce	An array containing the number of forces calculated per-calc
atomForce	An array containing per-atom forces per calc

Returns

1 if successful, 0 otherwise

Definition at line 1887 of file routines.c.

7.32.2.47 writedataMG()

```
VEXTERNC int writedataMG (
            int rank,
            NOsh * nosh,
            PBEparm * pbeparm,
            Vpmg * pmg )
```

Write out observables from MG calculation to file.

Author

Nathan Baker

Parameters

rank	Processor rank (if parallel calculation)
nosh	Parameters from input file
pbeparm	Generic PBE parameters
pmg	MG object

Returns

1 if successful, 0 otherwise

Definition at line 2383 of file routines.c.

7.32.2.48 writedataXML()

```
VEXTERNC int writedataXML (
            NOsh * nosh,
            Vcom * com,
            const char * fname,
            double totEnergy[NOSH_MAXCALC],
            double qfEnergy[NOSH_MAXCALC],
            double qmEnergy[NOSH_MAXCALC],
            double dielEnergy[NOSH_MAXCALC],
            int nenergy[NOSH_MAXCALC],
            double * atomEnergy[NOSH_MAXCALC],
            int nforce[NOSH_MAXCALC],
            AtomForce * atomForce[NOSH_MAXCALC] )
```

Write out information to an XML file.

Author

Todd Dolinsky

Parameters

nosh	Parameters from input file
com	The communications object
fname	The target XML file name
totEnergy	An array with per-calc total energies (in kT)
qfEnergy	An array with per-calc charge-potential energies (in kT)
qmEnergy	An array with per-calc mobile energies (in kT)
dielEnergy	An array with per-calc polarization energies (in kT)
nenergy	An array containing the number of atoms per-calc
atomEnergy	An array containing per-atom energies (in KT) per calc
nforce	An array containing the number of forces calculated per-calc
atomForce	An array containing per-atom forces per calc

Returns

1 if successful, 0 otherwise

Definition at line 2123 of file routines.c.

7.32.2.49 writematMG()

```
VEXTERNC int writematMG (
            int rank,
            NOsh * nosh,
            PBEparm * pbeparm,
            Vpmg * pmg )
```

Write out operator matrix from MG calculation to file.

Author

Nathan Baker

Parameters

rank	Processor rank (if parallel calculation)
nosh	Parameters from input file
pbeparm	Generic PBE parameters
pmg	MG object

Returns

1 if successful, 0 otherwise

Definition at line 1799 of file routines.c.

Chapter 8

Data Structure Documentation

8.1 AtomForce Struct Reference

Structure to hold atomic forces.
`#include </builddir/build/BUILD/apbs-3.0.0/src/routines.h>`

Data Fields

- double ibForce [3]
- double qfForce [3]
- double dbForce [3]
- double sasaForce [3]
- double savForce [3]
- double wcaForce [3]

8.1.1 Detailed Description

Structure to hold atomic forces.

Author

Nathan Baker

Definition at line 99 of file routines.h.

8.1.2 Field Documentation

8.1.2.1 dbForce

`double dbForce[3]`
Dielectric boundary force
Definition at line 102 of file routines.h.

8.1.2.2 ibForce

`double ibForce[3]`
Ion-boundary force
Definition at line 100 of file routines.h.

8.1.2.3 qfForce

`double qfForce[3]`
Charge-field force
Definition at line 101 of file routines.h.

8.1.2.4 sasaForce

`double sasaForce[3]`
SASA force (coupled to gamma)
Definition at line 103 of file routines.h.

8.1.2.5 savForce

`double savForce[3]`
SAV force (coupled to press)
Definition at line 104 of file routines.h.

8.1.2.6 wcaForce

`double wcaForce[3]`
WCA integral force (coupled to bconc)
Definition at line 105 of file routines.h.
The documentation for this struct was generated from the following file:

- src/routines.h

8.2 sAPOLparm Struct Reference

Parameter structure for APOL-specific variables from input files.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/apolparm.h>`

Data Fields

- int parsed
- double grid [3]
- int setgrid
- int molid
- int setmolid
- double bconc
- int setbconc
- double sdens
- int setsdens
- double dpos
- int setdpos
- double press
- int setpress
- Vsurf_Meth srfm
- int setsrfm
- double srad

- int setsrad
- double swin
- int setswin
- double temp
- int settemp
- double gamma
- int setgamma
- APOLparm_calcEnergy calcenergy
- int setcalcenergy
- APOLparm_calcForce calcforce
- int setcalcforce
- double watsigma
- double watepsilon
- double sasa
- double sav
- double wcaEnergy
- double totForce [3]
- int setwat

8.2.1 Detailed Description

Parameter structure for APOL-specific variables from input files.

Author

David Gohara

Definition at line 129 of file apolparm.h.

8.2.2 Field Documentation

8.2.2.1 bconc

`double bconc`
Vacc sphere density
Definition at line 139 of file apolparm.h.

8.2.2.2 calcenergy

`APOLparm_calcEnergy calcenergy`
Energy calculation flag
Definition at line 167 of file apolparm.h.

8.2.2.3 calcforce

`APOLparm_calcForce calcforce`
Atomic forces calculation
Definition at line 170 of file apolparm.h.

8.2.2.4 dpos

`double dpos`
Atom position offset
Definition at line 145 of file apolparm.h.

8.2.2.5 gamma

`double gamma`
Surface tension for apolar energies/forces (in kJ/mol/A^2)
Definition at line 163 of file apolparm.h.

8.2.2.6 grid

`double grid[3]`
Grid spacing
Definition at line 133 of file apolparm.h.

8.2.2.7 molid

`int molid`
Molecule ID to perform calculation on
Definition at line 136 of file apolparm.h.

8.2.2.8 parsed

`int parsed`
Flag: Has this structure been filled with anything other than the default values? (0 = no, 1 = yes)
Definition at line 131 of file apolparm.h.

8.2.2.9 press

`double press`
Solvent pressure
Definition at line 148 of file apolparm.h.

8.2.2.10 sasa

`double sasa`
Solvent accessible surface area for this calculation
Definition at line 175 of file apolparm.h.

8.2.2.11 sav

`double sav`
Solvent accessible volume for this calculation
Definition at line 176 of file apolparm.h.

8.2.2.12 sdens

```
double sdens
```

Vacc sphere density
Definition at line 142 of file apolparm.h.

8.2.2.13 setbconc

```
int setbconc
```

Flag,

See also

bconc

Definition at line 140 of file apolparm.h.

8.2.2.14 setcalcenergy

```
int setcalcenergy
```

Flag,

See also

calcenergy

Definition at line 168 of file apolparm.h.

8.2.2.15 setcalcforce

```
int setcalcforce
```

Flag,

See also

calcforce

Definition at line 171 of file apolparm.h.

8.2.2.16 setdpos

```
int setdpos
```

Flag,

See also

dpos

Definition at line 146 of file apolparm.h.

8.2.2.17 setgamma

```
int setgamma
```

Flag,

See also

gamma

Definition at line 165 of file apolparm.h.

8.2.2.18 setgrid

`int setgrid`
Flag,

See also

grid

Definition at line 134 of file apolparm.h.

8.2.2.19 setmolid

`int setmolid`
Flag,

See also

molid

Definition at line 137 of file apolparm.h.

8.2.2.20 setpress

`int setpress`
Flag,

See also

press

Definition at line 149 of file apolparm.h.

8.2.2.21 setsdens

`int setsdens`
Flag,

See also

sdens

Definition at line 143 of file apolparm.h.

8.2.2.22 setsrad

`int setsrad`
Flag,

See also

srad

Definition at line 155 of file apolparm.h.

8.2.2.23 setsrfm

```
int setsrfm
```

Flag,

See also

srfm

Definition at line 152 of file apolparm.h.

8.2.2.24 setswin

```
int setswin
```

Flag,

See also

swin

Definition at line 158 of file apolparm.h.

8.2.2.25 settemp

```
int settemp
```

Flag,

See also

temp

Definition at line 161 of file apolparm.h.

8.2.2.26 setwat

```
int setwat
```

Boolean for determining if a water parameter is supplied. Yes = 1, No = 0
Definition at line 180 of file apolparm.h.

8.2.2.27 srad

```
double srad
```

Solvent radius
Definition at line 154 of file apolparm.h.

8.2.2.28 srfm

```
Vsurf_Meth srfm
```

Surface calculation method
Definition at line 151 of file apolparm.h.

8.2.2.29 swin

`double swin`
Cubic spline window
Definition at line 157 of file apolparm.h.

8.2.2.30 temp

`double temp`
Temperature (in K)
Definition at line 160 of file apolparm.h.

8.2.2.31 totForce

`double totForce[3]`
Total forces on x, y, z
Definition at line 178 of file apolparm.h.

8.2.2.32 watepsilon

`double watepsilon`
Water oxygen Lennard-Jones well depth (kJ/mol)
Definition at line 174 of file apolparm.h.

8.2.2.33 watsigma

`double watsigma`
Water oxygen Lennard-Jones radius (A)
Definition at line 173 of file apolparm.h.

8.2.2.34 wcaEnergy

`double wcaEnergy`
wcaEnergy
Definition at line 177 of file apolparm.h.
The documentation for this struct was generated from the following file:

- src/generic/apolparm.h

8.3 sBEMparm Struct Reference

Parameter structure for BEM-specific variables from input files.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/bemparm.h>`

Data Fields

- BEMparm_CalcType type
- int parsed
- Vchrg_Src chgs
- int tree_order

- int settree_order
- int tree_n0
- int settree_n0
- double mac
- int setmac
- int nonlintype
- int setnonlintype
- int mesh
- int setmesh
- int outdata
- int setoutdata

8.3.1 Detailed Description

Parameter structure for BEM-specific variables from input files.

Author

Nathan Baker and Todd Dolinsky and Weihua Geng

Note

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

Definition at line 96 of file bemparm.h.

8.3.2 Field Documentation

8.3.2.1 chgs

`Vchrg_Src chgs`
Charge source (Charge, Multipole, Induced Dipole, NL Induced. Not currently implemented but should be relatively easy to add in the future (cf Pengyu Ren)
Definition at line 102 of file bemparm.h.

8.3.2.2 mac

`double mac`
Multipole acceptance criterion (should be between 0 and 1)
Definition at line 108 of file bemparm.h.

8.3.2.3 mesh

`int mesh`
0 for msms, 1 for NanoShaper SES, 2 for NanoShaper Skin
Definition at line 113 of file bemparm.h.

8.3.2.4 nonlintype

`int nonlintype`
Linearity Type Method to be used
Definition at line 110 of file bemparm.h.

8.3.2.5 outdata

`int outdata`
0 does not output vtk, 1 outputs vtk
Definition at line 116 of file bemparm.h.

8.3.2.6 parsed

`int parsed`
Has this structure been filled? (0 = no, 1 = yes)
Definition at line 99 of file bemparm.h.

8.3.2.7 setmac

`int setmac`
Flag,

See also

mac

Definition at line 109 of file bemparm.h.

8.3.2.8 setmesh

`int setmesh`
Flag,

See also

mesh

Definition at line 114 of file bemparm.h.

8.3.2.9 setnonlintype

`int setnonlintype`
Flag,

See also

nonlintype

Definition at line 111 of file bemparm.h.

8.3.2.10 setoutdata

`int setoutdata`
Flag,

See also

outdata

Definition at line 117 of file bemparm.h.

8.3.2.11 settree_n0

`int settree_n0`
Flag,

See also

tree_npart

Definition at line 107 of file bemparm.h.

8.3.2.12 settree_order

`int settree_order`
Flag,

See also

tree_order

Definition at line 105 of file bemparm.h.

8.3.2.13 tree_n0

`int tree_n0`
Number of particles per leaf of the tree
Definition at line 106 of file bemparm.h.

8.3.2.14 tree_order

`int tree_order`
User-defined order for the treecode expansion
Definition at line 104 of file bemparm.h.

8.3.2.15 type

`BEMparm_CalcType type`
What type of BEM calculation?
Definition at line 98 of file bemparm.h.
The documentation for this struct was generated from the following file:

- src/generic/bemparm.h

8.4 sFEMparm Struct Reference

Parameter structure for FEM-specific variables from input files.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/femparm.h>`

Data Fields

- int parsed
- FEMparm_CalcType type
- int settype
- double glen [3]
- int setglen
- double etol
- int setetol
- FEMparm_EtolType ekey
- int setekey
- FEMparm_EstType akeyPRE
- int setakeyPRE
- FEMparm_EstType akeySOLVE
- int setakeySOLVE
- int targetNum
- int settargetNum
- double targetRes
- int settargetRes
- int maxsolve
- int setmaxsolve
- int maxvert
- int setmaxvert
- int pkey
- int useMesh
- int meshID

8.4.1 Detailed Description

Parameter structure for FEM-specific variables from input files.

Author

Nathan Baker

Definition at line 133 of file femparm.h.

8.4.2 Field Documentation

8.4.2.1 akeyPRE

`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 148 of file femparm.h.

8.4.2.2 akeySOLVE

`FEMparm_EstType akeySOLVE`
Adaptive refinment error estimator method for a posteriori solution-based refinement.
Definition at line 152 of file femparm.h.

8.4.2.3 ekey

`FEMparm_EtolType ekey`
Adaptive refinment interpretation of error tolerance
Definition at line 145 of file femparm.h.

8.4.2.4 etol

`double etol`
Error tolerance for refinement; interpretation depends on the adaptive refinement method chosen
Definition at line 142 of file femparm.h.

8.4.2.5 glen

`double glen[3]`
Domain side lengths (in Å)
Definition at line 140 of file femparm.h.

8.4.2.6 maxsolve

`int maxsolve`
Maximum number of solve-estimate-refine cycles
Definition at line 165 of file femparm.h.

8.4.2.7 maxvert

`int maxvert`
Maximum number of vertices in mesh (ignored if less than zero)
Definition at line 167 of file femparm.h.

8.4.2.8 meshID

`int meshID`
External finite element mesh ID (if used)
Definition at line 174 of file femparm.h.

8.4.2.9 parsed

`int parsed`
Flag: Has this structure been filled with anything other than $*$ the default values? (0 = no, 1 = yes)
Definition at line 135 of file femparm.h.

8.4.2.10 pkey

`int pkey`
Boolean sets the pkey type for going into AM_Refine pkey = 0 for non-HB based methods pkey = 1 for HB based methods
Definition at line 170 of file femparm.h.

8.4.2.11 setakeyPRE

`int setakeyPRE`
Boolean
Definition at line 151 of file femparm.h.

8.4.2.12 setakeySOLVE

`int setakeySOLVE`
Boolean
Definition at line 154 of file femparm.h.

8.4.2.13 setekey

`int setekey`
Boolean
Definition at line 147 of file femparm.h.

8.4.2.14 setetol

`int setetol`
Boolean
Definition at line 144 of file femparm.h.

8.4.2.15 setglen

`int setglen`
Boolean
Definition at line 141 of file femparm.h.

8.4.2.16 setmaxsolve

`int setmaxsolve`
Boolean
Definition at line 166 of file femparm.h.

8.4.2.17 setmaxvert

`int setmaxvert`
Boolean
Definition at line 169 of file femparm.h.

8.4.2.18 settargetNum

```
int settargetNum
```
Boolean
Definition at line 159 of file femparm.h.

8.4.2.19 settargetRes

```
int settargetRes
```
Boolean
Definition at line 164 of file femparm.h.

8.4.2.20 settype

```
int settype
```
Boolean
Definition at line 139 of file femparm.h.

8.4.2.21 targetNum

```
int targetNum
```
Initial mesh will continue to be marked and refined with the method specified by akeyPRE until the mesh contains this many vertices or until targetRes is reached.
Definition at line 155 of file femparm.h.

8.4.2.22 targetRes

```
double targetRes
```
Initial mesh will continue to be marked and refined with the method specified by akeyPRE until the mesh contains no markable simplices with longest edges above this size or until targetNum is reached.
Definition at line 160 of file femparm.h.

8.4.2.23 type

```
FEMparm_CalcType type
```
Calculation type
Definition at line 138 of file femparm.h.

8.4.2.24 useMesh

```
int useMesh
```
Indicates whether we use external finite element mesh
Definition at line 173 of file femparm.h.
The documentation for this struct was generated from the following file:

- src/generic/femparm.h

8.5 sGEOFLOWparm Struct Reference

Parameter structure for GEOFLOW-specific variables from input files.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/geoflowparm.h>`

Data Fields

- GEOFLOWparm_CalcType type
- int parsed
- int **vdw**
- int **setvdw**
- double etol

8.5.1 Detailed Description

Parameter structure for GEOFLOW-specific variables from input files.

Author

Andrew Stevens, Kyle Monson

Note

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

Definition at line 98 of file geoflowparm.h.

8.5.2 Field Documentation

8.5.2.1 etol

`double etol`
user defined error tolerance
Definition at line 106 of file geoflowparm.h.

8.5.2.2 parsed

`int parsed`
Has this structure been filled? (0 = no, 1 = yes)
Definition at line 101 of file geoflowparm.h.

8.5.2.3 type

`GEOFLOWparm_CalcType type`
What type of GEOFLOW calculation?
Definition at line 100 of file geoflowparm.h.
The documentation for this struct was generated from the following file:

- src/generic/geoflowparm.h

8.6 sMGparm Struct Reference

Parameter structure for MG-specific variables from input files.

`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/mgparm.h>`

Data Fields

- MGparm_CalcType type
- int parsed
- int dime [3]
- int setdime
- Vchrg_Meth chgm
- int setchgm
- Vchrg_Src chgs
- int nlev
- int setnlev
- double etol
- int setetol
- double grid [3]
- int setgrid
- double glen [3]
- int setglen
- MGparm_CentMeth cmeth
- double center [3]
- int centmol
- int setgcent
- double cglen [3]
- int setcglen
- double fglen [3]
- int setfglen
- MGparm_CentMeth ccmeth
- double ccenter [3]
- int ccentmol
- int 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

8.6.1 Detailed Description

Parameter structure for MG-specific variables from input files.

Author

Nathan Baker and Todd Dolinsky

Note

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

Definition at line 114 of file mgparm.h.

8.6.2 Field Documentation

8.6.2.1 async

`int async`
Processor ID for asynchronous calculation
Definition at line 186 of file mgparm.h.

8.6.2.2 ccenter

`double ccenter[3]`
Coarse grid center.

Definition at line 157 of file mgparm.h.

8.6.2.3 ccentmol

`int ccentmol`
Particular molecule on which we want to center the grid. This should be the appropriate index in an array of molecules, not the positive definite integer specified by the user.
Definition at line 158 of file mgparm.h.

8.6.2.4 ccmeth

`MGparm_CentMeth ccmeth`
Coarse grid centering method
Definition at line 156 of file mgparm.h.

8.6.2.5 center

`double center[3]`
Grid center. If ispart = 0, then this is only meaningful if cmeth = 0. However, if ispart = 1 and cmeth = MCM_PNT, then this is the center of the non-disjoint (overlapping) partition. If ispart = 1 and cmeth = MCM_MOL, then this is the vector that must be added to the center of the molecule to give the center of the non-disjoint partition.

Definition at line 138 of file mgparm.h.

8.6.2.6 centmol

`int centmol`
Particular molecule on which we want to center the grid. This should be the appropriate index in an array of molecules, not the positive definite integer specified by the user.
Definition at line 146 of file mgparm.h.

8.6.2.7 cglen

`double cglen[3]`
Coarse grid side lengths
Definition at line 152 of file mgparm.h.

8.6.2.8 chgm

`Vchrg_Meth chgm`
Charge discretization method
Definition at line 122 of file mgparm.h.

8.6.2.9 chgs

`Vchrg_Src chgs`
Charge source (Charge, Multipole, Induced Dipole, NL Induced
Definition at line 124 of file mgparm.h.

8.6.2.10 cmeth

`MGparm_CentMeth cmeth`
Centering method
Definition at line 137 of file mgparm.h.

8.6.2.11 dime

`int dime[3]`
Grid dimensions
Definition at line 120 of file mgparm.h.

8.6.2.12 etol

`double etol`
User-defined error tolerance
Definition at line 131 of file mgparm.h.

8.6.2.13 fcenter

`double fcenter[3]`
Fine grid center.

Definition at line 163 of file mgparm.h.

8.6.2.14 fcentmol

`int fcentmol`
Particular molecule on which we want to center the grid. This should be the appropriate index in an array of molecules, not the positive definite integer specified by the user.
Definition at line 164 of file mgparm.h.

8.6.2.15 fcmeth

`MGparm_CentMeth fcmeth`
Fine grid centering method
Definition at line 162 of file mgparm.h.

8.6.2.16 fglen

`double fglen[3]`
Fine grid side lengths
Definition at line 154 of file mgparm.h.

8.6.2.17 glen

`double glen[3]`
Grid side lengths.
Definition at line 135 of file mgparm.h.

8.6.2.18 grid

`double grid[3]`
Grid spacings
Definition at line 133 of file mgparm.h.

8.6.2.19 method

`int method`
Solver Method
Definition at line 192 of file mgparm.h.

8.6.2.20 nlev

`int nlev`
Levels in multigrid hierarchy

Deprecated Just ignored now

Definition at line 128 of file mgparm.h.

8.6.2.21 nonlintype

`int nonlintype`
Linearity Type Method to be used
Definition at line 189 of file mgparm.h.

8.6.2.22 ofrac

`double ofrac`
Overlap fraction between procs
Definition at line 184 of file mgparm.h.

8.6.2.23 parsed

`int parsed`
Has this structure been filled? (0 = no, 1 = yes)
Definition at line 117 of file mgparm.h.

8.6.2.24 partDisjCenter

`double partDisjCenter[3]`
This gives the center of the disjoint partitions
Definition at line 171 of file mgparm.h.

8.6.2.25 partDisjLength

`double partDisjLength[3]`
This gives the lengths of the disjoint partitions
Definition at line 173 of file mgparm.h.

8.6.2.26 partDisjOwnSide

`int partDisjOwnSide[6]`
Tells whether the boundary points are ours (1) or not (0)
Definition at line 175 of file mgparm.h.

8.6.2.27 pdime

`int pdime[3]`
Grid of processors to be used in calculation
Definition at line 178 of file mgparm.h.

8.6.2.28 proc_rank

`int proc_rank`
Rank of this processor
Definition at line 180 of file mgparm.h.

8.6.2.29 proc_size

`int proc_size`
Total number of processors
Definition at line 182 of file mgparm.h.

8.6.2.30 setasync

`int setasync`
Flag,

See also

asynch

Definition at line 187 of file mgparm.h.

8.6.2.31 setcgcent

`int setcgcent`
Flag,

See also

ccmeth

Definition at line 161 of file mgparm.h.

8.6.2.32 setcglen

`int setcglen`
Flag,

See also

cglen

Definition at line 153 of file mgparm.h.

8.6.2.33 setchgm

`int setchgm`
Flag,

See also

chgm

Definition at line 123 of file mgparm.h.

8.6.2.34 setdime

`int setdime`
Flag,

See also

dime

Definition at line 121 of file mgparm.h.

8.6.2.35 setetol

`int setetol`
Flag,

See also

etol

Definition at line 132 of file mgparm.h.

8.6.2.36 setfgcent

`int setfgcent`
Flag,

See also

fcmeth

Definition at line 167 of file mgparm.h.

8.6.2.37 setfglen

`int setfglen`
Flag,

See also

fglen

Definition at line 155 of file mgparm.h.

8.6.2.38 setgcent

`int setgcent`
Flag,

See also

cmeth

Definition at line 149 of file mgparm.h.

8.6.2.39 setglen

`int setglen`
Flag,

See also

glen

Definition at line 136 of file mgparm.h.

8.6.2.40 setgrid

`int setgrid`
Flag,

See also

grid

Definition at line 134 of file mgparm.h.

8.6.2.41 setmethod

`int setmethod`
Flag,

See also

method

Definition at line 193 of file mgparm.h.

8.6.2.42 setnlev

`int setnlev`
Flag,

See also

nlev

Definition at line 130 of file mgparm.h.

8.6.2.43 setnonlintype

`int setnonlintype`
Flag,

See also

nonlintype

Definition at line 190 of file mgparm.h.

8.6.2.44 setofrac

`int setofrac`
Flag,

See also

ofrac

Definition at line 185 of file mgparm.h.

8.6.2.45 setpdime

`int setpdime`
Flag,

See also

pdime

Definition at line 179 of file mgparm.h.

8.6.2.46 setrank

`int setrank`
Flag,

See also

proc_rank

Definition at line 181 of file mgparm.h.

8.6.2.47 setsize

`int setsize`
Flag,

See also

proc_size

Definition at line 183 of file mgparm.h.

8.6.2.48 setUseAqua

`int setUseAqua`
Flag,

See also

useAqua

Definition at line 196 of file mgparm.h.

8.6.2.49 type

`MGparm_CalcType type`
What type of MG calculation?
Definition at line 116 of file mgparm.h.

8.6.2.50 useAqua

`int useAqua`
Enable use of lpbe/aqua
Definition at line 195 of file mgparm.h.
The documentation for this struct was generated from the following file:

- src/generic/mgparm.h

8.7 sNOsh Struct Reference

Class for parsing fixed format input files.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/nosh.h>`

Data Fields

- NOsh_calc * calc [NOSH_MAXCALC]
- int ncalc
- NOsh_calc * elec [NOSH_MAXCALC]
- int nelec
- NOsh_calc * apol [NOSH_MAXCALC]
- int napol
- int ispara
- int proc_rank
- int proc_size
- int bogus
- int elec2calc [NOSH_MAXCALC]
- int apol2calc [NOSH_MAXCALC]
- int nmol
- char molpath [NOSH_MAXMOL][VMAX_ARGLEN]
- NOsh_MolFormat molfmt [NOSH_MAXMOL]
- Valist * alist [NOSH_MAXMOL]
- int gotparm
- char parmpath [VMAX_ARGLEN]
- NOsh_ParmFormat parmfmt

- int ndiel
- char dielXpath [NOSH_MAXMOL][VMAX_ARGLEN]
- char dielYpath [NOSH_MAXMOL][VMAX_ARGLEN]
- char dielZpath [NOSH_MAXMOL][VMAX_ARGLEN]
- Vdata_Format dielfmt [NOSH_MAXMOL]
- int nkappa
- char kappapath [NOSH_MAXMOL][VMAX_ARGLEN]
- Vdata_Format kappafmt [NOSH_MAXMOL]
- int npot
- char potpath [NOSH_MAXMOL][VMAX_ARGLEN]
- Vdata_Format potfmt [NOSH_MAXMOL]
- int ncharge
- char chargepath [NOSH_MAXMOL][VMAX_ARGLEN]
- Vdata_Format chargefmt [NOSH_MAXMOL]
- int nmesh
- char meshpath [NOSH_MAXMOL][VMAX_ARGLEN]
- Vdata_Format meshfmt [NOSH_MAXMOL]
- int nprint
- NOsh_PrintType printwhat [NOSH_MAXPRINT]
- int printnarg [NOSH_MAXPRINT]
- int printcalc [NOSH_MAXPRINT][NOSH_MAXPOP]
- int printop [NOSH_MAXPRINT][NOSH_MAXPOP]
- int parsed
- char elecname [NOSH_MAXCALC][VMAX_ARGLEN]
- char apolname [NOSH_MAXCALC][VMAX_ARGLEN]

8.7.1 Detailed Description

Class for parsing fixed format input files.

Author

Nathan Baker

Definition at line 195 of file nosh.h.

8.7.2 Field Documentation

8.7.2.1 alist

```
Valist* alist[NOSH_MAXMOL]
```

Molecules for calculation (can be used in setting mesh centers
Definition at line 234 of file nosh.h.

8.7.2.2 apol

```
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 208 of file nosh.h.

8.7.2.3 apol2calc

```
int apol2calc[NOSH_MAXCALC]
```
(see elec2calc)
Definition at line 229 of file nosh.h.

8.7.2.4 apolname

```
char apolname[NOSH_MAXCALC][VMAX_ARGLEN]
```
Optional user-specified name for APOLAR statement
Definition at line 269 of file nosh.h.

8.7.2.5 bogus

```
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 217 of file nosh.h.

8.7.2.6 calc

```
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 197 of file nosh.h.

8.7.2.7 chargefmt

```
Vdata_Format chargefmt[NOSH_MAXMOL]
```
Charge maps fileformats
Definition at line 255 of file nosh.h.

8.7.2.8 chargepath

```
char chargepath[NOSH_MAXMOL][VMAX_ARGLEN]
```
Paths to charge map files
Definition at line 254 of file nosh.h.

8.7.2.9 dielfmt

```
Vdata_Format dielfmt[NOSH_MAXMOL]
```
Dielectric maps file formats
Definition at line 246 of file nosh.h.

8.7.2.10 dielXpath

```
char dielXpath[NOSH_MAXMOL][VMAX_ARGLEN]
```
Paths to x-shifted dielectric map files
Definition at line 240 of file nosh.h.

8.7.2.11 dielYpath

`char dielYpath[NOSH_MAXMOL][VMAX_ARGLEN]`
Paths to y-shifted dielectric map files
Definition at line 242 of file nosh.h.

8.7.2.12 dielZpath

`char dielZpath[NOSH_MAXMOL][VMAX_ARGLEN]`
Paths to z-shifted dielectric map files
Definition at line 244 of file nosh.h.

8.7.2.13 elec

`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 202 of file nosh.h.

8.7.2.14 elec2calc

`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 221 of file nosh.h.

8.7.2.15 elecname

`char elecname[NOSH_MAXCALC][VMAX_ARGLEN]`
Optional user-specified name for ELEC statement
Definition at line 267 of file nosh.h.

8.7.2.16 gotparm

`int gotparm`
Either have (1) or don't have (0) parm
Definition at line 236 of file nosh.h.

8.7.2.17 ispara

`int ispara`
1 => is a parallel calculation, 0 => is not
Definition at line 214 of file nosh.h.

8.7.2.18 kappafmt

`Vdata_Format kappafmt[NOSH_MAXMOL]`
Kappa maps file formats

Definition at line 249 of file nosh.h.

8.7.2.19 kappapath

```
char kappapath[NOSH_MAXMOL][VMAX_ARGLEN]
```

Paths to kappa map files
Definition at line 248 of file nosh.h.

8.7.2.20 meshfmt

```
Vdata_Format meshfmt[NOSH_MAXMOL]
```

Mesh fileformats
Definition at line 258 of file nosh.h.

8.7.2.21 meshpath

```
char meshpath[NOSH_MAXMOL][VMAX_ARGLEN]
```

Paths to mesh files
Definition at line 257 of file nosh.h.

8.7.2.22 molfmt

```
NOsh_MolFormat molfmt[NOSH_MAXMOL]
```

Mol files formats
Definition at line 233 of file nosh.h.

8.7.2.23 molpath

```
char molpath[NOSH_MAXMOL][VMAX_ARGLEN]
```

Paths to mol files
Definition at line 232 of file nosh.h.

8.7.2.24 napol

```
int napol
```

The number of apolar statements in the input file and in the apolar array
Definition at line 211 of file nosh.h.

8.7.2.25 ncalc

```
int ncalc
```

The number of calculations in the calc array
Definition at line 200 of file nosh.h.

8.7.2.26 ncharge

`int ncharge`
Number of charge maps
Definition at line 253 of file nosh.h.

8.7.2.27 ndiel

`int ndiel`
Number of dielectric maps
Definition at line 239 of file nosh.h.

8.7.2.28 nelec

`int nelec`
The number of elec statements in the input file and in the elec array
Definition at line 205 of file nosh.h.

8.7.2.29 nkappa

`int nkappa`
Number of kappa maps
Definition at line 247 of file nosh.h.

8.7.2.30 nmesh

`int nmesh`
Number of meshes
Definition at line 256 of file nosh.h.

8.7.2.31 nmol

`int nmol`
Number of molecules
Definition at line 231 of file nosh.h.

8.7.2.32 npot

`int npot`
Number of potential maps
Definition at line 250 of file nosh.h.

8.7.2.33 nprint

`int nprint`
How many print sections?
Definition at line 259 of file nosh.h.

8.7.2.34 parmfmt

`NOsh_ParmFormat parmfmt`
Parm file format
Definition at line 238 of file nosh.h.

8.7.2.35 parmpath

`char parmpath[VMAX_ARGLEN]`
Paths to parm file
Definition at line 237 of file nosh.h.

8.7.2.36 parsed

`int parsed`
Have we parsed an input file yet?
Definition at line 266 of file nosh.h.

8.7.2.37 potfmt

`Vdata_Format potfmt[NOSH_MAXMOL]`
Potential maps file formats
Definition at line 252 of file nosh.h.

8.7.2.38 potpath

`char potpath[NOSH_MAXMOL][VMAX_ARGLEN]`
Paths to potential map files
Definition at line 251 of file nosh.h.

8.7.2.39 printcalc

`int printcalc[NOSH_MAXPRINT][NOSH_MAXPOP]`
ELEC id (see elec2calc)
Definition at line 263 of file nosh.h.

8.7.2.40 printnarg

`int printnarg[NOSH_MAXPRINT]`
How many arguments in energy list
Definition at line 262 of file nosh.h.

8.7.2.41 printop

`int printop[NOSH_MAXPRINT][NOSH_MAXPOP]`
Operation id (0 = add, 1 = subtract)
Definition at line 264 of file nosh.h.

8.7.2.42 printwhat

`NOsh_PrintType printwhat[NOSH_MAXPRINT]`
What do we print:

- 0 = energy,
- 1 = force

Definition at line 260 of file nosh.h.

8.7.2.43 proc_rank

`int proc_rank`
Processor rank in parallel calculation
Definition at line 215 of file nosh.h.

8.7.2.44 proc_size

`int proc_size`
Number of processors in parallel calculation
Definition at line 216 of file nosh.h.
The documentation for this struct was generated from the following file:

- src/generic/nosh.h

8.8 sNOsh_calc Struct Reference

Calculation class for use when parsing fixed format input files.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/nosh.h>`

Data Fields

- MGparm * mgparm
- FEMparm * femparm
- BEMparm * bemparm
- GEOFLOWparm * geoflowparm
- PBAMparm * pbamparm
- PBSAMparm * pbsamparm
- PBEparm * pbeparm
- APOLparm * apolparm
- NOsh_CalcType calctype

8.8.1 Detailed Description

Calculation class for use when parsing fixed format input files.

Author

Nathan Baker

Definition at line 172 of file nosh.h.

8.8.2 Field Documentation

8.8.2.1 apolparm

`APOLparm* apolparm`
Non-polar parameters
Definition at line 180 of file nosh.h.

8.8.2.2 bemparm

`BEMparm* bemparm`
boundary element (tabi) parameters
Definition at line 175 of file nosh.h.

8.8.2.3 calctype

`NOsh_CalcType calctype`
Calculation type
Definition at line 181 of file nosh.h.

8.8.2.4 femparm

`FEMparm* femparm`
Finite element parameters
Definition at line 174 of file nosh.h.

8.8.2.5 geoflowparm

`GEOFLOWparm* geoflowparm`
Geometric Flow Solver
Definition at line 176 of file nosh.h.

8.8.2.6 mgparm

`MGparm* mgparm`
Multigrid parameters
Definition at line 173 of file nosh.h.

8.8.2.7 pbamparm

`PBAMparm* pbamparm`
Analytical Poisson-Boltzmann Solver
Definition at line 177 of file nosh.h.

8.8.2.8 pbeparm

PBEparm* pbeparm
Generic PBE parameters
Definition at line 179 of file nosh.h.

8.8.2.9 pbsamparm

PBSAMparm* pbsamparm
Semi-Analytical Poisson-Boltzmann Solver
Definition at line 178 of file nosh.h.
The documentation for this struct was generated from the following file:

- src/generic/nosh.h

8.9 sPBAMparm Struct Reference

Parameter structure for PBAM-specific variables from input files.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/pbamparm.h>`

Data Fields

- PBAMparm_CalcType type
- int parsed
- double **salt**
- int **setsalt**
- char **runtype** [CHR_MAXLEN]
- int **setruntype**
- char **runname** [CHR_MAXLEN]
- int **setrunname**
- int **setrandorient**
- double **pbcboxlen**
- int **setpbcs**
- char **units** [CHR_MAXLEN]
- int **setunits**
- int **gridpt**
- int **setgridpt**
- char **map3dname** [CHR_MAXLEN]
- int **set3dmap**
- char **grid2Dname** [PBAMPARM_MAXWRITE][CHR_MAXLEN]
- char **grid2Dax** [PBAMPARM_MAXWRITE][CHR_MAXLEN]
- double **grid2Dloc** [PBAMPARM_MAXWRITE]
- int **grid2Dct**
- int **setgrid2Dname**
- char **dxname** [CHR_MAXLEN]
- int **setdxname**
- int **ntraj**
- int **setntraj**
- char **termcombine** [CHR_MAXLEN]
- int **settermcombine**
- int **diffct**

- char **moveType** [PBAMPARM_MAXMOL][CHR_MAXLEN]
- double **transDiff** [PBAMPARM_MAXMOL]
- double **rotDiff** [PBAMPARM_MAXMOL]
- int **termct**
- int **setterm**
- char **termnam** [PBAMPARM_MAXWRITE][CHR_MAXLEN]
- int **termnu** [PBAMPARM_MAXWRITE][1]
- double **termVal** [PBAMPARM_MAXWRITE]
- char **confil** [PBAMPARM_MAXWRITE][CHR_MAXLEN]
- double **conpad** [PBAMPARM_MAXWRITE]
- int **confilct**
- int **setxyz**
- int **xyzct** [PBAMPARM_MAXMOL]
- char **xyzfil** [PBAMPARM_MAXMOL][PBAMPARM_MAXWRITE][CHR_MAXLEN]

8.9.1 Detailed Description

Parameter structure for PBAM-specific variables from input files.

Author

Andrew Stevens, Kyle Monson

Note

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

Definition at line 105 of file pbamparm.h.

8.9.2 Field Documentation

8.9.2.1 parsed

```
int parsed
```

Has this structure been filled? (0 = no, 1 = yes)
Definition at line 108 of file pbamparm.h.

8.9.2.2 type

```
PBAMparm_CalcType type
```

What type of PBAM calculation?
Definition at line 107 of file pbamparm.h.
The documentation for this struct was generated from the following file:

- src/generic/pbamparm.h

8.10 sPBEparm Struct Reference

Parameter structure for PBE variables from input files.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/pbeparm.h>`

Data Fields

- int molid
- int setmolid
- int useDielMap
- int dielMapID
- int useKappaMap
- int kappaMapID
- int usePotMap
- int potMapID
- int useChargeMap
- int chargeMapID
- Vhal_PBEType pbetype
- int setpbetype
- Vbcfl bcfl
- int setbcfl
- int nion
- int setnion
- double ionq [MAXION]
- double ionc [MAXION]
- double ionr [MAXION]
- int setion [MAXION]
- double pdie
- int setpdie
- double sdens
- int setsdens
- double sdie
- int setsdie
- Vsurf_Meth srfm
- int setsrfm
- double srad
- int setsrad
- double swin
- int setswin
- double temp
- int settemp
- double smsize
- int setsmsize
- double smvolume
- int setsmvolume
- PBEparm_calcEnergy calcenergy
- int setcalcenergy
- PBEparm_calcForce calcforce
- int setcalcforce
- double zmem
- int setzmem
- double Lmem
- int setLmem
- double mdie
- int setmdie
- double memv

- int setmemv
- int numwrite
- char writestem [PBEPARM_MAXWRITE][VMAX_ARGLEN]
- Vdata_Type writetype [PBEPARM_MAXWRITE]
- Vdata_Format writefmt [PBEPARM_MAXWRITE]
- int writemat
- int setwritemat
- char writematstem [VMAX_ARGLEN]
- int writematflag
- char **pbam_3dmapstem** [VMAX_ARGLEN]
- int **pbam_3dmapflag**
- int parsed

8.10.1 Detailed Description

Parameter structure for PBE variables from input files.

Author

Nathan Baker

Note

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

Definition at line 117 of file pbeparm.h.

8.10.2 Field Documentation

8.10.2.1 bcfl

`Vbcfl bcfl`
Boundary condition method
Definition at line 136 of file pbeparm.h.

8.10.2.2 calcenergy

`PBEparm_calcEnergy calcenergy`
Energy calculation flag
Definition at line 165 of file pbeparm.h.

8.10.2.3 calcforce

`PBEparm_calcForce calcforce`
Atomic forces calculation
Definition at line 167 of file pbeparm.h.

8.10.2.4 chargeMapID

`int chargeMapID`
Charge distribution map ID (if used)
Definition at line 133 of file pbeparm.h.

8.10.2.5 dielMapID

`int dielMapID`
Dielectric map ID (if used)
Definition at line 123 of file pbeparm.h.

8.10.2.6 ionc

`double ionc[MAXION]`
Counterion concentrations (in M)
Definition at line 141 of file pbeparm.h.

8.10.2.7 ionq

`double ionq[MAXION]`
Counterion charges (in e)
Definition at line 140 of file pbeparm.h.

8.10.2.8 ionr

`double ionr[MAXION]`
Counterion radii (in A)
Definition at line 142 of file pbeparm.h.

8.10.2.9 kappaMapID

`int kappaMapID`
Kappa map ID (if used)
Definition at line 126 of file pbeparm.h.

8.10.2.10 Lmem

`double Lmem`
membrane width
Definition at line 176 of file pbeparm.h.

8.10.2.11 mdie

`double mdie`
membrane dielectric constant
Definition at line 178 of file pbeparm.h.

8.10.2.12 memv

`double memv`
Membrane potential
Definition at line 180 of file pbeparm.h.

8.10.2.13 molid

`int molid`
Molecule ID to perform calculation on
Definition at line 119 of file pbeparm.h.

8.10.2.14 nion

`int nion`
Number of counterion species
Definition at line 138 of file pbeparm.h.

8.10.2.15 numwrite

`int numwrite`
Number of write statements encountered
Definition at line 185 of file pbeparm.h.

8.10.2.16 parsed

`int parsed`
Has this been filled with anything other than the default values?
Definition at line 205 of file pbeparm.h.

8.10.2.17 pbetype

`Vhal_PBEType pbetype`
Which version of the PBE are we solving?
Definition at line 134 of file pbeparm.h.

8.10.2.18 pdie

`double pdie`
Solute dielectric
Definition at line 144 of file pbeparm.h.

8.10.2.19 potMapID

`int potMapID`
Kappa map ID (if used)
Definition at line 129 of file pbeparm.h.

8.10.2.20 sdens

`double sdens`
Vacc sphere density
Definition at line 146 of file pbeparm.h.

8.10.2.21 sdie

`double sdie`
Solvent dielectric
Definition at line 148 of file pbeparm.h.

8.10.2.22 setbcfl

`int setbcfl`
Flag,

See also

bcfl

Definition at line 137 of file pbeparm.h.

8.10.2.23 setcalcenergy

`int setcalcenergy`
Flag,

See also

calcenergy

Definition at line 166 of file pbeparm.h.

8.10.2.24 setcalcforce

`int setcalcforce`
Flag,

See also

calcforce

Definition at line 168 of file pbeparm.h.

8.10.2.25 setion

`int setion[MAXION]`
Flag,

See also

ionq

Definition at line 143 of file pbeparm.h.

8.10.2.26 setLmem

`int setLmem`
Flag
Definition at line 177 of file pbeparm.h.

8.10.2.27 setmdie

`int setmdie`
Flag
Definition at line 179 of file pbeparm.h.

8.10.2.28 setmemv

`int setmemv`
Flag
Definition at line 181 of file pbeparm.h.

8.10.2.29 setmolid

`int setmolid`
Flag,

See also

molid

Definition at line 120 of file pbeparm.h.

8.10.2.30 setnion

`int setnion`
Flag,

See also

nion

Definition at line 139 of file pbeparm.h.

8.10.2.31 setpbetype

`int setpbetype`
Flag,

See also

pbetype

Definition at line 135 of file pbeparm.h.

8.10.2.32 setpdie

`int setpdie`
Flag,

See also

pdie

Definition at line 145 of file pbeparm.h.

8.10.2.33 setsdens

`int setsdens`
Flag,

See also

sdens

Definition at line 147 of file pbeparm.h.

8.10.2.34 setsdie

`int setsdie`
Flag,

See also

sdie

Definition at line 149 of file pbeparm.h.

8.10.2.35 setsmsize

`int setsmsize`
Flag,

See also

temp

Definition at line 160 of file pbeparm.h.

8.10.2.36 setsmvolume

`int setsmvolume`
Flag,

See also

temp

Definition at line 163 of file pbeparm.h.

8.10.2.37 setsrad

`int setsrad`
Flag,

See also

srad

Definition at line 153 of file pbeparm.h.

8.10.2.38 setsrfm

`int setsrfm`
Flag,

See also

srfm

Definition at line 151 of file pbeparm.h.

8.10.2.39 setswin

`int setswin`
Flag,

See also

swin

Definition at line 155 of file pbeparm.h.

8.10.2.40 settemp

`int settemp`
Flag,

See also

temp

Definition at line 157 of file pbeparm.h.

8.10.2.41 setwritemat

`int setwritemat`
Flag,

See also

writemat

Definition at line 194 of file pbeparm.h.

8.10.2.42 setzmem

`int setzmem`
Flag
Definition at line 175 of file pbeparm.h.

8.10.2.43 smsize

`double smsize`
SMPBE size
Definition at line 159 of file pbeparm.h.

8.10.2.44 smvolume

`double smvolume`
SMPBE size
Definition at line 162 of file pbeparm.h.

8.10.2.45 srad

`double srad`
Solvent radius
Definition at line 152 of file pbeparm.h.

8.10.2.46 srfm

`Vsurf_Meth srfm`
Surface calculation method
Definition at line 150 of file pbeparm.h.

8.10.2.47 swin

`double swin`
Cubic spline window
Definition at line 154 of file pbeparm.h.

8.10.2.48 temp

`double temp`
Temperature (in K)
Definition at line 156 of file pbeparm.h.

8.10.2.49 useChargeMap

`int useChargeMap`
Indicates whether we use an external charge distribution map
Definition at line 131 of file pbeparm.h.

8.10.2.50 useDielMap

`int useDielMap`
Indicates whether we use external dielectric maps (note plural)
Definition at line 121 of file pbeparm.h.

8.10.2.51 useKappaMap

`int useKappaMap`
Indicates whether we use an external kappa map
Definition at line 124 of file pbeparm.h.

8.10.2.52 usePotMap

`int usePotMap`
Indicates whether we use an external kappa map
Definition at line 127 of file pbeparm.h.

8.10.2.53 writefmt

`Vdata_Format writefmt[PBEPARM_MAXWRITE]`
File format to write data in
Definition at line 189 of file pbeparm.h.

8.10.2.54 writemat

`int writemat`
Write out the operator matrix?

- 0 => no
- 1 => yes

Definition at line 191 of file pbeparm.h.

8.10.2.55 writematflag

`int writematflag`
What matrix should we write:

- 0 => Poisson (differential operator)
- 1 => Poisson-Boltzmann operator linearized around solution (if applicable)

Definition at line 196 of file pbeparm.h.

8.10.2.56 writematstem

`char writematstem[VMAX_ARGLEN]`
File stem to write mat
Definition at line 195 of file pbeparm.h.

8.10.2.57 writestem

```
char writestem[PBEPARM_MAXWRITE][VMAX_ARGLEN]
```
File stem to write data to
Definition at line 186 of file pbeparm.h.

8.10.2.58 writetype

```
Vdata_Type writetype[PBEPARM_MAXWRITE]
```
What data to write
Definition at line 188 of file pbeparm.h.

8.10.2.59 zmem

```
double zmem
```
z value of membrane bottom
Definition at line 174 of file pbeparm.h.
The documentation for this struct was generated from the following file:

- src/generic/pbeparm.h

8.11 sPBSAMparm Struct Reference

Parameter structure for PBSAM-specific variables from input files.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/pbsamparm.h>`

Data Fields

- PBSAMparm_CalcType type
- int parsed
- int **settolsp**
- double **tolsp**
- int **setmsms**
- double **probe_radius**
- double **density**
- int **setsurf**
- int **surfct**
- char **surffil** [PBSAMPARM_MAXMOL][CHR_MAXLEN]
- int **setimat**
- int **imatct**
- char **imatfil** [PBSAMPARM_MAXMOL][CHR_MAXLEN]
- int **setexp**
- int **expct**
- char **expfil** [PBSAMPARM_MAXMOL][CHR_MAXLEN]

8.11.1 Detailed Description

Parameter structure for PBSAM-specific variables from input files.

Author

Lisa Felberg

Note

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

Definition at line 105 of file pbsamparm.h.

8.11.2 Field Documentation

8.11.2.1 parsed

`int parsed`
Has this structure been filled? (0 = no, 1 = yes)
Definition at line 108 of file pbsamparm.h.

8.11.2.2 type

`PBSAMparm_CalcType type`
What type of PBSAM calculation?
Definition at line 107 of file pbsamparm.h.
The documentation for this struct was generated from the following file:

- src/generic/pbsamparm.h

8.12 sVacc Struct Reference

Oracle for solvent- and ion-accessibility around a biomolecule.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/vacc.h>`

Data Fields

- Vmem * mem
- Valist * alist
- Vclist * clist
- int * atomFlags
- VaccSurf * refSphere
- VaccSurf ** surf
- Vset acc
- double surf_density

8.12.1 Detailed Description

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

Author

Nathan Baker

Definition at line 108 of file vacc.h.

8.12.2 Field Documentation

8.12.2.1 acc

`Vset acc`
An integer array (to be treated as bitfields) of Vset type with length equal to the number of vertices in the mesh
Definition at line 120 of file vacc.h.

8.12.2.2 alist

`Valist* alist`
Valist structure for list of atoms
Definition at line 111 of file vacc.h.

8.12.2.3 atomFlags

`int* atomFlags`
Array of boolean flags of length Valist_getNumberAtoms(thee->alist) to prevent double-counting atoms during calculations
Definition at line 113 of file vacc.h.

8.12.2.4 clist

`Vclist* clist`
Vclist structure for atom cell list
Definition at line 112 of file vacc.h.

8.12.2.5 mem

`Vmem* mem`
Memory management object for this class
Definition at line 110 of file vacc.h.

8.12.2.6 refSphere

`VaccSurf* refSphere`
Reference sphere for SASA calculations
Definition at line 116 of file vacc.h.

8.12.2.7 surf

`VaccSurf** surf`
Array of surface points for each atom; is not initialized until needed (test against VNULL to determine initialization state)
Definition at line 117 of file vacc.h.

8.12.2.8 surf_density

`double surf_density`
Minimum solvent accessible surface point density (in pts/A^2)
Definition at line 122 of file vacc.h.
The documentation for this struct was generated from the following file:

- src/generic/vacc.h

8.13 sVaccSurf Struct Reference

Surface object list of per-atom surface points.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/vacc.h>`

Data Fields

- Vmem * mem
- double * xpts
- double * ypts
- double * zpts
- char * bpts
- double area
- int npts
- double probe_radius

8.13.1 Detailed Description

Surface object list of per-atom surface points.

Author

Nathan Baker

Definition at line 84 of file vacc.h.

8.13.2 Field Documentation

8.13.2.1 area

`double area`
Area spanned by these points
Definition at line 91 of file vacc.h.

8.13.2.2 bpts

`char* bpts`
Array of booleans indicating whether a point is (1) or is not (0) part of the surface
Definition at line 89 of file vacc.h.

8.13.2.3 mem

`Vmem* mem`
Memory object
Definition at line 85 of file vacc.h.

8.13.2.4 npts

`int npts`
Length of thee->xpts, ypts, zpts arrays
Definition at line 92 of file vacc.h.

8.13.2.5 probe_radius

`double probe_radius`
Probe radius (A) with which this surface was constructed
Definition at line 93 of file vacc.h.

8.13.2.6 xpts

`double* xpts`
Array of point x-locations
Definition at line 86 of file vacc.h.

8.13.2.7 ypts

`double* ypts`
Array of point y-locations
Definition at line 87 of file vacc.h.

8.13.2.8 zpts

`double* zpts`
Array of point z-locations
Definition at line 88 of file vacc.h.
The documentation for this struct was generated from the following file:

- src/generic/vacc.h

8.14 sValist Struct Reference

Container class for list of atom objects.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/valist.h>`

Data Fields

- int number
- double center [3]
- double mincrd [3]
- double maxcrd [3]
- double maxrad
- double charge
- Vatom * atoms
- Vmem * vmem

8.14.1 Detailed Description

Container class for list of atom objects.

Author

Nathan Baker

Definition at line 78 of file valist.h.

8.14.2 Field Documentation

8.14.2.1 atoms

```
Vatom* atoms
```

Atom list
Definition at line 86 of file valist.h.

8.14.2.2 center

```
double center[3]
```

Molecule center (xmin - xmax)/2, etc.
Definition at line 81 of file valist.h.

8.14.2.3 charge

```
double charge
```

Net charge
Definition at line 85 of file valist.h.

8.14.2.4 maxcrd

```
double maxcrd[3]
```

Maximum coordinates
Definition at line 83 of file valist.h.

8.14.2.5 maxrad

```
double maxrad
```

Maximum radius
Definition at line 84 of file valist.h.

8.14.2.6 mincrd

```
double mincrd[3]
```

Minimum coordinates
Definition at line 82 of file valist.h.

8.14.2.7 number

`int number`
Number of atoms in list
Definition at line 80 of file valist.h.

8.14.2.8 vmem

`Vmem* vmem`
Memory management object
Definition at line 87 of file valist.h.
The documentation for this struct was generated from the following file:

- src/generic/valist.h

8.15 sVatom Struct Reference

Contains public data members for Vatom class/module.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/vatom.h>`

Data Fields

- double position [3]
- double radius
- double charge
- double partID
- double epsilon
- int id
- char resName [VMAX_RECLEN]
- char atomName [VMAX_RECLEN]

8.15.1 Detailed Description

Contains public data members for Vatom class/module.

Author

Nathan Baker, David Gohara, Mike Schneiders

Definition at line 84 of file vatom.h.

8.15.2 Field Documentation

8.15.2.1 atomName

`char atomName[VMAX_RECLEN]`
Atom name from PDB/PDR file
Definition at line 98 of file vatom.h.

8.15.2.2 charge

`double charge`
Atomic charge

Definition at line 88 of file vatom.h.

8.15.2.3 epsilon

`double epsilon`
Epsilon value for WCA calculations
Definition at line 91 of file vatom.h.

8.15.2.4 id

`int id`
Atomic ID; this should be a unique non-negative integer assigned based on the index of the atom in a Valist atom array
Definition at line 93 of file vatom.h.

8.15.2.5 partID

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

Definition at line 89 of file vatom.h.

8.15.2.6 position

`double position[3]`
Atomic position
Definition at line 86 of file vatom.h.

8.15.2.7 radius

`double radius`
Atomic radius

Definition at line 87 of file vatom.h.

8.15.2.8 resName

`char resName[VMAX_RECLEN]`
Residue name from PDB/PQR file
Definition at line 97 of file vatom.h.
The documentation for this struct was generated from the following file:

- src/generic/vatom.h

8.16 sVclist Struct Reference

Atom cell list.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/vclist.h>`

Data Fields

- Vmem * vmem
- Valist * alist
- Vclist_DomainMode mode
- int npts [VAPBS_DIM]
- int n
- double max_radius
- VclistCell * cells
- double lower_corner [VAPBS_DIM]
- double upper_corner [VAPBS_DIM]
- double spacs [VAPBS_DIM]

8.16.1 Detailed Description

Atom cell list.

Author

Nathan Baker

Definition at line 117 of file vclist.h.

8.16.2 Field Documentation

8.16.2.1 alist

`Valist* alist`
Original Valist structure for list of atoms
Definition at line 120 of file vclist.h.

8.16.2.2 cells

`VclistCell* cells`
Cell array of length thee->n
Definition at line 125 of file vclist.h.

8.16.2.3 lower_corner

`double lower_corner[VAPBS_DIM]`
Hash table grid corner
Definition at line 126 of file vclist.h.

8.16.2.4 max_radius

`double max_radius`
Maximum probe radius
Definition at line 124 of file vclist.h.

8.16.2.5 mode

`Vclist_DomainMode mode`
How the cell list was constructed
Definition at line 121 of file vclist.h.

8.16.2.6 n

`int n`
n = nx∗nz∗ny
Definition at line 123 of file vclist.h.

8.16.2.7 npts

`int npts[VAPBS_DIM]`
Hash table grid dimensions
Definition at line 122 of file vclist.h.

8.16.2.8 spacs

`double spacs[VAPBS_DIM]`
Hash table grid spacings
Definition at line 128 of file vclist.h.

8.16.2.9 upper_corner

`double upper_corner[VAPBS_DIM]`
Hash table grid corner
Definition at line 127 of file vclist.h.

8.16.2.10 vmem

`Vmem∗ vmem`
Memory management object for this class
Definition at line 119 of file vclist.h.
The documentation for this struct was generated from the following file:

- src/generic/vclist.h

8.17 sVclistCell Struct Reference

Atom cell list cell.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/vclist.h>`

Data Fields

- Vatom ** atoms
- int natoms

8.17.1 Detailed Description

Atom cell list cell.

Author

Nathan Baker

Definition at line 101 of file vclist.h.

8.17.2 Field Documentation

8.17.2.1 atoms

`Vatom** atoms`
Array of atom objects associated with this cell
Definition at line 102 of file vclist.h.

8.17.2.2 natoms

`int natoms`
Length of thee->atoms array
Definition at line 103 of file vclist.h.
The documentation for this struct was generated from the following file:

- src/generic/vclist.h

8.18 sVcsm Struct Reference

Charge-simplex map class.
`#include </builddir/build/BUILD/apbs-3.0.0/src/fem/vcsm.h>`

Data Fields

- Valist * alist
- int natom
- Gem * gm
- int ** sqm
- int * nsqm
- int nsimp
- int msimp
- int ** qsm
- int * nqsm
- int initFlag
- Vmem * vmem

8.18.1 Detailed Description

Charge-simplex map class.

Author

Nathan Baker

Definition at line 89 of file vcsm.h.

8.18.2 Field Documentation

8.18.2.1 alist

`Valist* alist`
Atom (charge) list
Definition at line 91 of file vcsm.h.

8.18.2.2 gm

`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.

8.18.2.3 initFlag

`int initFlag`
Indicates whether the maps have been initialized yet
Definition at line 112 of file vcsm.h.

8.18.2.4 msimp

`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.

8.18.2.5 natom

`int natom`
Size of thee->alist; redundant, but useful for convenience
Definition at line 92 of file vcsm.h.

8.18.2.6 nqsm

`int* nqsm`
The length of the simplex lists in thee->qsm
Definition at line 111 of file vcsm.h.

8.18.2.7 nsimp

`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.

8.18.2.8 nsqm

`int* nsqm`
The length of the charge lists in thee->sqm
Definition at line 104 of file vcsm.h.

8.18.2.9 qsm

`int** qsm`
The inverse of sqm; the list of simplices associated with a given charge
Definition at line 109 of file vcsm.h.

8.18.2.10 sqm

`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.

8.18.2.11 vmem

`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/vcsm.h

8.19 sVfetk Struct Reference

Contains public data members for Vfetk class/module.
`#include </builddir/build/BUILD/apbs-3.0.0/src/fem/vfetk.h>`

Data Fields

- Vmem * vmem
- Gem * gm
- AM * am
- Aprx * aprx
- PDE * pde
- Vpbe * pbe
- Vcsm * csm
- Vfetk_LsolvType lkey

- int lmax
- double ltol
- Vfetk_NsolvType nkey
- int nmax
- double ntol
- Vfetk_GuessType gues
- Vfetk_PrecType lprec
- int pjac
- PBEparm * pbeparm
- FEMparm * feparm
- Vhal_PBEType type
- int level

8.19.1 Detailed Description

Contains public data members for Vfetk class/module.

Author

Nathan Baker

Many of the routines and macros are borrowed from the main.c driver (written by Mike Holst) provided with the PMG code.
Definition at line 176 of file vfetk.h.

8.19.2 Field Documentation

8.19.2.1 am

```
AM* am
```

Multilevel algebra manager.
Definition at line 182 of file vfetk.h.

8.19.2.2 aprx

```
Aprx* aprx
```

Approximation manager.
Definition at line 183 of file vfetk.h.

8.19.2.3 csm

```
Vcsm* csm
```

Charge-simplex map
Definition at line 186 of file vfetk.h.

8.19.2.4 feparm

```
FEMparm* feparm
```

FEM-specific parameters
Definition at line 198 of file vfetk.h.

8.19.2.5 gm

`Gem* gm`
Grid manager (container class for master vertex and simplex lists as well as prolongation operator for updating after refinement).

Definition at line 179 of file vfetk.h.

8.19.2.6 gues

`Vfetk_GuessType gues`
Initial guess method
Definition at line 193 of file vfetk.h.

8.19.2.7 level

`int level`
Refinement level (starts at 0)
Definition at line 200 of file vfetk.h.

8.19.2.8 lkey

`Vfetk_LsolvType lkey`
Linear solver method
Definition at line 187 of file vfetk.h.

8.19.2.9 lmax

`int lmax`
Maximum number of linear solver iterations
Definition at line 188 of file vfetk.h.

8.19.2.10 lprec

`Vfetk_PrecType lprec`
Linear preconditioner
Definition at line 194 of file vfetk.h.

8.19.2.11 ltol

`double ltol`
Residual tolerance for linear solver
Definition at line 189 of file vfetk.h.

8.19.2.12 nkey

`Vfetk_NsolvType nkey`
Nonlinear solver method
Definition at line 190 of file vfetk.h.

8.19.2.13 nmax

`int nmax`
Maximum number of nonlinear solver iterations
Definition at line 191 of file vfetk.h.

8.19.2.14 ntol

`double ntol`
Residual tolerance for nonlinear solver
Definition at line 192 of file vfetk.h.

8.19.2.15 pbe

`Vpbe* pbe`
Poisson-Boltzmann object
Definition at line 185 of file vfetk.h.

8.19.2.16 pbeparm

`PBEparm* pbeparm`
Generic PB parameters
Definition at line 197 of file vfetk.h.

8.19.2.17 pde

`PDE* pde`
FEtk PDE object
Definition at line 184 of file vfetk.h.

8.19.2.18 pjac

`int pjac`
Flag to print the jacobians (usually set this to -1, please)
Definition at line 195 of file vfetk.h.

8.19.2.19 type

`Vhal_PBEType type`
Version of PBE to solve
Definition at line 199 of file vfetk.h.

8.19.2.20 vmem

`Vmem* vmem`
Memory management object

Definition at line 178 of file vfetk.h.
The documentation for this struct was generated from the following file:

- src/fem/vfetk.h

8.20 sVfetk_LocalVar Struct Reference

Vfetk LocalVar subclass.
`#include </builddir/build/BUILD/apbs-3.0.0/src/fem/vfetk.h>`

Data Fields

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

8.20.1 Detailed Description

Vfetk LocalVar subclass.

Author

Nathan Baker

Contains variables used when solving the PDE with FEtk
Definition at line 215 of file vfetk.h.

8.20.2 Field Documentation

8.20.2.1 A

`double A`
Second-order differential term
Definition at line 228 of file vfetk.h.

8.20.2.2 B

`double B`
Entire ionic strength term
Definition at line 230 of file vfetk.h.

8.20.2.3 d2W

`double d2W`
Coulomb regularization term Laplacia
Definition at line 223 of file vfetk.h.

8.20.2.4 DB

`double DB`
Entire ionic strength term derivative
Definition at line 231 of file vfetk.h.

8.20.2.5 delta

`double delta`
Store delta value
Definition at line 250 of file vfetk.h.

8.20.2.6 DFu_wv

`double DFu_wv`
Store DFu_wv value
Definition at line 249 of file vfetk.h.

8.20.2.7 dU

`double dU[MAXV][VAPBS_DIM]`
Solution gradient
Definition at line 220 of file vfetk.h.

8.20.2.8 dW

```
double dW[VAPBS_DIM]
```
Coulomb regularization term gradient
Definition at line 222 of file vfetk.h.

8.20.2.9 F

```
double F
```
RHS characteristic function value
Definition at line 229 of file vfetk.h.

8.20.2.10 fetk

```
Vfetk* fetk
```
Pointer to the VFETK object
Definition at line 233 of file vfetk.h.

8.20.2.11 fType

```
int fType
```
Face type
Definition at line 225 of file vfetk.h.

8.20.2.12 Fu_v

```
double Fu_v
```
Store Fu_v value
Definition at line 248 of file vfetk.h.

8.20.2.13 green

```
Vgreen* green
```
Pointer to a Green's function object
Definition at line 234 of file vfetk.h.

8.20.2.14 initGreen

```
int initGreen
```
Boolean to designate whether Green's function has been initialized
Definition at line 235 of file vfetk.h.

8.20.2.15 ionConc

```
double ionConc[MAXION]
```
Counterion species' concentrations
Definition at line 241 of file vfetk.h.

8.20.2.16 ionQ

`double ionQ[MAXION]`
Counterion species' valencies
Definition at line 242 of file vfetk.h.

8.20.2.17 ionRadii

`double ionRadii[MAXION]`
Counterion species' radii
Definition at line 243 of file vfetk.h.

8.20.2.18 ionstr

`double ionstr`
Ionic strength parameters (M)
Definition at line 246 of file vfetk.h.

8.20.2.19 jumpDiel

`double jumpDiel`
Dielectric value on one side of a simplex face
Definition at line 232 of file vfetk.h.

8.20.2.20 nion

`int nion`
Number of ion species
Definition at line 247 of file vfetk.h.

8.20.2.21 nvec

`double nvec[VAPBS_DIM]`
Normal vector for a simplex face
Definition at line 216 of file vfetk.h.

8.20.2.22 nverts

`int nverts`
number of vertices in the simplex
Definition at line 240 of file vfetk.h.

8.20.2.23 simp

`SS* simp`
Pointer to the latest simplex object; set in initElement() and delta()
Definition at line 237 of file vfetk.h.

8.20.2.24 sType

`int sType`
Simplex type
Definition at line 224 of file vfetk.h.

8.20.2.25 U

`double U[MAXV]`
Solution value
Definition at line 219 of file vfetk.h.

8.20.2.26 u_D

`double u_D`
Store Dirichlet value
Definition at line 251 of file vfetk.h.

8.20.2.27 u_T

`double u_T`
Store true value
Definition at line 252 of file vfetk.h.

8.20.2.28 verts

`VV* verts[4]`
Pointer to the latest vertices; set in initElement
Definition at line 239 of file vfetk.h.

8.20.2.29 vx

`double vx[4][VAPBS_DIM]`
Vertex coordinates
Definition at line 217 of file vfetk.h.

8.20.2.30 W

`double W`
Coulomb regularization term scalar value
Definition at line 221 of file vfetk.h.

8.20.2.31 xq

`double xq[VAPBS_DIM]`
Quadrature pt
Definition at line 218 of file vfetk.h.

8.20.2.32 zkappa2

```
double zkappa2
```

Ionic strength parameters
Definition at line 244 of file vfetk.h.

8.20.2.33 zks2

```
double zks2
```

Ionic strength parameters
Definition at line 245 of file vfetk.h.
The documentation for this struct was generated from the following file:

- src/fem/vfetk.h

8.21 sVgreen Struct Reference

Contains public data members for Vgreen class/module.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/vgreen.h>`

Data Fields

- Valist * alist
- Vmem * vmem
- double * xp
- double * yp
- double * zp
- double * qp
- int np

8.21.1 Detailed Description

Contains public data members for Vgreen class/module.

Author

Nathan Baker

Definition at line 82 of file vgreen.h.

8.21.2 Field Documentation

8.21.2.1 alist

```
Valist* alist
```

Atom (charge) list for Green's function
Definition at line 84 of file vgreen.h.

8.21.2.2 np

`int np`
Set to size of above arrays
Definition at line 94 of file vgreen.h.

8.21.2.3 qp

`double* qp`
Array of particle charges for use with treecode routines
Definition at line 92 of file vgreen.h.

8.21.2.4 vmem

`Vmem* vmem`
Memory management object
Definition at line 85 of file vgreen.h.

8.21.2.5 xp

`double* xp`
Array of particle x-coordinates for use with treecode routines
Definition at line 86 of file vgreen.h.

8.21.2.6 yp

`double* yp`
Array of particle y-coordinates for use with treecode routines
Definition at line 88 of file vgreen.h.

8.21.2.7 zp

`double* zp`
Array of particle z-coordinates for use with treecode routines
Definition at line 90 of file vgreen.h.
The documentation for this struct was generated from the following file:

- src/generic/vgreen.h

8.22 sVgrid Struct Reference

Electrostatic potential oracle for Cartesian mesh data.
`#include </builddir/build/BUILD/apbs-3.0.0/src/mg/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

8.22.1 Detailed Description

Electrostatic potential oracle for Cartesian mesh data.

Author

Nathan Baker

Definition at line 81 of file vgrid.h.

8.22.2 Field Documentation

8.22.2.1 ctordata

`int ctordata`
flag indicating whether data was included at construction
Definition at line 97 of file vgrid.h.

8.22.2.2 data

`double* data`
nx*ny*nz array of data
Definition at line 95 of file vgrid.h.

8.22.2.3 hx

`double hx`
Grid spacing in x direction
Definition at line 86 of file vgrid.h.

8.22.2.4 hy

`double hy`
Grid spacing in y direction
Definition at line 87 of file vgrid.h.

8.22.2.5 hzed

`double hzed`
Grid spacing in z direction
Definition at line 88 of file vgrid.h.

8.22.2.6 mem

`Vmem* mem`
Memory manager object
Definition at line 99 of file vgrid.h.

8.22.2.7 nx

`int nx`
Number grid points in x direction
Definition at line 83 of file vgrid.h.

8.22.2.8 ny

`int ny`
Number grid points in y direction
Definition at line 84 of file vgrid.h.

8.22.2.9 nz

`int nz`
Number grid points in z direction
Definition at line 85 of file vgrid.h.

8.22.2.10 readdata

`int readdata`
flag indicating whether data was read from file
Definition at line 96 of file vgrid.h.

8.22.2.11 xmax

`double xmax`
x coordinate of upper grid corner
Definition at line 92 of file vgrid.h.

8.22.2.12 xmin

`double xmin`
x coordinate of lower grid corner
Definition at line 89 of file vgrid.h.

8.22.2.13 ymax

`double ymax`
y coordinate of upper grid corner
Definition at line 93 of file vgrid.h.

8.22.2.14 ymin

`double ymin`
y coordinate of lower grid corner
Definition at line 90 of file vgrid.h.

8.22.2.15 zmax

`double zmax`
z coordinate of upper grid corner
Definition at line 94 of file vgrid.h.

8.22.2.16 zmin

`double zmin`
z coordinate of lower grid corner
Definition at line 91 of file vgrid.h.
The documentation for this struct was generated from the following file:

- src/mg/vgrid.h

8.23 sVmgrid Struct Reference

Multiresoltion oracle for Cartesian mesh data.
`#include </builddir/build/BUILD/apbs-3.0.0/src/mg/vmgrid.h>`

Data Fields

- int ngrids
- Vgrid * grids [VMGRIDMAX]

8.23.1 Detailed Description

Multiresoltion oracle for Cartesian mesh data.

Author

Nathan Baker

Definition at line 84 of file vmgrid.h.

8.23.2 Field Documentation

8.23.2.1 grids

```
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.

8.23.2.2 ngrids

```
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/vmgrid.h

8.24 sVopot Struct Reference

Electrostatic potential oracle for Cartesian mesh data.
`#include </builddir/build/BUILD/apbs-3.0.0/src/mg/vopot.h>`

Data Fields

- Vmgrid * mgrid
- Vpbe * pbe
- Vbcfl bcfl

8.24.1 Detailed Description

Electrostatic potential oracle for Cartesian mesh data.

Author

Nathan Baker

Definition at line 83 of file vopot.h.

8.24.2 Field Documentation

8.24.2.1 bcfl

```
Vbcfl bcfl
```

Boundary condition flag for returning potential values at points off the grid.
Definition at line 88 of file vopot.h.

8.24.2.2 mgrid

```
Vmgrid* mgrid
```

Multiple grid object containing potential data (in units kT/e)
Definition at line 85 of file vopot.h.

8.24.2.3 pbe

`Vpbe* pbe`
Pointer to PBE object
Definition at line 87 of file vopot.h.
The documentation for this struct was generated from the following file:

- src/mg/vopot.h

8.25 sVparam_AtomData Struct Reference

AtomData sub-class; stores atom data.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/vparam.h>`

Data Fields

- char atomName [VMAX_ARGLEN]
- char resName [VMAX_ARGLEN]
- double charge
- double radius
- double epsilon

8.25.1 Detailed Description

AtomData sub-class; stores atom data.

Author

Nathan Baker

Note

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

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

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

Definition at line 92 of file vparam.h.

8.25.2 Field Documentation

8.25.2.1 atomName

`char atomName[VMAX_ARGLEN]`
Atom name
Definition at line 93 of file vparam.h.

8.25.2.2 charge

`double charge`
Atom charge (in e)
Definition at line 95 of file vparam.h.

8.25.2.3 epsilon

`double epsilon`
Atom VdW well depth (ϵ_i above; in kJ/mol)
Definition at line 97 of file vparam.h.

8.25.2.4 radius

`double radius`
Atom VdW radius (σ_i above; in Å)
Definition at line 96 of file vparam.h.

8.25.2.5 resName

`char resName[VMAX_ARGLEN]`
Residue name
Definition at line 94 of file vparam.h.
The documentation for this struct was generated from the following file:

- src/generic/vparam.h

8.26 sVpbe Struct Reference

Contains public data members for Vpbe class/module.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/vpbe.h>`

Data Fields

- Vmem * vmem
- Valist * alist
- Vclist * clist
- Vacc * acc
- double T
- double soluteDiel
- double solventDiel
- double solventRadius
- double bulkIonicStrength
- double maxIonRadius
- int numIon
- double ionConc [MAXION]
- double ionRadii [MAXION]
- double ionQ [MAXION]
- double xkappa
- double deblen

- double zkappa2
- double zmagic
- double soluteCenter [3]
- double soluteRadius
- double soluteXlen
- double soluteYlen
- double soluteZlen
- double soluteCharge
- double smvolume
- double smsize
- int ipkey
- int paramFlag
- double z_mem
- double L
- double membraneDiel
- double V
- int param2Flag

8.26.1 Detailed Description

Contains public data members for Vpbe class/module.

Author

Nathan Baker

Definition at line 84 of file vpbe.h.

8.26.2 Field Documentation

8.26.2.1 acc

```
Vacc* acc
```

Accessibility object
Definition at line 90 of file vpbe.h.

8.26.2.2 alist

```
Valist* alist
```

Atom (charge) list
Definition at line 88 of file vpbe.h.

8.26.2.3 bulkIonicStrength

```
double bulkIonicStrength
```

Bulk ionic strength (M)
Definition at line 99 of file vpbe.h.

8.26.2.4 clist

```
Vclist* clist
```

Atom location cell list
Definition at line 89 of file vpbe.h.

8.26.2.5 deblen

```
double deblen
```

Debye length (bulk)
Definition at line 109 of file vpbe.h.

8.26.2.6 ionConc

```
double ionConc[MAXION]
```

Concentration (M) of each species
Definition at line 104 of file vpbe.h.

8.26.2.7 ionQ

```
double ionQ[MAXION]
```

Charge (e) of each species
Definition at line 106 of file vpbe.h.

8.26.2.8 ionRadii

```
double ionRadii[MAXION]
```

Ionic radius (A) of each species
Definition at line 105 of file vpbe.h.

8.26.2.9 ipkey

```
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.

8.26.2.10 L

```
double L
```

Length of the membrane (A)
Definition at line 132 of file vpbe.h.

8.26.2.11 maxIonRadius

```
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.

8.26.2.12 membraneDiel

`double membraneDiel`
Membrane dielectric constant
Definition at line 133 of file vpbe.h.

8.26.2.13 numIon

`int numIon`
Total number of ion species
Definition at line 103 of file vpbe.h.

8.26.2.14 param2Flag

`int param2Flag`
Check to see if bcfl=3 parms have been set
Definition at line 135 of file vpbe.h.

8.26.2.15 paramFlag

`int paramFlag`
Check to see if the parameters have been set
Definition at line 125 of file vpbe.h.

8.26.2.16 smsize

`double smsize`
Size-Modified PBE size
Definition at line 121 of file vpbe.h.

8.26.2.17 smvolume

`double smvolume`
Size-Modified PBE relative volume
Definition at line 120 of file vpbe.h.

8.26.2.18 soluteCenter

`double soluteCenter[3]`
Center of solute molecule (A)
Definition at line 113 of file vpbe.h.

8.26.2.19 soluteCharge

`double soluteCharge`
Charge of solute molecule (e)
Definition at line 118 of file vpbe.h.

8.26.2.20 soluteDiel

`double soluteDiel`
Solute dielectric constant (unitless)
Definition at line 93 of file vpbe.h.

8.26.2.21 soluteRadius

`double soluteRadius`
Radius of solute molecule (A)
Definition at line 114 of file vpbe.h.

8.26.2.22 soluteXlen

`double soluteXlen`
Solute length in x-direction
Definition at line 115 of file vpbe.h.

8.26.2.23 soluteYlen

`double soluteYlen`
Solute length in y-direction
Definition at line 116 of file vpbe.h.

8.26.2.24 soluteZlen

`double soluteZlen`
Solute length in z-direction
Definition at line 117 of file vpbe.h.

8.26.2.25 solventDiel

`double solventDiel`
Solvent dielectric constant (unitless)
Definition at line 94 of file vpbe.h.

8.26.2.26 solventRadius

`double solventRadius`
Solvent probe radius (angstroms) for accessibility; determining defining volumes for the dielectric coefficient
Definition at line 95 of file vpbe.h.

8.26.2.27 T

`double T`
Temperature (K)
Definition at line 92 of file vpbe.h.

8.26.2.28 V

`double V`
Membrane potential
Definition at line 134 of file vpbe.h.

8.26.2.29 vmem

`Vmem* vmem`
Memory management object
Definition at line 86 of file vpbe.h.

8.26.2.30 xkappa

`double xkappa`
Debye-Huckel parameter (bulk)
Definition at line 108 of file vpbe.h.

8.26.2.31 z_mem

`double z_mem`
Z value of the botton of the membrane (A)
Definition at line 131 of file vpbe.h.

8.26.2.32 zkappa2

`double zkappa2`
Square of modified Debye-Huckel parameter (bulk)
Definition at line 110 of file vpbe.h.

8.26.2.33 zmagic

`double zmagic`
Delta function scaling parameter
Definition at line 111 of file vpbe.h.
The documentation for this struct was generated from the following file:

- src/generic/vpbe.h

8.27 sVpee Struct Reference

Contains public data members for Vpee class/module.
`#include </builddir/build/BUILD/apbs-3.0.0/src/fem/vpee.h>`

Data Fields

- Gem * gm
- int localPartID
- double localPartCenter [3]
- double localPartRadius

- int killFlag
- double killParam
- Vmem * mem

8.27.1 Detailed Description

Contains public data members for Vpee class/module.

Author

Nathan Baker

Definition at line 89 of file vpee.h.

8.27.2 Field Documentation

8.27.2.1 gm

`Gem* gm`
Grid manager
Definition at line 91 of file vpee.h.

8.27.2.2 killFlag

`int killFlag`
A flag indicating the method we're using to artificially decrease the error esimate outside the local partition
Definition at line 99 of file vpee.h.

8.27.2.3 killParam

`double killParam`
A parameter for the error estimate attenuation method
Definition at line 102 of file vpee.h.

8.27.2.4 localPartCenter

`double localPartCenter[3]`
The coordinates of the center of the local partition
Definition at line 95 of file vpee.h.

8.27.2.5 localPartID

`int localPartID`
The local partition ID: i.e. the partition whose boundary simplices we're keeping track of
Definition at line 92 of file vpee.h.

8.27.2.6 localPartRadius

`double localPartRadius`
The radius of the circle/sphere which circumscribes the local partition
Definition at line 97 of file vpee.h.

8.27.2.7 mem

`Vmem* mem`
Memory manager
Definition at line 104 of file vpee.h.
The documentation for this struct was generated from the following file:

- src/fem/vpee.h

8.28 sVpmg Struct Reference

Contains public data members for Vpmg class/module.
`#include </builddir/build/BUILD/apbs-3.0.0/src/mg/vpmg.h>`

Data Fields

- Vmem * vmem
- Vpmgp * pmgp
- Vpbe * pbe
- double * epsx
- double * epsy
- double * epsz
- double * kappa
- double * pot
- double * charge
- int * iparm
- double * rparm
- int * iwork
- double * rwork
- double * a1cf
- double * a2cf
- double * a3cf
- double * ccf
- double * fcf
- double * tcf
- double * u
- double * xf
- double * yf
- double * zf
- double * gxcf
- double * gycf
- double * gzcf
- double * pvec
- double extDiEnergy
- double extQmEnergy

- double extQfEnergy
- double extNpEnergy
- Vsurf_Meth surfMeth
- double splineWin
- Vchrg_Meth chargeMeth
- Vchrg_Src chargeSrc
- int filled
- int useDielXMap
- Vgrid * dielXMap
- int useDielYMap
- Vgrid * dielYMap
- int useDielZMap
- Vgrid * dielZMap
- int useKappaMap
- Vgrid * kappaMap
- int usePotMap
- Vgrid * potMap
- int useChargeMap
- Vgrid * chargeMap

8.28.1 Detailed Description

Contains public data members for Vpmg class/module.

Author

Nathan Baker

Many of the routines and macros are borrowed from the main.c driver (written by Mike Holst) provided with the PMG code.
Definition at line 116 of file vpmg.h.

8.28.2 Field Documentation

8.28.2.1 a1cf

```
double* a1cf
```

Operator coefficient values (a11) – this array can be overwritten
Definition at line 138 of file vpmg.h.

8.28.2.2 a2cf

```
double* a2cf
```

Operator coefficient values (a22) – this array can be overwritten
Definition at line 140 of file vpmg.h.

8.28.2.3 a3cf

```
double* a3cf
```

Operator coefficient values (a33) – this array can be overwritten
Definition at line 142 of file vpmg.h.

8.28.2.4 ccf

`double* ccf`
Helmholtz term – this array can be overwritten
Definition at line 144 of file vpmg.h.

8.28.2.5 charge

`double* charge`
Charge map
Definition at line 132 of file vpmg.h.

8.28.2.6 chargeMap

`Vgrid* chargeMap`
External charge distribution map
Definition at line 188 of file vpmg.h.

8.28.2.7 chargeMeth

`Vchrg_Meth chargeMeth`
Charge discretization method
Definition at line 165 of file vpmg.h.

8.28.2.8 chargeSrc

`Vchrg_Src chargeSrc`
Charge source
Definition at line 166 of file vpmg.h.

8.28.2.9 dielXMap

`Vgrid* dielXMap`
External x-shifted dielectric map
Definition at line 172 of file vpmg.h.

8.28.2.10 dielYMap

`Vgrid* dielYMap`
External y-shifted dielectric map
Definition at line 175 of file vpmg.h.

8.28.2.11 dielZMap

`Vgrid* dielZMap`
External z-shifted dielectric map
Definition at line 178 of file vpmg.h.

8.28.2.12 epsx

`double* epsx`
X-shifted dielectric map
Definition at line 127 of file vpmg.h.

8.28.2.13 epsy

`double* epsy`
Y-shifted dielectric map
Definition at line 128 of file vpmg.h.

8.28.2.14 epsz

`double* epsz`
Y-shifted dielectric map
Definition at line 129 of file vpmg.h.

8.28.2.15 extDiEnergy

`double extDiEnergy`
Stores contributions to the dielectric energy from regions outside the problem domain
Definition at line 155 of file vpmg.h.

8.28.2.16 extNpEnergy

`double extNpEnergy`
Stores contributions to the apolar energy from regions outside the problem domain
Definition at line 161 of file vpmg.h.

8.28.2.17 extQfEnergy

`double extQfEnergy`
Stores contributions to the fixed charge energy from regions outside the problem domain
Definition at line 159 of file vpmg.h.

8.28.2.18 extQmEnergy

`double extQmEnergy`
Stores contributions to the mobile ion energy from regions outside the problem domain
Definition at line 157 of file vpmg.h.

8.28.2.19 fcf

`double* fcf`
Right-hand side – this array can be overwritten
Definition at line 145 of file vpmg.h.

8.28.2.20 filled

`int filled`
Indicates whether Vpmg_fillco has been called
Definition at line 168 of file vpmg.h.

8.28.2.21 gxcf

`double* gxcf`
Boundary conditions for x faces
Definition at line 151 of file vpmg.h.

8.28.2.22 gycf

`double* gycf`
Boundary conditions for y faces
Definition at line 152 of file vpmg.h.

8.28.2.23 gzcf

`double* gzcf`
Boundary conditions for z faces
Definition at line 153 of file vpmg.h.

8.28.2.24 iparm

`int* iparm`
Passing int parameters to FORTRAN
Definition at line 134 of file vpmg.h.

8.28.2.25 iwork

`int* iwork`
Work array
Definition at line 136 of file vpmg.h.

8.28.2.26 kappa

`double* kappa`
Ion accessibility map (0 <= kappa(x) <= 1)
Definition at line 130 of file vpmg.h.

8.28.2.27 kappaMap

`Vgrid* kappaMap`
External kappa map
Definition at line 181 of file vpmg.h.

8.28.2.28 pbe

```
Vpbe* pbe
```

Information about the PBE system
Definition at line 120 of file vpmg.h.

8.28.2.29 pmgp

```
Vpmgp* pmgp
```

Parameters
Definition at line 119 of file vpmg.h.

8.28.2.30 pot

```
double* pot
```

Potential map
Definition at line 131 of file vpmg.h.

8.28.2.31 potMap

```
Vgrid* potMap
```

External potential map
Definition at line 184 of file vpmg.h.

8.28.2.32 pvec

```
double* pvec
```

Partition mask array
Definition at line 154 of file vpmg.h.

8.28.2.33 rparm

```
double* rparm
```

Passing real parameters to FORTRAN
Definition at line 135 of file vpmg.h.

8.28.2.34 rwork

```
double* rwork
```

Work array
Definition at line 137 of file vpmg.h.

8.28.2.35 splineWin

```
double splineWin
```

Spline window parm for surf defs
Definition at line 164 of file vpmg.h.

8.28.2.36 surfMeth

```
Vsurf_Meth surfMeth
```

Surface definition method
Definition at line 163 of file vpmg.h.

8.28.2.37 tcf

```
double* tcf
```

True solution
Definition at line 146 of file vpmg.h.

8.28.2.38 u

```
double* u
```

Solution
Definition at line 147 of file vpmg.h.

8.28.2.39 useChargeMap

```
int useChargeMap
```

Indicates whether Vpmg_fillco was called with an external charge distribution map
Definition at line 186 of file vpmg.h.

8.28.2.40 useDielXMap

```
int useDielXMap
```

Indicates whether Vpmg_fillco was called with an external x-shifted dielectric map
Definition at line 170 of file vpmg.h.

8.28.2.41 useDielYMap

```
int useDielYMap
```

Indicates whether Vpmg_fillco was called with an external y-shifted dielectric map
Definition at line 173 of file vpmg.h.

8.28.2.42 useDielZMap

```
int useDielZMap
```

Indicates whether Vpmg_fillco was called with an external z-shifted dielectric map
Definition at line 176 of file vpmg.h.

8.28.2.43 useKappaMap

```
int useKappaMap
```

Indicates whether Vpmg_fillco was called with an external kappa map
Definition at line 179 of file vpmg.h.

8.28.2.44 usePotMap

```
int usePotMap
```

Indicates whether Vpmg_fillco was called with an external potential map
Definition at line 182 of file vpmg.h.

8.28.2.45 vmem

```
Vmem* vmem
```

Memory management object for this class
Definition at line 118 of file vpmg.h.

8.28.2.46 xf

```
double* xf
```

Mesh point x coordinates
Definition at line 148 of file vpmg.h.

8.28.2.47 yf

```
double* yf
```

Mesh point y coordinates
Definition at line 149 of file vpmg.h.

8.28.2.48 zf

```
double* zf
```

Mesh point z coordinates
Definition at line 150 of file vpmg.h.
The documentation for this struct was generated from the following file:

- src/mg/vpmg.h

8.29 sVpmgp Struct Reference

Contains public data members for Vpmgp class/module.
`#include </builddir/build/BUILD/apbs-3.0.0/src/mg/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
- size_t nrwk
- int niwk
- int narr
- int ipkey
- double xcent
- double ycent
- double zcent
- double errtol
- int itmax
- int istop
- int iinfo
- Vbcfl bcfl
- int key
- int iperf
- int meth
- int mgkey
- int nu1
- int nu2
- int mgsmoo
- int mgprol
- int mgcoar
- int mgsolv
- int mgdisc
- double omegal
- double omegan
- int irite
- int ipcon
- double xlen
- double ylen
- double zlen
- double xmin
- double ymin
- double zmin
- double xmax
- double ymax
- double zmax

8.29.1 Detailed Description

Contains public data members for Vpmgp class/module.

Author

Nathan Baker

Bug Value ipcon does not currently allow for preconditioning in PMG

Definition at line 80 of file vpmgp.h.

8.29.2 Field Documentation

8.29.2.1 bcfl

`Vbcfl bcfl`
Boundary condition method [default = BCFL_SDH]
Definition at line 135 of file vpmgp.h.

8.29.2.2 errtol

`double errtol`
Desired error tolerance [default = 1e-9]
Definition at line 121 of file vpmgp.h.

8.29.2.3 hx

`double hx`
Grid x spacings [no default]

Definition at line 87 of file vpmgp.h.

8.29.2.4 hy

`double hy`
Grid y spacings [no default]

Definition at line 88 of file vpmgp.h.

8.29.2.5 hzed

`double hzed`
Grid z spacings [no default]

Definition at line 89 of file vpmgp.h.

8.29.2.6 iinfo

`int iinfo`
Runtime status messages [default = 1]

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

Definition at line 130 of file vpmgp.h.

8.29.2.7 ipcon

`int ipcon`
Preconditioning method [default = 3]

- 0: diagonal
- 1: ICCG
- 2: ICCGDW
- 3: MICCGDW
- 4: none

Definition at line 183 of file vpmgp.h.

8.29.2.8 iperf

`int iperf`
Analysis of the operator [default = 0]

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

Definition at line 139 of file vpmgp.h.

8.29.2.9 ipkey

`int ipkey`
Toggles nonlinearity (set by nonlin)

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

Definition at line 109 of file vpmgp.h.

8.29.2.10 irite

`int irite`
FORTRAN output unit [default = 8]
Definition at line 182 of file vpmgp.h.

8.29.2.11 istop

`int istop`
Stopping criterion [default = 1]

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

Definition at line 123 of file vpmgp.h.

8.29.2.12 itmax

`int itmax`
Maximum number of iters [default = 100]
Definition at line 122 of file vpmgp.h.

8.29.2.13 key

`int key`
Print solution to file [default = 0]

- 0: no
- 1: yes

Definition at line 136 of file vpmgp.h.

8.29.2.14 meth

`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 144 of file vpmgp.h.

8.29.2.15 mgcoar

`int mgcoar`
Coarsening method [default = 2]

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

Definition at line 170 of file vpmgp.h.

8.29.2.16 mgdisc

`int mgdisc`
Discretization method [default = 0]

- 0: finite volume
- 1: finite element

Definition at line 177 of file vpmgp.h.

8.29.2.17 mgkey

`int mgkey`
Multigrid method [default = 0]

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

Definition at line 155 of file vpmgp.h.

8.29.2.18 mgprol

`int mgprol`
Prolongation method [default = 0]

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

Definition at line 166 of file vpmgp.h.

8.29.2.19 mgsmoo

`int mgsmoo`
Smoothing method [default = 1]

- 0: weighted jacobi
- 1: gauss-seidel

- 2: SOR
- 3: richardson
- 4: cghs

Definition at line 160 of file vpmgp.h.

8.29.2.20 mgsolv

`int mgsolv`
Coarse equation solve method [default = 1]

- 0: cghs
- 1: banded linpack

Definition at line 174 of file vpmgp.h.

8.29.2.21 n_ipc

`int n_ipc`
Integer info work array required storage
Definition at line 104 of file vpmgp.h.

8.29.2.22 n_iz

`int n_iz`
Integer storage parameter (index max)
Definition at line 103 of file vpmgp.h.

8.29.2.23 n_rpc

`int n_rpc`
Real info work array required storage
Definition at line 102 of file vpmgp.h.

8.29.2.24 narr

`int narr`
Array work storage
Definition at line 108 of file vpmgp.h.

8.29.2.25 narrc

`int narrc`
Size of vector on coarse level
Definition at line 101 of file vpmgp.h.

8.29.2.26 nc

`int nc`
Number of coarse grid unknowns
Definition at line 100 of file vpmgp.h.

8.29.2.27 nf

`int nf`
Number of fine grid unknowns
Definition at line 99 of file vpmgp.h.

8.29.2.28 niwk

`int niwk`
Integer work storage
Definition at line 107 of file vpmgp.h.

8.29.2.29 nlev

`int nlev`
Number of mesh levels [no default]
Definition at line 86 of file vpmgp.h.

8.29.2.30 nonlin

`int nonlin`
Problem type [no default]

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

Definition at line 90 of file vpmgp.h.

8.29.2.31 nrwk

`size_t nrwk`
Real work storage
Definition at line 106 of file vpmgp.h.

8.29.2.32 nu1

`int nu1`
Number of pre-smoothings [default = 2]
Definition at line 158 of file vpmgp.h.

8.29.2.33 nu2

`int nu2`
Number of post-smoothings [default = 2]
Definition at line 159 of file vpmgp.h.

8.29.2.34 nx

`int nx`
Grid x dimensions [no default]

Definition at line 83 of file vpmgp.h.

8.29.2.35 nxc

`int nxc`
Coarse level grid x dimensions
Definition at line 96 of file vpmgp.h.

8.29.2.36 ny

`int ny`
Grid y dimensions [no default]

Definition at line 84 of file vpmgp.h.

8.29.2.37 nyc

`int nyc`
Coarse level grid y dimensions
Definition at line 97 of file vpmgp.h.

8.29.2.38 nz

`int nz`
Grid z dimensions [no default]

Definition at line 85 of file vpmgp.h.

8.29.2.39 nzc

`int nzc`
Coarse level grid z dimensions
Definition at line 98 of file vpmgp.h.

8.29.2.40 omegal

`double omegal`
Linear relax parameter [default = 8e-1]

Definition at line 180 of file vpmgp.h.

8.29.2.41 omegan

`double omegan`
Nonlin relax parameter [default = 9e-1]
Definition at line 181 of file vpmgp.h.

8.29.2.42 xcent

`double xcent`
Grid x center [0]

Definition at line 118 of file vpmgp.h.

8.29.2.43 xlen

`double xlen`
Domain x length
Definition at line 189 of file vpmgp.h.

8.29.2.44 xmax

`double xmax`
Domain upper x corner
Definition at line 195 of file vpmgp.h.

8.29.2.45 xmin

`double xmin`
Domain lower x corner
Definition at line 192 of file vpmgp.h.

8.29.2.46 ycent

`double ycent`
Grid y center [0]

Definition at line 119 of file vpmgp.h.

8.29.2.47 ylen

`double ylen`
Domain y length
Definition at line 190 of file vpmgp.h.

8.29.2.48 ymax

`double ymax`
Domain upper y corner
Definition at line 196 of file vpmgp.h.

8.29.2.49 ymin

`double ymin`
Domain lower y corner
Definition at line 193 of file vpmgp.h.

8.29.2.50 zcent

`double zcent`
Grid z center [0]

Definition at line 120 of file vpmgp.h.

8.29.2.51 zlen

`double zlen`
Domain z length
Definition at line 191 of file vpmgp.h.

8.29.2.52 zmax

`double zmax`
Domain upper z corner
Definition at line 197 of file vpmgp.h.

8.29.2.53 zmin

`double zmin`
Domain lower z corner
Definition at line 194 of file vpmgp.h.
The documentation for this struct was generated from the following file:

- src/mg/vpmgp.h

8.30 Vparam Struct Reference

Reads and assigns charge/radii parameters.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/vparam.h>`

Data Fields

- Vmem * vmem
- int nResData
- Vparam_ResData * resData

8.30.1 Detailed Description

Reads and assigns charge/radii parameters.

Author

Nathan Baker

Definition at line 135 of file vparam.h.

8.30.2 Field Documentation

8.30.2.1 nResData

`int nResData`
Number of Vparam_ResData objects associated with this object
Definition at line 138 of file vparam.h.

8.30.2.2 resData

`Vparam_ResData* resData`
Array of nResData Vparam_ResData objects
Definition at line 140 of file vparam.h.

8.30.2.3 vmem

`Vmem* vmem`
Memory management object for this class
Definition at line 137 of file vparam.h.
The documentation for this struct was generated from the following file:

- src/generic/vparam.h

8.31 Vparam_ResData Struct Reference

ResData sub-class; stores residue data.
`#include </builddir/build/BUILD/apbs-3.0.0/src/generic/vparam.h>`

Data Fields

- Vmem * vmem
- char name [VMAX_ARGLEN]
- int nAtomData
- Vparam_AtomData * atomData

8.31.1 Detailed Description

ResData sub-class; stores residue data.

Author

Nathan Baker

Definition at line 114 of file vparam.h.

8.31.2 Field Documentation

8.31.2.1 atomData

`Vparam_AtomData* atomData`
Array of Vparam_AtomData natom objects
Definition at line 119 of file vparam.h.

8.31.2.2 name

`char name[VMAX_ARGLEN]`
Residue name
Definition at line 116 of file vparam.h.

8.31.2.3 nAtomData

`int nAtomData`
Number of Vparam_AtomData objects associated with this object
Definition at line 117 of file vparam.h.

8.31.2.4 vmem

`Vmem* vmem`
Pointer to memory manager from Vparam master class
Definition at line 115 of file vparam.h.
The documentation for this struct was generated from the following file:

- src/generic/vparam.h

Chapter 9

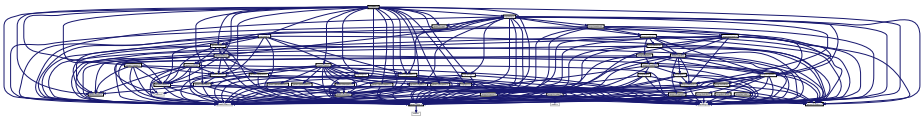
File Documentation

9.1 src/apbs.h File Reference

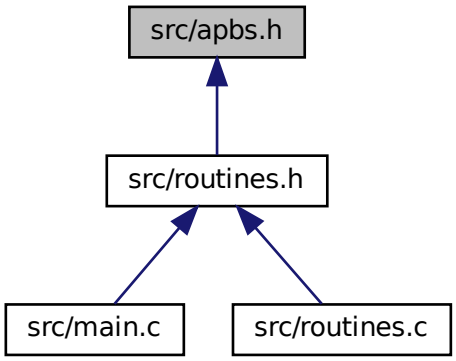
Header file for header dependencies.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/nosh.h"
#include "generic/mgparm.h"
#include "generic/pbeparm.h"
#include "generic/femparm.h"
#include "generic/bemparm.h"
#include "generic/geoflowparm.h"
#include "generic/vacc.h"
#include "generic/valist.h"
#include "generic/vatom.h"
#include "generic/vcap.h"
#include "generic/vhal.h"
#include "generic/vpbe.h"
#include "generic/vstring.h"
#include "generic/vunit.h"
#include "generic/vparam.h"
#include "generic/vgreen.h"
#include "mg/vgrid.h"
#include "mg/vmgrid.h"
#include "mg/vopot.h"
#include "mg/vpmg.h"
#include "mg/vpmgp.h"
```

Include dependency graph for apbs.h:



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



9.1.1 Detailed Description

Header file for header dependencies.

Author

Nathan Baker

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
```

```
 * Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * Neither the name of the developer nor the names of its contributors may be
 * used to endorse or promote products derived from this software without
 * specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 *
 *
```

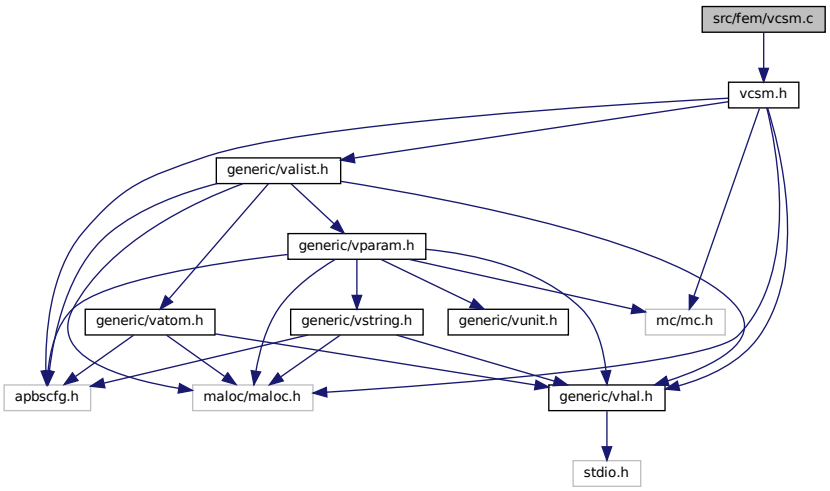
Definition in file apbs.h.

9.2 apbs.h

```
00001
00061 #ifndef _APBSHEADERS_H_
00062 #define _APBSHEADERS_H_
00063
00064 #include "apbscfg.h"
00065
00066 /* MALOC headers */
00067 #include "maloc/maloc.h"
00068
00069 /* Generic headers */
00070 #include "generic/nosh.h"
00071 #include "generic/mgparm.h"
00072 #include "generic/pbeparm.h"
00073 #include "generic/femparm.h"
00074 #include "generic/bemparm.h"
00075 #include "generic/geoflowparm.h"
00076 #include "generic/vacc.h"
00077 #include "generic/valist.h"
00078 #include "generic/vatom.h"
00079 #include "generic/vcap.h"
00080 #include "generic/vhal.h"
00081 #include "generic/vpbe.h"
00082 #include "generic/vstring.h"
00083 #include "generic/vunit.h"
00084 #include "generic/vparam.h"
00085 #include "generic/vgreen.h"
00086
00087 //#include "geoflow/cpbconcz2.h"
00088
00089 /* MG headers */
00090 #include "mg/vgrid.h"
00091 #include "mg/vmgrid.h"
00092 #include "mg/vopot.h"
00093 #include "mg/vpmg.h"
00094 #include "mg/vpmgp.h"
00095
00096 /* FEM headers */
00097 #if defined(FETK_ENABLED)
00098     #include "fem/vfetk.h"
00099     #include "fem/vpee.h"
00100 #endif
00101
00102 #endif /* _APBSHEADERS_H_ */
```

9.3 src/fem/vcsm.c File Reference

Class Vcsm methods.
`#include "vcsm.h"`
Include dependency graph for vcsm.c:



Functions

- VPUBLIC Valist ∗ Vcsm_getValist (Vcsm ∗thee)

  Get atom list.
- VPUBLIC int Vcsm_getNumberAtoms (Vcsm ∗thee, int isimp)

  Get number of atoms associated with a simplex.
- VPUBLIC Vatom ∗ Vcsm_getAtom (Vcsm ∗thee, int iatom, int isimp)

  Get particular atom associated with a simplex.
- VPUBLIC int Vcsm_getAtomIndex (Vcsm ∗thee, int iatom, int isimp)

  Get ID of particular atom in a simplex.
- VPUBLIC int Vcsm_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.

9.3.1 Detailed Description

Class Vcsm methods.

Author

Nathan Baker

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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) 2002-2020, 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.

9.4 vcsm.c

```
00001
00057 #include "vcsm.h"
00058
00059 /* Inlineable methods */
00060 #if !defined(VINLINE_VCSM)
00061
00062 VPUBLIC Valist* Vcsm_getValist(Vcsm *thee) {
00063
00064    VASSERT(thee != VNULL);
00065    return thee->alist;
00066
00067 }
00068
00069 VPUBLIC int Vcsm_getNumberAtoms(Vcsm *thee, int isimp) {
00070
00071    VASSERT(thee != VNULL);
00072    VASSERT(thee->initFlag);
00073    return thee->nsqm[isimp];
00074
00075 }
00076
00077 VPUBLIC Vatom* Vcsm_getAtom(Vcsm *thee, int iatom, int isimp) {
00078
00079
00080    VASSERT(thee != VNULL);
00081    VASSERT(thee->initFlag);
00082
00083    VASSERT(iatom < (thee->nsqm)[isimp]);
00084    return Valist_getAtom(thee->alist, (thee->sqm)[isimp][iatom]);
00085
00086 }
00087
00088 VPUBLIC int Vcsm_getAtomIndex(Vcsm *thee, int iatom, int isimp) {
00089
00090
00091    VASSERT(thee != VNULL);
00092    VASSERT(thee->initFlag);
00093
00094    VASSERT(iatom < (thee->nsqm)[isimp]);
00095    return (thee->sqm)[isimp][iatom];
00096
00097 }
00098
00099 VPUBLIC int Vcsm_getNumberSimplices(Vcsm *thee, int iatom) {
00100
00101
00102    VASSERT(thee != VNULL);
00103    VASSERT(thee->initFlag);
00104
00105    return (thee->nqsm)[iatom];
00106
00107 }
00108
00109 VPUBLIC SS* Vcsm_getSimplex(Vcsm *thee, int isimp, int iatom) {
00110
00111
00112    VASSERT(thee != VNULL);
00113    VASSERT(thee->initFlag);
00114
00115    return Gem_SS(thee->gm, (thee->qsm)[iatom][isimp]);
00116
00117 }
00118
00119 VPUBLIC int Vcsm_getSimplexIndex(Vcsm *thee, int isimp, int iatom) {
00120
00121
00122    VASSERT(thee != VNULL);
00123    VASSERT(thee->initFlag);
00124
```

```
00125    return (thee->qsm)[iatom][isimp];
00126
00127 }
00128
00129 VPUBLIC unsigned long int Vcsm_memChk(Vcsm *thee) {
00130     if (thee == VNULL) return 0;
00131     return Vmem_bytes(thee->vmem);
00132 }
00133
00134 #endif /* if !defined(VINLINE_VCSM) */
00135
00136 VPUBLIC Vcsm* Vcsm_ctor(Valist *alist, Gem *gm) {
00137
00138     /* Set up the structure */
00139     Vcsm *thee = VNULL;
00140     thee = (Vcsm*)Vmem_malloc(VNULL, 1, sizeof(Vcsm) );
00141     VASSERT( thee != VNULL);
00142     VASSERT( Vcsm_ctor2(thee, alist, gm));
00143
00144     return thee;
00145 }
00146
00147 VPUBLIC int Vcsm_ctor2(Vcsm *thee, Valist *alist, Gem *gm) {
00148
00149     VASSERT( thee != VNULL );
00150
00151     /* Memory management object */
00152     thee->vmem = Vmem_ctor("APBS:VCSM");
00153
00154     /* Set up the atom list and grid manager */
00155     if( alist == VNULL) {
00156         Vnm_print(2,"Vcsm_ctor2: got null pointer to Valist object!\n");
00157         return 0;
00158     }
00159     thee->alist = alist;
00160     if( gm == VNULL) {
00161         Vnm_print(2,"Vcsm_ctor2: got a null pointer to the Gem object!\n");
00162         return 0;
00163     }
00164     thee->gm = gm;
00165
00166     thee->initFlag = 0;
00167     return 1;
00168 }
00169
00170 VPUBLIC void Vcsm_init(Vcsm *thee) {
00171
00172     /* Counters */
00173     int iatom,
00174         jatom,
00175         isimp,
00176         jsimp,
00177         gotSimp;
00178     /* Atomic information */
00179     Vatom *atom;
00180     double *position;
00181     /* Simplex/Vertex information */
00182     SS *simplex;
00183     /* Basis function values */
00184
00185     if (thee == VNULL) {
00186         Vnm_print(2, "Vcsm_init:  Error!  Got NULL thee!\n");
00187         VASSERT(0);
00188     }
00189     if (thee->gm == VNULL) {
00190         Vnm_print(2, "Vcsm_init:  Error!  Got NULL thee->gm!\n");
00191         VASSERT(0);
00192     }
00193     thee->nsimp = Gem_numSS(thee->gm);
00194     if (thee->nsimp <= 0) {
00195         Vnm_print(2, "Vcsm_init:  Error!  Got %d simplices!\n", thee->nsimp);
00196         VASSERT(0);
00197     }
00198     thee->natom = Valist_getNumberAtoms(thee->alist);
00199
00200     /* Allocate and initialize space for the first dimensions of the
00201      * simplex-charge map, the simplex array, and the counters */
00202     thee->sqm = (int**)Vmem_malloc(thee->vmem, thee->nsimp, sizeof(int *));
00203     VASSERT(thee->sqm != VNULL);
00204     thee->nsqm = (int*)Vmem_malloc(thee->vmem, thee->nsimp, sizeof(int));
00205     VASSERT(thee->nsqm != VNULL);
```

```
00206     for (isimp=0; isimp<thee->nsimp; isimp++) (thee->nsqm)[isimp] = 0;
00207
00208     /* Count the number of charges per simplex. */
00209     for (iatom=0; iatom<thee->natom; iatom++) {
00210         atom = Valist_getAtom(thee->alist, iatom);
00211         position = Vatom_getPosition(atom);
00212         gotSimp = 0;
00213         for (isimp=0; isimp<thee->nsimp; isimp++) {
00214             simplex = Gem_SS(thee->gm, isimp);
00215             if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00216                 (thee->nsqm)[isimp]++;
00217                 gotSimp = 1;
00218              }
00219         }
00220     }
00221
00222     /* @todo Combine the following two loops? - PCE */
00223     /* Allocate the space for the simplex-charge map */
00224     for (isimp=0; isimp<thee->nsimp; isimp++) {
00225         if ((thee->nsqm)[isimp] > 0) {
00226             thee->sqm[isimp] = (int*)Vmem_malloc(thee->vmem, (thee->nsqm)[isimp],
00227               sizeof(int));
00228             VASSERT(thee->sqm[isimp] != VNULL);
00229         }
00230     }
00231
00232     /* Finally, set up the map */
00233     for (isimp=0; isimp<thee->nsimp; isimp++) {
00234         jsimp = 0;
00235         simplex = Gem_SS(thee->gm, isimp);
00236         for (iatom=0; iatom<thee->natom; iatom++) {
00237             atom = Valist_getAtom(thee->alist, iatom);
00238             position = Vatom_getPosition(atom);
00239             /* Check to see if the atom's in this simplex */
00240             if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00241                 /* Assign the entries in the next vacant spot */
00242                 (thee->sqm)[isimp][jsimp] = iatom;
00243                 jsimp++;
00244             }
00245         }
00246     }
00247
00248     thee->msimp = thee->nsimp;
00249
00250     /* Allocate space for the charge-simplex map */
00251     thee->qsm = (int**)Vmem_malloc(thee->vmem, thee->natom, sizeof(int *));
00252     VASSERT(thee->qsm != VNULL);
00253     thee->nqsm = (int*)Vmem_malloc(thee->vmem, thee->natom, sizeof(int));
00254     VASSERT(thee->nqsm != VNULL);
00255     for (iatom=0; iatom<thee->natom; iatom++) (thee->nqsm)[iatom] = 0;
00256     /* Loop through the list of simplices and count the number of times
00257      * each atom appears */
00258     for (isimp=0; isimp<thee->nsimp; isimp++) {
00259         for (iatom=0; iatom<thee->nsqm[isimp]; iatom++) {
00260             jatom = thee->sqm[isimp][iatom];
00261             thee->nqsm[jatom]++;
00262         }
00263     }
00264     /* Do a TIME-CONSUMING SANITY CHECK to make sure that each atom was
00265      * placed in at simplex */
00266     for (iatom=0; iatom<thee->natom; iatom++) {
00267         if (thee->nqsm[iatom] == 0) {
00268             Vnm_print(2, "Vcsm_init: Atom %d not placed in simplex!\n", iatom);
00269             VASSERT(0);
00270         }
00271     }
00272     /* Allocate the appropriate amount of space for each entry in the
00273      * charge-simplex map and clear the counter for re-use in assignment */
00274     for (iatom=0; iatom<thee->natom; iatom++) {
00275         thee->qsm[iatom] = (int*)Vmem_malloc(thee->vmem, (thee->nqsm)[iatom],
00276           sizeof(int));
00277         VASSERT(thee->qsm[iatom] != VNULL);
00278         thee->nqsm[iatom] = 0;
00279     }
00280     /* Assign the simplices to atoms */
00281     for (isimp=0; isimp<thee->nsimp; isimp++) {
00282         for (iatom=0; iatom<thee->nsqm[isimp]; iatom++) {
00283             jatom = thee->sqm[isimp][iatom];
00284             thee->qsm[jatom][thee->nqsm[jatom]] = isimp;
00285             thee->nqsm[jatom]++;
00286         }
```

```
00287     }
00288
00289     thee->initFlag = 1;
00290 }
00291
00292 VPUBLIC void Vcsm_dtor(Vcsm **thee) {
00293     if ((*thee) != VNULL) {
00294         Vcsm_dtor2(*thee);
00295         Vmem_free(VNULL, 1, sizeof(Vcsm), (void **)thee);
00296         (*thee) = VNULL;
00297     }
00298 }
00299
00300 VPUBLIC void Vcsm_dtor2(Vcsm *thee) {
00301     int i;
00302
00303     if ((thee != VNULL) && thee->initFlag) {
00304
00305         for (i=0; i<thee->msimp; i++) {
00306             if (thee->nsqm[i] > 0) Vmem_free(thee->vmem, thee->nsqm[i],
00307               sizeof(int), (void **)&(thee->sqm[i]));
00308         }
00309         for (i=0; i<thee->natom; i++) {
00310             if (thee->nqsm[i] > 0) Vmem_free(thee->vmem, thee->nqsm[i],
00311               sizeof(int), (void **)&(thee->qsm[i]));
00312         }
00313         Vmem_free(thee->vmem, thee->msimp, sizeof(int *),
00314           (void **)&(thee->sqm));
00315         Vmem_free(thee->vmem, thee->msimp, sizeof(int),
00316           (void **)&(thee->nsqm));
00317         Vmem_free(thee->vmem, thee->natom, sizeof(int *),
00318           (void **)&(thee->qsm));
00319         Vmem_free(thee->vmem, thee->natom, sizeof(int),
00320           (void **)&(thee->nqsm));
00321
00322     }
00323     Vmem_dtor(&(thee->vmem));
00324 }
00325
00326 VPUBLIC int Vcsm_update(Vcsm *thee, SS **simps, int num) {
00327
00328     /* Counters */
00329     int isimp, jsimp, iatom, jatom, atomID, simpID;
00330     int nsimps, gotMem;
00331     /* Object info */
00332     Vatom *atom;
00333     SS *simplex;
00334     double *position;
00335     /* Lists */
00336     int *qParent, nqParent;
00337     int **sqmNew, *nsqmNew;
00338     int *affAtoms, nAffAtoms;
00339     int *dnqsm, *nqsmNew, **qsmNew;
00340
00341     VASSERT(thee != VNULL);
00342     VASSERT(thee->initFlag);
00343
00344     /* If we don't have enough memory to accommodate the new entries,
00345      * add more by doubling the existing amount */
00346     isimp = thee->nsimp + num - 1;
00347     gotMem = 0;
00348     while (!gotMem) {
00349         if (isimp > thee->msimp) {
00350             isimp = 2 * isimp;
00351             thee->nsqm = (int*)Vmem_realloc(thee->vmem, thee->msimp, sizeof(int),
00352               (void **)&(thee->nsqm), isimp);
00353             VASSERT(thee->nsqm != VNULL);
00354             thee->sqm = (int**)Vmem_realloc(thee->vmem, thee->msimp, sizeof(int *),
00355               (void **)&(thee->sqm), isimp);
00356             VASSERT(thee->sqm != VNULL);
00357             thee->msimp = isimp;
00358         } else gotMem = 1;
00359     }
00360     /* Initialize the nsqm entires we just allocated */
00361     for (isimp = thee->nsimp; isimp<thee->nsimp+num-1 ; isimp++) {
00362        thee->nsqm[isimp] = 0;
00363     }
00364
00365     thee->nsimp = thee->nsimp + num - 1;
00366
00367     /* There's a simple case to deal with:  if simps[0] didn't have a
```

```
00368      * charge in the first place */
00369     isimp = SS_id(simps[0]);
00370     if (thee->nsqm[isimp] == 0) {
00371         for (isimp=1; isimp<num; isimp++) {
00372             thee->nsqm[SS_id(simps[isimp])] = 0;
00373         }
00374         return 1;
00375     }
00376
00377     /* The more complicated case has occured; the parent simplex had one or
00378      * more charges.  First, generate the list of affected charges. */
00379     isimp = SS_id(simps[0]);
00380     nqParent = thee->nsqm[isimp];
00381     qParent = thee->sqm[isimp];
00382
00383     sqmNew = (int**)Vmem_malloc(thee->vmem, num, sizeof(int *));
00384     VASSERT(sqmNew != VNULL);
00385     nsqmNew = (int*)Vmem_malloc(thee->vmem, num, sizeof(int));
00386     VASSERT(nsqmNew != VNULL);
00387     for (isimp=0; isimp<num; isimp++) nsqmNew[isimp] = 0;
00388
00389     /* Loop throught the affected atoms to determine how many atoms each
00390      * simplex will get. */
00391     for (iatom=0; iatom<nqParent; iatom++) {
00392
00393         atomID = qParent[iatom];
00394         atom = Valist_getAtom(thee->alist, atomID);
00395         position = Vatom_getPosition(atom);
00396         nsimps = 0;
00397
00398         jsimp = 0;
00399
00400         for (isimp=0; isimp<num; isimp++) {
00401             simplex = simps[isimp];
00402             if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00403                 nsqmNew[isimp]++;
00404                 jsimp = 1;
00405             }
00406         }
00407
00408         VASSERT(jsimp != 0);
00409     }
00410
00411     /* Sanity check that we didn't lose any atoms... */
00412     iatom = 0;
00413     for (isimp=0; isimp<num; isimp++) iatom += nsqmNew[isimp];
00414     if (iatom < nqParent) {
00415         Vnm_print(2,"Vcsm_update: Lost %d (of %d) atoms!\n",
00416             nqParent - iatom, nqParent);
00417         VASSERT(0);
00418     }
00419
00420     /* Allocate the storage */
00421     for (isimp=0; isimp<num; isimp++) {
00422         if (nsqmNew[isimp] > 0) {
00423             sqmNew[isimp] = (int*)Vmem_malloc(thee->vmem, nsqmNew[isimp],
00424               sizeof(int));
00425             VASSERT(sqmNew[isimp] != VNULL);
00426         }
00427     }
00428
00429     /* Assign charges to simplices */
00430     for (isimp=0; isimp<num; isimp++) {
00431
00432         jsimp = 0;
00433         simplex = simps[isimp];
00434
00435         /* Loop over the atoms associated with the parent simplex */
00436         for (iatom=0; iatom<nqParent; iatom++) {
00437
00438             atomID = qParent[iatom];
00439             atom = Valist_getAtom(thee->alist, atomID);
00440             position = Vatom_getPosition(atom);
00441             if (Gem_pointInSimplex(thee->gm, simplex, position)) {
00442                 sqmNew[isimp][jsimp] = atomID;
00443                 jsimp++;
00444             }
00445         }
00446     }
00447
00448     /* Update the QSM map using the old and new SQM lists */
```

```
00449     /* The affected atoms are those contained in the parent simplex; i.e.
00450      * thee->sqm[SS_id(simps[0])] */
00451     affAtoms = thee->sqm[SS_id(simps[0])];
00452     nAffAtoms = thee->nsqm[SS_id(simps[0])];
00453     /* Each of these atoms will go somewhere else; i.e., the entries in
00454      * thee->qsm are never destroyed and thee->nqsm never decreases.
00455      * However, it is possible that a subdivision could cause an atom to be
00456      * shared by two child simplices.  Here we record the change, if any,
00457      * in the number of simplices associated with each atom. */
00458     dnqsm = (int*)Vmem_malloc(thee->vmem, nAffAtoms, sizeof(int));
00459     VASSERT(dnqsm != VNULL);
00460     nqsmNew = (int*)Vmem_malloc(thee->vmem, nAffAtoms, sizeof(int));
00461     VASSERT(nqsmNew != VNULL);
00462     qsmNew = (int**)Vmem_malloc(thee->vmem, nAffAtoms, sizeof(int*));
00463     VASSERT(qsmNew != VNULL);
00464     for (iatom=0; iatom<nAffAtoms; iatom++) {
00465         dnqsm[iatom] = -1;
00466         atomID = affAtoms[iatom];
00467         for (isimp=0; isimp<num; isimp++) {
00468             for (jatom=0; jatom<nsqmNew[isimp]; jatom++) {
00469                 if (sqmNew[isimp][jatom] == atomID) dnqsm[iatom]++;
00470             }
00471         }
00472         VASSERT(dnqsm[iatom] > -1);
00473     }
00474     /* Setup the new entries in the array */
00475     for (iatom=0;iatom<nAffAtoms; iatom++) {
00476         atomID = affAtoms[iatom];
00477         qsmNew[iatom] = (int*)Vmem_malloc(thee->vmem,
00478                 (dnqsm[iatom] + thee->nqsm[atomID]),
00479                 sizeof(int));
00480         nqsmNew[iatom] = 0;
00481         VASSERT(qsmNew[iatom] != VNULL);
00482     }
00483     /* Fill the new entries in the array */
00484     /* First, do the modified entries */
00485     for (isimp=0; isimp<num; isimp++) {
00486         simpID = SS_id(simps[isimp]);
00487         for (iatom=0; iatom<nsqmNew[isimp]; iatom++) {
00488             atomID = sqmNew[isimp][iatom];
00489             for (jatom=0; jatom<nAffAtoms; jatom++) {
00490                 if (atomID == affAtoms[jatom]) break;
00491             }
00492             if (jatom < nAffAtoms) {
00493                 qsmNew[jatom][nqsmNew[jatom]] = simpID;
00494                 nqsmNew[jatom]++;
00495             }
00496         }
00497     }
00498     /* Now do the unmodified entries */
00499     for (iatom=0; iatom<nAffAtoms; iatom++) {
00500         atomID = affAtoms[iatom];
00501         for (isimp=0; isimp<thee->nqsm[atomID]; isimp++) {
00502             for (jsimp=0; jsimp<num; jsimp++) {
00503                 simpID = SS_id(simps[jsimp]);
00504                 if (thee->qsm[atomID][isimp] == simpID) break;
00505             }
00506             if (jsimp == num) {
00507                 qsmNew[iatom][nqsmNew[iatom]] = thee->qsm[atomID][isimp];
00508                 nqsmNew[iatom]++;
00509             }
00510         }
00511     }
00512
00513     /* Replace the existing entries in the table.  Do the QSM entires
00514      * first, since they require affAtoms = thee->sqm[simps[0]] */
00515     for (iatom=0; iatom<nAffAtoms; iatom++) {
00516         atomID = affAtoms[iatom];
00517         Vmem_free(thee->vmem, thee->nqsm[atomID], sizeof(int),
00518           (void **)&(thee->qsm[atomID]));
00519         thee->qsm[atomID] = qsmNew[iatom];
00520         thee->nqsm[atomID] = nqsmNew[iatom];
00521     }
00522     for (isimp=0; isimp<num; isimp++) {
00523         simpID = SS_id(simps[isimp]);
00524         if (thee->nsqm[simpID] > 0) Vmem_free(thee->vmem, thee->nsqm[simpID],
00525           sizeof(int), (void **)&(thee->sqm[simpID]));
00526         thee->sqm[simpID] = sqmNew[isimp];
00527         thee->nsqm[simpID] = nsqmNew[isimp];
00528     }
00529
```

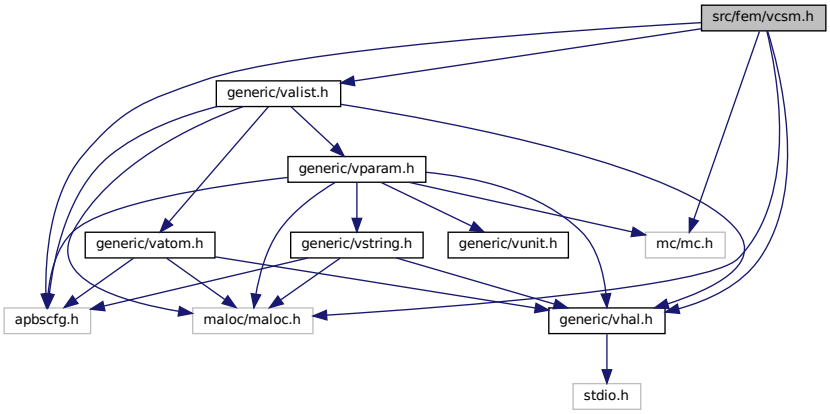
```
00530     Vmem_free(thee->vmem, num, sizeof(int *), (void **)&sqmNew);
00531     Vmem_free(thee->vmem, num, sizeof(int), (void **)&nsqmNew);
00532     Vmem_free(thee->vmem, nAffAtoms, sizeof(int *), (void **)&qsmNew);
00533     Vmem_free(thee->vmem, nAffAtoms, sizeof(int), (void **)&nqsmNew);
00534     Vmem_free(thee->vmem, nAffAtoms, sizeof(int), (void **)&dnqsm);
00535
00536
00537     return 1;
00538
00539
00540 }
```

9.5 src/fem/vcsm.h File Reference

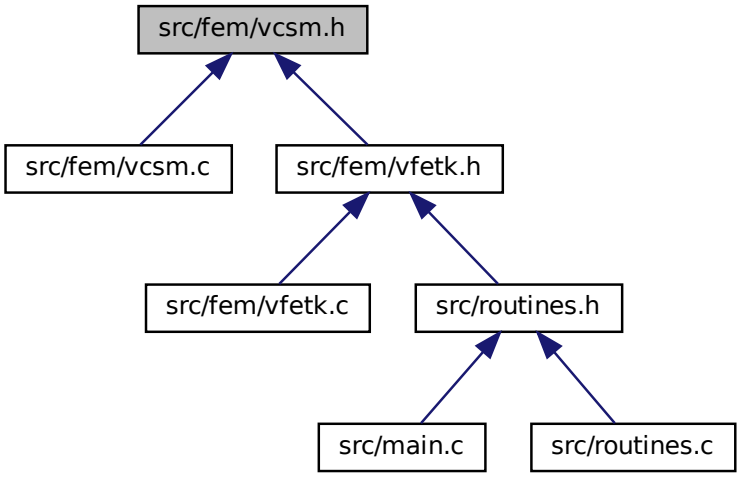
Contains declarations for the Vcsm class.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "mc/mc.h"
#include "generic/vhal.h"
#include "generic/valist.h"
```

Include dependency graph for vcsm.h:



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



Data Structures

- struct sVcsm

  Charge-simplex map class.

Typedefs

- typedef struct sVcsm Vcsm

  Declaration of the Vcsm class as the Vcsm structure.

Functions

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

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

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

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

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

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

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

  Get particular simplex associated with an atom.

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

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

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

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

  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.

9.5.1 Detailed Description

Contains declarations for the Vcsm class.

Version

Id

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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.

9.6 vcsm.h

```
00001
00063 #ifndef _VCSM_H_
00064 #define _VCSM_H_
00065
00066 #include "apbscfg.h"
00067
00068 #include "maloc/maloc.h"
00069 #include "mc/mc.h"
00070
00071 #include "generic/vhal.h"
00072 #include "generic/valist.h"
00073
00078 VEXTERNC void Gem_setExternalUpdateFunction(
00079         Gem *thee,
00080         void (*externalUpdate)(SS **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 /**< The Vcsm object */
00137             );
00138
00144     VEXTERNC int Vcsm_getNumberAtoms(
00145             Vcsm *thee,
00146             int isimp
00147             );
00148
00154     VEXTERNC Vatom* Vcsm_getAtom(
00155             Vcsm *thee,
00156             int iatom,
00157             int isimp
```

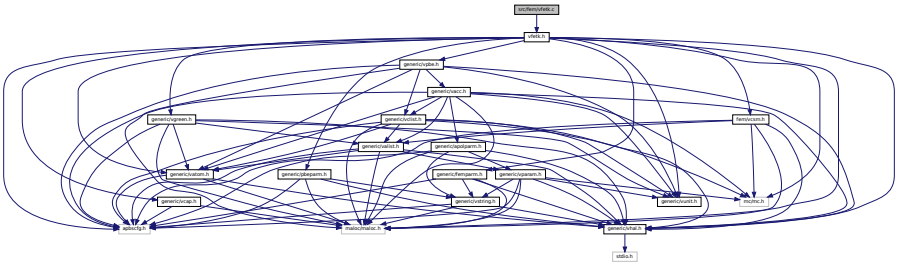
```
00158             );
00159
00165     VEXTERNC int Vcsm_getAtomIndex(
00166             Vcsm *thee,
00167             int iatom,
00168             int isimp
00169             );
00170
00176     VEXTERNC int Vcsm_getNumberSimplices(
00177             Vcsm *thee,
00178             int iatom
00179             );
00180
00186     VEXTERNC SS* Vcsm_getSimplex(
00187             Vcsm *thee,
00188             int isimp,
00189             int iatom
00190             );
00191
00197     VEXTERNC int Vcsm_getSimplexIndex(
00198             Vcsm *thee,
00199             int isimp,
00200             int iatom
00201             );
00202
00209     VEXTERNC unsigned long int Vcsm_memChk(
00210             Vcsm *thee
00211             );
00212
00213 #else /* if defined(VINLINE_VCSM) */
00214 #   define Vcsm_getValist(thee) ((thee)->alist)
00215 #   define Vcsm_getNumberAtoms(thee, isimp) ((thee)->nsqm[isimp])
00216 #   define Vcsm_getAtom(thee, iatom, isimp) (Valist_getAtom((thee)->alist, ((thee)->sqm)[isimp][iatom]))
00217 #   define Vcsm_getAtomIndex(thee, iatom, isimp) (((thee)->sqm)[isimp][iatom])
00218 #   define Vcsm_getNumberSimplices(thee, iatom) (((thee)->nqsm)[iatom])
00219 #   define Vcsm_getSimplex(thee, isimp, iatom) (Gem_SS((thee)->gm, ((thee)->qsm)[iatom][isimp]))
00220 #   define Vcsm_getSimplexIndex(thee, isimp, iatom) (((thee)->qsm)[iatom][isimp])
00221 #   define Vcsm_memChk(thee) (Vmem_bytes((thee)->vmem))
00222 #endif /* if !defined(VINLINE_VCSM) */
00223
00224 /* ///////////////////////////////////////////////////////////////////////////
00225 // Class Vcsm: Non-Inlineable methods (vcsm.c)
00227
00236 VEXTERNC Vcsm* Vcsm_ctor(
00237         Valist *alist,  /**< List of atoms */
00238         Gem *gm
00239         );
00240
00249 VEXTERNC int Vcsm_ctor2(
00250         Vcsm *thee,
00251         Valist *alist,
00252         Gem *gm
00253         );
00254
00259 VEXTERNC void Vcsm_dtor(
00260         Vcsm **thee
00261         );
00262
00267 VEXTERNC void Vcsm_dtor2(
00268         Vcsm *thee
00269         );
00270
00277 VEXTERNC void Vcsm_init(
00278         Vcsm *thee
00279         );
00280
00287 VEXTERNC int Vcsm_update(
00288         Vcsm *thee,
00289         SS **simps,
00294         int num
00295         );
00296
00297 #endif /* ifndef _VCSM_H_ */
```

9.7 src/fem/vfetk.c File Reference

Class Vfetk methods.

```
#include "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.

- VPUBLIC double Vfetk_energy (Vfetk ∗thee, int color, int nonlin)

  Return the total electrostatic energy.

- VPUBLIC double Vfetk_qfEnergy (Vfetk ∗thee, int color)

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

- VPUBLIC double Vfetk_dqmEnergy (Vfetk *thee, int color)

  Get the "mobile charge" and "polarization" contributions to the electrostatic energy.
- VPUBLIC void Vfetk_setAtomColors (Vfetk *thee)

  Transfer color (partition ID) information frmo a partitioned mesh to the atoms.
- VPUBLIC unsigned long int Vfetk_memChk (Vfetk *thee)

  Return the memory used by this structure (and its contents) in bytes.
- VPUBLIC Vrc_Codes Vfetk_genCube (Vfetk *thee, double center[3], double length[3], Vfetk_MeshLoad mesh↩ Type)

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

  Loads a mesh into the Vfetk (and associated) object(s).
- VPUBLIC void Bmat_printHB (Bmat *thee, char *fname)

  Writes a Bmat to disk in Harwell-Boeing sparse matrix format.
- VPUBLIC PDE * Vfetk_PDE_ctor (Vfetk *fetk)

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

  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:
- 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:
- 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:

- 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]

9.7.1 Detailed Description

Class Vfetk methods.

Author

Nathan Baker

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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) 2002-2020, 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.

9.7.2 Variable Documentation

9.7.2.1 lgr_2DP1

```
VPRIVATE int lgr_2DP1[3][VMAXP]
```

Initial value:

```
= {
{  2, -2, -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
}
```

Definition at line 386 of file vfetk.c.

9.7.2.2 lgr_2DP1x

```
VPRIVATE int lgr_2DP1x[3][VMAXP]
```

Initial value:

```
= {
{ -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
}
```

Definition at line 395 of file vfetk.c.

9.7.2.3 lgr_2DP1y

```
VPRIVATE int lgr_2DP1y[3][VMAXP]
```

Initial value:

```
= {
{ -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
}
```

Definition at line 402 of file vfetk.c.

9.7.2.4 lgr_2DP1z

```
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 }
}
```

Definition at line 409 of file vfetk.c.

9.7.2.5 lgr_3DP1

```
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,  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,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
}
```

Definition at line 438 of file vfetk.c.

9.7.2.6 lgr_3DP1x

```
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 },
{  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 }
}
```

Definition at line 446 of file vfetk.c.

9.7.2.7 lgr_3DP1y

```
VPRIVATE int lgr_3DP1y[VAPBS_NVS][VMAXP]
```

Initial value:

```
= {
{ -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
}
```

Definition at line 454 of file vfetk.c.

9.7.2.8 lgr_3DP1z

```
VPRIVATE int lgr_3DP1z[VAPBS_NVS][VMAXP]
```

Initial value:

```
= {
{ -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
{  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
}
```

Definition at line 462 of file vfetk.c.

9.8 vfetk.c

```
00001
00057 #include "vfetk.h"
00058
00059 /* Define the macro DONEUMANN to run with all-Neumann boundary conditions.
00060  * Set this macro at your own risk! */
00061 /* #define DONEUMANN 1 */
00062
00063 /*
00064  * @brief  Calculuate the contribution to the charge-potential energy from one
00065  * atom
00066  * @ingroup Vfetk
00067  * @author  Nathan Baker
00068  * @param  thee  current Vfetk object
00069  * @param  iatom  current atom index
00070  * @param  color  simplex subset (partition) under consideration
00071  * @param  sol  current solution
00072  * @returns Per-atom energy
00073  */
00074 VPRIVATE double Vfetk_qfEnergyAtom(
00075         Vfetk *thee,
00076         int iatom,
00077         int color,
00078         double *sol
00079         );
00080
00081 /*
00082  * @brief  Container for local variables
00083  * @ingroup  Vfetk
00084  * @bug  Not thread-safe
00085  */
00086 VPRIVATE Vfetk_LocalVar var;
00087
00088 /*
00089  * @brief  MCSF-format cube mesh (all Dirichlet)
00090  * @ingroup Vfetk
00091  * @author  Based on mesh by Mike Holst
00092  */
00093 VPRIVATE char *diriCubeString =
00094 "mcsf_begin=1;\n\
00095 \n\
00096 dim=3;\n\
00097 dimii=3;\n\
00098 vertices=8;\n\
00099 simplices=6;\n\
00100 \n\
00101 vert=[\n\
00102 0 0 -0.5 -0.5 -0.5\n\
00103 1 0  0.5 -0.5 -0.5\n\
00104 2 0 -0.5  0.5 -0.5\n\
00105 3 0  0.5  0.5 -0.5\n\
00106 4 0 -0.5 -0.5  0.5\n\
```

```
00107 5 0  0.5 -0.5  0.5\n\
00108 6 0 -0.5  0.5  0.5\n\
00109 7 0  0.5  0.5  0.5\n\
00110 ];\n\
00111 \n\
00112 simp=[\n\
00113 0 0 0 0 1 0 1 0 5 1 2\n\
00114 1 0 0 0 1 1 0 0 5 2 4\n\
00115 2 0 0 0 1 0 1 1 5 3 2\n\
00116 3 0 0 0 1 0 1 3 5 7 2\n\
00117 4 0 0 1 1 0 0 2 5 7 6\n\
00118 5 0 0 1 1 0 0 2 5 6 4\n\
00119 ];\n\
00120 \n\
00121 mcsf_end=1;\n\
00122 \n\
00123 ";
00124
00125 /*
00126  * @brief  MCSF-format cube mesh (all Neumann)
00127  * @ingroup Vfetk
00128  * @author  Based on mesh by Mike Holst
00129  */
00130 VPRIVATE char *neumCubeString =
00131 "mcsf_begin=1;\n\
00132 \n\
00133 dim=3;\n\
00134 dimii=3;\n\
00135 vertices=8;\n\
00136 simplices=6;\n\
00137 \n\
00138 vert=[\n\
00139 0 0 -0.5 -0.5 -0.5\n\
00140 1 0  0.5 -0.5 -0.5\n\
00141 2 0 -0.5  0.5 -0.5\n\
00142 3 0  0.5  0.5 -0.5\n\
00143 4 0 -0.5 -0.5  0.5\n\
00144 5 0  0.5 -0.5  0.5\n\
00145 6 0 -0.5  0.5  0.5\n\
00146 7 0  0.5  0.5  0.5\n\
00147 ];\n\
00148 \n\
00149 simp=[\n\
00150 0 0 0 0 2 0 2 0 5 1 2\n\
00151 1 0 0 0 2 2 0 0 5 2 4\n\
00152 2 0 0 0 2 0 2 1 5 3 2\n\
00153 3 0 0 0 2 0 2 3 5 7 2\n\
00154 4 0 0 2 2 0 0 2 5 7 6\n\
00155 5 0 0 2 2 0 0 2 5 6 4\n\
00156 ];\n\
00157 \n\
00158 mcsf_end=1;\n\
00159 \n\
00160 ";
00161
00162 /*
00163  * @brief  Return the smoothed value of the dielectric coefficient at the
00164  * current point using a fast, chart-based method
00165  * @ingroup  Vfetk
00166  * @author  Nathan Baker
00167  * @returns  Value of dielectric coefficient
00168  * @bug  Not thread-safe
00169  */
00170 VPRIVATE double diel();
00171
00172 /*
00173  * @brief  Return the smoothed value of the ion accessibility at the
00174  * current point using a fast, chart-based method
00175  * @ingroup  Vfetk
00176  * @author  Nathan Baker
00177  * @returns  Value of mobile ion coefficient
00178  * @bug  Not thread-safe
00179  */
00180 VPRIVATE double ionacc();
00181
00182 /*
00183  * @brief  Smooths a mesh-based coefficient with a simple harmonic function
00184  * @ingroup  Vfetk
00185  * @author  Nathan Baker
00186  * @param  meth  Method for smoothing
00187  *   \li  0 ==> arithmetic mean (gives bad results)
```

```
00188  *   \li  1 ==> geometric mean
00189  * @param  nverts  Number of vertices
00190  * @param  dist  distance from point to each vertex
00191  * @param  coeff  coefficient value at each vertex
00192  * @note  Thread-safe
00193  * @return smoothed value of coefficieent at point of interest */
00194 VPRIVATE double smooth(
00195         int nverts,
00196         double dist[VAPBS_NVS],
00197         double coeff[VAPBS_NVS],
00198         int meth
00199         );
00200
00201
00202 /*
00203  * @brief  Return the analytical multi-sphere Debye-Huckel approximation (in
00204  * kT/e) at the specified point
00205  * @ingroup  Vfetk
00206  * @author  Nathan Baker
00207  * @param  pbe  Vpbe object
00208  * @param  d  Dimension of x
00209  * @param  x  Coordinates of point of interest (in &Aring;)
00210  * @note  Thread-safe
00211  * @returns  Multi-sphere Debye-Huckel potential in kT/e
00212  */
00213 VPRIVATE double debye_U(
00214         Vpbe *pbe,
00215         int d,
00216         double x[]
00217         );
00218
00219 /*
00220  * @brief  Return the difference between the analytical multi-sphere
00221  * Debye-Huckel approximation and Coulomb's law (in kT/e) at the specified
00222  * point
00223  * @ingroup  Vfetk
00224  * @author  Nathan Baker
00225  * @param  pbe  Vpbe object
00226  * @param  d  Dimension of x
00227  * @param  x  Coordinates of point of interest (in &Aring;)
00228  * @note  Thread-safe
00229  * @returns  Multi-sphere Debye-Huckel potential in kT/e */
00230 VPRIVATE double debye_Udiff(
00231         Vpbe *pbe,
00232         int d,
00233         double x[]
00234         );
00235
00236 /*
00237  * @brief  Calculate the Coulomb's
00238  * Debye-Huckel approximation and Coulomb's law (in kT/e) at the specified
00239  * point
00240  * @ingroup  Vfetk
00241  * @author  Nathan Baker
00242  * @param  pbe  Vpbe object
00243  * @param  d  Dimension of x
00244  * @param  x  Coordinates of point of interest (in &Aring;)
00245  * @param  eps  Dielectric constant
00246  * @param  U  Set to potential (in kT/e)
00247  * @param  dU  Set to potential gradient (in kT/e/&Aring;)
00248  * @param  d2U  Set to Laplacian of potential (in \f$kT e^{-1} \AA^{-2}\f$)
00249  * @returns  Multi-sphere Debye-Huckel potential in kT/e */
00250 VPRIVATE void coulomb(
00251         Vpbe *pbe,
00252         int d,
00253         double x[],
00254         double eps,
00255         double *U,
00256         double dU[],
00257         double *d2U
00258         );
00259
00260 /*
00261  * @brief  2D linear master simplex information generator
00262  * @ingroup  Vfetk
00263  * @author  Mike Holst
00264  * @param dimIS  dunno
00265  * @param ndof  dunno
00266  * @param dof  dunno
00267  * @param c  dunno
00268  * @param cx  dunno
```

```
00269  * @note  Trust in Mike */
00270 VPRIVATE void init_2DP1(
00271         int dimIS[],
00272         int *ndof,
00273         int dof[],
00274         double c[][VMAXP],
00275         double cx[][VMAXP],
00276         double cy[][VMAXP],
00277         double cz[][VMAXP]
00278         );
00279
00280 /*
00281  * @brief  3D linear master simplex information generator
00282  * @ingroup  Vfetk
00283  * @author  Mike Holst
00284  * @param  dimIS  dunno
00285  * @param ndof dunno
00286  * @param dof dunno
00287  * @param c dunno
00288  * @param cx dunno
00289  * @param cy dunno
00290  * @param cz dunno
00291  * @note  Trust in Mike */
00292 VPRIVATE void init_3DP1(
00293         int dimIS[],
00294         int *ndof,
00295         int dof[],
00296         double c[][VMAXP],
00297         double cx[][VMAXP],
00298         double cy[][VMAXP],
00299         double cz[][VMAXP]
00300         );
00301
00302 /*
00303  * @brief  Setup coefficients of polynomials from integer table data
00304  * @ingroup  Vfetk
00305  * @author  Mike Holst
00306  * @param numP  dunno
00307  * @param c  dunno
00308  * @param cx  dunno
00309  * @param cy  dunno
00310  * @param cz  dunno
00311  * @param ic  dunno
00312  * @param icx  dunno
00313  * @param icy  dunno
00314  * @param icz  dunno
00315  * @note  Trust in Mike */
00316 VPRIVATE void setCoef(
00317         int numP,
00318         double c[][VMAXP],
00319         double cx[][VMAXP],
00320         double cy[][VMAXP],
00321         double cz[][VMAXP],
00322         int ic[][VMAXP],
00323         int icx[][VMAXP],
00324         int icy[][VMAXP],
00325         int icz[][VMAXP]
00326         );
00327
00328 /*
00329  * @brief  Evaluate a collection of at most cubic polynomials at a
00330  * specified point in at most R^3.
00331  * @ingroup  Vfetk
00332  * @author  Mike Holst
00333  * @param numP  the number of polynomials to evaluate
00334  * @param p  the results of the evaluation
00335  * @param c   the coefficients of each polynomial
00336  * @param xv  the point (x,y,z) to evaluate the polynomials.
00337  * @note  Mike says:
00338  * <pre>
00339  *  Note that "VMAXP" must be >= 19 for cubic polynomials.
00340  *  The polynomials are build from the coefficients c[][] as
00341  *  follows.  To build polynomial "k", fix k and set:
00342  *
00343  *  c0=c[k][0], c1=c[k][1], .... , cp=c[k][p]
00344  *
00345  *  Then evaluate as:
00346  *
00347  *  p3(x,y,z) = c0 + c1*x + c2*y + c3*z
00348  *            + c4*x*x + c5*y*y + c6*z*z + c7*x*y + c8*x*z + c9*y*z
00349  *            + c10*x*x*x + c11*y*y*y + c12*z*z*z
```

```
00350  *            + c13*x*x*y + c14*x*x*z + c15*x*y*y
00351  *            + c16*y*y*z + c17*x*z*z + c18*y*z*z
00352  * </pre>
00353  */
00354 VPRIVATE void polyEval(
00355         int numP,
00356         double p[],
00357         double c[][VMAXP],
00358         double xv[]
00359         );
00360
00361 /*
00362  * @brief  I have no clue what this variable does, but we need it to initialize
00363  * the simplices
00364  * @ingroup  Vfetk
00365  * @author  Mike Holst */
00366 VPRIVATE int dim_2DP1 = 3;
00367
00368 /*
00369  * @brief  I have no clue what these variable do, but we need it to initialize
00370  * the simplices
00371  * @ingroup  Vfetk
00372  * @author  Mike Holst
00373  * @note  Mike says:
00374  * <pre>
00375  *  2D-P1 Basis:
00376  *
00377  *  p1(x,y) = c0 + c1*x + c2*y
00378  *
00379  *  Lagrange Point    Lagrange Basis Function Definition
00380  *  --------------    ----------------------------------
00381  *  (0, 0)            p[0](x,y) = 1 - x - y
00382  *  (1, 0)            p[1](x,y) = x
00383  *  (0, 1)            p[2](x,y) = y
00384  *  </pre>
00385  */
00386 VPRIVATE int lgr_2DP1[3][VMAXP] = {
00387 /*c0  c1  c2  c3
00388 * ------------------------------------------------------- */
00389 /* 1   x   y   z
00390 * ------------------------------------------------------- */
00391 {  2, -2, -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00392 {  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00393 {  0,  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
00394 };
00395 VPRIVATE int lgr_2DP1x[3][VMAXP] = {
00396 /*c0 ------------------------------------------------------------------------ */
00397 /* 1 ------------------------------------------------------------------------ */
00398 { -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00399 {  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00400 {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
00401 };
00402 VPRIVATE int lgr_2DP1y[3][VMAXP] = {
00403 /*c0 ------------------------------------------------------------------------ */
00404 /* 1 ------------------------------------------------------------------------ */
00405 { -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00406 {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00407 {  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
00408 };
00409 VPRIVATE int lgr_2DP1z[3][VMAXP] = {
00410 /*c0 ------------------------------------------------------------------------ */
00411 /* 1 ------------------------------------------------------------------------ */
00412 {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00413 {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00414 {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
00415 };
00416
00417
00418 /*
00419  * @brief  I have no clue what these variable do, but we need it to initialize
00420  * the simplices
00421  * @ingroup  Vfetk
00422  * @author  Mike Holst
00423  * @note  Mike says:
00424  * <pre>
00425  * 3D-P1 Basis:
00426  *
00427  * p1(x,y,z) = c0 + c1*x + c2*y + c3*z
00428  *
00429  * Lagrange Point    Lagrange Basis Function Definition
00430  * --------------    ----------------------------------
```

```
00431  * (0, 0, 0)          p[0](x,y,z) = 1 - x - y - z
00432  * (1, 0, 0)          p[1](x,y,z) = x
00433  * (0, 1, 0)          p[2](x,y,z) = y
00434  * (0, 0, 1)          p[3](x,y,z) = z
00435  * </pre>
00436  */
00437 VPRIVATE int dim_3DP1 = VAPBS_NVS;
00438 VPRIVATE int lgr_3DP1[VAPBS_NVS][VMAXP] = {
00439 /*c0  c1  c2  c3 ------------------------------------------------------- */
00440 /* 1   x   y   z ------------------------------------------------------- */
00441 {  2, -2, -2, -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00442 {  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00443 {  0,  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00444 {  0,  0,  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
00445 };
00446 VPRIVATE int lgr_3DP1x[VAPBS_NVS][VMAXP] = {
00447 /*c0 ------------------------------------------------------------------- */
00448 /* 1 ------------------------------------------------------------------- */
00449 { -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00450 {  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00451 {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00452 {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
00453 };
00454 VPRIVATE int lgr_3DP1y[VAPBS_NVS][VMAXP] = {
00455 /*c0 ------------------------------------------------------------------- */
00456 /* 1 ------------------------------------------------------------------- */
00457 { -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00458 {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00459 {  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00460 {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 }
00461 };
00462 VPRIVATE int lgr_3DP1z[VAPBS_NVS][VMAXP] = {
00463 /*c0 ------------------------------------------------------------------- */
00464 /* 1 ------------------------------------------------------------------- */
00465 { -2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00466 {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00467 {  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00468 {  2,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0 },
00469 };
00470
00471 /*
00472  * @brief  Another Holst variable
00473  * @ingroup  Vfetk
00474  * @author  Mike Holst
00475  * @note  Mike says: 1 = linear, 2 = quadratic */
00476 VPRIVATE const int P_DEG=1;
00477
00478 /*
00479  * @brief  Another Holst variable
00480  * @ingroup  Vfetk
00481  * @author  Mike Holst */
00482 VPRIVATE int numP;
00483 VPRIVATE double c[VMAXP][VMAXP];
00484 VPRIVATE double cx[VMAXP][VMAXP];
00485 VPRIVATE double cy[VMAXP][VMAXP];
00486 VPRIVATE double cz[VMAXP][VMAXP];
00487
00488 #if !defined(VINLINE_VFETK)
00489
00490 VPUBLIC Gem* Vfetk_getGem(Vfetk *thee) {
00491
00492    VASSERT(thee != VNULL);
00493    return thee->gm;
00494
00495 }
00496
00497 VPUBLIC AM* Vfetk_getAM(Vfetk *thee) {
00498
00499    VASSERT(thee != VNULL);
00500    return thee->am;
00501 }
00502
00503 VPUBLIC Vpbe* Vfetk_getVpbe(Vfetk *thee) {
00504
00505    VASSERT(thee != VNULL);
00506    return thee->pbe;
00507
00508 }
00509
00510 VPUBLIC Vcsm* Vfetk_getVcsm(Vfetk *thee) {
00511
```

```
00512    VASSERT(thee != VNULL);
00513    return thee->csm;
00514
00515 }
00516
00517 VPUBLIC int Vfetk_getAtomColor(Vfetk *thee,
00518                                int iatom
00519                               ) {
00520
00521     int natoms;
00522
00523     VASSERT(thee != VNULL);
00524
00525     natoms = Valist_getNumberAtoms(Vpbe_getValist(thee->pbe));
00526     VASSERT(iatom < natoms);
00527
00528     return Vatom_getPartID(Valist_getAtom(Vpbe_getValist(thee->pbe), iatom));
00529 }
00530 #endif /* if !defined(VINLINE_VFETK) */
00531
00532 VPUBLIC Vfetk* Vfetk_ctor(Vpbe *pbe,
00533                           Vhal_PBEType type
00534                          ) {
00535
00536     /* Set up the structure */
00537     Vfetk *thee = VNULL;
00538     thee = (Vfetk*)Vmem_malloc(VNULL, 1, sizeof(Vfetk) );
00539     VASSERT(thee != VNULL);
00540     VASSERT(Vfetk_ctor2(thee, pbe, type));
00541
00542     return thee;
00543 }
00544
00545 VPUBLIC int Vfetk_ctor2(Vfetk *thee,
00546                         Vpbe *pbe,
00547                         Vhal_PBEType type
00548                        ) {
00549
00550     int i;
00551     double center[VAPBS_DIM];
00552
00553     /* Make sure things have been properly initialized & store them */
00554     VASSERT(pbe != VNULL);
00555     thee->pbe = pbe;
00556     VASSERT(pbe->alist != VNULL);
00557     VASSERT(pbe->acc != VNULL);
00558
00559     /* Store PBE type */
00560     thee->type = type;
00561
00562     /* Set up memory management object */
00563     thee->vmem = Vmem_ctor("APBS::VFETK");
00564
00565     /* Set up FEtk objects */
00566     Vnm_print(0, "Vfetk_ctor2:  Constructing PDE...\n");
00567     thee->pde = Vfetk_PDE_ctor(thee);
00568     Vnm_print(0, "Vfetk_ctor2:  Constructing Gem...\n");
00569     thee->gm = Gem_ctor(thee->vmem, thee->pde);
00570     Vnm_print(0, "Vfetk_ctor2:  Constructing Aprx...\n");
00571     thee->aprx = Aprx_ctor(thee->vmem, thee->gm, thee->pde);
00572     Vnm_print(0, "Vfetk_ctor2:  Constructing Aprx...\n");
00573     thee->am = AM_ctor(thee->vmem, thee->aprx);
00574
00575     /* Reset refinement level */
00576     thee->level = 0;
00577
00578     /* Set default solver variables */
00579     thee->lkey = VLT_MG;
00580     thee->lmax = 1000000;
00581     thee->ltol = 1e-5;
00582     thee->lprec = VPT_MG;
00583     thee->nkey = VNT_NEW;
00584     thee->nmax = 1000000;
00585     thee->ntol = 1e-5;
00586     thee->gues = VGT_ZERO;
00587     thee->pjac = -1;
00588
00589     /* Store local copy of myself */
00590     var.fetk = thee;
00591     var.initGreen = 0;
00592
```

```
00593     /* Set up the external Gem subdivision hook */
00594     Gem_setExternalUpdateFunction(thee->gm, Vfetk_externalUpdateFunction);
00595
00596     /* Set up ion-related variables */
00597     var.zkappa2 = Vpbe_getZkappa2(var.fetk->pbe);
00598     var.ionstr = Vpbe_getBulkIonicStrength(var.fetk->pbe);
00599     if (var.ionstr > 0.0) var.zks2 = 0.5*var.zkappa2/var.ionstr;
00600     else var.zks2 = 0.0;
00601     Vpbe_getIons(var.fetk->pbe, &(var.nion), var.ionConc, var.ionRadii,
00602       var.ionQ);
00603     for (i=0; i<var.nion; i++) {
00604         var.ionConc[i] = var.zks2 * var.ionConc[i] * var.ionQ[i];
00605     }
00606
00607     /* Set uninitialized objects to NULL */
00608     thee->pbeparm = VNULL;
00609     thee->feparm = VNULL;
00610     thee->csm = VNULL;
00611
00612     return 1;
00613 }
00614
00615 VPUBLIC void Vfetk_setParameters(Vfetk *thee,
00616                                  PBEparm *pbeparm,
00617                                  FEMparm *feparm
00618                                 ) {
00619
00620     VASSERT(thee != VNULL);
00621     thee->feparm = feparm;
00622     thee->pbeparm = pbeparm;
00623 }
00624
00625 VPUBLIC void Vfetk_dtor(Vfetk **thee) {
00626     if ((*thee) != VNULL) {
00627         Vfetk_dtor2(*thee);
00628         //Vmem_free(VNULL, 1, sizeof(Vfetk), (void **)thee);
00629         (*thee) = VNULL;
00630     }
00631 }
00632
00633 VPUBLIC void Vfetk_dtor2(Vfetk *thee) {
00634     Vcsm_dtor(&(thee->csm));
00635     AM_dtor(&(thee->am));
00636     Aprx_dtor(&(thee->aprx));
00637     Vfetk_PDE_dtor(&(thee->pde));
00638     Vmem_dtor(&(thee->vmem));
00639 }
00640
00641 VPUBLIC double* Vfetk_getSolution(Vfetk *thee,
00642                                   int *length
00643                                  ) {
00644
00645    int i;
00646    double *solution,
00647           *theAnswer;
00648    AM *am;
00649
00650    VASSERT(thee != VNULL);
00651
00652    /* Get the AM object */
00653    am = thee->am;
00654    /* Copy the solution into the w0 vector */
00655    Bvec_copy(am->w0, am->u);
00656    /* Add the Dirichlet conditions */
00657    Bvec_axpy(am->w0, am->ud, 1.);
00658    /* Get the data from the Bvec */
00659    solution = Bvec_addr(am->w0);
00660    /* Get the length of the data from the Bvec */
00661    *length = Bvec_numRT(am->w0);
00662    /* Make sure that we got scalar data (only one block) for the solution
00663     * to the FETK */
00664    VASSERT(1 == Bvec_numB(am->w0));
00665    /* Allocate space for the returned vector and copy the solution into it */
00666    theAnswer = VNULL;
00667    theAnswer = (double*)Vmem_malloc(VNULL, *length, sizeof(double));
00668    VASSERT(theAnswer != VNULL);
00669    for (i=0; i<(*length); i++) theAnswer[i] = solution[i];
00670
00671    return theAnswer;
00672 }
00673
```

```
00674
00693 VPUBLIC double Vfetk_energy(Vfetk *thee,
00694                             int color,
00698                             int nonlin
00700                            ) {
00701
00702     double totEnergy = 0.0,
00703            qfEnergy = 0.0,
00704            dqmEnergy = 0.0;
00706     VASSERT(thee != VNULL);
00707
00708     if (nonlin && (Vpbe_getBulkIonicStrength(thee->pbe) > 0.)) {
00709         Vnm_print(0, "Vfetk_energy:  calculating full PBE energy\n");
00710         Vnm_print(0, "Vfetk_energy:  bulk ionic strength = %g M\n",
00711           Vpbe_getBulkIonicStrength(thee->pbe));
00712         dqmEnergy = Vfetk_dqmEnergy(thee, color);
00713         Vnm_print(0, "Vfetk_energy:  dqmEnergy = %g kT\n", dqmEnergy);
00714         qfEnergy = Vfetk_qfEnergy(thee, color);
00715         Vnm_print(0, "Vfetk_energy:  qfEnergy = %g kT\n", qfEnergy);
00716
00717         totEnergy = qfEnergy - dqmEnergy;
00718     } else {
00719         Vnm_print(0, "Vfetk_energy:  calculating only q-phi energy\n");
00720         dqmEnergy = Vfetk_dqmEnergy(thee, color);
00721         Vnm_print(0, "Vfetk_energy:  dqmEnergy = %g kT (NOT USED)\n", dqmEnergy);
00722         qfEnergy = Vfetk_qfEnergy(thee, color);
00723         Vnm_print(0, "Vfetk_energy:  qfEnergy = %g kT\n", qfEnergy);
00724         totEnergy = 0.5*qfEnergy;
00725     }
00726
00727     return totEnergy;
00728
00729 }
00730
00731
00732 VPUBLIC double Vfetk_qfEnergy(Vfetk *thee,
00733                               int color
00734                              ) {
00735
00736     double *sol,
00737            energy = 0.0;
00738     int nsol,
00739         iatom,
00740         natoms;
00741     AM *am;
00742
00743     VASSERT(thee != VNULL);
00744     am = thee->am;
00745
00746     /* Get the finest level solution */
00747     sol= VNULL;
00748     sol = Vfetk_getSolution(thee, &nsol);
00749     VASSERT(sol != VNULL);
00750
00751     /* Make sure the number of entries in the solution array matches the
00752      * number of vertices currently in the mesh */
00753     if (nsol != Gem_numVV(thee->gm)) {
00754        Vnm_print(2, "Vfetk_qfEnergy: Number of unknowns in solution does not match\n");
00755        Vnm_print(2, "Vfetk_qfEnergy: number of vertices in mesh!!!  Bailing out!\n");
00756        VASSERT(0);
00757     }
00758
00759     /* Now we do the sum over atoms... */
00760     natoms = Valist_getNumberAtoms(thee->pbe->alist);
00761     for (iatom=0; iatom<natoms; iatom++) {
00762
00763         energy = energy + Vfetk_qfEnergyAtom(thee, iatom, color, sol);
00764
00765     } /* end for iatom */
00766
00767     /* Destroy the finest level solution */
00768     Vmem_free(VNULL, nsol, sizeof(double), (void **)&sol);
00769
00770     /* Return the energy */
00771     return energy;
00772 }
00773
00774 VPRIVATE double Vfetk_qfEnergyAtom(
00775         Vfetk *thee,
00776         int iatom,
00777         int color,
```

```
00778         double *sol) {
00779
00780     Vatom *atom;
00781     double charge,
00782            phi[VAPBS_NVS],
00783            phix[VAPBS_NVS][3],
00784            *position,
00785            uval,
00786            energy = 0.0;
00787     int isimp,
00788         nsimps,
00789         icolor,
00790         ivert,
00791         usingColor;
00792     SS *simp;
00793
00794
00795     /* Get atom information */
00796     atom = Valist_getAtom(thee->pbe->alist, iatom);
00797     icolor = Vfetk_getAtomColor(thee, iatom);
00798     charge = Vatom_getCharge(atom);
00799     position = Vatom_getPosition(atom);
00800
00801     /* Find out if we're using colors */
00802     usingColor = (color >= 0);
00803
00804     if (usingColor && (icolor<0)) {
00805         Vnm_print(2, "Vfetk_qfEnergy: Atom colors not set!\n");
00806         VASSERT(0);
00807     }
00808
00809     /* Check if this atom belongs to the specified partition */
00810     if ((icolor==color) || (!usingColor)) {
00811         /* Loop over the simps associated with this atom */
00812         nsimps =  Vcsm_getNumberSimplices(thee->csm, iatom);
00813
00814         /* Get the first simp of the correct color; we can use just one
00815          * simplex for energy evaluations, but not for force
00816          * evaluations */
00817         for (isimp=0; isimp<nsimps; isimp++) {
00818
00819             simp = Vcsm_getSimplex(thee->csm, isimp, iatom);
00820
00821             /* If we've asked for a particular partition AND if the atom
00822              * is our partition, then compute the energy */
00823             if ((SS_chart(simp)==color)||(color<0)) {
00824                 /* Get the value of each basis function evaluated at this
00825                  * point */
00826                 Gem_pointInSimplexVal(thee->gm, simp, position, phi, phix);
00827                 for (ivert=0; ivert<SS_dimVV(simp); ivert++) {
00828                     uval = sol[VV_id(SS_vertex(simp,ivert))];
00829                     energy += (charge*phi[ivert]*uval);
00830                 } /* end for ivert */
00831                 /* We only use one simplex of the appropriate color for
00832                  * energy calculations, so break here */
00833                 break;
00834             } /* endif (color) */
00835         } /* end for isimp */
00836     }
00837
00838    return energy;
00839 }
00840
00841
00842 VPUBLIC double Vfetk_dqmEnergy(Vfetk *thee,
00843                                int color) {
00844
00845     return AM_evalJ(thee->am);
00846
00847 }
00848
00849 VPUBLIC void Vfetk_setAtomColors(Vfetk *thee) {
00850
00851     SS *simp;
00852     Vatom *atom;
00853     int i,
00854         natoms;
00855
00856     VASSERT(thee != VNULL);
00857
00858     natoms = Valist_getNumberAtoms(thee->pbe->alist);
```

```
00859     for (i=0; i<natoms; i++) {
00860         atom = Valist_getAtom(thee->pbe->alist, i);
00861         simp = Vcsm_getSimplex(thee->csm, 0, i);
00862         Vatom_setPartID(atom, SS_chart(simp));
00863     }
00864
00865 }
00866
00867 VPUBLIC unsigned long int Vfetk_memChk(Vfetk *thee) {
00868
00869     int memUse = 0;
00870
00871     if (thee == VNULL) return 0;
00872
00873     memUse = memUse + sizeof(Vfetk);
00874     memUse = memUse + Vcsm_memChk(thee->csm);
00875
00876     return memUse;
00877 }
00878
00885 VPUBLIC Vrc_Codes Vfetk_genCube(Vfetk *thee,
00886                                 double center[3],
00887                                 double length[3],
00888                                 Vfetk_MeshLoad meshType
00889                                ) {
00890
00891     VASSERT(thee != VNULL);
00892
00893     AM *am = VNULL; /* @todo - no idea what this is */
00894     Gem *gm = VNULL; /* Geometry manager */
00895
00896     int skey = 0,  /* Simplex format */
00897         bufsize = 0, /* Buffer size */
00898         i, /* Loop counter */
00899         j; /* Loop counter */
00900     char *key = "r",  /* Read */
00901          *iodev = "BUFF",  /* Buffer */
00902          *iofmt = "ASC",  /* ASCII */
00903          *iohost = "localhost",  /* localhost (dummy) */
00904          *iofile = "0",  /*< socket 0 (dummy) */
00905          buf[VMAX_BUFSIZE]; /* Socket buffer */
00906     Vio *sock = VNULL; /* Socket object */
00907     VV *vx = VNULL; /* @todo - no idea what this is */
00908     double x;
00909
00910     am = thee->am;
00911     VASSERT(am != VNULL);
00912     gm = thee->gm;
00913     VASSERT(gm != VNULL);
00914
00915     /* @note This code is based on Gem_makeCube by Mike Holst */
00916     /* Write mesh string to buffer and read back */
00917     switch (meshType) {
00918         case VML_DIRICUBE:
00919             /* Create a new copy of the DIRICUBE mesh (see globals higher in this file) */
00920             bufsize = strlen(diriCubeString);
00921             VASSERT( bufsize <= VMAX_BUFSIZE );
00922             strncpy(buf, diriCubeString, VMAX_BUFSIZE);
00923             break;
00924         case VML_NEUMCUBE:
00925             /* Create a new copy of the NEUMCUBE mesh (see globals higher in this file) */
00926             bufsize = strlen(neumCubeString);
00927             Vnm_print(2, "Vfetk_genCube:  WARNING!  USING EXPERIMENTAL NEUMANN BOUNDARY CONDITIONS!\n");
00928             VASSERT( bufsize <= VMAX_BUFSIZE );
00929             strncpy(buf, neumCubeString, VMAX_BUFSIZE);
00930             break;
00931         case VML_EXTERNAL:
00932             Vnm_print(2, "Vfetk_genCube:  Got request for external mesh!\n");
00933             Vnm_print(2, "Vfetk_genCube:  How did we get here?\n");
00934             return VRC_FAILURE;
00935         default:
00936             Vnm_print(2, "Vfetk_genCube:  Unknown mesh type (%d)\n", meshType);
00937             return VRC_FAILURE;
00938     }
00939
00940     VASSERT( VNULL != (sock=Vio_socketOpen(key,iodev,iofmt,iohost,iofile)) ); /* Open socket */
00941     Vio_bufTake(sock, buf, bufsize); /* Initialize internal buffer for socket */
00942     AM_read(am, skey, sock); /* Take the initial mesh from the socket and load
00943                                 into internal AM data structure with simplex
00944                                 format */
00945     Vio_connectFree(sock); /* Purge output buffers */
```

```
00946     Vio_bufGive(sock); /* Get pointer to output buffer?  No assignment of return value... */
00947     Vio_dtor(&sock); /* Destroy output buffer */
00948
00949     /* @todo - could the following be done in a single pass? - PCE */
00950     /* Scale (unit) cube - for each vertex, set the new coordinates of that
00951        vertex based on the vertex length */
00952     for (i=0; i<Gem_numVV(gm); i++) {
00953         vx = Gem_VV(gm, i);
00954         for (j=0; j<3; j++) {
00955             x = VV_coord(vx, j);
00956             x *= length[j];
00957             VV_setCoord(vx, j, x);
00958         }
00959     }
00960
00961     /* Add new center - for each vertex, set a new center for the vertex */
00962     for (i=0; i<Gem_numVV(gm); i++) {
00963         vx = Gem_VV(gm, i);
00964         for (j=0; j<3; j++) {
00965             x = VV_coord(vx, j);
00966             x += center[j];
00967             VV_setCoord(vx, j, x);
00968         }
00969     }
00970
00971     return VRC_SUCCESS;
00972 }
00973
00980 VPUBLIC Vrc_Codes Vfetk_loadMesh(Vfetk *thee, /* Vfetk object to load into */
00981                                  double center[3], /* Center for mesh (if constructed) */
00982                                  double length[3], /* Mesh lengths (if constructed) */
00983                                  Vfetk_MeshLoad meshType, /* Type of mesh to load */
00984                                  Vio *sock /* Socket for external mesh data (NULL otherwise) */
00985                                 ) {
00986
00987     Vrc_Codes vrc; /* Function return codes - see vhal.h for enum */
00988     int skey = 0;  /* Simplex format */
00989
00990     /* Load mesh from socket if external mesh, otherwise generate mesh */
00991     switch (meshType) {
00992         case VML_EXTERNAL:
00993             if (sock == VNULL) {
00994                 Vnm_print(2, "Vfetk_loadMesh:  Got NULL socket!\n");
00995                 return VRC_FAILURE;
00996             }
00997             AM_read(thee->am, skey, sock);
00998             Vio_connectFree(sock);
00999             Vio_bufGive(sock);
01000             Vio_dtor(&sock);
01001             break;
01002         case VML_DIRICUBE:
01003         case VML_NEUMCUBE:
01004             /* Create new mesh and store in thee */
01005             vrc = Vfetk_genCube(thee, center, length, meshType);
01006             if (vrc == VRC_FAILURE) return VRC_FAILURE;
01007             break;
01008         default:
01009             Vnm_print(2, "Vfetk_loadMesh:  unrecognized mesh type (%d)!\n",
01010                       meshType);
01011             return VRC_FAILURE;
01012     };
01013
01014     /* Setup charge-simplex map */
01015     Vnm_print(0, "Vfetk_ctor2:  Constructing Vcsm...\n");
01016     thee->csm = VNULL;
01017     /* Construct a new Vcsm with the atom list and gem data */
01018     thee->csm = Vcsm_ctor(Vpbe_getValist(thee->pbe), thee->gm);
01019     VASSERT(thee->csm != VNULL);
01020     Vcsm_init(thee->csm);
01021
01022     return VRC_SUCCESS;
01023 }
01024
01025
01026 VPUBLIC void Bmat_printHB(Bmat *thee,
01027                           char *fname
01028                          ) {
01029
01030     Mat *Ablock;
01031     MATsym pqsym;
01032     int i, j, jj;
```

```
01033     int *IA, *JA;
01034     double *D, *L, *U;
01035     FILE *fp;
01036
01037     char mmtitle[72];
01038     char mmkey[] = {"8charkey"};
01039     int totc = 0, ptrc = 0, indc = 0, valc = 0;
01040     char mxtyp[] = {"RUA"}; /* Real Unsymmetric Assembled */
01041     int nrow = 0, ncol = 0, numZ = 0;
01042     int numZdigits = 0, nrowdigits = 0;
01043     int nptrline = 8, nindline = 8, nvalline = 5;
01044     char ptrfmt[] = {"(8I10)          "}, ptrfmtstr[] = {"%10d"};
01045     char indfmt[] = {"(8I10)          "}, indfmtstr[] = {"%10d"};
01046     char valfmt[] = {"(5E16.8)            "}, valfmtstr[] = {"%16.8E"};
01047
01048     VASSERT( thee->numB == 1 );             /* HARDWIRE FOR NOW */
01049     Ablock = thee->AD[0][0];
01050
01051     VASSERT( Mat_format( Ablock ) == DRC_FORMAT );  /* HARDWIRE FOR NOW */
01052
01053     pqsym = Mat_sym( Ablock );
01054
01055     if ( pqsym == IS_SYM ) {
01056         mxtyp[1] = 'S';
01057     } else if ( pqsym == ISNOT_SYM ) {
01058         mxtyp[1] = 'U';
01059     } else {
01060         VASSERT( 0 ); /* NOT VALID */
01061     }
01062
01063     nrow = Bmat_numRT( thee ); /* Number of rows */
01064     ncol = Bmat_numCT( thee ); /* Number of cols */
01065     numZ = Bmat_numZT( thee ); /* Number of entries */
01066
01067     nrowdigits = (int) (log( nrow )/log( 10 )) + 1;
01068     numZdigits = (int) (log( numZ )/log( 10 )) + 1;
01069
01070     nptrline = (int) ( 80 / (numZdigits + 1) );
01071     nindline = (int) ( 80 / (nrowdigits + 1) );
01072
01073     sprintf(ptrfmt,"(%dI%d)",nptrline,numZdigits+1);
01074     sprintf(ptrfmtstr,"%%%dd",numZdigits+1);
01075     sprintf(indfmt,"(%dI%d)",nindline,nrowdigits+1);
01076     sprintf(indfmtstr,"%%%dd",nrowdigits+1);
01077
01078     ptrc = (int) ( ( (ncol + 1) - 1 ) / nptrline ) + 1;
01079     indc = (int) ( (numZ - 1) / nindline ) + 1;
01080     valc = (int) ( (numZ - 1) / nvalline ) + 1;
01081
01082     totc = ptrc + indc + valc;
01083
01084     sprintf( mmtitle, "Sparse '%s' Matrix - Harwell-Boeing Format - '%s'",
01085              thee->name, fname );
01086
01087    /* Step 0:  Open the file for writing */
01088
01089     fp = fopen( fname, "w" );
01090     if (fp == VNULL) {
01091         Vnm_print(2,"Bmat_printHB:  Ouch couldn't open file <%s>\n",fname);
01092         return;
01093     }
01094
01095     /* Step 1:  Print the header information */
01096
01097     fprintf( fp, "%-72s%-8s\n", mmtitle, mmkey );
01098     fprintf( fp, "%14d%14d%14d%14d%14d\n", totc, ptrc, indc, valc, 0 );
01099     fprintf( fp, "%3s%11s%14d%14d%14d\n", mxtyp, " ", nrow, ncol, numZ );
01100     fprintf( fp, "%-16s%-16s%-20s%-20s\n", ptrfmt, indfmt, valfmt, "6E13.5" );
01101
01102     IA = Ablock->IA;
01103     JA = Ablock->JA;
01104     D = Ablock->diag;
01105     L = Ablock->offL;
01106     U = Ablock->offU;
01107
01108     if ( pqsym == IS_SYM ) {
01109
01110         /* Step 2:  Print the pointer information */
01111
01112         for (i=0; i<(ncol+1); i++) {
01113             fprintf( fp, ptrfmtstr, Ablock->IA[i] + (i+1) );
```

```
01114             if ( ( (i+1) % nptrline ) == 0 ) {
01115                 fprintf( fp, "\n" );
01116             }
01117         }
01118 
01119         if ( ( (ncol+1) % nptrline ) != 0 ) {
01120             fprintf( fp, "\n" );
01121         }
01122 
01123         /* Step 3:  Print the index information */
01124 
01125         j = 0;
01126         for (i=0; i<ncol; i++) {
01127             fprintf( fp, indfmtstr, i+1); /* diagonal */
01128             if ( ( (j+1) % nindline ) == 0 ) {
01129                 fprintf( fp, "\n" );
01130             }
01131             j++;
01132             for (jj=IA[i]; jj<IA[i+1]; jj++) {
01133                 fprintf( fp, indfmtstr, JA[jj] + 1 ); /* lower triangle */
01134                 if ( ( (j+1) % nindline ) == 0 ) {
01135                     fprintf( fp, "\n" );
01136                 }
01137                 j++;
01138             }
01139         }
01140 
01141         if ( ( j % nindline ) != 0 ) {
01142             fprintf( fp, "\n" );
01143         }
01144 
01145         /* Step 4:  Print the value information */
01146 
01147         j = 0;
01148         for (i=0; i<ncol; i++) {
01149             fprintf( fp, valfmtstr, D[i] );
01150             if ( ( (j+1) % nvalline ) == 0 ) {
01151                 fprintf( fp, "\n" );
01152             }
01153             j++;
01154             for (jj=IA[i]; jj<IA[i+1]; jj++) {
01155                 fprintf( fp, valfmtstr, L[jj] );
01156                 if ( ( (j+1) % nvalline ) == 0 ) {
01157                     fprintf( fp, "\n" );
01158                 }
01159                 j++;
01160             }
01161         }
01162 
01163         if ( ( j % nvalline ) != 0 ) {
01164             fprintf( fp, "\n" );
01165         }
01166 
01167     } else { /* ISNOT_SYM */
01168 
01169         VASSERT( 0 ); /* NOT CODED YET */
01170     }
01171 
01172     /* Step 5:  Close the file */
01173     fclose( fp );
01174 }
01175 
01176 VPUBLIC PDE* Vfetk_PDE_ctor(Vfetk *fetk) {
01177 
01178     PDE *thee = VNULL;
01179 
01180     thee = (PDE*)Vmem_malloc(fetk->vmem, 1, sizeof(PDE));
01181     VASSERT(thee != VNULL);
01182     VASSERT(Vfetk_PDE_ctor2(thee, fetk));
01183 
01184     return thee;
01185 }
01186 
01187 VPUBLIC int Vfetk_PDE_ctor2(PDE *thee,
01188                             Vfetk *fetk
01189                            ) {
01190 
01191     int i;
01192 
01193     if (thee == VNULL) {
01194         Vnm_print(2, "Vfetk_PDE_ctor2:  Got NULL thee!\n");
```

```
01195         return 0;
01196     }
01197
01198     /* Store a local copy of the Vfetk class */
01199     var.fetk = fetk;
01200
01201     /* PDE-specific parameters and function pointers */
01202     thee->initAssemble = Vfetk_PDE_initAssemble;
01203     thee->initElement  = Vfetk_PDE_initElement;
01204     thee->initFace     = Vfetk_PDE_initFace;
01205     thee->initPoint    = Vfetk_PDE_initPoint;
01206     thee->Fu           = Vfetk_PDE_Fu;
01207     thee->Fu_v         = Vfetk_PDE_Fu_v;
01208     thee->DFu_wv       = Vfetk_PDE_DFu_wv;
01209     thee->delta        = Vfetk_PDE_delta;
01210     thee->u_D          = Vfetk_PDE_u_D;
01211     thee->u_T          = Vfetk_PDE_u_T;
01212     thee->Ju           = Vfetk_PDE_Ju;
01213     thee->vec          = 1; /* FIX! */
01214     thee->sym[0][0]    = 1;
01215     thee->est[0]       = 1.0;
01216     for (i=0; i<VMAX_BDTYPE; i++) thee->bmap[0][i] = i;
01217
01218     /* Manifold-specific function pointers */
01219     thee->bisectEdge  = Vfetk_PDE_bisectEdge;
01220     thee->mapBoundary = Vfetk_PDE_mapBoundary;
01221     thee->markSimplex = Vfetk_PDE_markSimplex;
01222     thee->oneChart    = Vfetk_PDE_oneChart;
01223
01224     /* Element-specific function pointers */
01225     thee->simplexBasisInit = Vfetk_PDE_simplexBasisInit;
01226     thee->simplexBasisForm = Vfetk_PDE_simplexBasisForm;
01227
01228     return 1;
01229 }
01230
01231 VPUBLIC void Vfetk_PDE_dtor(PDE **thee) {
01232
01233     if ((*thee) != VNULL) {
01234         Vfetk_PDE_dtor2(*thee);
01235         /* TODO: The following line is commented out because at the moment,
01236             there is a seg fault when deallocating at the end of a run. Since
01237             this routine is called only once at the very end, we'll leave it
01238             commented out. However, this could be a memory leak.
01239          */
01240         /* Vmem_free(var.fetk->vmem, 1, sizeof(PDE), (void **)thee); */
01241         (*thee) = VNULL;
01242     }
01243
01244 }
01245
01246 VPUBLIC void Vfetk_PDE_dtor2(PDE *thee) {
01247     var.fetk = VNULL;
01248 }
01249
01250 VPRIVATE double smooth(int nverts, double dist[VAPBS_NVS], double coeff[VAPBS_NVS], int meth) {
01251
01252     int i;
01253     double weight;
01254     double num = 0.0;
01255     double den = 0.0;
01256
01257     for (i=0; i<nverts; i++) {
01258         if (dist[i] < VSMALL) return coeff[i];
01259         weight = 1.0/dist[i];
01260         if (meth == 0) {
01261             num += (weight * coeff[i]);
01262             den += weight;
01263         } else if (meth == 1) {
01264             /* Small coefficients reset the average to 0; we need to break out
01265              * of the loop */
01266             if (coeff[i] < VSMALL) {
01267                 num = 0.0;
01268                 break;
01269             } else {
01270                 num += weight; den += (weight/coeff[i]);
01271             }
01272         } else VASSERT(0);
01273     }
01274
01275     return (num/den);
```

```
01276
01277 }
01278
01279 VPRIVATE double diel() {
01280
01281     int i, j;
01282     double eps, epsp, epsw, dist[5], coeff[5], srad, swin, *vx;
01283     Vsurf_Meth srfm;
01284     Vacc *acc;
01285     PBEparm *pbeparm;
01286
01287     epsp = Vpbe_getSoluteDiel(var.fetk->pbe);
01288     epsw = Vpbe_getSolventDiel(var.fetk->pbe);
01289     VASSERT(var.fetk->pbeparm != VNULL);
01290     pbeparm = var.fetk->pbeparm;
01291     srfm = pbeparm->srfm;
01292     srad = pbeparm->srad;
01293     swin = pbeparm->swin;
01294     acc = var.fetk->pbe->acc;
01295
01296     eps = 0;
01297
01298     if (VABS(epsp - epsw) < VSMALL) return epsp;
01299     switch (srfm) {
01300         case VSM_MOL:
01301             eps = ((epsw-epsp)*Vacc_molAcc(acc, var.xq, srad) + epsp);
01302             break;
01303         case VSM_MOLSMOOTH:
01304             for (i=0; i<var.nverts; i++) {
01305                 dist[i] = 0;
01306                 vx = var.vx[i];
01307                 for (j=0; j<3; j++) {
01308                     dist[i] += VSQR(var.xq[j] - vx[j]);
01309                 }
01310                 dist[i] = VSQRT(dist[i]);
01311                 coeff[i] = (epsw-epsp)*Vacc_molAcc(acc, var.xq, srad) + epsp;
01312             }
01313             eps = smooth(var.nverts, dist, coeff, 1);
01314             break;
01315         case VSM_SPLINE:
01316             eps = ((epsw-epsp)*Vacc_splineAcc(acc, var.xq, swin, 0.0) + epsp);
01317             break;
01318         default:
01319             Vnm_print(2, "Undefined surface method (%d)!\n", srfm);
01320             VASSERT(0);
01321     }
01322
01323     return eps;
01324 }
01325
01326 VPRIVATE double ionacc() {
01327
01328     int i, j;
01329     double dist[5], coeff[5], irad, swin, *vx, accval;
01330     Vsurf_Meth srfm;
01331     Vacc *acc = VNULL;
01332     PBEparm *pbeparm = VNULL;
01333
01334     VASSERT(var.fetk->pbeparm != VNULL);
01335     pbeparm = var.fetk->pbeparm;
01336     srfm = pbeparm->srfm;
01337     irad = Vpbe_getMaxIonRadius(var.fetk->pbe);
01338     swin = pbeparm->swin;
01339     acc = var.fetk->pbe->acc;
01340
01341     if (var.zks2 < VSMALL) return 0.0;
01342     switch (srfm) {
01343         case VSM_MOL:
01344             accval = Vacc_ivdwAcc(acc, var.xq, irad);
01345             break;
01346         case VSM_MOLSMOOTH:
01347             for (i=0; i<var.nverts; i++) {
01348                 dist[i] = 0;
01349                 vx = var.vx[i];
01350                 for (j=0; j<3; j++) {
01351                     dist[i] += VSQR(var.xq[j] - vx[j]);
01352                 }
01353                 dist[i] = VSQRT(dist[i]);
01354                 coeff[i] = Vacc_ivdwAcc(acc, var.xq, irad);
01355             }
01356             accval = smooth(var.nverts, dist, coeff, 1);
```

```
01357             break;
01358         case VSM_SPLINE:
01359             accval = Vacc_splineAcc(acc, var.xq, swin, irad);
01360             break;
01361         default:
01362             Vnm_print(2, "Undefined surface method (%d)!\n", srfm);
01363             VASSERT(0);
01364     }
01365
01366     return accval;
01367 }
01368
01369 VPRIVATE double debye_U(Vpbe *pbe, int d, double x[]) {
01370
01371     double size, *position, charge, xkappa, eps_w, dist, T, pot, val;
01372     int iatom, i;
01373     Valist *alist;
01374     Vatom *atom;
01375
01376     eps_w = Vpbe_getSolventDiel(pbe);
01377     xkappa = (1.0e10)*Vpbe_getXkappa(pbe);
01378     T = Vpbe_getTemperature(pbe);
01379     alist = Vpbe_getValist(pbe);
01380     val = 0;
01381     pot = 0;
01382
01383     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
01384         atom = Valist_getAtom(alist, iatom);
01385         position = Vatom_getPosition(atom);
01386         charge = Vunit_ec*Vatom_getCharge(atom);
01387         size = (1e-10)*Vatom_getRadius(atom);
01388         dist = 0;
01389         for (i=0; i<d; i++) {
01390             dist += VSQR(position[i] - x[i]);
01391         }
01392         dist = (1.0e-10)*VSQRT(dist);
01393         val = (charge)/(4*VPI*Vunit_eps0*eps_w*dist);
01394         if (xkappa != 0.0) {
01395             val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
01396         }
01397         pot = pot + val;
01398     }
01399     pot = pot*Vunit_ec/(Vunit_kb*T);
01400
01401     return pot;
01402 }
01403
01404 VPRIVATE double debye_Udiff(Vpbe *pbe, int d, double x[]) {
01405
01406     double size, *position, charge, eps_p, dist, T, pot, val;
01407     double Ufull;
01408     int iatom, i;
01409     Valist *alist;
01410     Vatom *atom;
01411
01412     Ufull = debye_U(pbe, d, x);
01413
01414     eps_p = Vpbe_getSoluteDiel(pbe);
01415     T = Vpbe_getTemperature(pbe);
01416     alist = Vpbe_getValist(pbe);
01417     val = 0;
01418     pot = 0;
01419
01420     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
01421         atom = Valist_getAtom(alist, iatom);
01422         position = Vatom_getPosition(atom);
01423         charge = Vunit_ec*Vatom_getCharge(atom);
01424         size = (1e-10)*Vatom_getRadius(atom);
01425         dist = 0;
01426         for (i=0; i<d; i++) {
01427             dist += VSQR(position[i] - x[i]);
01428         }
01429         dist = (1.0e-10)*VSQRT(dist);
01430         val = (charge)/(4*VPI*Vunit_eps0*eps_p*dist);
01431         pot = pot + val;
01432     }
01433     pot = pot*Vunit_ec/(Vunit_kb*T);
01434
01435     pot = Ufull - pot;
01436
01437     return pot;
```

```
01438 }
01439
01440 VPRIVATE void coulomb(Vpbe *pbe, int d, double pt[], double eps, double *U,
01441   double dU[], double *d2U) {
01442
01443     int iatom, i;
01444     double T, pot, fx, fy, fz, x, y, z, scale;
01445     double *position, charge, dist, dist2, val, vec[3], dUold[3], Uold;
01446     Valist *alist;
01447     Vatom *atom;
01448
01449     /* Initialize variables */
01450     T = Vpbe_getTemperature(pbe);
01451     alist = Vpbe_getValist(pbe);
01452     pot = 0;  fx = 0; fy = 0; fz = 0;
01453     x = pt[0]; y = pt[1]; z = pt[2];
01454
01455     /* Calculate */
01456     if (!Vgreen_coulombD(var.green, 1, &x, &y, &z, &pot, &fx, &fy, &fz)) {
01457         Vnm_print(2, "Error calculating Green's function!\n");
01458         VASSERT(0);
01459     }
01460
01461
01462     /* Scale the results */
01463     scale = Vunit_ec/(eps*Vunit_kb*T);
01464     *U = pot*scale;
01465     *d2U = 0.0;
01466     dU[0] = -fx*scale;
01467     dU[1] = -fy*scale;
01468     dU[2] = -fz*scale;
01469
01470 #if 0
01471     /* Compare with old results */
01472     val = 0.0;
01473     Uold = 0.0; dUold[0] = 0.0; dUold[1] = 0.0; dUold[2] = 0.0;
01474     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
01475        atom = Valist_getAtom(alist, iatom);
01476        position = Vatom_getPosition(atom);
01477        charge = Vatom_getCharge(atom);
01478        dist2 = 0;
01479        for (i=0; i<d; i++) {
01480            vec[i] = (position[i] - pt[i]);
01481            dist2 += VSQR(vec[i]);
01482        }
01483        dist = VSQRT(dist2);
01484
01485        /* POTENTIAL */
01486        Uold = Uold + charge/dist;
01487
01488        /* GRADIENT */
01489        for (i=0; i<d; i++) dUold[i] = dUold[i] + vec[i]*charge/(dist2*dist);
01490
01491     }
01492     Uold = Uold*VSQR(Vunit_ec)*(1.0e10)/(4*VPI*Vunit_eps0*eps*Vunit_kb*T);
01493     for (i=0; i<d; i++) {
01494         dUold[i] = dUold[i]*VSQR(Vunit_ec)*(1.0e10)/(4*VPI*Vunit_eps0*eps*Vunit_kb*T);
01495     }
01496
01497     printf("Unew - Uold = %g - %g = %g\n", *U, Uold, (*U - Uold));
01498     printf("||dUnew - dUold||^2 = %g\n", (VSQR(dU[0] - dUold[0])
01499                 + VSQR(dU[1] - dUold[1]) + VSQR(dU[2] - dUold[2])));
01500     printf("dUnew[0] = %g, dUold[0] = %g\n", dU[0], dUold[0]);
01501     printf("dUnew[1] = %g, dUold[1] = %g\n", dU[1], dUold[1]);
01502     printf("dUnew[2] = %g, dUold[2] = %g\n", dU[2], dUold[2]);
01503
01504 #endif
01505
01506 }
01507
01508 VPUBLIC void Vfetk_PDE_initAssemble(PDE *thee, int ip[], double rp[]) {
01509
01510 #if 1
01511     /* Re-initialize the Green's function oracle in case the atom list has
01512      * changed */
01513     if (var.initGreen) {
01514         Vgreen_dtor(&(var.green));
01515         var.initGreen = 0;
01516     }
01517     var.green = Vgreen_ctor(var.fetk->pbe->alist);
01518     var.initGreen = 1;
```

```
01519 #else
01520     if (!var.initGreen) {
01521         var.green = Vgreen_ctor(var.fetk->pbe->alist);
01522         var.initGreen = 1;
01523     }
01524 #endif
01525
01526 }
01527
01528 VPUBLIC void Vfetk_PDE_initElement(PDE *thee, int elementType, int chart,
01529   double tvx[][3], void *data) {
01530
01531     int i, j;
01532     double epsp, epsw;
01533
01534     /* We assume that the simplex has been passed in as the void *data * *
01535      * argument.  Store it */
01536     VASSERT(data != NULL);
01537     var.simp = (SS *)data;
01538
01539     /* save the element type */
01540     var.sType = elementType;
01541
01542     /* Grab the vertices from this simplex */
01543     var.nverts = thee->dim+1;
01544     for (i=0; i<thee->dim+1; i++) var.verts[i] = SS_vertex(var.simp, i);
01545
01546     /* Vertex locations of this simplex */
01547     for (i=0; i<thee->dim+1; i++) {
01548         for (j=0; j<thee->dim; j++) {
01549             var.vx[i][j] = tvx[i][j];
01550         }
01551     }
01552
01553     /* Set the dielectric constant for this element for use in the jump term *
01554      * of the residual-based error estimator.  The value is set to the average
01555      * * value of the vertices */
01556     var.jumpDiel = 0;  /* NOT IMPLEMENTED YET! */
01557 }
01558
01559 VPUBLIC void Vfetk_PDE_initFace(PDE *thee, int faceType, int chart,
01560   double tnvec[]) {
01561
01562     int i;
01563
01564     /* unit normal vector of this face */
01565     for (i=0; i<thee->dim; i++) var.nvec[i] = tnvec[i];
01566
01567     /* save the face type */
01568     var.fType = faceType;
01569 }
01570
01571 VPUBLIC void Vfetk_PDE_initPoint(PDE *thee, int pointType, int chart,
01572   double txq[], double tU[], double tdU[][3]) {
01573
01574     int i, j, ichop;
01575     double u2, coef2, eps_p;
01576     Vhal_PBEType pdetype;
01577     Vpbe *pbe = VNULL;
01578
01579     eps_p = Vpbe_getSoluteDiel(var.fetk->pbe);
01580     pdetype = var.fetk->type;
01581     pbe = var.fetk->pbe;
01582
01583     /* the point, the solution value and gradient, and the Coulomb value and *
01584      * gradient at the point */
01585     if ((pdetype == PBE_LRPBE) || (pdetype == PBE_NRPBE)) {
01586         coulomb(pbe, thee->dim, txq, eps_p, &(var.W), var.dW, &(var.d2W));
01587     }
01588     for (i=0; i<thee->vec; i++) {
01589         var.U[i] = tU[i];
01590         for (j=0; j<thee->dim; j++) {
01591             var.xq[j] = txq[j];
01592             var.dU[i][j] = tdU[i][j];
01593         }
01594     }
01595
01596     /* interior form case */
01597     if (pointType == 0) {
01598
01599         /* Get the dielectric values */
```

```
01600         var.diel  = diel();
01601         var.ionacc  = ionacc();
01602         var.A = var.diel;
01603         var.F = (var.diel - eps_p);
01604 
01605         switch (pdetype) {
01606 
01607             case PBE_LPBE:
01608                 var.DB = var.ionacc*var.zkappa2*var.ionstr;
01609                 var.B = var.DB*var.U[0];
01610                 break;
01611 
01612             case PBE_NPBE:
01613 
01614                 var.B  = 0;
01615                 var.DB  = 0;
01616                 if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01617                     for (i=0; i<var.nion; i++) {
01618                         u2 = -1.0 * var.U[0] * var.ionQ[i];
01619 
01620                         /* NONLINEAR TERM */
01621                         coef2 = -1.0 * var.ionacc * var.zks2 * var.ionConc[i];
01622                         var.B += (coef2 * Vcap_exp(u2, &ichop));
01623                         /* LINEARIZED TERM */
01624                         coef2 = -1.0 * var.ionQ[i] * coef2;
01625                         var.DB += (coef2 * Vcap_exp(u2, &ichop));
01626                     }
01627                 }
01628                 break;
01629 
01630             case PBE_LRPBE:
01631                 var.DB = var.ionacc*var.zkappa2*var.ionstr;
01632                 var.B = var.DB*(var.U[0]+var.W);
01633                 break;
01634 
01635             case PBE_NRPBE:
01636 
01637                 var.B  = 0;
01638                 var.DB  = 0;
01639                 if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01640                     for (i=0; i<var.nion; i++) {
01641                         u2 = -1.0 * (var.U[0] + var.W) * var.ionQ[i];
01642 
01643                         /* NONLINEAR TERM */
01644                         coef2 = -1.0 * var.ionacc * var.zks2 * var.ionConc[i];
01645                         var.B += (coef2 * Vcap_exp(u2, &ichop));
01646 
01647                         /* LINEARIZED TERM */
01648                         coef2 = -1.0 * var.ionQ[i] * coef2;
01649                         var.DB += (coef2 * Vcap_exp(u2, &ichop));
01650                     }
01651                 }
01652                 break;
01653 
01654             case PBE_SMPBE: /* SMPBE Temp */
01655 
01656                 var.B  = 0;
01657                 var.DB  = 0;
01658                 if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
01659                     for (i=0; i<var.nion; i++) {
01660                         u2 = -1.0 * var.U[0] * var.ionQ[i];
01661 
01662                         /* NONLINEAR TERM */
01663                         coef2 = -1.0 * var.ionacc * var.zks2 * var.ionConc[i];
01664                         var.B += (coef2 * Vcap_exp(u2, &ichop));
01665                         /* LINEARIZED TERM */
01666                         coef2 = -1.0 * var.ionQ[i] * coef2;
01667                         var.DB += (coef2 * Vcap_exp(u2, &ichop));
01668                     }
01669                 }
01670                     break;
01671             default:
01672                 Vnm_print(2, "Vfetk_PDE_initPoint:  Unknown PBE type (%d)!\n",
01673                   pdetype);
01674                 VASSERT(0);
01675                 break;
01676         }
01677 
01678 
01679     /* boundary form case */
01680     } else {
```

```
01681 #ifdef DONEUMANN
01682         ;
01683 #else
01684         Vnm_print(2, "Vfetk:  Whoa!  I just got a boundary point to evaluate (%d)!\n", pointType);
01685         Vnm_print(2, "Vfetk:  Did you do that on purpose?\n");
01686 #endif
01687     }
01688
01689 #if 0 /* THIS IS VERY NOISY! */
01690     Vfetk_dumpLocalVar();
01691 #endif
01692
01693 }
01694
01695 VPUBLIC void Vfetk_PDE_Fu(PDE *thee, int key, double F[]) {
01696
01697     //Vnm_print(2, "Vfetk_PDE_Fu:  Setting error to zero!\n");
01698
01699     F[0] = 0.;
01700
01701 }
01702
01703 VPUBLIC double Vfetk_PDE_Fu_v(
01704         PDE *thee,
01705         int key,
01706         double V[],
01707         double dV[][VAPBS_DIM]
01708         ) {
01709
01710     Vhal_PBEType type;
01711     int i;
01712     double value = 0.;
01713
01714     type = var.fetk->type;
01715
01716     /* interior form case */
01717     if (key == 0) {
01718
01719         for (i=0; i<thee->dim; i++) value += ( var.A * var.dU[0][i] * dV[0][i] );
01720         value += var.B * V[0];
01721
01722         if ((type == PBE_LRPBE) || (type == PBE_NRPBE)) {
01723             for (i=0; i<thee->dim; i++) {
01724                 if (var.F > VSMALL) value += (var.F * var.dW[i] * dV[0][i]);
01725             }
01726         }
01727
01728     /* boundary form case */
01729     } else {
01730 #ifdef DONEUMANN
01731         value = 0.0;
01732 #else
01733         Vnm_print(2, "Vfetk:  Whoa! I was just asked to evaluate a boundary weak form for point type
       %d!\n", key);
01734 #endif
01735     }
01736
01737     var.Fu_v = value;
01738     return value;
01739 }
01740
01741 VPUBLIC double Vfetk_PDE_DFu_wv(
01742         PDE *thee,
01743         int key,
01744         double W[],
01745         double dW[][VAPBS_DIM],
01746         double V[],
01747         double dV[][3]
01748         ) {
01749
01750     Vhal_PBEType type;
01751     int i;
01752     double value = 0.;
01753
01754     type = var.fetk->type;
01755
01756     /* Interior form */
01757     if (key == 0) {
01758             value += var.DB * W[0] * V[0];
01759             for (i=0; i<thee->dim; i++) value += ( var.A * dW[0][i] * dV[0][i] );
01760
```

```
01761     /* boundary form case */
01762     } else {
01763 #ifdef DONEUMANN
01764         value = 0.0;
01765 #else
01766         Vnm_print(2, "Vfetk:  Whoa! I was just asked to evaluate a boundary weak form for point type
       %d!\n", key);
01767 #endif
01768     }
01769
01770     var.DFu_wv = value;
01771     return value;
01772 }
01773
01776 #define VRINGMAX 1000
01779 #define VATOMMAX 1000000
01780 VPUBLIC void Vfetk_PDE_delta(PDE *thee, int type, int chart, double txq[],
01781   void *user, double F[]) {
01782
01783     int iatom, jatom, natoms, atomIndex, atomList[VATOMMAX], nAtomList;
01784     int gotAtom, numSring, isimp, ivert, sid;
01785     double *position, charge, phi[VAPBS_NVS], phix[VAPBS_NVS][3], value;
01786     Vatom *atom;
01787     Vhal_PBEType pdetype;
01788     SS *sring[VRINGMAX];
01789     VV *vertex = (VV *)user;
01790
01791     pdetype = var.fetk->type;
01792
01793     F[0] = 0.0;
01794
01795     if ((pdetype == PBE_LPBE) || (pdetype == PBE_NPBE) || (pdetype == PBE_SMPBE) /* SMPBE Added */) {
01796         VASSERT( vertex != VNULL);
01797         numSring = 0;
01798         sring[numSring] = VV_firstSS(vertex);
01799         while (sring[numSring] != VNULL) {
01800             numSring++;
01801             sring[numSring] = SS_link(sring[numSring-1], vertex);
01802         }
01803         VASSERT( numSring > 0 );
01804         VASSERT( numSring <= VRINGMAX );
01805
01806         /* Move around the simplex ring and determine the charge locations */
01807         F[0] = 0.;
01808         charge = 0.;
01809         nAtomList = 0;
01810         for (isimp=0; isimp<numSring; isimp++) {
01811             sid = SS_id(sring[isimp]);
01812             natoms = Vcsm_getNumberAtoms(Vfetk_getVcsm(var.fetk), sid);
01813             for (iatom=0; iatom<natoms; iatom++) {
01814                 /* Get the delta function information * */
01815                 atomIndex = Vcsm_getAtomIndex(Vfetk_getVcsm(var.fetk),
01816                   iatom, sid);
01817                 gotAtom = 0;
01818                 for (jatom=0; jatom<nAtomList; jatom++) {
01819                     if (atomList[jatom] == atomIndex) {
01820                         gotAtom = 1;
01821                         break;
01822                     }
01823                 }
01824                 if (!gotAtom) {
01825                     VASSERT(nAtomList < VATOMMAX);
01826                     atomList[nAtomList] = atomIndex;
01827                     nAtomList++;
01828
01829                     atom = Vcsm_getAtom(Vfetk_getVcsm(var.fetk), iatom, sid);
01830                     charge = Vatom_getCharge(atom);
01831                     position = Vatom_getPosition(atom);
01832
01833                     /* Get the test function value at the delta function I
01834                      * used to do a VASSERT to make sure the point was in the
01835                      * simplex (i.e., make sure round-off error isn't an
01836                      * issue), but round off errors became an issue */
01837                     if (!Gem_pointInSimplexVal(Vfetk_getGem(var.fetk),
01838                       sring[isimp], position, phi, phix)) {
01839                         if (!Gem_pointInSimplex(Vfetk_getGem(var.fetk),
01840                           sring[isimp], position)) {
01841                             Vnm_print(2, "delta: Both Gem_pointInSimplexVal \
01842 and Gem_pointInSimplex detected misplaced point charge!\n");
01843                             Vnm_print(2, "delta: I think you have problems: \
01844 phi = {");
```

```
01845                             for (ivert=0; ivert<Gem_dimVV(Vfetk_getGem(var.fetk)); ivert++) Vnm_print(2,
       "%e ", phi[ivert]);
01846
        Vnm_print(2, "}\n");
01847                         }
01848                     }
01849                     value = 0;
01850                     for (ivert=0; ivert<Gem_dimVV(Vfetk_getGem(var.fetk)); ivert++) {
01851                         if (VV_id(SS_vertex(sring[isimp], ivert)) == VV_id(vertex)) value += phi[ivert];
01852                     }
01853
01854                     F[0] += (value * Vpbe_getZmagic(var.fetk->pbe) * charge);
01855                 } /* if !gotAtom */
01856             } /* for iatom */
01857         } /* for isimp */
01858
01859     } else if ((pdetype == PBE_LRPBE) || (pdetype == PBE_NRPBE)) {
01860         F[0] = 0.0;
01861     } else { VASSERT(0); }
01862
01863     var.delta = F[0];
01864
01865 }
01866
01867 VPUBLIC void Vfetk_PDE_u_D(PDE *thee, int type, int chart, double txq[],
01868   double F[]) {
01869
01870     if ((var.fetk->type == PBE_LPBE) || (var.fetk->type == PBE_NPBE) || (var.fetk->type == PBE_SMPBE) /*
       SMPBE Added */) {
01871         F[0] = debye_U(var.fetk->pbe, thee->dim, txq);
01872     } else if ((var.fetk->type == PBE_LRPBE) || (var.fetk->type == PBE_NRPBE)) {
01873         F[0] = debye_Udiff(var.fetk->pbe, thee->dim, txq);
01874     } else VASSERT(0);
01875
01876     var.u_D = F[0];
01877
01878 }
01879
01886 VPUBLIC void Vfetk_PDE_u_T(PDE *thee, int type, int chart, double txq[],
01887   double F[]) {
01888 /*VPUBLIC void Vfetk_PDE_u_T(sPDE *thee,
01889                            int type,
01890                            int chart,
01891                            double txq[],
01892                            double F[],
01893                            double dF[][3]
01894                           ) { */
01895
01896     F[0] = 0.0;
01897     var.u_T = F[0];
01898
01899 }
01900
01901
01902 VPUBLIC void Vfetk_PDE_bisectEdge(int dim, int dimII, int edgeType,
01903   int chart[], double vx[][3]) {
01904
01905     int i;
01906
01907     for (i=0; i<dimII; i++) vx[2][i] = .5 * (vx[0][i] + vx[1][i]);
01908     chart[2] = chart[0];
01909
01910 }
01911
01912 VPUBLIC void Vfetk_PDE_mapBoundary(int dim, int dimII, int vertexType,
01913   int chart, double vx[3]) {
01914
01915 }
01916
01917 VPUBLIC int Vfetk_PDE_markSimplex(int dim, int dimII, int simplexType,
01918   int faceType[VAPBS_NVS], int vertexType[VAPBS_NVS], int chart[], double vx[][3],
01919   void *simplex) {
01920
01921     double targetRes, edgeLength, srad, swin, myAcc, refAcc;
01922     int i, natoms;
01923     Vsurf_Meth srfm;
01924     Vhal_PBEType type;
01925     FEMparm *feparm = VNULL;
01926     PBEparm *pbeparm = VNULL;
01927     Vpbe *pbe = VNULL;
01928     Vacc *acc = VNULL;
```

```
01929     Vcsm *csm = VNULL;
01930     SS *simp = VNULL;
01931
01932     VASSERT(var.fetk->feparm != VNULL);
01933     feparm = var.fetk->feparm;
01934     VASSERT(var.fetk->pbeparm != VNULL);;
01935     pbeparm = var.fetk->pbeparm;
01936     pbe = var.fetk->pbe;
01937     csm = Vfetk_getVcsm(var.fetk);
01938     acc = pbe->acc;
01939     targetRes = feparm->targetRes;
01940     srfm = pbeparm->srfm;
01941     srad = pbeparm->srad;
01942     swin = pbeparm->swin;
01943     simp = (SS *)simplex;
01944     type = var.fetk->type;
01945
01946     /* Check to see if this simplex is smaller than the target size */
01947     /* NAB WARNING:  I am providing face=-1 here to conform to the new MC API; however, I'm not sure if
       this is the correct behavior. */
01948     Gem_longestEdge(var.fetk->gm, simp, -1, &edgeLength);
01949     if (edgeLength < targetRes) return 0;
01950
01951     /* For non-regularized PBE, check charge-simplex map */
01952     if ((type == PBE_LPBE) || (type == PBE_NPBE) || (type == PBE_SMPBE)  /* SMPBE Added */) {
01953         natoms = Vcsm_getNumberAtoms(csm, SS_id(simp));
01954         if (natoms > 0) {
01955             return 1;
01956         }
01957     }
01958
01959     /* We would like to resolve the mesh between the van der Waals surface the
01960      * max distance from this surface where there could be coefficient
01961      * changes */
01962     switch(srfm) {
01963         case VSM_MOL:
01964             refAcc = Vacc_molAcc(acc, vx[0], srad);
01965             for (i=1; i<(dim+1); i++) {
01966                 myAcc = Vacc_molAcc(acc, vx[i], srad);
01967                 if (myAcc != refAcc) {
01968                     return 1;
01969                 }
01970             }
01971             break;
01972         case VSM_MOLSMOOTH:
01973             refAcc = Vacc_molAcc(acc, vx[0], srad);
01974             for (i=1; i<(dim+1); i++) {
01975                 myAcc = Vacc_molAcc(acc, vx[i], srad);
01976                 if (myAcc != refAcc) {
01977                     return 1;
01978                 }
01979             }
01980             break;
01981         case VSM_SPLINE:
01982             refAcc = Vacc_splineAcc(acc, vx[0], swin, 0.0);
01983             for (i=1; i<(dim+1); i++) {
01984                 myAcc = Vacc_splineAcc(acc, vx[i], swin, 0.0);
01985                 if (myAcc != refAcc) {
01986                     return 1;
01987                 }
01988             }
01989             break;
01990         default:
01991             VASSERT(0);
01992             break;
01993     }
01994
01995     return 0;
01996 }
01997
01998 VPUBLIC void Vfetk_PDE_oneChart(int dim, int dimII, int objType, int chart[],
01999   double vx[][3], int dimV) {
02000
02001 }
02002
02003 VPUBLIC double Vfetk_PDE_Ju(PDE *thee, int key) {
02004
02005     int i, ichop;
02006     double dielE, qmE, coef2, u2;
02007     double value = 0.;
02008     Vhal_PBEType type;
```

```
02009
02010     type = var.fetk->type;
02011
02012     /* interior form case */
02013     if (key == 0) {
02014         dielE = 0;
02015         for (i=0; i<3; i++) dielE += VSQR(var.dU[0][i]);
02016         dielE = dielE*var.diel;
02017
02018         switch (type) {
02019             case PBE_LPBE:
02020                 if ((var.ionacc > VSMALL) && (var.zkappa2 > VSMALL)) {
02021                     qmE = var.ionacc*var.zkappa2*VSQR(var.U[0]);
02022                 } else qmE = 0;
02023                 break;
02024             case PBE_NPBE:
02025                 if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
02026                     qmE = 0.;
02027                     for (i=0; i<var.nion; i++) {
02028                         coef2 = var.ionacc * var.zks2 * var.ionConc[i] * var.ionQ[i];
02029                         u2 = -1.0 * (var.U[0]) * var.ionQ[i];
02030                         qmE += (coef2 * (Vcap_exp(u2, &ichop) - 1.0));
02031                     }
02032                 } else qmE = 0;
02033                 break;
02034             case PBE_LRPBE:
02035                 if ((var.ionacc > VSMALL) && (var.zkappa2 > VSMALL)) {
02036                     qmE = var.ionacc*var.zkappa2*VSQR((var.U[0] + var.W));
02037                 } else qmE = 0;
02038                 break;
02039             case PBE_NRPBE:
02040                 if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
02041                     qmE = 0.;
02042                     for (i=0; i<var.nion; i++) {
02043                         coef2 = var.ionacc * var.zks2 * var.ionConc[i] * var.ionQ[i];
02044                         u2 = -1.0 * (var.U[0] + var.W) * var.ionQ[i];
02045                         qmE += (coef2 * (Vcap_exp(u2, &ichop) - 1.0));
02046                     }
02047                 } else qmE = 0;
02048                 break;
02049             case PBE_SMPBE: /* SMPBE Temp */
02050                 if ((var.ionacc > VSMALL) && (var.zks2 > VSMALL)) {
02051                     qmE = 0.;
02052                     for (i=0; i<var.nion; i++) {
02053                         coef2 = var.ionacc * var.zks2 * var.ionConc[i] * var.ionQ[i];
02054                         u2 = -1.0 * (var.U[0]) * var.ionQ[i];
02055                         qmE += (coef2 * (Vcap_exp(u2, &ichop) - 1.0));
02056                     }
02057                 } else qmE = 0;
02058                 break;
02059             default:
02060                 Vnm_print(2, "Vfetk_PDE_Ju:  Invalid PBE type (%d)!\n", type);
02061                 VASSERT(0);
02062                 break;
02063         }
02064
02065         value = 0.5*(dielE + qmE)/Vpbe_getZmagic(var.fetk->pbe);
02066
02067     /* boundary form case */
02068     } else if (key == 1) {
02069         value = 0.0;
02070
02071     /* how did we get here? */
02072     } else VASSERT(0);
02073
02074     return value;
02075
02076 }
02077
02078 VPUBLIC void Vfetk_externalUpdateFunction(SS **simps, int num) {
02079
02080     Vcsm *csm = VNULL;
02081     int rc;
02082
02083     VASSERT(var.fetk != VNULL);
02084     csm = Vfetk_getVcsm(var.fetk);
02085     VASSERT(csm != VNULL);
02086
02087     rc = Vcsm_update(csm, simps, num);
02088
02089     if (!rc) {
```

```
02090         Vnm_print(2, "Error while updating charge-simplex map!\n");
02091         VASSERT(0);
02092     }
02093 }
02094
02095 VPRIVATE void polyEval(int numP, double p[], double c[][VMAXP], double xv[]) {
02096     int i;
02097     double x, y, z;
02098
02099     x = xv[0];
02100     y = xv[1];
02101     z = xv[2];
02102     for (i=0; i<numP; i++) {
02103         p[i] = c[i][0]
02104              + c[i][1]  * x
02105              + c[i][2]  * y
02106              + c[i][3]  * z
02107              + c[i][4]  * x*x
02108              + c[i][5]  * y*y
02109              + c[i][6]  * z*z
02110              + c[i][7]  * x*y
02111              + c[i][8]  * x*z
02112              + c[i][9]  * y*z
02113              + c[i][10] * x*x*x
02114              + c[i][11] * y*y*y
02115              + c[i][12] * z*z*z
02116              + c[i][13] * x*x*y
02117              + c[i][14] * x*x*z
02118              + c[i][15] * x*y*y
02119              + c[i][16] * y*y*z
02120              + c[i][17] * x*z*z
02121              + c[i][18] * y*z*z;
02122     }
02123 }
02124
02125 VPRIVATE void setCoef(int numP, double c[][VMAXP], double cx[][VMAXP],
02126   double cy[][VMAXP], double cz[][VMAXP], int ic[][VMAXP], int icx[][VMAXP],
02127   int icy[][VMAXP], int icz[][VMAXP]) {
02128
02129     int i, j;
02130     for (i=0; i<numP; i++) {
02131         for (j=0; j<VMAXP; j++) {
02132             c[i][j]  = 0.5 * (double)ic[i][j];
02133             cx[i][j] = 0.5 * (double)icx[i][j];
02134             cy[i][j] = 0.5 * (double)icy[i][j];
02135             cz[i][j] = 0.5 * (double)icz[i][j];
02136         }
02137     }
02138 }
02139
02140 VPUBLIC int Vfetk_PDE_simplexBasisInit(int key, int dim, int comp, int *ndof,
02141   int dof[]) {
02142
02143     int qorder, bump, dimIS[VAPBS_NVS];
02144
02145     /* necessary quadrature order to return at the end */
02146     qorder = P_DEG;
02147
02148     /* deal with bump function requests */
02149     if ((key == 0) || (key == 1)) {
02150         bump = 0;
02151     } else if ((key == 2) || (key == 3)) {
02152         bump = 1;
02153     } else { VASSERT(0); }
02154
02155     /* for now use same element for all components, both trial and test */
02156     if (dim==2) {
02157         /* 2D simplex dimensions */
02158         dimIS[0] = 3;  /* number of vertices             */
02159         dimIS[1] = 3;  /* number of edges                */
02160         dimIS[2] = 0;  /* number of faces (3D only)      */
02161         dimIS[3] = 1;  /* number of simplices (always=1) */
02162         if (bump==0) {
02163             if (P_DEG==1) {
02164                 init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
02165             } else if (P_DEG==2) {
02166                 init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
02167             } else if (P_DEG==3) {
02168                 init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
02169             } else Vnm_print(2, "..bad order..");
02170         } else if (bump==1) {
```

```
02171             if (P_DEG==1) {
02172                 init_2DP1(dimIS, ndof, dof, c, cx, cy, cz);
02173             } else Vnm_print(2, "..bad order..");
02174         } else Vnm_print(2, "..bad bump..");
02175     } else if (dim==3) {
02176         /* 3D simplex dimensions */
02177         dimIS[0] = 4;  /* number of vertices             */
02178         dimIS[1] = 6;  /* number of edges                */
02179         dimIS[2] = 4;  /* number of faces (3D only)      */
02180         dimIS[3] = 1;  /* number of simplices (always=1) */
02181         if (bump==0) {
02182             if (P_DEG==1) {
02183                 init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
02184             } else if (P_DEG==2) {
02185                 init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
02186             } else if (P_DEG==3) {
02187                 init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
02188             } else Vnm_print(2, "..bad order..");
02189         } else if (bump==1) {
02190             if (P_DEG==1) {
02191                 init_3DP1(dimIS, ndof, dof, c, cx, cy, cz);
02192             } else Vnm_print(2, "..bad order..");
02193         } else Vnm_print(2, "..bad bump..");
02194     } else Vnm_print(2, "..bad dimension..");
02195
02196     /* save number of DF */
02197     numP = *ndof;
02198
02199     /* return the required quarature order */
02200     return qorder;
02201 }
02202
02203 VPUBLIC void Vfetk_PDE_simplexBasisForm(int key, int dim, int comp, int pdkey,
02204   double xq[], double basis[]) {
02205
02206     if (pdkey == 0) {
02207         polyEval(numP, basis, c, xq);
02208     } else if (pdkey == 1) {
02209         polyEval(numP, basis, cx, xq);
02210     } else if (pdkey == 2) {
02211         polyEval(numP, basis, cy, xq);
02212     } else if (pdkey == 3) {
02213         polyEval(numP, basis, cz, xq);
02214     } else { VASSERT(0); }
02215 }
02216
02217 VPRIVATE void init_2DP1(int dimIS[], int *ndof, int dof[], double c[][VMAXP],
02218   double cx[][VMAXP], double cy[][VMAXP], double cz[][VMAXP]) {
02219
02220     int i;
02221
02222     /* dof number and locations */
02223     dof[0] = 1;
02224     dof[1] = 0;
02225     dof[2] = 0;
02226     dof[3] = 0;
02227     *ndof  = 0;
02228     for (i=0; i<VAPBS_NVS; i++) *ndof += dimIS[i] * dof[i];
02229     VASSERT( *ndof == dim_2DP1 );
02230     VASSERT( *ndof <= VMAXP );
02231
02232     /* coefficients of the polynomials */
02233     setCoef( *ndof, c, cx, cy, cz, lgr_2DP1, lgr_2DP1x, lgr_2DP1y, lgr_2DP1z );
02234 }
02235
02236 VPRIVATE void init_3DP1(int dimIS[], int *ndof, int dof[], double c[][VMAXP],
02237   double cx[][VMAXP], double cy[][VMAXP], double cz[][VMAXP]) {
02238
02239     int i;
02240
02241     /* dof number and locations */
02242     dof[0] = 1;
02243     dof[1] = 0;
02244     dof[2] = 0;
02245     dof[3] = 0;
02246     *ndof  = 0;
02247     for (i=0; i<VAPBS_NVS; i++) *ndof += dimIS[i] * dof[i];
02248     VASSERT( *ndof == dim_3DP1 );
02249     VASSERT( *ndof <= VMAXP );
02250
02251     /* coefficients of the polynomials */
```

```
02252     setCoef( *ndof, c, cx, cy, cz, lgr_3DP1, lgr_3DP1x, lgr_3DP1y, lgr_3DP1z );
02253 }
02254
02255 VPUBLIC void Vfetk_dumpLocalVar() {
02256
02257     int i;
02258
02259     Vnm_print(1, "DEBUG: nvec = (%g, %g, %g)\n", var.nvec[0], var.nvec[1],
02260       var.nvec[2]);
02261     Vnm_print(1, "DEBUG: nverts = %d\n", var.nverts);
02262     for (i=0; i<var.nverts; i++) {
02263         Vnm_print(1, "DEBUG: verts[%d] ID = %d\n", i, VV_id(var.verts[i]));
02264         Vnm_print(1, "DEBUG: vx[%d] = (%g, %g, %g)\n", i, var.vx[i][0],
02265           var.vx[i][1], var.vx[i][2]);
02266     }
02267     Vnm_print(1, "DEBUG: simp ID = %d\n", SS_id(var.simp));
02268     Vnm_print(1, "DEBUG: sType = %d\n", var.sType);
02269     Vnm_print(1, "DEBUG: fType = %d\n", var.fType);
02270     Vnm_print(1, "DEBUG: xq = (%g, %g, %g)\n", var.xq[0], var.xq[1], var.xq[2]);
02271     Vnm_print(1, "DEBUG: U[0] = %g\n", var.U[0]);
02272     Vnm_print(1, "DEBUG: dU[0] = (%g, %g, %g)\n", var.dU[0][0], var.dU[0][1],
02273       var.dU[0][2]);
02274     Vnm_print(1, "DEBUG: W = %g\n", var.W);
02275     Vnm_print(1, "DEBUG: d2W = %g\n", var.d2W);
02276     Vnm_print(1, "DEBUG: dW = (%g, %g, %g)\n", var.dW[0], var.dW[1], var.dW[2]);
02277     Vnm_print(1, "DEBUG: diel = %g\n", var.diel);
02278     Vnm_print(1, "DEBUG: ionacc = %g\n", var.ionacc);
02279     Vnm_print(1, "DEBUG: A = %g\n", var.A);
02280     Vnm_print(1, "DEBUG: F = %g\n", var.F);
02281     Vnm_print(1, "DEBUG: B = %g\n", var.B);
02282     Vnm_print(1, "DEBUG: DB = %g\n", var.DB);
02283     Vnm_print(1, "DEBUG: nion = %d\n", var.nion);
02284     for (i=0; i<var.nion; i++) {
02285         Vnm_print(1, "DEBUG: ionConc[%d] = %g\n", i, var.ionConc[i]);
02286         Vnm_print(1, "DEBUG: ionQ[%d] = %g\n", i, var.ionQ[i]);
02287         Vnm_print(1, "DEBUG: ionRadii[%d] = %g\n", i, var.ionRadii[i]);
02288     }
02289     Vnm_print(1, "DEBUG: zkappa2 = %g\n", var.zkappa2);
02290     Vnm_print(1, "DEBUG: zks2 = %g\n", var.zks2);
02291     Vnm_print(1, "DEBUG: Fu_v = %g\n", var.Fu_v);
02292     Vnm_print(1, "DEBUG: DFu_wv = %g\n", var.DFu_wv);
02293     Vnm_print(1, "DEBUG: delta = %g\n", var.delta);
02294     Vnm_print(1, "DEBUG: u_D = %g\n", var.u_D);
02295     Vnm_print(1, "DEBUG: u_T = %g\n", var.u_T);
02296
02297 };
02298
02299 VPUBLIC int Vfetk_fillArray(Vfetk *thee, Bvec *vec, Vdata_Type type) {
02300
02301     int i, j, ichop;
02302     double coord[3], chi, q, conc, val;
02303     VV *vert;
02304     Bvec *u, *u_d;
02305     AM *am;
02306     Gem *gm;
02307     PBEparm *pbeparm;
02308     Vacc *acc;
02309     Vpbe *pbe;
02310
02311     gm = thee->gm;
02312     am = thee->am;
02313     pbe = thee->pbe;
02314     pbeparm = thee->pbeparm;
02315     acc = pbe->acc;
02316
02317     /* Make sure vec has enough rows to accomodate the vertex data */
02318     if (Bvec_numRB(vec, 0) != Gem_numVV(gm)) {
02319         Vnm_print(2, "Vfetk_fillArray:  insufficient space in Bvec!\n");
02320         Vnm_print(2, "Vfetk_fillArray:  Have %d, need %d!\n", Bvec_numRB(vec, 0),
02321           Gem_numVV(gm));
02322         return 0;
02323     }
02324
02325     switch (type) {
02326
02327         case VDT_CHARGE:
02328             Vnm_print(2, "Vfetk_fillArray:  can't write out charge distribution!\n");
02329             return 0;
02330             break;
02331
02332         case VDT_POT:
```

```
02333             u = am->u;
02334             u_d = am->ud;
02335             /* Copy in solution */
02336             Bvec_copy(vec, u);
02337             /* Add dirichlet condition */
02338             Bvec_axpy(vec, u_d, 1.0);
02339             break;
02340
02341         case VDT_SMOL:
02342             for (i=0; i<Gem_numVV(gm); i++) {
02343                 vert = Gem_VV(gm, i);
02344                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02345                 chi = Vacc_molAcc(acc, coord, pbe->solventRadius);
02346                 Bvec_set(vec, i, chi);
02347             }
02348             break;
02349
02350         case VDT_SSPL:
02351             for (i=0; i<Gem_numVV(gm); i++) {
02352                 vert = Gem_VV(gm, i);
02353                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02354                 chi = Vacc_splineAcc(acc, coord, pbeparm->swin, 0.0);
02355                 Bvec_set(vec, i, chi);
02356             }
02357             break;
02358
02359         case VDT_VDW:
02360             for (i=0; i<Gem_numVV(gm); i++) {
02361                 vert = Gem_VV(gm, i);
02362                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02363                 chi = Vacc_vdwAcc(acc, coord);
02364                 Bvec_set(vec, i, chi);
02365             }
02366             break;
02367
02368         case VDT_IVDW:
02369             for (i=0; i<Gem_numVV(gm); i++) {
02370                 vert = Gem_VV(gm, i);
02371                 for (j=0; j<3; j++) coord[j] = VV_coord(vert, j);
02372                 chi = Vacc_ivdwAcc(acc, coord, pbe->maxIonRadius);
02373                 Bvec_set(vec, i, chi);
02374             }
02375             break;
02376
02377         case VDT_LAP:
02378             Vnm_print(2, "Vfetk_fillArray:  can't write out Laplacian!\n");
02379             return 0;
02380             break;
02381
02382         case VDT_EDENS:
02383             Vnm_print(2, "Vfetk_fillArray:  can't write out energy density!\n");
02384             return 0;
02385             break;
02386
02387         case VDT_NDENS:
02388             u = am->u;
02389             u_d = am->ud;
02390             /* Copy in solution */
02391             Bvec_copy(vec, u);
02392             /* Add dirichlet condition */
02393             Bvec_axpy(vec, u_d, 1.0);
02394             /* Load up ions */
02395             ichop = 0;
02396             for (i=0; i<Gem_numVV(gm); i++) {
02397                 val = 0;
02398                 for (j=0; j<pbe->numIon; j++) {
02399                     q = pbe->ionQ[j];
02400                     conc = pbe->ionConc[j];
02401                     if (thee->type == PBE_NPBE || thee->type == PBE_SMPBE /* SMPBE Added */) {
02402                         val += (conc*Vcap_exp(-q*Bvec_val(vec, i), &ichop));
02403                     } else if (thee->type == PBE_LPBE) {
02404                         val += (conc * ( 1 - q*Bvec_val(vec, i)));
02405                     }
02406                 }
02407                 Bvec_set(vec, i, val);
02408             }
02409             break;
02410
02411         case VDT_QDENS:
02412             u = am->u;
02413             u_d = am->ud;
```

```
02414             /* Copy in solution */
02415             Bvec_copy(vec, u);
02416             /* Add dirichlet condition */
02417             Bvec_axpy(vec, u_d, 1.0);
02418             /* Load up ions */
02419             ichop = 0;
02420             for (i=0; i<Gem_numVV(gm); i++) {
02421                 val = 0;
02422                 for (j=0; j<pbe->numIon; j++) {
02423                     q = pbe->ionQ[j];
02424                     conc = pbe->ionConc[j];
02425                     if (thee->type == PBE_NPBE || thee->type == PBE_SMPBE /* SMPBE Added */) {
02426                         val += (q*conc*Vcap_exp(-q*Bvec_val(vec, i), &ichop));
02427                     } else if (thee->type == PBE_LPBE) {
02428                         val += (q*conc*(1 - q*Bvec_val(vec, i)));
02429                     }
02430                 }
02431                 Bvec_set(vec, i, val);
02432             }
02433             break;
02434 
02435         case VDT_DIELX:
02436             Vnm_print(2, "Vfetk_fillArray:  can't write out x-shifted diel!\n");
02437             return 0;
02438             break;
02439 
02440         case VDT_DIELY:
02441             Vnm_print(2, "Vfetk_fillArray:  can't write out y-shifted diel!\n");
02442             return 0;
02443             break;
02444 
02445         case VDT_DIELZ:
02446             Vnm_print(2, "Vfetk_fillArray:  can't write out z-shifted diel!\n");
02447             return 0;
02448             break;
02449 
02450         case VDT_KAPPA:
02451             Vnm_print(2, "Vfetk_fillArray:  can't write out kappa!\n");
02452             return 0;
02453             break;
02454 
02455         default:
02456             Vnm_print(2, "Vfetk_fillArray:  invalid data type (%d)!\n", type);
02457             return 0;
02458             break;
02459     }
02460 
02461     return 1;
02462 }
02463 
02464 VPUBLIC int Vfetk_write(Vfetk *thee,  const char *iodev, const char *iofmt,
02465   const char *thost, const char *fname, Bvec *vec, Vdata_Format format) {
02466 
02467     int i, j, ichop;
02468     Aprx *aprx;
02469     Gem *gm;
02470     Vio *sock;
02471 
02472     VASSERT(thee != VNULL);
02473     aprx = thee->aprx;
02474     gm = thee->gm;
02475 
02476     sock = Vio_ctor(iodev,iofmt,thost,fname,"w");
02477     if (sock == VNULL) {
02478         Vnm_print(2, "Vfetk_write: Problem opening virtual socket %s\n",
02479           fname);
02480         return 0;
02481     }
02482     if (Vio_connect(sock, 0) < 0) {
02483         Vnm_print(2, "Vfetk_write: Problem connecting to virtual socket %s\n",
02484           fname);
02485         return 0;
02486     }
02487 
02488     /* Make sure vec has enough rows to accomodate the vertex data */
02489     if (Bvec_numRB(vec, 0) != Gem_numVV(gm)) {
02490         Vnm_print(2, "Vfetk_fillArray:  insufficient space in Bvec!\n");
02491         Vnm_print(2, "Vfetk_fillArray:  Have %d, need %d!\n", Bvec_numRB(vec, 0),
02492           Gem_numVV(gm));
02493         return 0;
02494     }
```

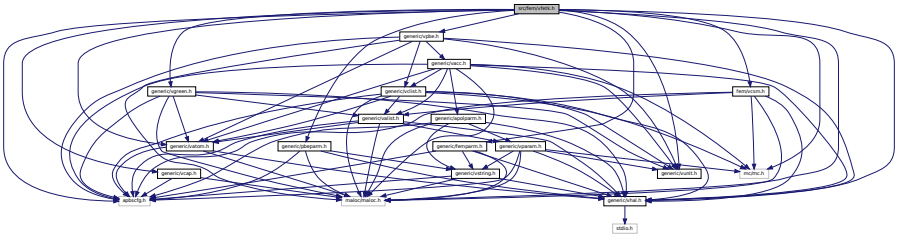
```
02495
02496     switch (format) {
02497
02498         case VDF_DX:
02499             Aprx_writeSOL(aprx, sock, vec, "DX");
02500             break;
02501         case VDF_AVS:
02502             Aprx_writeSOL(aprx, sock, vec, "UCD");
02503             break;
02504         case VDF_UHBD:
02505             Vnm_print(2, "Vfetk_write:  UHBD format not supported!\n");
02506             return 0;
02507         default:
02508             Vnm_print(2, "Vfetk_write:  Invalid data format (%d)!\n", format);
02509             return 0;
02510     }
02511
02512
02513     Vio_connectFree(sock);
02514     Vio_dtor(&sock);
02515
02516     return 1;
02517 }
```

9.9 src/fem/vfetk.h File Reference

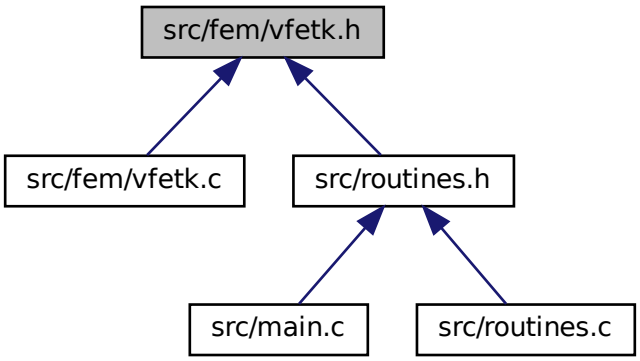
Contains declarations for class Vfetk.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "mc/mc.h"
#include "generic/vhal.h"
#include "generic/vatom.h"
#include "generic/vpbe.h"
#include "generic/vunit.h"
#include "generic/vgreen.h"
#include "generic/vcap.h"
#include "generic/pbeparm.h"
#include "generic/femparm.h"
#include "fem/vcsm.h"
```

Include dependency graph for vfetk.h:



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



Data Structures

- struct 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.
- VPUBLIC 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:
- 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:
- 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:
- 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:

- 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

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) 2010-2020 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 "apbscfg.h"
00066
00067 #include "maloc/maloc.h"
00068 #include "mc/mc.h"
00069
00070 #include "generic/vhal.h"
00071 #include "generic/vatom.h"
00072 // #include "generic/valist.h"
00073 #include "generic/vpbe.h"
00074 #include "generic/vunit.h"
00075 #include "generic/vgreen.h"
00076 #include "generic/vcap.h"
00077 #include "generic/pbeparm.h"
00078 #include "generic/femparm.h"
00079 #include "fem/vcsm.h"
00080
00086 enum eVfetk_LsolvType {
00087     VLT_SLU=0,
00088     VLT_MG=1,
00089     VLT_CG=2,
00090     VLT_BCG=3
00091 };
00092
00097 typedef enum eVfetk_LsolvType Vfetk_LsolvType;
00098
00099
00104 enum eVfetk_MeshLoad {
00105     VML_DIRICUBE,
00106     VML_NEUMCUBE,
00107     VML_EXTERNAL
00108 };
00109
00114 typedef enum eVfetk_MeshLoad Vfetk_MeshLoad;
00115
00121 enum eVfetk_NsolvType {
00122     VNT_NEW=0,
00123     VNT_INC=1,
00124     VNT_ARC=2
00125 };
00126
00131 typedef enum eVfetk_NsolvType Vfetk_NsolvType;
00132
```

```
00138 enum eVfetk_GuessType {
00139     VGT_ZERO=0,
00140     VGT_DIRI=1,
00141     VGT_PREV=2
00142 };
00143
00148 typedef enum eVfetk_GuessType Vfetk_GuessType;
00149
00155 enum eVfetk_PrecType {
00156     VPT_IDEN=0,
00157     VPT_DIAG=1,
00158     VPT_MG=2
00159 };
00160
00165 typedef enum eVfetk_PrecType Vfetk_PrecType;
00166
00176 struct sVfetk {
00177
00178   Vmem *vmem;
00179   Gem *gm;
00182   AM *am;
00183   Aprx *aprx;
00184   PDE *pde;
00185   Vpbe *pbe;
00186   Vcsm *csm;
00187   Vfetk_LsolvType lkey;
00188   int lmax;
00189   double ltol;
00190   Vfetk_NsolvType nkey;
00191   int nmax;
00192   double ntol;
00193   Vfetk_GuessType gues;
00194   Vfetk_PrecType lprec;
00195   int pjac;
00197   PBEparm *pbeparm;
00198   FEMparm *feparm;
00199   Vhal_PBEType type;
00200   int level;
00202 };
00203
00207 typedef struct sVfetk Vfetk;
00208
00215 struct sVfetk_LocalVar {
00216     double nvec[VAPBS_DIM];
00217     double vx[4][VAPBS_DIM];
00218     double xq[VAPBS_DIM];
00219     double U[MAXV];
00220     double dU[MAXV][VAPBS_DIM];
00221     double W;
00222     double dW[VAPBS_DIM];
00223     double d2W;
00224     int sType;
00225     int fType;
00226     double diel;
00227     double ionacc;
00228     double A;
00229     double F;
00230     double B;
00231     double DB;
00232     double jumpDiel;
00233     Vfetk *fetk;
00234     Vgreen *green;
00235     int initGreen;
00237     SS *simp;
00239     VV *verts[4];
00240     int nverts;
00241     double ionConc[MAXION];
00242     double ionQ[MAXION];
00243     double ionRadii[MAXION];
00244     double zkappa2;
00245     double zks2;
00246     double ionstr;
00247     int nion;
00248     double Fu_v;
00249     double DFu_wv;
00250     double delta;
00251     double u_D;
00252     double u_T;
00253 };
00254
00259 typedef struct sVfetk_LocalVar Vfetk_LocalVar;
```

```
00260
00261 #if !defined(VINLINE_VFETK)
00262
00268     VEXTERNC Gem* Vfetk_getGem(
00269             Vfetk *thee
00270             );
00271
00277     VEXTERNC AM* Vfetk_getAM(
00278             Vfetk *thee
00279             );
00280
00286     VEXTERNC Vpbe* Vfetk_getVpbe(
00287             Vfetk *thee
00288             );
00289
00295     VEXTERNC Vcsm* Vfetk_getVcsm(
00296             Vfetk *thee
00297             );
00298
00305     VEXTERNC int Vfetk_getAtomColor(
00306             Vfetk *thee,
00307             int iatom
00308             );
00309
00310 #else /* if defined(VINLINE_VFETK) */
00311 #   define Vfetk_getGem(thee) ((thee)->gm)
00312 #   define Vfetk_getAM(thee) ((thee)->am)
00313 #   define Vfetk_getVpbe(thee) ((thee)->pbe)
00314 #   define Vfetk_getVcsm(thee) ((thee)->csm)
00315 #   define Vfetk_getAtomColor(thee, iatom) (Vatom_getPartID(Valist_getAtom(Vpbe_getValist(thee->pbe),
      iatom)))
00316 #endif /* if !defined(VINLINE_VFETK) */
00317
00318 /* ////////////////////////////////////////////////////////////////////////////
00319 // Class Vfetk: Non-Inlineable methods (vfetk.c)
00321
00331 VEXTERNC Vfetk* Vfetk_ctor(
00332         Vpbe *pbe, /**< Vpbe (PBE manager object) */
00333         Vhal_PBEType type
00334         );
00335
00345 VEXTERNC int Vfetk_ctor2(
00346         Vfetk *thee,
00347         Vpbe *pbe,
00348         Vhal_PBEType type
00349         );
00350
00356 VEXTERNC void Vfetk_dtor(
00357         Vfetk **thee
00358         );
00359
00365 VEXTERNC void Vfetk_dtor2(
00366         Vfetk *thee
00367         );
00368
00378 VEXTERNC double* Vfetk_getSolution(
00379         Vfetk *thee,
00380         int *length
00381         );
00382
00388 VEXTERNC void Vfetk_setParameters(
00389         Vfetk *thee,
00390         PBEparm *pbeparm,
00391         FEMparm *feparm
00392         );
00393
00412 VEXTERNC double  Vfetk_energy(
00413         Vfetk *thee,
00414         int color,
00418         int nonlin
00420         );
00421
00451 VEXTERNC double  Vfetk_dqmEnergy(
00452         Vfetk *thee,
00453         int color
00457         );
00458
00476 VEXTERNC double Vfetk_qfEnergy(
00477         Vfetk *thee,
00478         int color
00480         );
```

```
00481
00489 VEXTERNC unsigned long int Vfetk_memChk(
00490         Vfetk *thee
00491         );
00492
00508 VEXTERNC void Vfetk_setAtomColors(
00509         Vfetk *thee
00510         );
00511
00520 VEXTERNC void Bmat_printHB(
00521         Bmat *thee,
00522         char *fname
00523         );
00524
00530 VEXTERNC Vrc_Codes Vfetk_genCube(
00531                                  Vfetk *thee,
00532                                  double center[3],
00533                                  double length[3],
00534                                  Vfetk_MeshLoad meshType
00535                                  );
00536
00542 VEXTERNC Vrc_Codes Vfetk_loadMesh(
00543                                   Vfetk *thee,
00544                                   double center[3],
00545                                   double length[3],
00546                                   Vfetk_MeshLoad meshType,
00547                                   Vio *sock
00548                                   );
00549
00556 VEXTERNC PDE* Vfetk_PDE_ctor(
00557         Vfetk *fetk
00558         );
00559
00566 VEXTERNC int Vfetk_PDE_ctor2(
00567         PDE *thee,
00568         Vfetk *fetk
00569         );
00570
00577 VEXTERNC void Vfetk_PDE_dtor(
00578         PDE **thee
00579         );
00580
00587 VEXTERNC void Vfetk_PDE_dtor2(
00588         PDE *thee
00589         );
00590
00596 VEXTERNC void Vfetk_PDE_initAssemble(
00597         PDE *thee,
00598         int ip[],
00599         double rp[]
00600         );
00601
00608 VEXTERNC void Vfetk_PDE_initElement(
00609         PDE *thee,
00610         int elementType,
00611         int chart,
00614         double tvx[][VAPBS_DIM],
00615         void *data
00616         );
00617
00623 VEXTERNC void Vfetk_PDE_initFace(
00624         PDE *thee,
00625         int faceType,
00627         int chart,
00629         double tnvec[]
00630         );
00631
00639 VEXTERNC void Vfetk_PDE_initPoint(
00640         PDE *thee,
00641         int pointType,
00642         int chart,
00644         double txq[],
00645         double tU[],
00646         double tdU[][VAPBS_DIM]
00647         );
00648
00666 VEXTERNC void Vfetk_PDE_Fu(
00667         PDE *thee,
00668         int key,
00670         double F[]
00671         );
```

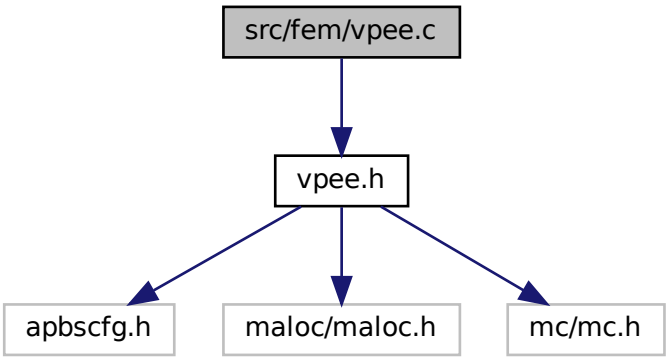
```
00672
00683 VEXTERNC double Vfetk_PDE_Fu_v(
00684         PDE *thee,
00685         int key,
00687         double V[],
00688         double dV[][VAPBS_DIM]
00689         );
00690
00702 VEXTERNC double Vfetk_PDE_DFu_wv(
00703         PDE *thee,
00704         int key,
00706         double W[],
00707         double dW[][VAPBS_DIM],
00708         double V[],
00709         double dV[][VAPBS_DIM]
00710         );
00711
00718 VEXTERNC void Vfetk_PDE_delta(
00719         PDE *thee,
00720         int type,
00721         int chart,
00722         double txq[],
00723         void *user,
00724         double F[]
00725         );
00726
00734 VEXTERNC void Vfetk_PDE_u_D(
00735         PDE *thee,
00736         int type,
00737         int chart,
00738         double txq[],
00739         double F[]
00740         );
00741
00749 VEXTERNC void Vfetk_PDE_u_T(
00750         PDE *thee,
00751         int type,
00752         int chart,
00753         double txq[],
00754         double F[]
00755         );
00756
00762 VEXTERNC void Vfetk_PDE_bisectEdge(
00763         int dim,
00764         int dimII,
00765         int edgeType,
00766         int chart[],
00768         double vx[][VAPBS_DIM]
00769         );
00770
00776 VEXTERNC void Vfetk_PDE_mapBoundary(
00777         int dim,
00778         int dimII,
00779         int vertexType,
00780         int chart,
00781         double vx[VAPBS_DIM]
00782         );
00783
00792 VEXTERNC int Vfetk_PDE_markSimplex(
00793         int dim,
00794         int dimII,
00795         int simplexType,
00796         int faceType[VAPBS_NVS],
00797         int vertexType[VAPBS_NVS],
00798         int chart[],
00799         double vx[][VAPBS_DIM],
00800         void *simplex
00801         );
00802
00808 VEXTERNC void Vfetk_PDE_oneChart(
00809         int dim,
00810         int dimII,
00811         int objType,
00812         int chart[],
00813         double vx[][VAPBS_DIM],
00814         int dimV
00815         );
00816
00826 VEXTERNC double Vfetk_PDE_Ju(
00827         PDE *thee,
00828         int key
```

```
00829         );
00830
00838 VEXTERNC void Vfetk_externalUpdateFunction(
00839         SS **simps,
00841         int num
00842         );
00843
00844
00907 VEXTERNC int Vfetk_PDE_simplexBasisInit(
00908         int key,
00910         int dim,
00911         int comp,
00913         int *ndof,
00914         int dof[]
00915         );
00916
00924 VEXTERNC void Vfetk_PDE_simplexBasisForm(
00925         int key,
00927         int dim,
00928         int comp ,
00929         int pdkey,
00938         double xq[],
00939         double basis[]
00941         );
00942
00948 VEXTERNC void Vfetk_readMesh(
00949         Vfetk *thee,
00950         int skey,
00951         Vio *sock
00952         );
00953
00959 VEXTERNC void Vfetk_dumpLocalVar();
00960
00968 VEXTERNC int Vfetk_fillArray(
00969         Vfetk *thee,
00970         Bvec *vec,
00971         Vdata_Type type
00972         );
00973
00988 VEXTERNC int Vfetk_write(
00989         Vfetk *thee,
00990         const char *iodev,
00992         const char *iofmt,
00994         const char *thost,
00995         const char *fname,
00996         Bvec *vec,
00997         Vdata_Format format
00998         );
00999
01005 VEXTERNC Vrc_Codes Vfetk_loadGem(
01006                                  Vfetk *thee,
01007                                  Gem *gm
01008                                  );
01009
01010
01011 #endif /* ifndef _VFETK_H_ */
```

9.11 src/fem/vpee.c File Reference

Class Vpee methods.

```
#include "vpee.h"
```
Include dependency graph for vpee.c:



Functions

- VPRIVATE int **Vpee_userDefined** (Vpee *thee, SS *sm)
- VPRIVATE int **Vpee_ourSimp** (Vpee *thee, SS *sm, int rcol)
- VEXTERNC double **Aprx_estNonlinResid** (Aprx *thee, SS *sm, Bvec *u, Bvec *ud, Bvec *f)
- VEXTERNC double **Aprx_estLocalProblem** (Aprx *thee, SS *sm, Bvec *u, Bvec *ud, Bvec *f)
- VEXTERNC double **Aprx_estDualProblem** (Aprx *thee, SS *sm, Bvec *u, Bvec *ud, Bvec *f)
- VPUBLIC Vpee * Vpee_ctor (Gem *gm, int localPartID, int killFlag, double killParam)

  Construct the Vpee object.
- VPUBLIC int Vpee_ctor2 (Vpee *thee, Gem *gm, int localPartID, int killFlag, double killParam)

  FORTRAN stub to construct the Vpee object.
- VPUBLIC void Vpee_dtor (Vpee **thee)

  Object destructor.
- VPUBLIC void Vpee_dtor2 (Vpee *thee)

  FORTRAN stub object destructor.
- VPUBLIC int Vpee_markRefine (Vpee *thee, AM *am, int level, int akey, int rcol, double etol, int bkey)

  Mark simplices for refinement based on attenuated error estimates.
- VPUBLIC int Vpee_numSS (Vpee *thee)

  Returns the number of simplices in the local partition.

9.11.1 Detailed Description

Class Vpee methods.

Author

Nathan Baker

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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) 2002-2020, 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.

9.12 vpee.c

```
00001
00057 #include "vpee.h"
00058
00059 VPRIVATE int Vpee_userDefined(Vpee *thee,
00060                               SS *sm
00061                               );
00062 VPRIVATE int Vpee_ourSimp(Vpee *thee,
00063                           SS *sm,
00064                           int rcol
00065                           );
00066 VEXTERNC double Aprx_estNonlinResid(Aprx *thee,
```

```
00067                                   SS *sm,
00068                                   Bvec *u,
00069                                   Bvec *ud,
00070                                   Bvec *f
00071                                   );
00072 VEXTERNC double Aprx_estLocalProblem(Aprx *thee,
00073                                    SS *sm,
00074                                    Bvec *u,
00075                                    Bvec *ud,
00076                                    Bvec *f);
00077 VEXTERNC double Aprx_estDualProblem(Aprx *thee,
00078                                   SS *sm,
00079                                   Bvec *u,
00080                                   Bvec *ud,
00081                                   Bvec *f
00082                                   );
00083
00084 /* ///////////////////////////////////////////////////////////////////////// */
00085 // Class Vpee: Non-inlineable methods
00087
00088 /* ///////////////////////////////////////////////////////////////////////// */
00089 // Routine:  Vpee_ctor
00090 //
00091 // Author:   Nathan Baker
00093 VPUBLIC Vpee* Vpee_ctor(Gem *gm,
00094                         int localPartID,
00095                         int killFlag,
00096                         double killParam
00097                         ) {
00098
00099     Vpee *thee = VNULL;
00100
00101     /* Set up the structure */
00102     thee = Vmem_malloc(VNULL, 1, sizeof(Vpee) );
00103     VASSERT( thee != VNULL);
00104     VASSERT( Vpee_ctor2(thee, gm, localPartID, killFlag, killParam));
00105
00106     return thee;
00107 }
00108
00109 /* ///////////////////////////////////////////////////////////////////////// */
00110 // Routine:  Vpee_ctor2
00111 //
00112 // Author:   Nathan Baker
00114 VPUBLIC int Vpee_ctor2(Vpee *thee,
00115                        Gem *gm,
00116                        int localPartID,
00117                        int killFlag,
00118                        double killParam
00119                        ) {
00120
00121     int ivert,
00122         nLocalVerts;
00123     SS *simp;
00124     VV *vert;
00125     double radius,
00126            dx,
00127            dy,
00128            dz;
00129
00130     VASSERT(thee != VNULL);
00131
00132     /* Sanity check on input values */
00133     if (killFlag == 0) {
00134         Vnm_print(0, "Vpee_ctor2: No error attenuation outside partition.\n");
00135     } else if (killFlag == 1) {
00136         Vnm_print(0, "Vpee_ctor2: Error outside local partition ignored.\n");
00137     } else if (killFlag == 2) {
00138         Vnm_print(0, "Vpee_ctor2: Error ignored outside sphere with radius %4.3f times the radius of the
      circumscribing sphere\n", killParam);
00139         if (killParam < 1.0) {
00140           Vnm_print(2, "Vpee_ctor2: Warning! Parameter killParam = %4.3 < 1.0!\n",
00141             killParam);
00142           Vnm_print(2, "Vpee_ctor2: This may result in non-optimal marking and refinement!\n");
00143         }
00144     } else if (killFlag == 3) {
00145         Vnm_print(0, "Vpee_ctor2: Error outside local partition and immediate neighbors ignored [NOT
      IMPLEMENTED].\n");
00146     } else {
00147         Vnm_print(2, "Vpee_ctor2: UNRECOGNIZED killFlag PARAMETER! BAILING!.\n");
00148         VASSERT(0);
```

```
00149     }
00150
00151     thee->gm = gm;
00152     thee->localPartID = localPartID;
00153     thee->killFlag = killFlag;
00154     thee->killParam = killParam;
00155     thee->mem = Vmem_ctor("APBS::VPEE");
00156
00157     /* Now, figure out the center of geometry for the local partition.  The
00158      * general plan is to loop through the vertices, loop through the
00159      * vertices' simplex lists and find the vertices with simplices containing
00160      * chart values we're interested in. */
00161     thee->localPartCenter[0] = 0.0;
00162     thee->localPartCenter[1] = 0.0;
00163     thee->localPartCenter[2] = 0.0;
00164     nLocalVerts = 0;
00165     for (ivert=0; ivert<Gem_numVV(thee->gm); ivert++) {
00166         vert = Gem_VV(thee->gm, ivert);
00167         simp = VV_firstSS(vert);
00168         VASSERT(simp != VNULL);
00169         while (simp != VNULL) {
00170             if (SS_chart(simp) == thee->localPartID) {
00171                 thee->localPartCenter[0] += VV_coord(vert, 0);
00172                 thee->localPartCenter[1] += VV_coord(vert, 1);
00173                 thee->localPartCenter[2] += VV_coord(vert, 2);
00174                 nLocalVerts++;
00175                 break;
00176             }
00177             simp = SS_link(simp, vert);
00178         }
00179     }
00180     VASSERT(nLocalVerts > 0);
00181     thee->localPartCenter[0] =
00182       thee->localPartCenter[0]/((double)(nLocalVerts));
00183     thee->localPartCenter[1] =
00184       thee->localPartCenter[1]/((double)(nLocalVerts));
00185     thee->localPartCenter[2] =
00186       thee->localPartCenter[2]/((double)(nLocalVerts));
00187     Vnm_print(0, "Vpee_ctor2: Part %d centered at (%4.3f, %4.3f, %4.3f)\n",
00188       thee->localPartID, thee->localPartCenter[0], thee->localPartCenter[1],
00189       thee->localPartCenter[2]);
00190
00191
00192     /* Now, figure out the radius of the sphere circumscribing the local
00193      * partition.  We need to keep track of vertices so we don't double count
00194      * them. */
00195     thee->localPartRadius = 0.0;
00196     for (ivert=0; ivert<Gem_numVV(thee->gm); ivert++) {
00197         vert = Gem_VV(thee->gm, ivert);
00198         simp = VV_firstSS(vert);
00199         VASSERT(simp != VNULL);
00200         while (simp != VNULL) {
00201             if (SS_chart(simp) == thee->localPartID) {
00202                 dx = thee->localPartCenter[0] - VV_coord(vert, 0);
00203                 dy = thee->localPartCenter[1] - VV_coord(vert, 1);
00204                 dz = thee->localPartCenter[2] - VV_coord(vert, 2);
00205                 radius = dx*dx + dy*dy + dz*dz;
00206                 if (radius > thee->localPartRadius) thee->localPartRadius =
00207                   radius;
00208                 break;
00209             }
00210             simp = SS_link(simp, vert);
00211         }
00212     }
00213     thee->localPartRadius = VSQRT(thee->localPartRadius);
00214     Vnm_print(0, "Vpee_ctor2: Part %d has circumscribing sphere of radius %4.3f\n",
00215       thee->localPartID, thee->localPartRadius);
00216
00217     return 1;
00218 }
00219
00220 /* ///////////////////////////////////////////////////////////////////////////
00221 // Routine:  Vpee_dtor
00222 //
00223 // Author:   Nathan Baker
00225 VPUBLIC void Vpee_dtor(Vpee **thee) {
00226
00227     if ((*thee) != VNULL) {
00228         Vpee_dtor2(*thee);
00229         Vmem_free(VNULL, 1, sizeof(Vpee), (void **)thee);
00230         (*thee) = VNULL;
```

```
00231     }
00232
00233 }
00234
00235 /* ////////////////////////////////////////////////////////////////////////
00236 // Routine:  Vpee_dtor2
00237 //
00238 // Author:   Nathan Baker
00240 VPUBLIC void Vpee_dtor2(Vpee *thee) {
00241     Vmem_dtor(&(thee->mem));
00242 }
00243
00244 /* ////////////////////////////////////////////////////////////////////////
00245 // Routine:  Vpee_markRefine
00246 //
00247 // Author:   Nathan Baker (and Michael Holst: the author of AM_markRefine, on
00248 //           which this is based)
00250 VPUBLIC int Vpee_markRefine(Vpee *thee,
00251                             AM *am,
00252                             int level,
00253                             int akey,
00254                             int rcol,
00255                             double etol,
00256                             int bkey
00257                             ) {
00258
00259     Aprx *aprx;
00260     int marked = 0,
00261         markMe,
00262         i,
00263         smid,
00264         count,
00265         currentQ;
00266     double minError = 0.0,
00267            maxError = 0.0,
00268            errEst = 0.0,
00269            mlevel,
00270            barrier;
00271     SS *sm;
00272
00273
00274     VASSERT(thee != VNULL);
00275
00276     /* Get the Aprx object from AM */
00277     aprx = am->aprx;
00278
00279     /* input check and some i/o */
00280     if ( ! ((-1 <= akey) && (akey <= 4)) ) {
00281         Vnm_print(0,"Vpee_markRefine: bad refine key; simplices marked = %d\n",
00282             marked);
00283         return marked;
00284     }
00285
00286     /* For uniform markings, we have no effect */
00287     if ((-1 <= akey) && (akey <= 0)) {
00288         marked = Gem_markRefine(thee->gm, akey, rcol);
00289         return marked;
00290     }
00291
00292     /* Informative I/O */
00293     if (akey == 2) {
00294         Vnm_print(0,"Vpee_estRefine: using Aprx_estNonlinResid().\n");
00295     } else if (akey == 3) {
00296         Vnm_print(0,"Vpee_estRefine: using Aprx_estLocalProblem().\n");
00297     } else if (akey == 4) {
00298         Vnm_print(0,"Vpee_estRefine: using Aprx_estDualProblem().\n");
00299     } else {
00300         Vnm_print(0,"Vpee_estRefine: bad key given; simplices marked = %d\n",
00301             marked);
00302         return marked;
00303     }
00304     if (thee->killFlag == 0) {
00305         Vnm_print(0, "Vpee_markRefine: No error attenuation -- simplices in all partitions will be
      marked.\n");
00306     } else if (thee->killFlag == 1) {
00307         Vnm_print(0, "Vpee_markRefine: Maximum error attenuation -- only simplices in local partition will
      be marked.\n");
00308     } else if (thee->killFlag == 2) {
00309         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",
00310           thee->killParam);
```

```
00311     } else if (thee->killFlag == 2) {
00312         Vnm_print(0, "Vpee_markRefine: Neighbor-based error attenuation -- simplices in the local and
      neighboring partitions will be marked [NOT IMPLEMENTED]!\n");
00313         VASSERT(0);
00314     } else {
00315         Vnm_print(2,"Vpee_markRefine: bogus killFlag given; simplices marked = %d\n",
00316             marked);
00317         return marked;
00318     }
00319
00320     /* set the barrier type */
00321     mlevel = (etol*etol) / Gem_numSS(thee->gm);
00322     if (bkey == 0) {
00323         barrier = (etol*etol);
00324         Vnm_print(0,"Vpee_estRefine: forcing [err per S] < [TOL] = %g\n",
00325             barrier);
00326     } else if (bkey == 1) {
00327         barrier = mlevel;
00328         Vnm_print(0,"Vpee_estRefine: forcing [err per S] < [(TOL^2/numS)^{1/2}] = %g\n",
00329             VSQRT(barrier));
00330     } else {
00331         Vnm_print(0,"Vpee_estRefine: bad bkey given; simplices marked = %d\n",
00332             marked);
00333         return marked;
00334     }
00335
00336     /* timer */
00337     Vnm_tstart(30, "error estimation");
00338
00339     /* count = num generations to produce from marked simplices (minimally) */
00340     count = 1; /* must be >= 1 */
00341
00342     /* check the refinement Q for emptyness */
00343     currentQ = 0;
00344     if (Gem_numSQ(thee->gm,currentQ) > 0) {
00345         Vnm_print(0,"Vpee_markRefine: non-empty refinement Q%d....clearing..",
00346             currentQ);
00347         Gem_resetSQ(thee->gm,currentQ);
00348         Vnm_print(0,"..done.\n");
00349     }
00350     if (Gem_numSQ(thee->gm,!currentQ) > 0) {
00351         Vnm_print(0,"Vpee_markRefine: non-empty refinement Q%d....clearing..",
00352             !currentQ);
00353         Gem_resetSQ(thee->gm,!currentQ);
00354         Vnm_print(0,"..done.\n");
00355     }
00356     VASSERT( Gem_numSQ(thee->gm,currentQ)  == 0 );
00357     VASSERT( Gem_numSQ(thee->gm,!currentQ) == 0 );
00358
00359     /* clear everyone's refinement flags */
00360     Vnm_print(0,"Vpee_markRefine: clearing all simplex refinement flags..");
00361     for (i=0; i<Gem_numSS(thee->gm); i++) {
00362         if ( (i>0) && (i % VPRTKEY) == 0 ) Vnm_print(0,"[MS:%d]",i);
00363         sm = Gem_SS(thee->gm,i);
00364         SS_setRefineKey(sm,currentQ,0);
00365         SS_setRefineKey(sm,!currentQ,0);
00366         SS_setRefinementCount(sm,0);
00367     }
00368     Vnm_print(0,"..done.\n");
00369
00370     /* NON-ERROR-BASED METHODS */
00371     /* Simplex flag clearing */
00372     if (akey == -1) return marked;
00373     /* Uniform & user-defined refinement*/
00374     if ((akey == 0) || (akey == 1)) {
00375         smid = 0;
00376         while ( smid < Gem_numSS(thee->gm)) {
00377             /* Get the simplex and find out if it's markable */
00378             sm = Gem_SS(thee->gm,smid);
00379             markMe = Vpee_ourSimp(thee, sm, rcol);
00380             if (markMe) {
00381                 if (akey == 0) {
00382                     marked++;
00383                     Gem_appendSQ(thee->gm,currentQ, sm);
00384                     SS_setRefineKey(sm,currentQ,1);
00385                     SS_setRefinementCount(sm,count);
00386                 } else if (Vpee_userDefined(thee, sm)) {
00387                     marked++;
00388                     Gem_appendSQ(thee->gm,currentQ, sm);
00389                     SS_setRefineKey(sm,currentQ,1);
00390                     SS_setRefinementCount(sm,count);
```

```
00391                 }
00392             }
00393             smid++;
00394         }
00395     }
00396
00397     /* ERROR-BASED METHODS */
00398     /* gerror = global error accumulation */
00399     aprx->gerror = 0.;
00400
00401     /* traverse the simplices and process the error estimates */
00402     Vnm_print(0,"Vpee_markRefine: estimating error..");
00403     smid = 0;
00404     while ( smid < Gem_numSS(thee->gm)) {
00405
00406         /* Get the simplex and find out if it's markable */
00407         sm = Gem_SS(thee->gm,smid);
00408         markMe = Vpee_ourSimp(thee, sm, rcol);
00409
00410         if ( (smid>0) && (smid % VPRTKEY) == 0 ) Vnm_print(0,"[MS:%d]",smid);
00411
00412         /* Produce an error estimate for this element if it is in the set */
00413         if (markMe) {
00414             if (akey == 2) {
00415                 errEst = Aprx_estNonlinResid(aprx, sm, am->u,am->ud,am->f);
00416             } else if (akey == 3) {
00417                 errEst = Aprx_estLocalProblem(aprx, sm, am->u,am->ud,am->f);
00418             } else if (akey == 4) {
00419                 errEst = Aprx_estDualProblem(aprx, sm, am->u,am->ud,am->f);
00420             }
00421             VASSERT( errEst >= 0. );
00422
00423             /* if error estimate above tol, mark element for refinement */
00424             if ( errEst > barrier ) {
00425                 marked++;
00426                 Gem_appendSQ(thee->gm,currentQ, sm); /*add to refinement Q*/
00427                 SS_setRefineKey(sm,currentQ,1);      /* note now on refine Q */
00428                 SS_setRefinementCount(sm,count);     /* refine X many times? */
00429             }
00430
00431             /* keep track of min/max errors over the mesh */
00432             minError = VMIN2( VSQRT(VABS(errEst)), minError );
00433             maxError = VMAX2( VSQRT(VABS(errEst)), maxError );
00434
00435             /* store the estimate */
00436             Bvec_set( aprx->wev, smid, errEst );
00437
00438             /* accumlate into global error (errEst is SQUAREd already) */
00439             aprx->gerror += errEst;
00440
00441         /* otherwise store a zero for the estimate */
00442         } else {
00443             Bvec_set( aprx->wev, smid, 0. );
00444         }
00445
00446         smid++;
00447     }
00448
00449     /* do some i/o */
00450     Vnm_print(0,"..done.  [marked=<%d/%d>]\n",marked,Gem_numSS(thee->gm));
00451     Vnm_print(0,"Vpee_estRefine: TOL=<%g>  Global_Error=<%g>\n",
00452         etol, aprx->gerror);
00453     Vnm_print(0,"Vpee_estRefine: (TOL^2/numS)^{1/2}=<%g>  Max_Ele_Error=<%g>\n",
00454         VSQRT(mlevel),maxError);
00455     Vnm_tstop(30, "error estimation");
00456
00457     /* check for making the error tolerance */
00458     if ((bkey == 1) && (aprx->gerror <= etol)) {
00459         Vnm_print(0,
00460             "Vpee_estRefine: ********************************************\n");
00461         Vnm_print(0,
00462             "Vpee_estRefine: Global Error criterion met; setting marked=0.\n");
00463         Vnm_print(0,
00464             "Vpee_estRefine: ********************************************\n");
00465         marked = 0;
00466     }
00467
00468
00469     /* return */
00470     return marked;
00471
```

```
00472 }
00473
00474 /* ////////////////////////////////////////////////////////////////////////
00475 // Routine:  Vpee_numSS
00476 //
00477 // Author:   Nathan Baker
00479 VPUBLIC int Vpee_numSS(Vpee *thee) {
00480     int num = 0;
00481     int isimp;
00482
00483     for (isimp=0; isimp<Gem_numSS(thee->gm); isimp++) {
00484         if (SS_chart(Gem_SS(thee->gm, isimp)) == thee->localPartID) num++;
00485     }
00486
00487     return num;
00488 }
00489
00490 /* ////////////////////////////////////////////////////////////////////////
00491 // Routine:  Vpee_userDefined
00492 //
00493 // Purpose:  Reduce code bloat by wrapping up the common steps for getting the
00494 //           user-defined error estimate
00495 //
00496 // Author:   Nathan Baker
00498 VPRIVATE int Vpee_userDefined(Vpee *thee,
00499                               SS *sm
00500                               ) {
00501
00502     int ivert,
00503         icoord,
00504         chart[4],
00505         fType[4],
00506         vType[4];
00507     double vx[4][3];
00508
00509     for (ivert=0; ivert<Gem_dimVV(thee->gm); ivert++) {
00510         fType[ivert] = SS_faceType(sm,ivert);
00511         vType[ivert] = VV_type(SS_vertex(sm,ivert) );
00512         chart[ivert] = VV_chart(SS_vertex(sm,ivert) );
00513         for (icoord=0; icoord<Gem_dimII(thee->gm); icoord++) {
00514             vx[ivert][icoord] = VV_coord(SS_vertex(sm,ivert), icoord );
00515         }
00516     }
00517     return thee->gm->pde->markSimplex(Gem_dim(thee->gm), Gem_dimII(thee->gm),
00518              SS_type(sm), fType, vType, chart, vx, sm);
00519 }
00520
00521 /* ////////////////////////////////////////////////////////////////////////
00522 // Routine:  Vpee_ourSimp
00523 //
00524 // Purpose:  Reduce code bloat by wrapping up the common steps for determining
00525 //           whether the given simplex can be marked (i.e., belongs to our
00526 //           partition or overlap region)
00527 //
00528 // Returns:  1 if could be marked, 0 otherwise
00529 //
00530 // Author:   Nathan Baker
00532 VPRIVATE int Vpee_ourSimp(Vpee *thee,
00533                           SS *sm,
00534                           int rcol
00535                           ) {
00536
00537     int ivert;
00538     double dist,
00539            dx,
00540            dy,
00541            dz;
00542
00543     if (thee->killFlag == 0) return 1;
00544     else if (thee->killFlag == 1) {
00545         if ((SS_chart(sm) == rcol) || (rcol < 0)) return 1;
00546     } else if (thee->killFlag == 2) {
00547         if (rcol < 0) return 1;
00548         else {
00549             /* We can only do distance-based searches on the local partition */
00550             VASSERT(rcol == thee->localPartID);
00551             /* Find the closest distance between this simplex and the
00552              * center of the local partition and check it against
00553              * (thee->localPartRadius*thee->killParam) */
00554             dist = 0;
00555             for (ivert=0; ivert<SS_dimVV(sm); ivert++) {
```

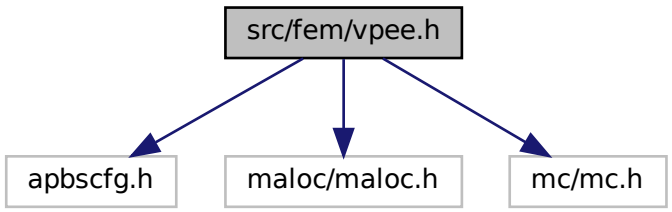
```
00556                 dx = VV_coord(SS_vertex(sm, ivert), 0) -
00557                   thee->localPartCenter[0];
00558                 dy = VV_coord(SS_vertex(sm, ivert), 1) -
00559                   thee->localPartCenter[1];
00560                 dz = VV_coord(SS_vertex(sm, ivert), 2) -
00561                   thee->localPartCenter[2];
00562                 dist = VSQRT((dx*dx + dy*dy + dz*dz));
00563             }
00564             if (dist < thee->localPartRadius*thee->killParam) return 1;
00565         }
00566     } else if (thee->killFlag == 3) VASSERT(0);
00567     else VASSERT(0);
00568
00569     return 0;
00570
00571 }
```

9.13 src/fem/vpee.h File Reference

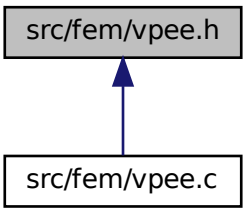
Contains declarations for class Vpee.

```
#include "apbscfg.h"
#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.

9.13.1 Detailed Description

Contains declarations for class Vpee.

Version

Id

Author

Nathan A. Baker

Attention

```
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * Redistributions of source code must retain the above copyright notice, this
 * list of conditions and the following disclaimer.
 *
 * Redistributions in binary form must reproduce the above copyright 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.

9.14 vpee.h

```
00001
00076 #ifndef _VPEE_H
00077 #define _VPEE_H
00078
00079 #include "apbscfg.h"
00080
00081 #include "maloc/maloc.h"
00082 #include "mc/mc.h"
00083
00089 struct sVpee {
00090
00091   Gem *gm;
00092   int localPartID;
00095   double localPartCenter[3];
00097   double localPartRadius;
00099   int killFlag;
00102   double killParam;
00104   Vmem *mem;
00106 };
00107
00112 typedef struct sVpee Vpee;
00113
00114 /* ///////////////////////////////////////////////////////////////////////// */
00115 // Class Vpee Inlineable methods
00117
00118 #if !defined(VINLINE_VPEE)
00119 #else /* if defined(VINLINE_VPEE) */
00120 #endif /* if !defined(VINLINE_VPEE) */
00121
00122 /* ///////////////////////////////////////////////////////////////////////// */
00123 // Class Vpee: Non-Inlineable methods (vpee.c)
00125
00132 VEXTERNC Vpee* Vpee_ctor(
00133         Gem *gm,  /**< FEtk geometry manager object */
00134         int localPartID,
00135         int killFlag,
00146         double killParam
00147         );
00148
00155 VEXTERNC int Vpee_ctor2(
```

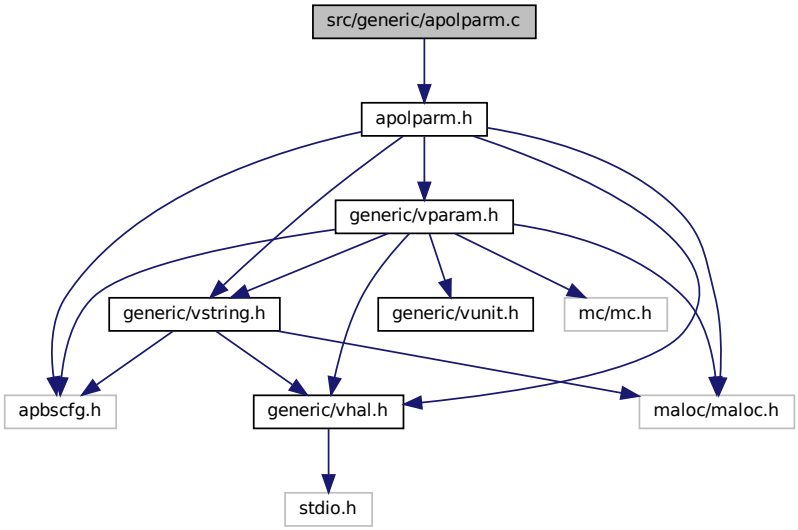
```
00156         Vpee *thee,
00157         Gem *gm,
00158         int localPartID,
00159         int killFlag,
00170         double killParam
00171         );
00172
00177 VEXTERNC void Vpee_dtor(
00178         Vpee **thee
00179         );
00180
00185 VEXTERNC void Vpee_dtor2(
00186         Vpee *thee
00187         );
00188
00204 VEXTERNC int Vpee_markRefine(
00205         Vpee *thee,
00206         AM *am,
00207         int level,
00208         int akey,
00216         int rcol,
00219         double etol,
00220         int bkey
00224         );
00225
00231 VEXTERNC int Vpee_numSS(
00232         Vpee *thee
00233         );
00234
00235 #endif    /* ifndef _VPEE_H_ */
```

9.15 src/generic/apolparm.c File Reference

Class APOLparm methods.

`#include "apolparm.h"`

Include dependency graph for apolparm.c:



Functions

- VPUBLIC APOLparm * APOLparm_ctor ()

Construct APOLparm.

- VPUBLIC Vrc_Codes APOLparm_ctor2 (APOLparm *thee)

  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.

9.15.1 Detailed Description

Class APOLparm methods.

Author

David Gohara

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
```

Definition in file apolparm.c.

9.16 apolparm.c

```
00001
00057 #include "apolparm.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_MGPARM)
00062
00063 #endif /* if !defined(VINLINE_MGPARM) */
00064
00065 VPUBLIC APOLparm* APOLparm_ctor() {
00066
00067     /* Set up the structure */
00068     APOLparm *thee = VNULL;
00069     thee = (APOLparm*)Vmem_malloc(VNULL, 1, sizeof(APOLparm));
00070     VASSERT( thee != VNULL);
00071     VASSERT( APOLparm_ctor2(thee) == VRC_SUCCESS );
00072
00073     return thee;
00074 }
00075
00076 VPUBLIC Vrc_Codes APOLparm_ctor2(APOLparm *thee) {
00077
00078     int i;
00079
00080     if (thee == VNULL) return VRC_FAILURE;
00081
00082     thee->parsed = 0;
00083
00084     thee->setgrid = 0;
00085     thee->setmolid = 0;
00086     thee->setbconc = 0;
00087     thee->setsdens = 0;
00088     thee->setdpos = 0;
00089     thee->setpress = 0;
00090     thee->setsrfm = 0;
00091     thee->setsrad = 0;
```

```
00092     thee->setswin = 0;
00093
00094     thee->settemp = 0;
00095     thee->setgamma = 0;
00096
00097     thee->setwat = 0;
00098
00099     thee->sav = 0.0;
00100     thee->sasa = 0.0;
00101     thee->wcaEnergy = 0.0;
00102
00103     for(i=0;i<3;i++) thee->totForce[i] = 0.0;
00104
00105     return VRC_SUCCESS;
00106 }
00107
00108 VPUBLIC void APOLparm_copy(
00109                           APOLparm *thee,
00110                           APOLparm *source
00111                           ) {
00112
00113     int i;
00114
00115     thee->parsed = source->parsed;
00116
00117     for (i=0; i<3; i++) thee->grid[i] = source->grid[i];
00118     thee->setgrid = source->setgrid;
00119
00120     thee->molid = source->molid;
00121     thee->setmolid = source->setmolid;
00122
00123     thee->bconc = source->bconc ;
00124     thee->setbconc= source->setbconc ;
00125
00126     thee->sdens = source->sdens ;
00127     thee->setsdens= source->setsdens ;
00128
00129     thee->dpos = source->dpos ;
00130     thee->setdpos= source->setdpos ;
00131
00132     thee->press = source->press ;
00133     thee->setpress = source->setpress ;
00134
00135     thee->srfm = source->srfm ;
00136     thee->setsrfm = source->setsrfm ;
00137
00138     thee->srad = source->srad ;
00139     thee->setsrad = source->setsrad ;
00140
00141     thee->swin = source->swin ;
00142     thee->setswin = source->setswin ;
00143
00144     thee->temp = source->temp ;
00145     thee->settemp = source->settemp ;
00146
00147     thee->gamma = source->gamma ;
00148     thee->setgamma = source->setgamma ;
00149
00150     thee->calcenergy = source->calcenergy ;
00151     thee->setcalcenergy = source->setcalcenergy ;
00152
00153     thee->calcforce = source->calcforce ;
00154     thee->setcalcforce = source->setcalcforce ;
00155
00156     thee->setwat = source->setwat ;
00157
00158     thee->sav = source->sav;
00159     thee->sasa = source->sasa;
00160     thee->wcaEnergy = source->wcaEnergy;
00161
00162     for(i=0;i<3;i++) thee->totForce[i] = source->totForce[i];
00163
00164     return;
00165 }
00166
00167 VPUBLIC void APOLparm_dtor(APOLparm **thee) {
00168     if ((*thee) != VNULL) {
00169         APOLparm_dtor2(*thee);
00170         Vmem_free(VNULL, 1, sizeof(APOLparm), (void **)thee);
00171         (*thee) = VNULL;
00172     }
```

```
00173
00174     return;
00175 }
00176
00177 VPUBLIC void APOLparm_dtor2(APOLparm *thee) { ; }
00178
00179 VPUBLIC Vrc_Codes APOLparm_check(APOLparm *thee) {
00180
00181
00182     Vrc_Codes rc;
00183     rc = VRC_SUCCESS;
00184
00185     if (!thee->parsed) {
00186         Vnm_print(2, "APOLparm_check:  not filled!\n");
00187         return VRC_FAILURE;
00188     }
00189     if (!thee->setgrid) {
00190         Vnm_print(2, "APOLparm_check:  grid not set!\n");
00191         rc = VRC_FAILURE;
00192     }
00193     if (!thee->setmolid) {
00194         Vnm_print(2, "APOLparm_check:  molid not set!\n");
00195         rc = VRC_FAILURE;
00196     }
00197     if (!thee->setbconc) {
00198         Vnm_print(2, "APOLparm_check:  bconc not set!\n");
00199         rc = VRC_FAILURE;
00200     }
00201     if (!thee->setsdens) {
00202         Vnm_print(2, "APOLparm_check:  sdens not set!\n");
00203         rc = VRC_FAILURE;
00204     }
00205     if (!thee->setdpos) {
00206         Vnm_print(2, "APOLparm_check:  dpos not set!\n");
00207         rc = VRC_FAILURE;
00208     }
00209     if (!thee->setpress) {
00210         Vnm_print(2, "APOLparm_check:  press not set!\n");
00211         rc = VRC_FAILURE;
00212     }
00213     if (!thee->setsrfm) {
00214         Vnm_print(2, "APOLparm_check:  srfm not set!\n");
00215         rc = VRC_FAILURE;
00216     }
00217     if (!thee->setsrad) {
00218         Vnm_print(2, "APOLparm_check:  srad not set!\n");
00219         rc = VRC_FAILURE;
00220     }
00221     if (!thee->setswin) {
00222         Vnm_print(2, "APOLparm_check:  swin not set!\n");
00223         rc = VRC_FAILURE;
00224     }
00225     if (!thee->settemp) {
00226         Vnm_print(2, "APOLparm_check:  temp not set!\n");
00227         rc = VRC_FAILURE;
00228     }
00229     if (!thee->setgamma) {
00230         Vnm_print(2, "APOLparm_check:  gamma not set!\n");
00231         rc = VRC_FAILURE;
00232     }
00233     return rc;
00234
00235 }
00236
00237 VPRIVATE Vrc_Codes APOLparm_parseGRID(APOLparm *thee, Vio *sock) {
00238
00239     char tok[VMAX_BUFSIZE];
00240     double tf;
00241
00242     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00243     if (sscanf(tok, "%lf", &tf) == 0) {
00244         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GRID \
00245 keyword!\n", tok);
00246         return VRC_WARNING;
00247     } else thee->grid[0] = tf;
00248     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00249     if (sscanf(tok, "%lf", &tf) == 0) {
00250         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GRID \
00251 keyword!\n", tok);
00252         return VRC_WARNING;
00253     } else thee->grid[1] = tf;
```

```
00254     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00255     if (sscanf(tok, "%lf", &tf) == 0) {
00256         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GRID \
00257 keyword!\n", tok);
00258         return VRC_WARNING;
00259     } else thee->grid[2] = tf;
00260     thee->setgrid = 1;
00261     return VRC_SUCCESS;
00262 
00263 VERROR1:
00264         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00265     return VRC_WARNING;
00266 }
00267 
00268 VPRIVATE Vrc_Codes APOLparm_parseMOL(APOLparm *thee, Vio *sock) {
00269     int ti;
00270     char tok[VMAX_BUFSIZE];
00271 
00272     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00273     if (sscanf(tok, "%d", &ti) == 0) {
00274         Vnm_print(2, "NOsh:  Read non-int (%s) while parsing MOL \
00275 keyword!\n", tok);
00276         return VRC_WARNING;
00277     }
00278     thee->molid = ti;
00279     thee->setmolid = 1;
00280     return VRC_SUCCESS;
00281 
00282 VERROR1:
00283         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00284     return VRC_WARNING;
00285 }
00286 
00287 VPRIVATE Vrc_Codes APOLparm_parseSRFM(APOLparm *thee, Vio *sock) {
00288     char tok[VMAX_BUFSIZE];
00289 
00290     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00291 
00292     if (Vstring_strcasecmp(tok, "sacc") == 0) {
00293         thee->srfm = VSM_MOL;
00294         thee->setsrfm = 1;
00295         return VRC_SUCCESS;
00296     } else {
00297         Vnm_print(2, "parseAPOL: Unrecongnized keyword (%s) when parsing srfm!\n", tok);
00298         Vnm_print(2, "parseAPOL: Accepted values for srfm = sacc\n");
00299         return VRC_WARNING;
00300     }
00301 
00302     return VRC_FAILURE;
00303 
00304 VERROR1:
00305         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00306     return VRC_WARNING;
00307 }
00308 
00309 VPRIVATE Vrc_Codes APOLparm_parseSRAD(APOLparm *thee, Vio *sock) {
00310     char tok[VMAX_BUFSIZE];
00311     double tf;
00312 
00313     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00314     if (sscanf(tok, "%lf", &tf) == 0) {
00315         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SRAD \
00316 keyword!\n", tok);
00317         return VRC_WARNING;
00318     }
00319     thee->srad = tf;
00320     thee->setsrad = 1;
00321     return VRC_SUCCESS;
00322 
00323 VERROR1:
00324         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00325     return VRC_WARNING;
00326 }
00327 
00328 VPRIVATE Vrc_Codes APOLparm_parseSWIN(APOLparm *thee, Vio *sock) {
00329     char tok[VMAX_BUFSIZE];
00330     double tf;
00331 
00332     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00333     if (sscanf(tok, "%lf", &tf) == 0) {
00334         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SWIN \
```

```
00335 keyword!\n", tok);
00336         return VRC_WARNING;
00337     }
00338     thee->swin = tf;
00339     thee->setswin = 1;
00340     return VRC_SUCCESS;
00341
00342 VERROR1:
00343         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00344     return VRC_WARNING;
00345 }
00346
00347 VPRIVATE Vrc_Codes APOLparm_parseTEMP(APOLparm *thee, Vio *sock) {
00348     char tok[VMAX_BUFSIZE];
00349     double tf;
00350
00351     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00352     if (sscanf(tok, "%lf", &tf) == 0) {
00353         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing TEMP \
00354 keyword!\n", tok);
00355         return VRC_WARNING;
00356     }
00357     thee->temp = tf;
00358     thee->settemp = 1;
00359     return VRC_SUCCESS;
00360
00361 VERROR1:
00362         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00363     return VRC_WARNING;
00364 }
00365
00366 VPRIVATE Vrc_Codes APOLparm_parseGAMMA(APOLparm *thee, Vio *sock) {
00367     char tok[VMAX_BUFSIZE];
00368     double tf;
00369
00370     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00371     if (sscanf(tok, "%lf", &tf) == 0) {
00372         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GAMMA \
00373 keyword!\n", tok);
00374         return VRC_WARNING;
00375     }
00376     thee->gamma = tf;
00377     thee->setgamma = 1;
00378     return VRC_SUCCESS;
00379
00380 VERROR1:
00381         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00382     return VRC_WARNING;
00383 }
00384
00385 VPRIVATE Vrc_Codes APOLparm_parseCALCENERGY(APOLparm *thee, Vio *sock) {
00386     char tok[VMAX_BUFSIZE];
00387     int ti;
00388
00389     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00390     /* Parse number */
00391     if (sscanf(tok, "%d", &ti) == 1) {
00392         thee->calcenergy = (APOLparm_calcEnergy)ti;
00393         thee->setcalcenergy = 1;
00394
00395         Vnm_print(2, "parseAPOL:  Warning -- parsed deprecated \"calcenergy \
00396 %d\" statement.\n", ti);
00397         Vnm_print(2, "parseAPOL:  Please use \"calcenergy ");
00398         switch (thee->calcenergy) {
00399             case ACE_NO:
00400                 Vnm_print(2, "no");
00401                 break;
00402             case ACE_TOTAL:
00403                 Vnm_print(2, "total");
00404                 break;
00405             case ACE_COMPS:
00406                 Vnm_print(2, "comps");
00407                 break;
00408             default:
00409                 Vnm_print(2, "UNKNOWN");
00410                 break;
00411         }
00412         Vnm_print(2, "\" instead.\n");
00413         return VRC_SUCCESS;
00414     } else if (Vstring_strcasecmp(tok, "no") == 0) {
00415         thee->calcenergy = ACE_NO;
```

```
00416         thee->setcalcenergy = 1;
00417         return VRC_SUCCESS;
00418     } else if (Vstring_strcasecmp(tok, "total") == 0) {
00419         thee->calcenergy = ACE_TOTAL;
00420         thee->setcalcenergy = 1;
00421         return VRC_SUCCESS;
00422     } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00423         thee->calcenergy = ACE_COMPS;
00424         thee->setcalcenergy = 1;
00425         return VRC_SUCCESS;
00426     } else {
00427         Vnm_print(2, "NOsh:  Unrecognized parameter (%s) while parsing \
00428 calcenergy!\n", tok);
00429         return VRC_WARNING;
00430     }
00431     return VRC_FAILURE;
00432
00433 VERROR1:
00434         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00435     return VRC_WARNING;
00436 }
00437
00438 VPRIVATE Vrc_Codes APOLparm_parseCALCFORCE(APOLparm *thee, Vio *sock) {
00439     char tok[VMAX_BUFSIZE];
00440     int ti;
00441
00442     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00443     /* Parse number */
00444     if (sscanf(tok, "%d", &ti) == 1) {
00445         thee->calcforce = (APOLparm_calcForce)ti;
00446         thee->setcalcforce = 1;
00447
00448         Vnm_print(2, "parseAPOL:  Warning -- parsed deprecated \"calcforce \
00449 %d\" statement.\n", ti);
00450         Vnm_print(2, "parseAPOL:  Please use \"calcforce ");
00451         switch (thee->calcenergy) {
00452             case ACF_NO:
00453                 Vnm_print(2, "no");
00454                 break;
00455             case ACF_TOTAL:
00456                 Vnm_print(2, "total");
00457                 break;
00458             case ACF_COMPS:
00459                 Vnm_print(2, "comps");
00460                 break;
00461             default:
00462                 Vnm_print(2, "UNKNOWN");
00463                 break;
00464         }
00465         Vnm_print(2, "\" instead.\n");
00466         return VRC_SUCCESS;
00467     } else if (Vstring_strcasecmp(tok, "no") == 0) {
00468         thee->calcforce = ACF_NO;
00469         thee->setcalcforce = 1;
00470         return VRC_SUCCESS;
00471     } else if (Vstring_strcasecmp(tok, "total") == 0) {
00472         thee->calcforce = ACF_TOTAL;
00473         thee->setcalcforce = 1;
00474         return VRC_SUCCESS;
00475     } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00476         thee->calcforce = ACF_COMPS;
00477         thee->setcalcforce = 1;
00478         return VRC_SUCCESS;
00479     } else {
00480         Vnm_print(2, "NOsh:  Unrecognized parameter (%s) while parsing \
00481 calcforce!\n", tok);
00482         return VRC_WARNING;
00483     }
00484     return VRC_FAILURE;
00485
00486 VERROR1:
00487         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00488     return VRC_WARNING;
00489 }
00490
00491 VPRIVATE Vrc_Codes APOLparm_parseBCONC(APOLparm *thee, Vio *sock) {
00492     char tok[VMAX_BUFSIZE];
00493     double tf;
00494
00495     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00496     if (sscanf(tok, "%lf", &tf) == 0) {
```

```
00497         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing BCONC \
00498 keyword!\n", tok);
00499         return VRC_WARNING;
00500     }
00501     thee->bconc = tf;
00502     thee->setbconc = 1;
00503     return VRC_SUCCESS;
00504 
00505 VERROR1:
00506         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00507     return VRC_WARNING;
00508 }
00509 
00510 VPRIVATE Vrc_Codes APOLparm_parseSDENS(APOLparm *thee, Vio *sock) {
00511     char tok[VMAX_BUFSIZE];
00512     double tf;
00513 
00514     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00515     if (sscanf(tok, "%lf", &tf) == 0) {
00516         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SDENS \
00517 keyword!\n", tok);
00518         return VRC_WARNING;
00519     }
00520     thee->sdens = tf;
00521     thee->setsdens = 1;
00522     return VRC_SUCCESS;
00523 
00524 VERROR1:
00525         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00526     return VRC_WARNING;
00527 }
00528 
00529 VPRIVATE Vrc_Codes APOLparm_parseDPOS(APOLparm *thee, Vio *sock) {
00530     char tok[VMAX_BUFSIZE];
00531     double tf;
00532 
00533     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00534     if (sscanf(tok, "%lf", &tf) == 0) {
00535         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SDENS \
00536 keyword!\n", tok);
00537         return VRC_WARNING;
00538     }
00539     thee->dpos = tf;
00540     thee->setdpos = 1;
00541 
00542     if(thee->dpos < 0.001){
00543         Vnm_print(1,"\nWARNING WARNING WARNING WARNING WARNING\n");
00544         Vnm_print(1,"NOsh: dpos is set to a very small value.\n");
00545         Vnm_print(1,"NOsh: If you are not using a PQR file, you can \
00546 safely ignore this message.\n");
00547         Vnm_print(1,"NOsh: Otherwise please choose a value greater than \
00548 or equal to 0.001.\n\n");
00549     }
00550 
00551     return VRC_SUCCESS;
00552 
00553 VERROR1:
00554         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00555     return VRC_WARNING;
00556 }
00557 
00558 VPRIVATE Vrc_Codes APOLparm_parsePRESS(APOLparm *thee, Vio *sock) {
00559     char tok[VMAX_BUFSIZE];
00560     double tf;
00561 
00562     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00563     if (sscanf(tok, "%lf", &tf) == 0) {
00564         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing PRESS \
00565 keyword!\n", tok);
00566         return VRC_WARNING;
00567     }
00568     thee->press = tf;
00569     thee->setpress = 1;
00570     return VRC_SUCCESS;
00571 
00572 VERROR1:
00573         Vnm_print(2, "parseAPOL:  ran out of tokens!\n");
00574     return VRC_WARNING;
00575 }
00576 
00577 VPUBLIC Vrc_Codes APOLparm_parseToken(APOLparm *thee, char tok[VMAX_BUFSIZE],
```

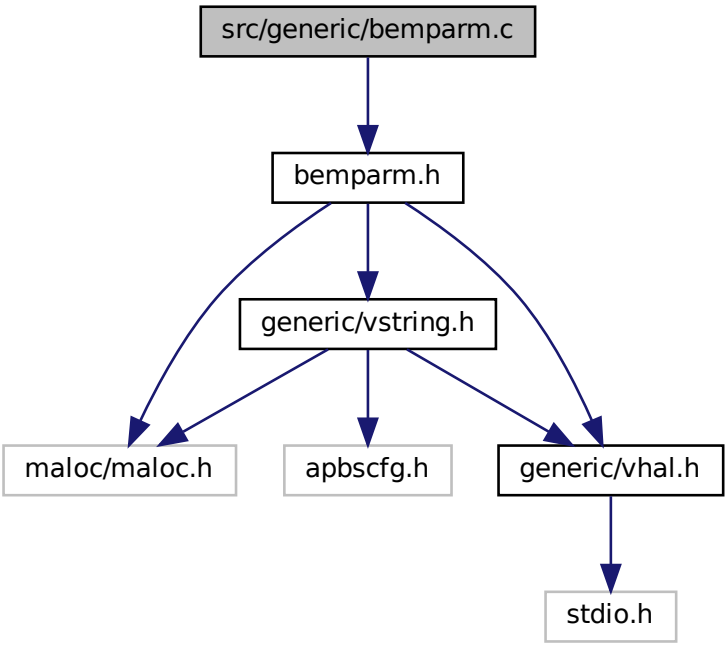
```
00578   Vio *sock) {
00579
00580     if (thee == VNULL) {
00581         Vnm_print(2, "parseAPOL:  got NULL thee!\n");
00582         return VRC_WARNING;
00583     }
00584
00585     if (sock == VNULL) {
00586         Vnm_print(2, "parseAPOL:  got NULL socket!\n");
00587         return VRC_WARNING;
00588     }
00589
00590     if (Vstring_strcasecmp(tok, "mol") == 0) {
00591         return APOLparm_parseMOL(thee, sock);
00592     } else if (Vstring_strcasecmp(tok, "grid") == 0) {
00593         return APOLparm_parseGRID(thee, sock);
00594     } else if (Vstring_strcasecmp(tok, "dime") == 0) {
00595         Vnm_print(2, "APOLparm_parseToken:  The DIME and GLEN keywords for APOLAR have been replaced with
    GRID.\n");
00596         Vnm_print(2, "APOLparm_parseToken:  Please see the APBS User Guide for more information.\n");
00597         return VRC_WARNING;
00598     } else if (Vstring_strcasecmp(tok, "glen") == 0) {
00599         Vnm_print(2, "APOLparm_parseToken:  The DIME and GLEN keywords for APOLAR have been replaced with
    GRID.\n");
00600         Vnm_print(2, "APOLparm_parseToken:  Please see the APBS User Guide for more information.\n");
00601         return VRC_WARNING;
00602     } else if (Vstring_strcasecmp(tok, "bconc") == 0) {
00603         return APOLparm_parseBCONC(thee, sock);
00604     } else if (Vstring_strcasecmp(tok, "sdens") == 0) {
00605         return APOLparm_parseSDENS(thee, sock);
00606     } else if (Vstring_strcasecmp(tok, "dpos") == 0) {
00607         return APOLparm_parseDPOS(thee, sock);
00608     }  else if (Vstring_strcasecmp(tok, "srfm") == 0) {
00609         return APOLparm_parseSRFM(thee, sock);
00610     } else if (Vstring_strcasecmp(tok, "srad") == 0) {
00611         return APOLparm_parseSRAD(thee, sock);
00612     } else if (Vstring_strcasecmp(tok, "swin") == 0) {
00613         return APOLparm_parseSWIN(thee, sock);
00614     } else if (Vstring_strcasecmp(tok, "temp") == 0) {
00615         return APOLparm_parseTEMP(thee, sock);
00616     } else if (Vstring_strcasecmp(tok, "gamma") == 0) {
00617         return APOLparm_parseGAMMA(thee, sock);
00618     } else if (Vstring_strcasecmp(tok, "press") == 0) {
00619         return APOLparm_parsePRESS(thee, sock);
00620     } else if (Vstring_strcasecmp(tok, "calcenergy") == 0) {
00621         return APOLparm_parseCALCENERGY(thee, sock);
00622     } else if (Vstring_strcasecmp(tok, "calcforce") == 0) {
00623         return APOLparm_parseCALCFORCE(thee, sock);
00624     }
00625
00626     return VRC_FAILURE;
00627
00628 }
```

9.17 src/generic/bemparm.c File Reference

Class BEMparm methods.

```
#include "bemparm.h"
```

Include dependency graph for bemparm.c:



Functions

- VPUBLIC BEMparm ∗ BEMparm_ctor (BEMparm_CalcType type)

  Construct BEMparm object.
- VPUBLIC Vrc_Codes BEMparm_ctor2 (BEMparm ∗thee, BEMparm_CalcType type)

  FORTRAN stub to construct BEMparm object.
- VPUBLIC void BEMparm_dtor (BEMparm ∗∗thee)

  Object destructor.
- VPUBLIC void BEMparm_dtor2 (BEMparm ∗thee)

  FORTRAN stub for object destructor.
- VPUBLIC Vrc_Codes BEMparm_check (BEMparm ∗thee)

  Consistency check for parameter values stored in object.
- VPUBLIC void BEMparm_copy (BEMparm ∗thee, BEMparm ∗parm)

  Copy object info into thee.
- VPRIVATE Vrc_Codes **BEMparm_parseTREE_ORDER** (BEMparm ∗thee, Vio ∗sock)
- VPRIVATE Vrc_Codes **BEMparm_parseTREE_N0** (BEMparm ∗thee, Vio ∗sock)
- VPRIVATE Vrc_Codes **BEMparm_parseMAC** (BEMparm ∗thee, Vio ∗sock)
- VPRIVATE Vrc_Codes **BEMparm_parseMESH** (BEMparm ∗thee, Vio ∗sock)
- VPRIVATE Vrc_Codes **BEMparm_parseOUTDATA** (BEMparm ∗thee, Vio ∗sock)
- VPUBLIC Vrc_Codes BEMparm_parseToken (BEMparm ∗thee, char tok[VMAX_BUFSIZE], Vio ∗sock)

  Parse an MG keyword from an input file.

9.17.1 Detailed Description

Class BEMparm methods.

Author

Nathan A. Baker, Weihua Geng, and Andrew J. Stevens

Version

Id

Attention

Definition in file bemparm.c.

9.17.2 Function Documentation

9.17.2.1 BEMparm_copy()

```
VPUBLIC void BEMparm_copy (
             BEMparm * thee,
             BEMparm * parm )
```

Copy object info into thee.

Author

Nathan Baker

Parameters

thee	destination object
parm	source object

Definition at line 174 of file bemparm.c.

9.18 bemparm.c

```
00001
00057 #include "bemparm.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_MGPARM)
00062
00063 #endif /* if !defined(VINLINE_MGPARM) */
00064
00065
00066 VPUBLIC BEMparm* BEMparm_ctor(BEMparm_CalcType type) {
00067
00068     /* Set up the structure */
00069     BEMparm *thee = VNULL;
00070     thee = (BEMparm*)Vmem_malloc(VNULL, 1, sizeof(BEMparm));
00071     VASSERT( thee != VNULL);
00072     VASSERT( BEMparm_ctor2(thee, type) == VRC_SUCCESS );
00073
00074     return thee;
00075 }
00076
00077 VPUBLIC Vrc_Codes BEMparm_ctor2(BEMparm *thee, BEMparm_CalcType type) {
00078
00079     int i;
00080
00081     if (thee == VNULL) return VRC_FAILURE;
00082
00083     thee->parsed = 0;
00084     thee->type = type;
00085
00086     /* *** GENERIC PARAMETERS *** */
00087
00088     /* *** TYPE 0 PARAMETERS *** */
00089     thee->tree_order = 1;
00090     thee->settree_order = 0;
00091     thee->tree_n0 = 500;
00092     thee->settree_n0 = 0;
00093     thee->mac = 0.8;
00094     thee->setmac = 0;
00095
00096     thee->mesh = 0;
00097     thee->setmesh = 0;
00098
00099     thee->outdata = 0;
00100     thee->setoutdata = 0;
00101
00102     /* *** TYPE 1 & 2 PARAMETERS *** */
00103
00104     /* *** TYPE 2 PARAMETERS *** */
00105     thee->nonlintype = 0;
00106     thee->setnonlintype = 0;
```

```
00107
00108     /* *** Default parameters for TINKER *** */
00109     thee->chgs = VCM_CHARGE;
00110
00111     return VRC_SUCCESS;
00112 }
00113
00114 VPUBLIC void BEMparm_dtor(BEMparm **thee) {
00115     if ((*thee) != VNULL) {
00116         BEMparm_dtor2(*thee);
00117         Vmem_free(VNULL, 1, sizeof(BEMparm), (void **)thee);
00118         (*thee) = VNULL;
00119     }
00120 }
00121
00122 VPUBLIC void BEMparm_dtor2(BEMparm *thee) { ; }
00123
00124 VPUBLIC Vrc_Codes BEMparm_check(BEMparm *thee) {
00125
00126     Vrc_Codes rc;
00127     int i, tdime[3], ti, tnlev[3], nlev;
00128
00129     rc = VRC_SUCCESS;
00130
00131     Vnm_print(0, "BEMparm_check:  checking BEMparm object of type %d.\n",
00132               thee->type);
00133
00134     /* Check to see if we were even filled... */
00135     if (!thee->parsed) {
00136         Vnm_print(2, "BEMparm_check:  not filled!\n");
00137         return VRC_FAILURE;
00138     }
00139
00140
00141     /* Check type settings */
00142     if ((thee->type != BCT_MANUAL) && (thee->type != BCT_NONE)) {
00143          Vnm_print(2,"BEMparm_check: type not set");
00144          rc = VRC_FAILURE;
00145     }
00146
00147     /* Check treecode setting*/
00148     if (thee->tree_order < 1) {
00149         Vnm_print(2, "BEMparm_check: treecode order is less than 1");
00150         rc = VRC_FAILURE;
00151     }
00152     if (thee->tree_n0 < 1) {
00153         Vnm_print(2, "BEMparm_check: treecode leaf size is less than 1");
00154         rc = VRC_FAILURE;
00155     }
00156     if (thee->mac > 1 || thee->mac <= 0) {
00157         Vnm_print(2, "BEMparm_check: MAC criterion fails");
00158         rc = VRC_FAILURE;
00159     }
00160
00161     if (thee->mesh > 2 || thee->mesh < 0) {
00162         Vnm_print(2, "BEMparm_check: mesh must be 0 (msms) or 1 and 2 (NanoShaper)");
00163         rc = VRC_FAILURE;
00164     }
00165
00166     if (thee->outdata > 2 || thee->outdata < 0) {
00167         Vnm_print(2, "BEMparm_check: outdata must be 0, 1 (vtk), or 2 (not specified)");
00168         rc = VRC_FAILURE;
00169     }
00170
00171     return rc;
00172 }
00173
00174 VPUBLIC void BEMparm_copy(BEMparm *thee, BEMparm *parm) {
00175
00176     int i;
00177
00178     VASSERT(thee != VNULL);
00179     VASSERT(parm != VNULL);
00180
00181
00182     thee->type = parm->type;
00183     thee->parsed = parm->parsed;
00184
00185     /* *** GENERIC PARAMETERS *** */
00186
00187     /* *** TYPE 0 PARMS *** */
```

```
00188
00189     thee->tree_order = parm->tree_order;
00190     thee->settree_order = parm->settree_order;
00191     thee->tree_n0 = parm->tree_n0;
00192     thee->settree_n0 = parm->settree_n0;
00193     thee->mac = parm->mac;
00194     thee->setmac = parm->setmac;
00195
00196     thee->mesh = parm->mesh;
00197     thee->setmesh = parm->setmesh;
00198
00199     thee->outdata = parm->outdata;
00200     thee->setoutdata = parm->setoutdata;
00201
00202     /* *** TYPE 1 & 2 PARMS *** */
00203
00204     /* *** TYPE 2 PARMS *** */
00205     thee->nonlintype = parm->nonlintype;
00206     thee->setnonlintype = parm->setnonlintype;
00207 }
00208
00209
00210 VPRIVATE Vrc_Codes BEMparm_parseTREE_ORDER(BEMparm *thee, Vio *sock) {
00211
00212     char tok[VMAX_BUFSIZE];
00213     int ti;
00214
00215     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00216     if (sscanf(tok, "%d", &ti) == 0) {
00217         Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing TREE_ORDER \
00218 keyword!\n", tok);
00219         return VRC_WARNING;
00220     } else if (ti <= 0) {
00221         Vnm_print(2, "parseBEM:  tree_order must be greater than 0!\n");
00222         return VRC_WARNING;
00223     } else thee->tree_order = ti;
00224     thee->settree_order = 1;
00225     return VRC_SUCCESS;
00226
00227     VERROR1:
00228         Vnm_print(2, "parseBEM:  ran out of tokens!\n");
00229         return VRC_WARNING;
00230 }
00231
00232
00233 VPRIVATE Vrc_Codes BEMparm_parseTREE_N0(BEMparm *thee, Vio *sock) {
00234
00235     char tok[VMAX_BUFSIZE];
00236     int ti;
00237
00238     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00239     if (sscanf(tok, "%d", &ti) == 0) {
00240         Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing TREE_N0 \
00241 keyword!\n", tok);
00242         return VRC_WARNING;
00243     } else if (ti <= 0) {
00244         Vnm_print(2, "parseBEM:  tree_n0 must be greater than 0!\n");
00245         return VRC_WARNING;
00246     } else thee->tree_n0 = ti;
00247     thee->settree_n0 = 1;
00248     return VRC_SUCCESS;
00249
00250     VERROR1:
00251         Vnm_print(2, "parseBEM:  ran out of tokens!\n");
00252         return VRC_WARNING;
00253 }
00254
00255
00256 VPRIVATE Vrc_Codes BEMparm_parseMAC(BEMparm *thee, Vio *sock) {
00257
00258     char tok[VMAX_BUFSIZE];
00259     double tf;
00260
00261     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00262     if (sscanf(tok, "%lf", &tf) == 0) {
00263         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing mac \
00264 keyword!\n", tok);
00265         return VRC_WARNING;
00266     } else if (tf <= 0.0 || tf > 1.0) {
00267         Vnm_print(2, "parseBEM:  mac must be between 0 and 1!\n");
00268         return VRC_WARNING;
```

```
00269     } else thee->mac = tf;
00270     thee->setmac = 1;
00271     return VRC_SUCCESS;
00272
00273     VERROR1:
00274         Vnm_print(2, "parseBEM:  ran out of tokens!\n");
00275         return VRC_WARNING;
00276 }
00277
00278
00279 VPRIVATE Vrc_Codes BEMparm_parseMESH(BEMparm *thee, Vio *sock) {
00280
00281     char tok[VMAX_BUFSIZE];
00282
00283     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00284     if(strcmp(tok, "msms") == 0){
00285       thee->mesh = 0;
00286     }
00287     else if(strcmp(tok, "ses") == 0){
00288       thee->mesh = 1;
00289     }
00290     else if(strcmp(tok, "skin") == 0){
00291       thee->mesh = 2;
00292     }
00293     else{
00294       Vnm_print(2, "parseBEM: mesh option %s is not recognized! It must be one of msms, \
00295                     ses, or skin.\n", tok);
00296       return VRC_WARNING;
00297     }
00298
00299     thee->setmesh = 1;
00300     return VRC_SUCCESS;
00301
00302     VERROR1:
00303         Vnm_print(2, "parseBEM:  ran out of tokens!\n");
00304         return VRC_WARNING;
00305 }
00306
00307
00308 VPRIVATE Vrc_Codes BEMparm_parseOUTDATA(BEMparm *thee, Vio *sock) {
00309
00310     char tok[VMAX_BUFSIZE];
00311     int ti;
00312
00313     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00314     if (sscanf(tok, "%d", &ti) == 0) {
00315         Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing OUTDATA \
00316                       keyword!\n", tok);
00317         return VRC_WARNING;
00318     } else if (ti < 0 || ti > 2) {
00319         Vnm_print(2, "parseBEM:  outdata must be 0, 1 (vtk), \
00320                       or 2 (unspecified)!\n");
00321         return VRC_WARNING;
00322     } else thee->outdata = ti;
00323     thee->setoutdata = 1;
00324     return VRC_SUCCESS;
00325
00326     VERROR1:
00327         Vnm_print(2, "parseBEM:  ran out of tokens!\n");
00328         return VRC_WARNING;
00329 }
00330
00331
00332 VPUBLIC Vrc_Codes BEMparm_parseToken(BEMparm *thee, char tok[VMAX_BUFSIZE],
00333   Vio *sock) {
00334
00335     if (thee == VNULL) {
00336         Vnm_print(2, "parseBEM:  got NULL thee!\n");
00337         return VRC_WARNING;
00338     }
00339     if (sock == VNULL) {
00340         Vnm_print(2, "parseBEM:  got NULL socket!\n");
00341         return VRC_WARNING;
00342     }
00343
00344     Vnm_print(0, "BEMparm_parseToken:  trying %s...\n", tok);
00345
00346
00347     if (Vstring_strcasecmp(tok, "tree_order") == 0) {
00348         return BEMparm_parseTREE_ORDER(thee, sock);
00349     } else if (Vstring_strcasecmp(tok, "tree_n0") == 0) {
```

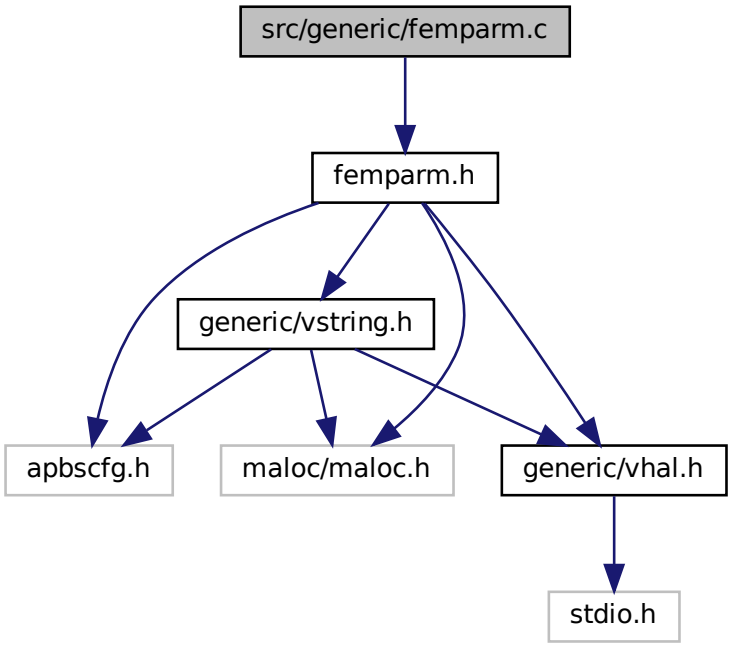
```
00350         return BEMparm_parseTREE_N0(thee, sock);
00351     } else if (Vstring_strcasecmp(tok, "mac") == 0) {
00352         return BEMparm_parseMAC(thee, sock);
00353     } else if (Vstring_strcasecmp(tok, "mesh") == 0) {
00354         return BEMparm_parseMESH(thee, sock);
00355     } else if (Vstring_strcasecmp(tok, "outdata") == 0) {
00356         return BEMparm_parseOUTDATA(thee, sock);
00357     } else {
00358         Vnm_print(2, "parseBEM:  Unrecognized keyword (%s)!\n", tok);
00359         return VRC_WARNING;
00360     }
00361
00362     return VRC_FAILURE;
00363
00364 }
```

9.19 src/generic/femparm.c File Reference

Class FEMparm methods.

`#include "femparm.h"`

Include dependency graph for femparm.c:



Functions

- VPUBLIC FEMparm * FEMparm_ctor (FEMparm_CalcType type)

  Construct FEMparm.
- VPUBLIC int FEMparm_ctor2 (FEMparm *thee, FEMparm_CalcType type)

  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.

9.19.1 Detailed Description

Class FEMparm methods.

Author

Nathan Baker

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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) 2002-2020, 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.

9.20 femparm.c

```
00001
00057 #include "femparm.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_MGPARM)
00062
00063 #endif /* if !defined(VINLINE_MGPARM) */
00064
00065 VPUBLIC FEMparm* FEMparm_ctor(FEMparm_CalcType type) {
00066
00067     /* Set up the structure */
00068     FEMparm *thee = VNULL;
00069     thee = (FEMparm*)Vmem_malloc(VNULL, 1, sizeof(FEMparm));
00070     VASSERT( thee != VNULL);
00071     VASSERT( FEMparm_ctor2(thee, type) );
00072
00073     return thee;
00074 }
00075
00076 VPUBLIC int FEMparm_ctor2(FEMparm *thee,
00077                           FEMparm_CalcType type
00078                           ) {
00079
00080     if (thee == VNULL) return 0;
00081
00082     thee->parsed = 0;
00083     thee->type = type;
00084     thee->settype = 1;
00085
00086     thee->setglen = 0;
00087     thee->setetol = 0;
00088     thee->setekey = 0;
00089     thee->setakeyPRE = 0;
00090     thee->setakeySOLVE = 0;
00091     thee->settargetNum = 0;
00092     thee->settargetRes = 0;
00093     thee->setmaxsolve = 0;
00094     thee->setmaxvert = 0;
00095     thee->useMesh = 0;
00096
00097     return 1;
00098 }
00099
00100 VPUBLIC void FEMparm_copy(
00101                           FEMparm *thee,
00102                           FEMparm *source
00103                           ) {
```

```
00104
00105     int i;
00106
00107     thee->parsed = source->parsed;
00108     thee->type = source->type;
00109     thee->settype = source->settype;
00110     for (i=0; i<3; i++) thee->glen[i] = source->glen[i];
00111     thee->setglen = source->setglen;
00112     thee->etol = source->etol;
00113     thee->setetol = source->setetol;
00114     thee->ekey = source->ekey;
00115     thee->setekey = source->setekey;
00116     thee->akeyPRE = source->akeyPRE;
00117     thee->setakeyPRE = source->setakeyPRE;
00118     thee->akeySOLVE = source->akeySOLVE;
00119     thee->setakeySOLVE = source->setakeySOLVE;
00120     thee->targetNum = source->targetNum;
00121     thee->settargetNum = source->settargetNum;
00122     thee->targetRes = source->targetRes;
00123     thee->settargetRes = source->settargetRes;
00124     thee->maxsolve = source->maxsolve;
00125     thee->setmaxsolve = source->setmaxsolve;
00126     thee->maxvert = source->maxvert;
00127     thee->setmaxvert = source->setmaxvert;
00128     thee->pkey = source->pkey;
00129     thee->useMesh = source->useMesh;
00130     thee->meshID = source->meshID;
00131 }
00132
00133 VPUBLIC void FEMparm_dtor(FEMparm **thee) {
00134     if ((*thee) != VNULL) {
00135         FEMparm_dtor2(*thee);
00136         Vmem_free(VNULL, 1, sizeof(FEMparm), (void **)thee);
00137         (*thee) = VNULL;
00138     }
00139 }
00140
00141 VPUBLIC void FEMparm_dtor2(FEMparm *thee) { ; }
00142
00143 VPUBLIC int FEMparm_check(FEMparm *thee) {
00144
00145     int rc;
00146     rc = 1;
00147
00148     if (!thee->parsed) {
00149         Vnm_print(2, "FEMparm_check:  not filled!\n");
00150         return 0;
00151     }
00152     if (!thee->settype) {
00153         Vnm_print(2, "FEMparm_check:  type not set!\n");
00154         rc = 0;
00155     }
00156     if (!thee->setglen) {
00157         Vnm_print(2, "FEMparm_check:  glen not set!\n");
00158         rc = 0;
00159     }
00160     if (!thee->setetol) {
00161         Vnm_print(2, "FEMparm_check:  etol not set!\n");
00162         rc = 0;
00163     }
00164     if (!thee->setekey) {
00165         Vnm_print(2, "FEMparm_check:  ekey not set!\n");
00166         rc = 0;
00167     }
00168     if (!thee->setakeyPRE) {
00169         Vnm_print(2, "FEMparm_check:  akeyPRE not set!\n");
00170         rc = 0;
00171     }
00172     if (!thee->setakeySOLVE) {
00173         Vnm_print(2, "FEMparm_check:  akeySOLVE not set!\n");
00174         rc = 0;
00175     }
00176     if (!thee->settargetNum) {
00177         Vnm_print(2, "FEMparm_check:  targetNum not set!\n");
00178         rc = 0;
00179     }
00180     if (!thee->settargetRes) {
00181         Vnm_print(2, "FEMparm_check:  targetRes not set!\n");
00182         rc = 0;
00183     }
00184     if (!thee->setmaxsolve) {
```

```
00185         Vnm_print(2, "FEMparm_check:  maxsolve not set!\n");
00186         rc = 0;
00187     }
00188     if (!thee->setmaxvert) {
00189         Vnm_print(2, "FEMparm_check:  maxvert not set!\n");
00190         rc = 0;
00191     }
00192 
00193     return rc;
00194 }
00195 
00196 VPRIVATE Vrc_Codes FEMparm_parseDOMAINLENGTH(FEMparm *thee,
00197                                              Vio *sock
00198                                              ) {
00199 
00200     int i;
00201     double tf;
00202     char tok[VMAX_BUFSIZE];
00203 
00204     for (i=0; i<3; i++) {
00205         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00206         if (sscanf(tok, "%lf", &tf) == 0) {
00207             Vnm_print(2, "parseFE:  Read non-double (%s) while parsing \
00208 DOMAINLENGTH keyword!\n", tok);
00209             return VRC_FAILURE;
00210         }
00211         thee->glen[i] = tf;
00212     }
00213     thee->setglen = 1;
00214     return VRC_SUCCESS;
00215 VERROR1:
00216     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00217     return VRC_FAILURE;
00218 
00219 }
00220 
00221 VPRIVATE Vrc_Codes FEMparm_parseETOL(FEMparm *thee,
00222                                      Vio *sock
00223                                      ) {
00224 
00225     double tf;
00226     char tok[VMAX_BUFSIZE];
00227 
00228     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00229     if (sscanf(tok, "%lf", &tf) == 0) {
00230         Vnm_print(2, "parseFE:  Read non-double (%s) while parsing \
00231 ETOL keyword!\n", tok);
00232         return VRC_FAILURE;
00233     }
00234     thee->etol = tf;
00235     thee->setetol = 1;
00236     return VRC_SUCCESS;
00237 VERROR1:
00238     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00239     return VRC_FAILURE;
00240 
00241 
00242 }
00243 
00244 VPRIVATE Vrc_Codes FEMparm_parseEKEY(FEMparm *thee,
00245                                      Vio *sock
00246                                      ) {
00247 
00248     char tok[VMAX_BUFSIZE];
00249     Vrc_Codes vrc = VRC_FAILURE;
00250 
00251     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00252     if (Vstring_strcasecmp(tok, "simp") == 0) {
00253         thee->ekey = FET_SIMP;
00254         thee->setekey = 1;
00255         vrc = VRC_SUCCESS;
00256     } else if (Vstring_strcasecmp(tok, "glob") == 0) {
00257         thee->ekey = FET_GLOB;
00258         thee->setekey = 1;
00259         vrc = VRC_SUCCESS;
00260     } else if (Vstring_strcasecmp(tok, "frac") == 0) {
00261         thee->ekey = FET_FRAC;
00262         thee->setekey = 1;
00263         vrc = VRC_SUCCESS;
00264     } else {
00265         Vnm_print(2, "parseFE:  undefined value (%s) for ekey!\n", tok);
```

```
00266         vrc = VRC_FAILURE;
00267     }
00268
00269     return vrc;
00270 VERROR1:
00271     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00272     return VRC_FAILURE;
00273
00274 }
00275
00276 VPRIVATE Vrc_Codes FEMparm_parseAKEYPRE(FEMparm *thee, Vio *sock) {
00277
00278     char tok[VMAX_BUFSIZE];
00279     Vrc_Codes vrc = VRC_FAILURE;
00280
00281     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00282     if (Vstring_strcasecmp(tok, "unif") == 0) {
00283         thee->akeyPRE = FRT_UNIF;
00284         thee->setakeyPRE = 1;
00285         vrc =  VRC_SUCCESS;
00286     } else if (Vstring_strcasecmp(tok, "geom") == 0) {
00287         thee->akeyPRE = FRT_GEOM;
00288         thee->setakeyPRE = 1;
00289         vrc =  VRC_SUCCESS;
00290     } else {
00291         Vnm_print(2, "parseFE:  undefined value (%s) for akeyPRE!\n", tok);
00292         vrc =  VRC_FAILURE;
00293     }
00294
00295     return vrc;
00296
00297 VERROR1:
00298     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00299     return VRC_FAILURE;
00300
00301 }
00302
00303 VPRIVATE Vrc_Codes FEMparm_parseAKEYSOLVE(FEMparm *thee, Vio *sock) {
00304
00305     char tok[VMAX_BUFSIZE];
00306     Vrc_Codes vrc = VRC_FAILURE;
00307
00308     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00309     if (Vstring_strcasecmp(tok, "resi") == 0) {
00310         thee->akeySOLVE = FRT_RESI;
00311         thee->setakeySOLVE = 1;
00312         vrc =  VRC_SUCCESS;
00313     } else if (Vstring_strcasecmp(tok, "dual") == 0) {
00314         thee->akeySOLVE = FRT_DUAL;
00315         thee->setakeySOLVE = 1;
00316         vrc =  VRC_SUCCESS;
00317     } else if (Vstring_strcasecmp(tok, "loca") == 0) {
00318         thee->akeySOLVE = FRT_LOCA;
00319         thee->setakeySOLVE = 1;
00320         vrc =  VRC_SUCCESS;
00321     } else {
00322         Vnm_print(2, "parseFE:  undefined value (%s) for akeyPRE!\n", tok);
00323         vrc =  VRC_FAILURE;
00324     }
00325
00326     return vrc;
00327 VERROR1:
00328     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00329     return VRC_SUCCESS;
00330
00331 }
00332
00333 VPRIVATE Vrc_Codes FEMparm_parseTARGETNUM(FEMparm *thee, Vio *sock) {
00334
00335     char tok[VMAX_BUFSIZE];
00336     int ti;
00337
00338     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00339     if (sscanf(tok, "%d", &ti) == 0) {
00340         Vnm_print(2, "parseFE:  read non-int (%s) for targetNum!\n", tok);
00341         return VRC_FAILURE;
00342     }
00343     thee->targetNum = ti;
00344     thee->settargetNum = 1;
00345     return VRC_SUCCESS;
00346 VERROR1:
```

```
00347     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00348     return VRC_FAILURE;
00349
00350 }
00351
00352 VPRIVATE Vrc_Codes FEMparm_parseTARGETRES(FEMparm *thee, Vio *sock) {
00353
00354     char tok[VMAX_BUFSIZE];
00355     double tf;
00356
00357     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00358     if (sscanf(tok, "%lf", &tf) == 0) {
00359         Vnm_print(2, "parseFE:  read non-double (%s) for targetNum!\n",
00360           tok);
00361         return VRC_FAILURE;
00362     }
00363     thee->targetRes = tf;
00364     thee->settargetRes = 1;
00365     return VRC_SUCCESS;
00366 VERROR1:
00367     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00368     return VRC_FAILURE;
00369
00370 }
00371
00372 VPRIVATE Vrc_Codes FEMparm_parseMAXSOLVE(FEMparm *thee, Vio *sock) {
00373
00374     char tok[VMAX_BUFSIZE];
00375     int ti;
00376
00377     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00378     if (sscanf(tok, "%d", &ti) == 0) {
00379         Vnm_print(2, "parseFE:  read non-int (%s) for maxsolve!\n", tok);
00380         return VRC_FAILURE;
00381     }
00382     thee->maxsolve = ti;
00383     thee->setmaxsolve = 1;
00384     return VRC_SUCCESS;
00385 VERROR1:
00386     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00387     return VRC_FAILURE;
00388
00389 }
00390
00391 VPRIVATE Vrc_Codes FEMparm_parseMAXVERT(FEMparm *thee, Vio *sock) {
00392
00393     char tok[VMAX_BUFSIZE];
00394     int ti;
00395
00396     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00397     if (sscanf(tok, "%d", &ti) == 0) {
00398         Vnm_print(2, "parseFE:  read non-int (%s) for maxvert!\n", tok);
00399         return VRC_FAILURE;
00400     }
00401     thee->maxvert = ti;
00402     thee->setmaxvert = 1;
00403     return VRC_SUCCESS;
00404
00405 VERROR1:
00406     Vnm_print(2, "parseFE:  ran out of tokens!\n");
00407     return VRC_FAILURE;
00408
00409 }
00410
00411 VPRIVATE Vrc_Codes FEMparm_parseUSEMESH(FEMparm *thee, Vio *sock) {
00412     char tok[VMAX_BUFSIZE];
00413     int ti;
00414
00415     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00416     if (sscanf(tok, "%d", &ti) == 0) {
00417         Vnm_print(2, "parseFE:  read non-int (%s) for usemesh!\n", tok);
00418         return VRC_FAILURE;
00419     }
00420     thee->useMesh = 1;
00421     thee->meshID = ti;
00422
00423     return VRC_SUCCESS;
00424
00425 VERROR1:
00426     Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00427     return VRC_FAILURE;
```

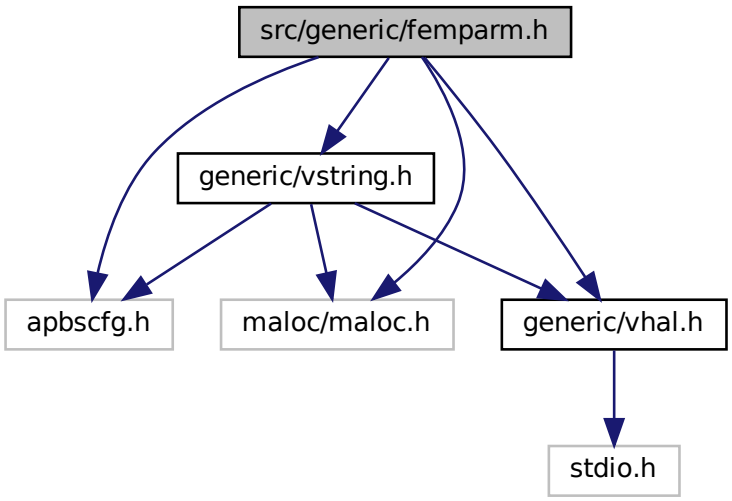
```
00428 }
00429
00430
00431 VPUBLIC Vrc_Codes FEMparm_parseToken(FEMparm *thee, char tok[VMAX_BUFSIZE],
00432   Vio *sock) {
00433
00434     //int i, ti; // gcc says unused
00435     //double tf; // gcc says unused
00436
00437     if (thee == VNULL) {
00438         Vnm_print(2, "parseFE:  got NULL thee!\n");
00439         return VRC_FAILURE;
00440     }
00441
00442     if (sock == VNULL) {
00443         Vnm_print(2, "parseFE:  got NULL socket!\n");
00444         return VRC_FAILURE;
00445     }
00446
00447     if (Vstring_strcasecmp(tok, "domainLength") == 0) {
00448         return FEMparm_parseDOMAINLENGTH(thee, sock);
00449     } else if (Vstring_strcasecmp(tok, "etol") == 0) {
00450         return FEMparm_parseETOL(thee, sock);
00451     } else if (Vstring_strcasecmp(tok, "ekey") == 0) {
00452         return FEMparm_parseEKEY(thee, sock);
00453     } else if (Vstring_strcasecmp(tok, "akeyPRE") == 0) {
00454         return FEMparm_parseAKEYPRE(thee, sock);
00455     } else if (Vstring_strcasecmp(tok, "akeySOLVE") == 0) {
00456         return FEMparm_parseAKEYSOLVE(thee, sock);
00457     } else if (Vstring_strcasecmp(tok, "targetNum") == 0) {
00458         return FEMparm_parseTARGETNUM(thee, sock);
00459     } else if (Vstring_strcasecmp(tok, "targetRes") == 0) {
00460         return FEMparm_parseTARGETRES(thee, sock);
00461     } else if (Vstring_strcasecmp(tok, "maxsolve") == 0) {
00462         return FEMparm_parseMAXSOLVE(thee, sock);
00463     } else if (Vstring_strcasecmp(tok, "maxvert") == 0) {
00464         return FEMparm_parseMAXVERT(thee, sock);
00465     } else if (Vstring_strcasecmp(tok, "usemesh") == 0) {
00466         return FEMparm_parseUSEMESH(thee, sock);
00467     }
00468
00469     return VRC_WARNING;
00470
00471 }
```

9.21 src/generic/femparm.h File Reference

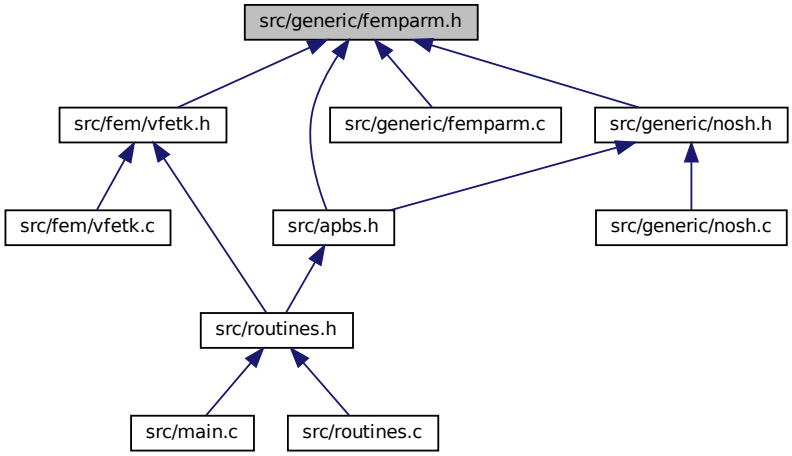
Contains declarations for class APOLparm.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/vstring.h"
```

Include dependency graph for femparm.h:



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



Data Structures

- struct sFEMparm

  Parameter structure for FEM-specific variables from input files.

Typedefs

- typedef enum eFEMparm_EtolType FEMparm_EtolType

  Declare FEparm_EtolType type.
- typedef enum eFEMparm_EstType FEMparm_EstType

  Declare FEMparm_EstType type.
- typedef enum eFEMparm_CalcType FEMparm_CalcType

  Declare FEMparm_CalcType type.
- typedef struct sFEMparm FEMparm

  Declaration of the FEMparm class as the FEMparm structure.

Enumerations

- enum eFEMparm_EtolType { FET_SIMP =0 , FET_GLOB =1 , FET_FRAC =2 }

  Adaptive 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

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) 2010-2020 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

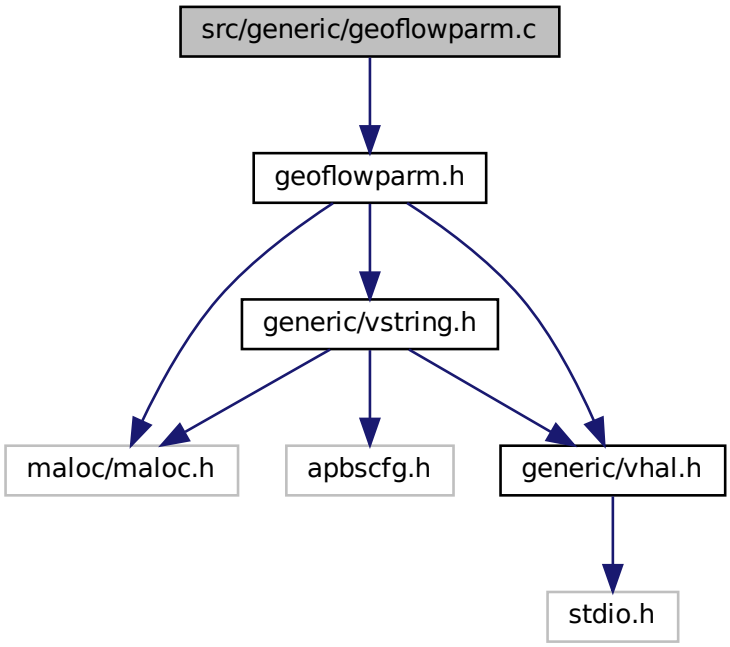
```
00001
00063 #ifndef _FEMPARM_H_
00064 #define _FEMPARM_H_
00065
00066 /* Generic header files */
00067 #include "apbscfg.h"
00068
00069 #include "maloc/maloc.h"
00070
00071 #include "generic/vhal.h"
00072 #include "generic/vstring.h"
00073
00079 enum eFEMparm_EtolType {
00080     FET_SIMP=0,
00081     FET_GLOB=1,
00082     FET_FRAC=2
00083 };
```

```
00084
00090 typedef enum eFEMparm_EtolType FEMparm_EtolType;
00091
00098 enum eFEMparm_EstType {
00099     FRT_UNIF=0,
00100     FRT_GEOM=1,
00101     FRT_RESI=2,
00102     FRT_DUAL=3,
00104     FRT_LOCA=4
00105 };
00106
00111 typedef enum eFEMparm_EstType FEMparm_EstType;
00112
00117 enum eFEMparm_CalcType {
00118     FCT_MANUAL,
00119     FCT_NONE
00120 };
00121
00126 typedef enum eFEMparm_CalcType FEMparm_CalcType;
00127
00133 struct sFEMparm {
00134
00135     int parsed;
00138     FEMparm_CalcType type;
00139     int settype;
00140     double glen[3];
00141     int setglen;
00142     double etol;
00144     int setetol;
00145     FEMparm_EtolType ekey;
00147     int setekey;
00148     FEMparm_EstType akeyPRE;
00151     int setakeyPRE;
00152     FEMparm_EstType akeySOLVE;
00154     int setakeySOLVE;
00155     int targetNum;
00159     int settargetNum;
00160     double targetRes;
00164     int settargetRes;
00165     int maxsolve;
00166     int setmaxsolve;
00167     int maxvert;
00169     int setmaxvert;
00170     int pkey;
00173     int useMesh;
00174     int meshID;
00176 };
00177
00182 typedef struct sFEMparm FEMparm;
00183
00184 /* ////////////////////////////////////////////////////////////////////////////
00185 // Class NOsh: Non-inlineable methods (nosh.c)
00187
00194 VEXTERNC FEMparm* FEMparm_ctor(FEMparm_CalcType type);
00195
00203 VEXTERNC int FEMparm_ctor2(FEMparm *thee, FEMparm_CalcType type);
00204
00210 VEXTERNC void FEMparm_dtor(FEMparm **thee);
00211
00217 VEXTERNC void FEMparm_dtor2(FEMparm *thee);
00218
00226 VEXTERNC int FEMparm_check(FEMparm *thee);
00227
00234 VEXTERNC void FEMparm_copy(FEMparm *thee, FEMparm *source);
00235
00246 VEXTERNC Vrc_Codes FEMparm_parseToken(FEMparm *thee, char tok[VMAX_BUFSIZE],
00247   Vio *sock);
00248
00249 #endif
00250
```

9.23 src/generic/geoflowparm.c File Reference

Class GEOFLOWparm methods.

```
#include "geoflowparm.h"
```
Include dependency graph for geoflowparm.c:



Functions

- VPUBLIC GEOFLOWparm * GEOFLOWparm_ctor (GEOFLOWparm_CalcType type)

  Construct GEOFLOWparm object.
- VPUBLIC Vrc_Codes GEOFLOWparm_ctor2 (GEOFLOWparm *thee, GEOFLOWparm_CalcType type)

  FORTRAN stub to construct GEOFLOWparm object ?????????!!!!!!!
- VPUBLIC void GEOFLOWparm_dtor (GEOFLOWparm **thee)

  Object destructor.
- VPUBLIC void GEOFLOWparm_dtor2 (GEOFLOWparm *thee)

  FORTRAN stub for object destructor ?????????!!!!!!!!!!!
- VPUBLIC Vrc_Codes GEOFLOWparm_check (GEOFLOWparm *thee)

  Consistency check for parameter values stored in object.
- VPUBLIC void GEOFLOWparm_copy (GEOFLOWparm *thee, GEOFLOWparm *parm)

  copy GEOFLOWparm object int thee.
- Vrc_Codes **FUBAR** (const char *name)
- Vrc_Codes **parseNonNeg** (double *tf, double def, int *set, char *name, Vio *sock)
- VPRIVATE Vrc_Codes **GEOFLOWparm_parseVDW** (GEOFLOWparm *thee, Vio *sock)
- VPRIVATE Vrc_Codes **GEOFLOWparm_parseETOL** (GEOFLOWparm *thee, Vio *sock)
- VPUBLIC Vrc_Codes GEOFLOWparm_parseToken (GEOFLOWparm *thee, char tok[VMAX_BUFSIZE], Vio *sock)

  Parse an MG keyword from an input file.

9.23.1 Detailed Description

Class GEOFLOWparm methods.

Author

Andrew Stevens

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file geoflowparm.c.

9.24 geoflowparm.c

```
00001
```

```
00057 #include "geoflowparm.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_MGPARM)
00062
00063 #endif /* if !defined(VINLINE_MGPARM) */
00064
00065
00066 VPUBLIC GEOFLOWparm* GEOFLOWparm_ctor(GEOFLOWparm_CalcType type) {
00067
00068     /* Set up the structure */
00069     GEOFLOWparm *thee = VNULL;
00070     thee = (GEOFLOWparm*)Vmem_malloc(VNULL, 1, sizeof(GEOFLOWparm));
00071     VASSERT( thee != VNULL);
00072     VASSERT( GEOFLOWparm_ctor2(thee, type) == VRC_SUCCESS );
00073
00074     return thee;
00075 }
00076
00077 VPUBLIC Vrc_Codes GEOFLOWparm_ctor2(GEOFLOWparm *thee, GEOFLOWparm_CalcType type) {
00078
00079     int i;
00080
00081     if (thee == VNULL) return VRC_FAILURE;
00082
00083     thee->parsed = 0;
00084     thee->type = type;
00085     thee->vdw = 0;
00086     thee->etol = 1.0e-6;
00087
00088     return VRC_SUCCESS;
00089 }
00090
00091 VPUBLIC void GEOFLOWparm_dtor(GEOFLOWparm **thee) {
00092     if ((*thee) != VNULL) {
00093         GEOFLOWparm_dtor2(*thee);
00094         Vmem_free(VNULL, 1, sizeof(GEOFLOWparm), (void **)thee);
00095         (*thee) = VNULL;
00096     }
00097 }
00098
00099 VPUBLIC void GEOFLOWparm_dtor2(GEOFLOWparm *thee) { ; }
00100
00101 VPUBLIC Vrc_Codes GEOFLOWparm_check(GEOFLOWparm *thee) {
00102
00103     Vrc_Codes rc;
00104
00105     rc = VRC_SUCCESS;
00106
00107     Vnm_print(0, "GEOFLOWparm_check:  checking GEOFLOWparm object of type %d.\n",
00108               thee->type);
00109
00110     /* Check to see if we were even filled... */
00111     if (!thee->parsed) {
00112         Vnm_print(2, "GEOFLOWparm_check:  not filled!\n");
00113         return VRC_FAILURE;
00114     }
00115
00116
00117     /* Check type settings */
00118     //if ((thee->type != GFCT_MANUAL)&& (thee->type != GFCT_AUTO)&& (thee->type != GFCT_NONE)) {
00119     if(thee->type != GFCT_AUTO) {
00120          Vnm_print(2,"GEOFLOWparm_check: type not set");
00121          rc = VRC_FAILURE;
00122     }
00123
00124     return rc;
00125 }
00126
00127 VPUBLIC void GEOFLOWparm_copy(GEOFLOWparm *thee, GEOFLOWparm *parm) {
00128     VASSERT(thee != VNULL);
00129     VASSERT(parm != VNULL);
00130
00131     thee->type = parm->type;
00132     thee->parsed = parm->parsed;
00133
00134     thee->vdw = parm->vdw;
00135     thee->etol = parm->etol;
00136 }
00137
```

```
00138 Vrc_Codes FUBAR(const char* name){
00139     Vnm_print(2, "parseGEOFLOW:  ran out of tokens on %s!\n", name);
00140     return VRC_WARNING;
00141 }
00142
00143 Vrc_Codes parseNonNeg(double* tf, double def, int* set, char* name, Vio* sock){
00144     char tok[VMAX_BUFSIZE];
00145     if(Vio_scanf(sock, "%s", tok) == 0) {
00146         *tf = def;
00147         return FUBAR(name);
00148     }
00149
00150     if (sscanf(tok, "%lf", tf) == 0){
00151         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing %s keyword!\n", tok, name);
00152         *tf = def;
00153         return VRC_WARNING;
00154     }else if(*tf < 0.0){
00155         Vnm_print(2, "parseGEOFLOW:  %s must be greater than 0!\n", name);
00156         *tf = def;
00157         return VRC_WARNING;
00158     }
00159
00160     *set = 1;
00161     return VRC_SUCCESS;
00162 }
00163
00164 VPRIVATE Vrc_Codes GEOFLOWparm_parseVDW(GEOFLOWparm *thee, Vio *sock){
00165     const char* name = "vdw";
00166     char tok[VMAX_BUFSIZE];
00167  int tf;
00168     if(Vio_scanf(sock, "%s", tok) == 0) {
00169         return FUBAR(name);
00170     }
00171
00172
00173     if (sscanf(tok, "%u", &tf) == 0){
00174         Vnm_print(2, "NOsh:  Read non-unsigned int (%s) while parsing %s keyword!\n", tok, name);
00175         return VRC_WARNING;
00176     }else if(tf != 0 && tf != 1){
00177         Vnm_print(2, "parseGEOFLOW:  %s must be 0 or 1!\n", name);
00178         return VRC_WARNING;
00179     }else{
00180         thee->vdw = tf;
00181     }
00182     thee->setvdw = 1;
00183     return VRC_SUCCESS;
00184 }
00185
00186 VPRIVATE Vrc_Codes GEOFLOWparm_parseETOL(GEOFLOWparm *thee, Vio *sock){
00187
00188  char tok[VMAX_BUFSIZE];
00189  double tf;
00190
00191  VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00192  if(sscanf(tok, "%lf", &tf) == 0){
00193   Vnm_print(2, "NOsh: Read non-float (%s) while parsing etol keyword!\n", tok);
00194   return VRC_WARNING;
00195  } else if(tf <= 0.0) {
00196   Vnm_print(2,"parseGEOFLOW: etol must be greater than 0!\n");
00197   return VRC_WARNING;
00198  } else {
00199   thee->etol = tf;
00200  }
00201
00202
00203  return VRC_SUCCESS;
00204
00205
00206  VERROR1:
00207   Vnm_print(2, "parseGEOFLOW: ran out of tokens!\n");
00208   return VRC_WARNING;
00209
00210 }
00211
00212 VPUBLIC Vrc_Codes GEOFLOWparm_parseToken(GEOFLOWparm *thee, char tok[VMAX_BUFSIZE],
00213   Vio *sock) {
00214
00215     if (thee == VNULL) {
00216         Vnm_print(2, "parseGEOFLOW:  got NULL thee!\n");
00217         return VRC_WARNING;
00218     }
```

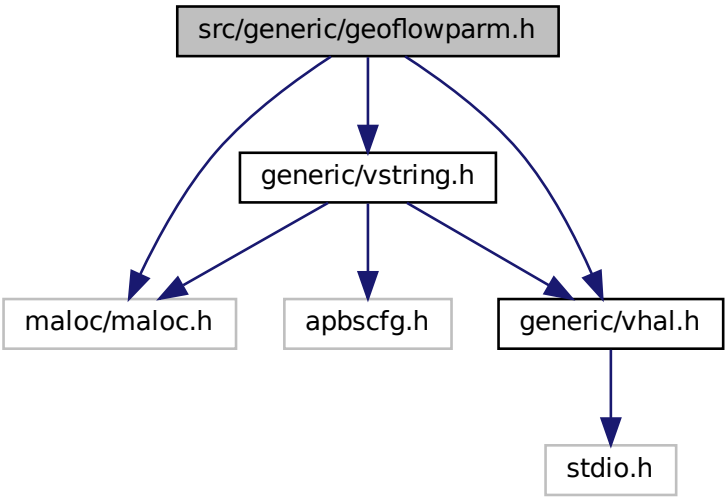
```
00219     if (sock == VNULL) {
00220         Vnm_print(2, "parseGEOFLOW:  got NULL socket!\n");
00221         return VRC_WARNING;
00222     }
00223
00224     Vnm_print(0, "GEOFLOWparm_parseToken:  trying %s...\n", tok);
00225
00226     if (Vstring_strcasecmp(tok, "vdwdisp") == 0) {
00227         return GEOFLOWparm_parseVDW(thee, sock);
00228     } else if(Vstring_strcasecmp(tok, "etol") == 0){
00229      return GEOFLOWparm_parseETOL(thee, sock);
00230     }else {
00231         Vnm_print(2, "parseGEOFLOW:  Unrecognized keyword (%s)!\n", tok);
00232         return VRC_WARNING;
00233     }
00234
00235
00236     return VRC_FAILURE;
00237 }
```

9.25 src/generic/geoflowparm.h File Reference

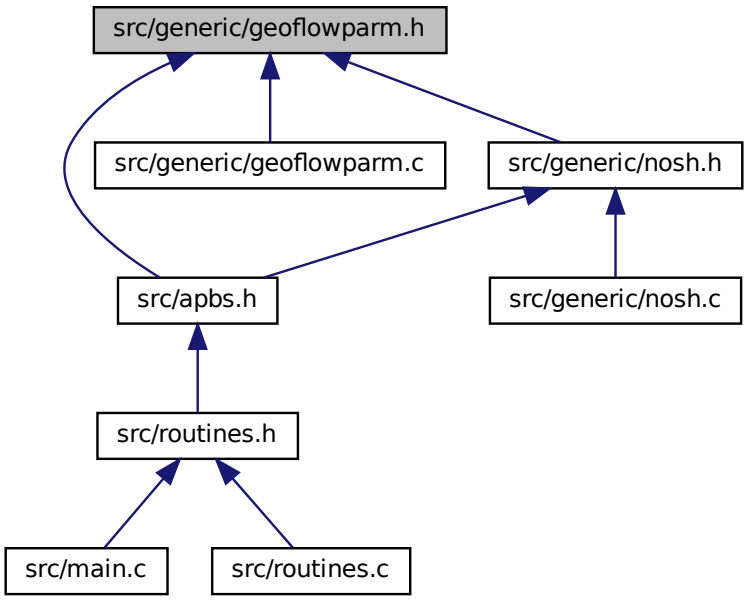
Contains declarations for class GEOFLOWparm.

```
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/vstring.h"
```

Include dependency graph for geoflowparm.h:



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



Data Structures

- struct sGEOFLOWparm

  Parameter structure for GEOFLOW-specific variables from input files.

Typedefs

- typedef enum eGEOFLOWparm_CalcType GEOFLOWparm_CalcType

  Declare GEOFLOWparm_CalcType type.
- typedef struct sGEOFLOWparm GEOFLOWparm

  Parameter structure for GEOFLOW-specific variables from input files.

Enumerations

- enum eGEOFLOWparm_CalcType { GFCT_AUTO =1 }

  Calculation type.

Functions

- VEXTERNC GEOFLOWparm * GEOFLOWparm_ctor (GEOFLOWparm_CalcType type)

  Construct GEOFLOWparm object.
- VEXTERNC Vrc_Codes GEOFLOWparm_ctor2 (GEOFLOWparm *thee, GEOFLOWparm_CalcType type)

  FORTRAN stub to construct GEOFLOWparm object ?????????!!!!!!!
- VEXTERNC void GEOFLOWparm_dtor (GEOFLOWparm **thee)

Object destructor.

- VEXTERNC void GEOFLOWparm_dtor2 (GEOFLOWparm *thee)

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

- VEXTERNC Vrc_Codes GEOFLOWparm_check (GEOFLOWparm *thee)

  Consistency check for parameter values stored in object.

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

  Parse an MG keyword from an input file.

- VEXTERNC void GEOFLOWparm_copy (GEOFLOWparm *thee, GEOFLOWparm *parm)

  copy GEOFLOWparm object int thee.

- VPRIVATE Vrc_Codes **GEOFLOWparm_parseVDW** (GEOFLOWparm *thee, Vio *sock)
- VPRIVATE Vrc_Codes **GEOFLOWparm_parseETOL** (GEOFLOWparm *thee, Vio *sock)

9.25.1 Detailed Description

Contains declarations for class GEOFLOWparm.

Version

Id

Author

Andrew Stevens

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 Battelle Memorial Institute. Developed at the
* Pacific Northwest National Laboratory, operated by Battelle Memorial
* Institute, Pacific Northwest Division for the U.S. Department of Energy.
*
* Portions Copyright (c) 2002-2010, Washington University in St. Louis.
* Portions Copyright (c) 2002-2010, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
```

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

Definition in file geoflowparm.h.

9.26 geoflowparm.h

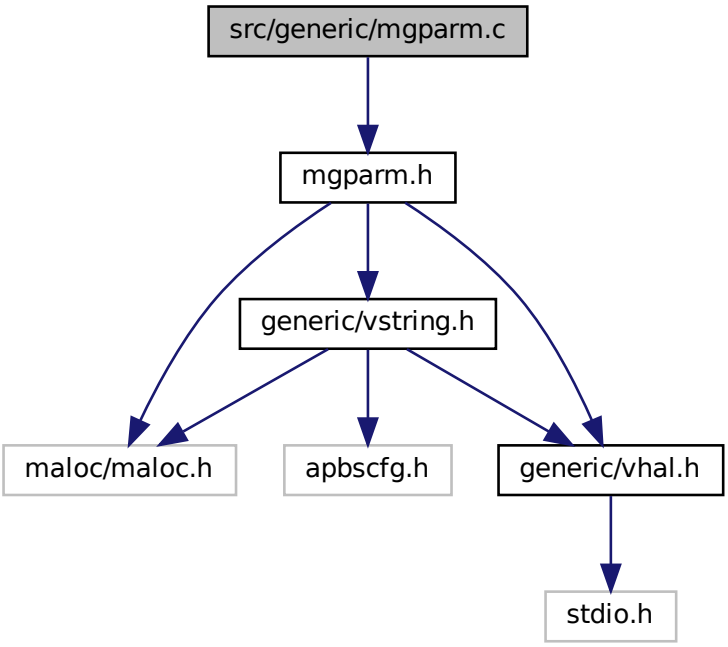
```
00001
00064 #ifndef _GEOFLOWPARM_H_
00065 #define _GEOFLOWPARM_H_
00066
00067 /* Generic header files */
00068 #include "maloc/maloc.h"
00069
00070 #include "generic/vhal.h"
00071 #include "generic/vstring.h"
00072
00077 enum eGEOFLOWparm_CalcType {
00078     //other methods disabled for now only auto currently implemented.
00079  //GFCT_MANUAL=0,  /**< GEOFLOW-manual */
00080     GFCT_AUTO=1,
00081     //GFCT_NONE=2 /**< not defined */
00082 };
00083
00088 typedef enum eGEOFLOWparm_CalcType GEOFLOWparm_CalcType;
00089
00098 typedef struct sGEOFLOWparm {
00099
00100     GEOFLOWparm_CalcType type;
00101     int parsed;
00103     /* *** GENERIC PARAMETERS *** */
00104     int vdw;
00105     int setvdw;
00106     double etol;
00108 } GEOFLOWparm;
00109
00116 VEXTERNC GEOFLOWparm*  GEOFLOWparm_ctor(GEOFLOWparm_CalcType type);
00117
00125 VEXTERNC Vrc_Codes      GEOFLOWparm_ctor2(GEOFLOWparm *thee, GEOFLOWparm_CalcType type);
00126
00132 VEXTERNC void     GEOFLOWparm_dtor(GEOFLOWparm **thee);
00133
00139 VEXTERNC void     GEOFLOWparm_dtor2(GEOFLOWparm *thee);
00140
00147 VEXTERNC Vrc_Codes      GEOFLOWparm_check(GEOFLOWparm *thee);
00148
00158 VEXTERNC Vrc_Codes      GEOFLOWparm_parseToken(GEOFLOWparm *thee, char tok[VMAX_BUFSIZE],
00159                     Vio *sock);
00167 VEXTERNC void GEOFLOWparm_copy(GEOFLOWparm *thee, GEOFLOWparm *parm);
00168
00169 VPRIVATE Vrc_Codes GEOFLOWparm_parseVDW(GEOFLOWparm *thee, Vio *sock);
00170
00171 VPRIVATE Vrc_Codes GEOFLOWparm_parseETOL(GEOFLOWparm *thee, Vio *sock);
00172
00173
00174
00175 #endif
00176
```

9.27 src/generic/mgparm.c File Reference

Class MGparm methods.

```
#include "mgparm.h"
```

Include dependency graph for mgparm.c:



Functions

- VPUBLIC void MGparm_setCenterX (MGparm *thee, double x)

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

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

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

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

  Get center y-coordinate.
- VPUBLIC double MGparm_getCenterZ (MGparm *thee)

  Get center z-coordinate.
- VPUBLIC int MGparm_getNx (MGparm *thee)

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

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

  Get number of grid points in z direction.

- VPUBLIC double MGparm_getHx (MGparm *thee)

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

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

  Get grid spacing in z direction (Å)
- VPUBLIC MGparm * MGparm_ctor (MGparm_CalcType type)

  Construct MGparm object.
- VPUBLIC Vrc_Codes MGparm_ctor2 (MGparm *thee, MGparm_CalcType type)

  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.

9.27.1 Detailed Description

Class MGparm methods.

Author

Nathan Baker

Version

Id

Attention

Definition in file mgparm.c.

9.28 mgparm.c

```
00001
00057 #include "mgparm.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_MGPARM)
00062
00063 #endif /* if !defined(VINLINE_MGPARM) */
00064
00065 VPUBLIC void MGparm_setCenterX(MGparm *thee, double x) {
00066     VASSERT(thee != VNULL);
00067     thee->center[0] = x;
00068 }
00069 VPUBLIC void MGparm_setCenterY(MGparm *thee, double y) {
00070     VASSERT(thee != VNULL);
00071     thee->center[1] = y;
00072 }
```

```
00073 VPUBLIC void MGparm_setCenterZ(MGparm *thee, double z) {
00074     VASSERT(thee != VNULL);
00075     thee->center[2] = z;
00076 }
00077 VPUBLIC double MGparm_getCenterX(MGparm *thee) {
00078     VASSERT(thee != VNULL);
00079     return thee->center[0];
00080 }
00081 VPUBLIC double MGparm_getCenterY(MGparm *thee) {
00082     VASSERT(thee != VNULL);
00083     return thee->center[1];
00084 }
00085 VPUBLIC double MGparm_getCenterZ(MGparm *thee) {
00086     VASSERT(thee != VNULL);
00087     return thee->center[2];
00088 }
00089 VPUBLIC int MGparm_getNx(MGparm *thee) {
00090     VASSERT(thee != VNULL);
00091     return thee->dime[0];
00092 }
00093 VPUBLIC int MGparm_getNy(MGparm *thee) {
00094     VASSERT(thee != VNULL);
00095     return thee->dime[1];
00096 }
00097 VPUBLIC int MGparm_getNz(MGparm *thee) {
00098     VASSERT(thee != VNULL);
00099     return thee->dime[2];
00100 }
00101 VPUBLIC double MGparm_getHx(MGparm *thee) {
00102     VASSERT(thee != VNULL);
00103     return thee->grid[0];
00104 }
00105 VPUBLIC double MGparm_getHy(MGparm *thee) {
00106     VASSERT(thee != VNULL);
00107     return thee->grid[1];
00108 }
00109 VPUBLIC double MGparm_getHz(MGparm *thee) {
00110     VASSERT(thee != VNULL);
00111     return thee->grid[2];
00112 }
00113
00114 VPUBLIC MGparm* MGparm_ctor(MGparm_CalcType type) {
00115
00116     /* Set up the structure */
00117     MGparm *thee = VNULL;
00118     thee = (MGparm*)Vmem_malloc(VNULL, 1, sizeof(MGparm));
00119     VASSERT( thee != VNULL);
00120     VASSERT( MGparm_ctor2(thee, type) == VRC_SUCCESS );
00121
00122     return thee;
00123 }
00124
00125 VPUBLIC Vrc_Codes MGparm_ctor2(MGparm *thee, MGparm_CalcType type) {
00126
00127     int i;
00128
00129     if (thee == VNULL) return VRC_FAILURE;
00130
00131     for (i=0; i<3; i++) {
00132         thee->dime[i] = -1;
00133         thee->pdime[i] = 1;
00134     }
00135
00136     thee->parsed = 0;
00137     thee->type = type;
00138
00139     /* *** GENERIC PARAMETERS *** */
00140     thee->setdime = 0;
00141     thee->setchgm = 0;
00142
00143     /* *** TYPE 0 PARAMETERS *** */
00144     thee->nlev = VMGNLEV;
00145     thee->setnlev = 1;
00146     thee->etol = 1.0e-6;
00147     thee->setetol = 0;
00148     thee->setgrid = 0;
00149     thee->setglen = 0;
00150     thee->setgcent = 0;
00151
00152     /* *** TYPE 1 & 2 PARAMETERS *** */
00153     thee->setcglen = 0;
```

```
00154     thee->setfglen = 0;
00155     thee->setcgcent = 0;
00156     thee->setfgcent = 0;
00157 
00158     /* *** TYPE 2 PARAMETERS *** */
00159     thee->setpdime = 0;
00160     thee->setrank = 0;
00161     thee->setsize = 0;
00162     thee->setofrac = 0;
00163     for (i=0; i<6; i++) thee->partDisjOwnSide[i] = 0;
00164     thee->setasync = 0;
00165 
00166     /* *** Default parameters for TINKER *** */
00167     thee->chgs = VCM_CHARGE;
00168 
00169     thee->useAqua = 0;
00170     thee->setUseAqua = 0;
00171 
00172     return VRC_SUCCESS;
00173 }
00174 
00175 VPUBLIC void MGparm_dtor(MGparm **thee) {
00176     if ((*thee) != VNULL) {
00177         MGparm_dtor2(*thee);
00178         Vmem_free(VNULL, 1, sizeof(MGparm), (void **)thee);
00179         (*thee) = VNULL;
00180     }
00181 }
00182 
00183 VPUBLIC void MGparm_dtor2(MGparm *thee) { ; }
00184 
00185 VPUBLIC Vrc_Codes MGparm_check(MGparm *thee) {
00186 
00187     Vrc_Codes rc;
00188     int i, tdime[3], ti, tnlev[3], nlev;
00189 
00190     rc = VRC_SUCCESS;
00191 
00192     Vnm_print(0, "MGparm_check:  checking MGparm object of type %d.\n",
00193               thee->type);
00194 
00195     /* Check to see if we were even filled... */
00196     if (!thee->parsed) {
00197         Vnm_print(2, "MGparm_check:  not filled!\n");
00198         return VRC_FAILURE;
00199     }
00200 
00201     /* Check generic settings */
00202     if (!thee->setdime) {
00203         Vnm_print(2, "MGparm_check:  DIME not set!\n");
00204         rc = VRC_FAILURE;
00205     }
00206     if (!thee->setchgm) {
00207         Vnm_print(2, "MGparm_check: CHGM not set!\n");
00208         return VRC_FAILURE;
00209     }
00210 
00211 
00212     /* Check sequential manual & dummy settings */
00213     if ((thee->type == MCT_MANUAL) || (thee->type == MCT_DUMMY)) {
00214         if ((!thee->setgrid) && (!thee->setglen)) {
00215             Vnm_print(2, "MGparm_check:  Neither GRID nor GLEN set!\n");
00216             rc = VRC_FAILURE;
00217         }
00218         if ((thee->setgrid) && (thee->setglen)) {
00219             Vnm_print(2, "MGparm_check:  Both GRID and GLEN set!\n");
00220             rc = VRC_FAILURE;
00221         }
00222         if (!thee->setgcent) {
00223             Vnm_print(2, "MGparm_check:  GCENT not set!\n");
00224             rc = VRC_FAILURE;
00225         }
00226     }
00227 
00228     /* Check sequential and parallel automatic focusing settings */
00229     if ((thee->type == MCT_AUTO) || (thee->type == MCT_PARALLEL)) {
00230         if (!thee->setcglen) {
00231             Vnm_print(2, "MGparm_check:  CGLEN not set!\n");
00232             rc = VRC_FAILURE;
00233         }
00234         if (!thee->setfglen) {
```

```
00235             Vnm_print(2, "MGparm_check:  FGLEN not set!\n");
00236             rc = VRC_FAILURE;
00237         }
00238         if (!thee->setcgcent) {
00239             Vnm_print(2, "MGparm_check:  CGCENT not set!\n");
00240             rc = VRC_FAILURE;
00241         }
00242         if (!thee->setfgcent) {
00243             Vnm_print(2, "MGparm_check:  FGCENT not set!\n");
00244             rc = VRC_FAILURE;
00245         }
00246     }
00247
00248     /* Check parallel automatic focusing settings */
00249     if (thee->type == MCT_PARALLEL) {
00250         if (!thee->setpdime) {
00251             Vnm_print(2, "MGparm_check:  PDIME not set!\n");
00252             rc = VRC_FAILURE;
00253         }
00254         if (!thee->setrank) {
00255             Vnm_print(2, "MGparm_check:  PROC_RANK not set!\n");
00256             rc = VRC_FAILURE;
00257         }
00258         if (!thee->setsize) {
00259             Vnm_print(2, "MGparm_check:  PROC_SIZE not set!\n");
00260             rc = VRC_FAILURE;
00261         }
00262         if (!thee->setofrac) {
00263             Vnm_print(2, "MGparm_check:  OFRAC not set!\n");
00264             rc = VRC_FAILURE;
00265         }
00266     }
00267
00268     /* Perform a sanity check on nlev and dime, resetting values as necessary */
00269     if (rc == 1) {
00270     /* Calculate the actual number of grid points and nlev to satisfy the
00271      * formula:  n = c * 2^(l+1) + 1, where n is the number of grid points,
00272      * c is an integer, and l is the number of levels */
00273     if (thee->type != MCT_DUMMY) {
00274         for (i=0; i<3; i++) {
00275             /* See if the user picked a reasonable value, if not then fix it */
00276             ti = thee->dime[i] - 1;
00277             if (ti == VPOW(2, (thee->nlev+1))) {
00278                 tnlev[i] = thee->nlev;
00279                 tdime[i] = thee->dime[i];
00280             } else {
00281                 tdime[i] = thee->dime[i];
00282                 ti = tdime[i] - 1;
00283                 tnlev[i] = 0;
00284                 /* Find the maximum number of times this dimension can be
00285                  * divided by two */
00286                 while (VEVEN(ti)) {
00287                     (tnlev[i])++;
00288                     ti = (int)ceil(0.5*ti);
00289                 }
00290                 (tnlev[i])--;
00291                 /* We'd like to have at least VMGNLEV levels in the multigrid
00292                  * hierarchy.  This means that the dimension needs to be
00293                  * c*2^VMGNLEV + 1, where c is an integer. */
00294                 if ((tdime[i] > 65) && (tnlev[i] < VMGNLEV)) {
00295                     Vnm_print(2, "NOsh:  Bad dime[%d]  = %d (%d nlev)!\n",
00296                       i, tdime[i], tnlev[i]);
00297                     ti = (int)(tdime[i]/VPOW(2.,(VMGNLEV+1)));
00298                     if (ti < 1) ti = 1;
00299                     tdime[i] = ti*(int)(VPOW(2.,(VMGNLEV+1))) + 1;
00300                     tnlev[i] = 4;
00301                     Vnm_print(2, "NOsh:  Reset dime[%d] to %d and (nlev = %d).\n", i, tdime[i], VMGNLEV);
00302                 }
00303             }
00304         }
00305     } else { /* We are a dummy calculation, but we still need positive numbers of points */
00306         for (i=0; i<3; i++) {
00307             tnlev[i] = thee->nlev;
00308             tdime[i] = thee->dime[i];
00309                 if (thee->dime[i] <= 0) {
00310                     Vnm_print(2, "NOsh:  Resetting dime[%d] from %d to 3.\n", i, thee->dime[i]);
00311                     thee->dime[i] = 3;
00312             }
00313         }
00314     }
00315
```

```
00316     /* The actual number of levels we'll be using is the smallest number of
00317          * possible levels in any dimensions */
00318         nlev = VMIN2(tnlev[0], tnlev[1]);
00319         nlev = VMIN2(nlev, tnlev[2]);
00320         /* Set the number of levels and dimensions */
00321         Vnm_print(0, "NOsh:  nlev = %d, dime = (%d, %d, %d)\n", nlev, tdime[0],
00322           tdime[1], tdime[2]);
00323         thee->nlev = nlev;
00324         if (thee->nlev <= 0) {
00325             Vnm_print(2, "MGparm_check:  illegal nlev (%d); check your grid dimensions!\n", thee->nlev);
00326             rc = VRC_FAILURE;
00327         }
00328         if (thee->nlev < 2) {
00329             Vnm_print(2, "MGparm_check:  you're using a very small nlev (%d) and therefore\n",
      thee->nlev);
00330             Vnm_print(2, "MGparm_check:  will not get the optimal performance of the multigrid\n");
00331             Vnm_print(2, "MGparm_check:  algorithm.  Please check your grid dimensions.\n");
00332         }
00333         for (i=0; i<3; i++) thee->dime[i] = tdime[i];
00334     }
00335
00336     if (!thee->setUseAqua) thee->useAqua = 0;
00337
00338     return rc;
00339 }
00340
00341 VPUBLIC void MGparm_copy(MGparm *thee, MGparm *parm) {
00342
00343     int i;
00344
00345     VASSERT(thee != VNULL);
00346     VASSERT(parm != VNULL);
00347
00348
00349     thee->type = parm->type;
00350     thee->parsed = parm->parsed;
00351
00352     /* *** GENERIC PARAMETERS *** */
00353     for (i=0; i<3; i++) thee->dime[i] = parm->dime[i];
00354     thee->setdime = parm->setdime;
00355     thee->chgm = parm->chgm;
00356     thee->setchgm = parm->setchgm;
00357     thee->chgs = parm->chgs;
00358
00359     /* *** TYPE 0 PARMS *** */
00360     thee->nlev = parm->nlev;
00361     thee->setnlev = parm->setnlev;
00362     thee->etol = parm->etol;
00363     thee->setetol = parm->setetol;
00364     for (i=0; i<3; i++) thee->grid[i] = parm->grid[i];
00365     thee->setgrid = parm->setgrid;
00366     for (i=0; i<3; i++) thee->glen[i] = parm->glen[i];
00367     thee->setglen = parm->setglen;
00368     thee->cmeth = parm->cmeth;
00369     for (i=0; i<3; i++) thee->center[i] = parm->center[i];
00370     thee->setgcent = parm->setgcent;
00371     thee->centmol = parm->centmol;
00372
00373     /* *** TYPE 1 & 2 PARMS *** */
00374     for (i=0; i<3; i++) thee->cglen[i] = parm->cglen[i];
00375     thee->setcglen = parm->setcglen;
00376     for (i=0; i<3; i++) thee->fglen[i] = parm->fglen[i];
00377     thee->setfglen = parm->setfglen;
00378     thee->ccmeth = parm->ccmeth;
00379     for (i=0; i<3; i++) thee->ccenter[i] = parm->ccenter[i];
00380     thee->setcgcent = parm->setcgcent;
00381     thee->ccentmol = parm->ccentmol;
00382     thee->fcmeth = parm->fcmeth;
00383     for (i=0; i<3; i++) thee->fcenter[i] = parm->fcenter[i];
00384     thee->setfgcent = parm->setfgcent;
00385     thee->fcentmol = parm->fcentmol;
00386
00387     /* *** TYPE 2 PARMS *** */
00388     for (i=0; i<3; i++)
00389       thee->partDisjCenter[i] = parm->partDisjCenter[i];
00390     for (i=0; i<3; i++)
00391       thee->partDisjLength[i] = parm->partDisjLength[i];
00392     for (i=0; i<6; i++)
00393       thee->partDisjOwnSide[i] = parm->partDisjOwnSide[i];
00394     for (i=0; i<3; i++) thee->pdime[i] = parm->pdime[i];
00395     thee->setpdime = parm->setpdime;
```

```
00396     thee->proc_rank = parm->proc_rank;
00397     thee->setrank = parm->setrank;
00398     thee->proc_size = parm->proc_size;
00399     thee->setsize = parm->setsize;
00400     thee->ofrac = parm->ofrac;
00401     thee->setofrac = parm->setofrac;
00402     thee->setasync = parm->setasync;
00403     thee->async = parm->async;
00404 
00405     thee->nonlintype = parm->nonlintype;
00406     thee->setnonlintype = parm->setnonlintype;
00407 
00408     thee->method = parm->method;
00409     thee->method = parm->method;
00410 
00411     thee->useAqua = parm->useAqua;
00412     thee->setUseAqua = parm->setUseAqua;
00413 }
00414 
00415 VPRIVATE Vrc_Codes MGparm_parseDIME(MGparm *thee, Vio *sock) {
00416 
00417     char tok[VMAX_BUFSIZE];
00418     int ti;
00419 
00420     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00421     if (sscanf(tok, "%d", &ti) == 0){
00422         Vnm_print(2, "parseMG:  Read non-integer (%s) while parsing DIME \
00423 keyword!\n", tok);
00424         return VRC_WARNING;
00425     } else thee->dime[0] = ti;
00426     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00427     if (sscanf(tok, "%d", &ti) == 0) {
00428         Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing DIME \
00429 keyword!\n", tok);
00430         return VRC_WARNING;
00431     } else thee->dime[1] = ti;
00432     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00433     if (sscanf(tok, "%d", &ti) == 0) {
00434         Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing DIME \
00435 keyword!\n", tok);
00436         return VRC_WARNING;
00437     } else thee->dime[2] = ti;
00438     thee->setdime = 1;
00439     return VRC_SUCCESS;
00440 
00441     VERROR1:
00442         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00443         return VRC_WARNING;
00444 }
00445 
00446 VPRIVATE Vrc_Codes MGparm_parseCHGM(MGparm *thee, Vio *sock) {
00447 
00448     char tok[VMAX_BUFSIZE];
00449     Vchrg_Meth ti;
00450 
00451     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00452     if (sscanf(tok, "%d", (int*)(&ti)) == 1) {
00453         thee->chgm = ti;
00454         thee->setchgm = 1;
00455         Vnm_print(2, "NOsh:  Warning -- parsed deprecated statment \"chgm %d\".\n", ti);
00456         Vnm_print(2, "NOsh:  Please use \"chgm ");
00457         switch (thee->chgm) {
00458             case VCM_TRIL:
00459                 Vnm_print(2, "spl0");
00460                 break;
00461             case VCM_BSPL2:
00462                 Vnm_print(2, "spl2");
00463                 break;
00464             case VCM_BSPL4:
00465                 Vnm_print(2, "spl4");
00466                 break;
00467             default:
00468                 Vnm_print(2, "UNKNOWN");
00469                 break;
00470         }
00471         Vnm_print(2, "\" instead!\n");
00472         return VRC_SUCCESS;
00473     } else if (Vstring_strcasecmp(tok, "spl0") == 0) {
00474         thee->chgm = VCM_TRIL;
00475         thee->setchgm = 1;
00476         return VRC_SUCCESS;
```

```
00477     } else if (Vstring_strcasecmp(tok, "spl2") == 0) {
00478         thee->chgm = VCM_BSPL2;
00479         thee->setchgm = 1;
00480         return VRC_SUCCESS;
00481     } else if (Vstring_strcasecmp(tok, "spl4") == 0) {
00482         thee->chgm = VCM_BSPL4;
00483         thee->setchgm = 1;
00484         return VRC_SUCCESS;
00485     } else {
00486         Vnm_print(2, "NOsh:  Unrecognized parameter (%s) when parsing \
00487 chgm!\n", tok);
00488         return VRC_WARNING;
00489     }
00490     return VRC_WARNING;
00491
00492     VERROR1:
00493         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00494         return VRC_WARNING;
00495 }
00496
00497 VPRIVATE Vrc_Codes MGparm_parseNLEV(MGparm *thee, Vio *sock) {
00498
00499     char tok[VMAX_BUFSIZE];
00500     int ti;
00501
00502     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00503     if (sscanf(tok, "%d", &ti) == 0) {
00504         Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing NLEV \
00505 keyword!\n", tok);
00506         return VRC_WARNING;
00507     } else thee->nlev = ti;
00508     thee->setnlev = 1;
00509     return VRC_SUCCESS;
00510
00511     VERROR1:
00512         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00513         return VRC_WARNING;
00514 }
00515
00516 VPRIVATE Vrc_Codes MGparm_parseETOL(MGparm *thee, Vio *sock) {
00517
00518     char tok[VMAX_BUFSIZE];
00519     double tf;
00520
00521     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00522     if (sscanf(tok, "%lf", &tf) == 0) {
00523         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing etol \
00524 keyword!\n", tok);
00525         return VRC_WARNING;
00526     } else if (tf <= 0.0) {
00527         Vnm_print(2, "parseMG:  etol must be greater than 0!\n");
00528         return VRC_WARNING;
00529     } else thee->etol = tf;
00530     thee->setetol = 1;
00531     return VRC_SUCCESS;
00532
00533     VERROR1:
00534         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00535         return VRC_WARNING;
00536 }
00537
00538
00539 VPRIVATE Vrc_Codes MGparm_parseGRID(MGparm *thee, Vio *sock) {
00540
00541     char tok[VMAX_BUFSIZE];
00542     double tf;
00543
00544     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00545     if (sscanf(tok, "%lf", &tf) == 0) {
00546         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GRID \
00547 keyword!\n", tok);
00548         return VRC_WARNING;
00549     } else thee->grid[0] = tf;
00550     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00551     if (sscanf(tok, "%lf", &tf) == 0) {
00552         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GRID \
00553 keyword!\n", tok);
00554         return VRC_WARNING;
00555     } else thee->grid[1] = tf;
00556     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00557     if (sscanf(tok, "%lf", &tf) == 0) {
```

```
00558         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GRID \
00559 keyword!\n", tok);
00560         return VRC_WARNING;
00561     } else thee->grid[2] = tf;
00562     thee->setgrid = 1;
00563     return VRC_SUCCESS;
00564 
00565     VERROR1:
00566         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00567         return VRC_WARNING;
00568 }
00569 
00570 VPRIVATE Vrc_Codes MGparm_parseGLEN(MGparm *thee, Vio *sock) {
00571 
00572     char tok[VMAX_BUFSIZE];
00573     double tf;
00574 
00575     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00576     if (sscanf(tok, "%lf", &tf) == 0) {
00577         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GLEN \
00578 keyword!\n", tok);
00579         return VRC_WARNING;
00580     } else thee->glen[0] = tf;
00581     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00582     if (sscanf(tok, "%lf", &tf) == 0) {
00583         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GLEN \
00584 keyword!\n", tok);
00585         return VRC_WARNING;
00586     } else thee->glen[1] = tf;
00587     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00588     if (sscanf(tok, "%lf", &tf) == 0) {
00589         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing GLEN \
00590 keyword!\n", tok);
00591         return VRC_WARNING;
00592     } else thee->glen[2] = tf;
00593     thee->setglen = 1;
00594     return VRC_SUCCESS;
00595 
00596     VERROR1:
00597         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00598         return VRC_WARNING;
00599 }
00600 
00601 VPRIVATE Vrc_Codes MGparm_parseGAMMA(MGparm *thee, Vio *sock) {
00602 
00603     char tok[VMAX_BUFSIZE];
00604 
00605     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00606     Vnm_print(2, "parseMG:  GAMMA keyword deprecated!\n");
00607     Vnm_print(2, "parseMG:  If you are using PyMOL or VMD and still seeing this message,\n");
00608     Vnm_print(2, "parseMG:  please contact the developers of those programs regarding this message.\n");
00609     return VRC_SUCCESS;
00610 
00611 VERROR1:
00612     Vnm_print(2, "parseMG:  ran out of tokens!\n");
00613     return VRC_WARNING;
00614 }
00615 
00616 VPRIVATE Vrc_Codes MGparm_parseGCENT(MGparm *thee, Vio *sock) {
00617 
00618     char tok[VMAX_BUFSIZE];
00619     double tf;
00620     int ti;
00621 
00622     /* If the next token isn't a float, it probably means we want to
00623      * center on a molecule */
00624     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00625     if (sscanf(tok, "%lf", &tf) == 0) {
00626         if (Vstring_strcasecmp(tok, "mol") == 0) {
00627             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00628             if (sscanf(tok, "%d", &ti) == 0) {
00629                 Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
00630 GCENT MOL keyword!\n", tok);
00631                 return VRC_WARNING;
00632             } else {
00633                 thee->cmeth = MCM_MOLECULE;
00634                 /* Subtract 1 here to convert user numbering (1, 2, 3, ...) into
00635                 array index */
00636                 thee->centmol = ti - 1;
00637             }
00638         } else {
```

```
00639             Vnm_print(2, "NOsh:  Unexpected keyword (%s) while parsing \
00640 GCENT!\n", tok);
00641             return VRC_WARNING;
00642         }
00643     } else {
00644         thee->center[0] = tf;
00645         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00646         if (sscanf(tok, "%lf", &tf) == 0) {
00647             Vnm_print(2, "NOsh:  Read non-float (%s) while parsing \
00648 GCENT keyword!\n", tok);
00649             return VRC_WARNING;
00650         }
00651         thee->center[1] = tf;
00652         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00653         if (sscanf(tok, "%lf", &tf) == 0) {
00654             Vnm_print(2, "NOsh:  Read non-float (%s) while parsing \
00655 GCENT keyword!\n", tok);
00656             return VRC_WARNING;
00657         }
00658         thee->center[2] = tf;
00659     }
00660     thee->setgcent = 1;
00661     return VRC_SUCCESS;
00662
00663     VERROR1:
00664         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00665         return VRC_WARNING;
00666 }
00667
00668 VPRIVATE Vrc_Codes MGparm_parseCGLEN(MGparm *thee, Vio *sock) {
00669
00670     char tok[VMAX_BUFSIZE];
00671     double tf;
00672
00673     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00674     if (sscanf(tok, "%lf", &tf) == 0) {
00675         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing CGLEN \
00676 keyword!\n", tok);
00677         return VRC_WARNING;
00678     } else thee->cglen[0] = tf;
00679     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00680     if (sscanf(tok, "%lf", &tf) == 0) {
00681         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing CGLEN \
00682 keyword!\n", tok);
00683         return VRC_WARNING;
00684     } else thee->cglen[1] = tf;
00685     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00686     if (sscanf(tok, "%lf", &tf) == 0) {
00687         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing CGLEN \
00688 keyword!\n", tok);
00689         return VRC_WARNING;
00690     } else thee->cglen[2] = tf;
00691     thee->setcglen = 1;
00692     return VRC_SUCCESS;
00693
00694     VERROR1:
00695         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00696         return VRC_WARNING;
00697 }
00698
00699 VPRIVATE Vrc_Codes MGparm_parseFGLEN(MGparm *thee, Vio *sock) {
00700
00701     char tok[VMAX_BUFSIZE];
00702     double tf;
00703
00704     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00705     if (sscanf(tok, "%lf", &tf) == 0) {
00706         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing FGLEN \
00707 keyword!\n", tok);
00708         return VRC_WARNING;
00709     } else thee->fglen[0] = tf;
00710     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00711     if (sscanf(tok, "%lf", &tf) == 0) {
00712         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing FGLEN \
00713 keyword!\n", tok);
00714         return VRC_WARNING;
00715     } else thee->fglen[1] = tf;
00716     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00717     if (sscanf(tok, "%lf", &tf) == 0) {
00718         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing FGLEN \
00719 keyword!\n", tok);
```

```
00720         return VRC_WARNING;
00721     } else thee->fglen[2] = tf;
00722     thee->setfglen = 1;
00723     return VRC_SUCCESS;
00724
00725     VERROR1:
00726         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00727         return VRC_WARNING;
00728 }
00729
00730 VPRIVATE Vrc_Codes MGparm_parseCGCENT(MGparm *thee, Vio *sock) {
00731
00732     char tok[VMAX_BUFSIZE];
00733     double tf;
00734     int ti;
00735
00736     /* If the next token isn't a float, it probably means we want to
00737      * center on a molecule */
00738     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00739     if (sscanf(tok, "%lf", &tf) == 0) {
00740         if (Vstring_strcasecmp(tok, "mol") == 0) {
00741             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00742             if (sscanf(tok, "%d", &ti) == 0) {
00743                 Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
00744 CGCENT MOL keyword!\n", tok);
00745                 return VRC_WARNING;
00746             } else {
00747                 thee->ccmeth = MCM_MOLECULE;
00748                 /* Subtract 1 here to convert user numbering (1, 2, 3, ...) into
00749                 array index */
00750                 thee->ccentmol = ti - 1;
00751             }
00752         } else {
00753             Vnm_print(2, "NOsh:  Unexpected keyword (%s) while parsing \
00754 CGCENT!\n", tok);
00755             return VRC_WARNING;
00756             }
00757     } else {
00758         thee->ccenter[0] = tf;
00759         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00760         if (sscanf(tok, "%lf", &tf) == 0) {
00761             Vnm_print(2, "NOsh:  Read non-float (%s) while parsing \
00762 CGCENT keyword!\n", tok);
00763             return VRC_WARNING;
00764         }
00765         thee->ccenter[1] = tf;
00766         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00767         if (sscanf(tok, "%lf", &tf) == 0) {
00768             Vnm_print(2, "NOsh:  Read non-float (%s) while parsing \
00769 CGCENT keyword!\n", tok);
00770             return VRC_WARNING;
00771         }
00772         thee->ccenter[2] = tf;
00773     }
00774     thee->setcgcent = 1;
00775     return VRC_SUCCESS;
00776
00777     VERROR1:
00778         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00779         return VRC_WARNING;
00780 }
00781
00782 VPRIVATE Vrc_Codes MGparm_parseFGCENT(MGparm *thee, Vio *sock) {
00783
00784     char tok[VMAX_BUFSIZE];
00785     double tf;
00786     int ti;
00787
00788     /* If the next token isn't a float, it probably means we want to
00789      * center on a molecule */
00790     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00791     if (sscanf(tok, "%lf", &tf) == 0) {
00792         if (Vstring_strcasecmp(tok, "mol") == 0) {
00793             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00794             if (sscanf(tok, "%d", &ti) == 0) {
00795                  Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
00796 FGCENT MOL keyword!\n", tok);
00797                 return VRC_WARNING;
00798             } else {
00799                 thee->fcmeth = MCM_MOLECULE;
00800                 /* Subtract 1 here to convert user numbering (1, 2, 3, ...) into
```

```
00801                 array index */
00802                 thee->fcentmol = ti - 1;
00803             }
00804         } else {
00805             Vnm_print(2, "NOsh:  Unexpected keyword (%s) while parsing \
00806 FGCENT!\n", tok);
00807             return VRC_WARNING;
00808         }
00809     } else {
00810         thee->fcenter[0] = tf;
00811         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00812         if (sscanf(tok, "%lf", &tf) == 0) {
00813             Vnm_print(2, "NOsh:  Read non-float (%s) while parsing \
00814 FGCENT keyword!\n", tok);
00815             return VRC_WARNING;
00816         }
00817         thee->fcenter[1] = tf;
00818         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00819         if (sscanf(tok, "%lf", &tf) == 0) {
00820             Vnm_print(2, "NOsh:  Read non-float (%s) while parsing \
00821 FGCENT keyword!\n", tok);
00822             return VRC_WARNING;
00823         }
00824         thee->fcenter[2] = tf;
00825     }
00826     thee->setfgcent = 1;
00827     return VRC_SUCCESS;
00828
00829     VERROR1:
00830         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00831         return VRC_WARNING;
00832 }
00833
00834 VPRIVATE Vrc_Codes MGparm_parsePDIME(MGparm *thee, Vio *sock) {
00835
00836     char tok[VMAX_BUFSIZE];
00837     int ti;
00838
00839     /* Read the number of grid points */
00840     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00841     if (sscanf(tok, "%d", &ti) == 0) {
00842         Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing PDIME \
00843 keyword!\n", tok);
00844         return VRC_WARNING;
00845     } else {
00846         thee->pdime[0] = ti;
00847     }
00848     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00849     if (sscanf(tok, "%d", &ti) == 0) {
00850         Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing PDIME \
00851 keyword!\n", tok);
00852         return VRC_WARNING;
00853     } else {
00854         thee->pdime[1] = ti;
00855     }
00856     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00857     if (sscanf(tok, "%d", &ti) == 0) {
00858         Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing PDIME \
00859 keyword!\n", tok);
00860         return VRC_WARNING;
00861     } else {
00862         thee->pdime[2] = ti;
00863     }
00864     thee->setpdime = 1;
00865     return VRC_SUCCESS;
00866
00867     VERROR1:
00868         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00869         return VRC_WARNING;
00870 }
00871
00872 VPRIVATE Vrc_Codes MGparm_parseOFRAC(MGparm *thee, Vio *sock) {
00873
00874     char tok[VMAX_BUFSIZE];
00875     double tf;
00876
00877     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00878     if (sscanf(tok, "%lf", &tf) == 0) {
00879         Vnm_print(2, "NOsh:  Read non-int (%s) while parsing OFRAC \
00880 keyword!\n", tok);
00881         return VRC_WARNING;
```

```
00882     }
00883     thee->ofrac = tf;
00884     thee->setofrac = 1;
00885     return VRC_SUCCESS;
00886 
00887     VERROR1:
00888         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00889         return VRC_WARNING;
00890 }
00891 
00892 VPRIVATE Vrc_Codes MGparm_parseASYNC(MGparm *thee, Vio *sock) {
00893 
00894     char tok[VMAX_BUFSIZE];
00895     int ti;
00896 
00897     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00898     if (sscanf(tok, "%i", &ti) == 0) {
00899         Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing ASYNC \
00900 keyword!\n", tok);
00901         return VRC_WARNING;
00902     }
00903     thee->async = ti;
00904     thee->setasync = 1;
00905     return VRC_SUCCESS;
00906 
00907     VERROR1:
00908         Vnm_print(2, "parseMG:  ran out of tokens!\n");
00909         return VRC_WARNING;
00910 }
00911 
00912 VPRIVATE Vrc_Codes MGparm_parseUSEAQUA(MGparm *thee, Vio *sock) {
00913     Vnm_print(0, "NOsh: parsed useaqua\n");
00914     thee->useAqua = 1;
00915     thee->setUseAqua = 1;
00916     return VRC_SUCCESS;
00917 }
00918 
00919 VPUBLIC Vrc_Codes MGparm_parseToken(MGparm *thee, char tok[VMAX_BUFSIZE],
00920   Vio *sock) {
00921 
00922     if (thee == VNULL) {
00923         Vnm_print(2, "parseMG:  got NULL thee!\n");
00924         return VRC_WARNING;
00925     }
00926     if (sock == VNULL) {
00927         Vnm_print(2, "parseMG:  got NULL socket!\n");
00928         return VRC_WARNING;
00929     }
00930 
00931     Vnm_print(0, "MGparm_parseToken:  trying %s...\n", tok);
00932 
00933 
00934     if (Vstring_strcasecmp(tok, "dime") == 0) {
00935         return MGparm_parseDIME(thee, sock);
00936     } else if (Vstring_strcasecmp(tok, "chgm") == 0) {
00937         return MGparm_parseCHGM(thee, sock);
00938     } else if (Vstring_strcasecmp(tok, "nlev") == 0) {
00939         Vnm_print(2, "Warning: The 'nlev' keyword is now deprecated!\n");
00940         return MGparm_parseNLEV(thee, sock);
00941     } else if (Vstring_strcasecmp(tok, "etol") == 0) {
00942         return MGparm_parseETOL(thee, sock);
00943     } else if (Vstring_strcasecmp(tok, "grid") == 0) {
00944         return MGparm_parseGRID(thee, sock);
00945     } else if (Vstring_strcasecmp(tok, "glen") == 0) {
00946         return MGparm_parseGLEN(thee, sock);
00947     } else if (Vstring_strcasecmp(tok, "gcent") == 0) {
00948         return MGparm_parseGCENT(thee, sock);
00949     } else if (Vstring_strcasecmp(tok, "cglen") == 0) {
00950         return MGparm_parseCGLEN(thee, sock);
00951     } else if (Vstring_strcasecmp(tok, "fglen") == 0) {
00952         return MGparm_parseFGLEN(thee, sock);
00953     } else if (Vstring_strcasecmp(tok, "cgcent") == 0) {
00954         return MGparm_parseCGCENT(thee, sock);
00955     } else if (Vstring_strcasecmp(tok, "fgcent") == 0) {
00956         return MGparm_parseFGCENT(thee, sock);
00957     } else if (Vstring_strcasecmp(tok, "pdime") == 0) {
00958         return MGparm_parsePDIME(thee, sock);
00959     } else if (Vstring_strcasecmp(tok, "ofrac") == 0) {
00960         return MGparm_parseOFRAC(thee, sock);
00961     } else if (Vstring_strcasecmp(tok, "async") == 0) {
00962         return MGparm_parseASYNC(thee, sock);
```

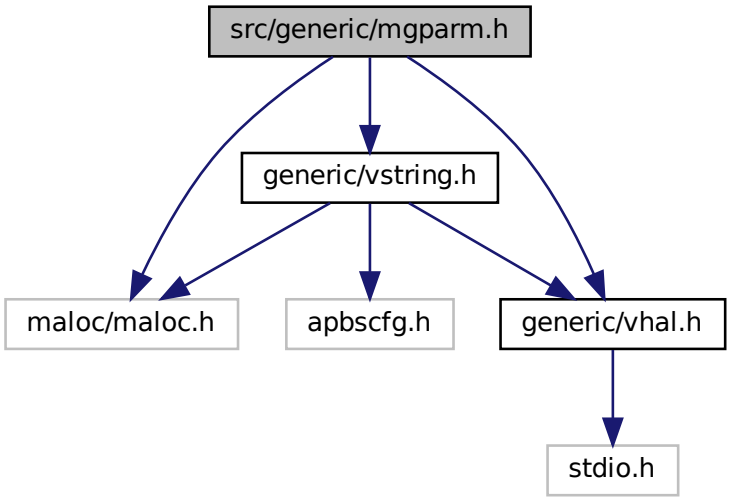
```
00963     } else if (Vstring_strcasecmp(tok, "gamma") == 0) {
00964         return MGparm_parseGAMMA(thee, sock);
00965     } else if (Vstring_strcasecmp(tok, "useaqua") == 0) {
00966         return MGparm_parseUSEAQUA(thee, sock);
00967     } else {
00968         Vnm_print(2, "parseMG:  Unrecognized keyword (%s)!\n", tok);
00969         return VRC_WARNING;
00970     }
00971
00972     return VRC_FAILURE;
00973
00974 }
```

9.29 src/generic/mgparm.h File Reference

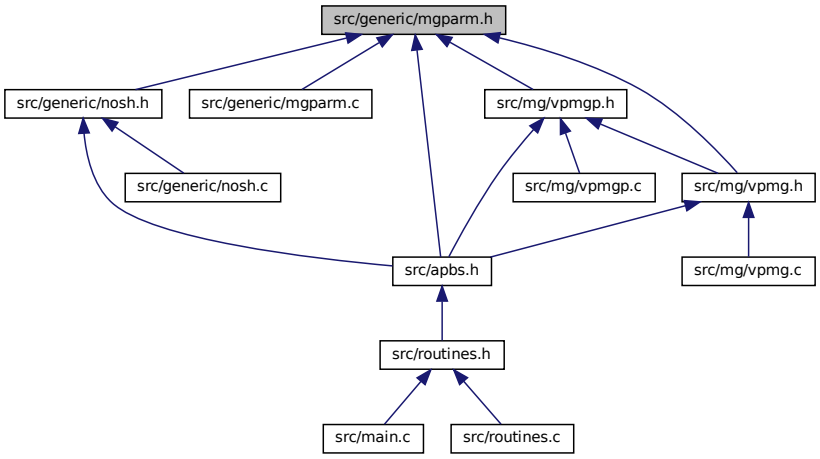
Contains declarations for class MGparm.

```
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/vstring.h"
```

Include dependency graph for mgparm.h:



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



Data Structures

- struct sMGparm

  Parameter structure for MG-specific variables from input files.

Typedefs

- typedef enum eMGparm_CalcType MGparm_CalcType

  Declare MGparm_CalcType type.
- typedef enum eMGparm_CentMeth MGparm_CentMeth

  Declare MGparm_CentMeth type.
- typedef struct sMGparm MGparm

  Declaration of the MGparm class as the MGparm structure.

Enumerations

- enum eMGparm_CalcType {
  MCT_MANUAL =0 , MCT_AUTO =1 , MCT_PARALLEL =2 , MCT_DUMMY =3 ,
  MCT_NONE =4 }

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

  Centering method.

Functions

- VEXTERNC int MGparm_getNx (MGparm *thee)

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

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

Get number of grid points in z direction.

- VEXTERNC double MGparm_getHx (MGparm *thee)

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

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

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

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

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

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

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

  Get center y-coordinate.
- VEXTERNC double MGparm_getCenterZ (MGparm *thee)

  Get center z-coordinate.
- VEXTERNC MGparm * MGparm_ctor (MGparm_CalcType type)

  Construct MGparm object.
- VEXTERNC Vrc_Codes MGparm_ctor2 (MGparm *thee, MGparm_CalcType type)

  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.

9.29.1 Detailed Description

Contains declarations for class MGparm.

Version

Id

Author

Nathan A. Baker

Attention

Definition in file mgparm.h.

9.30 mgparm.h

```
00001
00064 #ifndef _MGPARM_H_
00065 #define _MGPARM_H_
00066
00067 /* Generic header files */
00068 #include "maloc/maloc.h"
00069
00070 #include "generic/vhal.h"
00071 #include "generic/vstring.h"
00072
00077 enum eMGparm_CalcType {
00078     MCT_MANUAL=0,
00079     MCT_AUTO=1,
00080     MCT_PARALLEL=2,
00081     MCT_DUMMY=3,
00082     MCT_NONE=4
00083 };
```

```
00084
00089 typedef enum eMGparm_CalcType MGparm_CalcType;
00090
00095 enum eMGparm_CentMeth {
00096     MCM_POINT=0,
00097     MCM_MOLECULE=1,
00098     MCM_FOCUS=2
00099 };
00100
00105 typedef enum eMGparm_CentMeth MGparm_CentMeth;
00114 struct sMGparm {
00115
00116     MGparm_CalcType type;
00117     int parsed;
00119     /* *** GENERIC PARAMETERS *** */
00120     int dime[3];
00121     int setdime;
00122     Vchrg_Meth chgm;
00123     int setchgm;
00124     Vchrg_Src  chgs;
00127     /* *** TYPE 0 PARAMETERS (SEQUENTIAL MANUAL) *** */
00128     int nlev;
00130     int setnlev;
00131     double etol;
00132     int setetol;
00133     double grid[3];
00134     int setgrid;
00135     double glen[3];
00136     int setglen;
00137     MGparm_CentMeth cmeth;
00138     double center[3];
00146     int centmol;
00149     int setgcent;
00151     /* ******** TYPE 1 & 2 PARAMETERS (SEQUENTIAL & PARALLEL AUTO-FOCUS) *** */
00152     double cglen[3];
00153     int setcglen;
00154     double fglen[3];
00155     int setfglen;
00156     MGparm_CentMeth ccmeth;
00157     double ccenter[3];
00158     int ccentmol;
00161     int setcgcent;
00162     MGparm_CentMeth fcmeth;
00163     double fcenter[3];
00164     int fcentmol;
00167     int setfgcent;
00170     /* ********* TYPE 2 PARAMETERS (PARALLEL AUTO-FOCUS) ******** */
00171     double partDisjCenter[3];
00173     double partDisjLength[3];
00175     int partDisjOwnSide[6];
00178     int pdime[3];
00179     int setpdime;
00180     int proc_rank;
00181     int setrank;
00182     int proc_size;
00183     int setsize;
00184     double ofrac;
00185     int setofrac;
00186     int async;
00187     int setasync;
00189     int nonlintype;
00190     int setnonlintype;
00192     int method;
00193     int setmethod;
00195     int useAqua;
00196     int setUseAqua;
00197 };
00198
00203 typedef struct sMGparm MGparm;
00204
00211 VEXTERNC int MGparm_getNx(MGparm *thee);
00212
00219 VEXTERNC int MGparm_getNy(MGparm *thee);
00220
00227 VEXTERNC int MGparm_getNz(MGparm *thee);
00228
00235 VEXTERNC double MGparm_getHx(MGparm *thee);
00236
00243 VEXTERNC double MGparm_getHy(MGparm *thee);
00244
00251 VEXTERNC double MGparm_getHz(MGparm *thee);
```

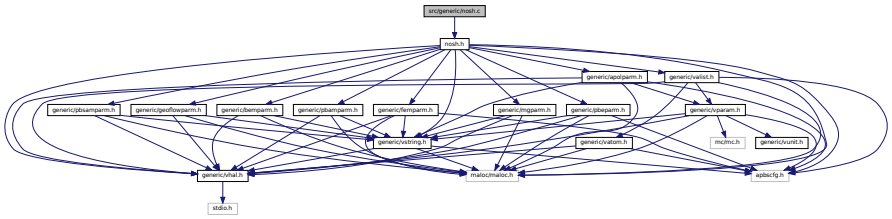
```
00252
00259 VEXTERNC void MGparm_setCenterX(MGparm *thee, double x);
00260
00267 VEXTERNC void MGparm_setCenterY(MGparm *thee, double y);
00268
00275 VEXTERNC void MGparm_setCenterZ(MGparm *thee, double z);
00276
00283 VEXTERNC double MGparm_getCenterX(MGparm *thee);
00284
00291 VEXTERNC double MGparm_getCenterY(MGparm *thee);
00292
00299 VEXTERNC double MGparm_getCenterZ(MGparm *thee);
00300
00307 VEXTERNC MGparm*  MGparm_ctor(MGparm_CalcType type);
00308
00316 VEXTERNC Vrc_Codes      MGparm_ctor2(MGparm *thee, MGparm_CalcType type);
00317
00323 VEXTERNC void     MGparm_dtor(MGparm **thee);
00324
00330 VEXTERNC void     MGparm_dtor2(MGparm *thee);
00331
00338 VEXTERNC Vrc_Codes      MGparm_check(MGparm *thee);
00339
00346 VEXTERNC void     MGparm_copy(MGparm *thee, MGparm *parm);
00347
00357 VEXTERNC Vrc_Codes      MGparm_parseToken(MGparm *thee, char tok[VMAX_BUFSIZE],
00358                     Vio *sock);
00359
00360 #endif
00361
```

9.31 src/generic/nosh.c File Reference

Class NOsh methods.

`#include "nosh.h"`

Include dependency graph for nosh.c:



Functions

- VPRIVATE int **NOsh_parseREAD** (NOsh *thee, Vio *sock)
- VPRIVATE int **NOsh_parsePRINT** (NOsh *thee, Vio *sock)
- VPRIVATE int **NOsh_parseELEC** (NOsh *thee, Vio *sock)
- VPRIVATE int **NOsh_parseAPOLAR** (NOsh *thee, Vio *sock)
- VEXTERNC int **NOsh_parseFEM** (NOsh *thee, Vio *sock, NOsh_calc *elec)
- VEXTERNC int **NOsh_parseMG** (NOsh *thee, Vio *sock, NOsh_calc *elec)
- VEXTERNC int **NOsh_parseBEM** (NOsh *thee, Vio *sock, NOsh_calc *elec)
- VEXTERNC int **NOsh_parseGEOFLOW** (NOsh *thee, Vio *sock, NOsh_calc *elec)
- VEXTERNC int **NOsh_parsePBAM** (NOsh *thee, Vio *sock, NOsh_calc *elec)
- VEXTERNC int **NOsh_parsePBSAM** (NOsh *thee, Vio *sock, NOsh_calc *elec)
- VEXTERNC int **NOsh_parseAPOL** (NOsh *thee, Vio *sock, NOsh_calc *elec)
- VPRIVATE int **NOsh_setupCalcMG** (NOsh *thee, NOsh_calc *elec)
- VPRIVATE int **NOsh_setupCalcMGAUTO** (NOsh *thee, NOsh_calc *elec)

- VPRIVATE int **NOsh_setupCalcMGMANUAL** (NOsh *thee, NOsh_calc *elec)
- VPRIVATE int **NOsh_setupCalcMGPARA** (NOsh *thee, NOsh_calc *elec)
- VPRIVATE int **NOsh_setupCalcFEM** (NOsh *thee, NOsh_calc *elec)
- VPRIVATE int **NOsh_setupCalcFEMANUAL** (NOsh *thee, NOsh_calc *elec)
- VPRIVATE int **NOsh_setupCalcBEM** (NOsh *thee, NOsh_calc *elec)
- VPRIVATE int **NOsh_setupCalcGEOFLOW** (NOsh *thee, NOsh_calc *elec)
- VPRIVATE int **NOsh_setupCalcPBAM** (NOsh *thee, NOsh_calc *elec)
- VPRIVATE int **NOsh_setupCalcPBSAM** (NOsh *thee, NOsh_calc *elec)
- VPRIVATE int **NOsh_setupCalcBEMMANUAL** (NOsh *thee, NOsh_calc *elec)
- VPRIVATE int **NOsh_setupCalcGEOFLOWMANUAL** (NOsh *thee, NOsh_calc *elec)
- VPRIVATE int **NOsh_setupCalcPBAMAUTO** (NOsh *thee, NOsh_calc *elec)
- VPRIVATE int **NOsh_setupCalcPBSAMAUTO** (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.

9.31.1 Detailed Description

Class NOsh methods.

Author

Nathan Baker

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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) 2002-2020, 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.

9.32 nosh.c

```
00001
00057 #include "nosh.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061
00062 VPRIVATE int NOsh_parseREAD(
00063                             NOsh *thee,
00064                             Vio *sock);
00065
00066 VPRIVATE int NOsh_parsePRINT(
00067                              NOsh *thee,
00068                              Vio *sock);
00069
00070 VPRIVATE int NOsh_parseELEC(
00071                             NOsh *thee,
00072                             Vio *sock
```

```
00073                             );
00074
00075 VPRIVATE int NOsh_parseAPOLAR(
00076                              NOsh *thee,
00077                              Vio *sock
00078                              );
00079
00080 VEXTERNC int NOsh_parseFEM(
00081                             NOsh *thee,
00082                             Vio *sock,
00083                             NOsh_calc *elec
00084                             );
00085
00086 VEXTERNC int NOsh_parseMG(
00087                            NOsh *thee,
00088                            Vio *sock,
00089                            NOsh_calc *elec
00090                            );
00091
00092 VEXTERNC int NOsh_parseBEM(
00093                            NOsh *thee,
00094                            Vio *sock,
00095                            NOsh_calc *elec
00096                            );
00097
00098 VEXTERNC int NOsh_parseGEOFLOW(
00099                            NOsh *thee,
00100                            Vio *sock,
00101                            NOsh_calc *elec
00102                            );
00103
00104 VEXTERNC int NOsh_parsePBAM(
00105                             NOsh *thee,
00106                             Vio *sock,
00107                             NOsh_calc *elec
00108                             );
00109
00110 VEXTERNC int NOsh_parsePBSAM(
00111                               NOsh *thee,
00112                               Vio *sock,
00113                               NOsh_calc *elec
00114                               );
00115
00116 VEXTERNC int NOsh_parseAPOL(
00117                             NOsh *thee,
00118                             Vio *sock,
00119                             NOsh_calc *elec
00120                             );
00121
00122 VPRIVATE int NOsh_setupCalcMG(
00123                                NOsh *thee,
00124                                NOsh_calc *elec
00125                                );
00126
00127
00128 VPRIVATE int NOsh_setupCalcMGAUTO(
00129                                   NOsh *thee,
00130                                   NOsh_calc *elec
00131                                   );
00132
00133 VPRIVATE int NOsh_setupCalcMGMANUAL(
00134                                     NOsh *thee,
00135                                     NOsh_calc *elec
00136                                     );
00137
00138 VPRIVATE int NOsh_setupCalcMGPARA(
00139                                   NOsh *thee,
00140                                   NOsh_calc *elec
00141                                   );
00142
00143 VPRIVATE int NOsh_setupCalcFEM(
00144                                NOsh *thee,
00145                                NOsh_calc *elec
00146                                );
00147
00148 VPRIVATE int NOsh_setupCalcFEMANUAL(
00149                                NOsh *thee,
00150                                NOsh_calc *elec
00151                                );
00152
00153 VPRIVATE int NOsh_setupCalcBEM(
```

```
00154                               NOsh *thee,
00155                               NOsh_calc *elec
00156                               );
00157
00158 VPRIVATE int NOsh_setupCalcGEOFLOW(
00159                               NOsh *thee,
00160                               NOsh_calc *elec
00161                               );
00162
00163 VPRIVATE int NOsh_setupCalcPBAM(
00164                                 NOsh *thee,
00165                                 NOsh_calc *elec
00166                                 );
00167
00168 VPRIVATE int NOsh_setupCalcPBSAM(
00169                                  NOsh *thee,
00170                                  NOsh_calc *elec
00171                                  );
00172
00173 VPRIVATE int NOsh_setupCalcBEMMANUAL(
00174                               NOsh *thee,
00175                               NOsh_calc *elec
00176                               );
00177
00178 VPRIVATE int NOsh_setupCalcGEOFLOWMANUAL(
00179                               NOsh *thee,
00180                               NOsh_calc *elec
00181                               );
00182
00183 VPRIVATE int NOsh_setupCalcPBAMAUTO(
00184                                    NOsh *thee,
00185                                    NOsh_calc *elec
00186                                    );
00187
00188 VPRIVATE int NOsh_setupCalcPBSAMAUTO(
00189                                     NOsh *thee,
00190                                     NOsh_calc *elec
00191                                     );
00192
00193 VPRIVATE int NOsh_setupCalcAPOL(
00194                                 NOsh *thee,
00195                                 NOsh_calc *elec
00196                                 );
00197
00198 #if !defined(VINLINE_NOSH)
00199
00200 VPUBLIC char* NOsh_getMolpath(NOsh *thee, int imol) {
00201     VASSERT(thee != VNULL);
00202     VASSERT(imol < thee->nmol);
00203     return thee->molpath[imol];
00204 }
00205 VPUBLIC char* NOsh_getDielXpath(NOsh *thee, int imol) {
00206     VASSERT(thee != VNULL);
00207     VASSERT(imol < thee->nmol);
00208     return thee->dielXpath[imol];
00209 }
00210 VPUBLIC char* NOsh_getDielYpath(NOsh *thee, int imol) {
00211     VASSERT(thee != VNULL);
00212     VASSERT(imol < thee->nmol);
00213     return thee->dielYpath[imol];
00214 }
00215 VPUBLIC char* NOsh_getDielZpath(NOsh *thee, int imol) {
00216     VASSERT(thee != VNULL);
00217     VASSERT(imol < thee->nmol);
00218     return thee->dielZpath[imol];
00219 }
00220 VPUBLIC char* NOsh_getKappapath(NOsh *thee, int imol) {
00221     VASSERT(thee != VNULL);
00222     VASSERT(imol < thee->nmol);
00223     return thee->kappapath[imol];
00224 }
00225 VPUBLIC char* NOsh_getPotpath(NOsh *thee, int imol) {
00226     VASSERT(thee != VNULL);
00227     VASSERT(imol < thee->nmol);
00228     return thee->potpath[imol];
00229 }
00230 VPUBLIC char* NOsh_getChargepath(NOsh *thee, int imol) {
00231     VASSERT(thee != VNULL);
00232     VASSERT(imol < thee->nmol);
00233     return thee->chargepath[imol];
00234 }
```

```
00235 VPUBLIC NOsh_calc* NOsh_getCalc(NOsh *thee, int icalc) {
00236     VASSERT(thee != VNULL);
00237     VASSERT(icalc < thee->ncalc);
00238     return thee->calc[icalc];
00239 }
00240 VPUBLIC int NOsh_getDielfmt(NOsh *thee, int i) {
00241     VASSERT(thee != VNULL);
00242     VASSERT(i < thee->ndiel);
00243     return (thee->dielfmt[i]);
00244 }
00245 VPUBLIC int NOsh_getKappafmt(NOsh *thee, int i) {
00246     VASSERT(thee != VNULL);
00247     VASSERT(i < thee->nkappa);
00248     return (thee->kappafmt[i]);
00249 }
00250 VPUBLIC int NOsh_getPotfmt(NOsh *thee, int i) {
00251     VASSERT(thee != VNULL);
00252     VASSERT(i < thee->npot);
00253     return (thee->potfmt[i]);
00254 }
00255 VPUBLIC int NOsh_getChargefmt(NOsh *thee, int i) {
00256     VASSERT(thee != VNULL);
00257     VASSERT(i < thee->ncharge);
00258     return (thee->chargefmt[i]);
00259 }
00260
00261
00262 #endif /* if !defined(VINLINE_NOSH) */
00263
00264 VPUBLIC NOsh_PrintType NOsh_printWhat(NOsh *thee, int iprint) {
00265     VASSERT(thee != VNULL);
00266     VASSERT(iprint < thee->nprint);
00267     return thee->printwhat[iprint];
00268 }
00269
00270 VPUBLIC int NOsh_printNarg(NOsh *thee, int iprint) {
00271     VASSERT(thee != VNULL);
00272     VASSERT(iprint < thee->nprint);
00273     return thee->printnarg[iprint];
00274 }
00275
00276 VPUBLIC int NOsh_elec2calc(NOsh *thee, int icalc) {
00277     VASSERT(thee != VNULL);
00278     VASSERT(icalc < thee->ncalc);
00279     return thee->elec2calc[icalc];
00280 }
00281
00282 VPUBLIC int NOsh_apol2calc(NOsh *thee, int icalc) {
00283     VASSERT(thee != VNULL);
00284     VASSERT(icalc < thee->ncalc);
00285     return thee->apol2calc[icalc];
00286 }
00287
00288 VPUBLIC char* NOsh_elecname(NOsh *thee, int ielec) {
00289     VASSERT(thee != VNULL);
00290     VASSERT(ielec < thee->nelec + 1);
00291     return thee->elecname[ielec];
00292 }
00293
00294 VPUBLIC int NOsh_printOp(NOsh *thee, int iprint, int iarg) {
00295     VASSERT(thee != VNULL);
00296     VASSERT(iprint < thee->nprint);
00297     VASSERT(iarg < thee->printnarg[iprint]);
00298     return thee->printop[iprint][iarg];
00299 }
00300
00301 VPUBLIC int NOsh_printCalc(NOsh *thee, int iprint, int iarg) {
00302     VASSERT(thee != VNULL);
00303     VASSERT(iprint < thee->nprint);
00304     VASSERT(iarg < thee->printnarg[iprint]);
00305     return thee->printcalc[iprint][iarg];
00306 }
00307
00308 VPUBLIC NOsh* NOsh_ctor(int rank, int size) {
00309
00310     /* Set up the structure */
00311     NOsh *thee = VNULL;
00312     thee = (NOsh*)Vmem_malloc(VNULL, 1, sizeof(NOsh) );
00313     VASSERT( thee != VNULL);
00314     VASSERT( NOsh_ctor2(thee, rank, size) );
00315
```

```
00316     return thee;
00317 }
00318
00319 VPUBLIC int NOsh_ctor2(NOsh *thee, int rank, int size) {
00320
00321     int i;
00322
00323     if (thee == VNULL) return 0;
00324
00325     thee->proc_rank = rank;
00326     thee->proc_size = size;
00327
00328     thee->ispara = 0;
00329     thee->parsed = 0;
00330
00331     thee->nmol = 0;
00332     thee->gotparm = 0;
00333     thee->ncharge = 0;
00334     thee->ndiel = 0;
00335     thee->nkappa = 0;
00336     thee->npot = 0;
00337     thee->nprint = 0;
00338
00339     for (i=0; i<NOSH_MAXCALC; i++) {
00340         thee->calc[i] = VNULL;
00341         thee->elec[i] = VNULL;
00342         thee->apol[i] = VNULL;
00343     }
00344     for (i=0; i<NOSH_MAXMOL; i++) {
00345         thee->alist[i] = VNULL;
00346     }
00347     thee->ncalc = 0;
00348     thee->nelec = 0;
00349     thee->napol = 0;
00350
00351     return 1;
00352 }
00353
00354 VPUBLIC void NOsh_dtor(NOsh **thee) {
00355     if ((*thee) != VNULL) {
00356         NOsh_dtor2(*thee);
00357         Vmem_free(VNULL, 1, sizeof(NOsh), (void **)thee);
00358         (*thee) = VNULL;
00359     }
00360 }
00361
00362 VPUBLIC void NOsh_dtor2(NOsh *thee) {
00363
00364     int i;
00365
00366     if (thee != VNULL) {
00367         for (i=0; i<(thee->ncalc); i++) NOsh_calc_dtor(&(thee->calc[i]));
00368         for (i=0; i<(thee->nelec); i++) NOsh_calc_dtor(&(thee->elec[i]));
00369         for (i=0; i<(thee->napol); i++) NOsh_calc_dtor(&(thee->apol[i]));
00370     }
00371
00372 }
00373
00374 VPUBLIC NOsh_calc* NOsh_calc_ctor(
00375                                   NOsh_CalcType calctype
00376                                   ) {
00377     NOsh_calc *thee;
00378     thee = (NOsh_calc *)Vmem_malloc(VNULL, 1, sizeof(NOsh_calc));
00379     thee->calctype = calctype;
00380
00381     thee->mgparm = VNULL;
00382     thee->femparm = VNULL;
00383     thee->apolparm = VNULL;
00384     thee->bemparm = VNULL;
00385     thee->geoflowparm = VNULL;
00386     thee->pbamparm = VNULL;
00387     thee->pbsamparm = VNULL;
00388
00389     switch (calctype) {
00390         case NCT_MG:
00391             thee->mgparm = MGparm_ctor(MCT_NONE);
00392             break;
00393         case NCT_FEM:
00394             thee->femparm = FEMparm_ctor(FCT_NONE);
00395             break;
00396         case NCT_APOL:
```

```
00397             thee->apolparm = APOLparm_ctor();
00398             break;
00399         case NCT_BEM:
00400             thee->bemparm = BEMparm_ctor(BCT_MANUAL);
00401             break;
00402         case NCT_GEOFLOW:
00403             thee->geoflowparm = GEOFLOWparm_ctor(GFCT_AUTO);
00404             thee->apolparm = APOLparm_ctor();
00405             break;
00406         case NCT_PBAM:
00407             thee->pbamparm = PBAMparm_ctor(PBAMCT_AUTO);
00408             break;
00409         case NCT_PBSAM:
00410             thee->pbamparm = PBAMparm_ctor(PBAMCT_AUTO);
00411             thee->pbsamparm = PBSAMparm_ctor(PBSAMCT_AUTO);
00412             break;
00413         default:
00414             Vnm_print(2, "NOsh_calc_ctor:  unknown calculation type (%d)!\n",
00415                       calctype);
00416             VASSERT(0);
00417     }
00418     thee->pbeparm = PBEparm_ctor();
00419
00420     return thee;
00421 }
00422
00423 VPUBLIC void NOsh_calc_dtor(
00424                             NOsh_calc **thee
00425                             ) {
00426
00427     NOsh_calc *calc = VNULL;
00428     calc = *thee;
00429     if (calc == VNULL) return;
00430
00431     switch (calc->calctype) {
00432         case NCT_MG:
00433             MGparm_dtor(&(calc->mgparm));
00434             break;
00435         case NCT_FEM:
00436             FEMparm_dtor(&(calc->femparm));
00437             break;
00438         case NCT_APOL:
00439             APOLparm_dtor(&(calc->apolparm));
00440             break;
00441         case NCT_BEM:
00442             BEMparm_dtor(&(calc->bemparm));
00443             break;
00444         case NCT_GEOFLOW:
00445             GEOFLOWparm_dtor(&(calc->geoflowparm));
00446             APOLparm_dtor(&(calc->apolparm));
00447             break;
00448         case NCT_PBAM:
00449             PBAMparm_dtor(&(calc->pbamparm));
00450             break;
00451         case NCT_PBSAM:
00452             PBAMparm_dtor(&(calc->pbamparm));
00453             PBSAMparm_dtor(&(calc->pbsamparm));
00454             break;
00455         default:
00456             Vnm_print(2, "NOsh_calc_ctor:  unknown calculation type (%d)!\n",
00457                       calc->calctype);
00458             VASSERT(0);
00459     }
00460     PBEparm_dtor(&(calc->pbeparm));
00461
00462     Vmem_free(VNULL, 1, sizeof(NOsh_calc), (void **)thee);
00463     calc = VNULL;
00464
00465 }
00466
00467 VPUBLIC int NOsh_calc_copy(
00468                             NOsh_calc *thee,
00469                             NOsh_calc *source
00470                             ) {
00471
00472     VASSERT(thee != VNULL);
00473     VASSERT(source != VNULL);
00474     VASSERT(thee->calctype == source->calctype);
00475     if (source->mgparm != VNULL)
00476         MGparm_copy(thee->mgparm, source->mgparm);
00477     if (source->femparm != VNULL)
```

```
00478         FEMparm_copy(thee->femparm, source->femparm);
00479     if (source->bemparm != VNULL)
00480         BEMparm_copy(thee->bemparm, source->bemparm);
00481     if (source->pbeparm != VNULL)
00482         PBEparm_copy(thee->pbeparm, source->pbeparm);
00483     if (source->apolparm != VNULL)
00484         APOLparm_copy(thee->apolparm, source->apolparm);
00485     /*I think here is where the the geoflow changes get lost*/
00486     if(source->geoflowparm != VNULL)
00487      GEOFLOWparm_copy(thee->geoflowparm, source->geoflowparm);
00488     if(source->pbamparm != VNULL)
00489       PBAMparm_copy(thee->pbamparm, source->pbamparm);
00490     if(source->pbsamparm != VNULL)
00491       PBSAMparm_copy(thee->pbsamparm, source->pbsamparm);
00492 
00493 
00494     return 1;
00495 
00496 }
00497 
00498 VPUBLIC int NOsh_parseInputFile(
00499                                 NOsh *thee,
00500                                 char *filename
00501                                 ) {
00502 
00503     Vio *sock;
00504     int rc;
00505 
00506     sock = Vio_ctor("FILE", "ASC", VNULL, filename, "r");
00507     rc = NOsh_parseInput(thee, sock);
00508     Vio_dtor(&sock);
00509 
00510     return rc;
00511 }
00512 
00513 VPUBLIC int NOsh_parseInput(
00514                             NOsh *thee,
00515                             Vio *sock
00516                             ) {
00517 
00518     char *MCwhiteChars = " =,;\t\r\n";
00519     char *MCcommChars  = "#%";
00520     char tok[VMAX_BUFSIZE];
00521 
00522     if (thee == VNULL) {
00523         Vnm_print(2, "NOsh_parseInput:  Got NULL thee!\n");
00524         return 0;
00525     }
00526 
00527     if (sock == VNULL) {
00528         Vnm_print(2, "NOsh_parseInput:  Got pointer to NULL socket!\n");
00529         Vnm_print(2, "NOsh_parseInput:  The specified input file was not found!\n");
00530         return 0;
00531     }
00532 
00533     if (thee->parsed) {
00534         Vnm_print(2, "NOsh_parseInput:  Already parsed an input file!\n");
00535         return 0;
00536     }
00537 
00538     if (Vio_accept(sock, 0) < 0) {
00539         Vnm_print(2, "NOsh_parseInput:  Problem reading from socket!\n");
00540         return 0;
00541     }
00542 
00543     /* Set up the whitespace and comment character definitions */
00544     Vio_setWhiteChars(sock, MCwhiteChars);
00545     Vio_setCommChars(sock, MCcommChars);
00546 
00547     /* We parse the file until we run out of tokens */
00548     Vnm_print(0, "NOsh_parseInput:  Starting file parsing...\n");
00549     while (Vio_scanf(sock, "%s", tok) == 1) {
00550         /* At the highest level, we look for keywords that indicate functions like:
00551 
00552         read => Read in a molecule file
00553         elec => Do an electrostatics calculation
00554         print => Print some results
00555         apolar => do a non-polar calculation
00556         quit => Quit
00557 
00558         These cause the code to go to a lower-level parser routine which
```

```
00559         handles keywords specific to the particular function.  Each
00560         lower-level parser routine then returns when it hits the "end"
00561         keyword.  Due to this simple layout, no nesting of these "function"
00562         sections is allowed.
00563         */
00564         if (Vstring_strcasecmp(tok, "read") == 0) {
00565           //           printf("read\n");
00566             Vnm_print(0, "NOsh: Parsing READ section\n");
00567             if (!NOsh_parseREAD(thee, sock)) return 0;
00568             Vnm_print(0, "NOsh: Done parsing READ section \
00569 (nmol=%d, ndiel=%d, nkappa=%d, ncharge=%d, npot=%d)\n", thee->nmol, thee->ndiel,
00570                       thee->nkappa, thee->ncharge,thee->npot);
00571         } else if (Vstring_strcasecmp(tok, "print") == 0) {
00572             Vnm_print(0, "NOsh: Parsing PRINT section\n");
00573             if (!NOsh_parsePRINT(thee, sock)) return 0;
00574             Vnm_print(0, "NOsh: Done parsing PRINT section\n");
00575         } else if (Vstring_strcasecmp(tok, "elec") == 0) {
00576             Vnm_print(0, "NOsh: Parsing ELEC section\n");
00577             if (!NOsh_parseELEC(thee, sock)) return 0;
00578             Vnm_print(0, "NOsh: Done parsing ELEC section (nelec = %d)\n",
00579                       thee->nelec);
00580         } else if (Vstring_strcasecmp(tok, "apolar") == 0) {
00581             Vnm_print(0, "NOsh: Parsing APOLAR section\n");
00582             if (!NOsh_parseAPOLAR(thee, sock)) return 0;
00583             Vnm_print(0, "NOsh: Done parsing APOLAR section (nelec = %d)\n",
00584                       thee->nelec);
00585         } else if (Vstring_strcasecmp(tok, "quit") == 0) {
00586             Vnm_print(0, "NOsh: Done parsing file (got QUIT)\n");
00587             break;
00588         } else {
00589             Vnm_print(2, "NOsh_parseInput: Ignoring undefined keyword %s!\n", tok);
00590         }
00591     }
00592
00593     thee->parsed = 1;
00594     return 1;
00595
00596 }
00597
00598 VPRIVATE int NOsh_parseREAD_MOL(NOsh *thee, Vio *sock) {
00599
00600     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00601     NOsh_MolFormat molfmt;
00602
00603     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00604     if (Vstring_strcasecmp(tok, "pqr") == 0) {
00605         molfmt = NMF_PQR;
00606         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00607         if (tok[0]=='"') {
00608             strcpy(strnew, "");
00609             while (tok[strlen(tok)-1] != '"') {
00610                 strcat(str, tok);
00611                 strcat(str, " ");
00612                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00613             }
00614             strcat(str, tok);
00615             strncpy(strnew, str+1, strlen(str)-2);
00616             strcpy(tok, strnew);
00617         }
00618         Vnm_print(0, "NOsh: Storing molecule %d path %s\n",
00619                   thee->nmol, tok);
00620         thee->molfmt[thee->nmol] = molfmt;
00621         strncpy(thee->molpath[thee->nmol], tok, VMAX_ARGLEN);
00622         (thee->nmol)++;
00623     } else if (Vstring_strcasecmp(tok, "pdb") == 0) {
00624         molfmt = NMF_PDB;
00625         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00626         if (tok[0]=='"') {
00627             strcpy(strnew, "");
00628             while (tok[strlen(tok)-1] != '"') {
00629                 strcat(str, tok);
00630                 strcat(str, " ");
00631                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00632             }
00633             strcat(str, tok);
00634             strncpy(strnew, str+1, strlen(str)-2);
00635             strcpy(tok, strnew);
00636         }
00637         Vnm_print(0, "NOsh: Storing molecule %d path %s\n",
00638                   thee->nmol, tok);
00639         thee->molfmt[thee->nmol] = molfmt;
```

```
00640         strncpy(thee->molpath[thee->nmol], tok, VMAX_ARGLEN);
00641         (thee->nmol)++;
00642     } else if (Vstring_strcasecmp(tok, "xml") == 0) {
00643         molfmt = NMF_XML;
00644         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00645         if (tok[0]=='"') {
00646             strcpy(strnew, "");
00647             while (tok[strlen(tok)-1] != '"') {
00648                 strcat(str, tok);
00649                 strcat(str, " ");
00650                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00651             }
00652             strcat(str, tok);
00653             strncpy(strnew, str+1, strlen(str)-2);
00654             strcpy(tok, strnew);
00655         }
00656         Vnm_print(0, "NOsh: Storing molecule %d path %s\n",
00657                   thee->nmol, tok);
00658         thee->molfmt[thee->nmol] = molfmt;
00659         strncpy(thee->molpath[thee->nmol], tok, VMAX_ARGLEN);
00660         (thee->nmol)++;
00661     } else {
00662         Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined mol format \
00663 %s!\n", tok);
00664     }
00665
00666     return 1;
00667
00668
00669 VERROR1:
00670         Vnm_print(2, "NOsh_parseREAD_MOL:  Ran out of tokens while parsing READ section!\n");
00671     return 0;
00672
00673 }
00674
00675 VPRIVATE int NOsh_parseREAD_PARM(NOsh *thee, Vio *sock) {
00676
00677     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00678     NOsh_ParmFormat parmfmt;
00679
00680     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00681     if (Vstring_strcasecmp(tok, "flat") == 0) {
00682         parmfmt = NPF_FLAT;
00683         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00684         if (tok[0]=='"') {
00685             strcpy(strnew, "");
00686             while (tok[strlen(tok)-1] != '"') {
00687                 strcat(str, tok);
00688                 strcat(str, " ");
00689                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00690             }
00691             strcat(str, tok);
00692             strncpy(strnew, str+1, strlen(str)-2);
00693             strcpy(tok, strnew);
00694         }
00695         if (thee->gotparm) {
00696             Vnm_print(2, "NOsh:  Hey!  You already specified a parameterfile (%s)!\n", thee->parmpath);
00697             Vnm_print(2, "NOsh:  I'm going to ignore this one (%s)!\n", tok);
00698         } else {
00699             thee->parmfmt = parmfmt;
00700             thee->gotparm = 1;
00701             strncpy(thee->parmpath, tok, VMAX_ARGLEN);
00702         }
00703     } else if(Vstring_strcasecmp(tok, "xml") == 0) {
00704         parmfmt = NPF_XML;
00705         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00706         if (tok[0]=='"') {
00707             strcpy(strnew, "");
00708             while (tok[strlen(tok)-1] != '"') {
00709                 strcat(str, tok);
00710                 strcat(str, " ");
00711                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00712             }
00713             strcat(str, tok);
00714             strncpy(strnew, str+1, strlen(str)-2);
00715             strcpy(tok, strnew);
00716         }
00717         if (thee->gotparm) {
00718             Vnm_print(2, "NOsh:  Hey!  You already specified a parameterfile (%s)!\n", thee->parmpath);
00719             Vnm_print(2, "NOsh:  I'm going to ignore this one (%s)!\n", tok);
00720         } else {
```

```
00721             thee->parmfmt = parmfmt;
00722             thee->gotparm = 1;
00723             strncpy(thee->parmpath, tok, VMAX_ARGLEN);
00724         }
00725
00726     } else {
00727         Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined parm format \
00728 %s!\n", tok);
00729     }
00730
00731     return 1;
00732
00733 VERROR1:
00734         Vnm_print(2, "NOsh_parseREAD_PARM:  Ran out of tokens while parsing READ section!\n");
00735     return 0;
00736
00737 }
00738
00739 VPRIVATE int NOsh_parseREAD_DIEL(NOsh *thee, Vio *sock) {
00740
00741     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00742     Vdata_Format dielfmt;
00743
00744     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00745     if (Vstring_strcasecmp(tok, "dx") == 0) {
00746         dielfmt = VDF_DX;
00747     //added VDF_BIN to take binary files.
00748     } else if (Vstring_strcasecmp(tok, "dxbin") == 0){
00749      dielfmt = VDF_DXBIN;
00750     }else if (Vstring_strcasecmp(tok, "gz") == 0) {
00751         dielfmt = VDF_GZ;
00752     } else {
00753         Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined format \
00754                   %s!\n", tok);
00755         return VRC_FAILURE;
00756     }
00757
00758     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00759     if (tok[0]=='"') {
00760         strcpy(strnew, "");
00761         while (tok[strlen(tok)-1] != '"') {
00762             strcat(str, tok);
00763             strcat(str, " ");
00764             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00765         }
00766         strcat(str, tok);
00767         strncpy(strnew, str+1, strlen(str)-2);
00768         strcpy(tok, strnew);
00769     }
00770     Vnm_print(0, "NOsh: Storing x-shifted dielectric map %d path \
00771               %s\n", thee->ndiel, tok);
00772     strncpy(thee->dielXpath[thee->ndiel], tok, VMAX_ARGLEN);
00773     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00774     Vnm_print(0, "NOsh: Storing y-shifted dielectric map %d path \
00775               %s\n", thee->ndiel, tok);
00776     strncpy(thee->dielYpath[thee->ndiel], tok, VMAX_ARGLEN);
00777     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00778     Vnm_print(0, "NOsh: Storing z-shifted dielectric map %d path \
00779               %s\n", thee->ndiel, tok);
00780     strncpy(thee->dielZpath[thee->ndiel], tok, VMAX_ARGLEN);
00781     thee->dielfmt[thee->ndiel] = dielfmt;
00782     (thee->ndiel)++;
00783
00784     return 1;
00785
00786 VERROR1:
00787         Vnm_print(2, "NOsh_parseREAD_DIEL:  Ran out of tokens while parsing READ \
00788 section!\n");
00789     return 0;
00790
00791 }
00792
00793 VPRIVATE int NOsh_parseREAD_KAPPA(NOsh *thee, Vio *sock) {
00794
00795     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00796     Vdata_Format kappafmt;
00797
00798     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00799     if (Vstring_strcasecmp(tok, "dx") == 0) {
00800         kappafmt = VDF_DX;
00801     } else if (Vstring_strcasecmp(tok, "gz") == 0) {
```

```
00802         kappafmt = VDF_GZ;
00803     } else if (Vstring_strcasecmp(tok,"dxbin") == 0) {
00804      kappafmt = VDF_DXBIN;
00805     } else {
00806 
00807         Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined format \
00808                   %s!\n", tok);
00809         return VRC_FAILURE;
00810     }
00811 
00812     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00813     if (tok[0]=='"') {
00814         strcpy(strnew, "");
00815         while (tok[strlen(tok)-1] != '"') {
00816             strcat(str, tok);
00817             strcat(str, " ");
00818             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00819         }
00820         strcat(str, tok);
00821         strncpy(strnew, str+1, strlen(str)-2);
00822         strcpy(tok, strnew);
00823     }
00824     Vnm_print(0, "NOsh: Storing kappa map %d path %s\n",
00825               thee->nkappa, tok);
00826     thee->kappafmt[thee->nkappa] = kappafmt;
00827     strncpy(thee->kappapath[thee->nkappa], tok, VMAX_ARGLEN);
00828     (thee->nkappa)++;
00829 
00830     return 1;
00831 
00832 VERROR1:
00833         Vnm_print(2, "NOsh_parseREAD:  Ran out of tokens while parsing READ \
00834 section!\n");
00835     return 0;
00836 
00837 }
00838 
00839 VPRIVATE int NOsh_parseREAD_POTENTIAL(NOsh *thee, Vio *sock) {
00840 
00841     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00842     Vdata_Format potfmt;
00843 
00844     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00845     if (Vstring_strcasecmp(tok, "dx") == 0) {
00846         potfmt = VDF_DX;
00847     } else if (Vstring_strcasecmp(tok, "gz") == 0) {
00848         potfmt = VDF_GZ;
00849     } else if(Vstring_strcasecmp(tok, "dxbin") == 0){
00850      potfmt = VDF_DXBIN;
00851     } else {
00852         Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined format \
00853                   %s!\n", tok);
00854         return VRC_FAILURE;
00855    }
00856 
00857     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00858     if (tok[0]=='"') {
00859         strcpy(strnew, "");
00860         while (tok[strlen(tok)-1] != '"') {
00861             strcat(str, tok);
00862             strcat(str, " ");
00863             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00864         }
00865         strcat(str, tok);
00866         strncpy(strnew, str+1, strlen(str)-2);
00867         strcpy(tok, strnew);
00868     }
00869     Vnm_print(0, "NOsh: Storing potential map %d path %s\n",
00870               thee->npot, tok);
00871     thee->potfmt[thee->npot] = potfmt;
00872     strncpy(thee->potpath[thee->npot], tok, VMAX_ARGLEN);
00873     (thee->npot)++;
00874 
00875     return 1;
00876 
00877 VERROR1:
00878     Vnm_print(2, "NOsh_parseREAD:  Ran out of tokens while parsing READ \
00879               section!\n");
00880     return 0;
00881 
00882 }
```

```
00883
00884 VPRIVATE int NOsh_parseREAD_CHARGE(NOsh *thee, Vio *sock) {
00885
00886     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00887     Vdata_Format chargefmt;
00888
00889     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00890     if (Vstring_strcasecmp(tok, "dx") == 0) {
00891         chargefmt = VDF_DX;
00892     }
00893     else if(Vstring_strcasecmp(tok, "dxbin") == 0){
00894      chargefmt = VDF_DXBIN;
00895     }else if (Vstring_strcasecmp(tok, "gz") == 0) {
00896         chargefmt = VDF_GZ;
00897     } else {
00898         Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined format \
00899                   %s!\n", tok);
00900         return VRC_FAILURE;
00901     }
00902
00903     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00904     if (tok[0]=='"') {
00905         strcpy(strnew, "");
00906         while (tok[strlen(tok)-1] != '"') {
00907             strcat(str, tok);
00908             strcat(str, " ");
00909             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00910         }
00911         strcat(str, tok);
00912         strncpy(strnew, str+1, strlen(str)-2);
00913         strcpy(tok, strnew);
00914     }
00915     Vnm_print(0, "NOsh: Storing charge map %d path %s\n",
00916               thee->ncharge, tok);
00917     thee->chargefmt[thee->ncharge] = chargefmt;
00918     strncpy(thee->chargepath[thee->ncharge], tok, VMAX_ARGLEN);
00919     (thee->ncharge)++;
00920
00921     return 1;
00922
00923 VERROR1:
00924         Vnm_print(2, "NOsh_parseREAD:  Ran out of tokens while parsing READ \
00925 section!\n");
00926     return 0;
00927
00928 }
00929
00930 VPRIVATE int NOsh_parseREAD_MESH(NOsh *thee, Vio *sock) {
00931
00932     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
00933     Vdata_Format meshfmt;
00934
00935     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00936     if (Vstring_strcasecmp(tok, "mcsf") == 0) {
00937         meshfmt = VDF_MCSF;
00938         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00939         if (tok[0]=='"') {
00940             strcpy(strnew, "");
00941             while (tok[strlen(tok)-1] != '"') {
00942                 strcat(str, tok);
00943                 strcat(str, " ");
00944                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00945             }
00946             strcat(str, tok);
00947             strncpy(strnew, str+1, strlen(str)-2);
00948             strcpy(tok, strnew);
00949         }
00950         Vnm_print(0, "NOsh: Storing mesh %d path %s\n",
00951                   thee->nmesh, tok);
00952         thee->meshfmt[thee->nmesh] = meshfmt;
00953         strncpy(thee->meshpath[thee->nmesh], tok, VMAX_ARGLEN);
00954         (thee->nmesh)++;
00955     } else {
00956         Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined mesh format \
00957                   %s!\n", tok);
00958     }
00959
00960     return 1;
00961
00962 VERROR1:
00963     Vnm_print(2, "NOsh_parseREAD:  Ran out of tokens while parsing READ \
```

```
00964                section!\n");
00965     return 0;
00966
00967 }
00968
00969
00970 VPRIVATE int NOsh_parseREAD(NOsh *thee, Vio *sock) {
00971
00972     char tok[VMAX_BUFSIZE];
00973
00974     if (thee == VNULL) {
00975         Vnm_print(2, "NOsh_parseREAD:  Got NULL thee!\n");
00976         return 0;
00977     }
00978
00979     if (sock == VNULL) {
00980         Vnm_print(2, "NOsh_parseREAD:  Got pointer to NULL socket!\n");
00981         return 0;
00982     }
00983
00984     if (thee->parsed) {
00985         Vnm_print(2, "NOsh_parseREAD:  Already parsed an input file!\n");
00986         return 0;
00987     }
00988
00989     /* Read until we run out of tokens (bad) or hit the "END" keyword (good) */
00990     while (Vio_scanf(sock, "%s", tok) == 1) {
00991         if (Vstring_strcasecmp(tok, "end") == 0) {
00992             Vnm_print(0, "NOsh: Done parsing READ section\n");
00993             return 1;
00994         } else if (Vstring_strcasecmp(tok, "mol") == 0) {
00995             NOsh_parseREAD_MOL(thee, sock);
00996         } else if (Vstring_strcasecmp(tok, "parm") == 0) {
00997             NOsh_parseREAD_PARM(thee,sock);
00998         } else if (Vstring_strcasecmp(tok, "diel") == 0) {
00999             NOsh_parseREAD_DIEL(thee,sock);
01000         } else if (Vstring_strcasecmp(tok, "kappa") == 0) {
01001             NOsh_parseREAD_KAPPA(thee,sock);
01002         } else if (Vstring_strcasecmp(tok, "pot") == 0) {
01003             NOsh_parseREAD_POTENTIAL(thee,sock);
01004         } else if (Vstring_strcasecmp(tok, "charge") == 0) {
01005             NOsh_parseREAD_CHARGE(thee,sock);
01006         } else if (Vstring_strcasecmp(tok, "mesh") == 0) {
01007             NOsh_parseREAD_MESH(thee,sock);
01008         } else {
01009             Vnm_print(2, "NOsh_parseREAD:  Ignoring undefined keyword %s!\n",
01010                       tok);
01011         }
01012     }
01013
01014     /* We ran out of tokens! */
01015     Vnm_print(2, "NOsh_parseREAD:  Ran out of tokens while parsing READ \
01016 section!\n");
01017     return 0;
01018
01019 }
01020
01021 VPRIVATE int NOsh_parsePRINT(NOsh *thee, Vio *sock) {
01022
01023     char tok[VMAX_BUFSIZE];
01024     char name[VMAX_BUFSIZE];
01025     int ti, idx, expect, ielec, iapol;
01026
01027     if (thee == VNULL) {
01028         Vnm_print(2, "NOsh_parsePRINT:  Got NULL thee!\n");
01029         return 0;
01030     }
01031
01032     if (sock == VNULL) {
01033         Vnm_print(2, "NOsh_parsePRINT:  Got pointer to NULL socket!\n");
01034         return 0;
01035     }
01036
01037     if (thee->parsed) {
01038         Vnm_print(2, "NOsh_parsePRINT:  Already parsed an input file!\n");
01039         return 0;
01040     }
01041
01042     idx = thee->nprint;
01043     if (thee->nprint >= NOSH_MAXPRINT) {
01044         Vnm_print(2, "NOsh_parsePRINT:  Exceeded max number (%d) of PRINT \
```

```
01045 sections\n",
01046                   NOSH_MAXPRINT);
01047         return 0;
01048     }
01049 
01050 
01051     /* The first thing we read is the thing we want to print */
01052     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01053     if (Vstring_strcasecmp(tok, "energy") == 0) {
01054         thee->printwhat[idx] = NPT_ENERGY;
01055         thee->printnarg[idx] = 0;
01056     } else if (Vstring_strcasecmp(tok, "force") == 0) {
01057         thee->printwhat[idx] = NPT_FORCE;
01058         thee->printnarg[idx] = 0;
01059     } else if (Vstring_strcasecmp(tok, "elecEnergy") == 0) {
01060         thee->printwhat[idx] = NPT_ELECENERGY;
01061         thee->printnarg[idx] = 0;
01062     } else if (Vstring_strcasecmp(tok, "elecForce") == 0) {
01063         thee->printwhat[idx] = NPT_ELECFORCE;
01064         thee->printnarg[idx] = 0;
01065     } else if (Vstring_strcasecmp(tok, "apolEnergy") == 0) {
01066         thee->printwhat[idx] = NPT_APOLENERGY;
01067         thee->printnarg[idx] = 0;
01068     } else if (Vstring_strcasecmp(tok, "apolForce") == 0) {
01069         thee->printwhat[idx] = NPT_APOLFORCE;
01070         thee->printnarg[idx] = 0;
01071     } else {
01072         Vnm_print(2, "NOsh_parsePRINT:  Undefined keyword %s while parsing \
01073 PRINT section!\n", tok);
01074         return 0;
01075     }
01076 
01077     expect = 0;   /* We first expect a calculation ID (0) then an op (1) */
01078 
01079     /* Read until we run out of tokens (bad) or hit the "END" keyword (good) */
01080     while (Vio_scanf(sock, "%s", tok) == 1) {
01081 
01082         /* The next thing we read is either END or an ARG OP ARG statement */
01083         if (Vstring_strcasecmp(tok, "end") == 0) {
01084             if (expect != 0) {
01085                 (thee->nprint)++;
01086                 (thee->printnarg[idx])++;
01087                 Vnm_print(0, "NOsh: Done parsing PRINT section\n");
01088                 return 1;
01089             } else {
01090                 Vnm_print(2, "NOsh_parsePRINT:  Got premature END to PRINT!\n");
01091                 return 0;
01092             }
01093         } else {
01094 
01095             /* Grab a calculation ID */
01096             if ((sscanf(tok, "%d", &ti) == 1) &&
01097                 (Vstring_isdigit(tok) == 1)) {
01098                 if (expect == 0) {
01099                     thee->printcalc[idx][thee->printnarg[idx]] = ti-1;
01100                     expect = 1;
01101                 } else {
01102                     Vnm_print(2, "NOsh_parsePRINT:  Syntax error in PRINT \
01103 section while reading %s!\n", tok);
01104                     return 0;
01105                 }
01106                 /* Grab addition operation */
01107             } else if (Vstring_strcasecmp(tok, "+") == 0) {
01108                 if (expect == 1) {
01109                     thee->printop[idx][thee->printnarg[idx]] = 0;
01110                     (thee->printnarg[idx])++;
01111                     expect = 0;
01112                     if (thee->printnarg[idx] >= NOSH_MAXPOP) {
01113                         Vnm_print(2, "NOsh_parsePRINT:  Exceeded max number \
01114 (%d) of arguments for PRINT section!\n",
01115                                   NOSH_MAXPOP);
01116                         return 0;
01117                     }
01118                 } else {
01119                     Vnm_print(2, "NOsh_parsePRINT:  Syntax error in PRINT \
01120 section while reading %s!\n", tok);
01121                     return 0;
01122                 }
01123                 /* Grab subtraction operation */
01124             } else if (Vstring_strcasecmp(tok, "-") == 0) {
01125                 if (expect == 1) {
```

```
01126                     thee->printop[idx][thee->printnarg[idx]] = 1;
01127                     (thee->printnarg[idx])++;
01128                     expect = 0;
01129                     if (thee->printnarg[idx] >= NOSH_MAXPOP) {
01130                         Vnm_print(2, "NOsh_parseREAD:  Exceeded max number \
01131 (%d) of arguments for PRINT section!\n",
01132                                   NOSH_MAXPOP);
01133                         return 0;
01134                     }
01135                 } else {
01136                     Vnm_print(2, "NOsh_parsePRINT:  Syntax error in PRINT \
01137 section while reading %s!\n", tok);
01138                     return 0;
01139                 }
01140                 /* Grab a calculation name from elec ID */
01141             } else if (sscanf(tok, "%s", name) == 1) {
01142                 if (expect == 0) {
01143                     for (ielec=0; ielec<thee->nelec; ielec++) {
01144                         if (Vstring_strcasecmp(thee->elecname[ielec], name) == 0) {
01145                             thee->printcalc[idx][thee->printnarg[idx]] = ielec;
01146                             expect = 1;
01147                             break;
01148                         }
01149                     }
01150                     for (iapol=0; iapol<thee->napol; iapol++) {
01151                         if (Vstring_strcasecmp(thee->apolname[iapol], name) == 0) {
01152                             thee->printcalc[idx][thee->printnarg[idx]] = iapol;
01153                             expect = 1;
01154                             break;
01155                         }
01156                     }
01157                     if (expect == 0) {
01158                         Vnm_print(2, "No ELEC or APOL statement has been named %s!\n",
01159                                   name);
01160                         return 0;
01161                     }
01162                 } else {
01163                     Vnm_print(2, "NOsh_parsePRINT:  Syntax error in PRINT \
01164 section while reading %s!\n", tok);
01165                     return 0;
01166                 }
01167                 /* Got bad operation */
01168             } else {
01169                 Vnm_print(2, "NOsh_parsePRINT:  Undefined keyword %s while \
01170 parsing PRINT section!\n", tok);
01171                 return 0;
01172             }
01173         } /* end parse token */
01174
01175     } /* end while */
01176
01177     VJMPERR1(0);
01178
01179     /* We ran out of tokens! */
01180 VERROR1:
01181         Vnm_print(2, "NOsh_parsePRINT:  Ran out of tokens while parsing PRINT \
01182 section!\n");
01183         return 0;
01184
01185 }
01186
01187 VPRIVATE int NOsh_parseELEC(NOsh *thee, Vio *sock) {
01188
01189     NOsh_calc *calc = VNULL;
01190
01191     char tok[VMAX_BUFSIZE];
01192
01193     if (thee == VNULL) {
01194         Vnm_print(2, "NOsh_parseELEC:  Got NULL thee!\n");
01195         return 0;
01196     }
01197
01198     if (sock == VNULL) {
01199         Vnm_print(2, "NOsh_parseELEC:  Got pointer to NULL socket!\n");
01200         return 0;
01201     }
01202
01203     if (thee->parsed) {
01204         Vnm_print(2, "NOsh_parseELEC:  Already parsed an input file!\n");
01205         return 0;
01206     }
```

```
01207
01208     /* Get a pointer to the latest ELEC calc object and update the ELEC
01209        statement number */
01210     if (thee->nelec >= NOSH_MAXCALC) {
01211         Vnm_print(2, "NOsh:  Too many electrostatics calculations in this \
01212 run!\n");
01213         Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
01214                   NOSH_MAXCALC);
01215         return 1;
01216     }
01217
01218     /* The next token HAS to be the method OR "name" */
01219     if (Vio_scanf(sock, "%s", tok) == 1) {
01220         if (Vstring_strcasecmp(tok, "name") == 0) {
01221             Vio_scanf(sock, "%s", tok);
01222             strncpy(thee->elecname[thee->nelec], tok, VMAX_ARGLEN);
01223             if (Vio_scanf(sock, "%s", tok) != 1) {
01224                 Vnm_print(2, "NOsh_parseELEC:  Ran out of tokens while reading \
01225 ELEC section!\n");
01226                 return 0;
01227             }
01228         }
01229         if (Vstring_strcasecmp(tok, "mg-manual") == 0) {
01230             thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_MG);
01231             calc = thee->elec[thee->nelec];
01232             (thee->nelec)++;
01233             calc->mgparm->type = MCT_MANUAL;
01234             return NOsh_parseMG(thee, sock, calc);
01235         } else if (Vstring_strcasecmp(tok, "mg-auto") == 0) {
01236             thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_MG);
01237             calc = thee->elec[thee->nelec];
01238             (thee->nelec)++;
01239             calc->mgparm->type = MCT_AUTO;
01240             return NOsh_parseMG(thee, sock, calc);
01241         } else if (Vstring_strcasecmp(tok, "mg-para") == 0) {
01242             thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_MG);
01243             calc = thee->elec[thee->nelec];
01244             (thee->nelec)++;
01245             calc->mgparm->type = MCT_PARALLEL;
01246             return NOsh_parseMG(thee, sock, calc);
01247         } else if (Vstring_strcasecmp(tok, "mg-dummy") == 0) {
01248             thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_MG);
01249             calc = thee->elec[thee->nelec];
01250             (thee->nelec)++;
01251             calc->mgparm->type = MCT_DUMMY;
01252             return NOsh_parseMG(thee, sock, calc);
01253         } else if (Vstring_strcasecmp(tok, "fe-manual") == 0) {
01254             thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_FEM);
01255             calc = thee->elec[thee->nelec];
01256             (thee->nelec)++;
01257             calc->femparm->type = FCT_MANUAL;
01258             return NOsh_parseFEM(thee, sock, calc);
01259         } else if (Vstring_strcasecmp(tok, "tabi") == 0) {
01260             thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_BEM);
01261             calc = thee->elec[thee->nelec];
01262             (thee->nelec)++;
01263             calc->bemparm->type = BCT_MANUAL;
01264             return NOsh_parseBEM(thee, sock, calc);
01265         } else if (Vstring_strcasecmp(tok, "bem") == 0) {
01266             thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_BEM);
01267             calc = thee->elec[thee->nelec];
01268             (thee->nelec)++;
01269             calc->bemparm->type = BCT_MANUAL;
01270             return NOsh_parseBEM(thee, sock, calc);
01271         } else if (Vstring_strcasecmp(tok, "bem-manual") == 0) {
01272             thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_BEM);
01273             calc = thee->elec[thee->nelec];
01274             (thee->nelec)++;
01275             calc->bemparm->type = BCT_MANUAL;
01276             return NOsh_parseBEM(thee, sock, calc);
01277         } else if (Vstring_strcasecmp(tok, "geoflow-manual") == 0) {
01278          Vnm_print(2, "Geoflow currently does not support geoflow-manual please use geoflow instead!\n");
01279          return 0;
01280         } else if(Vstring_strcasecmp(tok, "geoflow-none") == 0) {
01281          Vnm_print(2, "Geoflow currently does not support geoflow-none please use geoflow instead!\n");
01282          return 0;
01283         } else if (Vstring_strcasecmp(tok, "geoflow") == 0) {
01284             thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_GEOFLOW);
01285             calc = thee->elec[thee->nelec];
01286             (thee->nelec)++;
01287             calc->geoflowparm->type = GFCT_AUTO;
```

```
01288             return NOsh_parseGEOFLOW(thee, sock, calc);
01289         } else if (Vstring_strcasecmp(tok, "pbam") == 0) {
01290             thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_PBAM);
01291             calc = thee->elec[thee->nelec];
01292             (thee->nelec)++;
01293             calc->pbamparm->type = PBAMCT_AUTO;
01294             return NOsh_parsePBAM(thee, sock, calc);
01295         } else if (Vstring_strcasecmp(tok, "pbsam") == 0) {
01296             thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_PBSAM);
01297             calc = thee->elec[thee->nelec];
01298             (thee->nelec)++;
01299             calc->pbsamparm->type = PBSAMCT_AUTO;
01300             return NOsh_parsePBSAM(thee, sock, calc);
01301         } else {
01302             Vnm_print(2, "NOsh_parseELEC: The method (\"mg\",\"fem\", \"bem\", \"geoflow\" \"pbam\",
       \"pbsam\") or \
01303 \"name\" must be the first keyword in the ELEC section\n");
01304             return 0;
01305         }
01306     }
01307
01308     Vnm_print(2, "NOsh_parseELEC:  Ran out of tokens while reading ELEC section!\n");
01309     return 0;
01310
01311 }
01312
01313 VPRIVATE int NOsh_parseAPOLAR(NOsh *thee, Vio *sock) {
01314
01315     NOsh_calc *calc = VNULL;
01316
01317     char tok[VMAX_BUFSIZE];
01318
01319     if (thee == VNULL) {
01320         Vnm_print(2, "NOsh_parseAPOLAR:  Got NULL thee!\n");
01321         return 0;
01322     }
01323
01324     if (sock == VNULL) {
01325         Vnm_print(2, "NOsh_parseAPOLAR:  Got pointer to NULL socket!\n");
01326         return 0;
01327     }
01328
01329     if (thee->parsed) {
01330         Vnm_print(2, "NOsh_parseAPOLAR:  Already parsed an input file!\n");
01331         return 0;
01332     }
01333
01334     /* Get a pointer to the latest ELEC calc object and update the ELEC
01335         statement number */
01336     if (thee->napol >= NOSH_MAXCALC) {
01337         Vnm_print(2, "NOsh:  Too many non-polar calculations in this \
01338 run!\n");
01339         Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
01340                   NOSH_MAXCALC);
01341         return 1;
01342     }
01343
01344     /* The next token HAS to be the method OR "name" */
01345     if (Vio_scanf(sock, "%s", tok) == 1) {
01346         if (Vstring_strcasecmp(tok, "name") == 0) {
01347             Vio_scanf(sock, "%s", tok);
01348             strncpy(thee->apolname[thee->napol], tok, VMAX_ARGLEN);
01349
01350             /* Parse the non-polar parameters */
01351             thee->apol[thee->napol] = NOsh_calc_ctor(NCT_APOL);
01352             calc = thee->apol[thee->napol];
01353             (thee->napol)++;
01354             return NOsh_parseAPOL(thee, sock, calc);
01355 //        } else if (Vstring_strcasecmp(tok, "geoflow-manual") == 0) {
01356 //            thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_GEOFLOW);
01357 //            calc = thee->elec[thee->nelec];
01358 //            (thee->nelec)++;
01359 //            calc->geoflowparm->type = GFCT_MANUAL;
01360 //            return NOsh_parseGEOFLOW(thee, sock, calc);
01361 //        } else if (Vstring_strcasecmp(tok, "geoflow-auto") == 0) {
01362 //            thee->elec[thee->nelec] = NOsh_calc_ctor(NCT_GEOFLOW);
01363 //            calc = thee->elec[thee->nelec];
01364 //            (thee->nelec)++;
01365 //            calc->geoflowparm->type = GFCT_AUTO;
01366 //            return NOsh_parseGEOFLOW(thee, sock, calc);
01367         }
```

```
01368     }
01369
01370     return 1;
01371
01372 }
01373
01374 VPUBLIC int NOsh_setupElecCalc(
01375                             NOsh *thee,
01376                             Valist *alist[NOSH_MAXMOL]
01377                             ) {
01378     int ielec, imol, i;
01379     NOsh_calc *elec = VNULL;
01380     MGparm *mgparm = VNULL;
01381     Valist *mymol = VNULL;
01382
01383     VASSERT(thee != VNULL);
01384     for (imol=0; imol<thee->nmol; imol++) {
01385         thee->alist[imol] = alist[imol];
01386     }
01387
01388
01389     for (ielec=0; ielec<(thee->nelec); ielec++) {
01390         /* Unload the calculation object containing the ELEC information */
01391         elec = thee->elec[ielec];
01392
01393         if (((thee->ndiel != 0) || (thee->nkappa != 0) ||
01394              (thee->ncharge != 0) || (thee->npot != 0)) &&
01395                 (elec->pbeparm->calcforce != PCF_NO)) {
01396             Vnm_print(2, "NOsh_setupElecCalc:  Calculation of forces disabled because surface \
01397 map is used!\n");
01398             elec->pbeparm->calcforce = PCF_NO;
01399         }
01400
01401         /* Setup the calculation */
01402         switch (elec->calctype) {
01403             case NCT_MG:
01404                 /* Center on the molecules, if requested */
01405                 mgparm = elec->mgparm;
01406                 VASSERT(mgparm != VNULL);
01407                 if (elec->mgparm->cmeth == MCM_MOLECULE) {
01408                     VASSERT(mgparm->centmol >= 0);
01409                     VASSERT(mgparm->centmol < thee->nmol);
01410                     mymol = thee->alist[mgparm->centmol];
01411                     VASSERT(mymol != VNULL);
01412                     for (i=0; i<3; i++) {
01413                         mgparm->center[i] = mymol->center[i];
01414                     }
01415                 }
01416                 if (elec->mgparm->fcmeth == MCM_MOLECULE) {
01417                     VASSERT(mgparm->fcentmol >= 0);
01418                     VASSERT(mgparm->fcentmol < thee->nmol);
01419                     mymol = thee->alist[mgparm->fcentmol];
01420                     VASSERT(mymol != VNULL);
01421                     for (i=0; i<3; i++) {
01422                         mgparm->fcenter[i] = mymol->center[i];
01423                     }
01424                 }
01425                 if (elec->mgparm->ccmeth == MCM_MOLECULE) {
01426                     VASSERT(mgparm->ccentmol >= 0);
01427                     VASSERT(mgparm->ccentmol < thee->nmol);
01428                     mymol = thee->alist[mgparm->ccentmol];
01429                     VASSERT(mymol != VNULL);
01430                     for (i=0; i<3; i++) {
01431                         mgparm->ccenter[i] = mymol->center[i];
01432                     }
01433                 }
01434                 NOsh_setupCalcMG(thee, elec);
01435                 break;
01436             case NCT_FEM:
01437                 NOsh_setupCalcFEM(thee, elec);
01438                 break;
01439             case NCT_PBAM:
01440                 NOsh_setupCalcPBAM(thee, elec);
01441                 break;
01442             case NCT_PBSAM:
01443                 NOsh_setupCalcPBSAM(thee, elec);
01444                 break;
01445             case NCT_BEM:
01446                 NOsh_setupCalcBEM(thee, elec);
01447                 break;
01448             case NCT_GEOFLOW:
```

```
01449                 NOsh_setupCalcGEOFLOW(thee, elec);
01450                 break;
01451             default:
01452                 Vnm_print(2, "NOsh_setupCalc:  Invalid calculation type (%d)!\n",
01453                           elec->calctype);
01454                 return 0;
01455         }
01456 
01457         /* At this point, the most recently-created NOsh_calc object should be the
01458            one we use for results for this ELEC statement.  Assign it. */
01459         /* Associate ELEC statement with the calculation */
01460         thee->elec2calc[ielec] = thee->ncalc-1;
01461         Vnm_print(0, "NOsh_setupCalc:  Mapping ELEC statement %d (%d) to \
01462 calculation %d (%d)\n", ielec, ielec+1, thee->elec2calc[ielec],
01463                   thee->elec2calc[ielec]+1);
01464     }
01465 
01466     return 1;
01467 }
01468 
01469 VPUBLIC int NOsh_setupApolCalc(
01470                                 NOsh *thee,
01471                                 Valist *alist[NOSH_MAXMOL]
01472                                 ) {
01473     int iapol, imol;
01474     int doCalc = ACD_NO;
01475     NOsh_calc *calc = VNULL;
01476 
01477     VASSERT(thee != VNULL);
01478     for (imol=0; imol<thee->nmol; imol++) {
01479         thee->alist[imol] = alist[imol];
01480     }
01481 
01482     for (iapol=0; iapol<(thee->napol); iapol++) {
01483         /* Unload the calculation object containing the APOL information */
01484         calc = thee->apol[iapol];
01485 
01486         /* Setup the calculation */
01487         switch (calc->calctype) {
01488             case NCT_APOL:
01489                 NOsh_setupCalcAPOL(thee, calc);
01490                 doCalc = ACD_YES;
01491                 break;
01492             default:
01493                 Vnm_print(2, "NOsh_setupCalc:  Invalid calculation type (%d)!\n", calc->calctype);
01494                 return ACD_ERROR;
01495         }
01496         /* At this point, the most recently-created NOsh_calc object should be the
01497            one we use for results for this APOL statement.  Assign it. */
01498         /* Associate APOL statement with the calculation */
01499         thee->apol2calc[iapol] = thee->ncalc-1;
01500         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);
01501     }
01502 
01503     if(doCalc == ACD_YES){
01504         return ACD_YES;
01505     }else{
01506         return ACD_NO;
01507     }
01508 }
01509 
01510 VPUBLIC int NOsh_parseMG(
01511                          NOsh *thee,
01512                          Vio *sock,
01513                          NOsh_calc *elec
01514                          ) {
01515 
01516     char tok[VMAX_BUFSIZE];
01517     MGparm *mgparm = VNULL;
01518     PBEparm *pbeparm = VNULL;
01519     int rc;
01520 
01521     /* Check the arguments */
01522     if (thee == VNULL) {
01523         Vnm_print(2, "NOsh:  Got NULL thee!\n");
01524         return 0;
01525     }
01526     if (sock == VNULL) {
01527         Vnm_print(2, "NOsh:  Got pointer to NULL socket!\n");
01528         return 0;
```

```
01529     }
01530     if (elec == VNULL) {
01531         Vnm_print(2, "NOsh:  Got pointer to NULL elec object!\n");
01532         return 0;
01533     }
01534     mgparm = elec->mgparm;
01535     if (mgparm == VNULL) {
01536         Vnm_print(2, "NOsh:  Got pointer to NULL mgparm object!\n");
01537         return 0;
01538     }
01539     pbeparm = elec->pbeparm;
01540     if (pbeparm == VNULL) {
01541         Vnm_print(2, "NOsh:  Got pointer to NULL pbeparm object!\n");
01542         return 0;
01543     }
01544
01545     Vnm_print(0, "NOsh_parseMG: Parsing parameters for MG calculation\n");
01546
01547     /* Parallel stuff */
01548     if (mgparm->type == MCT_PARALLEL) {
01549         mgparm->proc_rank = thee->proc_rank;
01550         mgparm->proc_size = thee->proc_size;
01551         mgparm->setrank = 1;
01552         mgparm->setsize = 1;
01553     }
01554
01555
01556     /* Start snarfing tokens from the input stream */
01557     rc = 1;
01558     while (Vio_scanf(sock, "%s", tok) == 1) {
01559
01560         Vnm_print(0, "NOsh_parseMG:  Parsing %s...\n", tok);
01561
01562         /* See if it's an END token */
01563         if (Vstring_strcasecmp(tok, "end") == 0) {
01564             mgparm->parsed = 1;
01565             pbeparm->parsed = 1;
01566             rc = 1;
01567             break;
01568         }
01569
01570         /* Pass the token through a series of parsers */
01571         rc = PBEparm_parseToken(pbeparm, tok, sock);
01572         if (rc == -1) {
01573             Vnm_print(0, "NOsh_parseMG:  parsePBE error!\n");
01574             break;
01575         } else if (rc == 0) {
01576             /* Pass the token to the generic MG parser */
01577             rc = MGparm_parseToken(mgparm, tok, sock);
01578             if (rc == -1) {
01579                 Vnm_print(0, "NOsh_parseMG:  parseMG error!\n");
01580                 break;
01581             } else if (rc == 0) {
01582                 /* We ran out of parsers! */
01583                 Vnm_print(2, "NOsh:  Unrecognized keyword: %s\n", tok);
01584                 break;
01585             }
01586         }
01587     }
01588
01589     /* Handle various errors arising in the token-snarfing loop -- these all
01590         just result in simple returns right now */
01591     if (rc == -1) return 0;
01592     if (rc == 0) return 0;
01593
01594     /* Check the status of the parameter objects */
01595     if ((MGparm_check(mgparm) == VRC_FAILURE) || (!PBEparm_check(pbeparm))) {
01596         Vnm_print(2, "NOsh:  MG parameters not set correctly!\n");
01597         return 0;
01598     }
01599
01600     return 1;
01601 }
01602
01603 VPRIVATE int NOsh_setupCalcMG(
01604                                 NOsh *thee,
01605                                 NOsh_calc *calc
01606                                 ) {
01607
01608     MGparm *mgparm = VNULL;
01609
```

```
01610     VASSERT(thee != VNULL);
01611     VASSERT(calc != VNULL);
01612     mgparm = calc->mgparm;
01613     VASSERT(mgparm != VNULL);
01614
01615
01616     /* Now we're ready to whatever sorts of post-processing operations that are
01617         necessary for the various types of calculations */
01618     switch (mgparm->type) {
01619         case MCT_MANUAL:
01620             return NOsh_setupCalcMGMANUAL(thee, calc);
01621         case MCT_DUMMY:
01622             return NOsh_setupCalcMGMANUAL(thee, calc);
01623         case MCT_AUTO:
01624             return NOsh_setupCalcMGAUTO(thee, calc);
01625         case MCT_PARALLEL:
01626             return NOsh_setupCalcMGPARA(thee, calc);
01627         default:
01628             Vnm_print(2, "NOsh_setupCalcMG:  undefined MG calculation type (%d)!\n",
01629                       mgparm->type);
01630             return 0;
01631     }
01632
01633     /* Shouldn't get here */
01634     return 0;
01635 }
01636
01637
01638 VPRIVATE int NOsh_setupCalcBEM(
01639                               NOsh *thee,
01640                               NOsh_calc *calc
01641                               ) {
01642
01643     BEMparm *bemparm = VNULL;
01644
01645     VASSERT(thee != VNULL);
01646     VASSERT(calc != VNULL);
01647     bemparm = calc->bemparm;
01648     VASSERT(bemparm != VNULL);
01649
01650
01651     /* Now we're ready to whatever sorts of post-processing operations that are
01652         necessary for the various types of calculations */
01653     switch (bemparm->type) {
01654         case BCT_MANUAL:
01655             return NOsh_setupCalcBEMMANUAL(thee, calc);
01656         default:
01657             Vnm_print(2, "NOsh_setupCalcBEM:  undefined BEM calculation type (%d)!\n",
01658                       bemparm->type);
01659             return 0;
01660     }
01661
01662     /* Shouldn't get here */
01663     return 0;
01664 }
01665
01666 VPRIVATE int NOsh_setupCalcGEOFLOW(NOsh *thee, NOsh_calc *calc) {
01667
01668     GEOFLOWparm *parm = VNULL;
01669
01670     VASSERT(thee != VNULL);
01671     VASSERT(calc != VNULL);
01672     parm = calc->geoflowparm;
01673     VASSERT(parm != VNULL);
01674
01675
01676     /* Now we're ready to whatever sorts of post-processing operations that are
01677         necessary for the various types of calculations */
01678     if(/*parm->type == GFCT_MANUAL ||*/ parm->type == GFCT_AUTO){
01679         return NOsh_setupCalcGEOFLOWMANUAL(thee, calc);
01680     }else{
01681         Vnm_print(2, "NOsh_setupCalcGEOFLOW:  undefined GEOFLOW calculation type (%d)!\n", parm->type);
01682         return 0;
01683     }
01684 }
01685
01686 VPRIVATE int NOsh_setupCalcPBAM(NOsh *thee, NOsh_calc *calc){
01687
01688   PBAMparm *parm = VNULL;
01689
01690   VASSERT(thee!=VNULL);
```

```
01691   VASSERT(calc!=VNULL);
01692   parm = calc->pbamparm;
01693   VASSERT(parm!=VNULL);
01694
01695   if(parm->type == PBAMCT_AUTO){
01696     return NOsh_setupCalcPBAMAUTO(thee, calc);
01697   } else {
01698     Vnm_print(2, "NOsh_setupCalcPBAM: undefined PBAM calculation type (%d)!\n", parm->type);
01699     return 0;
01700   }
01701 }
01702
01703
01704 VPRIVATE int NOsh_setupCalcPBSAM(NOsh *thee, NOsh_calc *calc){
01705
01706   PBSAMparm *parm = VNULL;
01707
01708   VASSERT(thee!=VNULL);
01709   VASSERT(calc!=VNULL);
01710   parm = calc->pbsamparm;
01711   VASSERT(parm!=VNULL);
01712
01713   if(parm->type == PBSAMCT_AUTO){
01714     return NOsh_setupCalcPBSAMAUTO(thee, calc);
01715   } else {
01716     Vnm_print(2, "NOsh_setupCalcPBSAM: undefined PBSAM calculation type (%d)!\n", parm->type);
01717     return 0;
01718   }
01719 }
01720
01721
01722 VPRIVATE int NOsh_setupCalcFEM(
01723                                NOsh *thee,
01724                                NOsh_calc *calc
01725                                ) {
01726
01727     VASSERT(thee != VNULL);
01728     VASSERT(calc != VNULL);
01729     VASSERT(calc->femparm != VNULL);
01730
01731     /* Now we're ready to whatever sorts of post-processing operations that are
01732         * necessary for the various types of calculations */
01733     switch (calc->femparm->type) {
01734         case FCT_MANUAL:
01735             return NOsh_setupCalcFEMANUAL(thee, calc);
01736         default:
01737             Vnm_print(2, "NOsh_parseFEM:  unknown calculation type (%d)!\n",
01738                       calc->femparm->type);
01739             return 0;
01740     }
01741
01742     /* Shouldn't get here */
01743     return 0;
01744 }
01745
01746
01747 VPRIVATE int NOsh_setupCalcMGMANUAL(
01748                                     NOsh *thee,
01749                                     NOsh_calc *elec
01750                                     ) {
01751
01752     MGparm *mgparm = VNULL;
01753     PBEparm *pbeparm = VNULL;
01754     NOsh_calc *calc = VNULL;
01755
01756     if (thee == VNULL) {
01757         Vnm_print(2, "NOsh_setupCalcMGMANUAL:  Got NULL thee!\n");
01758         return 0;
01759     }
01760     if (elec == VNULL) {
01761         Vnm_print(2, "NOsh_setupCalcMGMANUAL:  Got NULL calc!\n");
01762         return 0;
01763     }
01764     mgparm = elec->mgparm;
01765     if (mgparm == VNULL) {
01766         Vnm_print(2, "NOsh_setupCalcMGMANUAL:  Got NULL mgparm -- was this calculation \
01767 set up?\n");
01768         return 0;
01769     }
01770     pbeparm = elec->pbeparm;
01771     if (pbeparm == VNULL) {
```

```
01772         Vnm_print(2, "NOsh_setupCalcMGMANUAL:  Got NULL pbeparm -- was this calculation \
01773 set up?\n");
01774         return 0;
01775     }
01776 
01777     /* Set up missing MG parameters */
01778     if (mgparm->setgrid == 0) {
01779         VASSERT(mgparm->setglen);
01780         mgparm->grid[0] = mgparm->glen[0]/((double)(mgparm->dime[0]-1));
01781         mgparm->grid[1] = mgparm->glen[1]/((double)(mgparm->dime[1]-1));
01782         mgparm->grid[2] = mgparm->glen[2]/((double)(mgparm->dime[2]-1));
01783     }
01784     if (mgparm->setglen == 0) {
01785         VASSERT(mgparm->setgrid);
01786         mgparm->glen[0] = mgparm->grid[0]*((double)(mgparm->dime[0]-1));
01787         mgparm->glen[1] = mgparm->grid[1]*((double)(mgparm->dime[1]-1));
01788         mgparm->glen[2] = mgparm->grid[2]*((double)(mgparm->dime[2]-1));
01789     }
01790 
01791     /* Check to see if he have any room left for this type of calculation, if
01792         so: set the calculation type, update the number of calculations of this type,
01793         and parse the rest of the section */
01794     if (thee->ncalc >= NOSH_MAXCALC) {
01795         Vnm_print(2, "NOsh:  Too many calculations in this run!\n");
01796         Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
01797                   NOSH_MAXCALC);
01798         return 0;
01799     }
01800 
01801     /* Get the next calculation object and increment the number of calculations */
01802     thee->calc[thee->ncalc] = NOsh_calc_ctor(NCT_MG);
01803     calc = thee->calc[thee->ncalc];
01804     (thee->ncalc)++;
01805 
01806 
01807 
01808     /* Copy over contents of ELEC */
01809     NOsh_calc_copy(calc, elec);
01810 
01811 
01812     return 1;
01813 }
01814 
01815 VPUBLIC int NOsh_setupCalcMGAUTO(
01816                                  NOsh *thee,
01817                                  NOsh_calc *elec
01818                                  ) {
01819 
01820     NOsh_calc *calcf = VNULL;
01821     NOsh_calc *calcc = VNULL;
01822     double fgrid[3], cgrid[3];
01823     double d[3], minf[3], maxf[3], minc[3], maxc[3];
01824     double redfrac, redrat[3], td;
01825     int ifocus, nfocus, tnfocus[3];
01826     int j;
01827     int icalc;
01828     int dofix;
01829 
01830     /* A comment about the coding style in this function.  I use lots and lots
01831         and lots of pointer deferencing.  I could (and probably should) save
01832         these in temporary variables.  However, since there are so many MGparm,
01833         etc. and NOsh_calc, etc. objects running around in this function, the
01834         current scheme is easiest to debug. */
01835 
01836 
01837     if (thee == VNULL) {
01838         Vnm_print(2, "NOsh_setupCalcMGAUTO:  Got NULL thee!\n");
01839         return 0;
01840     }
01841     if (elec == VNULL) {
01842         Vnm_print(2, "NOsh_setupCalcMGAUTO:  Got NULL elec!\n");
01843         return 0;
01844     }
01845     if (elec->mgparm == VNULL) {
01846         Vnm_print(2, "NOsh_setupCalcMGAUTO:  Got NULL mgparm!\n");
01847         return 0;
01848     }
01849     if (elec->pbeparm == VNULL) {
01850         Vnm_print(2, "NOsh_setupCalcMGAUTO:  Got NULL pbeparm!\n");
01851         return 0;
01852     }
```

```
01853
01854     Vnm_print(0, "NOsh_setupCalcMGAUTO(%s, %d):  coarse grid center = %g %g %g\n",
01855               __FILE__, __LINE__,
01856               elec->mgparm->ccenter[0],
01857               elec->mgparm->ccenter[1],
01858               elec->mgparm->ccenter[2]);
01859     Vnm_print(0, "NOsh_setupCalcMGAUTO(%s, %d):  fine grid center = %g %g %g\n",
01860               __FILE__, __LINE__,
01861               elec->mgparm->fcenter[0],
01862               elec->mgparm->fcenter[1],
01863               elec->mgparm->fcenter[2]);
01864
01865     /* Calculate the grid spacing on the coarse and fine levels */
01866     for (j=0; j<3; j++) {
01867         cgrid[j] = (elec->mgparm->cglen[j])/((double)(elec->mgparm->dime[j]-1));
01868         fgrid[j] = (elec->mgparm->fglen[j])/((double)(elec->mgparm->dime[j]-1));
01869         d[j] = elec->mgparm->fcenter[j] - elec->mgparm->ccenter[j];
01870     }
01871     Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  Coarse grid spacing = %g, %g, %g\n",
01872               __FILE__, __LINE__, cgrid[0], cgrid[1], cgrid[2]);
01873     Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  Fine grid spacing = %g, %g, %g\n",
01874               __FILE__, __LINE__, fgrid[0], fgrid[1], fgrid[2]);
01875     Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  Displacement between fine and \
01876 coarse grids = %g, %g, %g\n", __FILE__, __LINE__, d[0], d[1], d[2]);
01877
01878     /* Now calculate the number of focusing levels, never reducing the grid
01879         spacing by more than redfrac at each level */
01880     for (j=0; j<3; j++) {
01881         if (fgrid[j]/cgrid[j] < VREDFRAC) {
01882             redfrac = fgrid[j]/cgrid[j];
01883             td = log(redfrac)/log(VREDFRAC);
01884             tnfocus[j] = (int)ceil(td) + 1;
01885         } else tnfocus[j] = 2;
01886     }
01887     nfocus = VMAX2(VMAX2(tnfocus[0], tnfocus[1]), tnfocus[2]);
01888
01889     /* Now set redrat to the actual value by which the grid spacing is reduced
01890         at each level of focusing */
01891     for (j=0; j<3; j++) {
01892         redrat[j] = VPOW((fgrid[j]/cgrid[j]), 1.0/((double)nfocus-1.0));
01893     }
01894     Vnm_print(0, "NOsh:  %d levels of focusing with %g, %g, %g reductions\n",
01895               nfocus, redrat[0], redrat[1], redrat[2]);
01896
01897     /* Now that we know how many focusing levels to use, we're ready to set up
01898         the parameter objects */
01899     if (nfocus > (NOSH_MAXCALC-(thee->ncalc))) {
01900         Vnm_print(2, "NOsh:  Require more calculations than max (%d)!\n",
01901                   NOSH_MAXCALC);
01902         return 0;
01903     }
01904
01905     for (ifocus=0; ifocus<nfocus; ifocus++) {
01906
01907         /* Generate the new calc object */
01908         icalc = thee->ncalc;
01909         thee->calc[icalc] = NOsh_calc_ctor(NCT_MG);
01910         (thee->ncalc)++;
01911
01912         /* This is the _current_ NOsh_calc object */
01913         calcf = thee->calc[icalc];
01914         /* This is the _previous_ Nosh_calc object */
01915         if (ifocus != 0) {
01916             calcc = thee->calc[icalc-1];
01917         } else {
01918             calcc = VNULL;
01919         }
01920
01921         /* Copy over most of the parameters from the ELEC object */
01922         NOsh_calc_copy(calcf, elec);
01923
01924         /* Set up the grid lengths and spacings */
01925         if (ifocus == 0) {
01926             for (j=0; j<3; j++) {
01927                 calcf->mgparm->grid[j] = cgrid[j];
01928                 calcf->mgparm->glen[j] = elec->mgparm->cglen[j];
01929             }
01930         } else {
01931             for (j=0; j<3; j++) {
01932                 calcf->mgparm->grid[j] = redrat[j]*(calcc->mgparm->grid[j]);
01933                 calcf->mgparm->glen[j] = redrat[j]*(calcc->mgparm->glen[j]);
```

```
01934             }
01935         }
01936         calcf->mgparm->setgrid = 1;
01937         calcf->mgparm->setglen = 1;
01938 
01939         /* Get centers and centering method from coarse and fine meshes */
01940         if (ifocus == 0) {
01941             calcf->mgparm->cmeth = elec->mgparm->ccmeth;
01942             calcf->mgparm->centmol = elec->mgparm->ccentmol;
01943             for (j=0; j<3; j++) {
01944                 calcf->mgparm->center[j] = elec->mgparm->ccenter[j];
01945             }
01946         } else if (ifocus == (nfocus-1)) {
01947             calcf->mgparm->cmeth = elec->mgparm->fcmeth;
01948             calcf->mgparm->centmol = elec->mgparm->fcentmol;
01949             for (j=0; j<3; j++) {
01950                 calcf->mgparm->center[j] = elec->mgparm->fcenter[j];
01951             }
01952         } else {
01953             calcf->mgparm->cmeth = MCM_FOCUS;
01954             /* TEMPORARILY move the current grid center
01955             to the fine grid center.  In general, this will move portions of
01956             the current mesh off the immediately-coarser mesh.  We'll fix that
01957             in the next step. */
01958             for (j=0; j<3; j++) {
01959                 calcf->mgparm->center[j] = elec->mgparm->fcenter[j];
01960             }
01961         }
01962 
01963 
01964         /* As mentioned above, it is highly likely that the previous "jump"
01965             to the fine grid center put portions of the current mesh off the
01966             previous (coarser) mesh.  Fix this by displacing the current mesh
01967             back onto the previous coarser mesh.  */
01968         if (ifocus != 0) {
01969             Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  starting mesh \
01970 repositioning.\n", __FILE__, __LINE__);
01971             Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  coarse mesh center = \
01972 %g %g %g\n", __FILE__, __LINE__,
01973                       calcc->mgparm->center[0],
01974                       calcc->mgparm->center[1],
01975                       calcc->mgparm->center[2]);
01976             Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  coarse mesh upper corner = \
01977 %g %g %g\n", __FILE__, __LINE__,
01978                       calcc->mgparm->center[0]+0.5*(calcc->mgparm->glen[0]),
01979                       calcc->mgparm->center[1]+0.5*(calcc->mgparm->glen[1]),
01980                       calcc->mgparm->center[2]+0.5*(calcc->mgparm->glen[2]));
01981             Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  coarse mesh lower corner = \
01982 %g %g %g\n", __FILE__, __LINE__,
01983                       calcc->mgparm->center[0]-0.5*(calcc->mgparm->glen[0]),
01984                       calcc->mgparm->center[1]-0.5*(calcc->mgparm->glen[1]),
01985                       calcc->mgparm->center[2]-0.5*(calcc->mgparm->glen[2]));
01986             Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  initial fine mesh upper corner = \
01987 %g %g %g\n", __FILE__, __LINE__,
01988                       calcf->mgparm->center[0]+0.5*(calcf->mgparm->glen[0]),
01989                       calcf->mgparm->center[1]+0.5*(calcf->mgparm->glen[1]),
01990                       calcf->mgparm->center[2]+0.5*(calcf->mgparm->glen[2]));
01991             Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  initial fine mesh lower corner = \
01992 %g %g %g\n", __FILE__, __LINE__,
01993                       calcf->mgparm->center[0]-0.5*(calcf->mgparm->glen[0]),
01994                       calcf->mgparm->center[1]-0.5*(calcf->mgparm->glen[1]),
01995                       calcf->mgparm->center[2]-0.5*(calcf->mgparm->glen[2]));
01996             for (j=0; j<3; j++) {
01997                 /* Check if we've fallen off of the lower end of the mesh */
01998                 dofix = 0;
01999                 minf[j] = calcf->mgparm->center[j]
02000                     - 0.5*(calcf->mgparm->glen[j]);
02001                 minc[j] = calcc->mgparm->center[j]
02002                     - 0.5*(calcc->mgparm->glen[j]);
02003                 d[j] = minc[j] - minf[j];
02004                 if (d[j] >= VSMALL) {
02005                     if (ifocus == (nfocus-1)) {
02006                         Vnm_print(2, "NOsh_setupCalcMGAUTO:  Error!  Finest \
02007 mesh has fallen off the coarser meshes!\n");
02008                         Vnm_print(2, "NOsh_setupCalcMGAUTO:  difference in min %d-\
02009 direction = %g\n", j, d[j]);
02010                         Vnm_print(2, "NOsh_setupCalcMGAUTO:  min fine = %g %g %g\n",
02011                                   minf[0], minf[1], minf[2]);
02012                         Vnm_print(2, "NOsh_setupCalcMGAUTO:  min coarse = %g %g %g\n",
02013                                   minc[0], minc[1], minc[2]);
02014                         VASSERT(0);
```

```
02015                     } else {
02016                         Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  ifocus = %d, \
02017 fixing mesh min violation (%g in %d-direction).\n", __FILE__, __LINE__, ifocus,
02018                                   d[j], j);
02019                         calcf->mgparm->center[j] += d[j];
02020                         dofix = 1;
02021                     }
02022                 }
02023                 /* Check if we've fallen off of the upper end of the mesh */
02024                 maxf[j] = calcf->mgparm->center[j] \
02025                     + 0.5*(calcf->mgparm->glen[j]);
02026                 maxc[j] = calcc->mgparm->center[j] \
02027                     + 0.5*(calcc->mgparm->glen[j]);
02028                 d[j] = maxf[j] - maxc[j];
02029                 if (d[j] >= VSMALL) {
02030                     if (ifocus == (nfocus-1)) {
02031                         Vnm_print(2, "NOsh_setupCalcMGAUTO:  Error!  Finest \
02032 mesh has fallen off the coarser meshes!\n");
02033                         Vnm_print(2, "NOsh_setupCalcMGAUTO:  difference in %d-\
02034 direction = %g\n", j, d[j]);
02035                         VASSERT(0);
02036                     } else {
02037                         /* If we already fixed the lower boundary and we now need
02038                         to fix the upper boundary, we have a serious problem. */
02039                         if (dofix) {
02040                             Vnm_print(2, "NOsh_setupCalcMGAUTO:  Error!  Both \
02041 ends of the finer mesh do not fit in the bigger mesh!\n");
02042                             VASSERT(0);
02043                         }
02044                         Vnm_print(0, "NOsh_setupCalcMGAUTO(%s, %d):  ifocus = %d, \
02045 fixing mesh max violation (%g in %d-direction).\n", __FILE__, __LINE__, ifocus,
02046                                   d[j], j);
02047                         calcf->mgparm->center[j] -= d[j];
02048                         dofix = 1;
02049                     }
02050                 }
02051             }
02052             Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  final fine mesh upper corner = \
02053 %g %g %g\n", __FILE__, __LINE__,
02054                       calcf->mgparm->center[0]+0.5*(calcf->mgparm->glen[0]),
02055                       calcf->mgparm->center[1]+0.5*(calcf->mgparm->glen[1]),
02056                       calcf->mgparm->center[2]+0.5*(calcf->mgparm->glen[2]));
02057             Vnm_print(0, "NOsh_setupCalcMGAUTO (%s, %d):  final fine mesh lower corner = \
02058 %g %g %g\n", __FILE__, __LINE__,
02059                       calcf->mgparm->center[0]-0.5*(calcf->mgparm->glen[0]),
02060                       calcf->mgparm->center[1]-0.5*(calcf->mgparm->glen[1]),
02061                       calcf->mgparm->center[2]-0.5*(calcf->mgparm->glen[2]));
02062         }
02063 
02064         /* Finer levels have focusing boundary conditions */
02065         if (ifocus != 0) calcf->pbeparm->bcfl = BCFL_FOCUS;
02066 
02067         /* Only the finest level handles I/O and needs to worry about disjoint
02068            partitioning */
02069         if (ifocus != (nfocus-1)) calcf->pbeparm->numwrite = 0;
02070 
02071         /* Reset boundary flags for everything except parallel focusing */
02072         if (calcf->mgparm->type != MCT_PARALLEL)  {
02073             Vnm_print(0, "NOsh_setupMGAUTO:  Resetting boundary flags\n");
02074             for (j=0; j<6; j++) calcf->mgparm->partDisjOwnSide[j] = 0;
02075             for (j=0; j<3; j++) {
02076                 calcf->mgparm->partDisjCenter[j] = 0;
02077                 calcf->mgparm->partDisjLength[j] = calcf->mgparm->glen[j];
02078             }
02079         }
02080 
02081 
02082         calcf->mgparm->parsed = 1;
02083     }
02084 
02085 
02086     return 1;
02087 }
02088 
02089 /* Author:   Nathan Baker and Todd Dolinsky */
02090 VPUBLIC int NOsh_setupCalcMGPARA(
02091                                  NOsh *thee,
02092                                  NOsh_calc *elec
02093                                  ) {
02094 
02095     /* NEW (25-Jul-2006):  This code should produce modify the ELEC statement
```

```
02096     and pass it on to MGAUTO for further processing. */
02097
02098     MGparm *mgparm = VNULL;
02099     double ofrac;
02100     double hx, hy, hzed;
02101     double xofrac, yofrac, zofrac;
02102     int rank, size, npx, npy, npz, nproc, ip, jp, kp;
02103     int xeffGlob, yeffGlob, zeffGlob, xDisj, yDisj, zDisj;
02104     int xigminDisj, xigmaxDisj, yigminDisj, yigmaxDisj, zigminDisj, zigmaxDisj;
02105     int xigminOlap, xigmaxOlap, yigminOlap, yigmaxOlap, zigminOlap, zigmaxOlap;
02106     int xOlapReg, yOlapReg, zOlapReg;
02107     double xlenDisj, ylenDisj, zlenDisj;
02108     double xcentDisj, ycentDisj, zcentDisj;
02109     double xcentOlap, ycentOlap, zcentOlap;
02110     double xlenOlap, ylenOlap, zlenOlap;
02111     double xminOlap, xmaxOlap, yminOlap, ymaxOlap, zminOlap, zmaxOlap;
02112     double xminDisj, xmaxDisj, yminDisj, ymaxDisj, zminDisj, zmaxDisj;
02113     double xcent, ycent, zcent;
02114
02115     /* Grab some useful variables */
02116     VASSERT(thee != VNULL);
02117     VASSERT(elec != VNULL);
02118     mgparm = elec->mgparm;
02119     VASSERT(mgparm != VNULL);
02120
02121     /* Grab some useful variables */
02122     ofrac = mgparm->ofrac;
02123     npx = mgparm->pdime[0];
02124     npy = mgparm->pdime[1];
02125     npz = mgparm->pdime[2];
02126     nproc = npx*npy*npz;
02127
02128     /* If this is not an asynchronous calculation, then we need to make sure we
02129        have all the necessary MPI information */
02130     if (mgparm->setasync == 0) {
02131
02132 #ifndef HAVE_MPI_H
02133
02134         Vnm_tprint(2, "NOsh_setupCalcMGPARA:  Oops!  You're trying to perform \
02135 an 'mg-para' (parallel) calculation\n");
02136         Vnm_tprint(2, "NOsh_setupCalcMGPARA:  with a version of APBS that wasn't \
02137 compiled with MPI!\n");
02138         Vnm_tprint(2, "NOsh_setupCalcMGPARA:  Perhaps you meant to use the \
02139 'async' flag?\n");
02140         Vnm_tprint(2, "NOsh_setupCalcMGPARA:  Bailing out!\n");
02141
02142         return 0;
02143
02144 #endif
02145
02146         rank = thee->proc_rank;
02147         size = thee->proc_size;
02148         Vnm_print(0, "NOsh_setupCalcMGPARA:  Hello from processor %d of %d\n", rank,
02149                   size);
02150
02151         /* Check to see if we have too many processors.  If so, then simply set
02152            this processor to duplicating the work of processor 0. */
02153         if (rank > (nproc-1)) {
02154             Vnm_print(2, "NOsh_setupMGPARA:  There are more processors available than\
02155 the %d you requested.\n", nproc);
02156             Vnm_print(2, "NOsh_setupMGPARA:  Eliminating processor %d\n", rank);
02157             thee->bogus = 1;
02158             rank = 0;
02159         }
02160
02161         /* Check to see if we have too few processors.  If so, this is a fatal
02162            error. */
02163         if (size < nproc) {
02164             Vnm_print(2, "NOsh_setupMGPARA:  There are too few processors (%d) to \
02165 satisfy requirements (%d)\n", size, nproc);
02166             return 0;
02167         }
02168
02169         Vnm_print(0, "NOsh_setupMGPARA:  Hello (again) from processor %d of %d\n",
02170                   rank, size);
02171
02172     } else { /* Setting up for an asynchronous calculation. */
02173
02174         rank = mgparm->async;
02175
02176         thee->ispara = 1;
```

```
02177         thee->proc_rank = rank;
02178
02179         /* Check to see if the async id is greater than the number of
02180         * processors.  If so, this is a fatal error. */
02181         if (rank > (nproc-1)) {
02182             Vnm_print(2, "NOsh_setupMGPARA:  The processor id you requested (%d) \
02183 is not within the range of processors available (0-%d)\n", rank, (nproc-1));
02184             return 0;
02185         }
02186     }
02187
02188     /* Calculate the processor's coordinates in the processor grid */
02189     kp = (int)floor(rank/(npx*npy));
02190     jp = (int)floor((rank-kp*npx*npy)/npx);
02191     ip = rank - kp*npx*npy - jp*npx;
02192     Vnm_print(0, "NOsh_setupMGPARA:  Hello world from PE (%d, %d, %d)\n",
02193               ip, jp, kp);
02194
02195     /* Calculate effective overlap fractions for uneven processor distributions */
02196     if (npx == 1) xofrac = 0.0;
02197     else xofrac = ofrac;
02198     if (npy == 1) yofrac = 0.0;
02199     else yofrac = ofrac;
02200     if (npz == 1) zofrac = 0.0;
02201     else zofrac = ofrac;
02202
02203     /* Calculate the global grid size and spacing */
02204     xDisj = (int)VFLOOR(mgparm->dime[0]/(1 + 2*xofrac) + 0.5);
02205     xeffGlob = npx*xDisj;
02206     hx = mgparm->fglen[0]/(double)(xeffGlob-1);
02207     yDisj = (int)VFLOOR(mgparm->dime[1]/(1 + 2*yofrac) + 0.5);
02208     yeffGlob = npy*yDisj;
02209     hy = mgparm->fglen[1]/(double)(yeffGlob-1);
02210     zDisj = (int)VFLOOR(mgparm->dime[2]/(1 + 2*zofrac) + 0.5);
02211     zeffGlob = npz*zDisj;
02212     hzed = mgparm->fglen[2]/(double)(zeffGlob-1);
02213     Vnm_print(0, "NOsh_setupMGPARA:  Global Grid size = (%d, %d, %d)\n",
02214               xeffGlob, yeffGlob, zeffGlob);
02215     Vnm_print(0, "NOsh_setupMGPARA:  Global Grid Spacing = (%.3f, %.3f, %.3f)\n",
02216               hx, hy, hzed);
02217     Vnm_print(0, "NOsh_setupMGPARA:  Processor Grid Size = (%d, %d, %d)\n",
02218               xDisj, yDisj, zDisj);
02219
02220     /* Calculate the maximum and minimum processor grid points */
02221     xigminDisj = ip*xDisj;
02222     xigmaxDisj = xigminDisj + xDisj - 1;
02223     yigminDisj = jp*yDisj;
02224     yigmaxDisj = yigminDisj + yDisj - 1;
02225     zigminDisj = kp*zDisj;
02226     zigmaxDisj = zigminDisj + zDisj - 1;
02227     Vnm_print(0, "NOsh_setupMGPARA:  Min Grid Points for this proc. (%d, %d, %d)\n",
02228               xigminDisj, yigminDisj, zigminDisj);
02229     Vnm_print(0, "NOsh_setupMGPARA:  Max Grid Points for this proc. (%d, %d, %d)\n",
02230               xigmaxDisj, yigmaxDisj, zigmaxDisj);
02231
02232
02233     /* Calculate the disjoint partition length and center displacement */
02234     xminDisj = VMAX2(hx*(xigminDisj-0.5), 0.0);
02235     xmaxDisj = VMIN2(hx*(xigmaxDisj+0.5), mgparm->fglen[0]);
02236     xlenDisj = xmaxDisj - xminDisj;
02237     yminDisj = VMAX2(hy*(yigminDisj-0.5), 0.0);
02238     ymaxDisj = VMIN2(hy*(yigmaxDisj+0.5), mgparm->fglen[1]);
02239     ylenDisj = ymaxDisj - yminDisj;
02240     zminDisj = VMAX2(hzed*(zigminDisj-0.5), 0.0);
02241     zmaxDisj = VMIN2(hzed*(zigmaxDisj+0.5), mgparm->fglen[2]);
02242     zlenDisj = zmaxDisj - zminDisj;
02243
02244     xcent = 0.5*mgparm->fglen[0];
02245     ycent = 0.5*mgparm->fglen[1];
02246     zcent = 0.5*mgparm->fglen[2];
02247
02248     xcentDisj = xminDisj + 0.5*xlenDisj - xcent;
02249     ycentDisj = yminDisj + 0.5*ylenDisj - ycent;
02250     zcentDisj = zminDisj + 0.5*zlenDisj - zcent;
02251     if (VABS(xcentDisj) < VSMALL) xcentDisj = 0.0;
02252     if (VABS(ycentDisj) < VSMALL) ycentDisj = 0.0;
02253     if (VABS(zcentDisj) < VSMALL) zcentDisj = 0.0;
02254
02255     Vnm_print(0, "NOsh_setupMGPARA:  Disj part length = (%g, %g, %g)\n",
02256               xlenDisj, ylenDisj, zlenDisj);
02257     Vnm_print(0, "NOsh_setupMGPARA:  Disj part center displacement = (%g, %g, %g)\n",
```

```
02258               xcentDisj, ycentDisj, zcentDisj);
02259
02260     /* Calculate the overlapping partition length and center displacement */
02261     xOlapReg = 0;
02262     yOlapReg = 0;
02263     zOlapReg = 0;
02264     if (npx != 1) xOlapReg = (int)VFLOOR(xofrac*mgparm->fglen[0]/npx/hx + 0.5) + 1;
02265     if (npy != 1) yOlapReg = (int)VFLOOR(yofrac*mgparm->fglen[1]/npy/hy + 0.5) + 1;
02266     if (npz != 1) zOlapReg = (int)VFLOOR(zofrac*mgparm->fglen[2]/npz/hzed + 0.5) + 1;
02267
02268     Vnm_print(0, "NOsh_setupMGPARA:  No. of Grid Points in Overlap (%d, %d, %d)\n",
02269               xOlapReg, yOlapReg, zOlapReg);
02270
02271     if (ip == 0) xigminOlap = 0;
02272     else if (ip == (npx - 1)) xigminOlap = xeffGlob - mgparm->dime[0];
02273     else xigminOlap = xigminDisj - xOlapReg;
02274     xigmaxOlap = xigminOlap + mgparm->dime[0] - 1;
02275
02276     if (jp == 0) yigminOlap = 0;
02277     else if (jp == (npy - 1)) yigminOlap = yeffGlob - mgparm->dime[1];
02278     else yigminOlap = yigminDisj - yOlapReg;
02279     yigmaxOlap = yigminOlap + mgparm->dime[1] - 1;
02280
02281     if (kp == 0) zigminOlap = 0;
02282     else if (kp == (npz - 1)) zigminOlap = zeffGlob - mgparm->dime[2];
02283     else zigminOlap = zigminDisj - zOlapReg;
02284     zigmaxOlap = zigminOlap + mgparm->dime[2] - 1;
02285
02286     Vnm_print(0, "NOsh_setupMGPARA:  Min Grid Points with Overlap (%d, %d, %d)\n",
02287               xigminOlap, yigminOlap, zigminOlap);
02288     Vnm_print(0, "NOsh_setupMGPARA:  Max Grid Points with Overlap (%d, %d, %d)\n",
02289               xigmaxOlap, yigmaxOlap, zigmaxOlap);
02290
02291     xminOlap = hx * xigminOlap;
02292     xmaxOlap = hx * xigmaxOlap;
02293     yminOlap = hy * yigminOlap;
02294     ymaxOlap = hy * yigmaxOlap;
02295     zminOlap = hzed * zigminOlap;
02296     zmaxOlap = hzed * zigmaxOlap;
02297
02298     xlenOlap = xmaxOlap - xminOlap;
02299     ylenOlap = ymaxOlap - yminOlap;
02300     zlenOlap = zmaxOlap - zminOlap;
02301
02302     xcentOlap = (xminOlap + 0.5*xlenOlap) - xcent;
02303     ycentOlap = (yminOlap + 0.5*ylenOlap) - ycent;
02304     zcentOlap = (zminOlap + 0.5*zlenOlap) - zcent;
02305     if (VABS(xcentOlap) < VSMALL) xcentOlap = 0.0;
02306     if (VABS(ycentOlap) < VSMALL) ycentOlap = 0.0;
02307     if (VABS(zcentOlap) < VSMALL) zcentOlap = 0.0;
02308
02309     Vnm_print(0, "NOsh_setupMGPARA:  Olap part length = (%g, %g, %g)\n",
02310               xlenOlap, ylenOlap, zlenOlap);
02311     Vnm_print(0, "NOsh_setupMGPARA:  Olap part center displacement = (%g, %g, %g)\n",
02312               xcentOlap, ycentOlap, zcentOlap);
02313
02314
02315     /* Calculate the boundary flags:
02316         Flags are set to 1 when another processor is present along the boundary
02317         Flags are otherwise set to 0. */
02318
02319     if (ip == 0) mgparm->partDisjOwnSide[VAPBS_LEFT] = 0;
02320     else mgparm->partDisjOwnSide[VAPBS_LEFT] = 1;
02321     if (ip == (npx-1)) mgparm->partDisjOwnSide[VAPBS_RIGHT] = 0;
02322     else mgparm->partDisjOwnSide[VAPBS_RIGHT] = 1;
02323     if (jp == 0) mgparm->partDisjOwnSide[VAPBS_BACK] = 0;
02324     else mgparm->partDisjOwnSide[VAPBS_BACK] = 1;
02325     if (jp == (npy-1)) mgparm->partDisjOwnSide[VAPBS_FRONT] = 0;
02326     else mgparm->partDisjOwnSide[VAPBS_FRONT] = 1;
02327     if (kp == 0) mgparm->partDisjOwnSide[VAPBS_DOWN] = 0;
02328     else mgparm->partDisjOwnSide[VAPBS_DOWN] = 1;
02329     if (kp == (npz-1)) mgparm->partDisjOwnSide[VAPBS_UP] = 0;
02330     else mgparm->partDisjOwnSide[VAPBS_UP] = 1;
02331
02332     Vnm_print(0, "NOsh_setupMGPARA:  partDisjOwnSide[LEFT] = %d\n",
02333               mgparm->partDisjOwnSide[VAPBS_LEFT]);
02334     Vnm_print(0, "NOsh_setupMGPARA:  partDisjOwnSide[RIGHT] = %d\n",
02335               mgparm->partDisjOwnSide[VAPBS_RIGHT]);
02336     Vnm_print(0, "NOsh_setupMGPARA:  partDisjOwnSide[FRONT] = %d\n",
02337               mgparm->partDisjOwnSide[VAPBS_FRONT]);
02338     Vnm_print(0, "NOsh_setupMGPARA:  partDisjOwnSide[BACK] = %d\n",
```

```
02339                 mgparm->partDisjOwnSide[VAPBS_BACK]);
02340     Vnm_print(0, "NOsh_setupMGPARA:  partDisjOwnSide[UP] = %d\n",
02341                 mgparm->partDisjOwnSide[VAPBS_UP]);
02342     Vnm_print(0, "NOsh_setupMGPARA:  partDisjOwnSide[DOWN] = %d\n",
02343                 mgparm->partDisjOwnSide[VAPBS_DOWN]);
02344
02345     /* Set the mesh parameters */
02346     mgparm->fglen[0] = xlenOlap;
02347     mgparm->fglen[1] = ylenOlap;
02348     mgparm->fglen[2] = zlenOlap;
02349     mgparm->partDisjLength[0] = xlenDisj;
02350     mgparm->partDisjLength[1] = ylenDisj;
02351     mgparm->partDisjLength[2] = zlenDisj;
02352     mgparm->partDisjCenter[0] = mgparm->fcenter[0] + xcentDisj;
02353     mgparm->partDisjCenter[1] = mgparm->fcenter[1] + ycentDisj;
02354     mgparm->partDisjCenter[2] = mgparm->fcenter[2] + zcentDisj;
02355     mgparm->fcenter[0] += xcentOlap;
02356     mgparm->fcenter[1] += ycentOlap;
02357     mgparm->fcenter[2] += zcentOlap;
02358
02359     Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d):  Set up *relative* partition \
02360 centers...\n", __FILE__, __LINE__);
02361     Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d):  Absolute centers will be set \
02362 in  NOsh_setupMGAUTO\n", __FILE__, __LINE__);
02363     Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d):  partDisjCenter = %g %g %g\n",
02364                 __FILE__, __LINE__,
02365                 mgparm->partDisjCenter[0],
02366                 mgparm->partDisjCenter[1],
02367                 mgparm->partDisjCenter[2]);
02368     Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d):  ccenter = %g %g %g\n",
02369                 __FILE__, __LINE__,
02370                 mgparm->ccenter[0],
02371                 mgparm->ccenter[1],
02372                 mgparm->ccenter[2]);
02373     Vnm_print(0, "NOsh_setupCalcMGPARA (%s, %d):  fcenter = %g %g %g\n",
02374                 __FILE__, __LINE__,
02375                 mgparm->fcenter[0],
02376                 mgparm->fcenter[1],
02377                 mgparm->fcenter[2]);
02378
02379
02380     /* Setup the automatic focusing calculations associated with this processor */
02381     return NOsh_setupCalcMGAUTO(thee, elec);
02382
02383 }
02384
02385 VPUBLIC int NOsh_parseFEM(
02386                           NOsh *thee,
02387                           Vio *sock,
02388                           NOsh_calc *elec
02389                           ) {
02390
02391     char tok[VMAX_BUFSIZE];
02392     FEMparm *feparm = VNULL;
02393     PBEparm *pbeparm = VNULL;
02394     int rc;
02395     Vrc_Codes vrc;
02396
02397     /* Check the arguments */
02398     if (thee == VNULL) {
02399         Vnm_print(2, "NOsh_parseFEM:  Got NULL thee!\n");
02400         return 0;
02401     }
02402     if (sock == VNULL) {
02403         Vnm_print(2, "NOsh_parseFEM:  Got pointer to NULL socket!\n");
02404         return 0;
02405     }
02406     if (elec == VNULL) {
02407         Vnm_print(2, "NOsh_parseFEM:  Got pointer to NULL elec object!\n");
02408         return 0;
02409     }
02410     feparm = elec->femparm;
02411     if (feparm == VNULL) {
02412         Vnm_print(2, "NOsh_parseFEM:  Got pointer to NULL feparm object!\n");
02413         return 0;
02414     }
02415     pbeparm = elec->pbeparm;
02416     if (feparm == VNULL) {
02417         Vnm_print(2, "NOsh_parseFEM:  Got pointer to NULL pbeparm object!\n");
02418         return 0;
02419     }
```

```
02420
02421     Vnm_print(0, "NOsh_parseFEM: Parsing parameters for FEM calculation\n");
02422
02423     /* Start snarfing tokens from the input stream */
02424     rc = 1;
02425     while (Vio_scanf(sock, "%s", tok) == 1) {
02426
02427         Vnm_print(0, "NOsh_parseFEM:  Parsing %s...\n", tok);
02428
02429         /* See if it's an END token */
02430         if (Vstring_strcasecmp(tok, "end") == 0) {
02431             feparm->parsed = 1;
02432             pbeparm->parsed = 1;
02433             rc = 1;
02434             break;
02435         }
02436
02437         /* Pass the token through a series of parsers */
02438         rc = PBEparm_parseToken(pbeparm, tok, sock);
02439         if (rc == -1) {
02440             Vnm_print(0, "NOsh_parseFEM:  parsePBE error!\n");
02441             break;
02442         } else if (rc == 0) {
02443             /* Pass the token to the generic MG parser */
02444             vrc = FEMparm_parseToken(feparm, tok, sock);
02445             if (vrc == VRC_FAILURE) {
02446                 Vnm_print(0, "NOsh_parseFEM:  parseMG error!\n");
02447                 break;
02448             } else if (vrc == VRC_WARNING) {
02449                 /* We ran out of parsers! */
02450                 Vnm_print(2, "NOsh:  Unrecognized keyword: %s\n", tok);
02451                 break;
02452             }
02453         }
02454     }
02455
02456     /* Handle various errors arising in the token-snarfing loop -- these all
02457         * just result in simple returns right now */
02458     if (rc == -1) return 0;
02459     if (rc == 0) return 0;
02460
02461     /* Check the status of the parameter objects */
02462     if ((!FEMparm_check(feparm)) || (!PBEparm_check(pbeparm))) {
02463         Vnm_print(2, "NOsh:  FEM parameters not set correctly!\n");
02464         return 0;
02465     }
02466
02467     return 1;
02468
02469 }
02470
02471 VPRIVATE int NOsh_setupCalcFEMANUAL(
02472                                     NOsh *thee,
02473                                     NOsh_calc *elec
02474                                     ) {
02475
02476     FEMparm *feparm = VNULL;
02477     PBEparm *pbeparm = VNULL;
02478     NOsh_calc *calc = VNULL;
02479
02480     VASSERT(thee != VNULL);
02481     VASSERT(elec != VNULL);
02482     feparm = elec->femparm;
02483     VASSERT(feparm != VNULL);
02484     pbeparm = elec->pbeparm;
02485     VASSERT(pbeparm);
02486
02487     /* Check to see if he have any room left for this type of
02488      * calculation, if so: set the calculation type, update the number
02489      * of calculations of this type, and parse the rest of the section
02490      */
02491     if (thee->ncalc >= NOSH_MAXCALC) {
02492         Vnm_print(2, "NOsh:  Too many calculations in this run!\n");
02493         Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
02494                   NOSH_MAXCALC);
02495         return 0;
02496     }
02497     thee->calc[thee->ncalc] = NOsh_calc_ctor(NCT_FEM);
02498     calc = thee->calc[thee->ncalc];
02499     (thee->ncalc)++;
02500
```

```
02501     /* Copy over contents of ELEC */
02502     NOsh_calc_copy(calc, elec);
02503
02504
02505     return 1;
02506 }
02507
02508 VPUBLIC int NOsh_parseAPOL(
02509                             NOsh *thee,
02510                             Vio *sock,
02511                             NOsh_calc *elec
02512                             ) {
02513
02514     char tok[VMAX_BUFSIZE];
02515     APOLparm *apolparm = VNULL;
02516     int rc;
02517
02518     /* Check the arguments */
02519     if (thee == VNULL) {
02520         Vnm_print(2, "NOsh_parseAPOL:  Got NULL thee!\n");
02521         return 0;
02522     }
02523     if (sock == VNULL) {
02524         Vnm_print(2, "NOsh_parseAPOL:  Got pointer to NULL socket!\n");
02525         return 0;
02526     }
02527     if (elec == VNULL) {
02528         Vnm_print(2, "NOsh_parseAPOL:  Got pointer to NULL elec object!\n");
02529         return 0;
02530     }
02531     apolparm = elec->apolparm;
02532     if (apolparm == VNULL) {
02533         Vnm_print(2, "NOsh_parseAPOL:  Got pointer to NULL apolparm object!\n");
02534         return 0;
02535     }
02536
02537     Vnm_print(0, "NOsh_parseAPOL: Parsing parameters for APOL calculation\n");
02538
02539     /* Start snarfing tokens from the input stream */
02540     rc = 1;
02541     while (Vio_scanf(sock, "%s", tok) == 1) {
02542
02543         Vnm_print(0, "NOsh_parseAPOL:  Parsing %s...\n", tok);
02544         /* See if it's an END token */
02545         if (Vstring_strcasecmp(tok, "end") == 0) {
02546             apolparm->parsed = 1;
02547             rc = 1;
02548             break;
02549         }
02550
02551         /* Pass the token through a series of parsers */
02552         /* Pass the token to the generic non-polar parser */
02553         rc = APOLparm_parseToken(apolparm, tok, sock);
02554         if (rc == -1) {
02555             Vnm_print(0, "NOsh_parseFEM:  parseMG error!\n");
02556             break;
02557         } else if (rc == 0) {
02558             /* We ran out of parsers! */
02559             Vnm_print(2, "NOsh:  Unrecognized keyword: %s\n", tok);
02560             break;
02561         }
02562
02563     }
02564
02565     /* Handle various errors arising in the token-snarfing loop -- these all
02566         * just result in simple returns right now */
02567     if (rc == -1) return 0;
02568     if (rc == 0) return 0;
02569
02570     /* Check the status of the parameter objects */
02571     if (!APOLparm_check(apolparm)) {
02572         Vnm_print(2, "NOsh:  APOL parameters not set correctly!\n");
02573         return 0;
02574     }
02575
02576     return 1;
02577
02578 }
02579
02580
02581 VPRIVATE int NOsh_setupCalcAPOL(
```

```
02582                                 NOsh *thee,
02583                                 NOsh_calc *apol
02584                                 ) {
02585
02586     NOsh_calc *calc = VNULL;
02587
02588     VASSERT(thee != VNULL);
02589     VASSERT(apol != VNULL);
02590
02591     /* Check to see if he have any room left for this type of
02592         * calculation, if so: set the calculation type, update the number
02593         * of calculations of this type, and parse the rest of the section
02594         */
02595     if (thee->ncalc >= NOSH_MAXCALC) {
02596         Vnm_print(2, "NOsh:  Too many calculations in this run!\n");
02597         Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
02598                   NOSH_MAXCALC);
02599         return 0;
02600     }
02601     thee->calc[thee->ncalc] = NOsh_calc_ctor(NCT_APOL);
02602     calc = thee->calc[thee->ncalc];
02603     (thee->ncalc)++;
02604
02605     /* Copy over contents of APOL */
02606     NOsh_calc_copy(calc, apol);
02607
02608     return 1;
02609 }
02610
02611
02612 VPRIVATE int NOsh_setupCalcBEMMANUAL(
02613                                      NOsh *thee,
02614                                      NOsh_calc *elec
02615                                      ) {
02616
02617     BEMparm *bemparm = VNULL;
02618     PBEparm *pbeparm = VNULL;
02619     NOsh_calc *calc = VNULL;
02620
02621     if (thee == VNULL) {
02622         Vnm_print(2, "NOsh_setupCalcBEMMANUAL:  Got NULL thee!\n");
02623         return 0;
02624     }
02625     if (elec == VNULL) {
02626         Vnm_print(2, "NOsh_setupCalcBEMMANUAL:  Got NULL calc!\n");
02627         return 0;
02628     }
02629     bemparm = elec->bemparm;
02630     if (bemparm == VNULL) {
02631         Vnm_print(2, "NOsh_setupCalcBEMMANUAL:  Got NULL bemparm -- was this calculation \
02632 set up?\n");
02633         return 0;
02634     }
02635     pbeparm = elec->pbeparm;
02636     if (pbeparm == VNULL) {
02637         Vnm_print(2, "NOsh_setupCalcBEMMANUAL:  Got NULL pbeparm -- was this calculation \
02638 set up?\n");
02639         return 0;
02640     }
02641
02642     /* Set up missing BEM parameters */
02643     if (bemparm->settree_order == 0) {
02644         bemparm->tree_order=1;
02645     }
02646
02647     if (bemparm->settree_n0 == 0) {
02648         bemparm->tree_n0=500;
02649     }
02650
02651     if (bemparm->setmac == 0) {
02652         bemparm->mac=0.8;
02653     }
02654
02655     /* Check to see if he have any room left for this type of calculation, if
02656         so: set the calculation type, update the number of calculations of this type,
02657         and parse the rest of the section */
02658     if (thee->ncalc >= NOSH_MAXCALC) {
02659         Vnm_print(2, "NOsh:  Too many calculations in this run!\n");
02660         Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
02661                   NOSH_MAXCALC);
02662         return 0;
```

```
02663     }
02664
02665     /* Get the next calculation object and increment the number of calculations */
02666     thee->calc[thee->ncalc] = NOsh_calc_ctor(NCT_BEM);
02667     calc = thee->calc[thee->ncalc];
02668     (thee->ncalc)++;
02669
02670     /* Copy over contents of ELEC */
02671     NOsh_calc_copy(calc, elec);
02672
02673
02674     return 1;
02675 }
02676
02677 VPRIVATE int NOsh_setupCalcGEOFLOWMANUAL(
02678                                     NOsh *thee,
02679                                     NOsh_calc *elec
02680                                     ) {
02681
02682     GEOFLOWparm *parm = VNULL;
02683     APOLparm *apolparm = VNULL;
02684     PBEparm *pbeparm = VNULL;
02685     NOsh_calc *calc = VNULL;
02686
02687     if (thee == VNULL) {
02688         Vnm_print(2, "NOsh_setupCalcGEOFLOWMANUAL:  Got NULL thee!\n");
02689         return 0;
02690     }
02691     if (elec == VNULL) {
02692         Vnm_print(2, "NOsh_setupCalcGEOFLOWMANUAL:  Got NULL calc!\n");
02693         return 0;
02694     }
02695     parm = elec->geoflowparm;
02696     if (parm == VNULL) {
02697         Vnm_print(2, "NOsh_setupCalcGEOFLOWMANUAL:  Got NULL geoflowparm -- was this calculation \
02698 set up?\n");
02699         return 0;
02700     }
02701     apolparm = elec->apolparm;
02702     if (parm == VNULL) {
02703         Vnm_print(2, "NOsh_setupCalcGEOFLOWMANUAL:  Got NULL apolparm -- was this calculation \
02704 set up?\n");
02705         return 0;
02706     }
02707     pbeparm = elec->pbeparm;
02708     if (pbeparm == VNULL) {
02709         Vnm_print(2, "NOsh_setupCalcGEOFLOWMANUAL:  Got NULL pbeparm -- was this calculation \
02710 set up?\n");
02711         return 0;
02712     }
02713
02714     /* Check to see if he have any room left for this type of calculation, if
02715         so: set the calculation type, update the number of calculations of this type,
02716         and parse the rest of the section */
02717     if (thee->ncalc >= NOSH_MAXCALC) {
02718         Vnm_print(2, "NOsh:  Too many calculations in this run!\n");
02719         Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
02720                   NOSH_MAXCALC);
02721         return 0;
02722     }
02723
02724     /* Get the next calculation object and increment the number of calculations */
02725     thee->calc[thee->ncalc] = NOsh_calc_ctor(NCT_GEOFLOW);
02726     calc = thee->calc[thee->ncalc];
02727     (thee->ncalc)++;
02728
02729     /* Copy over contents of ELEC */
02730     NOsh_calc_copy(calc, elec);
02731
02732     return 1;
02733 }
02734
02735 VPRIVATE int NOsh_setupCalcPBAMAUTO(
02736                                     NOsh *thee,
02737                                     NOsh_calc *elec
02738                                     ) {
02739
02740     PBAMparm *parm = VNULL;
02741     PBEparm *pbeparm = VNULL;
02742     NOsh_calc *calc = VNULL;
02743
```

```
02744     if (thee == VNULL) {
02745         Vnm_print(2, "NOsh_setupCalcPBAMAUTO:  Got NULL thee!\n");
02746         return 0;
02747     }
02748     if (elec == VNULL) {
02749         Vnm_print(2, "NOsh_setupCalcPBAMAUTO:  Got NULL calc!\n");
02750         return 0;
02751     }
02752     parm = elec->pbamparm;
02753     if (parm == VNULL) {
02754         Vnm_print(2, "NOsh_setupCalcPBAMAUTO:  Got NULL pbamparm -- was this calculation \
02755 set up?\n");
02756         return 0;
02757     }
02758     pbeparm = elec->pbeparm;
02759     if (pbeparm == VNULL) {
02760         Vnm_print(2, "NOsh_setupCalcPBAMAUTO:  Got NULL pbeparm -- was this calculation \
02761 set up?\n");
02762         return 0;
02763     }
02764 
02765     /* Check to see if he have any room left for this type of calculation, if
02766         so: set the calculation type, update the number of calculations of this type,
02767         and parse the rest of the section */
02768     if (thee->ncalc >= NOSH_MAXCALC) {
02769         Vnm_print(2, "NOsh:  Too many calculations in this run!\n");
02770         Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
02771                   NOSH_MAXCALC);
02772         return 0;
02773     }
02774 
02775     /* Get the next calculation object and increment the number of calculations */
02776     thee->calc[thee->ncalc] = NOsh_calc_ctor(NCT_PBAM);
02777     calc = thee->calc[thee->ncalc];
02778     (thee->ncalc)++;
02779 
02780     /* Copy over contents of ELEC */
02781     NOsh_calc_copy(calc, elec);
02782 
02783     return 1;
02784 }
02785 
02786 VPRIVATE int NOsh_setupCalcPBSAMAUTO(
02787                                      NOsh *thee,
02788                                      NOsh_calc *elec
02789                                      ) {
02790 
02791     PBAMparm *parm = VNULL;
02792     PBSAMparm *samparm = VNULL;
02793     PBEparm *pbeparm = VNULL;
02794     NOsh_calc *calc = VNULL;
02795 
02796     if (thee == VNULL) {
02797         Vnm_print(2, "NOsh_setupCalcPBSAMAUTO:  Got NULL thee!\n");
02798         return 0;
02799     }
02800     if (elec == VNULL) {
02801         Vnm_print(2, "NOsh_setupCalcPBSAMAUTO:  Got NULL calc!\n");
02802         return 0;
02803     }
02804     parm = elec->pbamparm;
02805     if (parm == VNULL) {
02806         Vnm_print(2, "NOsh_setupCalcPBSAMAUTO:  Got NULL pbamparm -- was this calculation \
02807 set up?\n");
02808         return 0;
02809     }
02810     samparm = elec->pbsamparm;
02811     if (samparm == VNULL) {
02812         Vnm_print(2, "NOsh_setupCalcPBSAMAUTO:  Got NULL pbsamparm -- was this calculation \
02813 set up?\n");
02814         return 0;
02815     }
02816     pbeparm = elec->pbeparm;
02817     if (pbeparm == VNULL) {
02818         Vnm_print(2, "NOsh_setupCalcPBAMAUTO:  Got NULL pbeparm -- was this calculation \
02819 set up?\n");
02820         return 0;
02821     }
02822 
02823     /* Check to see if he have any room left for this type of calculation, if
02824         so: set the calculation type, update the number of calculations of this type,
```

```
02825         and parse the rest of the section */
02826     if (thee->ncalc >= NOSH_MAXCALC) {
02827         Vnm_print(2, "NOsh:  Too many calculations in this run!\n");
02828         Vnm_print(2, "NOsh:  Current max is %d; ignoring this calculation\n",
02829                   NOSH_MAXCALC);
02830         return 0;
02831     }
02832
02833     /* Get the next calculation object and increment the number of calculations */
02834     thee->calc[thee->ncalc] = NOsh_calc_ctor(NCT_PBSAM);
02835     calc = thee->calc[thee->ncalc];
02836     (thee->ncalc)++;
02837
02838     /* Copy over contents of ELEC */
02839     NOsh_calc_copy(calc, elec);
02840
02841     return 1;
02842 }
02843
02844
02845 VPUBLIC int NOsh_parseBEM(
02846                           NOsh *thee,
02847                           Vio *sock,
02848                           NOsh_calc *elec
02849                           ) {
02850
02851     char tok[VMAX_BUFSIZE];
02852     BEMparm *bemparm = VNULL;
02853     PBEparm *pbeparm = VNULL;
02854     int rc;
02855
02856     /* Check the arguments */
02857     if (thee == VNULL) {
02858         Vnm_print(2, "NOsh:  Got NULL thee!\n");
02859         return 0;
02860     }
02861     if (sock == VNULL) {
02862         Vnm_print(2, "NOsh:  Got pointer to NULL socket!\n");
02863         return 0;
02864     }
02865     if (elec == VNULL) {
02866         Vnm_print(2, "NOsh:  Got pointer to NULL elec object!\n");
02867         return 0;
02868     }
02869     bemparm = elec->bemparm;
02870     if (bemparm == VNULL) {
02871         Vnm_print(2, "NOsh:  Got pointer to NULL bemparm object!\n");
02872         return 0;
02873     }
02874     pbeparm = elec->pbeparm;
02875     if (pbeparm == VNULL) {
02876         Vnm_print(2, "NOsh:  Got pointer to NULL pbeparm object!\n");
02877         return 0;
02878     }
02879
02880     Vnm_print(0, "NOsh_parseBEM: Parsing parameters for BEM calculation\n");
02881
02882
02883     /* Start snarfing tokens from the input stream */
02884     rc = 1;
02885     while (Vio_scanf(sock, "%s", tok) == 1) {
02886
02887         Vnm_print(0, "NOsh_parseBEM:  Parsing %s...\n", tok);
02888
02889         /* See if it's an END token */
02890         if (Vstring_strcasecmp(tok, "end") == 0) {
02891             bemparm->parsed = 1;
02892             pbeparm->parsed = 1;
02893             rc = 1;
02894             break;
02895         }
02896
02897         /* Pass the token through a series of parsers */
02898         rc = PBEparm_parseToken(pbeparm, tok, sock);
02899         if (rc == -1) {
02900             Vnm_print(0, "NOsh_parseBEM:  parsePBE error!\n");
02901             break;
02902         } else if (rc == 0) {
02903             /* Pass the token to the generic BEM parser */
02904             rc = BEMparm_parseToken(bemparm, tok, sock);
02905             if (rc == -1) {
```

```
02906                 Vnm_print(0, "NOsh_parseBEM:  parseBEM error!\n");
02907                 break;
02908             } else if (rc == 0) {
02909                 /* We ran out of parsers! */
02910                 Vnm_print(2, "NOsh:  Unrecognized keyword: %s\n", tok);
02911                 break;
02912             }
02913         }
02914     }
02915
02916     pbeparm->setsrfm=1;     // unneeded srfm
02917     pbeparm->setpbetype=1;  // unneeded pbe type
02918     pbeparm->setbcfl=1;     // unneeded bcfl
02919
02920     /* Handle various errors arising in the token-snarfing loop -- these all
02921        just result in simple returns right now */
02922     if (rc == -1) return 0;
02923     if (rc == 0) return 0;
02924
02925     /* Check the status of the parameter objects */
02926     if ((BEMparm_check(bemparm) == VRC_FAILURE) || (!PBEparm_check(pbeparm))) {
02927         Vnm_print(2, "NOsh:  BEM parameters not set correctly!\n");
02928         return 0;
02929     }
02930
02931     return 1;
02932 }
02933
02934 VPUBLIC int NOsh_parseGEOFLOW(
02935                           NOsh *thee,
02936                           Vio *sock,
02937                           NOsh_calc *elec
02938                           ) {
02939
02940     char tok[VMAX_BUFSIZE];
02941     GEOFLOWparm *parm = VNULL;
02942     APOLparm *apolparm = VNULL;
02943     PBEparm *pbeparm = VNULL;
02944     int rc;
02945
02946     /* Check the arguments */
02947     if (thee == VNULL) {
02948         Vnm_print(2, "NOsh:  Got NULL thee!\n");
02949         return 0;
02950     }
02951     if (sock == VNULL) {
02952         Vnm_print(2, "NOsh:  Got pointer to NULL socket!\n");
02953         return 0;
02954     }
02955     if (elec == VNULL) {
02956         Vnm_print(2, "NOsh:  Got pointer to NULL elec object!\n");
02957         return 0;
02958     }
02959     parm = elec->geoflowparm;
02960     if (parm == VNULL) {
02961         Vnm_print(2, "NOsh:  Got pointer to NULL geoflowparm object!\n");
02962         return 0;
02963     }
02964     apolparm = elec->apolparm;
02965     if (parm == VNULL) {
02966         Vnm_print(2, "NOsh:  Got pointer to NULL apolparm object!\n");
02967         return 0;
02968     }
02969     pbeparm = elec->pbeparm;
02970     if (pbeparm == VNULL) {
02971         Vnm_print(2, "NOsh:  Got pointer to NULL pbeparm object!\n");
02972         return 0;
02973     }
02974
02975     Vnm_print(0, "NOsh_parseGEOFLOW: Parsing parameters for GEOFLOW calculation\n");
02976
02977
02978     /* Start snarfing tokens from the input stream */
02979     rc = 1;
02980     while (Vio_scanf(sock, "%s", tok) == 1) {
02981
02982         Vnm_print(0, "NOsh_parseGEOFLOW:  Parsing %s...\n", tok);
02983
02984         /* See if it's an END token */
02985         if (Vstring_strcasecmp(tok, "end") == 0) {
02986             parm->parsed = 1;
```

```
02987             pbeparm->parsed = 1;
02988             apolparm->parsed = 1;
02989             rc = 1;
02990             break;
02991         }
02992
02993         if (Vstring_strcasecmp(tok, "ion") == 0) {
02994             Vnm_print(2, "parseGEOFLOW: WARNING! ion not implemented for geometric flow!\n");
02995         }
02996
02997         /* Pass the token through a series of parsers */
02998         rc = PBEparm_parseToken(pbeparm, tok, sock);
02999         if (rc == -1) {
03000             Vnm_print(0, "NOsh_parseGEOFLOW:  parsePBE error!\n");
03001             break;
03002         } else if (rc == 0) {
03003             /* Pass the token to the generic GEOFLOW parser */
03004             rc = APOLparm_parseToken(apolparm, tok, sock);
03005             if (rc == -1) {
03006                 Vnm_print(0, "NOsh_parseAPOL:  parseAPOL error!\n");
03007                 break;
03008             } else if (rc == 0) {
03009                 rc = GEOFLOWparm_parseToken(parm, tok, sock);
03010                 if (rc == -1) {
03011                     Vnm_print(0, "NOsh_parseGEOFLOW:  parseGEOFLOW error!\n");
03012                     break;
03013                 } else if (rc == 0) {
03014                     /* We ran out of parsers! */
03015                     Vnm_print(2, "NOsh:  Unrecognized keyword: %s\n", tok);
03016                     break;
03017                 }
03018             }
03019         }
03020     }
03021
03022     pbeparm->setsrfm=1;
03023     pbeparm->srad=0.0;
03024     pbeparm->setsrad=1;
03025     pbeparm->settemp=1;
03026
03027     /* Handle various errors arising in the token-snarfing loop -- these all
03028         just result in simple returns right now */
03029     if (rc == -1) return 0;
03030     if (rc == 0) return 0;
03031
03032     /* Check the status of the parameter objects */
03033     if ((GEOFLOWparm_check(parm) == VRC_FAILURE) || (!PBEparm_check(pbeparm))) {
03034         Vnm_print(2, "NOsh:  GEOFLOW parameters not set correctly!\n");
03035         return 0;
03036     }
03037     /*currently the only bc handle by geoflow is mdh so we check here if mdh was read*/
03038  if(pbeparm->bcfl != BCFL_MDH){
03039   Vnm_print(2, "NOsh_parseGEOFLOW: Geoflow currently only supports mdh boundary conditions!\n");
03040   Vnm_print(2, "NOsh_parseGEOFLOW: please change bcfl keyword.\n");
03041   return 0;
03042  }
03043
03044     return 1;
03045 }
03046
03047
03048 VPUBLIC int NOsh_parsePBAM(
03049                           NOsh *thee,
03050                           Vio *sock,
03051                           NOsh_calc *elec
03052                           ) {
03053
03054     char tok[VMAX_BUFSIZE];
03055     PBAMparm *parm = VNULL;
03056     PBEparm *pbeparm = VNULL;
03057     int rc;
03058
03059     /* Check the arguments */
03060     if (thee == VNULL) {
03061         Vnm_print(2, "NOsh:  Got NULL thee!\n");
03062         return 0;
03063     }
03064     if (sock == VNULL) {
03065         Vnm_print(2, "NOsh:  Got pointer to NULL socket!\n");
03066         return 0;
03067     }
```

```
03068     if (elec == VNULL) {
03069         Vnm_print(2, "NOsh:  Got pointer to NULL elec object!\n");
03070         return 0;
03071     }
03072     parm = elec->pbamparm;
03073     if (parm == VNULL) {
03074         Vnm_print(2, "NOsh:  Got pointer to NULL pbam object!\n");
03075         return 0;
03076     }
03077     pbeparm = elec->pbeparm;
03078     if (pbeparm == VNULL) {
03079         Vnm_print(2, "NOsh:  Got pointer to NULL pbeparm object!\n");
03080         return 0;
03081     }
03082     Vnm_print(0, "NOsh_parsePBAM: Parsing parameters for PBAM calculation\n");
03083
03084     /* Start snarfing tokens from the input stream */
03085     rc = 1;
03086     while (Vio_scanf(sock, "%s", tok) == 1) {
03087
03088         Vnm_print(0, "NOsh_parsePBAM:  Parsing %s...\n", tok);
03089
03090         /* See if it's an END token */
03091         if (Vstring_strcasecmp(tok, "end") == 0) {
03092             parm->parsed = 1;
03093             pbeparm->parsed = 1;
03094             rc = 1;
03095             break;
03096         }
03097
03098         if (Vstring_strcasecmp(tok, "ion") == 0) {
03099             Vnm_print(2, "parsePBAM: WARNING! PBAM only uses the conc parameter of ion!\n");
03100         }
03101
03102         /* Pass the token through a series of parsers */
03103         rc = PBEparm_parseToken(pbeparm, tok, sock);
03104         if (rc == -1) {
03105             Vnm_print(0, "NOsh_parsePBAM:  parsePBE error!\n");
03106             break;
03107         } else if (rc == 0) {
03108              rc = PBAMparm_parseToken(parm, tok, sock);
03109              if (rc == -1) {
03110                  Vnm_print(0, "NOsh_parsePBAM:  parsePBAM error!\n");
03111                  break;
03112              } else if (rc == 0) {
03113                  /* We ran out of parsers! */
03114                  Vnm_print(2, "NOsh:  Unrecognized keyword: %s\n", tok);
03115                  break;
03116             }
03117         }
03118     }
03119
03120
03121
03122     pbeparm->setsrfm=1;
03123     pbeparm->setsrad=1;
03124     pbeparm->settemp=1; // do need temp, but have default, incase
03125     pbeparm->setmolid=1; // for unneeded mol flag
03126     pbeparm->setpbetype=1; // unneeded pbe type
03127     pbeparm->setbcfl=1;  // unneeded bcfl
03128     pbeparm->setsdens=1;
03129
03130     //This is a hacky fix at best for issue 501. This is so we don't need to change PBAM's
03131     //external code.
03132     if(pbeparm->setnion){
03133      parm->salt = pbeparm->ionc[pbeparm->nion-1];
03134      parm->setsalt = 1;
03135     }
03136
03137  //This is also a hacky fix for issue 488
03138  if (pbeparm->writefmt[pbeparm->numwrite - 1] == VDF_DX) {
03139   strncpy(parm->dxname, pbeparm->writestem[pbeparm->numwrite -1], CHR_MAXLEN);
03140   parm->setdxname = 1;
03141  }
03142  else {
03143   Vnm_print(2, "NOsh: PBAM only prints in dx format!\n");
03144   return 0;
03145  }
03146
03147  //Another hacky fix for issue 482
03148  if (pbeparm->pbam_3dmapflag == 1) {
```

```
03149   strcpy(parm->map3dname, pbeparm->pbam_3dmapstem);
03150   parm->set3dmap = 1;
03151  }
03152
03153     /* Handle various errors arising in the token-snarfing loop -- these all
03154         just result in simple returns right now */
03155     if (rc == -1) return 0;
03156     if (rc == 0) return 0;
03157
03158     /* Check the status of the parameter objects */
03159     if ((PBAMparm_check(parm) == VRC_FAILURE) || (!PBEparm_check(pbeparm))) {
03160         Vnm_print(2, "NOsh:  PBAM parameters not set correctly!\n");
03161         return 0;
03162     }
03163     return 1;
03164 }
03165
03166 VPUBLIC int NOsh_parsePBSAM(
03167                             NOsh *thee,
03168                             Vio *sock,
03169                             NOsh_calc *elec
03170                             ) {
03171
03172     char tok[VMAX_BUFSIZE];
03173     PBAMparm *parm = VNULL;
03174     PBSAMparm *samparm = VNULL;
03175     PBEparm *pbeparm = VNULL;
03176     int rc;
03177
03178     /* Check the arguments */
03179     if (thee == VNULL) {
03180         Vnm_print(2, "NOsh:  Got NULL thee!\n");
03181         return 0;
03182     }
03183     if (sock == VNULL) {
03184         Vnm_print(2, "NOsh:  Got pointer to NULL socket!\n");
03185         return 0;
03186     }
03187     if (elec == VNULL) {
03188         Vnm_print(2, "NOsh:  Got pointer to NULL elec object!\n");
03189         return 0;
03190     }
03191     parm = elec->pbamparm;
03192     if (parm == VNULL) {
03193         Vnm_print(2, "NOsh:  Got pointer to NULL pbam object!\n");
03194         return 0;
03195     }
03196     samparm = elec->pbsamparm;
03197     if (samparm == VNULL) {
03198         Vnm_print(2, "NOsh:  Got pointer to NULL pbsam object!\n");
03199         return 0;
03200     }
03201     pbeparm = elec->pbeparm;
03202     if (pbeparm == VNULL) {
03203         Vnm_print(2, "NOsh:  Got pointer to NULL pbeparm object!\n");
03204         return 0;
03205     }
03206     Vnm_print(0, "NOsh_parsePBSAM: Parsing parameters for PBSAM calculation\n");
03207
03208     /* Start snarfing tokens from the input stream */
03209     rc = 1;
03210     while (Vio_scanf(sock, "%s", tok) == 1) {
03211
03212         Vnm_print(0, "NOsh_parsePBSAM:  Parsing %s...\n", tok);
03213
03214         /* See if it's an END token */
03215         if (Vstring_strcasecmp(tok, "end") == 0) {
03216             parm->parsed = 1;
03217             samparm->parsed = 1;
03218             pbeparm->parsed = 1;
03219             rc = 1;
03220             break;
03221         }
03222
03223         if (Vstring_strcasecmp(tok, "ion") == 0) {
03224             Vnm_print(2, "parsePBSAM: WARNING! PBAM only uses the conc parameter of ion!\n");
03225         }
03226
03227         /* Pass the token through a series of parsers */
03228         rc = PBEparm_parseToken(pbeparm, tok, sock);
03229         if (rc == -1) {
```

```
03230             Vnm_print(0, "NOsh_parsePBSAM:  parsePBE error!\n");
03231             break;
03232         } else if (rc == 0) {
03233              rc = PBAMparm_parseToken(parm, tok, sock);
03234              if (rc == -1) {
03235                  Vnm_print(0, "NOsh_parsePBSAM:  parsePBAM error!\n");
03236                  break;
03237              } else if ( rc == 0 ) {
03238                   rc = PBSAMparm_parseToken(samparm, tok, sock);
03239                   if (rc == -1) {
03240                       Vnm_print(0, "NOsh_parsePBSAM:  parsePBSAM error!\n");
03241                       break;
03242                   } else if (rc == 0) {
03243                       /* We ran out of parsers! */
03244                       Vnm_print(2, "NOsh:  Unrecognized keyword: %s\n", tok);
03245                       break;
03246                   }
03247              }
03248         }
03249     }
03250
03251     pbeparm->setsrfm=1;
03252     pbeparm->setsrad=1;
03253     pbeparm->settemp=1; // do need temp, but have default, incase
03254     pbeparm->setmolid=1; // for unneeded mol flag
03255     pbeparm->setpbetype=1; // unneeded pbe type
03256     pbeparm->setbcfl=1;  // unneeded bcfl
03257     pbeparm->setsdens=1;
03258
03259     //This is a hacky fix at best for issue 501. This is so we don't need to change PBAM's
03260     //external code.
03261   if(pbeparm->setnion){
03262    parm->salt = pbeparm->ionc[pbeparm->nion-1];
03263    parm->setsalt = 1;
03264   }
03265
03266   //This is also a hacky fix for issue 488
03267   if (pbeparm->writefmt[pbeparm->numwrite - 1] == VDF_DX) {
03268    strncpy(parm->dxname, pbeparm->writestem[pbeparm->numwrite - 1], CHR_MAXLEN);
03269    parm->setdxname = 1;
03270   }
03271   else {
03272    Vnm_print(2, "NOsh: PBSAM only prints in dx format!\n");
03273    return 0;
03274   }
03275
03276
03277     /* Handle various errors arising in the token-snarfing loop -- these all
03278         just result in simple returns right now */
03279     if (rc == -1) return 0;
03280     if (rc == 0) return 0;
03281
03282     /* Check the status of the parameter objects */
03283     if ((PBSAMparm_check(samparm)==VRC_FAILURE) ||
03284          (PBAMparm_check(parm) == VRC_FAILURE) ||
03285          (!PBEparm_check(pbeparm))) {
03286         Vnm_print(2, "NOsh:  PBSAM parameters not set correctly!\n");
03287         return 0;
03288     }
03289     return 1;
03290 }
```

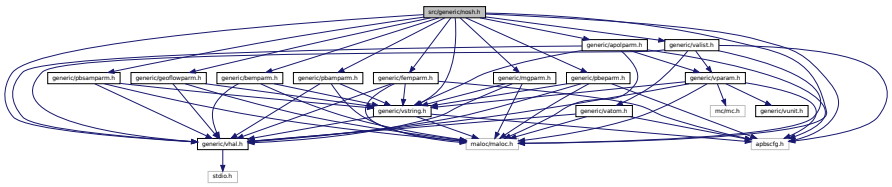
9.33 src/generic/nosh.h File Reference

Contains declarations for class NOsh.

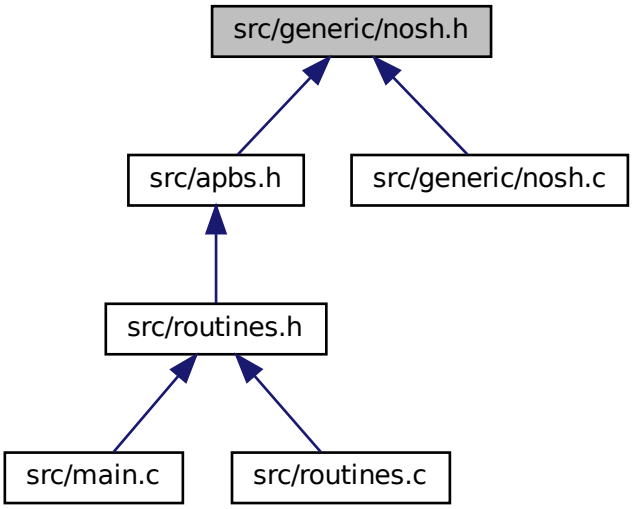
```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/vstring.h"
#include "generic/pbeparm.h"
#include "generic/mgparm.h"
#include "generic/apolparm.h"
#include "generic/femparm.h"
```

```
#include "generic/valist.h"
#include "generic/bemparm.h"
#include "generic/geoflowparm.h"
#include "generic/pbamparm.h"
#include "generic/pbsamparm.h"
```

Include dependency graph for nosh.h:



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



Data Structures

- struct sNOsh_calc

  Calculation class for use when parsing fixed format input files.
- struct sNOsh

  Class for parsing fixed format input files.

Macros

- #define NOSH_MAXMOL 20

  Maximum number of molecules in a run.

- #define NOSH_MAXCALC 20

  Maximum number of calculations in a run.
- #define NOSH_MAXPRINT 20

  Maximum number of PRINT statements in a run.
- #define NOSH_MAXPOP 20

  Maximum number of operations in a PRINT statement.

Typedefs

- typedef enum eNOsh_MolFormat NOsh_MolFormat

  Declare NOsh_MolFormat type.
- typedef enum eNOsh_CalcType NOsh_CalcType

  Declare NOsh_CalcType type.
- typedef enum eNOsh_ParmFormat NOsh_ParmFormat

  Declare NOsh_ParmFormat type.
- typedef enum eNOsh_PrintType NOsh_PrintType

  Declare NOsh_PrintType type.
- typedef struct sNOsh_calc NOsh_calc

  Declaration of the NOsh_calc class as the NOsh_calc structure.
- typedef struct sNOsh NOsh

  Declaration of the NOsh class as the NOsh structure.

Enumerations

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

  Molecule file format types.
- enum eNOsh_CalcType {
  NCT_MG =0 , NCT_FEM =1 , NCT_APOL =2 , NCT_BEM =3 ,
  NCT_GEOFLOW =4 , NCT_PBAM =5 , NCT_PBSAM =6 }

  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.

9.33.1 Detailed Description

Contains declarations for class NOsh.

Version

Id

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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.

9.34 nosh.h

```
00001
```

```
00062 #ifndef _NOSH_H_
00063 #define _NOSH_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "maloc/maloc.h"
00068
00069 #include "generic/vhal.h"
00070 #include "generic/vstring.h"
00071 #include "generic/pbeparm.h"
00072 #include "generic/mgparm.h"
00073 #include "generic/apolparm.h"
00074 #include "generic/femparm.h"
00075 #include "generic/valist.h"
00076 #include "generic/bemparm.h"
00077 #include "generic/geoflowparm.h"
00078 #include "generic/pbamparm.h"
00079 #include "generic/pbsamparm.h" //path might change
00080
00083 #define NOSH_MAXMOL 20
00084
00087 #define NOSH_MAXCALC 20
00088
00091 #define NOSH_MAXPRINT 20
00092
00095 #define NOSH_MAXPOP 20
00096
00101 enum eNOsh_MolFormat {
00102     NMF_PQR=0,
00103     NMF_PDB=1,
00104     NMF_XML=2
00105 };
00106
00111 typedef enum eNOsh_MolFormat NOsh_MolFormat;
00112
00117 enum eNOsh_CalcType {
00118     NCT_MG=0,
00119     NCT_FEM=1,
00120     NCT_APOL=2,
00121     NCT_BEM=3,
00122     NCT_GEOFLOW=4,
00123     NCT_PBAM=5,
00124     NCT_PBSAM=6
00125 };
00126
00131 typedef enum eNOsh_CalcType NOsh_CalcType;
00132
00137 enum eNOsh_ParmFormat {
00138     NPF_FLAT=0,
00139     NPF_XML=1
00140 };
00141
00146 typedef enum eNOsh_ParmFormat NOsh_ParmFormat;
00147
00152 enum eNOsh_PrintType {
00153     NPT_ENERGY=0,
00154     NPT_FORCE=1,
00155     NPT_ELECENERGY,
00156     NPT_ELECFORCE,
00157     NPT_APOLENERGY,
00158     NPT_APOLFORCE
00159 };
00160
00165 typedef enum eNOsh_PrintType NOsh_PrintType;
00166
00172 struct sNOsh_calc {
00173     MGparm *mgparm;
00174     FEMparm *femparm;
00175     BEMparm *bemparm;
00176     GEOFLOWparm *geoflowparm;
00177     PBAMparm *pbamparm;
00178     PBSAMparm *pbsamparm;
00179     PBEparm *pbeparm;
00180     APOLparm *apolparm;
00181     NOsh_CalcType calctype;
00182 };
00183
00188 typedef struct sNOsh_calc NOsh_calc;
00189
00195 struct sNOsh {
00196
```

```
00197     NOsh_calc *calc[NOSH_MAXCALC];
00200     int ncalc;
00202     NOsh_calc *elec[NOSH_MAXCALC];
00205     int nelec;
00208     NOsh_calc *apol[NOSH_MAXCALC];
00211     int napol;
00214     int ispara;
00215     int proc_rank;
00216     int proc_size;
00217     int bogus;
00221     int elec2calc[NOSH_MAXCALC];
00229     int apol2calc[NOSH_MAXCALC];
00231     int nmol;
00232     char molpath[NOSH_MAXMOL][VMAX_ARGLEN];
00233     NOsh_MolFormat molfmt[NOSH_MAXMOL];
00234     Valist *alist[NOSH_MAXMOL];
00236     int gotparm;
00237     char parmpath[VMAX_ARGLEN];
00238     NOsh_ParmFormat parmfmt;
00239     int ndiel;
00240     char dielXpath[NOSH_MAXMOL][VMAX_ARGLEN];
00242     char dielYpath[NOSH_MAXMOL][VMAX_ARGLEN];
00244     char dielZpath[NOSH_MAXMOL][VMAX_ARGLEN];
00246     Vdata_Format dielfmt[NOSH_MAXMOL];
00247     int nkappa;
00248     char kappapath[NOSH_MAXMOL][VMAX_ARGLEN];
00249     Vdata_Format kappafmt[NOSH_MAXMOL];
00250     int npot;
00251     char potpath[NOSH_MAXMOL][VMAX_ARGLEN];
00252     Vdata_Format potfmt[NOSH_MAXMOL];
00253     int ncharge;
00254     char chargepath[NOSH_MAXMOL][VMAX_ARGLEN];
00255     Vdata_Format chargefmt[NOSH_MAXMOL];
00256     int nmesh;
00257     char meshpath[NOSH_MAXMOL][VMAX_ARGLEN];
00258     Vdata_Format meshfmt[NOSH_MAXMOL];
00259     int nprint;
00260     NOsh_PrintType printwhat[NOSH_MAXPRINT];
00262     int printnarg[NOSH_MAXPRINT];
00263     int printcalc[NOSH_MAXPRINT][NOSH_MAXPOP];
00264     int printop[NOSH_MAXPRINT][NOSH_MAXPOP];
00266     int parsed;
00267     char elecname[NOSH_MAXCALC][VMAX_ARGLEN];
00269     char apolname[NOSH_MAXCALC][VMAX_ARGLEN];
00271 };
00272
00277 typedef struct sNOsh NOsh;
00278
00279 /* ///////////////////////////////////////////////////////////////////////////
00280    // Class NOsh: Inlineable methods (mcsh.c)
00282 #if !defined(VINLINE_NOSH)
00290 VEXTERNC char* NOsh_getMolpath(NOsh *thee, int imol);
00291
00299 VEXTERNC char* NOsh_getDielXpath(NOsh *thee, int imap);
00300
00308 VEXTERNC char* NOsh_getDielYpath(NOsh *thee, int imap);
00309
00317 VEXTERNC char* NOsh_getDielZpath(NOsh *thee, int imap);
00318
00326 VEXTERNC char* NOsh_getKappapath(NOsh *thee, int imap);
00327
00335 VEXTERNC char* NOsh_getPotpath(NOsh *thee, int imap);
00336
00344 VEXTERNC char* NOsh_getChargepath(NOsh *thee, int imap);
00345
00353 VEXTERNC NOsh_calc* NOsh_getCalc(NOsh *thee, int icalc);
00354
00362 VEXTERNC int NOsh_getDielfmt(NOsh *thee, int imap);
00363
00371 VEXTERNC int NOsh_getKappafmt(NOsh *thee, int imap);
00372
00380 VEXTERNC int NOsh_getPotfmt(NOsh *thee, int imap);
00381
00389 VEXTERNC int NOsh_getChargefmt(NOsh *thee, int imap);
00390
00391 #else
00392
00393 #   define NOsh_getMolpath(thee, imol) ((thee)->molpath[(imol)])
00394 #   define NOsh_getDielXpath(thee, imol) ((thee)->dielXpath[(imol)])
00395 #   define NOsh_getDielYpath(thee, imol) ((thee)->dielYpath[(imol)])
00396 #   define NOsh_getDielZpath(thee, imol) ((thee)->dielZpath[(imol)])
```

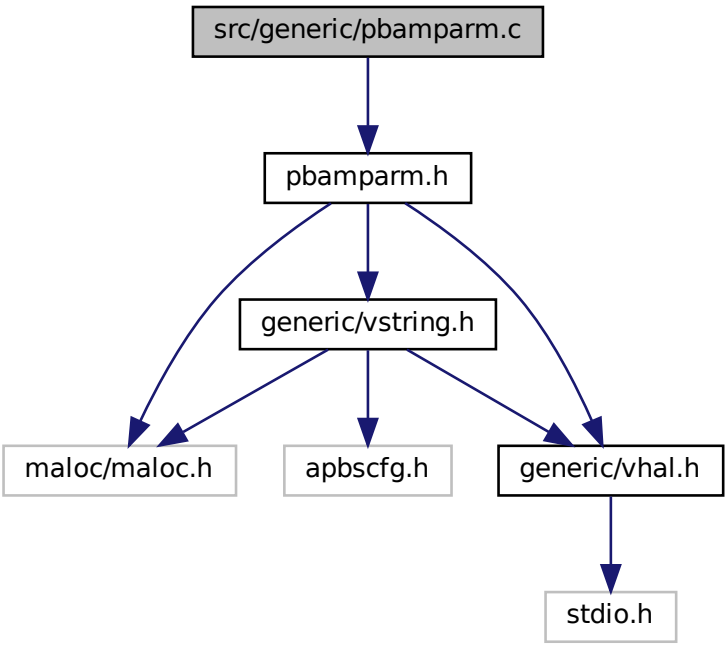
```
00397 #   define NOsh_getKappapath(thee, imol) ((thee)->kappapath[(imol)])
00398 #   define NOsh_getPotpath(thee, imol) ((thee)->potpath[(imol)])
00399 #   define NOsh_getChargepath(thee, imol) ((thee)->chargepath[(imol)])
00400 #   define NOsh_getCalc(thee, icalc) ((thee)->calc[(icalc)])
00401 #   define NOsh_getDielfmt(thee, imap) ((thee)->dielfmt[(imap)])
00402 #   define NOsh_getKappafmt(thee, imap) ((thee)->kappafmt[(imap)])
00403 #   define NOsh_getPotfmt(thee, imap) ((thee)->potfmt[(imap)])
00404 #   define NOsh_getChargefmt(thee, imap) ((thee)->chargefmt[(imap)])
00405
00406 #endif
00407
00408
00409 /* ///////////////////////////////////////////////////////////////////////////
00410    // Class NOsh: Non-inlineable methods (mcsh.c)
00412
00420 VEXTERNC NOsh_PrintType NOsh_printWhat(NOsh *thee, int iprint);
00421
00431 VEXTERNC char* NOsh_elecname(NOsh *thee, int ielec);
00432
00440 VEXTERNC int NOsh_elec2calc(NOsh *thee, int icalc);
00441
00449 VEXTERNC int NOsh_apol2calc(NOsh *thee, int icalc);
00450
00458 VEXTERNC int NOsh_printNarg(NOsh *thee, int iprint);
00459
00468 VEXTERNC int NOsh_printOp(NOsh *thee, int iprint, int iarg);
00469
00480 VEXTERNC int NOsh_printCalc(NOsh *thee, int iprint, int iarg);
00481
00491 VEXTERNC NOsh* NOsh_ctor(int rank, int size);
00492
00499 VEXTERNC NOsh_calc* NOsh_calc_ctor(
00500                               NOsh_CalcType calcType
00501                               );
00502
00509 VEXTERNC int NOsh_calc_copy(
00510                             NOsh_calc *thee,
00511                             NOsh_calc *source
00512                             );
00513
00519 VEXTERNC void  NOsh_calc_dtor(NOsh_calc **thee);
00520
00531 VEXTERNC int   NOsh_ctor2(NOsh *thee, int rank, int size);
00532
00538 VEXTERNC void  NOsh_dtor(NOsh **thee);
00539
00545 VEXTERNC void  NOsh_dtor2(NOsh *thee);
00546
00555 VEXTERNC int   NOsh_parseInput(NOsh *thee, Vio *sock);
00556
00566 VEXTERNC int   NOsh_parseInputFile(NOsh *thee, char *filename);
00567
00577 VEXTERNC int NOsh_setupElecCalc(
00578                             NOsh *thee, /**< NOsh object */
00579                             Valist *alist[NOSH_MAXMOL]
00580                             );
00581
00591 VEXTERNC int NOsh_setupApolCalc(
00592                                NOsh *thee,
00593                                Valist *alist[NOSH_MAXMOL]
00594                                );
00595
00596 #endif
00597
```

9.35 src/generic/pbamparm.c File Reference

Class PBAMparm methods.

```
#include "pbamparm.h"
```

Include dependency graph for pbamparm.c:



Functions

- VPUBLIC PBAMparm * PBAMparm_ctor (PBAMparm_CalcType type)

  Construct PBAMparm object.
- VPUBLIC Vrc_Codes PBAMparm_ctor2 (PBAMparm *thee, PBAMparm_CalcType type)

  FORTRAN stub to construct PBAMparm object ?????????!!!!!!!
- VPUBLIC void PBAMparm_dtor (PBAMparm **thee)

  Object destructor.
- VPUBLIC void PBAMparm_dtor2 (PBAMparm *thee)

  FORTRAN stub for object destructor ?????????!!!!!!!!!!!
- VPUBLIC Vrc_Codes PBAMparm_check (PBAMparm *thee)

  Consistency check for parameter values stored in object.
- VPUBLIC void PBAMparm_copy (PBAMparm *thee, PBAMparm *parm)

  copy PBAMparm object int thee.
- VPRIVATE Vrc_Codes PBAMparm_parseSalt (PBAMparm *thee, Vio *sock)

  Find salt conc and save it as a structure variable.
- VPRIVATE Vrc_Codes PBAMparm_parseRunType (PBAMparm *thee, Vio *sock)

  Find runType and save it as a structure variable.
- VPRIVATE Vrc_Codes PBAMparm_parseRunName (PBAMparm *thee, Vio *sock)

  Find runName and save it as a structure variable.

- VPRIVATE Vrc_Codes PBAMparm_parseRandorient (PBAMparm *thee, Vio *sock)

  Find randomorientation flag and save it as a boolean.
- VPRIVATE Vrc_Codes PBAMparm_parsePBCS (PBAMparm *thee, Vio *sock)

  Find PBC flag and save the type and the boxlength.
- VPRIVATE Vrc_Codes PBAMparm_parseUnits (PBAMparm *thee, Vio *sock)

  Find units flag and save units.
- VPRIVATE Vrc_Codes PBAMparm_parseGridPts (PBAMparm *thee, Vio *sock)

  Find Grid points and save them.
- VPRIVATE Vrc_Codes PBAMparm_parse3Dmap (PBAMparm *thee, Vio *sock)

  Find 3D map filename and save it.
- VPRIVATE Vrc_Codes PBAMparm_parseGrid2D (PBAMparm *thee, Vio *sock)

  Find 2D grid filename and save it.
- VPRIVATE Vrc_Codes PBAMparm_parseDX (PBAMparm *thee, Vio *sock)

  Find DX filename and save it.
- VPRIVATE Vrc_Codes PBAMparm_parseTermcombine (PBAMparm *thee, Vio *sock)

  Find Termination logic and save it.
- VPRIVATE Vrc_Codes **PBAMparm_parseNtraj** (PBAMparm *thee, Vio *sock)
- VPRIVATE Vrc_Codes PBAMparm_parseDiff (PBAMparm *thee, Vio *sock)

  Find diffusion coeffs for each molecule and save them.
- VPRIVATE Vrc_Codes **PBAMparm_parseTerm** (PBAMparm *thee, Vio *sock)
- VPRIVATE Vrc_Codes PBAMparm_parseXYZ (PBAMparm *thee, Vio *sock)

  Find xyz files for each molecule for each traj and save them.
- VPUBLIC Vrc_Codes PBAMparm_parseToken (PBAMparm *thee, char tok[VMAX_BUFSIZE], Vio *sock)

  Parse an MG keyword from an input file.

9.35.1 Detailed Description

Class PBAMparm methods.

Author

Andrew Stevens

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
```

Definition in file pbamparm.c.

9.36 pbamparm.c

```
00001
00057 #include "pbamparm.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_MGPARM)
00062
00063 #endif /* if !defined(VINLINE_MGPARM) */
00064
00065
00066 VPUBLIC PBAMparm* PBAMparm_ctor(PBAMparm_CalcType type) {
00067
00068     /* Set up the structure */
00069     PBAMparm *thee = VNULL;
00070     thee = (PBAMparm*)Vmem_malloc(VNULL, 1, sizeof(PBAMparm));
00071     VASSERT( thee != VNULL);
00072     VASSERT( PBAMparm_ctor2(thee, type) == VRC_SUCCESS );
00073
00074     return thee;
00075 }
00076
00077 VPUBLIC Vrc_Codes PBAMparm_ctor2(PBAMparm *thee, PBAMparm_CalcType type) {
00078
00079     int i;
00080
00081     if (thee == VNULL) return VRC_FAILURE;
00082
00083     thee->parsed = 0;
00084     thee->type = type;
00085     thee->salt = 0;
00086
00087     thee->setsalt = 0;
00088     thee->setruntype = 0;
00089     thee->setrunname = 0;
00090     thee->setunits = 0;
00091
00092     thee->setrandorient = 0;
```

```
00093
00094     thee->setpbcs = 0;
00095     thee->pbcboxlen = 1e15;
00096
00097     // Electrostatics
00098     thee->gridpt = 15;
00099     printf("Found a pts flag in ctor: %d\n", thee->gridpt);
00100     thee->setgridpt = 0;
00101     thee->set3dmap = 0;
00102     thee->setgrid2Dname = 0;
00103     thee->grid2Dct = 0;
00104     thee->setdxname = 0;
00105
00106     //Dynamics
00107     thee->ntraj = 1;
00108     thee->setntraj = 0;
00109
00110     thee->settermcombine = 0;
00111     thee->diffct = 0;
00112
00113     thee->termct = 0;
00114     thee->setterm = 0;
00115
00116     thee->setxyz = 0;
00117     for (i = 0; i<PBAMPARM_MAXMOL; i++) thee->xyzct[i] = 0;
00118
00119     return VRC_SUCCESS;
00120 }
00121
00122 VPUBLIC void PBAMparm_dtor(PBAMparm **thee) {
00123     if ((*thee) != VNULL) {
00124         PBAMparm_dtor2(*thee);
00125         Vmem_free(VNULL, 1, sizeof(PBAMparm), (void **)thee);
00126         (*thee) = VNULL;
00127     }
00128 }
00129
00130 VPUBLIC void PBAMparm_dtor2(PBAMparm *thee) { ; }
00131
00132 VPUBLIC Vrc_Codes PBAMparm_check(PBAMparm *thee) {
00133
00134     Vrc_Codes rc;
00135
00136     rc = VRC_SUCCESS;
00137
00138     Vnm_print(0, "PBAMparm_check:  checking PBAMparm object of type %d.\n",
00139               thee->type);
00140
00141     /* Check to see if we were even filled... */
00142     if (!thee->parsed) {
00143         Vnm_print(2, "PBAMparm_check:  not filled!\n");
00144         return VRC_FAILURE;
00145     }
00146
00147
00148     /* Check type settings */
00149     if(thee->type != PBAMCT_AUTO) {
00150          Vnm_print(2,"PBAMparm_check: type not set");
00151          rc = VRC_FAILURE;
00152     }
00153
00154     return rc;
00155 }
00156
00157 VPUBLIC void PBAMparm_copy(PBAMparm *thee, PBAMparm *parm) {
00158     int i, j, k;
00159     VASSERT(thee != VNULL);
00160     VASSERT(parm != VNULL);
00161
00162     thee->type = parm->type;
00163     thee->parsed = parm->parsed;
00164
00165     thee->salt = parm->salt;
00166     thee->setsalt = parm->setsalt;
00167     for (i=0; i<CHR_MAXLEN; i++) thee->runtype[i] = parm->runtype[i];
00168     thee->setruntype = parm->setruntype;
00169
00170     for (i=0; i<CHR_MAXLEN; i++) thee->runname[i] = parm->runname[i];
00171     thee->setrunname = parm->setrunname;
00172
00173     thee->setrandorient = parm->setrandorient;
```

```
00174
00175     thee->setpbcs = parm->setpbcs;
00176     thee->pbcboxlen = parm->pbcboxlen;
00177
00178     for (i=0; i<CHR_MAXLEN; i++) thee->units[i] = parm->units[i];
00179     thee->setunits = parm->setunits;
00180
00181     // Electrostatic parts
00182     thee->gridpt = parm->gridpt;
00183     thee->setgridpt = parm->setgridpt;
00184     for (i=0; i<CHR_MAXLEN; i++) thee->map3dname[i] = parm->map3dname[i];
00185     thee->set3dmap = parm->set3dmap;
00186
00187
00188     thee->grid2Dct = parm->grid2Dct;
00189     for (i=0; i<PBAMPARM_MAXWRITE; i++)
00190     {
00191         for (j=0; j<CHR_MAXLEN; j++)
00192         {
00193             thee->grid2Dname[i][j] = parm->grid2Dname[i][j];
00194             thee->grid2Dax[i][j] = parm->grid2Dax[i][j];
00195         }
00196         thee->grid2Dloc[i] = parm->grid2Dloc[i];
00197     }
00198
00199     for (i=0; i<CHR_MAXLEN; i++) thee->dxname[i] = parm->dxname[i];
00200     thee->setdxname = parm->setdxname;
00201
00202     // Dynamics parts
00203     thee->ntraj = parm->ntraj;
00204     thee->setntraj = parm->setntraj;
00205
00206     for (i=0; i<CHR_MAXLEN; i++) thee->termcombine[i] = parm->termcombine[i];
00207     thee->settermcombine = parm->settermcombine;
00208
00209     thee->diffct = parm->diffct;
00210
00211     for (i=0; i<PBAMPARM_MAXMOL; i++)
00212     {
00213         for (j=0; j<CHR_MAXLEN; j++)
00214         {
00215             thee->moveType[i][j] = parm->moveType[i][j];
00216         }
00217         thee->transDiff[i] = parm->transDiff[i];
00218         thee->rotDiff[i] = parm->rotDiff[i];
00219     }
00220
00221     thee->termct = parm->termct;
00222     thee->setterm = parm->setterm;
00223     thee->confilct = parm->confilct;
00224
00225     for (i=0; i<PBAMPARM_MAXWRITE; i++)
00226     {
00227         for (j=0; j<CHR_MAXLEN; j++)
00228         {
00229             thee->termnam[i][j] = parm->termnam[i][j];
00230             thee->confil[i][j] = parm->confil[i][j];
00231         }
00232         thee->termVal[i] = parm->termVal[i];
00233         thee->termnu[i][0] = parm->termnu[i][0];
00234     }
00235
00236     thee->setxyz = parm->setxyz;
00237     for (i = 0; i<PBAMPARM_MAXMOL; i++)
00238     {
00239         thee->xyzct[i] = parm->xyzct[i];
00240         for (j = 0; j<PBAMPARM_MAXWRITE; j++)
00241         {
00242             for (k = 0; k<CHR_MAXLEN; k++)
00243             {
00244                 thee->xyzfil[i][j][k] = parm->xyzfil[i][j][k];
00245             }
00246         }
00247     }
00248
00249 }
00250
00251
00252 VPRIVATE Vrc_Codes PBAMparm_parseSalt(PBAMparm *thee, Vio *sock){
00253     const char* name = "salt";
00254     char tok[VMAX_BUFSIZE];
```

```
00255  double tf;
00256     if(Vio_scanf(sock, "%s", tok) == 0) {
00257         Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00258         return VRC_WARNING;
00259     }
00260
00261     if (sscanf(tok, "%lf", &tf) == 0){
00262         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing %s keyword!\n", tok, name);
00263         return VRC_WARNING;
00264     }else{
00265         thee->salt = tf;
00266     }
00267     thee->setsalt = 1;
00268     return VRC_SUCCESS;
00269 }
00270
00271 VPRIVATE Vrc_Codes PBAMparm_parseRunType(PBAMparm *thee, Vio *sock){
00272     const char* name = "runtype";
00273     char tok[VMAX_BUFSIZE];
00274
00275     if(Vio_scanf(sock, "%s", tok) == 0) {
00276       Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00277       return VRC_WARNING;
00278     } else if(Vstring_strcasecmp(tok, "dynamics") == 0){
00279       Vnm_print(2, "parsePBAM:  Dynamics has been moved out of the ELEC section!\n");
00280       return VRC_WARNING;
00281     } else {
00282       strncpy(thee->runtype, tok, CHR_MAXLEN);
00283       thee->setruntype=1;
00284     }
00285
00286     return VRC_SUCCESS;
00287 }
00288
00289 VPRIVATE Vrc_Codes PBAMparm_parseRunName(PBAMparm *thee, Vio *sock){
00290     const char* name = "runname";
00291     char tok[VMAX_BUFSIZE];
00292
00293     if(Vio_scanf(sock, "%s", tok) == 0) {
00294       Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00295       return VRC_WARNING;
00296     } else {
00297       strncpy(thee->runname, tok, CHR_MAXLEN);
00298       thee->setrunname=1;
00299     }
00300
00301     return VRC_SUCCESS;
00302 }
00303
00304 VPRIVATE Vrc_Codes PBAMparm_parseRandorient(PBAMparm *thee, Vio *sock){
00305     const char* name = "randorient";
00306     thee->setrandorient=1;
00307     return VRC_SUCCESS;
00308 }
00309
00310 VPRIVATE Vrc_Codes PBAMparm_parsePBCS(PBAMparm *thee, Vio *sock){
00311     const char* name = "pbc";
00312     char tok[VMAX_BUFSIZE];
00313     double tf;
00314     int td;
00315     if(Vio_scanf(sock, "%s", tok) == 0) {
00316         Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00317         return VRC_WARNING;
00318     }
00319
00320     if (sscanf(tok, "%d", &td) == 0) {
00321         Vnm_print(2, "parsePBAM:  Read non-int (%s) while parsing pbc keyword!\n", tok);
00322         return VRC_FAILURE;
00323     } else{
00324         thee->setpbcs = td;
00325     }
00326
00327     if (sscanf(tok, "%lf", &tf) == 0){
00328         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing %s keyword!\n", tok, name);
00329         return VRC_WARNING;
00330     }else{
00331         thee->pbcboxlen = tf;
00332     }
00333     return VRC_SUCCESS;
00334 }
00335
```

```
00336 VPRIVATE Vrc_Codes PBAMparm_parseUnits(PBAMparm *thee, Vio *sock){
00337     const char* name = "units";
00338     char tok[VMAX_BUFSIZE];
00339
00340     if(Vio_scanf(sock, "%s", tok) == 0) {
00341       Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00342       return VRC_WARNING;
00343     } else {
00344       strncpy(thee->units, tok, CHR_MAXLEN);
00345       thee->setunits=1;
00346     }
00347
00348     return VRC_SUCCESS;
00349 }
00350
00351 VPRIVATE Vrc_Codes PBAMparm_parseGridPts(PBAMparm *thee, Vio *sock){
00352     const char* name = "dime";
00353     char tok[VMAX_BUFSIZE];
00354     int td;
00355     if(Vio_scanf(sock, "%s", tok) == 0) {
00356         Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00357         return VRC_WARNING;
00358     }
00359
00360     if (sscanf(tok, "%d", &td) == 0){
00361         Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing %s keyword!\n", tok, name);
00362         return VRC_WARNING;
00363     }else{
00364         printf("Found a dime flag in parse: %d\n", td);
00365         thee->gridpt = td;
00366     }
00367     thee->setgridpt = 1;
00368     return VRC_SUCCESS;
00369 }
00370
00371 VPRIVATE Vrc_Codes PBAMparm_parse3Dmap(PBAMparm *thee, Vio *sock){
00372     const char* name = "3dmap";
00373     char tok[VMAX_BUFSIZE];
00374
00375  Vnm_print(2, "PBAM: 3dmap keyword has been deprecated! Please use in conjuction with the write
       keyword.");
00376  return VRC_FAILURE;
00377
00378  /*
00379     if(Vio_scanf(sock, "%s", tok) == 0) {
00380       Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00381       return VRC_WARNING;
00382     } else {
00383       strncpy(thee->map3dname, tok, CHR_MAXLEN);
00384       thee->set3dmap=1;
00385     }
00386
00387     return VRC_SUCCESS;
00388  */
00389 }
00390
00391 VPRIVATE Vrc_Codes PBAMparm_parseGrid2D(PBAMparm *thee, Vio *sock){
00392     const char* name = "grid2d";
00393     char tok[VMAX_BUFSIZE];
00394     double tf;
00395
00396     if(Vio_scanf(sock, "%s", tok) == 0) {
00397         Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00398         return VRC_WARNING;
00399     } else {
00400       strncpy(thee->grid2Dname[thee->grid2Dct], tok, CHR_MAXLEN);
00401       thee->setgrid2Dname=1;
00402     }
00403
00404     if(Vio_scanf(sock, "%s", tok) == 0) {
00405         Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00406         return VRC_WARNING;
00407     } else {
00408       strncpy(thee->grid2Dax[thee->grid2Dct], tok, CHR_MAXLEN);
00409     }
00410
00411     if(Vio_scanf(sock, "%s", tok) == 0) {
00412         Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00413         return VRC_WARNING;
00414     }
00415
```

```
00416     if (sscanf(tok, "%lf", &tf) == 0){
00417         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing %s keyword!\n", tok, name);
00418         return VRC_WARNING;
00419     }else{
00420         thee->grid2Dloc[thee->grid2Dct] = tf;
00421         thee->grid2Dct = thee->grid2Dct+1;
00422     }
00423     return VRC_SUCCESS;
00424 }
00425
00426 VPRIVATE Vrc_Codes PBAMparm_parseDX(PBAMparm *thee, Vio *sock){
00427  Vnm_print(2, "PBAM's dx keyword is deprecated. Please use the write keyword!\n");
00428  return VRC_FAILURE;
00429  /*
00430  const char* name = "dx";
00431     char tok[VMAX_BUFSIZE];
00432
00433     if(Vio_scanf(sock, "%s", tok) == 0) {
00434       Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00435       return VRC_WARNING;
00436     } else {
00437       strncpy(thee->dxname, tok, CHR_MAXLEN);
00438       thee->setdxname=1;
00439     }
00440     return VRC_SUCCESS;
00441  */
00442 }
00443
00444 VPRIVATE Vrc_Codes PBAMparm_parseTermcombine(PBAMparm *thee, Vio *sock){
00445     const char* name = "termcombine";
00446     char tok[VMAX_BUFSIZE];
00447
00448     if(Vio_scanf(sock, "%s", tok) == 0) {
00449       Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00450       return VRC_WARNING;
00451     } else {
00452       strncpy(thee->termcombine, tok, CHR_MAXLEN);
00453       thee->settermcombine=1;
00454     }
00455     return VRC_SUCCESS;
00456 }
00457
00458 VPRIVATE Vrc_Codes PBAMparm_parseNtraj(PBAMparm *thee, Vio *sock){
00459     const char* name = "ntraj";
00460     char tok[VMAX_BUFSIZE];
00461     int td;
00462     if(Vio_scanf(sock, "%s", tok) == 0) {
00463         Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00464         return VRC_WARNING;
00465     }
00466
00467     if (sscanf(tok, "%d", &td) == 0){
00468         Vnm_print(2, "NOsh:  Read non-integer (%s) while parsing %s keyword!\n", tok, name);
00469         return VRC_WARNING;
00470     }else{
00471         thee->ntraj = td;
00472     }
00473     thee->setntraj = 1;
00474     return VRC_SUCCESS;
00475 }
00476
00477 VPRIVATE Vrc_Codes PBAMparm_parseDiff(PBAMparm *thee, Vio *sock){
00478     const char* name = "diff";
00479     char tok[VMAX_BUFSIZE];
00480     int molind;
00481     double tf;
00482
00483     if(Vio_scanf(sock, "%s", tok) == 0) {
00484         Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00485         return VRC_WARNING;
00486     }
00487
00488     // // looking for index
00489     if (sscanf(tok, "%d", &molind) == 0){
00490         Vnm_print(2, "NOsh:  Read non-int (%s) while parsing %s keyword!\n", tok, name);
00491         return VRC_WARNING;
00492     }
00493
00494     molind -= 1;
00495     // looking for move type = move, stat, rot
00496     if(Vio_scanf(sock, "%s", tok) == 0) {
```

```
00497         Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00498         return VRC_WARNING;
00499     } else {
00500        strncpy(thee->moveType[molind], tok, CHR_MAXLEN);
00501       thee->diffct += 1;
00502     }
00503
00504     if (strncmp(thee->moveType[molind], "move", 4) == 0)
00505     {
00506       if(Vio_scanf(sock, "%s", tok) == 0) {
00507           Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00508           return VRC_WARNING;
00509       }
00510       if (sscanf(tok, "%lf", &tf) == 0){
00511           Vnm_print(2, "NOsh:  Read non-float (%s) while parsing %s keyword!\n", tok, name);
00512           return VRC_WARNING;
00513       }else{
00514           thee->transDiff[molind] = tf;
00515       }
00516
00517       // rot diffusion coeff
00518       if(Vio_scanf(sock, "%s", tok) == 0) {
00519           Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00520           return VRC_WARNING;
00521       }
00522       if (sscanf(tok, "%lf", &tf) == 0){
00523           Vnm_print(2, "NOsh:  Read non-float (%s) while parsing %s keyword!\n", tok, name);
00524           return VRC_WARNING;
00525       }else{
00526           thee->rotDiff[molind] = tf;
00527       }
00528     } else if (strncmp(thee->moveType[molind], "rot", 3) == 0)
00529     {
00530       if(Vio_scanf(sock, "%s", tok) == 0) {
00531           Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00532           return VRC_WARNING;
00533       }
00534       if (sscanf(tok, "%lf", &tf) == 0){
00535           Vnm_print(2, "NOsh:  Read non-float (%s) while parsing %s keyword!\n", tok, name);
00536           return VRC_WARNING;
00537       }else{
00538           thee->rotDiff[molind] = tf;
00539           thee->transDiff[molind] = 0.0;
00540       }
00541     } else{
00542        thee->transDiff[molind] = 0.0;
00543        thee->rotDiff[molind] = 0.0;
00544     }
00545
00546     return VRC_SUCCESS;
00547 }
00548
00549 VPRIVATE Vrc_Codes PBAMparm_parseTerm(PBAMparm *thee, Vio *sock){
00550     const char* name = "term";
00551     char tok[VMAX_BUFSIZE];
00552     double tf;
00553     int td;
00554
00555     // looking for term name
00556     if(Vio_scanf(sock, "%s", tok) == 0) {
00557         Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00558         return VRC_WARNING;
00559     }else {
00560        if(strncmp(tok, "position", 8)==0){
00561         return PBAMparm_parseTerm(thee, sock);
00562        }else{
00563         strncpy(thee->termnam[thee->termct], tok, CHR_MAXLEN);
00564        }
00565     }
00566
00567     if (strncmp(thee->termnam[thee->termct], "contact", 7) == 0)
00568     {
00569       if(Vio_scanf(sock, "%s", tok) == 0) {
00570           Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00571           return VRC_WARNING;
00572       }else{
00573           strncpy(thee->confil[thee->confilct], tok, CHR_MAXLEN);
00574       }
00575
00576       if(Vio_scanf(sock, "%s", tok) == 0) {
00577           Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
```

```
00578           return VRC_WARNING;
00579       }
00580       if (sscanf(tok, "%lf", &tf) == 0){
00581           Vnm_print(2, "NOsh:  Read non-float (%s) while parsing %s keyword!\n", tok, name);
00582           return VRC_WARNING;
00583       }else
00584       {
00585           thee->termVal[thee->termct] = tf;
00586           thee->termnu[thee->termct][0] = 0;
00587           thee->confilct += 1;
00588       }
00589     } else if (strncmp(thee->termnam[thee->termct], "time", 4) == 0)
00590     {
00591       if(Vio_scanf(sock, "%s", tok) == 0) {
00592           Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00593           return VRC_WARNING;
00594       }
00595       if (sscanf(tok, "%lf", &tf) == 0){
00596           Vnm_print(2, "NOsh:  Read non-float (%s) while parsing %s keyword!\n", tok, name);
00597           return VRC_WARNING;
00598       }else{
00599           thee->termVal[thee->termct] = tf;
00600           thee->termnu[thee->termct][0] = 0;
00601       }
00602     } else //if (strncmp(thee->termnam[thee->termct], "position", 8) == 0)
00603     {
00604       if(Vio_scanf(sock, "%s", tok) == 0) {
00605           Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00606           return VRC_WARNING;
00607       }
00608       if (sscanf(tok, "%lf", &tf) == 0){
00609           Vnm_print(2, "NOsh:  Read non-float (%s) while parsing %s keyword!\n", tok, name);
00610           return VRC_WARNING;
00611       }else{
00612           thee->termVal[thee->termct] = tf;
00613       }
00614
00615       if(Vio_scanf(sock, "%s", tok) == 0) {
00616           Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00617           return VRC_WARNING;
00618       }
00619       if (sscanf(tok, "%d", &td) == 0){
00620           Vnm_print(2, "NOsh:  Read non-float (%s) while parsing %s keyword!\n", tok, name);
00621           return VRC_WARNING;
00622       }else{
00623           thee->termnu[thee->termct][0] = td-1;
00624       }
00625     }
00626
00627     thee->setterm = 1;
00628     thee->termct += 1;
00629     return VRC_SUCCESS;
00630 }
00631
00632 VPRIVATE Vrc_Codes PBAMparm_parseXYZ(PBAMparm *thee, Vio *sock){
00633     const char* name = "xyz";
00634     char tok[VMAX_BUFSIZE];
00635     int td, mol;
00636
00637     if(Vio_scanf(sock, "%s", tok) == 0) {
00638         Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00639         return VRC_WARNING;
00640     }
00641
00642     // // looking for index
00643     if (sscanf(tok, "%d", &td) == 0){
00644         Vnm_print(2, "NOsh:  Read non-int (%s) while parsing %s keyword!\n", tok, name);
00645         return VRC_WARNING;
00646     } else{
00647         printf("This is my mol in parseXYZ: %d", td);
00648         mol = td-1;
00649     }
00650
00651     // looking for move type = move, stat, rot
00652     if(Vio_scanf(sock, "%s", tok) == 0) {
00653         Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00654         return VRC_WARNING;
00655     } else {
00656        strncpy(thee->xyzfil[mol][thee->xyzct[mol]], tok, CHR_MAXLEN);
00657       thee->xyzct[mol] += 1;
00658     }
```

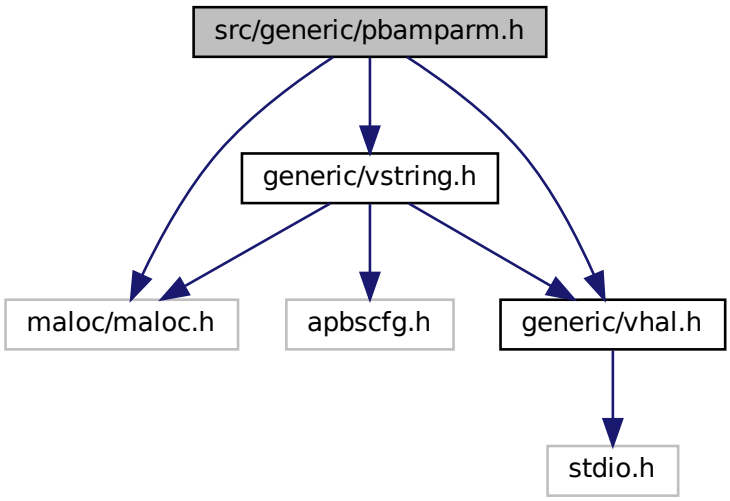
```
00659     return VRC_SUCCESS;
00660 }
00661
00662 VPUBLIC Vrc_Codes PBAMparm_parseToken(PBAMparm *thee, char tok[VMAX_BUFSIZE],
00663   Vio *sock) {
00664
00665     if (thee == VNULL) {
00666         Vnm_print(2, "parsePBAM:  got NULL thee!\n");
00667         return VRC_WARNING;
00668     }
00669     if (sock == VNULL) {
00670         Vnm_print(2, "parsePBAM:  got NULL socket!\n");
00671         return VRC_WARNING;
00672     }
00673
00674     Vnm_print(0, "PBAMparm_parseToken:  trying %s...\n", tok);
00675
00676     // General terms to parse
00677     if (Vstring_strcasecmp(tok, "salt") == 0) {
00678         return PBAMparm_parseSalt(thee, sock);
00679     }else if (Vstring_strcasecmp(tok, "runtype") == 0) {
00680         return PBAMparm_parseRunType(thee, sock);
00681     }else if (Vstring_strcasecmp(tok, "runname") == 0) {
00682         return PBAMparm_parseRunName(thee, sock);
00683     }else if (Vstring_strcasecmp(tok, "randorient") == 0) {
00684         return PBAMparm_parseRandorient(thee, sock);
00685     }else if (Vstring_strcasecmp(tok, "pbc") == 0) {
00686         return PBAMparm_parsePBCS(thee, sock);
00687     }else if (Vstring_strcasecmp(tok, "units") == 0) {
00688         return PBAMparm_parseUnits(thee, sock);
00689     }
00690
00691     // Electrostatic parsing
00692     else if (Vstring_strcasecmp(tok, "dime") == 0) {
00693         return PBAMparm_parseGridPts(thee, sock);
00694     }else if (Vstring_strcasecmp(tok, "3dmap") == 0) {
00695         return PBAMparm_parse3Dmap(thee, sock);
00696     }else if (Vstring_strcasecmp(tok, "grid2d") == 0) {
00697         return PBAMparm_parseGrid2D(thee, sock);
00698     }else if (Vstring_strcasecmp(tok, "dx") == 0) {
00699         return PBAMparm_parseDX(thee, sock);
00700     }
00701
00702     // Dynamics parsing
00703     else if (Vstring_strcasecmp(tok, "ntraj") == 0) {
00704         return PBAMparm_parseNtraj(thee, sock);
00705     }else if (Vstring_strcasecmp(tok, "termcombine") == 0) {
00706         return PBAMparm_parseTermcombine(thee, sock);
00707     }else if (Vstring_strcasecmp(tok, "diff") == 0) {
00708         return PBAMparm_parseDiff(thee, sock);
00709     }else if (Vstring_strcasecmp(tok, "term") == 0) {
00710         return PBAMparm_parseTerm(thee, sock);
00711     }else if (Vstring_strcasecmp(tok, "xyz") == 0) {
00712         return PBAMparm_parseXYZ(thee, sock);
00713     }
00714
00715
00716     else
00717      return 0;
00718
00719   /*else {
00720         Vnm_print(2, "parsePBAM:  Unrecognized keyword (%s)!\n", tok);
00721         return VRC_WARNING;
00722     }
00723     return VRC_FAILURE;
00724   */
00725 }
```

9.37 src/generic/pbamparm.h File Reference

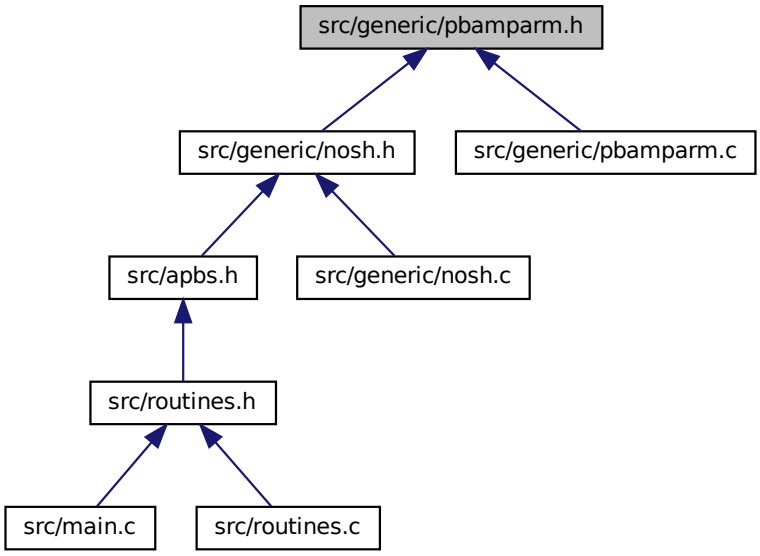
Contains declarations for class PBAMparm.

```
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/vstring.h"
```

Include dependency graph for pbamparm.h:



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



Data Structures

- struct sPBAMparm

  Parameter structure for PBAM-specific variables from input files.

Macros

- #define CHR_MAXLEN 1000

  Number of things that can be written out in a single calculation.
- #define **PBAMPARM_MAXWRITE** 15
- #define **PBAMPARM_MAXMOL** 150

Typedefs

- typedef enum ePBAMparm_CalcType PBAMparm_CalcType

  Declare PBAMparm_CalcType type.
- typedef struct sPBAMparm PBAMparm

  Parameter structure for PBAM-specific variables from input files.

Enumerations

- enum ePBAMparm_CalcType { PBAMCT_AUTO =1 }

  Calculation type.

Functions

- VEXTERNC PBAMparm * PBAMparm_ctor (PBAMparm_CalcType type)

  Construct PBAMparm object.
- VEXTERNC Vrc_Codes PBAMparm_ctor2 (PBAMparm *thee, PBAMparm_CalcType type)

  FORTRAN stub to construct PBAMparm object ?????????!!!!!!!
- VEXTERNC void PBAMparm_dtor (PBAMparm **thee)

  Object destructor.
- VEXTERNC void PBAMparm_dtor2 (PBAMparm *thee)

  FORTRAN stub for object destructor ?????????!!!!!!!!!!!!
- VEXTERNC Vrc_Codes PBAMparm_check (PBAMparm *thee)

  Consistency check for parameter values stored in object.
- VEXTERNC Vrc_Codes PBAMparm_parseToken (PBAMparm *thee, char tok[VMAX_BUFSIZE], Vio *sock)

  Parse an MG keyword from an input file.
- VEXTERNC void PBAMparm_copy (PBAMparm *thee, PBAMparm *parm)

  copy PBAMparm object int thee.
- VPRIVATE Vrc_Codes PBAMparm_parseSalt (PBAMparm *thee, Vio *sock)

  Find salt conc and save it as a structure variable.
- VPRIVATE Vrc_Codes PBAMparm_parseRunType (PBAMparm *thee, Vio *sock)

  Find runType and save it as a structure variable.
- VPRIVATE Vrc_Codes PBAMparm_parseRunName (PBAMparm *thee, Vio *sock)

  Find runName and save it as a structure variable.
- VPRIVATE Vrc_Codes PBAMparm_parseRandorient (PBAMparm *thee, Vio *sock)

  Find randomorientation flag and save it as a boolean.
- VPRIVATE Vrc_Codes PBAMparm_parsePBCS (PBAMparm *thee, Vio *sock)

Find PBC flag and save the type and the boxlength.

- VPRIVATE Vrc_Codes PBAMparm_parseUnits (PBAMparm *thee, Vio *sock)

  Find units flag and save units.
- VPRIVATE Vrc_Codes PBAMparm_parse3Dmap (PBAMparm *thee, Vio *sock)

  Find 3D map filename and save it.
- VPRIVATE Vrc_Codes PBAMparm_parseGrid2D (PBAMparm *thee, Vio *sock)

  Find 2D grid filename and save it.
- VPRIVATE Vrc_Codes PBAMparm_parseDX (PBAMparm *thee, Vio *sock)

  Find DX filename and save it.
- VPRIVATE Vrc_Codes PBAMparm_parseGridPts (PBAMparm *thee, Vio *sock)

  Find Grid points and save them.
- VPRIVATE Vrc_Codes PBAMparm_parseTermcombine (PBAMparm *thee, Vio *sock)

  Find Termination logic and save it.
- VPRIVATE Vrc_Codes PBAMparm_parseDiff (PBAMparm *thee, Vio *sock)

  Find diffusion coeffs for each molecule and save them.
- VPRIVATE Vrc_Codes PBAMparm_parseXYZ (PBAMparm *thee, Vio *sock)

  Find xyz files for each molecule for each traj and save them.

9.37.1 Detailed Description

Contains declarations for class PBAMparm.

Version

Id

Author

Lisa Felberg

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) 2010-2020 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 pbamparm.h.

9.38 pbamparm.h

```
00001
00064 #ifndef _PBAMPARM_H_
00065 #define _PBAMPARM_H_
00066
00067 /* Generic header files */
00068 #include "maloc/maloc.h"
00069
00070 #include "generic/vhal.h"
00071 #include "generic/vstring.h"
00072
00076 #define CHR_MAXLEN 1000
00077 #define PBAMPARM_MAXWRITE 15
00078 #define PBAMPARM_MAXMOL 150
00079
00084 enum ePBAMparm_CalcType {
00085     //other methods disabled for now only auto currently implemented.
00086  //PBAMCT_MANUAL=0,  /**< PBAM-manual */
00087     PBAMCT_AUTO=1,
00088     //PBAMCT_NONE=2 /**< not defined */
00089 };
00090
00095 typedef enum ePBAMparm_CalcType PBAMparm_CalcType;
00096
00105 typedef struct sPBAMparm {
00106
00107     PBAMparm_CalcType type;
00108     int parsed;
00110     /* *** GENERIC PARAMETERS *** */
00111     double salt;
00112     int setsalt;
00113
00114     // This is the type of run you want
00115     char runtype[CHR_MAXLEN];
00116     int setruntype;
00117
00118     // This is the name for output files
00119     char runname[CHR_MAXLEN];
00120     int setrunname;
00121
00122     // For setting random orientation of molecules
00123     int setrandorient;
00124
00125     // For periodic boundary conditions
00126     double pbcboxlen;
00127     int setpbcs;
00128
00129     // This is units of the calculation
00130     char units[CHR_MAXLEN];
00131     int setunits;
```

```
00132
00133     //
00134     // ELECTROSTATICS
00135     //
00136     // For the grid, store gridpt
00137     int gridpt;
00138     int setgridpt;
00139
00140     // For 3d map printing
00141     char map3dname[CHR_MAXLEN];
00142     int set3dmap;
00143
00144     // For 2D
00145     char grid2Dname[PBAMPARM_MAXWRITE][CHR_MAXLEN];
00146     char grid2Dax[PBAMPARM_MAXWRITE][CHR_MAXLEN];
00147     double grid2Dloc[PBAMPARM_MAXWRITE];
00148     int grid2Dct;
00149     int setgrid2Dname;
00150
00151     // For dx
00152     char dxname[CHR_MAXLEN];
00153     int setdxname;
00154
00155     //
00156     // DYNAMICS
00157     //
00158     int ntraj;
00159     int setntraj;
00160
00161     char termcombine[CHR_MAXLEN];
00162     int settermcombine;
00163
00164     int diffct;
00165     char moveType[PBAMPARM_MAXMOL][CHR_MAXLEN];
00166     double transDiff[PBAMPARM_MAXMOL];
00167     double rotDiff[PBAMPARM_MAXMOL];
00168
00169     int termct;
00170     int setterm;
00171
00172     char termnam[PBAMPARM_MAXWRITE][CHR_MAXLEN];
00173     int termnu[PBAMPARM_MAXWRITE][1];
00174     double termVal[PBAMPARM_MAXWRITE];
00175     char confil[PBAMPARM_MAXWRITE][CHR_MAXLEN];
00176     double conpad[PBAMPARM_MAXWRITE];
00177     int confilct;
00178
00179     int setxyz;
00180     int xyzct[PBAMPARM_MAXMOL];
00181     char xyzfil[PBAMPARM_MAXMOL][PBAMPARM_MAXWRITE][CHR_MAXLEN];
00182
00183 } PBAMparm;
00184
00191 VEXTERNC PBAMparm* PBAMparm_ctor(PBAMparm_CalcType type);
00192
00200 VEXTERNC Vrc_Codes PBAMparm_ctor2(PBAMparm *thee, PBAMparm_CalcType type);
00201
00207 VEXTERNC void PBAMparm_dtor(PBAMparm **thee);
00208
00214 VEXTERNC void PBAMparm_dtor2(PBAMparm *thee);
00215
00222 VEXTERNC Vrc_Codes PBAMparm_check(PBAMparm *thee);
00223
00233 VEXTERNC Vrc_Codes PBAMparm_parseToken(PBAMparm *thee, char tok[VMAX_BUFSIZE],
00234                                        Vio *sock);
00242 VEXTERNC void PBAMparm_copy(PBAMparm *thee, PBAMparm *parm);
00243
00251 VPRIVATE Vrc_Codes PBAMparm_parseSalt(PBAMparm *thee, Vio *sock);
00252
00260 VPRIVATE Vrc_Codes PBAMparm_parseRunType(PBAMparm *thee, Vio *sock);
00261
00269 VPRIVATE Vrc_Codes PBAMparm_parseRunName(PBAMparm *thee, Vio *sock);
00270
00278 VPRIVATE Vrc_Codes PBAMparm_parseRandorient(PBAMparm *thee, Vio *sock);
00279
00287 VPRIVATE Vrc_Codes PBAMparm_parsePBCS(PBAMparm *thee, Vio *sock);
00288
00296 VPRIVATE Vrc_Codes PBAMparm_parseUnits(PBAMparm *thee, Vio *sock);
00297
00305 VPRIVATE Vrc_Codes PBAMparm_parse3Dmap(PBAMparm *thee, Vio *sock);
00306
```

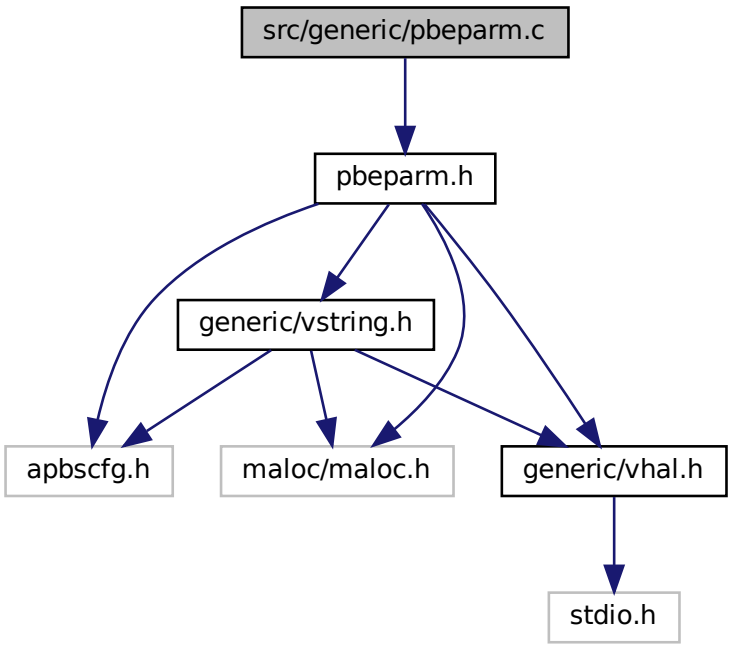
```
00314 VPRIVATE Vrc_Codes PBAMparm_parseGrid2D(PBAMparm *thee, Vio *sock);
00315
00323 VPRIVATE Vrc_Codes PBAMparm_parseDX(PBAMparm *thee, Vio *sock);
00324
00332 VPRIVATE Vrc_Codes PBAMparm_parseGridPts(PBAMparm *thee, Vio *sock);
00333
00341 VPRIVATE Vrc_Codes PBAMparm_parseTermcombine(PBAMparm *thee, Vio *sock);
00342
00350 VPRIVATE Vrc_Codes PBAMparm_parseDiff(PBAMparm *thee, Vio *sock);
00351
00359 VPRIVATE Vrc_Codes PBAMparm_parseXYZ(PBAMparm *thee, Vio *sock);
00360
00361
00362
00363
00364 #endif
00365
```

9.39 src/generic/pbeparm.c File Reference

Class PBEparm methods.

`#include "pbeparm.h"`

Include dependency graph for pbeparm.c:



Functions

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

  Get charge (e) of specified ion species.

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

  Get concentration (M) of specified ion species.

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

  Get radius (A) of specified ion species.
- VPUBLIC double **PBEparm_getzmem** (PBEparm *thee)
- VPUBLIC double **PBEparm_getLmem** (PBEparm *thee)
- VPUBLIC double **PBEparm_getmembraneDiel** (PBEparm *thee)
- VPUBLIC double **PBEparm_getmemv** (PBEparm *thee)
- VPUBLIC PBEparm * PBEparm_ctor ()

  Construct PBEparm object.
- VPUBLIC int PBEparm_ctor2 (PBEparm *thee)

  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.

9.39.1 Detailed Description

Class PBEparm methods.

Author

Nathan Baker

Version

Id

Attention

Definition in file pbeparm.c.

9.40 pbeparm.c

```
00001
```

```
00057 #include "pbeparm.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_MGPARM)
00062
00063 #endif /* if !defined(VINLINE_MGPARM) */
00064
00065 VPUBLIC double PBEparm_getIonCharge(PBEparm *thee, int i) {
00066     VASSERT(thee != VNULL);
00067     VASSERT(i < thee->nion);
00068     return thee->ionq[i];
00069 }
00070
00071 VPUBLIC double PBEparm_getIonConc(PBEparm *thee, int i) {
00072     VASSERT(thee != VNULL);
00073     VASSERT(i < thee->nion);
00074     return thee->ionc[i];
00075 }
00076
00077 VPUBLIC double PBEparm_getIonRadius(PBEparm *thee, int i) {
00078     VASSERT(thee != VNULL);
00079     VASSERT(i < thee->nion);
00080     return thee->ionr[i];
00081 }
00082
00083 /*-------------------------------------------------------------------------*/
00084 /* Added by Michael Grabe                                                  */
00085 /*-------------------------------------------------------------------------*/
00086
00087 VPUBLIC double PBEparm_getzmem(PBEparm *thee) {
00088     VASSERT(thee != VNULL);
00089     return thee->zmem;
00090 }
00091 VPUBLIC double PBEparm_getLmem(PBEparm *thee) {
00092     VASSERT(thee != VNULL);
00093     return thee->Lmem;
00094 }
00095 VPUBLIC double PBEparm_getmembraneDiel(PBEparm *thee) {
00096     VASSERT(thee != VNULL);
00097     return thee->mdie;
00098 }
00099 VPUBLIC double PBEparm_getmemv(PBEparm *thee) {
00100     VASSERT(thee != VNULL);
00101     return thee->memv;
00102 }
00103
00104 VPUBLIC PBEparm* PBEparm_ctor() {
00105
00106     /* Set up the structure */
00107     PBEparm *thee = VNULL;
00108     thee = (PBEparm*)Vmem_malloc(VNULL, 1, sizeof(PBEparm));
00109     VASSERT( thee != VNULL);
00110     VASSERT( PBEparm_ctor2(thee) );
00111
00112     return thee;
00113 }
00114
00115 VPUBLIC int PBEparm_ctor2(PBEparm *thee) {
00116
00117     int i;
00118
00119     if (thee == VNULL) return 0;
00120
00121     thee->parsed = 0;
00122
00123     thee->setmolid = 0;
00124     thee->setpbetype = 0;
00125     thee->setbcfl = 0;
00126     thee->setnion = 0;
00127     for (i=0; i<MAXION; i++){
00128         thee->setion[i] = 0;
00129         thee->ionq[i] = 0.0;
00130         thee->ionc[i] = 0.0;
00131         thee->ionr[i] = 0.0;
00132     }
00133     thee->setpdie = 0;
00134     thee->setsdie = 0;
00135     thee->setsrfm = 0;
00136     thee->setsrad = 0;
00137     thee->setswin = 0;
```

```
00138     thee->settemp = 0;
00139     thee->setcalcenergy = 0;
00140     thee->setcalcforce = 0;
00141     thee->setsdens = 0;
00142     thee->numwrite = 0;
00143     thee->setwritemat = 0;
00144     thee->nion = 0;
00145     thee->sdens = 0;
00146     thee->swin = 0;
00147     thee->srad = 1.4;
00148     thee->useDielMap = 0;
00149     thee->useKappaMap = 0;
00150     thee->usePotMap = 0;
00151     thee->useChargeMap = 0;
00152
00153     /*-----------------------------------------------*/
00154     /* Added by Michael Grabe                        */
00155     /*-----------------------------------------------*/
00156
00157     thee->setzmem = 0;
00158     thee->setLmem = 0;
00159     thee->setmdie = 0;
00160     thee->setmemv = 0;
00161
00162     /*-----------------------------------------------*/
00163
00164     thee->smsize = 0.0;
00165     thee->smvolume = 0.0;
00166
00167     thee->setsmsize = 0;
00168     thee->setsmvolume = 0;
00169
00170     return 1;
00171 }
00172
00173 VPUBLIC void PBEparm_dtor(PBEparm **thee) {
00174     if ((*thee) != VNULL) {
00175         PBEparm_dtor2(*thee);
00176         Vmem_free(VNULL, 1, sizeof(PBEparm), (void **)thee);
00177         (*thee) = VNULL;
00178     }
00179 }
00180
00181 VPUBLIC void PBEparm_dtor2(PBEparm *thee) { ; }
00182
00183 VPUBLIC int PBEparm_check(PBEparm *thee) {
00184
00185     int i;
00186
00187     /* Check to see if we were even filled... */
00188     if (!thee->parsed) {
00189         Vnm_print(2, "PBEparm_check:  not filled!\n");
00190         return 0;
00191     }
00192
00193     if (!thee->setmolid) {
00194         Vnm_print(2, "PBEparm_check:  MOL not set!\n");
00195         return 0;
00196     }
00197     if (!thee->setpbetype) {
00198         Vnm_print(2, "PBEparm_check:  LPBE/NPBE/LRPBE/NRPBE/SMPBE not set!\n");
00199         return 0;
00200     }
00201     if (!thee->setbcfl) {
00202         Vnm_print(2, "PBEparm_check:  BCFL not set!\n");
00203         return 0;
00204     }
00205     if (!thee->setnion) {
00206         thee->setnion = 1;
00207         thee->nion = 0;
00208     }
00209     for (i=0; i<thee->nion; i++) {
00210         if (!thee->setion[i]) {
00211             Vnm_print(2, "PBEparm_check: ION #%d not set!\n",i);
00212             return 0;
00213         }
00214     }
00215     if (!thee->setpdie) {
00216         Vnm_print(2, "PBEparm_check: PDIE not set!\n");
00217         return 0;
00218     }
```

```
00219     if (((thee->srfm==VSM_MOL) || (thee->srfm==VSM_MOLSMOOTH)) \
00220       && (!thee->setsdens) && (thee->srad > VSMALL)) {
00221         Vnm_print(2, "PBEparm_check: SDENS not set!\n");
00222         return 0;
00223     }
00224     if (!thee->setsdie) {
00225         Vnm_print(2, "PBEparm_check: SDIE not set!\n");
00226         return 0;
00227     }
00228     if (!thee->setsrfm) {
00229         Vnm_print(2, "PBEparm_check: SRFM not set!\n");
00230         return 0;
00231     }
00232     if (((thee->srfm==VSM_MOL) || (thee->srfm==VSM_MOLSMOOTH)) \
00233       && (!thee->setsrad)) {
00234         Vnm_print(2, "PBEparm_check: SRAD not set!\n");
00235         return 0;
00236     }
00237     if ((thee->srfm==VSM_SPLINE) && (!thee->setswin)) {
00238         Vnm_print(2, "PBEparm_check: SWIN not set!\n");
00239         return 0;
00240     }
00241     if ((thee->srfm==VSM_SPLINE3) && (!thee->setswin)) {
00242         Vnm_print(2, "PBEparm_check: SWIN not set!\n");
00243         return 0;
00244     }
00245     if ((thee->srfm==VSM_SPLINE4) && (!thee->setswin)) {
00246         Vnm_print(2, "PBEparm_check: SWIN not set!\n");
00247         return 0;
00248     }
00249     if (!thee->settemp) {
00250         Vnm_print(2, "PBEparm_check: TEMP not set!\n");
00251         return 0;
00252     }
00253     if (!thee->setcalcenergy) thee->calcenergy = PCE_NO;
00254     if (!thee->setcalcforce) thee->calcforce = PCF_NO;
00255     if (!thee->setwritemat) thee->writemat = 0;
00256 
00257     /*-----------------------------------------------------------*/
00258     /* Added by Michael Grabe                                    */
00259     /*-----------------------------------------------------------*/
00260 
00261     if ((!thee->setzmem) && (thee->bcfl == 3)){
00262         Vnm_print(2, "PBEparm_check: ZMEM not set!\n");
00263         return 0;
00264     }
00265     if ((!thee->setLmem) && (thee->bcfl == 3)){
00266         Vnm_print(2, "PBEparm_check: LMEM not set!\n");
00267         return 0;
00268     }
00269     if ((!thee->setmdie) && (thee->bcfl == 3)){
00270         Vnm_print(2, "PBEparm_check: MDIE not set!\n");
00271         return 0;
00272     }
00273     if ((!thee->setmemv) && (thee->bcfl == 3)){
00274         Vnm_print(2, "PBEparm_check: MEMV not set!\n");
00275         return 0;
00276     }
00277 
00278     /*-----------------------------------------------------------*/
00279 
00280     return 1;
00281 }
00282 
00283 VPUBLIC void PBEparm_copy(PBEparm *thee, PBEparm *parm) {
00284 
00285     int i, j;
00286 
00287     VASSERT(thee != VNULL);
00288     VASSERT(parm != VNULL);
00289 
00290     thee->molid = parm->molid;
00291     thee->setmolid = parm->setmolid;
00292     thee->useDielMap = parm->useDielMap;
00293     thee->dielMapID = parm->dielMapID;
00294     thee->useKappaMap = parm->useKappaMap;
00295     thee->kappaMapID = parm->kappaMapID;
00296     thee->usePotMap = parm->usePotMap;
00297     thee->potMapID = parm->potMapID;
00298     thee->useChargeMap = parm->useChargeMap;
00299     thee->chargeMapID = parm->chargeMapID;
```

```
00300     thee->pbetype = parm->pbetype;
00301     thee->setpbetype = parm->setpbetype;
00302     thee->bcfl = parm->bcfl;
00303     thee->setbcfl = parm->setbcfl;
00304     thee->nion = parm->nion;
00305     thee->setnion = parm->setnion;
00306     for (i=0; i<MAXION; i++) {
00307         thee->ionq[i] = parm->ionq[i];
00308         thee->ionc[i] = parm->ionc[i];
00309         thee->ionr[i] = parm->ionr[i];
00310         thee->setion[i] = parm->setion[i];
00311     };
00312     thee->pdie = parm->pdie;
00313     thee->setpdie = parm->setpdie;
00314     thee->sdens = parm->sdens;
00315     thee->setsdens = parm->setsdens;
00316     thee->sdie = parm->sdie;
00317     thee->setsdie = parm->setsdie;
00318     thee->srfm = parm->srfm;
00319     thee->setsrfm = parm->setsrfm;
00320     thee->srad = parm->srad;
00321     thee->setsrad = parm->setsrad;
00322     thee->swin = parm->swin;
00323     thee->setswin = parm->setswin;
00324     thee->temp = parm->temp;
00325     thee->settemp = parm->settemp;
00326     thee->calcenergy = parm->calcenergy;
00327     thee->setcalcenergy = parm->setcalcenergy;
00328     thee->calcforce = parm->calcforce;
00329     thee->setcalcforce = parm->setcalcforce;
00330
00331     /*-------------------------------------------------------*/
00332     /* Added by Michael Grabe                                */
00333     /*-------------------------------------------------------*/
00334
00335     thee->zmem = parm->zmem;
00336     thee->setzmem = parm->setzmem;
00337     thee->Lmem = parm->Lmem;
00338     thee->setLmem = parm->setLmem;
00339     thee->mdie = parm->mdie;
00340     thee->setmdie = parm->setmdie;
00341     thee->memv = parm->memv;
00342     thee->setmemv = parm->setmemv;
00343
00344     /*-------------------------------------------------------*/
00345
00346     thee->numwrite = parm->numwrite;
00347     for (i=0; i<PBEPARM_MAXWRITE; i++) {
00348         thee->writetype[i] = parm->writetype[i];
00349         thee->writefmt[i] = parm->writefmt[i];
00350         for (j=0; j<VMAX_ARGLEN; j++)
00351           thee->writestem[i][j] = parm->writestem[i][j];
00352     }
00353     thee->writemat = parm->writemat;
00354     thee->setwritemat = parm->setwritemat;
00355     for (i=0; i<VMAX_ARGLEN; i++) thee->writematstem[i] = parm->writematstem[i];
00356     thee->writematflag = parm->writematflag;
00357
00358     thee->smsize = parm->smsize;
00359     thee->smvolume = parm->smvolume;
00360
00361     thee->setsmsize = parm->setsmsize;
00362     thee->setsmvolume = parm->setsmvolume;
00363
00364     thee->parsed = parm->parsed;
00365
00366 }
00367
00368 VPRIVATE int PBEparm_parseLPBE(PBEparm *thee, Vio *sock) {
00369     Vnm_print(0, "NOsh: parsed lpbe\n");
00370     thee->pbetype = PBE_LPBE;
00371     thee->setpbetype = 1;
00372     return 1;
00373 }
00374
00375 VPRIVATE int PBEparm_parseNPBE(PBEparm *thee, Vio *sock) {
00376     Vnm_print(0, "NOsh: parsed npbe\n");
00377     thee->pbetype = PBE_NPBE;
00378     thee->setpbetype = 1;
00379     return 1;
00380 }
```

```
00381
00382 VPRIVATE int PBEparm_parseMOL(PBEparm *thee, Vio *sock) {
00383     int ti;
00384     char tok[VMAX_BUFSIZE];
00385
00386     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00387     if (sscanf(tok, "%d", &ti) == 0) {
00388         Vnm_print(2, "NOsh:  Read non-int (%s) while parsing MOL \
00389 keyword!\n", tok);
00390         return -1;
00391     }
00392     thee->molid = ti;
00393     thee->setmolid = 1;
00394     return 1;
00395
00396     VERROR1:
00397         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00398         return -1;
00399 }
00400
00401 VPRIVATE int PBEparm_parseLRPBE(PBEparm *thee, Vio *sock) {
00402     Vnm_print(0, "NOsh: parsed lrpbe\n");
00403     thee->pbetype = PBE_LRPBE;
00404     thee->setpbetype = 1;
00405     return 1;
00406 }
00407
00408 VPRIVATE int PBEparm_parseNRPBE(PBEparm *thee, Vio *sock) {
00409     Vnm_print(0, "NOsh: parsed nrpbe\n");
00410     thee->pbetype = PBE_NRPBE;
00411     thee->setpbetype = 1;
00412     return 1;
00413 }
00414
00415 VPRIVATE int PBEparm_parseSMPBE(PBEparm *thee, Vio *sock) {
00416
00417     int i;
00418
00419     char type[VMAX_BUFSIZE]; /* vol or size (keywords) */
00420     char value[VMAX_BUFSIZE]; /* floating point value */
00421
00422     char setVol = 1;
00423     char setSize = 1;
00424     char keyValuePairs = 2;
00425
00426     double size, volume;
00427
00428     for(i=0;i<keyValuePairs;i++){
00429
00430         /* The line two tokens at a time */
00431         VJMPERR1(Vio_scanf(sock, "%s", type) == 1);
00432         VJMPERR1(Vio_scanf(sock, "%s", value) == 1);
00433
00434         if(!strcmp(type,"vol")){
00435             if ((setVol = sscanf(value, "%lf", &volume)) == 0){
00436                 Vnm_print(2,"NOsh:  Read non-float (%s) while parsing smpbe keyword!\n", value);
00437                 return VRC_FAILURE;
00438             }
00439         }else if(!strcmp(type,"size")){
00440             if ((setSize = sscanf(value, "%lf", &size)) == 0){
00441                 Vnm_print(2,"NOsh:  Read non-float (%s) while parsing smpbe keyword!\n", value);
00442                 return VRC_FAILURE;
00443             }
00444         }else{
00445             Vnm_print(2,"NOsh:  Read non-float (%s) while parsing smpbe keyword!\n", value);
00446             return VRC_FAILURE;
00447         }
00448     }
00449
00450     /* If either the volume or size isn't set, throw an error */
00451     if((setVol == 0) || (setSize == 0)){
00452         Vnm_print(2,"NOsh:  Error while parsing smpbe keywords! Only size or vol was specified.\n");
00453         return VRC_FAILURE;
00454     }
00455
00456     Vnm_print(0, "NOsh: parsed smpbe\n");
00457     thee->pbetype = PBE_SMPBE;
00458     thee->setpbetype = 1;
00459
00460     thee->smsize = size;
00461     thee->setsmsize = 1;
```

```
00462
00463     thee->smvolume = volume;
00464     thee->setsmvolume = 1;
00465
00466     return VRC_SUCCESS;
00467
00468 VERROR1:
00469     Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00470     return VRC_FAILURE;
00471
00472 }
00473
00474 VPRIVATE int PBEparm_parseBCFL(PBEparm *thee, Vio *sock) {
00475     char tok[VMAX_BUFSIZE];
00476     int ti;
00477
00478     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00479
00480     /* We can either parse int flag... */
00481     if (sscanf(tok, "%d", &ti) == 1) {
00482
00483         thee->bcfl = (Vbcfl)ti;
00484         thee->setbcfl = 1;
00485         /* Warn that this usage is deprecated */
00486         Vnm_print(2, "parsePBE:  Warning -- parsed deprecated \"bcfl %d\" \
00487 statement\n", ti);
00488         Vnm_print(2, "parsePBE:  Please use \"bcfl ");
00489         switch (thee->bcfl) {
00490             case BCFL_ZERO:
00491                 Vnm_print(2, "zero");
00492                 break;
00493             case BCFL_SDH:
00494                 Vnm_print(2, "sdh");
00495                 break;
00496             case BCFL_MDH:
00497                 Vnm_print(2, "mdh");
00498                 break;
00499             case BCFL_FOCUS:
00500                 Vnm_print(2, "focus");
00501                 break;
00502             case BCFL_MEM:
00503                 Vnm_print(2, "mem");
00504                 break;
00505             case BCFL_MAP:
00506                 Vnm_print(2, "map");
00507                 break;
00508             default:
00509                 Vnm_print(2, "UKNOWN");
00510                 break;
00511         }
00512         Vnm_print(2, "\" instead.\n");
00513         return 1;
00514
00515     /* ...or the word */
00516     } else {
00517
00518         if (Vstring_strcasecmp(tok, "zero") == 0) {
00519             thee->bcfl = BCFL_ZERO;
00520             thee->setbcfl = 1;
00521             return 1;
00522         } else if (Vstring_strcasecmp(tok, "sdh") == 0) {
00523             thee->bcfl = BCFL_SDH;
00524             thee->setbcfl = 1;
00525             return 1;
00526         } else if (Vstring_strcasecmp(tok, "mdh") == 0) {
00527             thee->bcfl = BCFL_MDH;
00528             thee->setbcfl = 1;
00529             return 1;
00530         } else if (Vstring_strcasecmp(tok, "focus") == 0) {
00531             thee->bcfl = BCFL_FOCUS;
00532             thee->setbcfl = 1;
00533             return 1;
00534         } else if (Vstring_strcasecmp(tok, "mem") == 0) {
00535             thee->bcfl = BCFL_MEM;
00536             thee->setbcfl = 1;
00537             return 1;
00538         } else if (Vstring_strcasecmp(tok, "map") == 0) {
00539             thee->bcfl = BCFL_MAP;
00540             thee->setbcfl = 1;
00541             return 1;
00542         } else {
```

```
00543             Vnm_print(2, "NOsh:  parsed unknown BCFL parameter (%s)!\n",
00544               tok);
00545             return -1;
00546         }
00547     }
00548     return 0;
00549
00550     VERROR1:
00551         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00552         return -1;
00553 }
00554
00555 VPRIVATE int PBEparm_parseION(PBEparm *thee, Vio *sock) {
00556
00557     int i;
00558     int meth = 0;
00559
00560     char tok[VMAX_BUFSIZE];
00561     char value[VMAX_BUFSIZE];
00562
00563     double tf;
00564     double charge, conc, radius;
00565
00566     int setCharge = 0;
00567     int setConc = 0;
00568     int setRadius = 0;
00569     int keyValuePairs = 3;
00570
00571     /* Get the initial token for the ION statement */
00572     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00573
00574     /* Scan the token once to determine the type (old style or new keyValue pair) */
00575     meth = sscanf(tok, "%lf", &tf);
00576     /* If tok is a non-zero float value, we are using the old method */
00577     if(meth != 0){
00578
00579         Vnm_print(2, "NOsh:  Deprecated use of ION keyword! Use key-value pairs\n", tok);
00580
00581         if (sscanf(tok, "%lf", &tf) == 0) {
00582             Vnm_print(2, "NOsh:  Read non-float (%s) while parsing ION keyword!\n", tok);
00583             return VRC_FAILURE;
00584         }
00585         thee->ionq[thee->nion] = tf;
00586         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00587         if (sscanf(tok, "%lf", &tf) == 0) {
00588             Vnm_print(2, "NOsh:  Read non-float (%s) while parsing ION keyword!\n", tok);
00589             return VRC_FAILURE;
00590         }
00591         thee->ionc[thee->nion] = tf;
00592         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00593         if (sscanf(tok, "%lf", &tf) == 0) {
00594             Vnm_print(2, "NOsh:  Read non-float (%s) while parsing ION keyword!\n", tok);
00595             return VRC_FAILURE;
00596         }
00597         thee->ionr[thee->nion] = tf;
00598
00599     }else{
00600
00601         /* Three key-value pairs (charge, radius and conc) */
00602         for(i=0;i<keyValuePairs;i++){
00603
00604             /* Now scan for the value (float) to be used with the key token parsed
00605              * above the if-else statement */
00606             VJMPERR1(Vio_scanf(sock, "%s", value) == 1);
00607             if(!strcmp(tok,"charge")){
00608                 setCharge = sscanf(value, "%lf", &charge);
00609                 if (setCharge == 0){
00610                     Vnm_print(2,"NOsh:  Read non-float (%s) while parsing ION %s keyword!\n", value, tok);
00611                     return VRC_FAILURE;
00612                 }
00613                 thee->ionq[thee->nion] = charge;
00614             }else if(!strcmp(tok,"radius")){
00615                 setRadius = sscanf(value, "%lf", &radius);
00616                 if (setRadius == 0){
00617                     Vnm_print(2,"NOsh:  Read non-float (%s) while parsing ION %s keyword!\n", value, tok);
00618                     return VRC_FAILURE;
00619                 }
00620                 thee->ionr[thee->nion] = radius;
00621             }else if(!strcmp(tok,"conc")){
00622                 setConc = sscanf(value, "%lf", &conc);
00623                 if (setConc == 0){
```

```
00624                     Vnm_print(2,"NOsh:  Read non-float (%s) while parsing ION %s keyword!\n", value, tok);
00625                     return VRC_FAILURE;
00626                 }
00627                 thee->ionc[thee->nion] = conc;
00628             }else{
00629                 Vnm_print(2,"NOsh:  Illegal or missing key-value pair for ION keyword!\n");
00630                 return VRC_FAILURE;
00631             }
00632 
00633             /* If all three values haven't be set (setValue = 0) then read the next token */
00634             if((setCharge != 1) || (setConc != 1) || (setRadius != 1)){
00635                 VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00636             }
00637 
00638         } /* end for */
00639     } /* end if */
00640 
00641     /* Finally set the setion, nion and setnion flags and return success */
00642     thee->setion[thee->nion] = 1;
00643     (thee->nion)++;
00644     thee->setnion = 1;
00645     return VRC_SUCCESS;
00646 
00647     VERROR1:
00648         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00649         return VRC_FAILURE;
00650 }
00651 
00652 VPRIVATE int PBEparm_parsePDIE(PBEparm *thee, Vio *sock) {
00653     char tok[VMAX_BUFSIZE];
00654     double tf;
00655 
00656     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00657     if (sscanf(tok, "%lf", &tf) == 0) {
00658         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing PDIE \
00659 keyword!\n", tok);
00660         return -1;
00661     }
00662     thee->pdie = tf;
00663     thee->setpdie = 1;
00664     return 1;
00665 
00666     VERROR1:
00667         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00668         return -1;
00669 }
00670 
00671 VPRIVATE int PBEparm_parseSDENS(PBEparm *thee, Vio *sock) {
00672     char tok[VMAX_BUFSIZE];
00673     double tf;
00674 
00675     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00676     if (sscanf(tok, "%lf", &tf) == 0) {
00677         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SDENS \
00678 keyword!\n", tok);
00679         return -1;
00680     }
00681     thee->sdens = tf;
00682     thee->setsdens = 1;
00683     return 1;
00684 
00685     VERROR1:
00686         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00687         return -1;
00688 }
00689 
00690 VPRIVATE int PBEparm_parseSDIE(PBEparm *thee, Vio *sock) {
00691     char tok[VMAX_BUFSIZE];
00692     double tf;
00693 
00694     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00695     if (sscanf(tok, "%lf", &tf) == 0) {
00696         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SDIE \
00697 keyword!\n", tok);
00698         return -1;
00699     }
00700     thee->sdie = tf;
00701     thee->setsdie = 1;
00702     return 1;
00703 
00704     VERROR1:
```

```
00705         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00706         return -1;
00707 }
00708
00709 VPRIVATE int PBEparm_parseSRFM(PBEparm *thee, Vio *sock) {
00710     char tok[VMAX_BUFSIZE];
00711     int ti;
00712
00713     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00714
00715     /* Parse old-style int arg */
00716     if (sscanf(tok, "%d", &ti) == 1) {
00717         thee->srfm = (Vsurf_Meth)ti;
00718         thee->setsrfm = 1;
00719
00720         Vnm_print(2, "parsePBE:  Warning -- parsed deprecated \"srfm %d\" \
00721 statement.\n", ti);
00722         Vnm_print(2, "parsePBE:  Please use \"srfm ");
00723         switch (thee->srfm) {
00724             case VSM_MOL:
00725                 Vnm_print(2, "mol");
00726                 break;
00727             case VSM_MOLSMOOTH:
00728                 Vnm_print(2, "smol");
00729                 break;
00730             case VSM_SPLINE:
00731                 Vnm_print(2, "spl2");
00732                 break;
00733             case VSM_SPLINE3:
00734                 Vnm_print(2, "spl3");
00735                 break;
00736             case VSM_SPLINE4:
00737                 Vnm_print(2, "spl4");
00738                 break;
00739             default:
00740                 Vnm_print(2, "UNKNOWN");
00741                 break;
00742         }
00743         Vnm_print(2, "\" instead.\n");
00744         return 1;
00745
00746     /* Parse newer text-based args */
00747     } else if (Vstring_strcasecmp(tok, "mol") == 0) {
00748         thee->srfm = VSM_MOL;
00749         thee->setsrfm = 1;
00750         return 1;
00751     } else if (Vstring_strcasecmp(tok, "smol") == 0) {
00752         thee->srfm = VSM_MOLSMOOTH;
00753         thee->setsrfm = 1;
00754         return 1;
00755     } else if (Vstring_strcasecmp(tok, "spl2") == 0) {
00756         thee->srfm = VSM_SPLINE;
00757         thee->setsrfm = 1;
00758         return 1;
00759     } else if (Vstring_strcasecmp(tok, "spl3") == 0) {
00760         thee->srfm = VSM_SPLINE3;
00761         thee->setsrfm = 1;
00762         return 1;
00763     } else if (Vstring_strcasecmp(tok, "spl4") == 0) {
00764         thee->srfm = VSM_SPLINE4;
00765         thee->setsrfm = 1;
00766         return 1;
00767     } else {
00768         Vnm_print(2, "NOsh:  Unrecongnized keyword (%s) when parsing \
00769 srfm!\n", tok);
00770         return -1;
00771     }
00772
00773     return 0;
00774
00775     VERROR1:
00776         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00777         return -1;
00778 }
00779
00780 VPRIVATE int PBEparm_parseSRAD(PBEparm *thee, Vio *sock) {
00781     char tok[VMAX_BUFSIZE];
00782     double tf;
00783
00784     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00785     if (sscanf(tok, "%lf", &tf) == 0) {
```

```
00786         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SRAD \
00787 keyword!\n", tok);
00788         return -1;
00789     }
00790     thee->srad = tf;
00791     thee->setsrad = 1;
00792     return 1;
00793
00794     VERROR1:
00795         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00796         return -1;
00797 }
00798
00799 VPRIVATE int PBEparm_parseSWIN(PBEparm *thee, Vio *sock) {
00800     char tok[VMAX_BUFSIZE];
00801     double tf;
00802
00803     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00804     if (sscanf(tok, "%lf", &tf) == 0) {
00805         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing SWIN \
00806 keyword!\n", tok);
00807         return -1;
00808     }
00809     thee->swin = tf;
00810     thee->setswin = 1;
00811     return 1;
00812
00813     VERROR1:
00814         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00815         return -1;
00816 }
00817
00818 VPRIVATE int PBEparm_parseTEMP(PBEparm *thee, Vio *sock) {
00819     char tok[VMAX_BUFSIZE];
00820     double tf;
00821
00822     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00823     if (sscanf(tok, "%lf", &tf) == 0) {
00824         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing TEMP \
00825 keyword!\n", tok);
00826         return -1;
00827     }
00828     thee->temp = tf;
00829     thee->settemp = 1;
00830     return 1;
00831
00832     VERROR1:
00833         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00834         return -1;
00835 }
00836
00837 VPRIVATE int PBEparm_parseUSEMAP(PBEparm *thee, Vio *sock) {
00838     char tok[VMAX_BUFSIZE];
00839     int ti;
00840
00841     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00842     Vnm_print(0, "PBEparm_parseToken:  Read %s...\n", tok);
00843     if (Vstring_strcasecmp(tok, "diel") == 0) {
00844         thee->useDielMap = 1;
00845         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00846         if (sscanf(tok, "%d", &ti) == 0) {
00847             Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
00848 USEMAP DIEL keyword!\n", tok);
00849             return -1;
00850         }
00851         thee->dielMapID = ti;
00852         return 1;
00853     } else if (Vstring_strcasecmp(tok, "kappa") == 0) {
00854         thee->useKappaMap = 1;
00855         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00856         if (sscanf(tok, "%d", &ti) == 0) {
00857             Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
00858 USEMAP KAPPA keyword!\n", tok);
00859             return -1;
00860         }
00861         thee->kappaMapID = ti;
00862         return 1;
00863     } else if (Vstring_strcasecmp(tok, "pot") == 0) {
00864         thee->usePotMap = 1;
00865         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00866         if (sscanf(tok, "%d", &ti) == 0) {
```

```
00867             Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
00868                       USEMAP POT keyword!\n", tok);
00869             return -1;
00870         }
00871         thee->potMapID = ti;
00872         return 1;
00873     } else if (Vstring_strcasecmp(tok, "charge") == 0) {
00874         thee->useChargeMap = 1;
00875         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00876         if (sscanf(tok, "%d", &ti) == 0) {
00877             Vnm_print(2, "NOsh:  Read non-int (%s) while parsing \
00878 USEMAP CHARGE keyword!\n", tok);
00879             return -1;
00880         }
00881         thee->chargeMapID = ti;
00882         return 1;
00883     } else {
00884         Vnm_print(2, "NOsh:  Read undefined keyword (%s) while parsing \
00885 USEMAP statement!\n", tok);
00886         return -1;
00887     }
00888     return 0;
00889
00890     VERROR1:
00891         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00892         return -1;
00893 }
00894
00895 VPRIVATE int PBEparm_parseCALCENERGY(PBEparm *thee, Vio *sock) {
00896     char tok[VMAX_BUFSIZE];
00897     int ti;
00898
00899     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00900     /* Parse number */
00901     if (sscanf(tok, "%d", &ti) == 1) {
00902         thee->calcenergy = (PBEparm_calcEnergy)ti;
00903         thee->setcalcenergy = 1;
00904
00905         Vnm_print(2, "parsePBE:  Warning -- parsed deprecated \"calcenergy \
00906 %d\" statement.\n", ti);
00907         Vnm_print(2, "parsePBE:  Please use \"calcenergy ");
00908         switch (thee->calcenergy) {
00909             case PCE_NO:
00910                 Vnm_print(2, "no");
00911                 break;
00912             case PCE_TOTAL:
00913                 Vnm_print(2, "total");
00914                 break;
00915             case PCE_COMPS:
00916                 Vnm_print(2, "comps");
00917                 break;
00918             default:
00919                 Vnm_print(2, "UNKNOWN");
00920                 break;
00921         }
00922         Vnm_print(2, "\" instead.\n");
00923         return 1;
00924     } else if (Vstring_strcasecmp(tok, "no") == 0) {
00925         thee->calcenergy = PCE_NO;
00926         thee->setcalcenergy = 1;
00927         return 1;
00928     } else if (Vstring_strcasecmp(tok, "total") == 0) {
00929         thee->calcenergy = PCE_TOTAL;
00930         thee->setcalcenergy = 1;
00931         return 1;
00932     } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00933         thee->calcenergy = PCE_COMPS;
00934         thee->setcalcenergy = 1;
00935         return 1;
00936     } else {
00937         Vnm_print(2, "NOsh:  Unrecognized parameter (%s) while parsing \
00938 calcenergy!\n", tok);
00939         return -1;
00940     }
00941     return 0;
00942
00943     VERROR1:
00944         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00945         return -1;
00946 }
00947
```

```
00948 VPRIVATE int PBEparm_parseCALCFORCE(PBEparm *thee, Vio *sock) {
00949     char tok[VMAX_BUFSIZE];
00950     int ti;
00951
00952     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00953     /* Parse number */
00954     if (sscanf(tok, "%d", &ti) == 1) {
00955         thee->calcforce = (PBEparm_calcForce)ti;
00956         thee->setcalcforce = 1;
00957
00958         Vnm_print(2, "parsePBE:  Warning -- parsed deprecated \"calcforce \
00959 %d\" statement.\n", ti);
00960         Vnm_print(2, "parsePBE:  Please use \"calcforce ");
00961         switch (thee->calcenergy) {
00962             case PCF_NO:
00963                 Vnm_print(2, "no");
00964                 break;
00965             case PCF_TOTAL:
00966                 Vnm_print(2, "total");
00967                 break;
00968             case PCF_COMPS:
00969                 Vnm_print(2, "comps");
00970                 break;
00971             default:
00972                 Vnm_print(2, "UNKNOWN");
00973                 break;
00974         }
00975         Vnm_print(2, "\" instead.\n");
00976         return 1;
00977     } else if (Vstring_strcasecmp(tok, "no") == 0) {
00978         thee->calcforce = PCF_NO;
00979         thee->setcalcforce = 1;
00980         return 1;
00981     } else if (Vstring_strcasecmp(tok, "total") == 0) {
00982         thee->calcforce = PCF_TOTAL;
00983         thee->setcalcforce = 1;
00984         return 1;
00985     } else if (Vstring_strcasecmp(tok, "comps") == 0) {
00986         thee->calcforce = PCF_COMPS;
00987         thee->setcalcforce = 1;
00988         return 1;
00989     } else {
00990         Vnm_print(2, "NOsh:  Unrecognized parameter (%s) while parsing \
00991 calcforce!\n", tok);
00992         return -1;
00993     }
00994     return 0;
00995
00996     VERROR1:
00997         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
00998         return -1;
00999 }
01000
01001 /*-----------------------------------------------------------*/
01002 /* Added by Michael Grabe                                    */
01003 /*-----------------------------------------------------------*/
01004
01005 VPRIVATE int PBEparm_parseZMEM(PBEparm *thee, Vio *sock) {
01006     char tok[VMAX_BUFSIZE];
01007     double tf;
01008
01009     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01010     if (sscanf(tok, "%lf", &tf) == 0) {
01011         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing ZMEM \
01012                   keyword!\n", tok);
01013         return -1;
01014     }
01015     thee->zmem = tf;
01016     thee->setzmem = 1;
01017     return 1;
01018
01019 VERROR1:
01020     Vnm_print(2, "parsePBE:  ran out of tokens!\n");
01021     return -1;
01022 }
01023
01024
01025 VPRIVATE int PBEparm_parseLMEM(PBEparm *thee, Vio *sock) {
01026     char tok[VMAX_BUFSIZE];
01027     double tf;
01028
```

```
01029     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01030     if (sscanf(tok, "%lf", &tf) == 0) {
01031         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing LMEM \
01032                   keyword!\n", tok);
01033         return -1;
01034     }
01035     thee->Lmem = tf;
01036     thee->setLmem = 1;
01037     return 1;
01038
01039 VERROR1:
01040     Vnm_print(2, "parsePBE:  ran out of tokens!\n");
01041     return -1;
01042 }
01043
01044 VPRIVATE int PBEparm_parseMDIE(PBEparm *thee, Vio *sock) {
01045     char tok[VMAX_BUFSIZE];
01046     double tf;
01047
01048     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01049     if (sscanf(tok, "%lf", &tf) == 0) {
01050         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing MDIE \
01051                   keyword!\n", tok);
01052         return -1;
01053     }
01054     thee->mdie = tf;
01055     thee->setmdie  = 1;
01056     return 1;
01057
01058 VERROR1:
01059     Vnm_print(2, "parsePBE:  ran out of tokens!\n");
01060     return -1;
01061 }
01062
01063 VPRIVATE int PBEparm_parseMEMV(PBEparm *thee, Vio *sock) {
01064     char tok[VMAX_BUFSIZE];
01065     double tf;
01066
01067     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01068     if (sscanf(tok, "%lf", &tf) == 0) {
01069         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing MEMV \
01070                   keyword!\n", tok);
01071         return -1;
01072     }
01073     thee->memv = tf;
01074     thee->setmemv = 1;
01075     return 1;
01076
01077 VERROR1:
01078     Vnm_print(2, "parsePBE:  ran out of tokens!\n");
01079     return -1;
01080 }
01081
01082 /*-----------------------------------------------------------*/
01083
01084 VPRIVATE int PBEparm_parseWRITE(PBEparm *thee, Vio *sock) {
01085     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
01086     Vdata_Type writetype;
01087     Vdata_Format writefmt;
01088
01089     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01090     if (Vstring_strcasecmp(tok, "pot") == 0) {
01091         writetype = VDT_POT;
01092     } else if (Vstring_strcasecmp(tok, "atompot") == 0) {
01093         writetype = VDT_ATOMPOT;
01094     }  else if (Vstring_strcasecmp(tok, "charge") == 0) {
01095         writetype = VDT_CHARGE;
01096     } else if (Vstring_strcasecmp(tok, "smol") == 0) {
01097         writetype = VDT_SMOL;
01098     } else if (Vstring_strcasecmp(tok, "dielx") == 0) {
01099         writetype = VDT_DIELX;
01100     } else if (Vstring_strcasecmp(tok, "diely") == 0) {
01101         writetype = VDT_DIELY;
01102     } else if (Vstring_strcasecmp(tok, "dielz") == 0) {
01103         writetype = VDT_DIELZ;
01104     } else if (Vstring_strcasecmp(tok, "kappa") == 0) {
01105         writetype = VDT_KAPPA;
01106     } else if (Vstring_strcasecmp(tok, "sspl") == 0) {
01107         writetype = VDT_SSPL;
01108     } else if (Vstring_strcasecmp(tok, "vdw") == 0) {
01109         writetype = VDT_VDW;
```

```
01110     } else if (Vstring_strcasecmp(tok, "ivdw") == 0) {
01111         writetype = VDT_IVDW;
01112     } else if (Vstring_strcasecmp(tok, "lap") == 0) {
01113         writetype = VDT_LAP;
01114     } else if (Vstring_strcasecmp(tok, "edens") == 0) {
01115         writetype = VDT_EDENS;
01116     } else if (Vstring_strcasecmp(tok, "ndens") == 0) {
01117         writetype = VDT_NDENS;
01118     } else if (Vstring_strcasecmp(tok, "qdens") == 0) {
01119         writetype = VDT_QDENS;
01120     } else if (Vstring_strcasecmp(tok, "3dmap") == 0) {
01121   VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01122   strcpy(thee->pbam_3dmapstem, tok);
01123   thee->pbam_3dmapflag = 1;
01124   return 1;
01125     } else {
01126         Vnm_print(2, "PBEparm_parse:  Invalid data type (%s) to write!\n",
01127            tok);
01128         return -1;
01129     }
01130
01131     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01132     if (Vstring_strcasecmp(tok, "dx") == 0) {
01133         writefmt = VDF_DX;
01134     }
01135     else if(Vstring_strcasecmp(tok, "dxbin") == 0){
01136      writefmt = VDF_DXBIN;
01137     } else if (Vstring_strcasecmp(tok, "uhbd") == 0) {
01138         writefmt = VDF_UHBD;
01139     } else if (Vstring_strcasecmp(tok, "avs") == 0) {
01140         writefmt = VDF_AVS;
01141     } else if (Vstring_strcasecmp(tok, "gz") == 0) {
01142         writefmt = VDF_GZ;
01143     } else if (Vstring_strcasecmp(tok, "flat") == 0) {
01144         writefmt = VDF_FLAT;
01145     } else {
01146         Vnm_print(2, "PBEparm_parse:  Invalid data format (%s) to write!\n",
01147            tok);
01148         return -1;
01149     }
01150     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01151     if (tok[0]=='"') {
01152         strcpy(strnew, "");
01153         while (tok[strlen(tok)-1] != '"') {
01154             strcat(str, tok);
01155             strcat(str, " ");
01156             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01157         }
01158         strcat(str, tok);
01159         strncpy(strnew, str+1, strlen(str)-2);
01160         strcpy(tok, strnew);
01161     }
01162     if (thee->numwrite < (PBEPARM_MAXWRITE-1)) {
01163         strncpy(thee->writestem[thee->numwrite], tok, VMAX_ARGLEN);
01164         thee->writetype[thee->numwrite] = writetype;
01165         thee->writefmt[thee->numwrite] = writefmt;
01166         (thee->numwrite)++;
01167     } else {
01168         Vnm_print(2, "PBEparm_parse:  You have exceeded the maximum number of write statements!\n");
01169         Vnm_print(2, "PBEparm_parse:  Ignoring additional write statements!\n");
01170     }
01171     return 1;
01172
01173     VERROR1:
01174         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
01175         return -1;
01176 }
01177
01178 VPRIVATE int PBEparm_parseWRITEMAT(PBEparm *thee, Vio *sock) {
01179     char tok[VMAX_BUFSIZE], str[VMAX_BUFSIZE]="", strnew[VMAX_BUFSIZE]="";
01180
01181     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01182     if (Vstring_strcasecmp(tok, "poisson") == 0) {
01183         thee->writematflag = 0;
01184     } else if (Vstring_strcasecmp(tok, "full") == 0) {
01185         thee->writematflag = 1;
01186     } else {
01187         Vnm_print(2, "NOsh:  Invalid format (%s) while parsing \
01188 WRITEMAT keyword!\n", tok);
01189         return -1;
01190     }
```

```
01191
01192     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01193     if (tok[0]=='"') {
01194         strcpy(strnew, "");
01195         while (tok[strlen(tok)-1] != '"') {
01196             strcat(str, tok);
01197             strcat(str, " ");
01198             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
01199         }
01200         strcat(str, tok);
01201         strncpy(strnew, str+1, strlen(str)-2);
01202         strcpy(tok, strnew);
01203     }
01204     strncpy(thee->writematstem, tok, VMAX_ARGLEN);
01205     thee->setwritemat = 1;
01206     thee->writemat = 1;
01207     return 1;
01208
01209     VERROR1:
01210         Vnm_print(2, "parsePBE:  ran out of tokens!\n");
01211         return -1;
01212
01213 }
01214
01215 VPUBLIC int PBEparm_parseToken(PBEparm *thee, char tok[VMAX_BUFSIZE],
01216   Vio *sock) {
01217
01218     if (thee == VNULL) {
01219         Vnm_print(2, "parsePBE:  got NULL thee!\n");
01220         return -1;
01221     }
01222     if (sock == VNULL) {
01223         Vnm_print(2, "parsePBE:  got NULL socket!\n");
01224         return -1;
01225     }
01226
01227     Vnm_print(0, "PBEparm_parseToken:  trying %s...\n", tok);
01228
01229     if (Vstring_strcasecmp(tok, "mol") == 0) {
01230         return PBEparm_parseMOL(thee, sock);
01231     } else if (Vstring_strcasecmp(tok, "lpbe") == 0) {
01232         return PBEparm_parseLPBE(thee, sock);
01233     } else if (Vstring_strcasecmp(tok, "npbe") == 0) {
01234         return PBEparm_parseNPBE(thee, sock);
01235     } else if (Vstring_strcasecmp(tok, "lrpbe") == 0) {
01236         return PBEparm_parseLRPBE(thee, sock);
01237     } else if (Vstring_strcasecmp(tok, "nrpbe") == 0) {
01238         return PBEparm_parseNRPBE(thee, sock);
01239     } else if (Vstring_strcasecmp(tok, "smpbe") == 0) {
01240         return PBEparm_parseSMPBE(thee, sock);
01241     } else if (Vstring_strcasecmp(tok, "bcfl") == 0) {
01242         return PBEparm_parseBCFL(thee, sock);
01243     } else if (Vstring_strcasecmp(tok, "ion") == 0) {
01244         return PBEparm_parseION(thee, sock);
01245     } else if (Vstring_strcasecmp(tok, "pdie") == 0) {
01246         return PBEparm_parsePDIE(thee, sock);
01247     } else if (Vstring_strcasecmp(tok, "sdens") == 0) {
01248         return PBEparm_parseSDENS(thee, sock);
01249     } else if (Vstring_strcasecmp(tok, "sdie") == 0) {
01250         return PBEparm_parseSDIE(thee, sock);
01251     } else if (Vstring_strcasecmp(tok, "srfm") == 0) {
01252         return PBEparm_parseSRFM(thee, sock);
01253     } else if (Vstring_strcasecmp(tok, "srad") == 0) {
01254         return PBEparm_parseSRAD(thee, sock);
01255     } else if (Vstring_strcasecmp(tok, "swin") == 0) {
01256         return PBEparm_parseSWIN(thee, sock);
01257     } else if (Vstring_strcasecmp(tok, "temp") == 0) {
01258         return PBEparm_parseTEMP(thee, sock);
01259     } else if (Vstring_strcasecmp(tok, "usemap") == 0) {
01260         return PBEparm_parseUSEMAP(thee, sock);
01261     } else if (Vstring_strcasecmp(tok, "calcenergy") == 0) {
01262         return PBEparm_parseCALCENERGY(thee, sock);
01263     } else if (Vstring_strcasecmp(tok, "calcforce") == 0) {
01264         return PBEparm_parseCALCFORCE(thee, sock);
01265     } else if (Vstring_strcasecmp(tok, "write") == 0) {
01266         return PBEparm_parseWRITE(thee, sock);
01267     } else if (Vstring_strcasecmp(tok, "writemat") == 0) {
01268         return PBEparm_parseWRITEMAT(thee, sock);
01269
01270     /*-----------------------------------------------------------*/
01271     /* Added by Michael Grabe                                    */
```

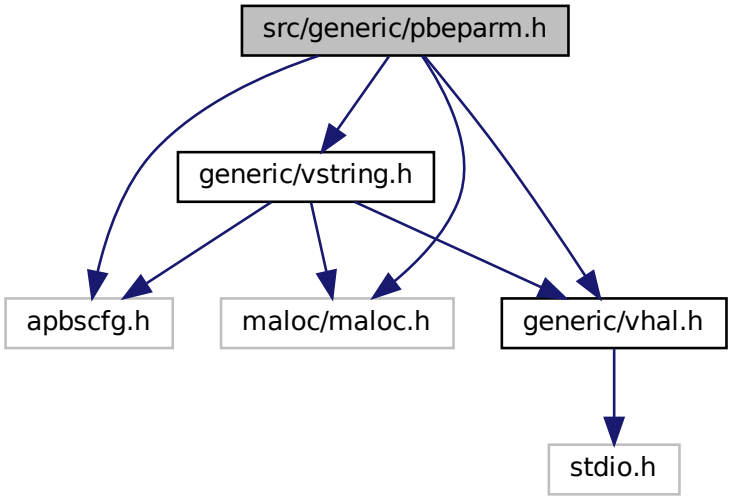
```
01272     /*--------------------------------------------------------------*/
01273
01274     } else if (Vstring_strcasecmp(tok, "zmem") == 0) {
01275         return PBEparm_parseZMEM(thee, sock);
01276     } else if (Vstring_strcasecmp(tok, "Lmem") == 0) {
01277         return PBEparm_parseLMEM(thee, sock);
01278     } else if (Vstring_strcasecmp(tok, "mdie") == 0) {
01279         return PBEparm_parseMDIE(thee, sock);
01280     } else if (Vstring_strcasecmp(tok, "memv") == 0) {
01281         return PBEparm_parseMEMV(thee, sock);
01282     }
01283
01284     /*--------------------------------------------------------------*/
01285
01286     return 0;
01287
01288 }
```

9.41 src/generic/pbeparm.h File Reference

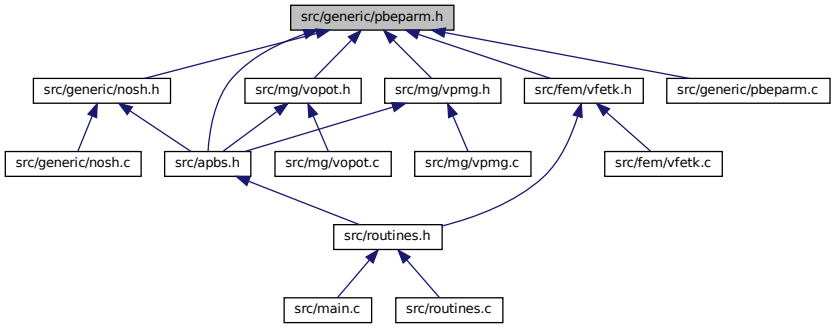
Contains declarations for class PBEparm.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/vstring.h"
```

Include dependency graph for pbeparm.h:



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



Data Structures

- struct sPBEparm

  Parameter structure for PBE variables from input files.

Macros

- #define PBEPARM_MAXWRITE 20

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

Typedefs

- typedef enum ePBEparm_calcEnergy PBEparm_calcEnergy

  Define ePBEparm_calcEnergy enumeration as PBEparm_calcEnergy.
- typedef enum ePBEparm_calcForce PBEparm_calcForce

  Define ePBEparm_calcForce enumeration as PBEparm_calcForce.
- typedef struct sPBEparm PBEparm

  Declaration of the PBEparm class as the PBEparm structure.

Enumerations

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

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

  Define force calculation enumeration.

Functions

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

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

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

  Get radius (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.

9.41.1 Detailed Description

Contains declarations for class PBEparm.

Version

Id

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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.

9.42 pbeparm.h

```
00001
00062 #ifndef _PBEPARM_H_
00063 #define _PBEPARM_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "maloc/maloc.h"
00068
00069 #include "generic/vhal.h"
00070 #include "generic/vstring.h"
00071
00075 #define PBEPARM_MAXWRITE 20
00076
00081 enum ePBEparm_calcEnergy {
00082     PCE_NO=0,
00083     PCE_TOTAL=1,
00084     PCE_COMPS=2
00085 };
00086
00091 typedef enum ePBEparm_calcEnergy PBEparm_calcEnergy;
00092
00097 enum ePBEparm_calcForce {
00098     PCF_NO=0,
00099     PCF_TOTAL=1,
00100     PCF_COMPS=2
00101 };
00102
00107 typedef enum ePBEparm_calcForce PBEparm_calcForce;
00108
00117 struct sPBEparm {
00118
00119     int molid;
00120     int setmolid;
00121     int useDielMap;
00123     int dielMapID;
00124     int useKappaMap;
00126     int kappaMapID;
00127     int usePotMap;
00129     int potMapID;
00131     int useChargeMap;
00133     int chargeMapID;
00134     Vhal_PBEType pbetype;
00135     int setpbetype;
00136     Vbcfl bcfl;
00137     int setbcfl;
00138     int nion;
00139     int setnion;
00140     double ionq[MAXION];
00141     double ionc[MAXION];
00142     double ionr[MAXION];
00143     int setion[MAXION];
00144     double pdie;
00145     int setpdie;
00146     double sdens;
00147     int setsdens;
```

```
00148     double sdie;
00149     int setsdie;
00150     Vsurf_Meth srfm;
00151     int setsrfm;
00152     double srad;
00153     int setsrad;
00154     double swin;
00155     int setswin;
00156     double temp;
00157     int settemp;
00159     double smsize;
00160     int setsmsize;
00162     double smvolume;
00163     int setsmvolume;
00165     PBEparm_calcEnergy calcenergy;
00166     int setcalcenergy;
00167     PBEparm_calcForce calcforce;
00168     int setcalcforce;
00170     /*-----------------------------------------------------------------*/
00171     /* Added by Michael Grabe                                          */
00172     /*-----------------------------------------------------------------*/
00173
00174     double zmem;
00175     int setzmem;
00176     double Lmem;
00177     int setLmem;
00178     double mdie;
00179     int setmdie;
00180     double memv;
00181     int setmemv;
00183     /*-----------------------------------------------------------------*/
00184
00185     int numwrite;
00186     char writestem[PBEPARM_MAXWRITE][VMAX_ARGLEN];
00188     Vdata_Type writetype[PBEPARM_MAXWRITE];
00189     Vdata_Format writefmt[PBEPARM_MAXWRITE];
00191     int writemat;
00194     int setwritemat;
00195     char writematstem[VMAX_ARGLEN];
00196     int writematflag;
00201  /*Added for issue 482*/
00202  char pbam_3dmapstem[VMAX_ARGLEN];
00203  int pbam_3dmapflag;
00204
00205     int parsed;
00207 };
00208
00213 typedef struct sPBEparm PBEparm;
00214
00215 /* ////////////////////////////////////////////////////////////////////////
00216 // Class NOsh: Non-inlineable methods (mcsh.c)
00218
00224 VEXTERNC double PBEparm_getIonCharge(
00225         PBEparm *thee, /**< PBEparm object */
00226         int iion
00227         );
00228
00234 VEXTERNC double PBEparm_getIonConc(
00235         PBEparm *thee,
00236         int iion
00237         );
00238
00244 VEXTERNC double PBEparm_getIonRadius(
00245         PBEparm *thee,
00246         int iion
00247         );
00248
00249
00255 VEXTERNC PBEparm* PBEparm_ctor();
00256
00262 VEXTERNC int PBEparm_ctor2(
00263         PBEparm *thee
00264         );
00265
00270 VEXTERNC void PBEparm_dtor(
00271         PBEparm **thee
00272         );
00273
00278 VEXTERNC void PBEparm_dtor2(
00279         PBEparm *thee
00280         );
```

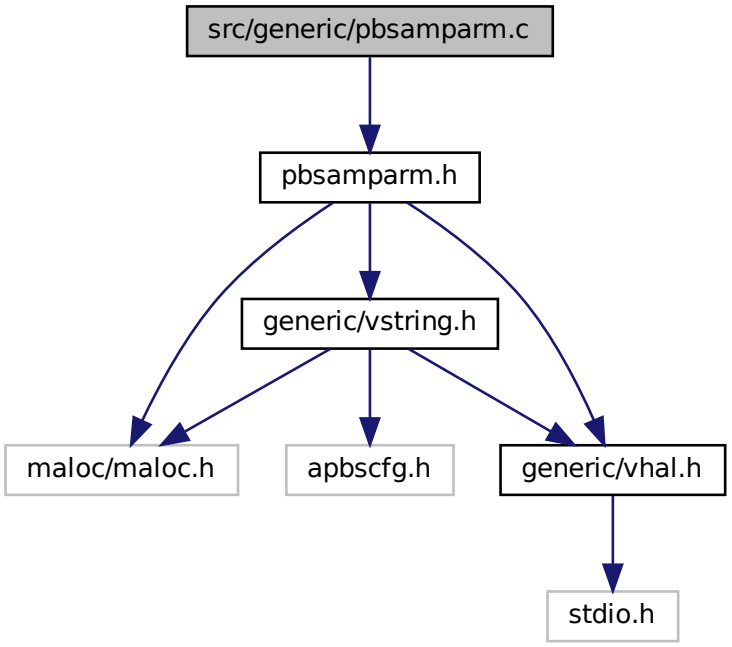
```
00281
00287 VEXTERNC int PBEparm_check(
00288         PBEparm *thee
00289         );
00290
00295 VEXTERNC void PBEparm_copy(
00296         PBEparm *thee,
00297         PBEparm *parm
00298         );
00299
00306 VEXTERNC int PBEparm_parseToken(
00307         PBEparm *thee,
00308         char tok[VMAX_BUFSIZE],
00309         Vio *sock
00310         );
00311
00312
00313 #endif
00314
```

9.43 src/generic/pbsamparm.c File Reference

Class PBSAMparm methods.

`#include "pbsamparm.h"`

Include dependency graph for pbsamparm.c:



Functions

- VPUBLIC PBSAMparm * PBSAMparm_ctor (PBSAMparm_CalcType type)

  Construct PBSAMparm object.
- VPUBLIC Vrc_Codes PBSAMparm_ctor2 (PBSAMparm *thee, PBSAMparm_CalcType type)

FORTRAN stub to construct PBSAMparm object ?????????!!!!!!!

- VPUBLIC void PBSAMparm_dtor (PBSAMparm **thee)

  Object destructor.

- VPUBLIC void PBSAMparm_dtor2 (PBSAMparm *thee)

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

- VPUBLIC Vrc_Codes PBSAMparm_check (PBSAMparm *thee)

  Consistency check for parameter values stored in object.

- VPUBLIC void PBSAMparm_copy (PBSAMparm *thee, PBSAMparm *parm)

  copy PBSAMparm object int thee.

- VPRIVATE Vrc_Codes PBSAMparm_parseSurf (PBSAMparm *thee, Vio *sock)

  Find vertex files for each molecule and save them.

- VPRIVATE Vrc_Codes PBSAMparm_parseMSMS (PBSAMparm *thee, Vio *sock)

  Find msms flag for if MSMS is to be run.

- VPRIVATE Vrc_Codes PBSAMparm_parseImat (PBSAMparm *thee, Vio *sock)

  Find IMAT files for each molecule and save them.

- VPRIVATE Vrc_Codes PBSAMparm_parseExp (PBSAMparm *thee, Vio *sock)

  Find expansion files for each molecule and save them.

- VPRIVATE Vrc_Codes PBSAMparm_parseTolsp (PBSAMparm *thee, Vio *sock)

  Find sphere tolerance for coarse-graining.

- VPUBLIC Vrc_Codes PBSAMparm_parseToken (PBSAMparm *thee, char tok[VMAX_BUFSIZE], Vio *sock)

  Parse an MG keyword from an input file.

9.43.1 Detailed Description

Class PBSAMparm methods.

Author

Andrew Stevens

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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) 2002-2020, 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 pbsamparm.c.

9.44 pbsamparm.c

```
00001
00057 #include "pbsamparm.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_MGPARM)
00062
00063 #endif /* if !defined(VINLINE_MGPARM) */
00064
00065
00066 VPUBLIC PBSAMparm* PBSAMparm_ctor(PBSAMparm_CalcType type) {
00067
00068     /* Set up the structure */
00069     PBSAMparm *thee = VNULL;
00070     thee = (PBSAMparm*)Vmem_malloc(VNULL, 1, sizeof(PBSAMparm));
00071     VASSERT( thee != VNULL);
00072     VASSERT( PBSAMparm_ctor2(thee, type) == VRC_SUCCESS );
00073
00074     return thee;
00075 }
00076
00077 VPUBLIC Vrc_Codes PBSAMparm_ctor2(PBSAMparm *thee, PBSAMparm_CalcType type) {
00078
00079     int i;
00080
00081     if (thee == VNULL) return VRC_FAILURE;
00082
00083     thee->tolsp = 2.5;
00084     thee->setmsms = 0;
00085     thee->probe_radius = 1.5;
00086     thee->density = 3.0;
00087
00088     thee->setsurf = 0;
00089     thee->surfct = 0;
00090
00091     thee->setimat = 0;
00092     thee->imatct = 0;
00093
00094     thee->setexp = 0;
00095     thee->expct = 0;
00096
00097     return VRC_SUCCESS;
00098 }
```

```
00099
00100 VPUBLIC void PBSAMparm_dtor(PBSAMparm **thee) {
00101     if ((*thee) != VNULL) {
00102         PBSAMparm_dtor2(*thee);
00103         Vmem_free(VNULL, 1, sizeof(PBSAMparm), (void **)thee);
00104         (*thee) = VNULL;
00105     }
00106 }
00107
00108 VPUBLIC void PBSAMparm_dtor2(PBSAMparm *thee) { ; }
00109
00110 VPUBLIC Vrc_Codes PBSAMparm_check(PBSAMparm *thee) {
00111
00112     Vrc_Codes rc;
00113
00114     rc = VRC_SUCCESS;
00115
00116     Vnm_print(0, "PBSAMparm_check:  checking PBSAMparm object of type %d.\n",
00117               thee->type);
00118
00119     /* Check to see if we were even filled... */
00120     if (!thee->parsed) {
00121         Vnm_print(2, "PBSAMparm_check:  not filled!\n");
00122         return VRC_FAILURE;
00123     }
00124
00125
00126     /* Check type settings */
00127     if(thee->type != PBSAMCT_AUTO) {
00128          Vnm_print(2,"PBSAMparm_check: type not set");
00129          rc = VRC_FAILURE;
00130     }
00131
00132     return rc;
00133 }
00134
00135 VPUBLIC void PBSAMparm_copy(PBSAMparm *thee, PBSAMparm *parm) {
00136     int i, j;
00137     VASSERT(thee != VNULL);
00138     VASSERT(parm != VNULL);
00139
00140
00141     thee->settolsp = parm->settolsp;
00142     thee->tolsp = parm->tolsp;
00143
00144     thee->setmsms = parm->setmsms;
00145     thee->probe_radius = parm->probe_radius;
00146     thee->density = parm->density;
00147     thee->setsurf = parm->setsurf;
00148     thee->surfct  = parm->surfct;
00149     thee->setimat = parm->setimat;
00150     thee->imatct  = parm->imatct;
00151     thee->setexp  = parm->setexp;
00152     thee->expct   = parm->expct;
00153
00154     for (i=0; i<PBSAMPARM_MAXWRITE; i++)
00155     {
00156         for (j=0; j<CHR_MAXLEN; j++)
00157         {
00158             thee->surffil[i][j] = parm->surffil[i][j];
00159             thee->imatfil[i][j] = parm->imatfil[i][j];
00160             thee->expfil[i][j] = parm->expfil[i][j];
00161         }
00162     }
00163 }
00164
00165 //Parsing vertex file
00166 VPRIVATE Vrc_Codes PBSAMparm_parseSurf(PBSAMparm *thee, Vio *sock){
00167     const char* name = "usemesh";
00168     char tok[VMAX_BUFSIZE];
00169
00170     if(Vio_scanf(sock, "%s", tok) == 0) {
00171         Vnm_print(2, "parsePBSAM:  ran out of tokens on %s!\n", name);
00172         return VRC_WARNING;
00173     } else {
00174        strncpy(thee->surffil[thee->surfct], tok, CHR_MAXLEN);
00175       thee->surfct += 1;
00176     }
00177     return VRC_SUCCESS;
00178 }
00179
```

```
00180
00181 //Parsing imat prefix file
00182 VPRIVATE Vrc_Codes PBSAMparm_parseMSMS(PBSAMparm *thee, Vio *sock){
00183  int td;
00184  char tok[VMAX_BUFSIZE];
00185  const char *name = "mesh";
00186
00187  if(Vio_scanf(sock, "%s", tok) == 0){
00188   Vnm_print(2, "parsePBSAM:  ran out of tokens on %s!\n", name);
00189   return VRC_WARNING;
00190  }
00191
00192  if(strcmp(tok, "msms") == 0){
00193    thee->setmsms = 1;
00194  }
00195  else{
00196    Vnm_print(2, "parsePBSAM: %s is not currently supported in PBSAM! Change to msms\n", tok);
00197    return VRC_WARNING;
00198  }
00199
00200     return VRC_SUCCESS;
00201 }
00202 //Parsing imat prefix file
00203 VPRIVATE Vrc_Codes PBSAMparm_parseImat(PBSAMparm *thee, Vio *sock){
00204     const char* name = "imat";
00205     char tok[VMAX_BUFSIZE];
00206
00207     if(Vio_scanf(sock, "%s", tok) == 0) {
00208         Vnm_print(2, "parsePBSAM:  ran out of tokens on %s!\n", name);
00209         return VRC_WARNING;
00210     } else {
00211        strncpy(thee->imatfil[thee->imatct], tok, CHR_MAXLEN);
00212       thee->imatct += 1;
00213     }
00214     return VRC_SUCCESS;
00215 }
00216
00217 //Parsing imat prefix file
00218 VPRIVATE Vrc_Codes PBSAMparm_parseExp(PBSAMparm *thee, Vio *sock){
00219     const char* name = "exp";
00220     char tok[VMAX_BUFSIZE];
00221
00222     if(Vio_scanf(sock, "%s", tok) == 0) {
00223         Vnm_print(2, "parsePBSAM:  ran out of tokens on %s!\n", name);
00224         return VRC_WARNING;
00225     } else {
00226        strncpy(thee->expfil[thee->expct], tok, CHR_MAXLEN);
00227       thee->expct += 1;
00228     }
00229     return VRC_SUCCESS;
00230 }
00231
00232 VPRIVATE Vrc_Codes PBSAMparm_parseTolsp(PBSAMparm *thee, Vio *sock){
00233     const char* name = "tolsp";
00234     char tok[VMAX_BUFSIZE];
00235     double tf;
00236     if(Vio_scanf(sock, "%s", tok) == 0) {
00237         Vnm_print(2, "parsePBAM:  ran out of tokens on %s!\n", name);
00238         return VRC_WARNING;
00239     }
00240
00241     if (sscanf(tok, "%lf", &tf) == 0){
00242         Vnm_print(2, "NOsh:  Read non-float (%s) while parsing %s keyword!\n", tok, name);
00243         return VRC_WARNING;
00244     }else{
00245         thee->tolsp = tf;
00246     }
00247     thee->settolsp = 1;
00248     return VRC_SUCCESS;
00249 }
00250
00251
00252 VPUBLIC Vrc_Codes PBSAMparm_parseToken(PBSAMparm *thee, char tok[VMAX_BUFSIZE],
00253   Vio *sock) {
00254
00255     if (thee == VNULL) {
00256         Vnm_print(2, "parsePBSAM:  got NULL thee!\n");
00257         return VRC_WARNING;
00258     }
00259     if (sock == VNULL) {
00260         Vnm_print(2, "parsePBSAM:  got NULL socket!\n");
```

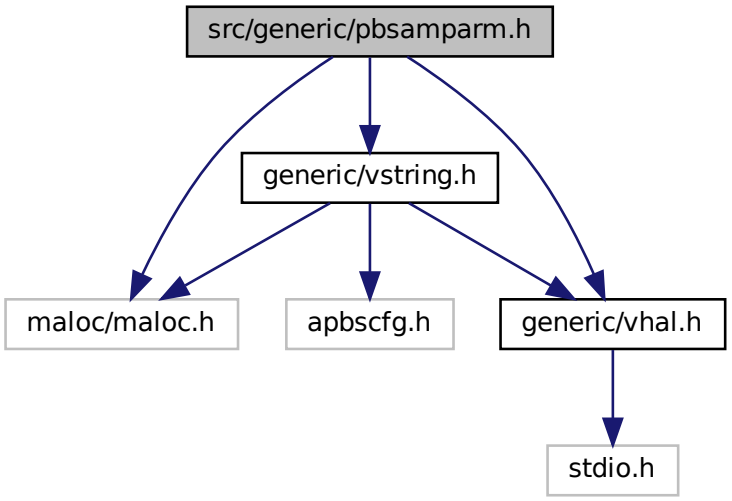
```
00261         return VRC_WARNING;
00262     }
00263
00264     Vnm_print(0, "PBSAMparm_parseToken:  trying %s...\n", tok);
00265
00266     // Molecule terms
00267     if (Vstring_strcasecmp(tok, "usemesh") == 0) {
00268         return PBSAMparm_parseSurf(thee, sock);
00269     }else if (Vstring_strcasecmp(tok, "mesh") == 0) {
00270         return PBSAMparm_parseMSMS(thee, sock);
00271     }else if (Vstring_strcasecmp(tok, "imat") == 0) {
00272         return PBSAMparm_parseImat(thee, sock);
00273     }else if (Vstring_strcasecmp(tok, "exp") == 0) {
00274         return PBSAMparm_parseExp(thee, sock);
00275     }else if (Vstring_strcasecmp(tok, "tolsp") == 0) {
00276         return PBSAMparm_parseTolsp(thee, sock);
00277     }
00278
00279     else {
00280         Vnm_print(2, "parsePBSAM:  Unrecognized keyword (%s)!\n", tok);
00281         return VRC_WARNING;
00282     }
00283     return VRC_FAILURE;
00284 }
```

9.45 src/generic/pbsamparm.h File Reference

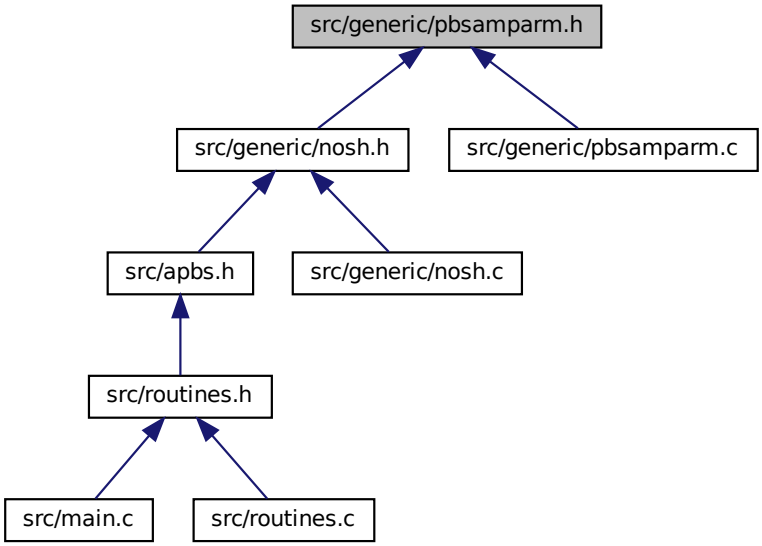
Contains declarations for class PBSAMparm.

```
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/vstring.h"
```

Include dependency graph for pbsamparm.h:



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



Data Structures

- struct sPBSAMparm

  Parameter structure for PBSAM-specific variables from input files.

Macros

- #define CHR_MAXLEN 1000

  Number of things that can be written out in a single calculation.
- #define **PBSAMPARM_MAXWRITE** 15
- #define **PBSAMPARM_MAXMOL** 150

Typedefs

- typedef enum ePBSAMparm_CalcType PBSAMparm_CalcType

  Declare PBSAMparm_CalcType type.
- typedef struct sPBSAMparm PBSAMparm

  Parameter structure for PBSAM-specific variables from input files.

Enumerations

- enum ePBSAMparm_CalcType { PBSAMCT_AUTO =1 }

  Calculation type.

Functions

- VEXTERNC PBSAMparm * PBSAMparm_ctor (PBSAMparm_CalcType type)

  Construct PBSAMparm object.
- VEXTERNC Vrc_Codes PBSAMparm_ctor2 (PBSAMparm *thee, PBSAMparm_CalcType type)

  FORTRAN stub to construct PBSAMparm object ?????????!!!!!!!
- VEXTERNC void PBSAMparm_dtor (PBSAMparm **thee)

  Object destructor.
- VEXTERNC void PBSAMparm_dtor2 (PBSAMparm *thee)

  FORTRAN stub for object destructor ?????????!!!!!!!!!!!
- VEXTERNC Vrc_Codes PBSAMparm_check (PBSAMparm *thee)

  Consistency check for parameter values stored in object.
- VEXTERNC Vrc_Codes PBSAMparm_parseToken (PBSAMparm *thee, char tok[VMAX_BUFSIZE], Vio *sock)

  Parse an MG keyword from an input file.
- VEXTERNC void PBSAMparm_copy (PBSAMparm *thee, PBSAMparm *parm)

  copy PBSAMparm object int thee.
- VPRIVATE Vrc_Codes PBSAMparm_parseTolsp (PBSAMparm *thee, Vio *sock)

  Find sphere tolerance for coarse-graining.
- VPRIVATE Vrc_Codes PBSAMparm_parseSurf (PBSAMparm *thee, Vio *sock)

  Find vertex files for each molecule and save them.
- VPRIVATE Vrc_Codes PBSAMparm_parseImat (PBSAMparm *thee, Vio *sock)

  Find IMAT files for each molecule and save them.
- VPRIVATE Vrc_Codes PBSAMparm_parseExp (PBSAMparm *thee, Vio *sock)

  Find expansion files for each molecule and save them.
- VPRIVATE Vrc_Codes PBSAMparm_parseMSMS (PBSAMparm *thee, Vio *sock)

  Find msms flag for if MSMS is to be run.

9.45.1 Detailed Description

Contains declarations for class PBSAMparm.

Version

Id

Author

Lisa Felberg

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) 2010-2020 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.
```

Definition in file pbsamparm.h.

9.46 pbsamparm.h

```
00001
00064 #ifndef _PBSAMPARM_H_
00065 #define _PBSAMPARM_H_
00066
00067 /* Generic header files */
00068 #include "maloc/maloc.h"
00069
00070 #include "generic/vhal.h"
00071 #include "generic/vstring.h"
00072
00076 #define CHR_MAXLEN 1000
00077 #define PBSAMPARM_MAXWRITE 15
00078 #define PBSAMPARM_MAXMOL 150
00079
00084 enum ePBSAMparm_CalcType {
00085     //other methods disabled for now only auto currently implemented.
00086  //PBSAMCT_MANUAL=0,  /**< PBSAM-manual */
00087     PBSAMCT_AUTO=1,
00088     //PBSAMCT_NONE=2 /**< not defined */
00089 };
00090
00095 typedef enum ePBSAMparm_CalcType PBSAMparm_CalcType;
00096
00105 typedef struct sPBSAMparm {
00106
00107     PBSAMparm_CalcType type;
00108     int parsed;
00110     /* The only parms in addition to PBAM would be MSMS
00111        IMAT and Selfpol */
00112     int settolsp;
00113     double tolsp;
00114
00115     int setmsms;
00116     double probe_radius;
00117     double density;
```

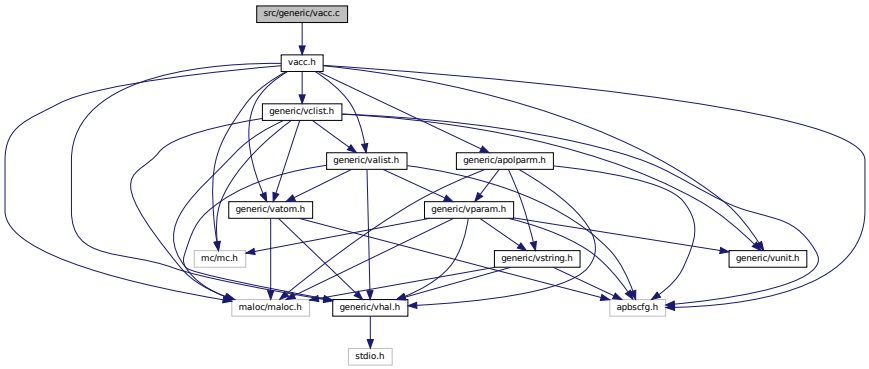
```
00118
00119     int setsurf;
00120     int surfct;
00121     char surffil[PBSAMPARM_MAXMOL][CHR_MAXLEN];
00122
00123     int setimat;
00124     int imatct;
00125     char imatfil[PBSAMPARM_MAXMOL][CHR_MAXLEN];
00126
00127     int setexp;
00128     int expct;
00129     char expfil[PBSAMPARM_MAXMOL][CHR_MAXLEN];
00130
00131 } PBSAMparm;
00132
00139 VEXTERNC PBSAMparm* PBSAMparm_ctor(PBSAMparm_CalcType type);
00140
00148 VEXTERNC Vrc_Codes PBSAMparm_ctor2(PBSAMparm *thee, PBSAMparm_CalcType type);
00149
00155 VEXTERNC void PBSAMparm_dtor(PBSAMparm **thee);
00156
00162 VEXTERNC void PBSAMparm_dtor2(PBSAMparm *thee);
00163
00170 VEXTERNC Vrc_Codes PBSAMparm_check(PBSAMparm *thee);
00171
00181 VEXTERNC Vrc_Codes PBSAMparm_parseToken(PBSAMparm *thee, char tok[VMAX_BUFSIZE],
00182                                         Vio *sock);
00190 VEXTERNC void PBSAMparm_copy(PBSAMparm *thee, PBSAMparm *parm);
00191
00199 VPRIVATE Vrc_Codes PBSAMparm_parseTolsp(PBSAMparm *thee, Vio *sock);
00200
00208 VPRIVATE Vrc_Codes PBSAMparm_parseSurf(PBSAMparm *thee, Vio *sock);
00209
00217 VPRIVATE Vrc_Codes PBSAMparm_parseImat(PBSAMparm *thee, Vio *sock);
00218
00226 VPRIVATE Vrc_Codes PBSAMparm_parseExp(PBSAMparm *thee, Vio *sock);
00227
00235 VPRIVATE Vrc_Codes PBSAMparm_parseMSMS(PBSAMparm *thee, Vio *sock);
00236
00237 #endif
00238
```

9.47 src/generic/vacc.c File Reference

Class Vacc methods.

`#include "vacc.h"`

Include dependency graph for vacc.c:



Functions

- VPUBLIC unsigned long int Vacc_memChk (Vacc *thee)

Get number of bytes in this object and its members.

- VPRIVATE int ivdwAccExclus (Vacc *thee, double center[3], double radius, int atomID)

  Determines if a point is within the union of the spheres centered at the atomic centers with radii equal to the sum of their van der Waals radii and the probe radius. Does not include contributions from the specified atom.
- VPUBLIC Vacc * Vacc_ctor (Valist *alist, Vclist *clist, double surf_density)

  Construct the accessibility object.
- VPRIVATE int Vacc_storeParms (Vacc *thee, Valist *alist, Vclist *clist, double surf_density)
- VPRIVATE int Vacc_allocate (Vacc *thee)
- VPUBLIC int Vacc_ctor2 (Vacc *thee, Valist *alist, Vclist *clist, double surf_density)

  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.

9.47.1 Detailed Description

Class Vacc methods.

Author

Nathan Baker

Version

Id

Attention

Definition in file vacc.c.

9.47.2 Function Documentation

9.47.2.1 ivdwAccExclus()

```
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 80 of file vacc.c.

9.47.2.2 splineAcc()

```
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 493 of file vacc.c.

9.47.2.3 Vacc_allocate()

```
VPRIVATE int Vacc_allocate (
            Vacc * thee )
```

Allocate (and clear) space for storage
Definition at line 193 of file vacc.c.

9.47.2.4 Vacc_storeParms()

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

Check and store parameters passed to constructor
Definition at line 148 of file vacc.c.

9.48 vacc.c

```
00001
00057 #include "vacc.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_VACC)
00062
00063 VPUBLIC unsigned long int Vacc_memChk(Vacc *thee) {
00064     if (thee == VNULL)
00065         return 0;
00066     return Vmem_bytes(thee->mem);
00067 }
00068
00069 #endif /* if !defined(VINLINE_VACC) */
00070
00080 VPRIVATE int ivdwAccExclus(
00081                            Vacc *thee,
00082                            double center[3],
00083                            double radius,
00084                            int atomID
00085                            ) {
00086
00087     int iatom;
00088     double dist2,
00089            *apos;
00090     Vatom *atom;
00091     VclistCell *cell;
00092
00093     VASSERT(thee != VNULL);
00094
00095     /* We can only test probes with radii less than the max specified */
00096     if (radius > Vclist_maxRadius(thee->clist)) {
00097         Vnm_print(2,
00098                   "Vacc_ivdwAcc: got radius (%g) bigger than max radius (%g)\n",
00099                   radius, Vclist_maxRadius(thee->clist));
00100         VASSERT(0);
00101     }
00102
00103     /* Get the relevant cell from the cell list */
00104     cell = Vclist_getCell(thee->clist, center);
00105
00106     /* If we have no cell, then no atoms are nearby and we're definitely
00107      * accessible */
00108     if (cell == VNULL) {
00109         return 1;
00110     }
00111
00112     /* Otherwise, check for overlap with the atoms in the cell */
00113     for (iatom=0; iatom<cell->natoms; iatom++) {
00114         atom = cell->atoms[iatom];
00115
00116         // We don't actually need to test this if the atom IDs do match; don't compute this if we're
       comparing atom against itself.
00117         if (atom->id == atomID) continue;
00118
00119         apos = atom->position;
00120         dist2 = VSQR(center[0]-apos[0]) + VSQR(center[1]-apos[1])
00121                       + VSQR(center[2]-apos[2]);
00122         if (dist2 < VSQR(atom->radius+radius)){
00123             return 0;
00124         }
00125     }
00126
```

```
00127     /* If we're still here, then the point is accessible */
00128     return 1;
00129
00130 }
00131
00132 VPUBLIC Vacc* Vacc_ctor(Valist *alist,
00133                         Vclist *clist,
00134                         double surf_density /* Surface density */
00135                         ) {
00136
00137
00138     Vacc *thee = VNULL;
00139
00140     /* Set up the structure */
00141     thee = (Vacc*)Vmem_malloc(VNULL, 1, sizeof(Vacc) );
00142     VASSERT( thee != VNULL);
00143     VASSERT( Vacc_ctor2(thee, alist, clist, surf_density));
00144     return thee;
00145 }
00146
00148 VPRIVATE int Vacc_storeParms(Vacc *thee,
00149                              Valist *alist,
00150                              Vclist *clist,
00151                              double surf_density /* Surface density */
00152                              ) {
00153
00154     int nsphere,
00155         iatom;
00156     double maxrad = 0.0,
00157            maxarea,
00158            rad;
00159     Vatom *atom;
00160
00161     if (alist == VNULL) {
00162         Vnm_print(2, "Vacc_storeParms:  Got NULL Valist!\n");
00163         return 0;
00164     } else thee->alist = alist;
00165     if (clist == VNULL) {
00166         Vnm_print(2, "Vacc_storeParms:  Got NULL Vclist!\n");
00167         return 0;
00168     } else thee->clist = clist;
00169     thee->surf_density = surf_density;
00170
00171     /* Loop through the atoms to determine the maximum radius */
00172     maxrad = 0.0;
00173     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
00174         atom = Valist_getAtom(alist, iatom);
00175         rad = Vatom_getRadius(atom);
00176         if (rad > maxrad) maxrad = rad;
00177     }
00178     maxrad = maxrad + Vclist_maxRadius(thee->clist);
00179
00180     maxarea = 4.0*VPI*maxrad*maxrad;
00181     nsphere = (int)ceil(maxarea*surf_density);
00182
00183     Vnm_print(0, "Vacc_storeParms:  Surf. density = %g\n", surf_density);
00184     Vnm_print(0, "Vacc_storeParms:  Max area = %g\n", maxarea);
00185     thee->refSphere = VaccSurf_refSphere(thee->mem, nsphere);
00186     Vnm_print(0, "Vacc_storeParms:  Using %d-point reference sphere\n",
00187               thee->refSphere->npts);
00188
00189     return 1;
00190 }
00191
00193 VPRIVATE int Vacc_allocate(Vacc *thee) {
00194
00195     int i,
00196         natoms;
00197
00198     natoms = Valist_getNumberAtoms(thee->alist);
00199
00200     thee->atomFlags = (int*)Vmem_malloc(thee->mem, natoms, sizeof(int));
00201     if (thee->atomFlags == VNULL) {
00202         Vnm_print(2,
00203                 "Vacc_allocate:  Failed to allocate %d (int)s for atomFlags!\n",
00204                  natoms);
00205         return 0;
00206     }
00207     for (i=0; i<natoms; i++) (thee->atomFlags)[i] = 0;
00208
00209     return 1;
```

```
00210 }
00211
00212
00213 VPUBLIC int Vacc_ctor2(Vacc *thee,
00214                        Valist *alist,
00215                        Vclist *clist,
00216                        double surf_density
00217                        ) {
00218
00219     /* Check and store parameters */
00220     if (!Vacc_storeParms(thee, alist, clist, surf_density)) {
00221         Vnm_print(2, "Vacc_ctor2:  parameter check failed!\n");
00222         return 0;
00223     }
00224
00225     /* Set up memory management object */
00226     thee->mem = Vmem_ctor("APBS::VACC");
00227     if (thee->mem == VNULL) {
00228         Vnm_print(2, "Vacc_ctor2:  memory object setup failed!\n");
00229         return 0;
00230     }
00231
00232     /* Setup and check probe */
00233     thee->surf = VNULL;
00234
00235     /* Allocate space */
00236     if (!Vacc_allocate(thee)) {
00237         Vnm_print(2, "Vacc_ctor2:  memory allocation failed!\n");
00238         return 0;
00239     }
00240
00241     return 1;
00242 }
00243
00244
00245 VPUBLIC void Vacc_dtor(Vacc **thee) {
00246
00247     if ((*thee) != VNULL) {
00248         Vacc_dtor2(*thee);
00249         Vmem_free(VNULL, 1, sizeof(Vacc), (void **)thee);
00250         (*thee) = VNULL;
00251     }
00252
00253 }
00254
00255 VPUBLIC void Vacc_dtor2(Vacc *thee) {
00256
00257     int i,
00258         natoms;
00259
00260     natoms = Valist_getNumberAtoms(thee->alist);
00261     Vmem_free(thee->mem, natoms, sizeof(int), (void **)&(thee->atomFlags));
00262
00263     if (thee->refSphere != VNULL) {
00264         VaccSurf_dtor(&(thee->refSphere));
00265         thee->refSphere = VNULL;
00266     }
00267     if (thee->surf != VNULL) {
00268         for (i=0; i<natoms; i++) VaccSurf_dtor(&(thee->surf[i]));
00269         Vmem_free(thee->mem, natoms, sizeof(VaccSurf *),
00270                   (void **)&(thee->surf));
00271         thee->surf = VNULL;
00272     }
00273
00274     Vmem_dtor(&(thee->mem));
00275 }
00276
00277 VPUBLIC double Vacc_vdwAcc(Vacc *thee,
00278                            double center[3]
00279                            ) {
00280
00281     VclistCell *cell;
00282     Vatom *atom;
00283     int iatom;
00284     double *apos,
00285            dist2;
00286
00287     /* Get the relevant cell from the cell list */
00288     cell = Vclist_getCell(thee->clist, center);
00289
00290     /* If we have no cell, then no atoms are nearby and we're definitely
```

```
00291      * accessible */
00292     if (cell == VNULL) return 1.0;
00293
00294     /* Otherwise, check for overlap with the atoms in the cell */
00295     for (iatom=0; iatom<cell->natoms; iatom++) {
00296         atom = cell->atoms[iatom];
00297         apos = Vatom_getPosition(atom);
00298         dist2 = VSQR(center[0]-apos[0]) + VSQR(center[1]-apos[1])
00299               + VSQR(center[2]-apos[2]);
00300         if (dist2 < VSQR(Vatom_getRadius(atom))) return 0.0;
00301     }
00302
00303     /* If we're still here, then the point is accessible */
00304     return 1.0;
00305 }
00306
00307 VPUBLIC double Vacc_ivdwAcc(Vacc *thee,
00308                             double center[3],
00309                             double radius
00310                             ) {
00311
00312     return (double)ivdwAccExclus(thee, center, radius, -1);
00313
00314 }
00315
00316 VPUBLIC void Vacc_splineAccGradAtomNorm(Vacc *thee,
00317                                         double center[VAPBS_DIM],
00318                                         double win,
00319                                         double infrad,
00320                                         Vatom *atom,
00321                                         double *grad
00322                                         ) {
00323
00324     int i;
00325     double dist,
00326            *apos,
00327            arad,
00328            sm,
00329            sm2,
00330            w2i, /* inverse of win squared */
00331            w3i, /* inverse of win cubed */
00332            mygrad,
00333            mychi = 1.0;           /* Char. func. value for given atom */
00334
00335     VASSERT(thee != NULL);
00336
00337     /* Inverse squared window parameter */
00338     w2i = 1.0/(win*win);
00339     w3i = 1.0/(win*win*win);
00340
00341     /* The grad is zero by default */
00342     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
00343
00344     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
00345      * *** MAGNITUDE OF THE FORCE *** */
00346     apos = Vatom_getPosition(atom);
00347     /* Zero-radius atoms don't contribute */
00348     if (Vatom_getRadius(atom) > 0.0) {
00349         arad = Vatom_getRadius(atom) + infrad;
00350         dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
00351           + VSQR(apos[2]-center[2]));
00352         /* If we're inside an atom, the entire characteristic function
00353          * will be zero and the grad will be zero, so we can stop */
00354         if (dist < (arad - win)) return;
00355         /* Likewise, if we're outside the smoothing window, the characteristic
00356          * function is unity and the grad will be zero, so we can stop */
00357         else if (dist > (arad + win)) return;
00358         /* Account for floating point error at the border
00359          * NAB:  COULDN'T THESE TESTS BE COMBINED AS BELOW
00360          * (Vacc_splineAccAtom)? */
00361         else if ((VABS(dist - (arad - win)) < VSMALL) ||
00362                  (VABS(dist - (arad + win)) < VSMALL)) return;
00363         /* If we're inside the smoothing window */
00364         else {
00365             sm = dist - arad + win;
00366             sm2 = VSQR(sm);
00367             mychi = 0.75*sm2*w2i -0.25*sm*sm2*w3i;
00368             mygrad = 1.5*sm*w2i - 0.75*sm2*w3i;
00369         }
00370         /* Now assemble the grad vector */
00371         VASSERT(mychi > 0.0);
```

```
00372         for (i=0; i<VAPBS_DIM; i++)
00373             grad[i] = -(mygrad/mychi)*((center[i] - apos[i])/dist);
00374     }
00375 }
00376
00377 VPUBLIC void Vacc_splineAccGradAtomUnnorm(Vacc *thee,
00378                                           double center[VAPBS_DIM],
00379                                           double win,
00380                                           double infrad,
00381                                           Vatom *atom,
00382                                           double *grad
00383                                           ) {
00384
00385     int i;
00386     double dist,
00387            *apos,
00388            arad,
00389            sm,
00390            sm2,
00391            w2i, /* Inverse of win squared */
00392            w3i, /* Inverse of win cubed */
00393            mygrad,
00394            mychi = 1.0;           /* Char. func. value for given atom */
00395
00396     VASSERT(thee != NULL);
00397
00398     /* Inverse squared window parameter */
00399     w2i = 1.0/(win*win);
00400     w3i = 1.0/(win*win*win);
00401
00402     /* The grad is zero by default */
00403     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
00404
00405     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
00406      * *** MAGNITUDE OF THE FORCE *** */
00407     apos = Vatom_getPosition(atom);
00408     /* Zero-radius atoms don't contribute */
00409     if (Vatom_getRadius(atom) > 0.0) {
00410         arad = Vatom_getRadius(atom) + infrad;
00411         dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
00412           + VSQR(apos[2]-center[2]));
00413         /* If we're inside an atom, the entire characteristic function
00414          * will be zero and the grad will be zero, so we can stop */
00415         if (dist < (arad - win)) return;
00416         /* Likewise, if we're outside the smoothing window, the characteristic
00417          * function is unity and the grad will be zero, so we can stop */
00418         else if (dist > (arad + win)) return;
00419         /* Account for floating point error at the border
00420          * NAB:  COULDN'T THESE TESTS BE COMBINED AS BELOW
00421          * (Vacc_splineAccAtom)? */
00422         else if ((VABS(dist - (arad - win)) < VSMALL) ||
00423                  (VABS(dist - (arad + win)) < VSMALL)) return;
00424         /* If we're inside the smoothing window */
00425         else {
00426             sm = dist - arad + win;
00427             sm2 = VSQR(sm);
00428             mychi = 0.75*sm2*w2i -0.25*sm*sm2*w3i;
00429             mygrad = 1.5*sm*w2i - 0.75*sm2*w3i;
00430         }
00431         /* Now assemble the grad vector */
00432         VASSERT(mychi > 0.0);
00433         for (i=0; i<VAPBS_DIM; i++)
00434             grad[i] = -(mygrad)*((center[i] - apos[i])/dist);
00435     }
00436 }
00437
00438 VPUBLIC double Vacc_splineAccAtom(Vacc *thee,
00439                                   double center[VAPBS_DIM],
00440                                   double win,
00441                                   double infrad,
00442                                   Vatom *atom
00443                                   ) {
00444
00445     double dist,
00446            *apos,
00447            arad,
00448            sm,
00449            sm2,
00450            w2i, /* Inverse of win squared */
00451            w3i, /* Inverse of win cubed */
00452            value,
```

```
00453             stot,
00454             sctot;
00455
00456     VASSERT(thee != NULL);
00457
00458     /* Inverse squared window parameter */
00459     w2i = 1.0/(win*win);
00460     w3i = 1.0/(win*win*win);
00461
00462     apos = Vatom_getPosition(atom);
00463     /* Zero-radius atoms don't contribute */
00464     if (Vatom_getRadius(atom) > 0.0) {
00465         arad = Vatom_getRadius(atom) + infrad;
00466         stot = arad + win;
00467         sctot = VMAX2(0, (arad - win));
00468         dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
00469           + VSQR(apos[2]-center[2]));
00470         /* If we're inside an atom, the entire characteristic function
00471          * will be zero */
00472         if ((dist < sctot) || (VABS(dist - sctot) < VSMALL)){
00473             value = 0.0;
00474         /* We're outside the smoothing window */
00475         } else if ((dist > stot) || (VABS(dist - stot) < VSMALL)) {
00476             value = 1.0;
00477         /* We're inside the smoothing window */
00478         } else {
00479             sm = dist - arad + win;
00480             sm2 = VSQR(sm);
00481             value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
00482         }
00483     } else value = 1.0;
00484
00485     return value;
00486 }
00487
00493 VPRIVATE double splineAcc(
00494         Vacc *thee,
00495         double center[VAPBS_DIM],
00497         double win,
00498         double infrad,
00499         VclistCell *cell
00500         ) {
00501
00502     int atomID, iatom;
00503     Vatom *atom;
00504     double value = 1.0;
00505
00506     VASSERT(thee != NULL);
00507
00508     /* Now loop through the atoms assembling the characteristic function */
00509     for (iatom=0; iatom<cell->natoms; iatom++) {
00510
00511         atom = cell->atoms[iatom];
00512         atomID = atom->id;
00513
00514         /* Check to see if we've counted this atom already */
00515         if ( !(thee->atomFlags[atomID]) ) {
00516
00517             thee->atomFlags[atomID] = 1;
00518             value *= Vacc_splineAccAtom(thee, center, win, infrad, atom);
00519
00520             if (value < VSMALL) return value;
00521         }
00522     }
00523
00524     return value;
00525 }
00526
00527
00528 VPUBLIC double Vacc_splineAcc(Vacc *thee, double center[VAPBS_DIM], double win,
00529   double infrad) {
00530
00531     VclistCell *cell;
00532     Vatom *atom;
00533     int iatom, atomID;
00534
00535
00536     VASSERT(thee != NULL);
00537
00538     if (Vclist_maxRadius(thee->clist) < (win + infrad)) {
00539         Vnm_print(2, "Vacc_splineAcc:  Vclist has max_radius=%g;\n",
```

```
00540                 Vclist_maxRadius(thee->clist));
00541         Vnm_print(2, "Vacc_splineAcc:  Insufficient for win=%g, infrad=%g\n",
00542                 win, infrad);
00543         VASSERT(0);
00544     }
00545
00546     /* Get a cell or VNULL; in the latter case return 1.0 */
00547     cell = Vclist_getCell(thee->clist, center);
00548     if (cell == VNULL) return 1.0;
00549
00550     /* First, reset the list of atom flags
00551      * NAB:  THIS SEEMS VERY INEFFICIENT */
00552     for (iatom=0; iatom<cell->natoms; iatom++) {
00553         atom = cell->atoms[iatom];
00554         atomID = atom->id;
00555         thee->atomFlags[atomID] = 0;
00556     }
00557
00558     return splineAcc(thee, center, win, infrad, cell);
00559 }
00560
00561 VPUBLIC void Vacc_splineAccGrad(Vacc *thee, double center[VAPBS_DIM],
00562         double win, double infrad, double *grad) {
00563
00564     int iatom, i, atomID;
00565     double acc = 1.0;
00566     double tgrad[VAPBS_DIM];
00567     VclistCell *cell;
00568     Vatom *atom = VNULL;
00569
00570     VASSERT(thee != NULL);
00571
00572     if (Vclist_maxRadius(thee->clist) < (win + infrad)) {
00573         Vnm_print(2, "Vacc_splineAccGrad: Vclist max_radius=%g;\n",
00574                 Vclist_maxRadius(thee->clist));
00575         Vnm_print(2, "Vacc_splineAccGrad: Insufficient for win=%g, infrad=%g\n",
00576                 win, infrad);
00577         VASSERT(0);
00578     }
00579
00580     /* Reset the gradient */
00581     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
00582
00583     /* Get the cell; check for nullity */
00584     cell = Vclist_getCell(thee->clist, center);
00585     if (cell == VNULL) return;
00586
00587     /* Reset the list of atom flags */
00588     for (iatom=0; iatom<cell->natoms; iatom++) {
00589         atom = cell->atoms[iatom];
00590         atomID = atom->id;
00591         thee->atomFlags[atomID] = 0;
00592     }
00593
00594     /* Get the local accessibility */
00595     acc = splineAcc(thee, center, win, infrad, cell);
00596
00597     /* Accumulate the gradient of all local atoms */
00598     if (acc > VSMALL) {
00599         for (iatom=0; iatom<cell->natoms; iatom++) {
00600             atom = cell->atoms[iatom];
00601             Vacc_splineAccGradAtomNorm(thee, center, win, infrad, atom, tgrad);
00602         }
00603         for (i=0; i<VAPBS_DIM; i++) grad[i] += tgrad[i];
00604     }
00605     for (i=0; i<VAPBS_DIM; i++) grad[i] *= -acc;
00606 }
00607
00608 VPUBLIC double Vacc_molAcc(Vacc *thee, double center[VAPBS_DIM],
00609         double radius) {
00610
00611     double rc;
00612
00613     /* ******* CHECK IF OUTSIDE ATOM+PROBE RADIUS SURFACE ***** */
00614     if (Vacc_ivdwAcc(thee, center, radius) == 1.0) {
00615
00616         /* Vnm_print(2, "DEBUG:  ivdwAcc = 1.0\n"); */
00617         rc = 1.0;
00618
00619     /* ******* CHECK IF INSIDE ATOM RADIUS SURFACE ***** */
00620     } else if (Vacc_vdwAcc(thee, center) == 0.0) {
```

```
00621
00622         /* Vnm_print(2, "DEBUG:  vdwAcc = 0.0\n"); */
00623         rc = 0.0;
00624
00625     /* ******* CHECK IF OUTSIDE MOLECULAR SURFACE ***** */
00626     } else {
00627
00628         /* Vnm_print(2, "DEBUG:  calling fastMolAcc...\n"); */
00629         rc = Vacc_fastMolAcc(thee, center, radius);
00630
00631     }
00632
00633     return rc;
00634
00635 }
00636
00637 VPUBLIC double Vacc_fastMolAcc(Vacc *thee, double center[VAPBS_DIM],
00638         double radius) {
00639
00640     Vatom *atom;
00641     VaccSurf *surf;
00642     VclistCell *cell;
00643     int ipt, iatom, atomID;
00644     double dist2, rad2;
00645
00646     rad2 = radius*radius;
00647
00648     /* Check to see if the SAS has been defined */
00649     if (thee->surf == VNULL) Vacc_SASA(thee, radius);
00650
00651     /* Get the cell associated with this point */
00652     cell = Vclist_getCell(thee->clist, center);
00653     if (cell == VNULL) {
00654         Vnm_print(2, "Vacc_fastMolAcc:  unexpected VNULL VclistCell!\n");
00655         return 1.0;
00656     }
00657
00658     /* Loop through all the atoms in the cell */
00659     for (iatom=0; iatom<cell->natoms; iatom++) {
00660         atom = cell->atoms[iatom];
00661         atomID = Vatom_getAtomID(atom);
00662         surf = thee->surf[atomID];
00663         /* Loop through all SAS points associated with this atom */
00664         for (ipt=0; ipt<surf->npts; ipt++) {
00665             /* See if we're within a probe radius of the point */
00666             dist2 = VSQR(center[0]-(surf->xpts[ipt]))
00667                 + VSQR(center[1]-(surf->ypts[ipt]))
00668                 + VSQR(center[2]-(surf->zpts[ipt]));
00669             if (dist2 < rad2) return 1.0;
00670         }
00671     }
00672
00673     /* If all else failed, we are not inside the molecular surface */
00674     return 0.0;
00675 }
00676
00677
00678 #if defined(HAVE_MC_H)
00679 VPUBLIC void Vacc_writeGMV(Vacc *thee, double radius, int meth, Gem *gm,
00680   char *iodev, char *iofmt, char *iohost, char *iofile) {
00681
00682     double *accVals[MAXV], coord[3];
00683     Vio *sock;
00684     int ivert, icoord;
00685
00686     for (ivert=0; ivert<MAXV; ivert++) accVals[ivert] = VNULL;
00687     accVals[0] = (void *)Vmem_malloc(thee->mem, Gem_numVV(gm), sizeof(double));
00688     accVals[1] = (void *)Vmem_malloc(thee->mem, Gem_numVV(gm), sizeof(double));
00689     for (ivert=0; ivert<Gem_numVV(gm); ivert++) {
00690         for (icoord=0;icoord<3;icoord++)
00691           coord[icoord] = VV_coord(Gem_VV(gm, ivert), icoord);
00692         if (meth == 0) {
00693             accVals[0][ivert] = Vacc_molAcc(thee, coord, radius);
00694             accVals[1][ivert] = Vacc_molAcc(thee, coord, radius);
00695         } else if (meth == 1) {
00696             accVals[0][ivert] = Vacc_ivdwAcc(thee, coord, radius);
00697             accVals[1][ivert] = Vacc_ivdwAcc(thee, coord, radius);
00698         } else if (meth == 2) {
00699             accVals[0][ivert] = Vacc_vdwAcc(thee, coord);
00700             accVals[1][ivert] = Vacc_vdwAcc(thee, coord);
00701         } else VASSERT(0);
```

```
00702     }
00703     sock = Vio_ctor(iodev, iofmt, iohost, iofile, "w");
00704     Gem_writeGMV(gm, sock, 1, accVals);
00705     Vio_dtor(&sock);
00706     Vmem_free(thee->mem, Gem_numVV(gm), sizeof(double),
00707       (void **)&(accVals[0]));
00708     Vmem_free(thee->mem, Gem_numVV(gm), sizeof(double),
00709       (void **)&(accVals[1]));
00710 }
00711 #endif /* defined(HAVE_MC_H) */
00712
00713 VPUBLIC double Vacc_SASA(Vacc *thee,
00714                          double radius
00715                          ) {
00716
00717     int i,
00718         natom;
00719     double area;
00720           //*apos; // gcc says unused
00721     Vatom *atom;
00722     VaccSurf *asurf;
00723
00724     time_t ts; // PCE: temp
00725     ts = clock();
00726
00727     //unsigned long long mbeg; // gcc says unused
00728
00729     natom = Valist_getNumberAtoms(thee->alist);
00730
00731     /* Check to see if we need to build the surface */
00732     if (thee->surf == VNULL) {
00733         thee->surf = Vmem_malloc(thee->mem, natom, sizeof(VaccSurf *));
00734
00735 #if defined(DEBUG_MAC_OSX_OCL) || defined(DEBUG_MAC_OSX_STANDARD)
00736 #include "mach_chud.h"
00737         machm_(&mbeg);
00738 #pragma omp parallel for private(i,atom)
00739 #endif
00740         for (i=0; i<natom; i++) {
00741             atom = Valist_getAtom(thee->alist, i);
00742             /* NOTE:  RIGHT NOW WE DO THIS FOR THE ENTIRE MOLECULE WHICH IS
00743              * INCREDIBLY INEFFICIENT, PARTICULARLY DURING FOCUSING!!! */
00744             thee->surf[i] = Vacc_atomSurf(thee, atom, thee->refSphere,
00745                                           radius);
00746         }
00747     }
00748
00749     /* Calculate the area */
00750     area = 0.0;
00751     for (i=0; i<natom; i++) {
00752         atom = Valist_getAtom(thee->alist, i);
00753         asurf = thee->surf[i];
00754         /* See if this surface needs to be rebuilt */
00755         if (asurf->probe_radius != radius) {
00756             Vnm_print(2, "Vacc_SASA:  Warning -- probe radius changed from %g to %g!\n",
00757                       asurf->probe_radius, radius);
00758             VaccSurf_dtor2(asurf);
00759             thee->surf[i] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
00760             asurf = thee->surf[i];
00761         }
00762         area += (asurf->area);
00763     }
00764
00765 #if defined(DEBUG_MAC_OSX_OCL) || defined(DEBUG_MAC_OSX_STANDARD)
00766     mets_(&mbeg, "Vacc_SASA - Parallel");
00767 #endif
00768
00769     Vnm_print(0, "Vacc_SASA: Time elapsed: %f\n", ((double)clock() - ts) / CLOCKS_PER_SEC);
00770     return area;
00771
00772 }
00773
00774 VPUBLIC double Vacc_totalSASA(Vacc *thee, double radius) {
00775
00776     return Vacc_SASA(thee, radius);
00777
00778 }
00779
00780 VPUBLIC double Vacc_atomSASA(Vacc *thee, double radius, Vatom *atom) {
00781
00782     VaccSurf *asurf;
```

```
00783     int id;
00784
00785     if (thee->surf == VNULL) Vacc_SASA(thee, radius);
00786
00787     id = Vatom_getAtomID(atom);
00788     asurf = thee->surf[id];
00789
00790     /* See if this surface needs to be rebuilt */
00791     if (asurf->probe_radius != radius) {
00792         Vnm_print(2, "Vacc_SASA:  Warning -- probe radius changed from %g to %g!\n",
00793                 asurf->probe_radius, radius);
00794         VaccSurf_dtor2(asurf);
00795         thee->surf[id] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
00796         asurf = thee->surf[id];
00797     }
00798
00799     return asurf->area;
00800
00801 }
00802
00803 VPUBLIC VaccSurf* VaccSurf_ctor(Vmem *mem, double probe_radius, int nsphere) {
00804     VaccSurf *thee;
00805
00806     //thee = Vmem_malloc(mem, 1, sizeof(Vacc) );
00807     thee = (VaccSurf*)calloc(1,sizeof(Vacc));
00808     VASSERT( VaccSurf_ctor2(thee, mem, probe_radius, nsphere) );
00809
00810     return thee;
00811 }
00812
00813 VPUBLIC int VaccSurf_ctor2(VaccSurf *thee, Vmem *mem, double probe_radius,
00814         int nsphere) {
00815
00816     if (thee == VNULL)
00817         return 0;
00818
00819     thee->mem = mem;
00820     thee->npts = nsphere;
00821     thee->probe_radius = probe_radius;
00822     thee->area = 0.0;
00823
00824     if (thee->npts > 0) {
00825         thee->xpts = Vmem_malloc(thee->mem, thee->npts, sizeof(double));
00826         thee->ypts = Vmem_malloc(thee->mem, thee->npts, sizeof(double));
00827         thee->zpts = Vmem_malloc(thee->mem, thee->npts, sizeof(double));
00828         thee->bpts = Vmem_malloc(thee->mem, thee->npts, sizeof(char));
00829     } else {
00830         thee->xpts = VNULL;
00831         thee->ypts = VNULL;
00832         thee->zpts = VNULL;
00833         thee->bpts = VNULL;
00834     }
00835
00836     return 1;
00837 }
00838
00839 VPUBLIC void VaccSurf_dtor(VaccSurf **thee) {
00840
00841     Vmem *mem;
00842
00843     if ((*thee) != VNULL) {
00844         mem = (*thee)->mem;
00845         VaccSurf_dtor2(*thee);
00846         //Vmem_free(mem, 1, sizeof(VaccSurf), (void **)thee);
00847         free(*thee);
00848         (*thee) = VNULL;
00849     }
00850
00851 }
00852
00853 VPUBLIC void VaccSurf_dtor2(VaccSurf *thee) {
00854
00855     if (thee->npts > 0) {
00856         Vmem_free(thee->mem, thee->npts, sizeof(double),
00857                 (void **)&(thee->xpts));
00858         Vmem_free(thee->mem, thee->npts, sizeof(double),
00859                 (void **)&(thee->ypts));
00860         Vmem_free(thee->mem, thee->npts, sizeof(double),
00861                 (void **)&(thee->zpts));
00862         Vmem_free(thee->mem, thee->npts, sizeof(char),
00863                 (void **)&(thee->bpts));
```

```
00864     }
00865
00866 }
00867
00868 VPUBLIC VaccSurf* Vacc_atomSurf(Vacc *thee, Vatom *atom,
00869                                 VaccSurf *ref, double prad) {
00870
00871  VaccSurf *surf;
00872  size_t i, j, npts;
00873     int atomID;
00874     double arad, rad, pos[3], *apos;
00875
00876     /* Get atom information */
00877     arad = Vatom_getRadius(atom);
00878     apos = Vatom_getPosition(atom);
00879     atomID = Vatom_getAtomID(atom);
00880
00881     if (arad < VSMALL) {
00882         return VaccSurf_ctor(thee->mem, prad, 0);
00883     }
00884
00885     rad = arad + prad;
00886
00887     /* Determine which points will contribute */
00888     npts = 0;
00889     for (i=0; i<ref->npts; i++) {
00890         /* Reset point flag: zero-radius atoms do not contribute */
00891         pos[0] = rad*(ref->xpts[i]) + apos[0];
00892         pos[1] = rad*(ref->ypts[i]) + apos[1];
00893         pos[2] = rad*(ref->zpts[i]) + apos[2];
00894         if (ivdwAccExclus(thee, pos, prad, atomID)) {
00895             npts++;
00896             ref->bpts[i] = (ref->bpts[i] « 1) + 1;
00897         } else {
00898             ref->bpts[i] «= 1;
00899         }
00900     }
00901
00902     /* Allocate space for the points */
00903     surf = VaccSurf_ctor(thee->mem, prad, npts);
00904
00905     /* Assign the points */
00906     j = 0;
00907     for (i=0; i<ref->npts; i++) {
00908      char flag = ref->bpts[i] & 1;
00909      ref->bpts[i] »= 1;
00910      if (flag) {
00911             surf->bpts[j] = 1;
00912             surf->xpts[j] = rad*(ref->xpts[i]) + apos[0];
00913             surf->ypts[j] = rad*(ref->ypts[i]) + apos[1];
00914             surf->zpts[j] = rad*(ref->zpts[i]) + apos[2];
00915             j++;
00916         }
00917     }
00918
00919     /* Assign the area */
00920     surf->area = 4.0*VPI*rad*rad*((double)(surf->npts))/((double)(ref->npts));
00921
00922     return surf;
00923
00924 }
00925
00926 VPUBLIC VaccSurf* VaccSurf_refSphere(Vmem *mem, int npts) {
00927
00928     VaccSurf *surf;
00929     int nactual, i, itheta, ntheta, iphi, nphimax, nphi;
00930     double frac;
00931     double sintheta, costheta, theta, dtheta;
00932     double sinphi, cosphi, phi, dphi;
00933
00934     /* Setup "constants" */
00935     frac = ((double)(npts))/4.0;
00936     ntheta = VRINT(VSQRT(Vunit_pi*frac));
00937     dtheta = Vunit_pi/((double)(ntheta));
00938     nphimax = 2*ntheta;
00939
00940     /* Count the actual number of points to be used */
00941     nactual = 0;
00942     for (itheta=0; itheta<ntheta; itheta++) {
00943         theta = dtheta*((double)(itheta));
00944         sintheta = VSIN(theta);
```

```
00945         costheta = VCOS(theta);
00946         nphi = VRINT(sintheta*nphimax);
00947         nactual += nphi;
00948     }
00949
00950     /* Allocate space for the points */
00951     surf = VaccSurf_ctor(mem, 1.0, nactual);
00952
00953     /* Clear out the boolean array */
00954     for (i=0; i<nactual; i++) surf->bpts[i] = 1;
00955
00956     /* Assign the points */
00957     nactual = 0;
00958     for (itheta=0; itheta<ntheta; itheta++) {
00959         theta = dtheta*((double)(itheta));
00960         sintheta = VSIN(theta);
00961         costheta = VCOS(theta);
00962         nphi = VRINT(sintheta*nphimax);
00963         if (nphi != 0) {
00964             dphi = 2*Vunit_pi/((double)(nphi));
00965             for (iphi=0; iphi<nphi; iphi++) {
00966                 phi = dphi*((double)(iphi));
00967                 sinphi = VSIN(phi);
00968                 cosphi = VCOS(phi);
00969                 surf->xpts[nactual] = cosphi * sintheta;
00970                 surf->ypts[nactual] = sinphi * sintheta;
00971                 surf->zpts[nactual] = costheta;
00972                 nactual++;
00973             }
00974         }
00975     }
00976
00977     surf->npts = nactual;
00978
00979     return surf;
00980 }
00981
00982 VPUBLIC VaccSurf* Vacc_atomSASPoints(Vacc *thee, double radius,
00983         Vatom *atom) {
00984
00985     VaccSurf *asurf = VNULL;
00986     int id;
00987
00988     if (thee->surf == VNULL) Vacc_SASA(thee, radius);
00989     id = Vatom_getAtomID(atom);
00990
00991     asurf = thee->surf[id];
00992
00993     /* See if this surface needs to be rebuilt */
00994     if (asurf->probe_radius != radius) {
00995         Vnm_print(2, "Vacc_SASA:  Warning -- probe radius changed from %g to %g!\n",
00996                 asurf->probe_radius, radius);
00997         VaccSurf_dtor2(asurf);
00998         thee->surf[id] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
00999         asurf = thee->surf[id];
01000     }
01001
01002     return asurf;
01003
01004 }
01005
01006 VPUBLIC void Vacc_splineAccGradAtomNorm4(Vacc *thee, double center[VAPBS_DIM],
01007                                          double win, double infrad, Vatom *atom, double *grad) {
01008
01009     int i;
01010     double dist, *apos, arad, sm, sm2, sm3, sm4, sm5, sm6, sm7;
01011     double e, e2, e3, e4, e5, e6, e7;
01012     double b, b2, b3, b4, b5, b6, b7;
01013     double c0, c1, c2, c3, c4, c5, c6, c7;
01014     double denom, mygrad;
01015     double mychi = 1.0;           /* Char. func. value for given atom */
01016
01017     VASSERT(thee != NULL);
01018
01019     /* The grad is zero by default */
01020     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
01021
01022     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
01023         * *** MAGNITUDE OF THE FORCE *** */
01024     apos = Vatom_getPosition(atom);
01025     /* Zero-radius atoms don't contribute */
```

```
01026     if (Vatom_getRadius(atom) > 0.0) {
01027
01028         arad = Vatom_getRadius(atom);
01029         arad = arad + infrad;
01030         b = arad - win;
01031         e = arad + win;
01032
01033         e2 = e * e;
01034         e3 = e2 * e;
01035         e4 = e3 * e;
01036         e5 = e4 * e;
01037         e6 = e5 * e;
01038         e7 = e6 * e;
01039         b2 = b * b;
01040         b3 = b2 * b;
01041         b4 = b3 * b;
01042         b5 = b4 * b;
01043         b6 = b5 * b;
01044         b7 = b6 * b;
01045
01046         denom = e7  - 7.0*b*e6 + 21.0*b2*e5 - 35.0*e4*b3
01047             + 35.0*e3*b4 - 21.0*b5*e2  + 7.0*e*b6 - b7;
01048         c0 = b4*(35.0*e3 - 21.0*b*e2 + 7*e*b2 - b3)/denom;
01049         c1 = -140.0*b3*e3/denom;
01050         c2 = 210.0*e2*b2*(e + b)/denom;
01051         c3 = -140.0*e*b*(e2 + 3.0*b*e + b2)/denom;
01052         c4 =  35.0*(e3 + 9.0*b*e2 + + 9.0*e*b2 + b3)/denom;
01053         c5 = -84.0*(e2 + 3.0*b*e + b2)/denom;
01054         c6 =  70.0*(e + b)/denom;
01055         c7 = -20.0/denom;
01056
01057         dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
01058                      + VSQR(apos[2]-center[2]));
01059
01060         /* If we're inside an atom, the entire characteristic function
01061             * will be zero and the grad will be zero, so we can stop */
01062         if (dist < (arad - win)) return;
01063         /* Likewise, if we're outside the smoothing window, the characteristic
01064             * function is unity and the grad will be zero, so we can stop */
01065         else if (dist > (arad + win)) return;
01066         /* Account for floating point error at the border
01067             * NAB:  COULDN'T THESE TESTS BE COMBINED AS BELOW
01068             * (Vacc_splineAccAtom)? */
01069         else if ((VABS(dist - (arad - win)) < VSMALL) ||
01070                  (VABS(dist - (arad + win)) < VSMALL)) return;
01071         /* If we're inside the smoothing window */
01072         else {
01073             sm = dist;
01074             sm2 = sm * sm;
01075             sm3 = sm2 * sm;
01076             sm4 = sm3 * sm;
01077             sm5 = sm4 * sm;
01078             sm6 = sm5 * sm;
01079             sm7 = sm6 * sm;
01080             mychi = c0 + c1*sm + c2*sm2 + c3*sm3
01081                 + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
01082             mygrad = c1 + 2.0*c2*sm  + 3.0*c3*sm2 + 4.0*c4*sm3
01083                 + 5.0*c5*sm4 + 6.0*c6*sm5 + 7.0*c7*sm6;
01084             if (mychi <= 0.0) {
01085                 /* Avoid numerical round off errors */
01086                 return;
01087             } else if (mychi > 1.0) {
01088                 /* Avoid numerical round off errors */
01089                 mychi = 1.0;
01090             }
01091         }
01092         /* Now assemble the grad vector */
01093         VASSERT(mychi > 0.0);
01094         for (i=0; i<VAPBS_DIM; i++)
01095             grad[i] = -(mygrad/mychi)*((center[i] - apos[i])/dist);
01096     }
01097 }
01098
01099 VPUBLIC void Vacc_splineAccGradAtomNorm3(Vacc *thee, double center[VAPBS_DIM],
01100                                          double win, double infrad, Vatom *atom, double *grad) {
01101
01102     int i;
01103     double dist, *apos, arad, sm, sm2, sm3, sm4, sm5;
01104     double e, e2, e3, e4, e5;
01105     double b, b2, b3, b4, b5;
01106     double c0, c1, c2, c3, c4, c5;
```

```
01107     double denom, mygrad;
01108     double mychi = 1.0;           /* Char. func. value for given atom */
01109
01110     VASSERT(thee != NULL);
01111
01112     /* The grad is zero by default */
01113     for (i=0; i<VAPBS_DIM; i++) grad[i] = 0.0;
01114
01115     /* *** CALCULATE THE CHARACTERISTIC FUNCTION VALUE FOR THIS ATOM AND THE
01116         * *** MAGNITUDE OF THE FORCE *** */
01117     apos = Vatom_getPosition(atom);
01118     /* Zero-radius atoms don't contribute */
01119     if (Vatom_getRadius(atom) > 0.0) {
01120
01121         arad = Vatom_getRadius(atom);
01122         arad = arad + infrad;
01123         b = arad - win;
01124         e = arad + win;
01125
01126         e2 = e * e;
01127         e3 = e2 * e;
01128         e4 = e3 * e;
01129         e5 = e4 * e;
01130         b2 = b * b;
01131         b3 = b2 * b;
01132         b4 = b3 * b;
01133         b5 = b4 * b;
01134
01135         denom = pow((e - b), 5.0);
01136         c0 = -10.0*e2*b3 + 5.0*e*b4 - b5;
01137         c1 = 30.0*e2*b2;
01138         c2 = -30.0*(e2*b + e*b2);
01139         c3 = 10.0*(e2 + 4.0*e*b + b2);
01140         c4 = -15.0*(e + b);
01141         c5 = 6;
01142         c0 = c0/denom;
01143         c1 = c1/denom;
01144         c2 = c2/denom;
01145         c3 = c3/denom;
01146         c4 = c4/denom;
01147         c5 = c5/denom;
01148
01149         dist = VSQRT(VSQR(apos[0]-center[0]) + VSQR(apos[1]-center[1])
01150                      + VSQR(apos[2]-center[2]));
01151
01152         /* If we're inside an atom, the entire characteristic function
01153             * will be zero and the grad will be zero, so we can stop */
01154         if (dist < (arad - win)) return;
01155         /* Likewise, if we're outside the smoothing window, the characteristic
01156             * function is unity and the grad will be zero, so we can stop */
01157         else if (dist > (arad + win)) return;
01158         /* Account for floating point error at the border
01159             * NAB:  COULDN'T THESE TESTS BE COMBINED AS BELOW
01160             * (Vacc_splineAccAtom)? */
01161         else if ((VABS(dist - (arad - win)) < VSMALL) ||
01162                  (VABS(dist - (arad + win)) < VSMALL)) return;
01163         /* If we're inside the smoothing window */
01164         else {
01165             sm = dist;
01166             sm2 = sm * sm;
01167             sm3 = sm2 * sm;
01168             sm4 = sm3 * sm;
01169             sm5 = sm4 * sm;
01170             mychi = c0 + c1*sm + c2*sm2 + c3*sm3
01171                 + c4*sm4 + c5*sm5;
01172             mygrad = c1 + 2.0*c2*sm  + 3.0*c3*sm2 + 4.0*c4*sm3
01173                 + 5.0*c5*sm4;
01174             if (mychi <= 0.0) {
01175                 /* Avoid numerical round off errors */
01176                 return;
01177             } else if (mychi > 1.0) {
01178                 /* Avoid numerical round off errors */
01179                 mychi = 1.0;
01180             }
01181         }
01182         /* Now assemble the grad vector */
01183         VASSERT(mychi > 0.0);
01184         for (i=0; i<VAPBS_DIM; i++)
01185             grad[i] = -(mygrad/mychi)*((center[i] - apos[i])/dist);
01186     }
01187 }
```

```
01188
01189 /* ///////////////////////////////////////////////////////////////////////////
01190    // Routine:  Vacc_atomdSAV
01191    //
01192    // Purpose:  Calculates the vector valued atomic derivative of volume
01193    //
01194    // Args:     radius  The radius of the solvent probe in Angstroms
01195    //           iatom   Index of the atom in thee->alist
01196    //
01197    // Author:   Jason Wagoner
01198    //           Nathan Baker (original FORTRAN routine from UHBD by Brock Luty)
01200 VPUBLIC void Vacc_atomdSAV(Vacc *thee,
01201                            double srad,
01202                            Vatom *atom,
01203                            double *dSA
01204                           ) {
01205
01206     int ipt, iatom;
01207
01208     double area;
01209     double *tPos, tRad, vec[3];
01210     double dx,dy,dz;
01211     VaccSurf *ref;
01212     dx = 0.0;
01213     dy = 0.0;
01214     dz = 0.0;
01215     /* Get the atom information */
01216     ref = thee->refSphere;
01217     iatom = Vatom_getAtomID(atom);
01218
01219     dSA[0] = 0.0;
01220     dSA[1] = 0.0;
01221     dSA[2] = 0.0;
01222
01223     tPos = Vatom_getPosition(atom);
01224     tRad = Vatom_getRadius(atom);
01225
01226     if(tRad == 0.0) return;
01227
01228     area = 4.0*VPI*(tRad+srad)*(tRad+srad)/((double)(ref->npts));
01229     for (ipt=0; ipt<ref->npts; ipt++) {
01230         vec[0] = (tRad+srad)*ref->xpts[ipt] + tPos[0];
01231         vec[1] = (tRad+srad)*ref->ypts[ipt] + tPos[1];
01232         vec[2] = (tRad+srad)*ref->zpts[ipt] + tPos[2];
01233         if (ivdwAccExclus(thee, vec, srad, iatom)) {
01234             dx = dx+vec[0]-tPos[0];
01235             dy = dy+vec[1]-tPos[1];
01236             dz = dz+vec[2]-tPos[2];
01237         }
01238     }
01239
01240     if ((tRad+srad) != 0){
01241         dSA[0] = dx*area/(tRad+srad);
01242         dSA[1] = dy*area/(tRad+srad);
01243         dSA[2] = dz*area/(tRad+srad);
01244     }
01245
01246 }
01247
01248 /* Note: This is purely test code to make certain that the dSASA code is
01249          behaving properly. This function should NEVER be called by anyone
01250          other than an APBS developer at Wash U.
01251 */
01252 VPRIVATE double Vacc_SASAPos(Vacc *thee, double radius) {
01253
01254     int i, natom;
01255     double area;
01256     Vatom *atom;
01257     VaccSurf *asurf;
01258
01259     natom = Valist_getNumberAtoms(thee->alist);
01260
01261     /* Calculate the area */
01262     area = 0.0;
01263     for (i=0; i<natom; i++) {
01264         atom = Valist_getAtom(thee->alist, i);
01265         asurf = thee->surf[i];
01266
01267         VaccSurf_dtor2(asurf);
01268         thee->surf[i] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
01269         asurf = thee->surf[i];
```

```
01270         area += (asurf->area);
01271     }
01272
01273     return area;
01274
01275 }
01276
01277 VPRIVATE double Vacc_atomSASAPos(Vacc *thee,
01278                                  double radius,
01279                                  Vatom *atom, /* The atom being manipulated */
01280                                  int mode
01281                                 ) {
01282
01283     VaccSurf *asurf;
01284     int id;
01285     static int warned = 0;
01286
01287     if ((thee->surf == VNULL) || (mode == 1)){
01288         if(!warned){
01289             Vnm_print(2, "WARNING: Recalculating entire surface!!!!\n");
01290             warned = 1;
01291         }
01292         Vacc_SASAPos(thee, radius); // reinitialize before we can do anything about doing a calculation on
      a repositioned atom
01293     }
01294
01295     id = Vatom_getAtomID(atom);
01296     asurf = thee->surf[id];
01297
01298     VaccSurf_dtor(&asurf);
01299     thee->surf[id] = Vacc_atomSurf(thee, atom, thee->refSphere, radius);
01300     asurf = thee->surf[id];
01301
01302     //printf("%s: Time elapsed: %f\n", __func__, ((double)clock() - ts) / CLOCKS_PER_SEC);
01303
01304     return asurf->area;
01305 }
01306
01307 /* ///////////////////////////////////////////////////////////////////////////
01308    // Routine:  Vacc_atomdSASA
01309    //
01310    // Purpose:  Calculates the derivative of surface area with respect to atomic
01311    //           displacement using finite difference methods.
01312    //
01313    // Args:     radius  The radius of the solvent probe in Angstroms
01314    //           iatom   Index of the atom in thee->alist
01315    //
01316    // Author:   Jason Wagoner
01317    //   David Gohara
01318    //           Nathan Baker (original FORTRAN routine from UHBD by Brock Luty)
01320 VPUBLIC void Vacc_atomdSASA(Vacc *thee,
01321                             double dpos,
01322                             double srad,
01323                             Vatom *atom,
01324                             double *dSA
01325                            ) {
01326
01327     double *temp_Pos,
01328            tPos[3],
01329            axb1,
01330            axt1,
01331            ayb1,
01332            ayt1,
01333            azb1,
01334            azt1;
01335     VaccSurf *ref;
01336
01337     //printf("%s: entering\n", __func__);
01338     time_t ts;
01339     ts = clock();
01340
01341     /* Get the atom information */
01342     ref = thee->refSphere;
01343     temp_Pos = Vatom_getPosition(atom); // Get a pointer to the position object.  You actually manipulate
      the atom doing this...
01344
01345     tPos[0] = temp_Pos[0];
01346     tPos[1] = temp_Pos[1];
01347     tPos[2] = temp_Pos[2];
01348
01349     /* Shift by pos -/+ on x */
```

```
01350     temp_Pos[0] -= dpos;
01351     axb1 = Vacc_atomSASAPos(thee, srad, atom,0);
01352     temp_Pos[0] = tPos[0];
01353
01354     temp_Pos[0] += dpos;
01355     axt1 = Vacc_atomSASAPos(thee, srad, atom,0);
01356     temp_Pos[0] = tPos[0];
01357
01358     /* Shift by pos -/+ on y */
01359     temp_Pos[1] -= dpos;
01360     ayb1 = Vacc_atomSASAPos(thee, srad, atom,0);
01361     temp_Pos[1] = tPos[1];
01362
01363     temp_Pos[1] += dpos;
01364     ayt1 = Vacc_atomSASAPos(thee, srad, atom,0);
01365     temp_Pos[1] = tPos[1];
01366
01367     /* Shift by pos -/+ on z */
01368     temp_Pos[2] -= dpos;
01369     azb1 = Vacc_atomSASAPos(thee, srad, atom,0);
01370     temp_Pos[2] = tPos[2];
01371
01372     temp_Pos[2] += dpos;
01373     azt1 = Vacc_atomSASAPos(thee, srad, atom,0);
01374     temp_Pos[2] = tPos[2];
01375
01376     /* Reset the atom SASA to zero displacement */
01377     Vacc_atomSASAPos(thee, srad, atom,0);
01378
01379     /* Calculate the final value */
01380     dSA[0] = (axt1-axb1)/(2.0 * dpos);
01381     dSA[1] = (ayt1-ayb1)/(2.0 * dpos);
01382     dSA[2] = (azt1-azb1)/(2.0 * dpos);
01383 }
01384
01385 /* Note: This is purely test code to make certain that the dSASA code is
01386          behaving properly. This function should NEVER be called by anyone
01387          other than an APBS developer at Wash U.
01388 */
01389 VPUBLIC void Vacc_totalAtomdSASA(Vacc *thee, double dpos, double srad, Vatom *atom, double *dSA) {
01390
01391     int iatom;
01392     double *temp_Pos, tRad;
01393     double tPos[3];
01394     double axb1,axt1,ayb1,ayt1,azb1,azt1;
01395     VaccSurf *ref;
01396
01397     /* Get the atom information */
01398     ref = thee->refSphere;
01399     temp_Pos = Vatom_getPosition(atom);
01400     tRad = Vatom_getRadius(atom);
01401     iatom = Vatom_getAtomID(atom);
01402
01403     dSA[0] = 0.0;
01404     dSA[1] = 0.0;
01405     dSA[2] = 0.0;
01406
01407     tPos[0] = temp_Pos[0];
01408     tPos[1] = temp_Pos[1];
01409     tPos[2] = temp_Pos[2];
01410
01411     /* Shift by pos -/+ on x */
01412     temp_Pos[0] -= dpos;
01413     axb1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01414     temp_Pos[0] = tPos[0];
01415
01416     temp_Pos[0] += dpos;
01417     axt1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01418     temp_Pos[0] = tPos[0];
01419
01420     /* Shift by pos -/+ on y */
01421     temp_Pos[1] -= dpos;
01422     ayb1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01423     temp_Pos[1] = tPos[1];
01424
01425     temp_Pos[1] += dpos;
01426     ayt1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01427     temp_Pos[1] = tPos[1];
01428
01429     /* Shift by pos -/+ on z */
01430     temp_Pos[2] -= dpos;
```

```
01431     azb1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01432     temp_Pos[2] = tPos[2];
01433
01434     temp_Pos[2] += dpos;
01435     azt1 = Vacc_atomSASAPos(thee, srad, atom, 1);
01436     temp_Pos[2] = tPos[2];
01437
01438     /* Calculate the final value */
01439     dSA[0] = (axt1-axb1)/(2.0 * dpos);
01440     dSA[1] = (ayt1-ayb1)/(2.0 * dpos);
01441     dSA[2] = (azt1-azb1)/(2.0 * dpos);
01442 }
01443
01444 /* Note: This is purely test code to make certain that the dSASA code is
01445          behaving properly. This function should NEVER be called by anyone
01446          other than an APBS developer at Wash U.
01447 */
01448 VPUBLIC void Vacc_totalAtomdSAV(Vacc *thee, double dpos, double srad, Vatom *atom, double *dSA, Vclist
       *clist) {
01449
01450     int iatom;
01451     double *temp_Pos, tRad;
01452     double tPos[3];
01453     double axb1,axt1,ayb1,ayt1,azb1,azt1;
01454     VaccSurf *ref;
01455
01456     /* Get the atom information */
01457     ref = thee->refSphere;
01458     temp_Pos = Vatom_getPosition(atom);
01459     tRad = Vatom_getRadius(atom);
01460     iatom = Vatom_getAtomID(atom);
01461
01462     dSA[0] = 0.0;
01463     dSA[1] = 0.0;
01464     dSA[2] = 0.0;
01465
01466     tPos[0] = temp_Pos[0];
01467     tPos[1] = temp_Pos[1];
01468     tPos[2] = temp_Pos[2];
01469
01470     /* Shift by pos -/+ on x */
01471     temp_Pos[0] -= dpos;
01472     axb1 = Vacc_totalSAV(thee,clist, VNULL, srad);
01473     temp_Pos[0] = tPos[0];
01474
01475     temp_Pos[0] += dpos;
01476     axt1 = Vacc_totalSAV(thee,clist, VNULL, srad);
01477     temp_Pos[0] = tPos[0];
01478
01479     /* Shift by pos -/+ on y */
01480     temp_Pos[1] -= dpos;
01481     ayb1 = Vacc_totalSAV(thee,clist, VNULL, srad);
01482     temp_Pos[1] = tPos[1];
01483
01484     temp_Pos[1] += dpos;
01485     ayt1 = Vacc_totalSAV(thee,clist, VNULL, srad);
01486     temp_Pos[1] = tPos[1];
01487
01488     /* Shift by pos -/+ on z */
01489     temp_Pos[2] -= dpos;
01490     azb1 = Vacc_totalSAV(thee,clist, VNULL, srad);
01491     temp_Pos[2] = tPos[2];
01492
01493     temp_Pos[2] += dpos;
01494     azt1 = Vacc_totalSAV(thee,clist, VNULL, srad);
01495     temp_Pos[2] = tPos[2];
01496
01497     /* Calculate the final value */
01498     dSA[0] = (axt1-axb1)/(2.0 * dpos);
01499     dSA[1] = (ayt1-ayb1)/(2.0 * dpos);
01500     dSA[2] = (azt1-azb1)/(2.0 * dpos);
01501 }
01502
01503 VPUBLIC double Vacc_totalSAV(Vacc *thee, Vclist *clist, APOLparm *apolparm, double radius) {
01504
01505     int i;
01506     int npts[3];
01507
01508     double spacs[3], vec[3];
01509     double w, wx, wy, wz, len, fn, x, y, z, vol;
01510     double vol_density,sav;
```

```
01511     double *lower_corner, *upper_corner;
01512
01513     sav = 0.0;
01514     vol = 1.0;
01515     vol_density = 2.0;
01516
01517     lower_corner = clist->lower_corner;
01518     upper_corner = clist->upper_corner;
01519
01520     for (i=0; i<3; i++) {
01521         len = upper_corner[i] - lower_corner[i];
01522         vol *= len;
01523         fn = len*vol_density + 1;
01524         npts[i] = (int)ceil(fn);
01525         spacs[i] = len/((double)(npts[i])-1.0);
01526         if (apolparm != VNULL) {
01527             if (apolparm->setgrid) {
01528                 if (apolparm->grid[i] > spacs[i]) {
01529                     Vnm_print(2, "Vacc_totalSAV:  Warning, your GRID value (%g) is larger than the
      recommended value (%g)!\n",
01530                         apolparm->grid[i], spacs[i]);
01531                 }
01532                 spacs[i] = apolparm->grid[i];
01533
01534             }
01535         }
01536     }
01537
01538     for (x=lower_corner[0]; x<=upper_corner[0]; x=x+spacs[0]) {
01539         if ( VABS(x - lower_corner[0]) < VSMALL) {
01540             wx = 0.5;
01541         } else if ( VABS(x - upper_corner[0]) < VSMALL) {
01542             wx = 0.5;
01543         } else {
01544             wx = 1.0;
01545         }
01546         vec[0] = x;
01547         for (y=lower_corner[1]; y<=upper_corner[1]; y=y+spacs[1]) {
01548             if ( VABS(y - lower_corner[1]) < VSMALL) {
01549                 wy = 0.5;
01550             } else if ( VABS(y - upper_corner[1]) < VSMALL) {
01551                 wy = 0.5;
01552             } else {
01553                 wy = 1.0;
01554             }
01555             vec[1] = y;
01556             for (z=lower_corner[2]; z<=upper_corner[2]; z=z+spacs[2]) {
01557                 if ( VABS(z - lower_corner[2]) < VSMALL) {
01558                     wz = 0.5;
01559                 } else if ( VABS(z - upper_corner[2]) < VSMALL) {
01560                     wz = 0.5;
01561                 } else {
01562                     wz = 1.0;
01563                 }
01564                 vec[2] = z;
01565
01566                 w = wx*wy*wz;
01567
01568                 sav += (w*(1.0-Vacc_ivdwAcc(thee, vec, radius)));
01569
01570             } /* z loop */
01571         } /* y loop */
01572     } /* x loop */
01573
01574     w  = spacs[0]*spacs[1]*spacs[2];
01575     sav *= w;
01576
01577     return sav;
01578 }
01579
01580 int Vacc_wcaEnergyAtom(Vacc *thee, APOLparm *apolparm, Valist *alist,
01581                                Vclist *clist, int iatom, double *value) {
01582
01583     int i;
01584     int npts[3];
01585     int pad = 14;
01586
01587         int xmin, ymin, zmin;
01588         int xmax, ymax, zmax;
01589
01590         double sigma6, sigma12;
```

```
01591
01592     double spacs[3], vec[3];
01593     double w, wx, wy, wz, len, fn, x, y, z, vol;
01594     double x2,y2,z2,r;
01595     double vol_density, energy, rho, srad;
01596     double psig, epsilon, watepsilon, sigma, watsigma, eni, chi;
01597
01598     double *pos;
01599     double *lower_corner, *upper_corner;
01600
01601     Vatom *atom = VNULL;
01602     VASSERT(apolparm != VNULL);
01603
01604     energy = 0.0;
01605     vol = 1.0;
01606     vol_density = 2.0;
01607
01608     lower_corner = clist->lower_corner;
01609     upper_corner = clist->upper_corner;
01610
01611     atom = Valist_getAtom(alist, iatom);
01612     pos = Vatom_getPosition(atom);
01613
01614     /* Note:  these are the original temporary water parameters... they have been
01615         replaced by entries in a parameter file:
01616     watsigma = 1.7683;
01617     watepsilon =  0.1521;
01618     watepsilon = watepsilon*4.184;
01619     */
01620
01621     srad = apolparm->srad;
01622     rho = apolparm->bconc;
01623     watsigma = apolparm->watsigma;
01624     watepsilon = apolparm->watepsilon;
01625     psig = atom->radius;
01626     epsilon = atom->epsilon;
01627     sigma = psig + watsigma;
01628     epsilon = VSQRT((epsilon * watepsilon));
01629
01630     /* parameters */
01631     sigma6 = VPOW(sigma,6);
01632     sigma12 = VPOW(sigma,12);
01633     /* OPLS-style radius:  double sigmar = sigma*VPOW(2, (1.0/6.0)); */
01634
01635     xmin = pos[0] - pad;
01636     xmax = pos[0] + pad;
01637     ymin = pos[1] - pad;
01638     ymax = pos[1] + pad;
01639     zmin = pos[2] - pad;
01640     zmax = pos[2] + pad;
01641
01642     for (i=0; i<3; i++) {
01643         len = (upper_corner[i] + pad) - (lower_corner[i] - pad);
01644         vol *= len;
01645         fn = len*vol_density + 1;
01646         npts[i] = (int)ceil(fn);
01647         spacs[i] = 0.5;
01648         if (apolparm->setgrid) {
01649             if (apolparm->grid[i] > spacs[i]) {
01650                 Vnm_print(2, "Vacc_totalSAV:  Warning, your GRID value (%g) is larger than the recommended
    value (%g)!\n",
01651                       apolparm->grid[i], spacs[i]);
01652             }
01653             spacs[i] = apolparm->grid[i];
01654         }
01655     }
01656
01657     for (x=xmin; x<=xmax; x=x+spacs[0]) {
01658         if ( VABS(x - xmin) < VSMALL) {
01659             wx = 0.5;
01660         } else if ( VABS(x - xmax) < VSMALL) {
01661             wx = 0.5;
01662         } else {
01663             wx = 1.0;
01664         }
01665         vec[0] = x;
01666         for (y=ymin; y<=ymax; y=y+spacs[1]) {
01667             if ( VABS(y - ymin) < VSMALL) {
01668                 wy = 0.5;
01669             } else if ( VABS(y - ymax) < VSMALL) {
01670                 wy = 0.5;
```

```
01671             } else {
01672                 wy = 1.0;
01673             }
01674             vec[1] = y;
01675             for (z=zmin; z<=zmax; z=z+spacs[2]) {
01676                 if ( VABS(z - zmin) < VSMALL) {
01677                     wz = 0.5;
01678                 } else if ( VABS(z - zmax) < VSMALL) {
01679                     wz = 0.5;
01680                 } else {
01681                     wz = 1.0;
01682                 }
01683                 vec[2] = z;
01684 
01685                 w = wx*wy*wz;
01686 
01687                 chi = Vacc_ivdwAcc(thee, vec, srad);
01688 
01689                 if (VABS(chi) > VSMALL) {
01690 
01691                     x2 = VSQR(vec[0]-pos[0]);
01692                     y2 = VSQR(vec[1]-pos[1]);
01693                     z2 = VSQR(vec[2]-pos[2]);
01694                     r = VSQRT(x2+y2+z2);
01695 
01696                     if (r <= 14 && r >= sigma) {
01697                         eni = chi*rho*epsilon*(-2.0*sigma6/VPOW(r,6)+sigma12/VPOW(r,12));
01698                     }else if (r <= 14){
01699                         eni = -1.0*epsilon*chi*rho;
01700                     }else{
01701                         eni = 0.0;
01702                     }
01703                 }else{
01704                     eni = 0.0;
01705                 }
01706 
01707                 energy += eni*w;
01708 
01709             } /* z loop */
01710         } /* y loop */
01711     } /* x loop */
01712 
01713     w  = spacs[0]*spacs[1]*spacs[2];
01714     energy *= w;
01715 
01716     *value = energy;
01717 
01718     return VRC_SUCCESS;
01719 }
01720 
01721 VPUBLIC int Vacc_wcaEnergy(Vacc *acc, APOLparm *apolparm, Valist *alist,
01722                            Vclist *clist){
01723 
01724     int iatom;
01725     int rc = 0;
01726 
01727     double energy = 0.0;
01728     double tenergy = 0.0;
01729     double rho = apolparm->bconc;
01730 
01731     /* Do a sanity check to make sure that watepsilon and watsigma are set
01732      * If not, return with an error. */
01733     if(apolparm->setwat == 0){
01734         Vnm_print(2,"Vacc_wcaEnergy: Error. No value was set for watsigma and watepsilon.\n");
01735         return VRC_FAILURE;
01736     }
01737 
01738     if (VABS(rho) < VSMALL) {
01739         apolparm->wcaEnergy = tenergy;
01740         return 1;
01741     }
01742 
01743     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++){
01744         rc = Vacc_wcaEnergyAtom(acc, apolparm, alist, clist, iatom, &energy);
01745         if(rc == 0) return 0;
01746 
01747         tenergy += energy;
01748     }
01749 
01750     apolparm->wcaEnergy = tenergy;
01751 
```

```
01752     return VRC_SUCCESS;
01753
01754 }
01755
01756 VPUBLIC int Vacc_wcaForceAtom(Vacc *thee,
01757                               APOLparm *apolparm,
01758                               Vclist *clist,
01759                               Vatom *atom,
01760                               double *force
01761                              ){
01762     int i,
01763         si,
01764         npts[3],
01765         pad = 14,
01766         xmin,
01767         ymin,
01768         zmin,
01769         xmax,
01770         ymax,
01771         zmax;
01772
01773     double sigma6,
01774            sigma12,
01775            spacs[3],
01776            vec[3],
01777            fpt[3],
01778            w,
01779            wx,
01780            wy,
01781            wz,
01782            len,
01783            fn,
01784            x,
01785            y,
01786            z,
01787            vol,
01788            x2,
01789            y2,
01790            z2,
01791            r,
01792            vol_density,
01793            fo,
01794            rho,
01795            srad,
01796            psig,
01797            epsilon,
01798            watepsilon,
01799            sigma,
01800            watsigma,
01801            chi,
01802            *pos,
01803            *lower_corner,
01804            *upper_corner;
01805
01806     /* Allocate needed variables now that we've asserted required conditions. */
01807     time_t ts;
01808     ts = clock();
01809
01810     VASSERT(apolparm != VNULL);
01811
01812     /* Do a sanity check to make sure that watepsilon and watsigma are set
01813      * If not, return with an error. */
01814     if(apolparm->setwat == 0){
01815         Vnm_print(2,"Vacc_wcaEnergy: Error. No value was set for watsigma and watepsilon.\n");
01816         return VRC_FAILURE;
01817     }
01818
01819     vol = 1.0;
01820     vol_density = 2.0;
01821
01822     lower_corner = clist->lower_corner;
01823     upper_corner = clist->upper_corner;
01824
01825     pos = Vatom_getPosition(atom);
01826
01827     srad = apolparm->srad;
01828     rho = apolparm->bconc;
01829     watsigma = apolparm->watsigma;
01830     watepsilon = apolparm->watepsilon;
01831
01832     psig = atom->radius;
```

```
01833     epsilon = atom->epsilon;
01834     sigma = psig + watsigma;
01835     epsilon = VSQRT((epsilon * watepsilon));
01836 
01837     /* parameters */
01838     sigma6 = VPOW(sigma,6);
01839     sigma12 = VPOW(sigma,12);
01840     /* OPLS-style radius:  double sigmar = sigma*VPOW(2, (1.0/6.0));  */
01841 
01842     for (i=0; i<3; i++) {
01843         len = (upper_corner[i] + pad) - (lower_corner[i] - pad);
01844         vol *= len;
01845         fn = len*vol_density + 1;
01846         npts[i] = (int)ceil(fn);
01847         spacs[i] = 0.5;
01848         force[i] = 0.0;
01849         if (apolparm->setgrid) {
01850             if (apolparm->grid[i] > spacs[i]) {
01851                 Vnm_print(2, "Vacc_totalSAV:  Warning, your GRID value (%g) is larger than the recommended
      value (%g)!\n",
01852                         apolparm->grid[i], spacs[i]);
01853             }
01854             spacs[i] = apolparm->grid[i];
01855         }
01856     }
01857 
01858     xmin = pos[0] - pad;
01859     xmax = pos[0] + pad;
01860     ymin = pos[1] - pad;
01861     ymax = pos[1] + pad;
01862     zmin = pos[2] - pad;
01863     zmax = pos[2] + pad;
01864 
01865     for (x=xmin; x<=xmax; x=x+spacs[0]) {
01866         if ( VABS(x - xmin) < VSMALL) {
01867             wx = 0.5;
01868         } else if ( VABS(x - xmax) < VSMALL) {
01869             wx = 0.5;
01870         } else {
01871             wx = 1.0;
01872         }
01873         vec[0] = x;
01874         for (y=ymin; y<=ymax; y=y+spacs[1]) {
01875             if ( VABS(y - ymin) < VSMALL) {
01876                 wy = 0.5;
01877             } else if ( VABS(y - ymax) < VSMALL) {
01878                 wy = 0.5;
01879             } else {
01880                 wy = 1.0;
01881             }
01882             vec[1] = y;
01883             for (z=zmin; z<=zmax; z=z+spacs[2]) {
01884                 if ( VABS(z - zmin) < VSMALL) {
01885                     wz = 0.5;
01886                 } else if ( VABS(z - zmax) < VSMALL) {
01887                     wz = 0.5;
01888                 } else {
01889                     wz = 1.0;
01890                 }
01891                 vec[2] = z;
01892 
01893                 w = wx*wy*wz;
01894 
01895                 chi = Vacc_ivdwAcc(thee, vec, srad);
01896 
01897                 if (chi != 0.0) {
01898 
01899                     x2 = VSQR(vec[0]-pos[0]);
01900                     y2 = VSQR(vec[1]-pos[1]);
01901                     z2 = VSQR(vec[2]-pos[2]);
01902                     r = VSQRT(x2+y2+z2);
01903 
01904                     if (r <= 14 && r >= sigma){
01905 
01906                         fo = 12.0*chi*rho*epsilon*(sigma6/VPOW(r,7)-sigma12/VPOW(r,13));
01907 
01908                         fpt[0] = -1.0*(pos[0]-vec[0])*fo/r;
01909                         fpt[1] = -1.0*(pos[1]-vec[1])*fo/r;
01910                         fpt[2] = -1.0*(pos[2]-vec[2])*fo/r;
01911 
01912                     }else {
```

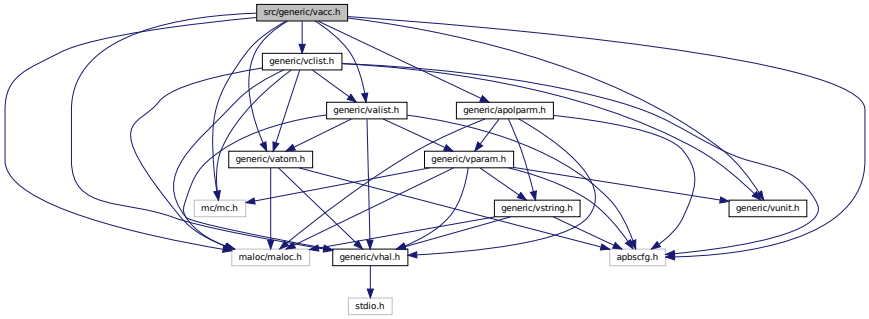
```
01913                         for (si=0; si < 3; si++) fpt[si] = 0.0;
01914                     }
01915                 }else {
01916                     for (si=0; si < 3; si++) fpt[si] = 0.0;
01917                 }
01918 
01919                 for(i=0;i<3;i++){
01920                     force[i] += (w*fpt[i]);
01921                 }
01922 
01923             } /* z loop */
01924         } /* y loop */
01925     } /* x loop */
01926 
01927     w  = spacs[0]*spacs[1]*spacs[2];
01928     for(i=0;i<3;i++) force[i] *= w;
01929 
01930     return VRC_SUCCESS;
01931 }
```

9.49 src/generic/vacc.h File Reference

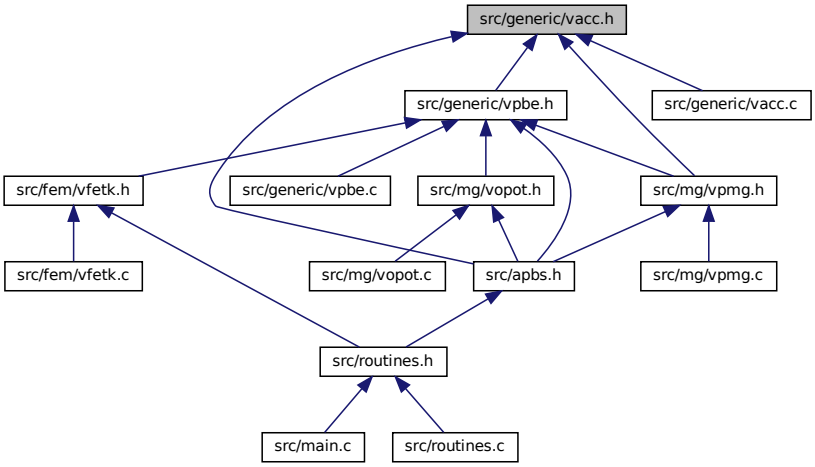
Contains declarations for class Vacc.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "mc/mc.h"
#include "generic/vhal.h"
#include "generic/valist.h"
#include "generic/vclist.h"
#include "generic/vatom.h"
#include "generic/vunit.h"
#include "generic/apolparm.h"
```

Include dependency graph for vacc.h:



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



Data Structures

- struct sVaccSurf

  Surface object list of per-atom surface points.
- struct sVacc

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

Typedefs

- typedef struct sVaccSurf VaccSurf

  Declaration of the VaccSurf class as the VaccSurf structure.
- typedef struct sVacc Vacc

  Declaration of the Vacc class as the Vacc structure.

Functions

- VEXTERNC unsigned long int Vacc_memChk (Vacc *thee)

  Get number of bytes in this object and its members.
- VEXTERNC VaccSurf * VaccSurf_ctor (Vmem *mem, double probe_radius, int nsphere)

  Allocate and construct the surface object; do not assign surface points to positions.
- VEXTERNC int VaccSurf_ctor2 (VaccSurf *thee, Vmem *mem, double probe_radius, int nsphere)

  Construct the surface object using previously allocated memory; do not assign surface points to positions.
- VEXTERNC void VaccSurf_dtor (VaccSurf **thee)

  Destroy the surface object and free its memory.
- VEXTERNC void VaccSurf_dtor2 (VaccSurf *thee)

  Destroy the surface object.
- VEXTERNC VaccSurf * VaccSurf_refSphere (Vmem *mem, int npts)

  Set up an array of points for a reference sphere of unit radius.
- VEXTERNC VaccSurf * Vacc_atomSurf (Vacc *thee, Vatom *atom, VaccSurf *ref, double probe_radius)

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

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

  Construct the accessibility object.
- VEXTERNC int Vacc_ctor2 (Vacc *thee, Valist *alist, Vclist *clist, double surf_density)

  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.

9.49.1 Detailed Description

Contains declarations for class Vacc.

Version

Id

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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.

9.50 vacc.h

```
00001
00062 #ifndef _VACC_H_
00063 #define _VACC_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "maloc/maloc.h"
00068 #if defined(HAVE_MC_H)
00069 #include "mc/mc.h"
00070 #endif
00071
00072 #include "generic/vhal.h"
00073 #include "generic/valist.h"
00074 #include "generic/vclist.h"
00075 #include "generic/vatom.h"
00076 #include "generic/vunit.h"
00077 #include "generic/apolparm.h"
00078
00084 struct sVaccSurf {
00085     Vmem *mem;
00086     double *xpts;
00087     double *ypts;
00088     double *zpts;
00089     char *bpts;
00091     double area;
00092     int npts;
00093     double probe_radius;
00095 };
00096
00101 typedef struct sVaccSurf VaccSurf;
00102
00108 struct sVacc {
00109
00110   Vmem *mem;
00111   Valist *alist;
00112   Vclist *clist;
00113   int *atomFlags;
00116   VaccSurf *refSphere;
00117   VaccSurf **surf;
00120   Vset acc;
00122   double surf_density;
00125 };
00126
00131 typedef struct sVacc Vacc;
00132
00133 #if !defined(VINLINE_VACC)
00134
```

```
00140     VEXTERNC unsigned long int Vacc_memChk(
00141             Vacc *thee
00142             );
00143
00144 #else /* if defined(VINLINE_VACC) */
00145
00146 #   define Vacc_memChk(thee) (Vmem_bytes((thee)->mem))
00147
00148 #endif /* if !defined(VINLINE_VACC) */
00149
00157 VEXTERNC VaccSurf* VaccSurf_ctor(
00158         Vmem *mem,
00159         double probe_radius,
00160         int nsphere
00161         );
00162
00170 VEXTERNC int VaccSurf_ctor2(
00171         VaccSurf *thee,
00172         Vmem *mem,
00173         double probe_radius,
00174         int nsphere
00175         );
00176
00182 VEXTERNC void VaccSurf_dtor(
00183         VaccSurf **thee
00184         );
00185
00191 VEXTERNC void VaccSurf_dtor2(
00192         VaccSurf *thee
00193         );
00194
00209 VEXTERNC VaccSurf* VaccSurf_refSphere(
00210         Vmem *mem,
00211         int npts
00212         );
00213
00221 VEXTERNC VaccSurf* Vacc_atomSurf(
00222         Vacc *thee,
00223         Vatom *atom,
00224         VaccSurf *ref,
00226         double probe_radius
00227         );
00228
00233 VEXTERNC Vacc* Vacc_ctor(
00234         Valist *alist,
00235         Vclist *clist,
00237         double surf_density
00239         );
00240
00245 VEXTERNC int Vacc_ctor2(
00246         Vacc *thee,
00247         Valist *alist,
00248         Vclist *clist,
00250         double surf_density
00252         );
00253
00258 VEXTERNC void Vacc_dtor(
00259         Vacc **thee
00260         );
00261
00266 VEXTERNC void Vacc_dtor2(
00267         Vacc *thee
00268         );
00269
00280 VEXTERNC double Vacc_vdwAcc(
00281         Vacc *thee,
00282         double center[VAPBS_DIM]
00283         );
00284
00296 VEXTERNC double Vacc_ivdwAcc(
00297         Vacc *thee,
00298         double center[VAPBS_DIM],
00299         double radius
00300         );
00301
00316 VEXTERNC double Vacc_molAcc(
00317         Vacc *thee,
00318         double center[VAPBS_DIM],
00319         double radius
00320         );
00321
```

```
00340 VEXTERNC double Vacc_fastMolAcc(
00341         Vacc *thee,
00342         double center[VAPBS_DIM],
00343         double radius
00344         );
00345
00357 VEXTERNC double Vacc_splineAcc(
00358         Vacc *thee,
00359         double center[VAPBS_DIM],
00360         double win,
00361         double infrad
00362         );
00363
00369 VEXTERNC void Vacc_splineAccGrad(
00370         Vacc *thee,
00371         double center[VAPBS_DIM],
00372         double win,
00373         double infrad,
00374         double *grad
00375         );
00376
00388 VEXTERNC double Vacc_splineAccAtom(
00389         Vacc *thee,
00390         double center[VAPBS_DIM],
00391         double win,
00392         double infrad,
00393         Vatom *atom
00394         );
00395
00406 VEXTERNC void Vacc_splineAccGradAtomUnnorm(
00407         Vacc *thee,
00408         double center[VAPBS_DIM],
00409         double win,
00410         double infrad,
00411         Vatom *atom,
00412         double *force
00413         );
00414
00426 VEXTERNC void Vacc_splineAccGradAtomNorm(
00427         Vacc *thee,
00428         double center[VAPBS_DIM],
00429         double win,
00430         double infrad,
00431         Vatom *atom,
00432         double *force
00433         );
00434
00442 VEXTERNC void Vacc_splineAccGradAtomNorm4(
00443         Vacc *thee,
00444         double center[VAPBS_DIM],
00445         double win,
00446         double infrad,
00447         Vatom *atom,
00448         double *force
00449         );
00450
00458 VEXTERNC void Vacc_splineAccGradAtomNorm3(
00459         Vacc *thee,
00460         double center[VAPBS_DIM],
00461         double win,
00462         double infrad,
00463         Vatom *atom,
00464         double *force
00465         );
00466
00467
00477 VEXTERNC double Vacc_SASA(
00478         Vacc *thee,
00479         double radius
00480         );
00481
00489 VEXTERNC double Vacc_totalSASA(
00490         Vacc *thee,
00491         double radius
00492         );
00493
00501 VEXTERNC double Vacc_atomSASA(
00502         Vacc *thee,
00503         double radius,
00504         Vatom *atom
00505         );
```

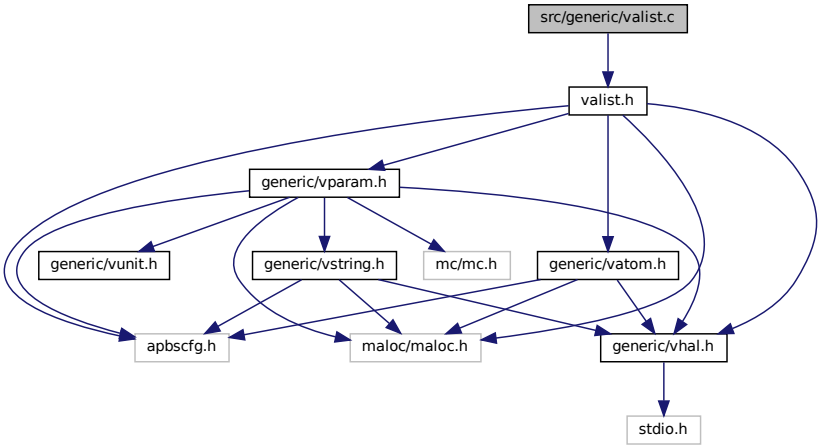
```
00506
00513 VEXTERNC VaccSurf* Vacc_atomSASPoints(
00514         Vacc *thee,
00515         double radius,
00516         Vatom *atom
00517         );
00518
00524 VEXTERNC void Vacc_atomdSAV(
00525                             Vacc *thee,
00526                             double radius,
00527                             Vatom *atom,
00528                             double *dSA
00529                             );
00530
00536 VEXTERNC void Vacc_atomdSASA(
00537                             Vacc *thee,
00538                             double dpos,
00539                             double radius,
00540                             Vatom *atom,
00541                             double *dSA
00542                             );
00543
00549 VEXTERNC void Vacc_totalAtomdSASA(
00550                              Vacc *thee,
00551                              double dpos,
00552                              double radius,
00553                              Vatom *atom,
00554                              double *dSA
00555                              );
00556
00562 VEXTERNC void Vacc_totalAtomdSAV(
00563                                  Vacc *thee,
00564                                  double dpos,
00565                                  double radius,
00566                                  Vatom *atom,
00567                                  double *dSA,
00568                                  Vclist *clist
00569                                  );
00570
00578 VEXTERNC double Vacc_totalSAV(
00579     Vacc *thee,
00580     Vclist *clist,
00581     APOLparm *apolparm,
00583     double radius
00584     );
00585
00592 VEXTERNC int Vacc_wcaEnergy(
00593                            Vacc *thee,
00594                              APOLparm *apolparm,
00595                              Valist *alist,
00596                              Vclist *clist
00597                              );
00604 VEXTERNC int Vacc_wcaForceAtom(Vacc *thee,
00605                               APOLparm *apolparm,
00606                               Vclist *clist,
00607                               Vatom *atom,
00608                               double *force
00609                             );
00610
00616 VEXTERNC int Vacc_wcaEnergyAtom(
00617     Vacc *thee,
00618     APOLparm *apolparm,
00619     Valist *alist,
00620     Vclist *clist,
00621     int iatom,
00622     double *value
00623     );
00624
00625 #endif    /* ifndef _VACC_H_ */
```

9.51 src/generic/valist.c File Reference

Class Valist methods.

```
#include "valist.h"
```

Include dependency graph for valist.c:



Functions

- VPUBLIC double Valist_getCenterX (Valist *thee)

  Get x-coordinate of molecule center.
- VPUBLIC double Valist_getCenterY (Valist *thee)

  Get y-coordinate of molecule center.
- VPUBLIC double Valist_getCenterZ (Valist *thee)

  Get z-coordinate of molecule center.
- VPUBLIC Vatom * Valist_getAtomList (Valist *thee)

  Get actual array of atom objects from the list.
- VPUBLIC int Valist_getNumberAtoms (Valist *thee)

  Get number of atoms in the list.
- VPUBLIC Vatom * Valist_getAtom (Valist *thee, int i)

  Get pointer to particular atom in list.
- VPUBLIC unsigned long int Valist_memChk (Valist *thee)

  Get total memory allocated for this object and its members.
- VPUBLIC Valist * Valist_ctor ()

  Construct the atom list object.
- VPUBLIC Vrc_Codes Valist_ctor2 (Valist *thee)

  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 atom↩ Name[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<>"

9.51.1 Detailed Description

Class Valist methods.

Author

Nathan Baker

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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) 2002-2020, 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.

9.52 valist.c

```
00001
00056 #include "valist.h"
00057
00058 VEMBED(rcsid="$Id$")
00059
00060 VPRIVATE char *Valist_whiteChars = " \t\r\n";
00061 VPRIVATE char *Valist_commChars  = "#%";
00062 VPRIVATE char *Valist_xmlwhiteChars = " \t\r\n<>";
00063
00064 #if !defined(VINLINE_VATOM)
00065
00066 VPUBLIC double Valist_getCenterX(Valist *thee) {
00067
00068     if (thee == NULL) {
00069         Vnm_print(2, "Valist_getCenterX:  Found null pointer when getting the center of X coordinate!\n");
00070         VASSERT(0);
00071     }
00072     return thee->center[0];
00073
00074 }
00075
00076 VPUBLIC double Valist_getCenterY(Valist *thee) {
00077
00078     if (thee == NULL) {
00079         Vnm_print(2, "Valist_getCenterY:  Found null pointer when getting the center of Y coordinate!\n");
00080         VASSERT(0);
00081     }
00082     return thee->center[1];
00083
00084 }
00085 VPUBLIC double Valist_getCenterZ(Valist *thee) {
00086
00087     if (thee == NULL) {
00088         Vnm_print(2, "Valist_getCenterZ:  Found null pointer when getting the center of Z coordinate!\n");
00089         VASSERT(0);
00090     }
00091     return thee->center[2];
00092
00093 }
00094
00095 VPUBLIC Vatom* Valist_getAtomList(Valist *thee) {
00096
00097     if (thee == NULL) {
00098         Vnm_print(2, "Valist_getAtomList:  Found null pointer when getting the atom list!\n");
00099         VASSERT(0);
00100     }
```

```
00101     return thee->atoms;
00102
00103 }
00104
00105 VPUBLIC int Valist_getNumberAtoms(Valist *thee) {
00106
00107     if (thee == NULL) {
00108         Vnm_print(2, "Valist_getNumberAtoms:  Found null pointer when getting the number of atoms!\n");
00109         VASSERT(0);
00110     }
00111     return thee->number;
00112
00113 }
00114
00115 VPUBLIC Vatom* Valist_getAtom(Valist *thee, int i) {
00116
00117     if (thee == NULL) {
00118         Vnm_print(2, "Valist_getAtom:  Found null pointer when getting atoms!\n");
00119         VASSERT(0);
00120     }
00121     if (i >= thee->number) {
00122         Vnm_print(2, "Valist_getAtom:  Requested atom number (%d) outside of atom list range (%d)!\n", i,
       thee->number);
00123         VASSERT(0);
00124     }
00125     return &(thee->atoms[i]);
00126
00127 }
00128
00129 VPUBLIC unsigned long int Valist_memChk(Valist *thee) {
00130
00131     if (thee == NULL) return 0;
00132     return Vmem_bytes(thee->vmem);
00133
00134 }
00135
00136 #endif /* if !defined(VINLINE_VATOM) */
00137
00138 VPUBLIC Valist* Valist_ctor() {
00139
00140     /* Set up the structure */
00141     Valist *thee = VNULL;
00142     thee = (Valist*)Vmem_malloc(VNULL, 1, sizeof(Valist));
00143     if ( thee == VNULL) {
00144         Vnm_print(2, "Valist_ctor:  Got NULL pointer when constructing the atom list object!\n");
00145         VASSERT(0);
00146     }
00147     if ( Valist_ctor2(thee) != VRC_SUCCESS) {
00148         Vnm_print(2, "Valist_ctor:   Error in constructing the atom list object!\n");
00149         VASSERT(0);
00150     }
00151
00152     return thee;
00153 }
00154
00155 VPUBLIC Vrc_Codes Valist_ctor2(Valist *thee) {
00156
00157     thee->atoms = VNULL;
00158     thee->number = 0;
00159
00160     /* Initialize the memory management object */
00161     thee->vmem = Vmem_ctor("APBS:VALIST");
00162
00163     return VRC_SUCCESS;
00164
00165 }
00166
00167 VPUBLIC void Valist_dtor(Valist **thee)
00168 {
00169     if ((*thee) != VNULL) {
00170         Valist_dtor2(*thee);
00171         Vmem_free(VNULL, 1, sizeof(Valist), (void **)thee);
00172         (*thee) = VNULL;
00173     }
00174 }
00175
00176 VPUBLIC void Valist_dtor2(Valist *thee) {
00177
00178     Vmem_free(thee->vmem, thee->number, sizeof(Vatom), (void **)&(thee->atoms));
00179     thee->atoms = VNULL;
00180     thee->number = 0;
```

```
00181
00182     Vmem_dtor(&(thee->vmem));
00183 }
00184
00185 /* Read serial number from PDB ATOM/HETATM field */
00186 VPRIVATE Vrc_Codes Valist_readPDBSerial(Valist *thee, Vio *sock, int *serial) {
00187
00188     char tok[VMAX_BUFSIZE];
00189     int ti = 0;
00190
00191     if (Vio_scanf(sock, "%s", tok) != 1) {
00192         Vnm_print(2, "Valist_readPDB:  Ran out of tokens while parsing serial!\n");
00193         return VRC_FAILURE;
00194     }
00195     if (sscanf(tok, "%d", &ti) != 1) {
00196         Vnm_print(2, "Valist_readPDB:  Unable to parse serial token (%s) as int!\n",
00197                 tok);
00198         return VRC_FAILURE;
00199     }
00200     *serial = ti;
00201
00202     return VRC_SUCCESS;
00203 }
00204
00205 /* Read atom name from PDB ATOM/HETATM field */
00206 VPRIVATE Vrc_Codes Valist_readPDBAtomName(Valist *thee, Vio *sock,
00207         char atomName[VMAX_ARGLEN]) {
00208
00209     char tok[VMAX_BUFSIZE];
00210
00211     if (Vio_scanf(sock, "%s", tok) != 1) {
00212         Vnm_print(2, "Valist_readPDB:  Ran out of tokens while parsing atom name!\n");
00213         return VRC_FAILURE;
00214     }
00215     if (strlen(tok) < VMAX_ARGLEN) strcpy(atomName, tok);
00216     else {
00217         Vnm_print(2, "Valist_readPDB:  Atom name (%s) too long!\n", tok);
00218         return VRC_FAILURE;
00219     }
00220     return VRC_SUCCESS;
00221 }
00222
00223 /* Read residue name from PDB ATOM/HETATM field */
00224 VPRIVATE Vrc_Codes Valist_readPDBResidueName(Valist *thee, Vio *sock,
00225         char resName[VMAX_ARGLEN]) {
00226
00227     char tok[VMAX_BUFSIZE];
00228
00229     if (Vio_scanf(sock, "%s", tok) != 1) {
00230         Vnm_print(2, "Valist_readPDB:  Ran out of tokens while parsing residue name!\n");
00231         return VRC_FAILURE;
00232     }
00233     if (strlen(tok) < VMAX_ARGLEN) strcpy(resName, tok);
00234     else {
00235         Vnm_print(2, "Valist_readPDB:  Residue name (%s) too long!\n", tok);
00236         return VRC_FAILURE;
00237     }
00238     return VRC_SUCCESS;
00239 }
00240
00241 /* Read residue number from PDB ATOM/HETATM field */
00242 VPRIVATE Vrc_Codes Valist_readPDBResidueNumber(
00243         Valist *thee, Vio *sock, int *resSeq) {
00244
00245     char tok[VMAX_BUFSIZE];
00246     char *resstring;
00247     int ti = 0;
00248
00249     if (Vio_scanf(sock, "%s", tok) != 1) {
00250         Vnm_print(2, "Valist_readPDB:  Ran out of tokens while parsing resSeq!\n");
00251         return VRC_FAILURE;
00252     }
00253     if (sscanf(tok, "%d", &ti) != 1) {
00254
00255         /* One of three things can happen here:
00256             1)  There is a chainID in the line:    THR A   1
00257             2)  The chainID is merged with resSeq: THR A1001
00258             3)  An actual error:                   THR foo
00259
00260         */
00261
```

```
00262         if (strlen(tok) == 1) {
00263             /* Case 1: Chain ID Present
00264                       Read the next field and hope its a float */
00265
00266             if (Vio_scanf(sock, "%s", tok) != 1) {
00267                 Vnm_print(2, "Valist_readPDB:  Ran out of tokens while parsing resSeq!\n");
00268                 return VRC_FAILURE;
00269             }
00270             if (sscanf(tok, "%d", &ti) != 1) {
00271                 Vnm_print(2, "Valist_readPDB:  Unable to parse resSeq token (%s) as int!\n",
00272                 tok);
00273                 return VRC_FAILURE;
00274             }
00275
00276         } else {
00277             /* Case 2: Chain ID, merged string.
00278                       Move pointer forward past the chainID and check
00279             */
00280             //strcpy(resstring, tok);
00281             resstring = tok;
00282             resstring++;
00283
00284             if (sscanf(resstring, "%d", &ti) != 1) {
00285                 /* Case 3:  More than one non-numeral char is present. Error.*/
00286                 Vnm_print(2, "Valist_readPDB:  Unable to parse resSeq token (%s) as int!\n",
00287                 resstring);
00288                 return VRC_FAILURE;
00289             }
00290         }
00291     }
00292     *resSeq = ti;
00293
00294     return VRC_SUCCESS;
00295 }
00296
00297 /* Read atom coordinate from PDB ATOM/HETATM field */
00298 VPRIVATE Vrc_Codes Valist_readPDBAtomCoord(Valist *thee, Vio *sock, double *coord) {
00299
00300     char tok[VMAX_BUFSIZE];
00301     double tf = 0;
00302
00303     if (Vio_scanf(sock, "%s", tok) != 1) {
00304         Vnm_print(2, "Valist_readPDB:  Ran out of tokens while parsing atom coordinate!\n");
00305         return VRC_FAILURE;
00306     }
00307     if (sscanf(tok, "%lf", &tf) != 1) {
00308         return VRC_FAILURE;
00309     }
00310     *coord = tf;
00311
00312     return VRC_SUCCESS;
00313 }
00314
00315 /* Read charge and radius from PQR ATOM/HETATM field */
00316 VPRIVATE Vrc_Codes Valist_readPDBChargeRadius(Valist *thee, Vio *sock,
00317         double *charge, double *radius) {
00318
00319     char tok[VMAX_BUFSIZE];
00320     double tf = 0;
00321
00322     if (Vio_scanf(sock, "%s", tok) != 1) {
00323         Vnm_print(2, "Valist_readPQR:  Ran out of tokens while parsing charge!\n");
00324         return VRC_FAILURE;
00325     }
00326     if (sscanf(tok, "%lf", &tf) != 1) {
00327         return VRC_FAILURE;
00328     }
00329     *charge = tf;
00330
00331     if (Vio_scanf(sock, "%s", tok) != 1) {
00332         Vnm_print(2, "Valist_readPQR:  Ran out of tokens while parsing radius!\n");
00333         return VRC_FAILURE;
00334     }
00335     if (sscanf(tok, "%lf", &tf) != 1) {
00336         return VRC_FAILURE;
00337     }
00338     *radius = tf;
00339
00340     return VRC_SUCCESS;
00341 }
00342
```

```
00343 /* Read ATOM/HETATM field of PDB through the X/Y/Z fields */
00344 VPRIVATE Vrc_Codes Valist_readPDB_throughXYZ(
00345         Valist *thee,
00346         Vio *sock, /* Socket ready for reading */
00347         int *serial, /* Set to atom number */
00348         char atomName[VMAX_ARGLEN], /* Set to atom name */
00349         char resName[VMAX_ARGLEN], /* Set to residue name */
00350         int *resSeq, /* Set to residue number */
00351         double *x, /* Set to x-coordinate */
00352         double *y, /* Set to y-coordinate */
00353         double *z  /* Set to z-coordinate */
00354         ) {
00355
00356
00357     int i, njunk, gotit;
00358
00359     /* Grab serial */
00360     if (Valist_readPDBSerial(thee, sock, serial) == VRC_FAILURE) {
00361         Vnm_print(2, "Valist_readPDB:  Error while parsing serial!\n");
00362     }
00363
00364     /* Grab atom name */
00365     if (Valist_readPDBAtomName(thee, sock, atomName) == VRC_FAILURE) {
00366         Vnm_print(2, "Valist_readPDB:  Error while parsing atom name!\n");
00367         return VRC_FAILURE;
00368     }
00369
00370     /* Grab residue name */
00371     if (Valist_readPDBResidueName(thee, sock, resName) == VRC_FAILURE) {
00372         Vnm_print(2, "Valist_readPDB:  Error while parsing residue name!\n");
00373         return VRC_FAILURE;
00374     }
00375
00376
00377     /* Grab residue number */
00378     if (Valist_readPDBResidueNumber(thee, sock, resSeq) == VRC_FAILURE) {
00379         Vnm_print(2, "Valist_readPDB:  Error while parsing residue number!\n");
00380         return VRC_FAILURE;
00381     }
00382
00383
00384     /* Read tokens until we find one that can be parsed as an atom
00385      * x-coordinate.  We will allow njunk=1 intervening field that
00386      * cannot be parsed as a coordinate */
00387     njunk = 1;
00388     gotit = 0;
00389     for (i=0; i<(njunk+1); i++) {
00390         if (Valist_readPDBAtomCoord(thee, sock, x) == VRC_SUCCESS) {
00391             gotit = 1;
00392             break;
00393         }
00394     }
00395     if (!gotit) {
00396         Vnm_print(2, "Valist_readPDB:  Can't find x!\n");
00397         return VRC_FAILURE;
00398     }
00399     /* Read y-coordinate */
00400     if (Valist_readPDBAtomCoord(thee, sock, y) == VRC_FAILURE) {
00401         Vnm_print(2, "Valist_readPDB:  Can't find y!\n");
00402         return VRC_FAILURE;
00403     }
00404     /* Read z-coordinate */
00405     if (Valist_readPDBAtomCoord(thee, sock, z) == VRC_FAILURE) {
00406         Vnm_print(2, "Valist_readPDB:  Can't find z!\n");
00407         return VRC_FAILURE;
00408     }
00409
00410 #if 0 /* Set to 1 if you want to debug */
00411     Vnm_print(1, "Valist_readPDB:  serial = %d\n", *serial);
00412     Vnm_print(1, "Valist_readPDB:  atomName = %s\n", atomName);
00413     Vnm_print(1, "Valist_readPDB:  resName = %s\n", resName);
00414     Vnm_print(1, "Valist_readPDB:  resSeq = %d\n", *resSeq);
00415     Vnm_print(1, "Valist_readPDB:  pos = (%g, %g, %g)\n",
00416               *x, *y, *z);
00417 #endif
00418
00419     return VRC_SUCCESS;
00420 }
00421
00422 /* Get a the next available atom storage location, increasing the storage
00423  * space if necessary.  Return VNULL if something goes wrong. */
```

```
00424 VPRIVATE Vatom* Valist_getAtomStorage(
00425         Valist *thee,
00426         Vatom **plist, /* Pointer to existing list of atoms */
00427         int *pnlist, /* Size of existing list, may be changed */
00428         int *pnatoms /* Existing number of atoms in list; incremented
00429                       before exit */
00430         ) {
00431
00432     Vatom *oldList, *newList, *theList;
00433     Vatom *oldAtom, *newAtom;
00434     int iatom, inext, oldLength, newLength, natoms;
00435
00436     newList = VNULL;
00437
00438     /* See if we need more space */
00439     if (*pnatoms >= *pnlist) {
00440
00441         /* Double the storage space */
00442         oldLength = *pnlist;
00443         newLength = 2*oldLength;
00444         newList = (Vatom*)Vmem_malloc(thee->vmem, newLength, sizeof(Vatom));
00445         oldList = *plist;
00446
00447         /* Check the allocation */
00448         if (newList == VNULL) {
00449             Vnm_print(2, "Valist_readPDB:  failed to allocate space for %d (Vatom)s!\n", newLength);
00450             return VNULL;
00451         }
00452
00453         /* Copy the atoms over */
00454         natoms = *pnatoms;
00455         for (iatom=0; iatom<natoms; iatom++) {
00456             oldAtom = &(oldList[iatom]);
00457             newAtom = &(newList[iatom]);
00458             Vatom_copyTo(oldAtom, newAtom);
00459             Vatom_dtor2(oldAtom);
00460         }
00461
00462         /* Free the old list */
00463         Vmem_free(thee->vmem, oldLength, sizeof(Vatom), (void **)plist);
00464
00465         /* Copy new list to plist */
00466         *plist = newList;
00467         *pnlist = newLength;
00468     }
00469
00470     theList = *plist;
00471     inext = *pnatoms;
00472
00473     /* Get the next available spot and increment counters */
00474     newAtom = &(theList[inext]);
00475     *pnatoms = inext + 1;
00476
00477     return newAtom;
00478 }
00479
00480 VPRIVATE Vrc_Codes Valist_setAtomArray(Valist *thee,
00481         Vatom **plist, /* Pointer to list of atoms to store */
00482         int nlist, /* Length of list */
00483         int natoms /* Number of real atom entries in list */
00484         ) {
00485
00486     Vatom *list, *newAtom, *oldAtom;
00487     int i;
00488
00489     list = *plist;
00490
00491     /* Allocate necessary space */
00492     thee->number = 0;
00493     thee->atoms = (Vatom*)Vmem_malloc(thee->vmem, natoms, sizeof(Vatom));
00494     if (thee->atoms == VNULL) {
00495         Vnm_print(2, "Valist_readPDB:  Unable to allocate space for %d (Vatom)s!\n",
00496                 natoms);
00497         return VRC_FAILURE;
00498     }
00499     thee->number = natoms;
00500
00501     /* Copy over data */
00502     for (i=0; i<thee->number; i++) {
00503         newAtom = &(thee->atoms[i]);
00504         oldAtom = &(list[i]);
```

```
00505         Vatom_copyTo(oldAtom, newAtom);
00506         Vatom_dtor2(oldAtom);
00507     }
00508
00509     /* Free old array */
00510     Vmem_free(thee->vmem, nlist, sizeof(Vatom), (void **)plist);
00511
00512     return VRC_SUCCESS;
00513 }
00514
00515 VPUBLIC Vrc_Codes Valist_readPDB(Valist *thee, Vparam *param, Vio *sock) {
00516
00517     /* WE DO NOT DIRECTLY CONFORM TO PDB STANDARDS -- TO ALLOW LARGER FILES, WE
00518      * REQUIRE ALL FIELDS TO BE WHITESPACE DELIMITED */
00519
00520     Vatom *atoms = VNULL;
00521     Vatom *nextAtom = VNULL;
00522     Vparam_AtomData *atomData = VNULL;
00523
00524     char tok[VMAX_BUFSIZE];
00525     char atomName[VMAX_ARGLEN], resName[VMAX_ARGLEN];
00526
00527     int nlist, natoms, serial, resSeq;
00528
00529     double x, y, z, charge, radius, epsilon;
00530     double pos[3];
00531
00532     if (thee == VNULL) {
00533         Vnm_print(2, "Valist_readPDB:  Got NULL pointer when reading PDB file!\n");
00534         VASSERT(0);
00535     }
00536     thee->number = 0;
00537
00538     Vio_setWhiteChars(sock, Valist_whiteChars);
00539     Vio_setCommChars(sock, Valist_commChars);
00540
00541     /* Allocate some initial space for the atoms */
00542     nlist = 200;
00543     atoms = (Vatom*)Vmem_malloc(thee->vmem, nlist, sizeof(Vatom));
00544
00545     natoms = 0;
00546     /* Read until we run out of lines */
00547     while (Vio_scanf(sock, "%s", tok) == 1) {
00548
00549         /* Parse only ATOM/HETATOM fields */
00550         if ((Vstring_strcasecmp(tok, "ATOM") == 0) ||
00551             (Vstring_strcasecmp(tok, "HETATM") == 0)) {
00552
00553             /* Read ATOM/HETATM field of PDB through the X/Y/Z fields */
00554             if (Valist_readPDB_throughXYZ(thee, sock, &serial, atomName,
00555                         resName, &resSeq, &x, &y, &z) == VRC_FAILURE) {
00556                 Vnm_print(2, "Valist_readPDB:  Error parsing atom %d!\n",
00557                           serial);
00558                 return VRC_FAILURE;
00559             }
00560
00561             /* Try to find the parameters. */
00562             atomData = Vparam_getAtomData(param, resName, atomName);
00563             if (atomData == VNULL) {
00564                 Vnm_print(2, "Valist_readPDB:  Couldn't find parameters for \
00565 atom = %s, residue = %s\n", atomName, resName);
00566                 return VRC_FAILURE;
00567             }
00568             charge = atomData->charge;
00569             radius = atomData->radius;
00570             epsilon = atomData->epsilon;
00571
00572             /* Get pointer to next available atom position */
00573             nextAtom = Valist_getAtomStorage(thee, &atoms, &nlist, &natoms);
00574             if (nextAtom == VNULL) {
00575                 Vnm_print(2, "Valist_readPDB:  Error in allocating spacing for atoms!\n");
00576                 return VRC_FAILURE;
00577             }
00578
00579             /* Store the information */
00580             pos[0] = x; pos[1] = y; pos[2] = z;
00581             Vatom_setPosition(nextAtom, pos);
00582             Vatom_setCharge(nextAtom, charge);
00583             Vatom_setRadius(nextAtom, radius);
00584             Vatom_setEpsilon(nextAtom, epsilon);
00585             Vatom_setAtomID(nextAtom, natoms-1);
```

```
00586             Vatom_setResName(nextAtom, resName);
00587             Vatom_setAtomName(nextAtom, atomName);
00588
00589         } /* if ATOM or HETATM */
00590     } /* while we haven't run out of tokens */
00591
00592     Vnm_print(0, "Valist_readPDB: Counted %d atoms\n", natoms);
00593     fflush(stdout);
00594
00595     /* Store atoms internally */
00596     if (Valist_setAtomArray(thee, &atoms, nlist, natoms) == VRC_FAILURE) {
00597         Vnm_print(2, "Valist_readPDB:  unable to store atoms!\n");
00598         return VRC_FAILURE;
00599     }
00600
00601     return Valist_getStatistics(thee);
00602
00603
00604 }
00605
00606 VPUBLIC Vrc_Codes Valist_readPQR(Valist *thee, Vparam *params, Vio *sock) {
00607
00608     /* WE DO NOT DIRECTLY CONFORM TO PDB STANDARDS -- TO ALLOW LARGER FILES, WE
00609      * REQUIRE ALL FIELDS TO BE WHITESPACE DELIMITED */
00610
00611
00612     Vatom *atoms = VNULL;
00613     Vatom *nextAtom = VNULL;
00614     Vparam_AtomData *atomData = VNULL;
00615
00616     char tok[VMAX_BUFSIZE];
00617     char atomName[VMAX_ARGLEN], resName[VMAX_ARGLEN];
00618     char chs[VMAX_BUFSIZE];
00619
00620     int use_params = 0;
00621     int nlist, natoms, serial, resSeq;
00622
00623     double x, y, z, charge, radius, epsilon;
00624     double pos[3];
00625
00626     epsilon = 0.0;
00627
00628     if (thee == VNULL) {
00629         Vnm_print(2, "Valist_readPQR:  Got NULL pointer when reading PQR file!\n");
00630         VASSERT(0);
00631     }
00632     thee->number = 0;
00633
00634     Vio_setWhiteChars(sock, Valist_whiteChars);
00635     Vio_setCommChars(sock, Valist_commChars);
00636
00637     /* Allocate some initial space for the atoms */
00638     nlist = 200;
00639     atoms = (Vatom*)Vmem_malloc(thee->vmem, nlist, sizeof(Vatom));
00640
00641     /* Check if we are using a parameter file or not */
00642     if(params != VNULL) use_params = 1;
00643
00644     natoms = 0;
00645     /* Read until we run out of lines */
00646     while (Vio_scanf(sock, "%s", tok) == 1) {
00647
00648         /* Parse only ATOM/HETATOM fields */
00649         if ((Vstring_strcasecmp(tok, "ATOM") == 0) ||
00650             (Vstring_strcasecmp(tok, "HETATM") == 0)) {
00651
00652             /* Read ATOM/HETATM field of PDB through the X/Y/Z fields */
00653             if (Valist_readPDB_throughXYZ(thee, sock, &serial, atomName,
00654                         resName, &resSeq, &x, &y, &z) == VRC_FAILURE) {
00655                 Vnm_print(2, "Valist_readPQR:  Error parsing atom %d!\n",serial);
00656                 Vnm_print(2, "Please double check this atom in the pqr file, e.g., make sure there are no
       concatenated fields.\n");
00657                 return VRC_FAILURE;
00658             }
00659
00660             /* Read Q/R fields */
00661             if (Valist_readPDBChargeRadius(thee, sock, &charge, &radius) == VRC_FAILURE) {
00662                 Vnm_print(2, "Valist_readPQR:  Error parsing atom %d!\n",
00663                           serial);
00664                 Vnm_print(2, "Please double check this atom in the pqr file, e.g., make sure there are no
       concatenated fields.\n");
```

```
00665                 return VRC_FAILURE;
00666             }
00667
00668             if(use_params){
00669                 /* Try to find the parameters. */
00670                 atomData = Vparam_getAtomData(params, resName, atomName);
00671                 if (atomData == VNULL) {
00672                     Vnm_print(2, "Valist_readPDB:  Couldn't find parameters for \
00673 atom = %s, residue = %s\n", atomName, resName);
00674                     return VRC_FAILURE;
00675                 }
00676                 charge = atomData->charge;
00677                 radius = atomData->radius;
00678                 epsilon = atomData->epsilon;
00679             }
00680
00681             /* Get pointer to next available atom position */
00682             nextAtom = Valist_getAtomStorage(thee, &atoms, &nlist, &natoms);
00683             if (nextAtom == VNULL) {
00684                 Vnm_print(2, "Valist_readPQR:  Error in allocating spacing for atoms!\n");
00685                 return VRC_FAILURE;
00686             }
00687
00688             /* Store the information */
00689             pos[0] = x; pos[1] = y; pos[2] = z;
00690             Vatom_setPosition(nextAtom, pos);
00691             Vatom_setCharge(nextAtom, charge);
00692             Vatom_setRadius(nextAtom, radius);
00693             Vatom_setEpsilon(nextAtom, epsilon);
00694             Vatom_setAtomID(nextAtom, natoms-1);
00695             Vatom_setResName(nextAtom, resName);
00696             Vatom_setAtomName(nextAtom, atomName);
00697
00698         } /* if ATOM or HETATM */
00699         else {
00700             /*
00701              * nop
00702              * Note that if we find a line that starts with something that's not
00703              * ATOM or HETATM we'll just keep parsing strings until we find one
00704              * of the acceptable keywords.
00705              * Extraordinary measures are not necessary, and only add  to the
00706              * befuddlement.
00707              */
00708         }
00709     } /* while we haven't run out of tokens */
00710
00711     Vnm_print(0, "Valist_readPQR: Counted %d atoms\n", natoms);
00712     fflush(stdout);
00713
00714     /* Store atoms internally */
00715     if (Valist_setAtomArray(thee, &atoms, nlist, natoms) == VRC_FAILURE) {
00716         Vnm_print(2, "Valist_readPDB:  unable to store atoms!\n");
00717         return VRC_FAILURE;
00718     }
00719
00720     return Valist_getStatistics(thee);
00721
00722
00723 }
00724
00725 VPUBLIC Vrc_Codes Valist_readXML(Valist *thee, Vparam *params, Vio *sock) {
00726
00727     Vatom *atoms = VNULL;
00728     Vatom *nextAtom = VNULL;
00729
00730     char tok[VMAX_BUFSIZE];
00731     char endtag[VMAX_BUFSIZE];
00732
00733     int nlist, natoms;
00734     int xset, yset, zset, chgset, radset;
00735
00736     double x, y, z, charge, radius, dtmp;
00737     double pos[3];
00738
00739     if (thee == VNULL) {
00740         Vnm_print(2, "Valist_readXML:  Got NULL pointer when reading XML file!\n");
00741         VASSERT(0);
00742     }
00743     thee->number = 0;
00744
00745     Vio_setWhiteChars(sock, Valist_xmlwhiteChars);
```

```
00746     Vio_setCommChars(sock, Valist_commChars);
00747
00748     /* Allocate some initial space for the atoms */
00749     nlist = 200;
00750     atoms = (Vatom*)Vmem_malloc(thee->vmem, nlist, sizeof(Vatom));
00751
00752     /* Initialize some variables */
00753     natoms = 0;
00754     xset = 0;
00755     yset = 0;
00756     zset = 0;
00757     chgset = 0;
00758     radset = 0;
00759     strcpy(endtag,"/");
00760
00761     if(params == VNULL){
00762         Vnm_print(1,"\nValist_readXML: Warning Warning Warning Warning Warning\n");
00763         Vnm_print(1,"Valist_readXML: The use of XML input files with parameter\n");
00764         Vnm_print(1,"Valist_readXML: files is currently not supported.\n");
00765         Vnm_print(1,"Valist_readXML: Warning Warning Warning Warning Warning\n\n");
00766     }
00767
00768     /* Read until we run out of lines */
00769     while (Vio_scanf(sock, "%s", tok) == 1) {
00770
00771         /* The first tag taken is the start tag - save it to detect end */
00772         if (Vstring_strcasecmp(endtag, "/") == 0) strcat(endtag, tok);
00773
00774         if (Vstring_strcasecmp(tok, "x") == 0) {
00775             Vio_scanf(sock, "%s", tok);
00776             if (sscanf(tok, "%lf", &dtmp) != 1) {
00777                 Vnm_print(2, "Valist_readXML:  Unexpected token (%s) while \
00778 reading x!\n", tok);
00779                 return VRC_FAILURE;
00780               }
00781             x = dtmp;
00782             xset = 1;
00783         } else if (Vstring_strcasecmp(tok, "y") == 0) {
00784             Vio_scanf(sock, "%s", tok);
00785             if (sscanf(tok, "%lf", &dtmp) != 1) {
00786                 Vnm_print(2, "Valist_readXML:  Unexpected token (%s) while \
00787 reading y!\n", tok);
00788                 return VRC_FAILURE;
00789               }
00790             y = dtmp;
00791             yset = 1;
00792         } else if (Vstring_strcasecmp(tok, "z") == 0) {
00793             Vio_scanf(sock, "%s", tok);
00794             if (sscanf(tok, "%lf", &dtmp) != 1) {
00795                 Vnm_print(2, "Valist_readXML:  Unexpected token (%s) while \
00796 reading z!\n", tok);
00797                 return VRC_FAILURE;
00798               }
00799             z = dtmp;
00800             zset = 1;
00801         } else if (Vstring_strcasecmp(tok, "charge") == 0) {
00802             Vio_scanf(sock, "%s", tok);
00803             if (sscanf(tok, "%lf", &dtmp) != 1) {
00804                 Vnm_print(2, "Valist_readXML:  Unexpected token (%s) while \
00805 reading charge!\n", tok);
00806                 return VRC_FAILURE;
00807               }
00808             charge = dtmp;
00809             chgset = 1;
00810         } else if (Vstring_strcasecmp(tok, "radius") == 0) {
00811             Vio_scanf(sock, "%s", tok);
00812             if (sscanf(tok, "%lf", &dtmp) != 1) {
00813                 Vnm_print(2, "Valist_readXML:  Unexpected token (%s) while \
00814 reading radius!\n", tok);
00815                 return VRC_FAILURE;
00816               }
00817             radius = dtmp;
00818             radset = 1;
00819         } else if (Vstring_strcasecmp(tok, "/atom") == 0) {
00820
00821           /* Get pointer to next available atom position */
00822             nextAtom = Valist_getAtomStorage(thee, &atoms, &nlist, &natoms);
00823             if (nextAtom == VNULL) {
00824                 Vnm_print(2, "Valist_readXML:  Error in allocating spacing for atoms!\n");
00825                 return VRC_FAILURE;
00826             }
```

```
00827
00828             if (xset && yset && zset && chgset && radset){
00829
00830                 /* Store the information */
00831                 pos[0] = x; pos[1] = y; pos[2] = z;
00832                 Vatom_setPosition(nextAtom, pos);
00833                 Vatom_setCharge(nextAtom, charge);
00834                 Vatom_setRadius(nextAtom, radius);
00835                 Vatom_setAtomID(nextAtom, natoms-1);
00836
00837                 /* Reset the necessary flags */
00838                 xset = 0;
00839                 yset = 0;
00840                 zset = 0;
00841                 chgset = 0;
00842                 radset = 0;
00843             } else {
00844                 Vnm_print(2,  "Valist_readXML:  Missing field(s) in atom tag:\n");
00845                 if (!xset) Vnm_print(2,"\tx value not set!\n");
00846                 if (!yset) Vnm_print(2,"\ty value not set!\n");
00847                 if (!zset) Vnm_print(2,"\tz value not set!\n");
00848                 if (!chgset) Vnm_print(2,"\tcharge value not set!\n");
00849                 if (!radset) Vnm_print(2,"\tradius value not set!\n");
00850                 return VRC_FAILURE;
00851             }
00852         } else if (Vstring_strcasecmp(tok, endtag) == 0) break;
00853     }
00854
00855     Vnm_print(0, "Valist_readXML: Counted %d atoms\n", natoms);
00856     fflush(stdout);
00857
00858     /* Store atoms internally */
00859     if (Valist_setAtomArray(thee, &atoms, nlist, natoms) == VRC_FAILURE) {
00860         Vnm_print(2, "Valist_readXML:  unable to store atoms!\n");
00861         return VRC_FAILURE;
00862     }
00863
00864     return Valist_getStatistics(thee);
00865
00866 }
00867
00868 /* Load up Valist with various statistics */
00869 VPUBLIC Vrc_Codes Valist_getStatistics(Valist *thee) {
00870
00871     Vatom *atom;
00872     int i, j;
00873
00874     if (thee == VNULL) {
00875         Vnm_print(2, "Valist_getStatistics:  Got NULL pointer when loading up Valist with various
      statistics!\n");
00876         VASSERT(0);
00877     }
00878
00879     thee->center[0] = 0.;
00880     thee->center[1] = 0.;
00881     thee->center[2] = 0.;
00882     thee->maxrad = 0.;
00883     thee->charge = 0.;
00884
00885     if (thee->number == 0) return VRC_FAILURE;
00886
00887     /* Reset stat variables */
00888     atom = &(thee->atoms[0]);
00889     for (i=0; i<3; i++) {
00890         thee->maxcrd[i] = thee->mincrd[i] = atom->position[i];
00891     }
00892     thee->maxrad = atom->radius;
00893     thee->charge = 0.0;
00894
00895     for (i=0; i<thee->number; i++) {
00896
00897         atom = &(thee->atoms[i]);
00898         for (j=0; j<3; j++) {
00899             if (atom->position[j] < thee->mincrd[j])
00900               thee->mincrd[j] = atom->position[j];
00901             if (atom->position[j] > thee->maxcrd[j])
00902               thee->maxcrd[j] = atom->position[j];
00903         }
00904         if (atom->radius > thee->maxrad) thee->maxrad = atom->radius;
00905         thee->charge = thee->charge + atom->charge;
00906     }
```

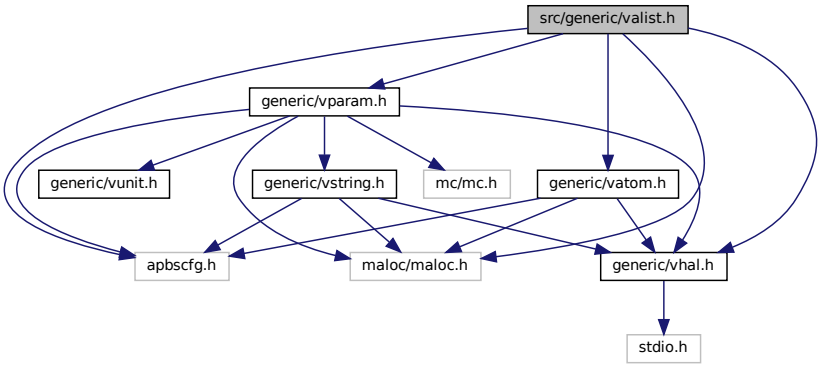
```
00907
00908     thee->center[0] = 0.5*(thee->maxcrd[0] + thee->mincrd[0]);
00909     thee->center[1] = 0.5*(thee->maxcrd[1] + thee->mincrd[1]);
00910     thee->center[2] = 0.5*(thee->maxcrd[2] + thee->mincrd[2]);
00911
00912     Vnm_print(0, "Valist_getStatistics:  Max atom coordinate:  (%g, %g, %g)\n",
00913               thee->maxcrd[0], thee->maxcrd[1], thee->maxcrd[2]);
00914     Vnm_print(0, "Valist_getStatistics:  Min atom coordinate:  (%g, %g, %g)\n",
00915               thee->mincrd[0], thee->mincrd[1], thee->mincrd[2]);
00916     Vnm_print(0, "Valist_getStatistics:  Molecule center:  (%g, %g, %g)\n",
00917               thee->center[0], thee->center[1], thee->center[2]);
00918
00919     return VRC_SUCCESS;
00920 }
```

9.53 src/generic/valist.h File Reference

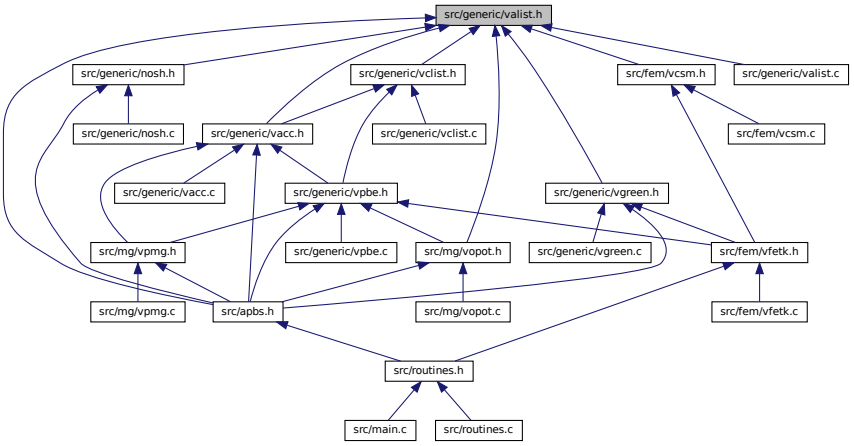
Contains declarations for class Valist.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/vatom.h"
#include "generic/vparam.h"
```

Include dependency graph for valist.h:



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



Data Structures

- struct sValist

  Container class for list of atom objects.

Typedefs

- typedef struct sValist Valist

  Declaration of the Valist class as the Valist structure.

Functions

- VEXTERNC Vatom * Valist_getAtomList (Valist *thee)

  Get actual array of atom objects from the list.
- VEXTERNC double Valist_getCenterX (Valist *thee)

  Get x-coordinate of molecule center.
- VEXTERNC double Valist_getCenterY (Valist *thee)

  Get y-coordinate of molecule center.
- VEXTERNC double Valist_getCenterZ (Valist *thee)

  Get z-coordinate of molecule center.
- VEXTERNC int Valist_getNumberAtoms (Valist *thee)

  Get number of atoms in the list.
- VEXTERNC Vatom * Valist_getAtom (Valist *thee, int i)

  Get pointer to particular atom in list.
- VEXTERNC unsigned long int Valist_memChk (Valist *thee)

  Get total memory allocated for this object and its members.
- VEXTERNC Valist * Valist_ctor ()

  Construct the atom list object.
- VEXTERNC Vrc_Codes Valist_ctor2 (Valist *thee)

  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.

9.53.1 Detailed Description

Contains declarations for class Valist.

Version

Id

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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.

9.54 valist.h

```
00001
00062 #ifndef _VALIST_H_
00063 #define _VALIST_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "maloc/maloc.h"
00068
00069 #include "generic/vhal.h"
00070 #include "generic/vatom.h"
00071 #include "generic/vparam.h"
00072
00078 struct sValist {
00079
00080   int number;
00081   double center[3];
00082   double mincrd[3];
00083   double maxcrd[3];
00084   double maxrad;
00085   double charge;
00086   Vatom *atoms;
00087   Vmem *vmem;
00089 };
00090
00095 typedef struct sValist Valist;
00096
00097 #if !defined(VINLINE_VATOM)
00098
00105 VEXTERNC Vatom* Valist_getAtomList(
00106         Valist *thee
00107         );
00108
00114 VEXTERNC double Valist_getCenterX(
00115         Valist *thee
00116         );
00117
00123 VEXTERNC double Valist_getCenterY(
00124         Valist *thee
00125         );
00126
00132 VEXTERNC double Valist_getCenterZ(
00133         Valist *thee
00134         );
00135
00141 VEXTERNC int Valist_getNumberAtoms(
00142         Valist *thee
00143         );
00144
00150 VEXTERNC Vatom* Valist_getAtom(
00151         Valist *thee,
00152         int i
00153         );
00154
00160 VEXTERNC unsigned long int Valist_memChk(
00161         Valist *thee
00162         );
00163
00164 #else /* if defined(VINLINE_VATOM) */
00165 #   define Valist_getAtomList(thee) ((thee)->atoms)
```

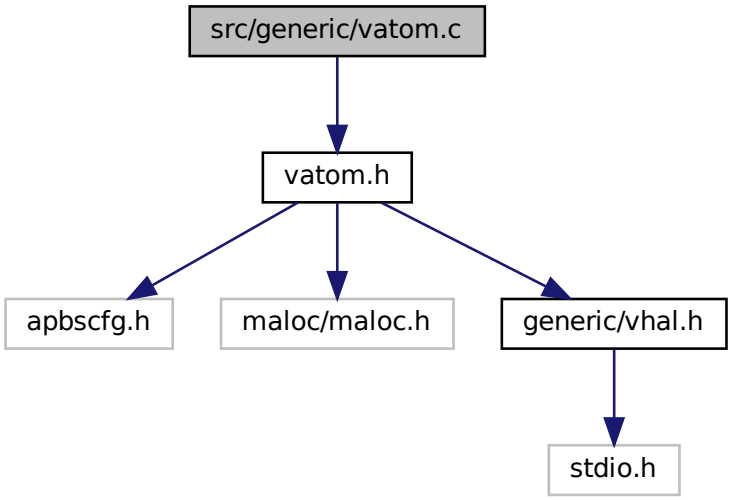
```
00166 #   define Valist_getNumberAtoms(thee) ((thee)->number)
00167 #   define Valist_getAtom(thee, i) (&((thee)->atoms[i]))
00168 #   define Valist_memChk(thee) (Vmem_bytes((thee)->vmem))
00169 #   define Valist_getCenterX(thee) ((thee)->center[0])
00170 #   define Valist_getCenterY(thee) ((thee)->center[1])
00171 #   define Valist_getCenterZ(thee) ((thee)->center[2])
00172 #endif /* if !defined(VINLINE_VATOM) */
00173
00179 VEXTERNC Valist* Valist_ctor();
00180
00186 VEXTERNC Vrc_Codes Valist_ctor2(
00187         Valist *thee
00188         );
00189
00194 VEXTERNC void Valist_dtor(
00195         Valist **thee
00196         );
00197
00202 VEXTERNC void Valist_dtor2(
00203         Valist *thee
00204         );
00205
00217 VEXTERNC Vrc_Codes Valist_readPQR(
00218         Valist *thee,
00219         Vparam *param,
00220         Vio *sock
00221         );
00222
00232 VEXTERNC Vrc_Codes Valist_readPDB(
00233         Valist *thee,
00234         Vparam *param,
00235         Vio *sock
00236         );
00237
00247 VEXTERNC Vrc_Codes Valist_readXML(
00248         Valist *thee,
00249         Vparam *param,
00250         Vio *sock
00251         );
00252
00259 VEXTERNC Vrc_Codes Valist_getStatistics(Valist *thee);
00260
00261
00262 #endif /* ifndef _VALIST_H_ */
```

9.55 src/generic/vatom.c File Reference

Class Vatom methods.

```
#include "vatom.h"
```

Include dependency graph for vatom.c:



Functions

- VPUBLIC double * Vatom_getPosition (Vatom *thee)

  Get atomic position.
- VPUBLIC double Vatom_getPartID (Vatom *thee)

  Get partition ID.
- VPUBLIC void Vatom_setPartID (Vatom *thee, int partID)

  Set partition ID.
- VPUBLIC double Vatom_getAtomID (Vatom *thee)

  Get atom ID.
- VPUBLIC void Vatom_setAtomID (Vatom *thee, int atomID)

  Set atom ID.
- VPUBLIC void Vatom_setRadius (Vatom *thee, double radius)

  Set atomic radius.
- VPUBLIC double Vatom_getRadius (Vatom *thee)

  Get atomic position.
- VPUBLIC void Vatom_setCharge (Vatom *thee, double charge)

  Set atomic charge.
- VPUBLIC double Vatom_getCharge (Vatom *thee)

  Get atomic charge.
- VPUBLIC void Vatom_setEpsilon (Vatom *thee, double epsilon)

  Set atomic epsilon.
- VPUBLIC double Vatom_getEpsilon (Vatom *thee)

  Get atomic epsilon.

- VPUBLIC unsigned long int Vatom_memChk (Vatom *thee)

  Return the memory used by this structure (and its contents) in bytes.
- VPUBLIC Vatom * Vatom_ctor ()

  Constructor for the Vatom class.
- VPUBLIC int Vatom_ctor2 (Vatom *thee)

  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.

9.55.1 Detailed Description

Class Vatom methods.

Author

Nathan Baker

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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) 2002-2020, 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.

9.56 vatom.c

```
00001
00057 #include "vatom.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_VATOM)
00062
00063 VPUBLIC double *Vatom_getPosition(Vatom *thee) {
00064
00065    VASSERT(thee != VNULL);
00066    return thee->position;
00067
00068 }
00069
00070 VPUBLIC double Vatom_getPartID(Vatom *thee) {
00071
00072    VASSERT(thee != VNULL);
00073    return thee->partID;
00074
00075 }
00076
00077 VPUBLIC void Vatom_setPartID(Vatom *thee, int partID) {
00078
00079    VASSERT(thee != VNULL);
00080    thee->partID = (double)partID;
00081
00082 }
00083
00084 VPUBLIC double Vatom_getAtomID(Vatom *thee) {
00085
00086    VASSERT(thee != VNULL);
00087    return thee->id;
00088
00089 }
00090
00091 VPUBLIC void Vatom_setAtomID(Vatom *thee, int atomID) {
00092
```

```
00093     VASSERT(thee != VNULL);
00094     thee->id = atomID;
00095
00096 }
00097
00098 VPUBLIC void Vatom_setRadius(Vatom *thee, double radius) {
00099
00100     VASSERT(thee != VNULL);
00101     thee->radius = radius;
00102
00103 }
00104
00105 VPUBLIC double Vatom_getRadius(Vatom *thee) {
00106
00107     VASSERT(thee != VNULL);
00108     return thee->radius;
00109
00110 }
00111
00112 VPUBLIC void Vatom_setCharge(Vatom *thee, double charge) {
00113
00114     VASSERT(thee != VNULL);
00115     thee->charge = charge;
00116
00117 }
00118
00119 VPUBLIC double Vatom_getCharge(Vatom *thee) {
00120
00121     VASSERT(thee != VNULL);
00122     return thee->charge;
00123
00124 }
00125
00126 VPUBLIC void Vatom_setEpsilon(Vatom *thee, double epsilon) {
00127
00128     VASSERT(thee != VNULL);
00129     thee->epsilon = epsilon;
00130 }
00131
00132 VPUBLIC double Vatom_getEpsilon(Vatom *thee) {
00133
00134     VASSERT(thee != VNULL);
00135     return thee->epsilon;
00136 }
00137
00138 VPUBLIC unsigned long int Vatom_memChk(Vatom *thee) { return sizeof(Vatom); }
00139
00140 #endif /* if !defined(VINLINE_VATOM) */
00141
00142 VPUBLIC Vatom* Vatom_ctor() {
00143
00144     /* Set up the structure */
00145     Vatom *thee = VNULL;
00146     thee = (Vatom *)Vmem_malloc( VNULL, 1, sizeof(Vatom) );
00147     VASSERT( thee != VNULL);
00148     VASSERT( Vatom_ctor2(thee));
00149
00150     return thee;
00151 }
00152
00153 VPUBLIC int Vatom_ctor2(Vatom *thee) {
00154     thee->partID = -1;
00155     return 1;
00156 }
00157
00158 VPUBLIC void Vatom_dtor(Vatom **thee) {
00159     if ((*thee) != VNULL) {
00160         Vatom_dtor2(*thee);
00161         Vmem_free(VNULL, 1, sizeof(Vatom), (void **)thee);
00162         (*thee) = VNULL;
00163     }
00164 }
00165
00166 VPUBLIC void Vatom_dtor2(Vatom *thee) { ; }
00167
00168 VPUBLIC void Vatom_setPosition(Vatom *thee, double position[3]) {
00169
00170     VASSERT(thee != VNULL);
00171     (thee->position)[0] = position[0];
00172     (thee->position)[1] = position[1];
00173     (thee->position)[2] = position[2];
```

```
00174
00175 }
00176
00177 VPUBLIC void Vatom_copyTo(Vatom *thee, Vatom *dest) {
00178
00179     VASSERT(thee != VNULL);
00180     VASSERT(dest != VNULL);
00181
00182     memcpy(dest, thee, sizeof(Vatom));
00183
00184 }
00185
00186 VPUBLIC void Vatom_copyFrom(Vatom *thee, Vatom *src) {
00187
00188     Vatom_copyTo(src, thee);
00189
00190 }
00191
00192 VPUBLIC void Vatom_setResName(Vatom *thee, char resName[VMAX_RECLEN]) {
00193
00194     VASSERT(thee != VNULL);
00195     strcpy(thee->resName, resName);
00196
00197 }
00198
00199 VPUBLIC void Vatom_getResName(Vatom *thee, char resName[VMAX_RECLEN]) {
00200
00201
00202     VASSERT(thee != VNULL);
00203     strcpy(resName,thee->resName);
00204
00205 }
00206
00207 VPUBLIC void Vatom_setAtomName(Vatom *thee, char atomName[VMAX_RECLEN]) {
00208
00209     VASSERT(thee != VNULL);
00210     strcpy(thee->atomName, atomName);
00211
00212 }
00213
00214 VPUBLIC void Vatom_getAtomName(Vatom *thee, char atomName[VMAX_RECLEN]) {
00215
00216     VASSERT(thee != VNULL);
00217     strcpy(atomName,thee->atomName);
00218
00219 }
00220
00221 #if defined(WITH_TINKER)
00222
00223 VPUBLIC void Vatom_setDipole(Vatom *thee, double dipole[3]) {
00224
00225    VASSERT(thee != VNULL);
00226    (thee->dipole)[0] = dipole[0];
00227    (thee->dipole)[1] = dipole[1];
00228    (thee->dipole)[2] = dipole[2];
00229
00230 }
00231
00232 VPUBLIC void Vatom_setQuadrupole(Vatom *thee, double quadrupole[9]) {
00233
00234    int i;
00235    VASSERT(thee != VNULL);
00236    for (i=0; i<9; i++)  (thee->quadrupole)[i] = quadrupole[i];
00237 }
00238
00239 VPUBLIC void Vatom_setInducedDipole(Vatom *thee, double dipole[3]) {
00240
00241    VASSERT(thee != VNULL);
00242    (thee->inducedDipole)[0] = dipole[0];
00243    (thee->inducedDipole)[1] = dipole[1];
00244    (thee->inducedDipole)[2] = dipole[2];
00245 }
00246
00247 VPUBLIC void Vatom_setNLInducedDipole(Vatom *thee, double dipole[3]) {
00248
00249    VASSERT(thee != VNULL);
00250    (thee->nlInducedDipole)[0] = dipole[0];
00251    (thee->nlInducedDipole)[1] = dipole[1];
00252    (thee->nlInducedDipole)[2] = dipole[2];
00253
00254 }
```

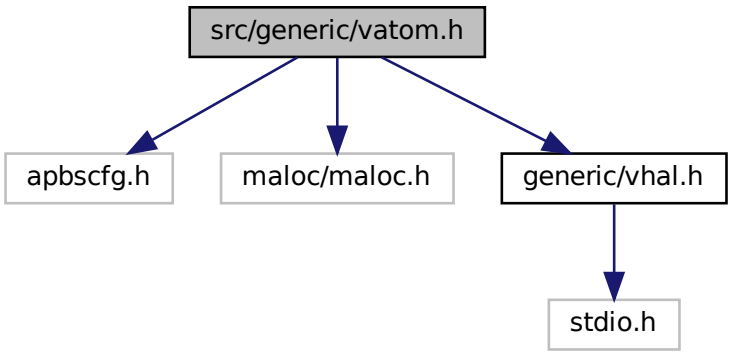
```
00255
00256 VPUBLIC double *Vatom_getDipole(Vatom *thee) {
00257
00258    VASSERT(thee != VNULL);
00259    return thee->dipole;
00260
00261 }
00262
00263 VPUBLIC double *Vatom_getQuadrupole(Vatom *thee) {
00264
00265    VASSERT(thee != VNULL);
00266    return thee->quadrupole;
00267
00268 }
00269
00270 VPUBLIC double *Vatom_getInducedDipole(Vatom *thee) {
00271
00272    VASSERT(thee != VNULL);
00273    return thee->inducedDipole;
00274
00275 }
00276
00277 VPUBLIC double *Vatom_getNLInducedDipole(Vatom *thee) {
00278
00279    VASSERT(thee != VNULL);
00280    return thee->nlInducedDipole;
00281
00282 }
00283
00284 #endif /* if defined(WITH_TINKER) */
```

9.57 src/generic/vatom.h File Reference

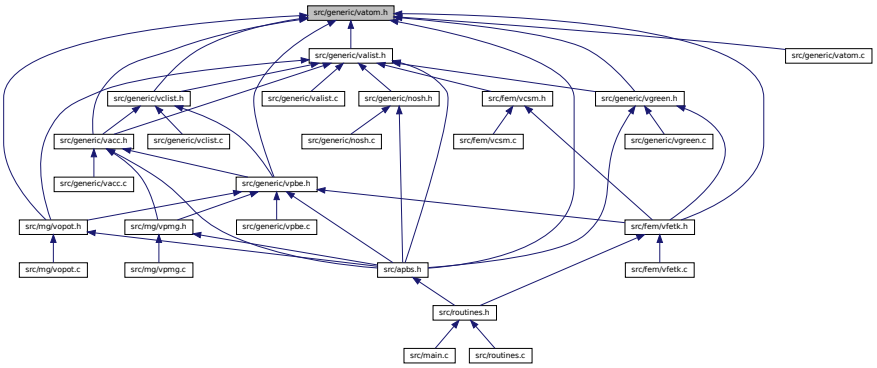
Contains declarations for class Vatom.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/vhal.h"
```

Include dependency graph for vatom.h:



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



Data Structures

- struct sVatom

  Contains public data members for Vatom class/module.

Macros

- #define VMAX_RECLEN 64

  Residue name length.

Typedefs

- typedef struct sVatom Vatom

  Declaration of the Vatom class as the Vatom structure.

Functions

- VEXTERNC double * Vatom_getPosition (Vatom *thee)

  Get atomic position.
- VEXTERNC void Vatom_setRadius (Vatom *thee, double radius)

  Set atomic radius.
- VEXTERNC double Vatom_getRadius (Vatom *thee)

  Get atomic position.
- VEXTERNC void Vatom_setPartID (Vatom *thee, int partID)

  Set partition ID.
- VEXTERNC double Vatom_getPartID (Vatom *thee)

  Get partition ID.
- VEXTERNC void Vatom_setAtomID (Vatom *thee, int id)

  Set atom ID.
- VEXTERNC double Vatom_getAtomID (Vatom *thee)

  Get atom ID.
- VEXTERNC void Vatom_setCharge (Vatom *thee, double charge)

  Set atomic charge.
- VEXTERNC double Vatom_getCharge (Vatom *thee)

Get atomic charge.

- VEXTERNC void Vatom_setEpsilon (Vatom *thee, double epsilon)

  Set atomic epsilon.
- VEXTERNC double Vatom_getEpsilon (Vatom *thee)

  Get atomic epsilon.
- VEXTERNC unsigned long int Vatom_memChk (Vatom *thee)

  Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC void Vatom_setResName (Vatom *thee, char resName[VMAX_RECLEN])

  Set residue name.
- VEXTERNC void Vatom_setAtomName (Vatom *thee, char atomName[VMAX_RECLEN])

  Set atom name.
- VEXTERNC void Vatom_getResName (Vatom *thee, char resName[VMAX_RECLEN])

  Retrieve residue name.
- VEXTERNC void Vatom_getAtomName (Vatom *thee, char atomName[VMAX_RECLEN])

  Retrieve atom name.
- VEXTERNC Vatom * Vatom_ctor ()

  Constructor for the Vatom class.
- VEXTERNC int Vatom_ctor2 (Vatom *thee)

  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.

9.57.1 Detailed Description

Contains declarations for class Vatom.

Version

Id

Author

Nathan A. Baker

Attention

Definition in file vatom.h.

9.58 vatom.h

```
00001
00062 #ifndef _VATOM_H_
00063 #define _VATOM_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "maloc/maloc.h"
00068
00069 #include "generic/vhal.h"
00070
00077 #define VMAX_RECLEN     64
00078
00084 struct sVatom {
00085
00086     double position[3];
00087     double radius;
00088     double charge;
```

```
00089     double partID;
00091     double epsilon;
00093     int id;
00097     char resName[VMAX_RECLEN];
00098     char atomName[VMAX_RECLEN];
00100 #if defined(WITH_TINKER)
00101
00102     double dipole[3];
00103     double quadrupole[9];
00104     double inducedDipole[3];
00105     double nlInducedDipole[3];
00107 #endif /* if defined(WITH_TINKER) */
00108 };
00109
00114 typedef struct sVatom Vatom;
00115
00116 #if !defined(VINLINE_VATOM)
00117
00124     VEXTERNC double* Vatom_getPosition(Vatom *thee);
00125
00132     VEXTERNC void    Vatom_setRadius(Vatom *thee, double radius);
00133
00140     VEXTERNC double  Vatom_getRadius(Vatom *thee);
00141
00149     VEXTERNC void    Vatom_setPartID(Vatom *thee, int partID);
00150
00158     VEXTERNC double     Vatom_getPartID(Vatom *thee);
00159
00166     VEXTERNC void Vatom_setAtomID(Vatom *thee, int id);
00167
00174     VEXTERNC double Vatom_getAtomID(Vatom *thee);
00175
00182     VEXTERNC void    Vatom_setCharge(Vatom *thee, double charge);
00183
00190     VEXTERNC double  Vatom_getCharge(Vatom *thee);
00191
00198     VEXTERNC void    Vatom_setEpsilon(Vatom *thee, double epsilon);
00199
00206     VEXTERNC double  Vatom_getEpsilon(Vatom *thee);
00207
00215     VEXTERNC unsigned long int Vatom_memChk(Vatom *thee);
00216
00217 #else /* if defined(VINLINE_VATOM) */
00218 #   define Vatom_getPosition(thee) ((thee)->position)
00219 #   define Vatom_setRadius(thee, tRadius) ((thee)->radius = (tRadius))
00220 #   define Vatom_getRadius(thee) ((thee)->radius)
00221 #   define Vatom_setPartID(thee, tpartID) ((thee)->partID = (double)(tpartID))
00222 #   define Vatom_getPartID(thee) ((thee)->partID)
00223 #   define Vatom_setAtomID(thee, tatomID) ((thee)->id = (tatomID))
00224 #   define Vatom_getAtomID(thee) ((thee)->id)
00225 #   define Vatom_setCharge(thee, tCharge) ((thee)->charge = (tCharge))
00226 #   define Vatom_getCharge(thee) ((thee)->charge)
00227 #   define Vatom_setEpsilon(thee, tEpsilon) ((thee)->epsilon = (tEpsilon))
00228 #   define Vatom_getEpsilon(thee) ((thee)->epsilon)
00229 #   define Vatom_memChk(thee) (sizeof(Vatom))
00230 #endif /* if !defined(VINLINE_VATOM) */
00231
00232 /* //////////////////////////////////////////////////////////////////////////
00233 // Class Vatom: Non-Inlineable methods (vatom.c)
00235
00242 VEXTERNC void    Vatom_setResName(Vatom *thee, char resName[VMAX_RECLEN]);
00243
00248 VEXTERNC void    Vatom_setAtomName(
00249         Vatom *thee,  /**< Vatom object */
00250         char atomName[VMAX_RECLEN]
00251         );
00252
00259 VEXTERNC void    Vatom_getResName(Vatom *thee, char resName[VMAX_RECLEN]);
00260
00265 VEXTERNC void   Vatom_getAtomName(
00266         Vatom *thee,
00267         char atomName[VMAX_RECLEN]
00268         );
00269
00275 VEXTERNC Vatom* Vatom_ctor();
00276
00283 VEXTERNC int     Vatom_ctor2(Vatom *thee);
00284
00290 VEXTERNC void    Vatom_dtor(Vatom **thee);
00291
00297 VEXTERNC void    Vatom_dtor2(Vatom *thee);
```

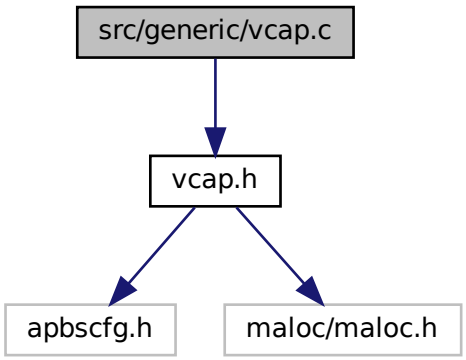
```
00298
00305 VEXTERNC void   Vatom_setPosition(Vatom *thee, double position[3]);
00306
00314 VEXTERNC void Vatom_copyTo(Vatom *thee, Vatom *dest);
00315
00323 VEXTERNC void Vatom_copyFrom(Vatom *thee, Vatom *src);
00324
00325 #if defined(WITH_TINKER)
00326
00333 VEXTERNC void   Vatom_setInducedDipole(Vatom *thee,
00334                                        double inducedDipole[3]);
00335
00342 VEXTERNC void   Vatom_setNLInducedDipole(Vatom *thee,
00343                                        double nlInducedDipole[3]);
00344
00351 VEXTERNC void   Vatom_setDipole(Vatom *thee, double dipole[3]);
00352
00359 VEXTERNC void   Vatom_setQuadrupole(Vatom *thee, double quadrupole[9]);
00360
00366 VEXTERNC double*  Vatom_getDipole(Vatom *thee);
00367
00373 VEXTERNC double*  Vatom_getQuadrupole(Vatom *thee);
00374
00380 VEXTERNC double*  Vatom_getInducedDipole(Vatom *thee);
00381
00387 VEXTERNC double*  Vatom_getNLInducedDipole(Vatom *thee);
00388 #endif /* if defined(WITH_TINKER) */
00389
00390 #endif /* ifndef _VATOM_H_ */
```

9.59 src/generic/vcap.c File Reference

Class Vcap methods.

`#include "vcap.h"`

Include dependency graph for vcap.c:



Functions

- VPUBLIC double Vcap_exp (double x, int *ichop)

  Provide a capped exp() function.

- VPUBLIC double Vcap_sinh (double x, int *ichop)

  Provide a capped sinh() function.

- VPUBLIC double Vcap_cosh (double x, int *ichop)

  Provide a capped cosh() function.

9.59.1 Detailed Description

Class Vcap methods.

Author

Nathan Baker

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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) 2002-2020, 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.

9.60 vcap.c

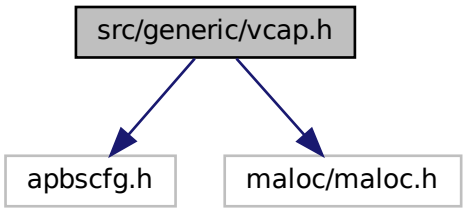
```
00001
00057 #include "vcap.h"
00058
00059 VPUBLIC double Vcap_exp(double x, int *ichop) {
00060
00061     /* The two chopped arguments */
00062     if (x > EXPMAX) {
00063        (*ichop) = 1;
00064        return VEXP(EXPMAX);
00065     } else if (x < EXPMIN) {
00066        (*ichop) = 1;
00067        return VEXP(EXPMIN);
00068     }
00069
00070     /* The normal EXP */
00071     (*ichop) = 0;
00072     return VEXP(x);
00073 }
00074
00075 VPUBLIC double Vcap_sinh(double x, int *ichop) {
00076
00077     /* The two chopped arguments */
00078     if (x > EXPMAX) {
00079        (*ichop) = 1;
00080        return VSINH(EXPMAX);
00081     } else if (x < EXPMIN) {
00082        (*ichop) = 1;
00083        return VSINH(EXPMIN);
00084     }
00085
00086     /* The normal SINH */
00087     (*ichop) = 0;
00088     return VSINH(x);
00089 }
00090
00091 VPUBLIC double Vcap_cosh(double x, int *ichop) {
00092
00093     /* The two chopped arguments */
00094     if (x > EXPMAX) {
00095        (*ichop) = 1;
00096        return VCOSH(EXPMAX);
00097     } else if (x < EXPMIN) {
00098        (*ichop) = 1;
00099        return VCOSH(EXPMIN);
00100     }
00101
00102     /* The normal COSH */
00103     (*ichop) = 0;
00104     return VCOSH(x);
00105 }
```

9.61 src/generic/vcap.h File Reference

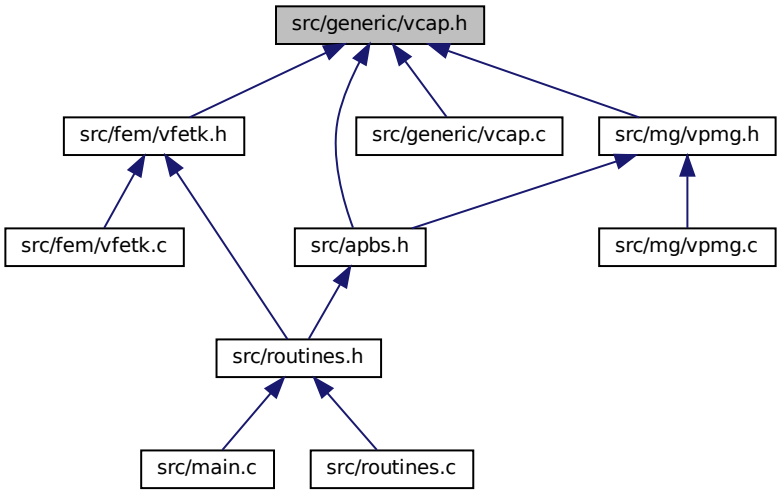
Contains declarations for class Vcap.

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

Include dependency graph for vcap.h:



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



Macros

- #define EXPMAX 85.00

  Maximum argument for exp(), sinh(), or cosh()
- #define EXPMIN -85.00

  Minimum argument for exp(), sinh(), or cosh()

Functions

- VEXTERNC double Vcap_exp (double x, int *ichop)

  Provide a capped exp() function.
- VEXTERNC double Vcap_sinh (double x, int *ichop)

Provide a capped sinh() function.

- VEXTERNC double Vcap_cosh (double x, int *ichop)

  Provide a capped cosh() function.

9.61.1 Detailed Description

Contains declarations for class Vcap.

Version

Id

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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.

9.62 vcap.h

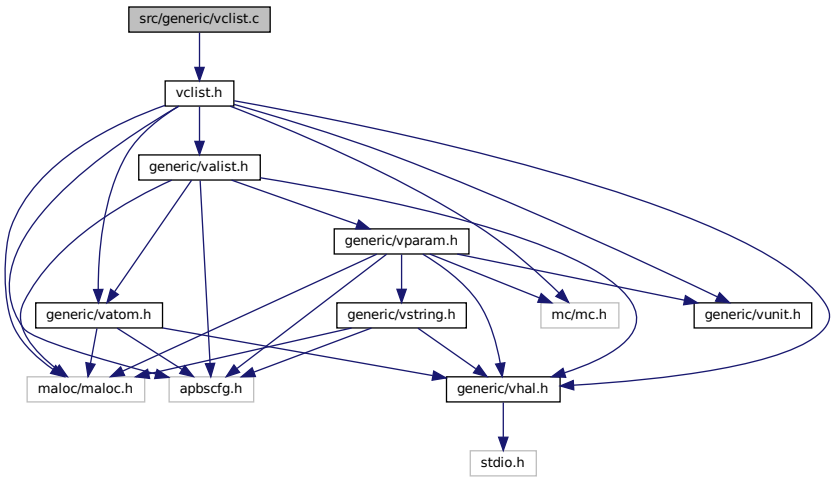
```
00001
00064 #ifndef _VCAP_H_
00065 #define _VCAP_H_
00066
00067 #include "apbscfg.h"
00068
00072 #define EXPMAX  85.00
00073
00077 #define EXPMIN -85.00
00078
00079 #include "maloc/maloc.h"
00080
00099 VEXTERNC double Vcap_exp(
00100         double x,
00101         int *ichop
00102         );
00103
00104
00123 VEXTERNC double Vcap_sinh(
00124         double x,
00125         int *ichop
00126         );
00127
00146 VEXTERNC double Vcap_cosh(
00147         double x,
00148         int *ichop
00149         );
00150
00151 #endif    /* ifndef _VCAP_H_ */
```

9.63 src/generic/vclist.c File Reference

Class Vclist methods.

`#include "vclist.h"`

Include dependency graph for vclist.c:



Macros

- #define **VCLIST_INFLATE** 1.42

Functions

- VPUBLIC unsigned long int Vclist_memChk (Vclist *thee)

  Get number of bytes in this object and its members.
- VPUBLIC double Vclist_maxRadius (Vclist *thee)

  Get the max probe radius value (in 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.

9.63.1 Detailed Description

Class Vclist methods.

Author

Nathan Baker

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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) 2002-2020, 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.

9.64 vclist.c

```
00001
00057 #include "vclist.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 #if !defined(VINLINE_VCLIST)
00062
00063 VPUBLIC unsigned long int Vclist_memChk(Vclist *thee) {
00064     if (thee == VNULL) return 0;
00065     return Vmem_bytes(thee->vmem);
00066 }
00067
00068 VPUBLIC double Vclist_maxRadius(Vclist *thee) {
00069     VASSERT(thee != VNULL);
00070     return thee->max_radius;
00071 }
00072
```

```
00073 #endif /* if !defined(VINLINE_VCLIST) */
00074
00075 VPUBLIC Vclist* Vclist_ctor(Valist *alist, double max_radius,
00076         int npts[VAPBS_DIM], Vclist_DomainMode mode,
00077         double lower_corner[VAPBS_DIM],  double upper_corner[VAPBS_DIM]) {
00078
00079     Vclist *thee = VNULL;
00080
00081     /* Set up the structure */
00082     thee = (Vclist*)Vmem_malloc(VNULL, 1, sizeof(Vclist) );
00083     VASSERT( thee != VNULL);
00084     VASSERT( Vclist_ctor2(thee, alist, max_radius, npts, mode, lower_corner,
00085                 upper_corner) == VRC_SUCCESS );
00086     return thee;
00087 }
00088
00089 /* Get the dimensions of the molecule stored in thee->alist */
00090 VPRIVATE void Vclist_getMolDims(
00091         Vclist *thee,
00092         double lower_corner[VAPBS_DIM], /* Set to lower corner of molecule */
00093         double upper_corner[VAPBS_DIM], /* Set to lower corner of molecule */
00094         double *r_max /* Set to max atom radius */
00095         ) {
00096
00097     int i, j;
00098     double pos;
00099     Valist *alist;
00100     Vatom *atom;
00101
00102     alist = thee->alist;
00103
00104     /* Initialize */
00105     for (i=0; i<VAPBS_DIM; i++) {
00106         lower_corner[i] = VLARGE;
00107         upper_corner[i] = -VLARGE;
00108     }
00109     *r_max = -1.0;
00110
00111     /* Check each atom */
00112     for (i=0; i<Valist_getNumberAtoms(alist); i++) {
00113         atom = Valist_getAtom(alist, i);
00114         for (j=0; j<VAPBS_DIM; j++) {
00115             pos = (Vatom_getPosition(atom))[j];
00116             if ( pos < lower_corner[j] ) lower_corner[j] = pos;
00117             if ( pos > upper_corner[j] ) upper_corner[j] = pos;
00118         }
00119         if (Vatom_getRadius(atom) > *r_max) *r_max = Vatom_getRadius(atom);
00120     }
00121
00122 }
00123
00124 /* Setup lookup grid */
00125 VPRIVATE Vrc_Codes Vclist_setupGrid(Vclist *thee) {
00126
00127     /* Inflation factor ~ sqrt(2)*/
00128     #define VCLIST_INFLATE 1.42
00129
00130     int i;
00131     double length[VAPBS_DIM], r_max;
00132
00133     /* Set up the grid corners */
00134     switch (thee->mode) {
00135         case CLIST_AUTO_DOMAIN:
00136             /* Get molecule dimensions */
00137             Vclist_getMolDims(thee, thee->lower_corner, thee->upper_corner,
00138                     &r_max);
00139             /* Set up grid spacings */
00140             for (i=0; i<VAPBS_DIM; i++) {
00141                 thee->upper_corner[i] = thee->upper_corner[i]
00142                     + VCLIST_INFLATE*(r_max+thee->max_radius);
00143                 thee->lower_corner[i] = thee->lower_corner[i]
00144                     - VCLIST_INFLATE*(r_max+thee->max_radius);
00145             }
00146             break;
00147         case CLIST_MANUAL_DOMAIN:
00148             /* Grid corners established in constructor */
00149             break;
00150         default:
00151             Vnm_print(2, "Vclist_setupGrid:  invalid setup mode (%d)!\n",
00152                     thee->mode);
00153             return VRC_FAILURE;
```

```
00154     }
00155
00156     /* Set up the grid lengths and spacings */
00157     for (i=0; i<VAPBS_DIM; i++) {
00158         length[i] = thee->upper_corner[i] - thee->lower_corner[i];
00159         thee->spacs[i] = length[i]/((double)(thee->npts[i] - 1));
00160     }
00161     Vnm_print(0, "Vclist_setupGrid:  Grid lengths = (%g, %g, %g)\n",
00162             length[0], length[1], length[2]);
00163
00164     Vnm_print(0, "Vclist_setupGrid:  Grid lower corner = (%g, %g, %g)\n",
00165       (thee->lower_corner)[0], (thee->lower_corner)[1],
00166       (thee->lower_corner)[2]);
00167
00168     return VRC_SUCCESS;
00169
00170     #undef VCLIST_INFLATE
00171 }
00172
00173 /* Check and store parameters passed to constructor */
00174 VPRIVATE Vrc_Codes Vclist_storeParms(Vclist *thee, Valist *alist,
00175         double max_radius, int npts[VAPBS_DIM], Vclist_DomainMode mode,
00176         double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM] ) {
00177
00178     int i = 0;
00179
00180     if (alist == VNULL) {
00181         Vnm_print(2, "Vclist_ctor2:  Got NULL Valist!\n");
00182         return VRC_FAILURE;
00183     } else thee->alist = alist;
00184
00185     thee->n = 1;
00186     for (i=0; i<VAPBS_DIM; i++) {
00187         if (npts[i] < 3) {
00188             Vnm_print(2,
00189                     "Vclist_ctor2:  n[%d] (%d) must be greater than 2!\n",
00190                     i, npts[i]);
00191             return VRC_FAILURE;
00192         }
00193         thee->npts[i] = npts[i];
00194         thee->n *= npts[i];
00195     }
00196     Vnm_print(0, "Vclist_ctor2:  Using %d x %d x %d hash table\n",
00197             npts[0], npts[1], npts[2]);
00198
00199     thee->mode = mode;
00200     switch (thee->mode) {
00201         case CLIST_AUTO_DOMAIN:
00202             Vnm_print(0, "Vclist_ctor2:  automatic domain setup.\n");
00203             break;
00204         case CLIST_MANUAL_DOMAIN:
00205             Vnm_print(0, "Vclist_ctor2:  manual domain setup.\n");
00206             Vnm_print(0, "Vclist_ctor2:  lower corner = [ \n");
00207             for (i=0; i<VAPBS_DIM; i++) {
00208                 thee->lower_corner[i] = lower_corner[i];
00209                 Vnm_print(0, "%g ", lower_corner[i]);
00210             }
00211             Vnm_print(0, "]\n");
00212             Vnm_print(0, "Vclist_ctor2:  upper corner = [ \n");
00213             for (i=0; i<VAPBS_DIM; i++) {
00214                 thee->upper_corner[i] = upper_corner[i];
00215                 Vnm_print(0, "%g ", upper_corner[i]);
00216             }
00217             Vnm_print(0, "]\n");
00218             break;
00219         default:
00220             Vnm_print(2, "Vclist_ctor2:  invalid setup mode (%d)!\n", mode);
00221             return VRC_FAILURE;
00222     }
00223
00224     thee->max_radius = max_radius;
00225     Vnm_print(0, "Vclist_ctor2:  Using %g max radius\n", max_radius);
00226
00227     return VRC_SUCCESS;
00228 }
00229
00230 /* Calculate the gridpoints an atom spans */
00231 VPRIVATE void Vclist_gridSpan(Vclist *thee,
00232         Vatom *atom, /* Atom */
00233         int imin[VAPBS_DIM], /* Set to min grid indices */
00234         int imax[VAPBS_DIM]  /* Set to max grid indices */
```

```
00235         ) {
00236
00237     int i;
00238     double *coord, dc, idc, rtot;
00239
00240     /* Get the position in the grid's frame of reference */
00241     coord = Vatom_getPosition(atom);
00242
00243     /* Get the range the atom radius + probe radius spans */
00244     rtot = Vatom_getRadius(atom) + thee->max_radius;
00245
00246     /* Calculate the range of grid points the inflated atom spans in the x
00247      * direction. */
00248     for (i=0; i<VAPBS_DIM; i++) {
00249         dc = coord[i] - (thee->lower_corner)[i];
00250         idc = (dc + rtot)/(thee->spacs[i]);
00251         imax[i] = (int)(ceil(idc));
00252         imax[i] = VMIN2(imax[i], thee->npts[i]-1);
00253         idc = (dc - rtot)/(thee->spacs[i]);
00254         imin[i] = (int)(floor(idc));
00255         imin[i] = VMAX2(imin[i], 0);
00256     }
00257
00258 }
00259
00260 /* Get the array index for a particular cell based on its i,j,k
00261  * coordinates */
00262 VPRIVATE int Vclist_arrayIndex(Vclist *thee, int i, int j, int k) {
00263
00264     return (thee->npts[2])*(thee->npts[1])*i + (thee->npts[2])*j + k;
00265
00266 }
00267
00268
00269 /* Assign atoms to cells */
00270 VPRIVATE Vrc_Codes Vclist_assignAtoms(Vclist *thee) {
00271
00272     int iatom, i, j, k, ui, inext;
00273     int imax[VAPBS_DIM], imin[VAPBS_DIM];
00274     int totatoms;
00275     Vatom *atom;
00276     VclistCell *cell;
00277
00278
00279     /* Find out how many atoms are associated with each grid point */
00280     totatoms = 0;
00281     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
00282
00283         /* Get grid span for atom */
00284         atom = Valist_getAtom(thee->alist, iatom);
00285         Vclist_gridSpan(thee, atom, imin, imax);
00286
00287         /* Now find and assign the grid points */
00288         VASSERT(VAPBS_DIM == 3);
00289         for ( i = imin[0]; i <= imax[0]; i++) {
00290             for ( j = imin[1]; j <= imax[1]; j++) {
00291                 for ( k = imin[2]; k <= imax[2]; k++) {
00292                     /* Get index to array */
00293                     ui = Vclist_arrayIndex(thee, i, j, k);
00294                     /* Increment number of atoms for this grid point */
00295                     cell = &(thee->cells[ui]);
00296                     (cell->natoms)++;
00297                     totatoms++;
00298                 }
00299             }
00300         }
00301     }
00302     Vnm_print(0, "Vclist_assignAtoms:  Have %d atom entries\n", totatoms);
00303
00304     /* Allocate the space to store the pointers to the atoms */
00305     for (ui=0; ui<thee->n; ui++) {
00306         cell = &(thee->cells[ui]);
00307         if ( VclistCell_ctor2(cell, cell->natoms) == VRC_FAILURE ) {
00308             Vnm_print(2, "Vclist_assignAtoms:  cell error!\n");
00309             return VRC_FAILURE;
00310         }
00311         /* Clear the counter for later use */
00312         cell->natoms = 0;
00313     }
00314
00315     /* Assign the atoms to grid points */
```

```
00316     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
00317
00318         /* Get grid span for atom */
00319         atom = Valist_getAtom(thee->alist, iatom);
00320         Vclist_gridSpan(thee, atom, imin, imax);
00321
00322         /* Now find and assign the grid points */
00323         for (i = imin[0]; i <= imax[0]; i++) {
00324             for (j = imin[1]; j <= imax[1]; j++) {
00325                 for (k = imin[2]; k <= imax[2]; k++) {
00326                     /* Get index to array */
00327                     ui = Vclist_arrayIndex(thee, i, j, k);
00328                     cell = &(thee->cells[ui]);
00329                     /* Index of next available array location */
00330                     inext = cell->natoms;
00331                     cell->atoms[inext] = atom;
00332                     /* Increment number of atoms */
00333                     (cell->natoms)++;
00334                 }
00335             }
00336         }
00337     }
00338
00339     return VRC_SUCCESS;
00340 }
00341
00342 /* Main (FORTRAN stub) constructor */
00343 VPUBLIC Vrc_Codes Vclist_ctor2(Vclist *thee, Valist *alist, double max_radius,
00344         int npts[VAPBS_DIM], Vclist_DomainMode mode,
00345         double lower_corner[VAPBS_DIM], double upper_corner[VAPBS_DIM]) {
00346
00347     int i;
00348     VclistCell *cell;
00349
00350     /* Check and store parameters */
00351     if ( Vclist_storeParms(thee, alist, max_radius, npts, mode, lower_corner,
00352                 upper_corner) == VRC_FAILURE ) {
00353         Vnm_print(2, "Vclist_ctor2:  parameter check failed!\n");
00354         return VRC_FAILURE;
00355     }
00356
00357     /* Set up memory */
00358     thee->vmem = Vmem_ctor("APBS::VCLIST");
00359     if (thee->vmem == VNULL) {
00360         Vnm_print(2, "Vclist_ctor2:  memory object setup failed!\n");
00361         return VRC_FAILURE;
00362     }
00363
00364     /* Set up cells */
00365     thee->cells = (VclistCell*)Vmem_malloc( thee->vmem, thee->n, sizeof(VclistCell) );
00366     if (thee->cells == VNULL) {
00367         Vnm_print(2,
00368                 "Vclist_ctor2:  Failed allocating %d VclistCell objects!\n",
00369                 thee->n);
00370         return VRC_FAILURE;
00371     }
00372     for (i=0; i<thee->n; i++) {
00373         cell = &(thee->cells[i]);
00374         cell->natoms = 0;
00375     }
00376
00377     /* Set up the grid */
00378     if ( Vclist_setupGrid(thee) == VRC_FAILURE ) {
00379         Vnm_print(2, "Vclist_ctor2:  grid setup failed!\n");
00380         return VRC_FAILURE;
00381     }
00382
00383     /* Assign atoms to grid cells */
00384     if (Vclist_assignAtoms(thee) == VRC_FAILURE) {
00385         Vnm_print(2, "Vclist_ctor2:  atom assignment failed!\n");
00386         return VRC_FAILURE;
00387     }
00388
00389
00390
00391
00392
00393     return VRC_SUCCESS;
00394 }
00395
00396 /* Destructor */
```

```
00397 VPUBLIC void Vclist_dtor(Vclist **thee) {
00398
00399     if ((*thee) != VNULL) {
00400         Vclist_dtor2(*thee);
00401         Vmem_free(VNULL, 1, sizeof(Vclist), (void **)thee);
00402         (*thee) = VNULL;
00403     }
00404
00405 }
00406
00407 /* Main (stub) destructor */
00408 VPUBLIC void Vclist_dtor2(Vclist *thee) {
00409
00410     VclistCell *cell;
00411     int i;
00412
00413     for (i=0; i<thee->n; i++) {
00414         cell = &(thee->cells[i]);
00415         VclistCell_dtor2(cell);
00416     }
00417     Vmem_free(thee->vmem, thee->n, sizeof(VclistCell),
00418               (void **)&(thee->cells));
00419     Vmem_dtor(&(thee->vmem));
00420
00421 }
00422
00423 VPUBLIC VclistCell* Vclist_getCell(Vclist *thee,
00424                                    double pos[VAPBS_DIM]
00425                                   ) {
00426
00427     int i,
00428         ic[VAPBS_DIM],
00429         ui;
00430     double c[VAPBS_DIM];
00431
00432     /* Assert this before we do anything else, since its failure should fail the function */
00433     VASSERT(VAPBS_DIM == 3);
00434
00435     /* Convert to grid based coordinates */
00436     for (i=0; i<VAPBS_DIM; i++) {
00437         c[i] = pos[i] - (thee->lower_corner)[i];
00438         ic[i] = (int)(c[i]/thee->spacs[i]);
00439
00440         if (ic[i] < 0 || ic[i] >= thee->npts[i]) {
00441             return VNULL;
00442         }
00443     }
00444
00445     /* Get the array index */
00446     ui = Vclist_arrayIndex(thee, ic[0], ic[1], ic[2]);
00447
00448     return &(thee->cells[ui]);
00449
00450 }
00451
00452 VPUBLIC VclistCell* VclistCell_ctor(int natoms) {
00453
00454     VclistCell *thee = VNULL;
00455
00456     /* Set up the structure */
00457     thee = (VclistCell*)Vmem_malloc(VNULL, 1, sizeof(VclistCell));
00458     VASSERT( thee != VNULL);
00459     VASSERT( VclistCell_ctor2(thee, natoms) == VRC_SUCCESS );
00460
00461     return thee;
00462 }
00463
00464 VPUBLIC Vrc_Codes VclistCell_ctor2(VclistCell *thee, int natoms) {
00465
00466     if (thee == VNULL) {
00467         Vnm_print(2, "VclistCell_ctor2:  NULL thee!\n");
00468         return VRC_FAILURE;
00469     }
00470
00471     thee->natoms = natoms;
00472     if (thee->natoms > 0) {
00473         thee->atoms = (Vatom**)Vmem_malloc(VNULL, natoms, sizeof(Vatom *));
00474         if (thee->atoms == VNULL) {
00475             Vnm_print(2,
00476           "VclistCell_ctor2:  unable to allocate space for %d atom pointers!\n",
00477               natoms);
```

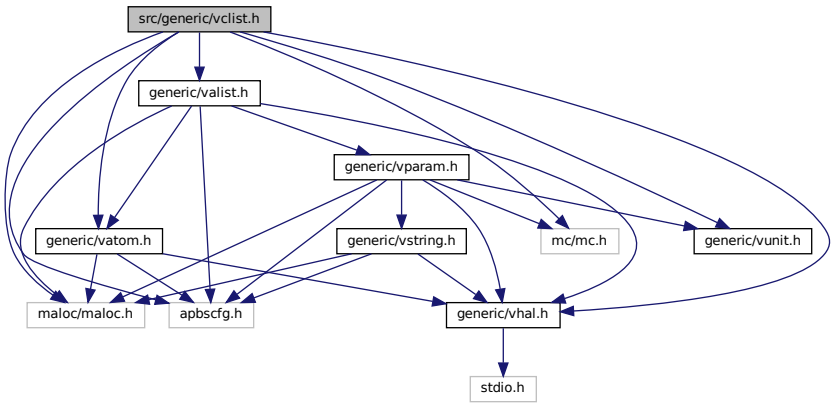
```
00478             return VRC_FAILURE;
00479         }
00480     }
00481
00482     return VRC_SUCCESS;
00483
00484 }
00485
00486 VPUBLIC void VclistCell_dtor(VclistCell **thee) {
00487
00488     if ((*thee) != VNULL) {
00489         VclistCell_dtor2(*thee);
00490         Vmem_free(VNULL, 1, sizeof(VclistCell), (void **)thee);
00491         (*thee) = VNULL;
00492     }
00493
00494 }
00495
00496 /* Main (stub) destructor */
00497 VPUBLIC void VclistCell_dtor2(VclistCell *thee) {
00498
00499     if (thee->natoms > 0) {
00500         Vmem_free(VNULL, thee->natoms, sizeof(Vatom *),
00501                 (void **)&(thee->atoms));
00502     }
00503
00504 }
```

9.65 src/generic/vclist.h File Reference

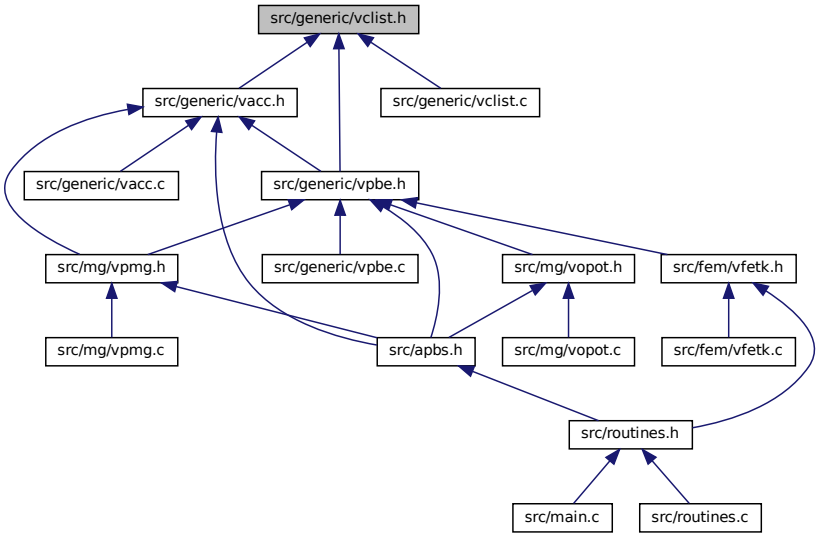
Contains declarations for class Vclist.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "mc/mc.h"
#include "generic/vhal.h"
#include "generic/valist.h"
#include "generic/vatom.h"
#include "generic/vunit.h"
```

Include dependency graph for vclist.h:



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



Data Structures

- struct sVclistCell

  Atom cell list cell.
- struct sVclist

  Atom cell list.

Typedefs

- typedef enum eVclist_DomainMode Vclist_DomainMode

  Declaration of Vclist_DomainMode enumeration type.
- typedef struct sVclistCell VclistCell

  Declaration of the VclistCell class as the VclistCell structure.
- typedef struct sVclist Vclist

  Declaration of the Vclist class as the Vclist structure.

Enumerations

- enum eVclist_DomainMode { CLIST_AUTO_DOMAIN , CLIST_MANUAL_DOMAIN }

  Atom cell list domain setup mode.

Functions

- VEXTERNC unsigned long int Vclist_memChk (Vclist *thee)

  Get number of bytes in this object and its members.
- VEXTERNC double Vclist_maxRadius (Vclist *thee)

  Get the max probe radius value (in 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.

9.65.1 Detailed Description

Contains declarations for class Vclist.

Version

Id

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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.

9.66 vclist.h

```
00001
00062 #ifndef _VCLIST_H_
00063 #define _VCLIST_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "maloc/maloc.h"
00068 #if defined(HAVE_MC_H)
00069 #include "mc/mc.h"
00070 #endif
00071
00072 #include "generic/vhal.h"
00073 #include "generic/valist.h"
00074 #include "generic/vatom.h"
00075 #include "generic/vunit.h"
00076
00082 enum eVclist_DomainMode {
00083     CLIST_AUTO_DOMAIN,
00085     CLIST_MANUAL_DOMAIN
00087 };
00088
00094 typedef enum eVclist_DomainMode Vclist_DomainMode;
00095
00101 struct sVclistCell {
00102     Vatom **atoms;
00103     int natoms;
00104 };
00105
00110 typedef struct sVclistCell VclistCell;
00111
00117 struct sVclist {
00118
00119   Vmem *vmem;
00120   Valist *alist;
00121   Vclist_DomainMode mode;
00122   int npts[VAPBS_DIM];
00123   int n;
00124   double max_radius;
00125   VclistCell *cells;
00126   double lower_corner[VAPBS_DIM];
00127   double upper_corner[VAPBS_DIM];
00128   double spacs[VAPBS_DIM];
00130 };
00131
00136 typedef struct sVclist Vclist;
00137
```

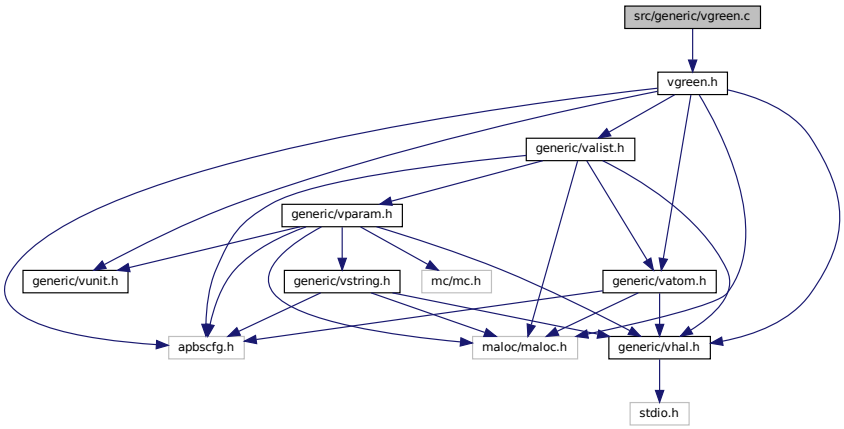
```
00138 #if !defined(VINLINE_VCLIST)
00139
00145     VEXTERNC unsigned long int Vclist_memChk(
00146             Vclist *thee
00147             );
00148
00156     VEXTERNC double Vclist_maxRadius(
00157             Vclist *thee
00158             );
00159
00160 #else /* if defined(VINLINE_VCLIST) */
00161
00162 #   define Vclist_memChk(thee) (Vmem_bytes((thee)->vmem))
00163 #   define Vclist_maxRadius(thee) ((thee)->max_radius)
00164
00165 #endif /* if !defined(VINLINE_VCLIST) */
00166
00167 /* ///////////////////////////////////////////////////////////////////////////
00168 // Class Vclist: Non-Inlineable methods (vclist.c)
00170
00175 VEXTERNC Vclist* Vclist_ctor(
00176         Valist *alist, /**< Molecule for cell list queries */
00177         double max_radius,
00178         int npts[VAPBS_DIM],
00180         Vclist_DomainMode mode,
00181         double lower_corner[VAPBS_DIM],
00184         double upper_corner[VAPBS_DIM]
00187         );
00188
00193 VEXTERNC Vrc_Codes Vclist_ctor2(
00194         Vclist *thee,
00195         Valist *alist,
00196         double max_radius,
00197         int npts[VAPBS_DIM],
00199         Vclist_DomainMode mode,
00200         double lower_corner[VAPBS_DIM],
00203         double upper_corner[VAPBS_DIM]
00206         );
00207
00212 VEXTERNC void Vclist_dtor(
00213         Vclist **thee
00214         );
00215
00220 VEXTERNC void Vclist_dtor2(
00221         Vclist *thee
00222         );
00223
00231 VEXTERNC VclistCell* Vclist_getCell(
00232         Vclist *thee,
00233         double position[VAPBS_DIM]
00234         );
00235
00242 VEXTERNC VclistCell* VclistCell_ctor(
00243         int natoms
00244         );
00245
00252 VEXTERNC Vrc_Codes VclistCell_ctor2(
00253         VclistCell *thee,
00254         int natoms
00255         );
00256
00261 VEXTERNC void VclistCell_dtor(
00262         VclistCell **thee
00263         );
00264
00269 VEXTERNC void VclistCell_dtor2(
00270         VclistCell *thee
00271         );
00272
00273 #endif    /* ifndef _VCLIST_H_ */
```

9.67 src/generic/vgreen.c File Reference

Class Vgreen methods.

```
#include "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)

9.67.1 Detailed Description

Class Vgreen methods.

Author

Nathan Baker

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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) 2002-2020, 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.

9.68 vgreen.c

```
00001
00057 #include "vgreen.h"
00058
00059 /* Define wrappers for F77 treecode routines */
00060 #ifdef HAVE_TREE
00061 #  define F77TREEPEFORCE VF77_MANGLE(treepeforce, TREEPEFORCE)
00062 #  define F77DIRECT_ENG_FORCE VF77_MANGLE(direct_eng_force, DIRECT_ENG_FORCE)
00063 #  define F77CLEANUP VF77_MANGLE(mycleanup, MYCLEANUP)
00064 #  define F77TREE_COMPP VF77_MANGLE(mytree_compp, MYTREE_COMP)
00065 #  define F77TREE_COMPFP VF77_MANGLE(mytree_compfp, MYTREE_COMPFP)
00066 #  define F77CREATE_TREE VF77_MANGLE(mycreate_tree, MYCREATE_TREE)
00067 #  define F77INITLEVELS VF77_MANGLE(myinitlevels, MYINITLEVELS)
00068 #  define F77SETUP VF77_MANGLE(mysetup, MYSETUP)
00069 #endif  /* ifdef HAVE_TREE */
00070
00071 /* Some constants associated with the tree code */
00072 #ifdef HAVE_TREE
00076 #   define FMM_DIST_TOL VSMALL
00082 #   define FMM_IFLAG 2
00086 #   define FMM_ORDER 4
00090 #   define FMM_THETA 0.5
00094 #   define FMM_MAXPARNODE 150
00098 #   define FMM_SHRINK 1
00102 #   define FMM_MINLEVEL 50000
00106 #   define FMM_MAXLEVEL 0
00107 #endif  /* ifdef HAVE_TREE */
00108
00109
00110 /*
00111  * @brief  Setup treecode internal structures
00112  * @ingroup  Vgreen
00113  * @author  Nathan Baker
00114  * @param  thee  Vgreen object
00115  * @return  1 if successful, 0 otherwise
00116  */
00117 VPRIVATE int treesetup(Vgreen *thee);
00118
00119 /*
00120  * @brief  Clean up treecode internal structures
00121  * @ingroup  Vgreen
00122  * @author  Nathan Baker
00123  * @param  thee  Vgreen object
00124  * @return  1 if successful, 0 otherwise
00125  */
00126 VPRIVATE int treecleanup(Vgreen *thee);
00127
00128 /*
00129  * @brief  Calculate forces or potential
00130  * @ingroup  Vgreen
00131  * @author  Nathan Baker
00132  * @param  thee  Vgreen object
00133  * @return  1 if successful, 0 otherwise
00134  */
00135 VPRIVATE int treecalc(Vgreen *thee, double *xtar, double *ytar, double *ztar,
00136         double *qtar, int numtars, double *tpengtar, double *x, double *y,
00137         double *z, double *q, int numpars, double *fx, double *fy, double *fz,
00138         int iflag, int farrdim, int arrdim);
00139
00140 #if !defined(VINLINE_VGREEN)
00141
00142 VPUBLIC Valist* Vgreen_getValist(Vgreen *thee) {
00143
00144    VASSERT(thee != VNULL);
00145    return thee->alist;
```

```
00146
00147 }
00148
00149 VPUBLIC unsigned long int Vgreen_memChk(Vgreen *thee) {
00150     if (thee == VNULL) return 0;
00151     return Vmem_bytes(thee->vmem);
00152 }
00153
00154 #endif /* if !defined(VINLINE_VGREEN) */
00155
00156 VPUBLIC Vgreen* Vgreen_ctor(Valist *alist) {
00157
00158     /* Set up the structure */
00159     Vgreen *thee = VNULL;
00160     thee = (Vgreen *)Vmem_malloc(VNULL, 1, sizeof(Vgreen) );
00161     VASSERT( thee != VNULL);
00162     VASSERT( Vgreen_ctor2(thee, alist));
00163
00164     return thee;
00165 }
00166
00167 VPUBLIC int Vgreen_ctor2(Vgreen *thee, Valist *alist) {
00168
00169     VASSERT( thee != VNULL );
00170
00171     /* Memory management object */
00172     thee->vmem = Vmem_ctor("APBS:VGREEN");
00173
00174     /* Set up the atom list and grid manager */
00175     if (alist == VNULL) {
00176         Vnm_print(2,"Vgreen_ctor2: got null pointer to Valist object!\n");
00177     }
00178
00179     thee->alist = alist;
00180
00181     /* Setup FMM tree (if applicable) */
00182 #ifdef HAVE_TREE
00183     if (!treesetup(thee)) {
00184         Vnm_print(2, "Vgreen_ctor2:  Error setting up FMM tree!\n");
00185         return 0;
00186     }
00187 #endif /* ifdef HAVE_TREE */
00188
00189     return 1;
00190 }
00191
00192 VPUBLIC void Vgreen_dtor(Vgreen **thee) {
00193     if ((*thee) != VNULL) {
00194         Vgreen_dtor2(*thee);
00195         Vmem_free(VNULL, 1, sizeof(Vgreen), (void **)thee);
00196         (*thee) = VNULL;
00197     }
00198 }
00199
00200 VPUBLIC void Vgreen_dtor2(Vgreen *thee) {
00201
00202 #ifdef HAVE_TREE
00203     treecleanup(thee);
00204 #endif
00205     Vmem_dtor(&(thee->vmem));
00206
00207 }
00208
00209 VPUBLIC int Vgreen_helmholtz(Vgreen *thee, int npos, double *x, double *y,
00210   double *z, double *val, double kappa) {
00211
00212     Vnm_print(2, "Error -- Vgreen_helmholtz not implemented yet!\n");
00213     return 0;
00214 }
00215
00216 VPUBLIC int Vgreen_helmholtzD(Vgreen *thee, int npos, double *x, double *y,
00217   double *z, double *gradx, double *grady, double *gradz, double kappa) {
00218
00219     Vnm_print(2, "Error -- Vgreen_helmholtzD not implemented yet!\n");
00220     return 0;
00221
00222 }
00223
00224 VPUBLIC int Vgreen_coulomb_direct(Vgreen *thee, int npos, double *x,
00225         double *y, double *z, double *val) {
00226
```

```
00227     Vatom *atom;
00228     double *apos, charge, dist, dx, dy, dz, scale;
00229     double *q, qtemp, fx, fy, fz;
00230     int iatom, ipos;
00231
00232     if (thee == VNULL) {
00233         Vnm_print(2, "Vgreen_coulomb:  Got NULL thee!\n");
00234         return 0;
00235     }
00236
00237     for (ipos=0; ipos<npos; ipos++) val[ipos] = 0.0;
00238
00239     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
00240         atom = Valist_getAtom(thee->alist, iatom);
00241         apos = Vatom_getPosition(atom);
00242         charge = Vatom_getCharge(atom);
00243         for (ipos=0; ipos<npos; ipos++) {
00244             dx = apos[0] - x[ipos];
00245             dy = apos[1] - y[ipos];
00246             dz = apos[2] - z[ipos];
00247             dist = VSQRT(VSQR(dx) + VSQR(dy) + VSQR(dz));
00248             if (dist > VSMALL) val[ipos] += (charge/dist);
00249         }
00250     }
00251
00252     scale = Vunit_ec/(4*Vunit_pi*Vunit_eps0*1.0e-10);
00253     for (ipos=0; ipos<npos; ipos++) val[ipos] = val[ipos]*scale;
00254
00255     return 1;
00256 }
00257
00258 VPUBLIC int Vgreen_coulomb(Vgreen *thee, int npos, double *x, double *y,
00259   double *z, double *val) {
00260
00261     Vatom *atom;
00262     double *apos, charge, dist, dx, dy, dz, scale;
00263     double *q, qtemp, fx, fy, fz;
00264     int iatom, ipos;
00265
00266     if (thee == VNULL) {
00267         Vnm_print(2, "Vgreen_coulomb:  Got NULL thee!\n");
00268         return 0;
00269     }
00270
00271     for (ipos=0; ipos<npos; ipos++) val[ipos] = 0.0;
00272
00273 #ifdef HAVE_TREE
00274
00275     /* Allocate charge array (if necessary) */
00276     if (Valist_getNumberAtoms(thee->alist) > 1) {
00277         if (npos > 1) {
00278             q = VNULL;
00279             q = Vmem_malloc(thee->vmem, npos, sizeof(double));
00280             if (q == VNULL) {
00281                 Vnm_print(2, "Vgreen_coulomb:  Error allocating charge array!\n");
00282                 return 0;
00283             }
00284         } else {
00285             q = &(qtemp);
00286         }
00287         for (ipos=0; ipos<npos; ipos++) q[ipos] = 1.0;
00288
00289         /* Calculate */
00290         treecalc(thee, x, y, z, q, npos, val, thee->xp, thee->yp, thee->zp,
00291           thee->qp, thee->np, &fx, &fy, &fz, 1, 1, thee->np);
00292     } else return Vgreen_coulomb_direct(thee, npos, x, y, z, val);
00293
00294     /* De-allocate charge array (if necessary) */
00295     if (npos > 1) Vmem_free(thee->vmem, npos, sizeof(double), (void **)&q);
00296
00297     scale = Vunit_ec/(4*Vunit_pi*Vunit_eps0*1.0e-10);
00298     for (ipos=0; ipos<npos; ipos++) val[ipos] = val[ipos]*scale;
00299
00300     return 1;
00301
00302 #else /* ifdef HAVE_TREE */
00303
00304     return Vgreen_coulomb_direct(thee, npos, x, y, z, val);
00305
00306 #endif
00307
```

```
00308 }
00309
00310 VPUBLIC int Vgreen_coulombD_direct(Vgreen *thee, int npos,
00311         double *x, double *y, double *z, double *pot, double *gradx,
00312         double *grady, double *gradz) {
00313
00314     Vatom *atom;
00315     double *apos, charge, dist, dist2, idist3, dy, dz, dx, scale;
00316     double *q, qtemp;
00317     int iatom, ipos;
00318
00319     if (thee == VNULL) {
00320         Vnm_print(2, "Vgreen_coulombD:  Got VNULL thee!\n");
00321         return 0;
00322     }
00323
00324     for (ipos=0; ipos<npos; ipos++) {
00325         pot[ipos] = 0.0;
00326         gradx[ipos] = 0.0;
00327         grady[ipos] = 0.0;
00328         gradz[ipos] = 0.0;
00329     }
00330
00331     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
00332         atom = Valist_getAtom(thee->alist, iatom);
00333         apos = Vatom_getPosition(atom);
00334         charge = Vatom_getCharge(atom);
00335         for (ipos=0; ipos<npos; ipos++) {
00336             dx = apos[0] - x[ipos];
00337             dy = apos[1] - y[ipos];
00338             dz = apos[2] - z[ipos];
00339             dist2 = VSQR(dx) + VSQR(dy) + VSQR(dz);
00340             dist = VSQRT(dist2);
00341             if (dist > VSMALL) {
00342                 idist3 = 1.0/(dist*dist2);
00343                 gradx[ipos] -= (charge*dx*idist3);
00344                 grady[ipos] -= (charge*dy*idist3);
00345                 gradz[ipos] -= (charge*dz*idist3);
00346                 pot[ipos] += (charge/dist);
00347             }
00348         }
00349     }
00350
00351     scale = Vunit_ec/(4*VPI*Vunit_eps0*(1.0e-10));
00352     for (ipos=0; ipos<npos; ipos++) {
00353         gradx[ipos] = gradx[ipos]*scale;
00354         grady[ipos] = grady[ipos]*scale;
00355         gradz[ipos] = gradz[ipos]*scale;
00356         pot[ipos] = pot[ipos]*scale;
00357     }
00358
00359     return 1;
00360 }
00361
00362 VPUBLIC int Vgreen_coulombD(Vgreen *thee, int npos, double *x, double *y,
00363         double *z, double *pot, double *gradx, double *grady, double *gradz) {
00364
00365     Vatom *atom;
00366     double *apos, charge, dist, dist2, idist3, dy, dz, dx, scale;
00367     double *q, qtemp;
00368     int iatom, ipos;
00369
00370     if (thee == VNULL) {
00371         Vnm_print(2, "Vgreen_coulombD:  Got VNULL thee!\n");
00372         return 0;
00373     }
00374
00375     for (ipos=0; ipos<npos; ipos++) {
00376         pot[ipos] = 0.0;
00377         gradx[ipos] = 0.0;
00378         grady[ipos] = 0.0;
00379         gradz[ipos] = 0.0;
00380     }
00381
00382 #ifdef HAVE_TREE
00383
00384     if (Valist_getNumberAtoms(thee->alist) > 1) {
00385         if (npos > 1) {
00386             q = VNULL;
00387             q = Vmem_malloc(thee->vmem, npos, sizeof(double));
00388             if (q == VNULL) {
```

```
00389                 Vnm_print(2, "Vgreen_coulomb:  Error allocating charge array!\n");
00390                 return 0;
00391             }
00392         } else {
00393             q = &(qtemp);
00394         }
00395         for (ipos=0; ipos<npos; ipos++) q[ipos] = 1.0;
00396 
00397         /* Calculate */
00398         treecalc(thee, x, y, z, q, npos, pot, thee->xp, thee->yp, thee->zp,
00399                 thee->qp, thee->np, gradx, grady, gradz, 2, npos, thee->np);
00400 
00401         /* De-allocate charge array (if necessary) */
00402         if (npos > 1) Vmem_free(thee->vmem, npos, sizeof(double), (void **)&q);
00403     } else return Vgreen_coulombD_direct(thee, npos, x, y, z, pot,
00404             gradx, grady, gradz);
00405 
00406     scale = Vunit_ec/(4*VPI*Vunit_eps0*(1.0e-10));
00407     for (ipos=0; ipos<npos; ipos++) {
00408         gradx[ipos] = gradx[ipos]*scale;
00409         grady[ipos] = grady[ipos]*scale;
00410         gradz[ipos] = gradz[ipos]*scale;
00411         pot[ipos] = pot[ipos]*scale;
00412     }
00413 
00414     return 1;
00415 
00416 #else /* ifdef HAVE_TREE */
00417 
00418     return Vgreen_coulombD_direct(thee, npos, x, y, z, pot,
00419             gradx, grady, gradz);
00420 
00421 #endif
00422 
00423 }
00424 
00425 VPRIVATE int treesetup(Vgreen *thee) {
00426 
00427 #ifdef HAVE_TREE
00428 
00429     double dist_tol = FMM_DIST_TOL;
00430     int iflag = FMM_IFLAG;
00431     double order = FMM_ORDER;
00432     int theta = FMM_THETA;
00433     int shrink = FMM_SHRINK;
00434     int maxparnode = FMM_MAXPARNODE;
00435     int minlevel = FMM_MINLEVEL;
00436     int maxlevel = FMM_MAXLEVEL;
00437     int level = 0;
00438     int one = 1;
00439     Vatom *atom;
00440     double xyzminmax[6], *pos;
00441     int i;
00442 
00443     /* Set up particle arrays with atomic coordinates and charges */
00444     Vnm_print(0, "treesetup:  Initializing FMM particle arrays...\n");
00445     thee->np = Valist_getNumberAtoms(thee->alist);
00446     thee->xp = VNULL;
00447     thee->xp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(double));
00448     if (thee->xp == VNULL) {
00449         Vnm_print(2, "Vgreen_ctor2:  Failed to allocate %d*sizeof(double)!\n",
00450           thee->np);
00451         return 0;
00452     }
00453     thee->yp = VNULL;
00454     thee->yp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(double));
00455     if (thee->yp == VNULL) {
00456         Vnm_print(2, "Vgreen_ctor2:  Failed to allocate %d*sizeof(double)!\n",
00457           thee->np);
00458         return 0;
00459     }
00460     thee->zp = VNULL;
00461     thee->zp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(double));
00462     if (thee->zp == VNULL) {
00463         Vnm_print(2, "Vgreen_ctor2:  Failed to allocate %d*sizeof(double)!\n",
00464           thee->np);
00465         return 0;
00466     }
00467     thee->qp = VNULL;
00468     thee->qp = (double *)Vmem_malloc(thee->vmem, thee->np, sizeof(double));
00469     if (thee->qp == VNULL) {
```

```
00470         Vnm_print(2, "Vgreen_ctor2:  Failed to allocate %d*sizeof(double)!\n",
00471           thee->np);
00472         return 0;
00473     }
00474     for (i=0; i<thee->np; i++) {
00475         atom = Valist_getAtom(thee->alist, i);
00476         pos = Vatom_getPosition(atom);
00477         thee->xp[i] = pos[0];
00478         thee->yp[i] = pos[1];
00479         thee->zp[i] = pos[2];
00480         thee->qp[i] = Vatom_getCharge(atom);
00481     }
00482
00483     Vnm_print(0, "treesetup:  Setting things up...\n");
00484     F77SETUP(thee->xp, thee->yp, thee->zp, &(thee->np), &order, &theta, &iflag,
00485             &dist_tol, xyzminmax, &(thee->np));
00486
00487
00488     Vnm_print(0, "treesetup:  Initializing levels...\n");
00489     F77INITLEVELS(&minlevel, &maxlevel);
00490
00491     Vnm_print(0, "treesetup:  Creating tree...\n");
00492     F77CREATE_TREE(&one, &(thee->np), thee->xp, thee->yp, thee->zp, thee->qp,
00493       &shrink, &maxparnode, xyzminmax, &level, &(thee->np));
00494
00495     return 1;
00496
00497 #else /* ifdef HAVE_TREE */
00498
00499     Vnm_print(2, "treesetup:  Error!  APBS not linked with treecode!\n");
00500     return 0;
00501
00502 #endif /* ifdef HAVE_TREE */
00503 }
00504
00505 VPRIVATE int treecleanup(Vgreen *thee) {
00506
00507 #ifdef HAVE_TREE
00508
00509     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->xp));
00510     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->yp));
00511     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->zp));
00512     Vmem_free(thee->vmem, thee->np, sizeof(double), (void **)&(thee->qp));
00513     F77CLEANUP();
00514
00515     return 1;
00516
00517 #else /* ifdef HAVE_TREE */
00518
00519     Vnm_print(2, "treecleanup:  Error!  APBS not linked with treecode!\n");
00520     return 0;
00521
00522 #endif /* ifdef HAVE_TREE */
00523 }
00524
00525 VPRIVATE int treecalc(Vgreen *thee, double *xtar, double *ytar, double *ztar,
00526         double *qtar, int numtars, double *tpengtar, double *x, double *y,
00527         double *z, double *q, int numpars, double *fx, double *fy, double *fz,
00528         int iflag, int farrdim, int arrdim) {
00529
00530 #ifdef HAVE_TREE
00531     int i, level, err, maxlevel, minlevel, one;
00532     double xyzminmax[6];
00533
00534
00535     if (iflag != 1) {
00536         F77TREE_COMPFP(xtar, ytar, ztar, qtar, &numtars, tpengtar, x, y, z, q,
00537                 fx, fy, fz, &numpars, &farrdim, &arrdim);
00538     } else {
00539         F77TREE_COMPP(xtar, ytar, ztar, qtar, &numtars, tpengtar, &farrdim, x,
00540                 y, z, q, &numpars, &arrdim);
00541     }
00542
00543
00544     return 1;
00545
00546 #else /* ifdef HAVE_TREE */
00547
00548     Vnm_print(2, "treecalc:  Error!  APBS not linked with treecode!\n");
00549     return 0;
00550
```

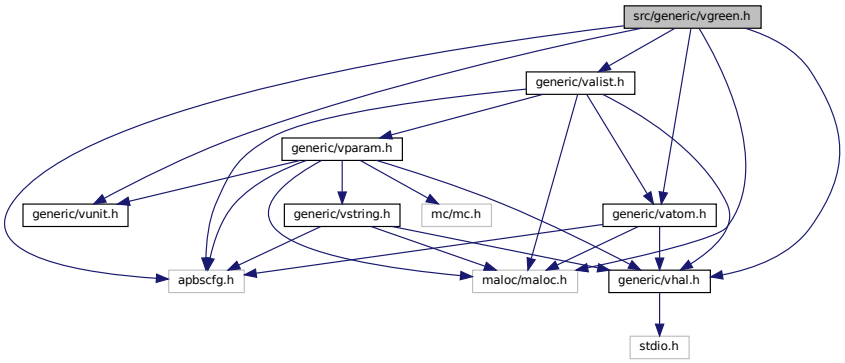
```
00551 #endif /* ifdef HAVE_TREE */
00552 }
```

9.69 src/generic/vgreen.h File Reference

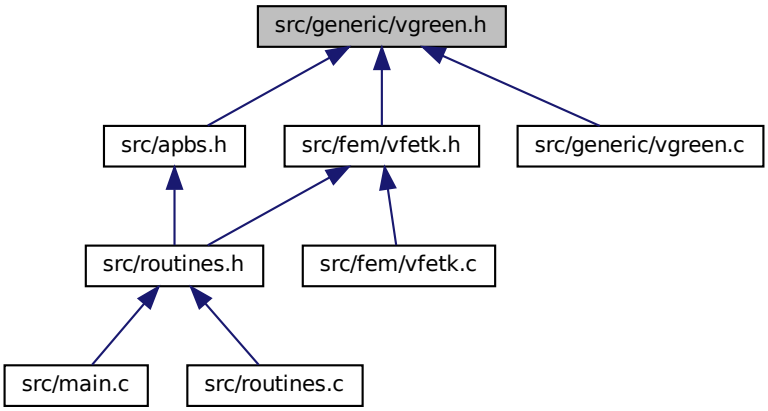
Contains declarations for class Vgreen.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/vunit.h"
#include "generic/vatom.h"
#include "generic/valist.h"
```

Include dependency graph for vgreen.h:



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



Data Structures

- struct sVgreen

  Contains public data members for Vgreen class/module.

Typedefs

- typedef struct sVgreen Vgreen

  Declaration of the Vgreen class as the Vgreen structure.

Functions

- VEXTERNC Valist * Vgreen_getValist (Vgreen *thee)

  Get the atom list associated with this Green's function object.
- VEXTERNC unsigned long int Vgreen_memChk (Vgreen *thee)

  Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC Vgreen * Vgreen_ctor (Valist *alist)

  Construct the Green's function oracle.
- VEXTERNC int Vgreen_ctor2 (Vgreen *thee, Valist *alist)

  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)

9.69.1 Detailed Description

Contains declarations for class Vgreen.

Version

Id

Author

Nathan A. Baker

Definition in file vgreen.h.

9.70 vgreen.h

```
00001
00065 #ifndef _VGREEN_H_
00066 #define _VGREEN_H_
00067
00068 #include "apbscfg.h"
00069
00070 #include "maloc/maloc.h"
00071
00072 #include "generic/vhal.h"
00073 #include "generic/vunit.h"
00074 #include "generic/vatom.h"
00075 #include "generic/valist.h"
00076
00082 struct sVgreen {
00083
00084   Valist *alist;
00085   Vmem *vmem;
00086   double *xp;
00088   double *yp;
00090   double *zp;
00092   double *qp;
00094   int np;
00095 };
00096
00101 typedef struct sVgreen Vgreen;
00102
00103 /* ///////////////////////////////////////////////////////////////////////////
00104 // Class Vgreen: Inlineable methods (vgreen.c)
00106
00107 #if !defined(VINLINE_VGREEN)
00108
00116     VEXTERNC Valist* Vgreen_getValist(Vgreen *thee);
00117
00125     VEXTERNC unsigned long int Vgreen_memChk(Vgreen *thee);
00126
00127 #else /* if defined(VINLINE_VGREEN) */
00128 #   define Vgreen_getValist(thee) ((thee)->alist)
00129 #   define Vgreen_memChk(thee) (Vmem_bytes((thee)->vmem))
00130 #endif /* if !defined(VINLINE_VGREEN) */
00131
00132 /* ///////////////////////////////////////////////////////////////////////////
00133 // Class Vgreen: Non-Inlineable methods (vgreen.c)
00135
00142 VEXTERNC Vgreen* Vgreen_ctor(Valist *alist);
00143
00151 VEXTERNC int Vgreen_ctor2(Vgreen *thee, Valist *alist);
00152
00158 VEXTERNC void Vgreen_dtor(Vgreen **thee);
00159
00165 VEXTERNC void Vgreen_dtor2(Vgreen *thee);
00166
00191 VEXTERNC int Vgreen_helmholtz(Vgreen *thee, int npos, double *x, double *y,
00192   double *z, double *val, double kappa);
00193
00221 VEXTERNC int Vgreen_helmholtzD(Vgreen *thee, int npos, double *x, double *y,
00222   double *z, double *gradx, double *grady, double *gradz, double kappa);
00223
00244 VEXTERNC int Vgreen_coulomb_direct(Vgreen *thee, int npos, double *x,
00245         double *y, double *z, double *val);
00246
00267 VEXTERNC int Vgreen_coulomb(Vgreen *thee, int npos, double *x, double *y,
00268   double *z, double *val);
00269
00293 VEXTERNC int Vgreen_coulombD_direct(Vgreen *thee, int npos, double *x,
```

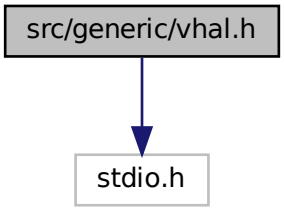
```
00294         double *y, double *z, double *pot, double *gradx, double *grady, double
00295         *gradz);
00296
00321 VEXTERNC int Vgreen_coulombD(Vgreen *thee, int npos, double *x, double *y,
00322   double *z, double *pot, double *gradx, double *grady, double *gradz);
00323
00324 #endif /* ifndef _VGREEN_H_ */
```

9.71 src/generic/vhal.h File Reference

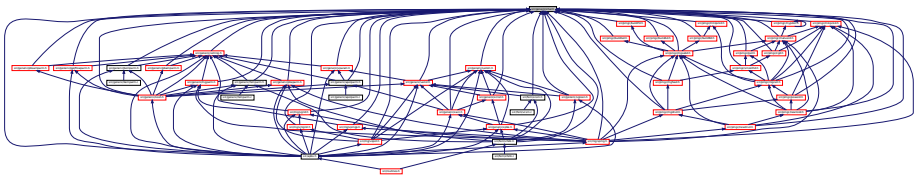
Contains generic macro definitions for APBS.

`#include "stdio.h"`

Include dependency graph for vhal.h:



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



Macros

- #define APBS_TIMER_WALL_CLOCK 26

  APBS total execution timer ID.
- #define APBS_TIMER_SETUP 27

  APBS setup timer ID.
- #define APBS_TIMER_SOLVER 28

  APBS solver timer ID.
- #define APBS_TIMER_ENERGY 29

  APBS energy timer ID.
- #define APBS_TIMER_FORCE 30

  APBS force timer ID.
- #define APBS_TIMER_TEMP1 31

  APBS temp timer #1 ID.
- #define APBS_TIMER_TEMP2 32

APBS temp timer #2 ID.

- #define MAXMOL 5

  The maximum number of molecules that can be involved in a single PBE calculation.
- #define MAXION 10

  The maximum number of ion species that can be involved in a single PBE calculation.
- #define MAXFOCUS 5

  The maximum number of times an MG calculation can be focused.
- #define VMGNLEV 4

  Minimum number of levels in a multigrid calculations.
- #define VREDFRAC 0.25

  Maximum reduction of grid spacing during a focusing calculation.
- #define VAPBS_NVS 4

  Number of vertices per simplex (hard-coded to 3D)
- #define VAPBS_DIM 3

  Our dimension.
- #define VAPBS_RIGHT 0

  Face definition for a volume.
- #define 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 **MAX_HASH_DIM** 75
- #define VF77_MANGLE(name, NAME) name

  Name-mangling macro for using FORTRAN functions in C code.
- #define VFLOOR(value) floor(value)

  Wrapped floor to fix floating point issues in the Intel compiler.
- #define VEMBED(rctag)

  Allows embedding of RCS ID tags in object files.
- #define PRINT_FUNC __PRETTY_FUNCTION__
- #define **OS_SEP_STR** "/"
- #define **OS_SEP_CHAR** '/'
- #define **ANNOUNCE_FUNCTION**
- #define **WARN_UNTESTED**
- #define **WARN_PARTTESTED**
- #define **VCHANNELEDMESSAGE0**(channel, msg)
- #define **VCHANNELEDMESSAGE1**(channel, msg, arg0)

- #define **VCHANNELEDMESSAGE2**(channel, msg, arg0, arg1)
- #define **VCHANNELEDMESSAGE3**(channel, msg, arg0, arg1, arg2)
- #define **VMESSAGE0**(msg) VCHANNELEDMESSAGE0(0, msg)
- #define **VMESSAGE1**(msg, arg0) VCHANNELEDMESSAGE1(0, msg, arg0)
- #define **VMESSAGE2**(msg, arg0, arg1) VCHANNELEDMESSAGE2(0, msg, arg0, arg1)
- #define **VMESSAGE3**(msg, arg0, arg1, arg2) VCHANNELEDMESSAGE3(0, msg, arg0, arg1, arg2)
- #define **VERRMSG0**(msg) VCHANNELEDMESSAGE0(2, msg)
- #define **VERRMSG1**(msg, arg0) VCHANNELEDMESSAGE1(2, msg, arg0)
- #define **VERRMSG2**(msg, arg0, arg1) VCHANNELEDMESSAGE2(2, msg, arg0, arg1)
- #define **VERRMSG3**(msg, arg0, arg1, arg2) VCHANNELEDMESSAGE3(2, msg, arg0, arg1, arg2)
- #define **VASSERT_MSG0**(cnd, msg)
- #define **VASSERT_MSG1**(cnd, msg, arg)
- #define **VASSERT_MSG2**(cnd, msg, arg0, arg1)
- #define **VWARN_MSG0**(cnd, msg)
- #define **VWARN_MSG1**(cnd, msg, arg0)
- #define **VWARN_MSG2**(cnd, msg, arg0, arg1)
- #define **VABORT_MSG0**(msg)
- #define **VABORT_MSG1**(msg, arg)
- #define **VABORT_MSG2**(msg, arg0, arg1)
- #define **PRINT_INT**(expr)
- #define **PRINT_DBL**(expr)
- #define **VMALLOC**(vmem, n, type) ((type∗)Vmem_malloc(vmem, n, sizeof(type)))
- #define **VFREE**(vmem, n, type, ptr) (Vmem_free(vmem, n, sizeof(type), (void ∗∗)&(ptr)))
- #define **VFILL**(vec, n, val)
- #define **VCOPY**(srcvec, dstvec, i, n)
- #define **VAT**(array, i) ((array)[(i) - 1])
- #define **RAT**(array, i) ((array) + i - 1)

Typedefs

- typedef enum eVrc_Codes **Vrc_Codes**
- typedef enum eVsol_Meth **Vsol_Meth**
- typedef enum eVsurf_Meth Vsurf_Meth

  Declaration of the Vsurf_Meth type as the Vsurf_Meth enum.
- typedef enum eVhal_PBEType Vhal_PBEType

  Declaration of the Vhal_PBEType type as the Vhal_PBEType enum.
- typedef enum eVhal_IPKEYType Vhal_IPKEYType

  Declaration of the Vhal_IPKEYType type as the Vhal_IPKEYType enum.
- typedef enum eVhal_NONLINType Vhal_NONLINType

  Declaration of the Vhal_NONLINType type as the Vhal_NONLINType enum.
- typedef enum eVoutput_Format Voutput_Format

  Declaration of the Voutput_Format type as the VOutput_Format enum.
- typedef enum eVbcfl Vbcfl

  Declare Vbcfl type.
- typedef enum eVchrg_Meth Vchrg_Meth

  Declaration of the Vchrg_Meth type as the Vchrg_Meth enum.
- typedef enum eVchrg_Src Vchrg_Src

  Declaration of the Vchrg_Src type as the Vchrg_Meth enum.
- typedef enum eVdata_Type Vdata_Type

  Declaration of the Vdata_Type type as the Vdata_Type enum.
- typedef enum eVdata_Format Vdata_Format

  Declaration of the Vdata_Format type as the Vdata_Format enum.

Enumerations

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

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

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

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

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

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

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

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

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

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

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

  Types of (scalar) data that can be written out of APBS.
- enum eVdata_Format {
  VDF_DX =0 , VDF_UHBD =1 , VDF_AVS =2 , VDF_MCSF =3 ,
  VDF_GZ =4 , VDF_FLAT =5 , VDF_DXBIN =6 }

  Format of data for APBS I/O.

Functions

- char ∗ **wrap_text** (char ∗str, int right_margin, int left_padding)

9.71.1 Detailed Description

Contains generic macro definitions for APBS.

Version

Id

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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.

9.71.2 Macro Definition Documentation

9.71.2.1 PRINT_FUNC

```
#define PRINT_FUNC __PRETTY_FUNCTION__
```

OS specific flags and etcetera
Definition at line 571 of file vhal.h.

9.71.2.2 VABORT_MSG0

```
#define VABORT_MSG0(
                msg )
```

Value:

```
        do {                                        \
            Vnm_print(2, "%[%s()]: ABORTING:\n" \
                         "    %s\n\n",          \
                      __FUNCTION__, msg);       \
            abort();                                \
        } while(0)
```

Definition at line 899 of file vhal.h.

9.71.2.3 VABORT_MSG1

```
#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 907 of file vhal.h.

9.71.2.4 VABORT_MSG2

```
#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 917 of file vhal.h.

9.71.2.5 VASSERT_MSG0

```
#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 731 of file vhal.h.

9.71.2.6 VASSERT_MSG1

```
#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 741 of file vhal.h.

9.71.2.7 VASSERT_MSG2

```
#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 753 of file vhal.h.

9.71.2.8 VCHANNELEDMESSAGE0

```
#define VCHANNELEDMESSAGE0(
                channel,
                msg )
```

Value:

```
        do {                                                    \
            Vnm_print(channel, "%s: %s\n", __FUNCTION__, msg); \
        } while(0)
```

Definition at line 653 of file vhal.h.

9.71.2.9 VCHANNELEDMESSAGE1

```
#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 658 of file vhal.h.

9.71.2.10 VCHANNELEDMESSAGE2

```
#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 665 of file vhal.h.

9.71.2.11 VCHANNELEDMESSAGE3

```
#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 672 of file vhal.h.

9.71.2.12 VCOPY

```
#define VCOPY(
                srcvec,
                dstvec,
                i,
                n )
```

Value:

```
    do {                            \
        for (i = 0; i < n; i++)     \
            dstvec[i] = srcvec[i];  \
    } while(0)
```

Definition at line 959 of file vhal.h.

9.71.2.13 VFILL

```
#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 952 of file vhal.h.

9.71.2.14 VWARN_MSG0

```
#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 825 of file vhal.h.

9.71.2.15 VWARN_MSG1

```
#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 838 of file vhal.h.

9.71.2.16 VWARN_MSG2

```
#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 853 of file vhal.h.

9.72 vhal.h

```
00001
00055 #ifndef _VAPBSHAL_H_
00056 #define _VAPBSHAL_H_
00057
00058 #include "stdio.h"
00059
00066 enum eVrc_Codes {
00067
00068     VRC_WARNING=-1,
00069     VRC_FAILURE=0,
00070     VRC_SUCCESS=1
00072 };
00073 typedef enum eVrc_Codes Vrc_Codes;
00074
00081 enum eVsol_Meth {
00082
00083     VSOL_CGMG,        /* 0: conjugate gradient multigrid */
00084     VSOL_Newton,    /* 1: newton */
00085     VSOL_MG,        /* 2: multigrid */
00086     VSOL_CG,        /* 3: conjugate gradient */
00087     VSOL_SOR,        /* 4: sucessive overrelaxation */
00088     VSOL_RBGS,        /* 5: red-black gauss-seidel */
00089     VSOL_WJ,        /* 6: weighted jacobi */
00090     VSOL_Richardson,/* 7: richardson  */
00091     VSOL_CGMGAqua,    /* 8: conjugate gradient multigrid aqua */
00092     VSOL_NewtonAqua    /* 9: newton aqua */
00093
00094 };
00095 typedef enum eVsol_Meth Vsol_Meth;
00096
00102 enum eVsurf_Meth {
00103     VSM_MOL=0,
00107     VSM_MOLSMOOTH=1,
00109     VSM_SPLINE=2,
00119     VSM_SPLINE3=3,
00123     VSM_SPLINE4=4
00127 };
00128
00133 typedef enum eVsurf_Meth Vsurf_Meth;
00134
00139 enum eVhal_PBEType {
00140     PBE_LPBE,
00141     PBE_NPBE,
00142     PBE_LRPBE,
00143     PBE_NRPBE,
00144     PBE_SMPBE
00145 };
00146
00151 typedef enum eVhal_PBEType Vhal_PBEType;
00152
00157 enum eVhal_IPKEYType {
00158     IPKEY_SMPBE = -2,
00159     IPKEY_LPBE,
00160     IPKEY_NPBE
00161 };
00162
00167 typedef enum eVhal_IPKEYType Vhal_IPKEYType;
00168
00173 enum eVhal_NONLINType {
00174     NONLIN_LPBE = 0,
00175     NONLIN_NPBE,
00176     NONLIN_SMPBE,
00177     NONLIN_LPBEAQUA,
00178     NONLIN_NPBEAQUA
00179 };
00180
00185 typedef enum eVhal_NONLINType Vhal_NONLINType;
00186
00191 enum eVoutput_Format {
00192     OUTPUT_NULL,
```

```
00193     OUTPUT_FLAT,
00194 };
00195
00200 typedef enum eVoutput_Format Voutput_Format;
00201
00207 enum eVbcfl {
00208     BCFL_ZERO=0,
00209     BCFL_SDH=1,
00211     BCFL_MDH=2,
00213     BCFL_UNUSED=3,
00214     BCFL_FOCUS=4,
00215     BCFL_MEM=5,
00216     BCFL_MAP=6
00217 };
00218
00223 typedef enum eVbcfl Vbcfl;
00224
00230 enum eVchrg_Meth {
00231     VCM_TRIL=0,
00234     VCM_BSPL2=1,
00237     VCM_BSPL4=2
00238 };
00239
00244 typedef enum eVchrg_Meth Vchrg_Meth;
00245
00251 enum eVchrg_Src {
00252     VCM_CHARGE=0,
00253     VCM_PERMANENT=1,
00254     VCM_INDUCED=2,
00255     VCM_NLINDUCED=3
00256 };
00257
00262 typedef enum eVchrg_Src Vchrg_Src;
00263
00269 enum eVdata_Type {
00270     VDT_CHARGE,
00271     VDT_POT,
00272     VDT_ATOMPOT,
00273     VDT_SMOL,
00275     VDT_SSPL,
00277     VDT_VDW,
00279     VDT_IVDW,
00281     VDT_LAP,
00282     VDT_EDENS,
00284     VDT_NDENS,
00286     VDT_QDENS,
00288     VDT_DIELX,
00290     VDT_DIELY,
00292     VDT_DIELZ,
00294     VDT_KAPPA
00296 };
00297
00302 typedef enum eVdata_Type Vdata_Type;
00303
00309 enum eVdata_Format {
00310     VDF_DX=0,
00311     VDF_UHBD=1,
00312     VDF_AVS=2,
00313     VDF_MCSF=3,
00314     VDF_GZ=4,
00315     VDF_FLAT=5,
00316  VDF_DXBIN=6
00317 };
00318
00323 typedef enum eVdata_Format Vdata_Format;
00324
00329 #define APBS_TIMER_WALL_CLOCK 26
00330
00335 #define APBS_TIMER_SETUP 27
00336
00341 #define APBS_TIMER_SOLVER 28
00342
00347 #define APBS_TIMER_ENERGY 29
00348
00353 #define APBS_TIMER_FORCE 30
00354
00359 #define APBS_TIMER_TEMP1 31
00360
00365 #define APBS_TIMER_TEMP2 32
00366
00371 #define MAXMOL 5
```

```
00372
00377 #define MAXION 10
00378
00382 #define MAXFOCUS 5
00383
00387 #define VMGNLEV 4
00388
00392 #define VREDFRAC 0.25
00393
00397 #define VAPBS_NVS 4
00398
00402 #define VAPBS_DIM 3
00403
00408 #define VAPBS_RIGHT 0
00409
00414 #define VAPBS_FRONT 1
00415
00420 #define VAPBS_UP    2
00421
00426 #define VAPBS_LEFT  3
00427
00432 #define VAPBS_BACK  4
00433
00438 #define VAPBS_DOWN  5
00439
00444 #define VPMGSMALL 1e-12
00445
00450 #define SINH_MIN -85.0
00451
00456 #define SINH_MAX 85.0
00457
00458 #define MAX_HASH_DIM 75
00459
00460 #if defined(VDEBUG)
00461 #   if !defined(APBS_NOINLINE)
00462 #       define APBS_NOINLINE 1
00463 #   endif
00464 #endif
00465
00466 #if !defined(APBS_NOINLINE)
00467
00471 #   define VINLINE_VACC
00472
00476 #   define VINLINE_VATOM
00477
00481 #   define VINLINE_VCSM
00482
00486 #   define VINLINE_VPBE
00487
00491 #   define VINLINE_VPEE
00492
00496 #   define VINLINE_VGREEN
00497
00501 #   define VINLINE_VFETK
00502
00506 #   define VINLINE_VPMG
00507
00512 #endif
00513
00514 /* Fortran name mangling */
00515 #if defined(VF77_UPPERCASE)
00516 #   if defined(VF77_NOUNDERSCORE)
00517 #       define VF77_MANGLE(name,NAME) NAME
00518 #   elif defined(VF77_ONEUNDERSCORE)
00519 #       define VF77_MANGLE(name,NAME) NAME ## _
00520 #   else
00521 #       define VF77_MANGLE(name,NAME) name
00522 #   endif
00523 #else
00524 #   if defined(VF77_NOUNDERSCORE)
00525 #       define VF77_MANGLE(name,NAME) name
00526 #   elif defined(VF77_ONEUNDERSCORE)
00527 #       define VF77_MANGLE(name,NAME) name ## _
00528 #   else
00532 #       define VF77_MANGLE(name,NAME) name
00533 #   endif
00534 #endif
00535
00536 /* Floating Point Error */
00537 #if defined(FLOAT_EPSILON)
00538 #     define VFLOOR(value) \
```

```
00539              ((floor(value) != floor(value + FLOAT_EPSILON)) ? \
00540              floor(value + FLOAT_EPSILON) : floor(value))
00541 #else
00547 #     define VFLOOR(value) floor(value)
00548 #endif
00549
00550 /* String embedding for ident */
00551 #if defined(HAVE_EMBED)
00556 #    define VEMBED(rctag) \
00557         VPRIVATE const char* rctag; \
00558         static void* use_rcsid=(0 ? &use_rcsid : (void**)&rcsid);
00559 #else
00564 #    define VEMBED(rctag)
00565 #endif /* if defined(HAVE_EMBED) */
00566
00567
00568
00570 #if !defined(_WIN32) || defined(__MINGW32__)
00571 #define PRINT_FUNC __PRETTY_FUNCTION__
00572 #define OS_SEP_STR "/"
00573 #define OS_SEP_CHAR '/'
00574 #else
00575 #define OS_SEP_STR "\\"
00576 #define OS_SEP_CHAR '\\'
00577 #define PRINT_FUNC __FUNCSIG__
00578 #define snprintf sprintf_s
00579 #endif
00580
00581 #ifdef VERGBOSE_DEBUG
00582 #define ANNOUNCE_FUNCTION                                       \
00583         do {                                                    \
00584             Vnm_prrint(2, "%s() [%s:%d]\n",                     \
00585                        PRINT_FUNC, __FILE__, __LINE__ ); \
00586         } while(0)
00587
00588 #define WARN_UNTESTED                                                   \
00589         do {                                                            \
00590             Vnm_print(2, "%s() [%s:%d]: Untested Translation!\n", \
00591                       __FUNCTION__, __FILE__, __LINE__);          \
00592         } while(0)
00593
00594 #define WARN_PARTTESTED                                                         \
00595         do{                                                                     \
00596             Vnm_print(2, "%s() [%s:%d]: Partially Tested Translation.\n", \
00597                       __FUNCTION__, __FILE__, __LINE__);                  \
00598         } while(0)
00599 #else
00600 #define ANNOUNCE_FUNCTION
00601 #define WARN_UNTESTED
00602 #define WARN_PARTTESTED
00603 #endif
00604
00605
00606
00607 /* Utility messages.  Print out messages with location information */
00608 #ifdef DEBUG
00609 #define VCHANNELEDMESSAGE0(channel, msg)                        \
00610         do {                                                    \
00611             Vnm_print(channel, "%s:%d [%s()]: MESSAGE:\n"     \
00612                       "    %s\n\n",                            \
00613                       __FILE__, __LINE__, __FUNCTION__, msg); \
00614         } while(0)
00615
00616 #define VCHANNELEDMESSAGE1(channel, msg, arg)                   \
00617         do {                                                    \
00618             char buff[1000];                                    \
00619             snprintf( buff, 1000, msg, arg );                   \
00620             Vnm_print(channel, "%s:%d [%s()]: MESSAGE:\n"      \
00621                       "    %s\n\n",                            \
00622                       __FILE__, __LINE__, __FUNCTION__, buff); \
00623         } while(0)
00624
00625 #define VCHANNELEDMESSAGE2(channel, msg, arg0, arg1)            \
00626         do {                                                    \
00627             char buff[1000];                                    \
00628             snprintf( buff, 1000, msg, arg0, arg1 );            \
00629             Vnm_print(channel, "%s:%d [%s()]: MESSAGE:\n"      \
00630                       "    %s\n\n",                            \
00631                       __FILE__, __LINE__, __FUNCTION__, buff); \
00632         } while(0)
00633
```

```
00634 #define VCHANNELEDMESSAGE3(channel, msg, arg0, arg1, arg2)     \
00635         do {                                                    \
00636             char buff[1000];                                    \
00637             snprintf(buff, 1000, msg, arg0, arg1, arg2);        \
00638             Vnm_print(channel, "%s:%d [%s()]: MESSAGE:\n"       \
00639                         "    %s\n\n",                           \
00640                       __FILE__, __LINE__, __FUNCTION__, buff); \
00641         } while(0)
00642
00643 #define VMESSAGE0(msg)                   VCHANNELEDMESSAGE0(2, msg)
00644 #define VMESSAGE1(msg, arg0)             VCHANNELEDMESSAGE1(2, msg, arg0)
00645 #define VMESSAGE2(msg, arg0, arg1)       VCHANNELEDMESSAGE2(2, msg, arg0, arg1)
00646 #define VMESSAGE3(msg, arg0, arg1, arg2) VCHANNELEDMESSAGE3(2, msg, arg0, arg1, arg2)
00647
00648 #define VERRMSG0(msg)                   VMESSAGE0(msg)
00649 #define VERRMSG1(msg, arg0)             VMESSAGE1(msg, arg0)
00650 #define VERRMSG2(msg, arg0, arg1)       VMESSAGE2(msg, arg0, arg1)
00651 #define VERRMSG3(msg, arg0, arg1, arg2) VMESSAGE3(msg, arg0, arg1, arg2)
00652 #else
00653 #define VCHANNELEDMESSAGE0(channel, msg)                        \
00654         do {                                                    \
00655             Vnm_print(channel, "%s: %s\n", __FUNCTION__, msg); \
00656         } while(0)
00657
00658 #define VCHANNELEDMESSAGE1(channel, msg, arg0)                   \
00659         do {                                                    \
00660             char buff[1000];                                    \
00661             snprintf( buff, 1000, msg, arg0 );                  \
00662             Vnm_print(channel, "%s: %s\n", __FUNCTION__, buff); \
00663         } while(0)
00664
00665 #define VCHANNELEDMESSAGE2(channel, msg, arg0, arg1)             \
00666         do {                                                    \
00667             char buff[1000];                                    \
00668             snprintf( buff, 1000, msg, arg0, arg1 );            \
00669             Vnm_print(channel, "%s: %s\n", __FUNCTION__, buff); \
00670         } while(0)
00671
00672 #define VCHANNELEDMESSAGE3(channel, msg, arg0, arg1, arg2)       \
00673         do {                                                    \
00674             char buff[1000];                                    \
00675             snprintf(buff, 1000, msg, arg0, arg1, arg2);        \
00676             Vnm_print(channel, "%s: %s\n", __FUNCTION__, buff);  \
00677         } while(0)
00678
00679 #define VMESSAGE0(msg)                   VCHANNELEDMESSAGE0(0, msg)
00680 #define VMESSAGE1(msg, arg0)             VCHANNELEDMESSAGE1(0, msg, arg0)
00681 #define VMESSAGE2(msg, arg0, arg1)       VCHANNELEDMESSAGE2(0, msg, arg0, arg1)
00682 #define VMESSAGE3(msg, arg0, arg1, arg2) VCHANNELEDMESSAGE3(0, msg, arg0, arg1, arg2)
00683
00684 #define VERRMSG0(msg)                   VCHANNELEDMESSAGE0(2, msg)
00685 #define VERRMSG1(msg, arg0)             VCHANNELEDMESSAGE1(2, msg, arg0)
00686 #define VERRMSG2(msg, arg0, arg1)       VCHANNELEDMESSAGE2(2, msg, arg0, arg1)
00687 #define VERRMSG3(msg, arg0, arg1, arg2)  VCHANNELEDMESSAGE3(2, msg, arg0, arg1, arg2)
00688 #endif
00689
00690
00691
00692 /* Utility assertions.  If they fail, they print out messages with possible
00693  * arguments and then abort
00694  * The do{...} while(0) simply enforces that a semicolon appears at the end
00695  */
00696 #ifdef DEBUG
00697 #define VASSERT_MSG0(cnd, msg)                                                                  \
00698     do {                                                                                        \
00699         if( (cnd) == 0 ) {                                                                      \
00700             Vnm_print(2, "%s:%d [%s()]: ERROR:\n"                                               \
00701                         "    Assertion Failed (%s): %s\n\n",                                    \
00702                       __FILE__, __LINE__, __FUNCTION__, #cnd, msg); \
00703             abort();                                                                            \
00704         }                                                                                       \
00705     } while(0)
00706
00707 #define VASSERT_MSG1(cnd, msg, arg)                                                              \
00708     do {                                                                                        \
00709         if( (cnd) == 0 ) {                                                                      \
00710             char buff[1000];                                                                    \
00711             snprintf( buff, 1000, msg, arg );                                                   \
00712             Vnm_print(2, "%s:%d [%s()]: ERROR:\n"                                               \
00713                         "    Assertion Failed (%s): %s\n\n",                                    \
00714                       __FILE__, __LINE__, __FUNCTION__, #cnd, buff); \
```

```
00715             abort();                                                    \
00716         }                                                               \
00717     } while(0)
00718
00719 #define VASSERT_MSG2(cnd, msg, arg0, arg1)                              \
00720     do {                                                                \
00721         if( (cnd) == 0 ) {                                              \
00722             char buff[1000];                                            \
00723             snprintf( buff, 1000, msg, arg0, arg1 );                    \
00724             Vnm_print(2, "%s:%d [%s()]: ERROR:\n"                       \
00725                          "    Assertion Failed (%s): %s\n\n",           \
00726                       __FILE__, __LINE__, __FUNCTION__, #cnd, buff);    \
00727             abort();                                                    \
00728         }                                                               \
00729     } while(0)
00730 #else
00731 #define VASSERT_MSG0(cnd, msg)                                                  \
00732     do {                                                                \
00733         if( (cnd) == 0 ) {                                              \
00734             Vnm_print(2, "%[%s()]: ERROR:\n"                            \
00735                          "    Assertion Failed (%s): %s\n\n",           \
00736                       __FUNCTION__, #cnd, msg);                         \
00737             abort();                                                    \
00738         }                                                               \
00739     } while(0)
00740
00741 #define VASSERT_MSG1(cnd, msg, arg)                                     \
00742     do {                                                                \
00743         if( (cnd) == 0 ) {                                              \
00744             char buff[1000];                                            \
00745             snprintf( buff, 1000, msg, arg );                           \
00746             Vnm_print(2, "[%s()]: ERROR:\n"                             \
00747                          "    Assertion Failed (%s): %s\n\n",           \
00748                       __FUNCTION__, #cnd, buff);                        \
00749             abort();                                                    \
00750         }                                                               \
00751     } while(0)
00752
00753 #define VASSERT_MSG2(cnd, msg, arg0, arg1)                              \
00754     do {                                                                \
00755         if( (cnd) == 0 ) {                                              \
00756             char buff[1000];                                            \
00757             snprintf( buff, 1000, msg, arg0, arg1 );                    \
00758             Vnm_print(2, "[%s()]: ERROR:\n"                             \
00759                          "    Assertion Failed (%s): %s\n\n",           \
00760                       __FUNCTION__, #cnd, buff);                        \
00761             abort();                                                    \
00762         }                                                               \
00763     } while(0)
00764 #endif
00765
00766
00767
00768 /* Utility warning.  Tests a condition and if it fails prints out a message
00769  * with optional arguments
00770  * The do{...} while(0) simply enforces that a semicolon at the end
00771  */
00772 #ifdef DEBUG
00773 #define VWARN_MSG0(cnd, msg)                                                \
00774         do {                                                                \
00775                 if( (cnd) == 0 ) {                                          \
00776                         Vnm_print(                                          \
00777                                 2,                                          \
00778                                 "%s:%d [%s()]: WARNING:\n"                  \
00779                                 "    Condition Failed (%s):\n    %s\n\n", \
00780                                 __FILE__,                                   \
00781                                 __LINE__,                                   \
00782                                 __FUNCTION__,                               \
00783                                 #cnd,                                       \
00784                                 msg                                         \
00785                                 );                                          \
00786                 }                                                           \
00787         } while(0)
00788
00789 #define VWARN_MSG1(cnd, msg, arg0)                                          \
00790         do {                                                                \
00791                 if( (cnd) == 0 ) {                                          \
00792                         char buff[1000];                                    \
00793                         snprintf(buff, 1000, msg, arg0);                    \
00794                         Vnm_print(                                          \
00795                                 2,                                          \
```

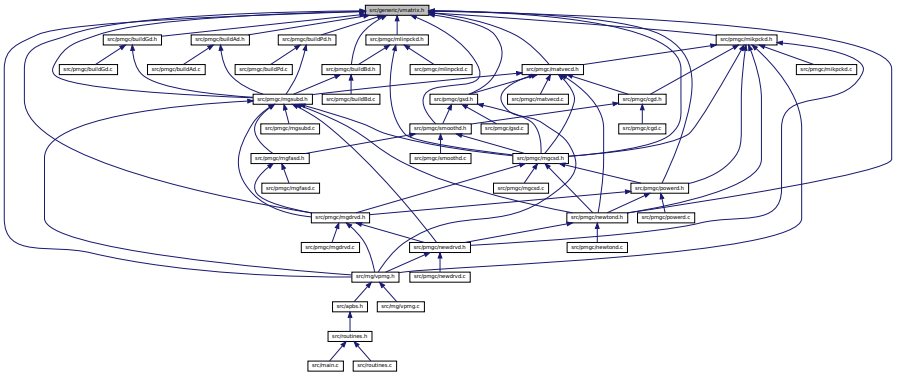
```
00796                                   "%s:%d [%s()]: WARNING:\n"                \
00797                                   "    Condition Failed (%s):\n    %s\n\n", \
00798                                   __FILE__,                                 \
00799                                   __LINE__,                                 \
00800                                   __FUNCTION__,                             \
00801                                   #cnd,                                     \
00802                                   buff                                      \
00803                                   );                                        \
00804                   }                                                         \
00805           } while(0)
00806
00807   #define VWARN_MSG2(cnd, msg, arg0, arg1)                                  \
00808           do {                                                              \
00809                   if( (cnd) == 0 ) {                                        \
00810                           char buff[1000];                                  \
00811                           snprintf(buff, 1000, msg, arg0, arg1);            \
00812                           Vnm_print(                                        \
00813                                   2,                                        \
00814                                   "%s:%d [%s()]: WARNING:\n"                \
00815                                   "    Condition Failed (%s):\n    %s\n\n", \
00816                                   __FILE__,                                 \
00817                                   __LINE__,                                 \
00818                                   __FUNCTION__,                             \
00819                                   #cnd,                                     \
00820                                   buff                                      \
00821                                   );                                        \
00822                   }                                                         \
00823           } while(0)
00824   #else
00825   #define VWARN_MSG0(cnd, msg)                                              \
00826           do {                                                              \
00827                   if( (cnd) == 0 ) {                                        \
00828                           Vnm_print(                                        \
00829                                   2,                                        \
00830                                   "[%s()]: WARNING:\n"                      \
00831                                   "    %s\n\n",                             \
00832                                   __FUNCTION__,                             \
00833                                   msg                                       \
00834                                   );                                        \
00835                   }                                                         \
00836           } while(0)
00837
00838   #define VWARN_MSG1(cnd, msg, arg0)                                        \
00839           do {                                                              \
00840                   if( (cnd) == 0 ) {                                        \
00841                           char buff[1000];                                  \
00842                           snprintf(buff, 1000, msg, arg0);                  \
00843                           Vnm_print(                                        \
00844                                   2,                                        \
00845                                   "[%s()]: WARNING:\n"                      \
00846                                   "    %s\n\n",                             \
00847                                   __FUNCTION__,                             \
00848                                   buff                                      \
00849                                   );                                        \
00850                   }                                                         \
00851           } while(0)
00852
00853   #define VWARN_MSG2(cnd, msg, arg0, arg1)                                  \
00854           do {                                                              \
00855                   if( (cnd) == 0 ) {                                        \
00856                           char buff[1000];                                  \
00857                           snprintf(buff, 1000, msg, arg0, arg1);            \
00858                           Vnm_print(                                        \
00859                                   2,                                        \
00860                                   "[%s()]: WARNING:\n"                      \
00861                                   "    %s\n\n",                             \
00862                                   __FUNCTION__,                             \
00863                                   buff                                      \
00864                                   );                                        \
00865                   }                                                         \
00866           } while(0)
00867   #endif
00868
00869   /* Utility Abort.  Prints a message with optional arugments and aborts */
00870   #ifdef DEBUG
00871   #define VABORT_MSG0(msg)                                                  \
00872           do {                                                              \
00873               Vnm_print(2, "%s:%d [%s()]: ABORTING:\n"                      \
00874                            "    %s\n\n",                                    \
00875                         __FILE__, __LINE__, __FUNCTION__, msg);            \
00876               abort();                                                      \
```

```
00877         } while(0)
00878
00879 #define VABORT_MSG1(msg, arg)                                       \
00880         do {                                                        \
00881             char buff[1000];                                        \
00882             snprintf( buff, 1000, msg, arg );                       \
00883             Vnm_print(2, "%s:%d [%s()]: ABORTING:\n"                \
00884                       "    %s\n\n",                                 \
00885                       __FILE__, __LINE__, __FUNCTION__, buff); \
00886             abort();                                                \
00887         } while(0)
00888
00889 #define VABORT_MSG2(msg, arg0, arg1)                                \
00890         do {                                                        \
00891             char buff[1000];                                        \
00892             snprintf( buff, 1000, msg, arg0, arg1);                 \
00893             Vnm_print(2, "%s:%d [%s()]: ABORTING:\n"                \
00894                       "    %s\n\n",                                 \
00895                       __FILE__, __LINE__, __FUNCTION__, buff); \
00896             abort();                                                \
00897         } while(0)
00898 #else
00899 #define VABORT_MSG0(msg)                          \
00900         do {                                      \
00901             Vnm_print(2, "%[%s()]: ABORTING:\n" \
00902                       "    %s\n\n",              \
00903                       __FUNCTION__, msg);       \
00904             abort();                              \
00905         } while(0)
00906
00907 #define VABORT_MSG1(msg, arg)                    \
00908         do {                                     \
00909             char buff[1000];                     \
00910             snprintf( buff, 1000, msg, arg );  \
00911             Vnm_print(2, "[%s()]: ABORTING:\n" \
00912                       "    %s\n\n",             \
00913                       __FUNCTION__, buff);     \
00914             abort();                             \
00915         } while(0)
00916
00917 #define VABORT_MSG2(msg, arg0, arg1)                \
00918         do {                                        \
00919             char buff[1000];                        \
00920             snprintf( buff, 1000, msg, arg0, arg1); \
00921             Vnm_print(2, "[%s()]: ABORTING:\n"      \
00922                       "    %s\n\n",                \
00923                       __FUNCTION__, buff);          \
00924             abort();                                \
00925         } while(0)
00926 #endif
00927
00928
00929
00930 /* Utility expression printers.  Print the expression and its value */
00931 #ifdef DEBUG
00932 #define PRINT_INT(expr)                                                 \
00933     do {                                                                \
00934         Vnm_print(2, "%s:%d [%s()]: %s == %d\n",                        \
00935                   __FILE__, __LINE__, __FUNCTION__, #expr, expr);   \
00936     } while(0)
00937
00938 #define PRINT_DBL(expr)                                                \
00939     do {                                                               \
00940         Vnm_print(2, "%s:%d [%s()]: %s == %f\n",                       \
00941                   __FILE__, __LINE__, __FUNCTION__, #expr, expr); \
00942     } while(0)
00943 #else
00944 #define PRINT_INT(expr)
00945 #define PRINT_DBL(expr)
00946 #endif
00947
00948 #define VMALLOC(vmem, n, type) ((type*)Vmem_malloc(vmem, n, sizeof(type)))
00949
00950 #define VFREE(vmem, n, type, ptr) (Vmem_free(vmem, n, sizeof(type), (void **)&(ptr)))
00951
00952 #define VFILL(vec, n, val)                          \
00953     do {                                            \
00954         int fill_idx;                               \
00955         for (fill_idx = 0; fill_idx < n; fill_idx++) \
00956             vec[fill_idx] = val;                    \
00957     } while(0)
```

```
00958
00959 #define VCOPY(srcvec, dstvec, i, n) \
00960     do {                            \
00961         for (i = 0; i < n; i++)     \
00962             dstvec[i] = srcvec[i];  \
00963     } while(0)
00964
00965
00966 char* wrap_text( char* str, int right_margin, int left_padding );
00967
00968 #define VAT(array, i) ((array)[(i) - 1])
00969 #define RAT(array, i) ((array) + i - 1)
00970
00971 #endif /* #ifndef _VAPBSHAL_H_ */
```

9.73 src/generic/vmatrix.h File Reference

Contains inclusions for matrix data wrappers.
This graph shows which files directly or indirectly include this file:



Macros

- #define **MAT2**(mat, dx, dy)
- #define **RAT2**(mat, x, y) &VAT2(mat, x, y)
- #define **VAT2**(mat, x, y) mat[(y - 1) ∗ dx_##mat + (x - 1)]
- #define **MAT3**(mat, dx, dy, dz)
- #define **RAT3**(mat, x, y, z) &VAT3(mat, x, y, z)
- #define **VAT3**(mat, x, y, z)

9.73.1 Detailed Description

Contains inclusions for matrix data wrappers.

Version

Author

Tucker A. Beck

Attention

Definition in file vmatrix.h.

9.73.2 Macro Definition Documentation

9.73.2.1 MAT2

```
#define MAT2(
              mat,
              dx,
              dy )
```

Value:

```
int dx_##mat = dx;    \
int dy_##mat = dy
```

Definition at line 64 of file vmatrix.h.

9.73.2.2 MAT3

```
#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.

9.73.2.3 VAT3

```
#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.

9.74 vmatrix.h

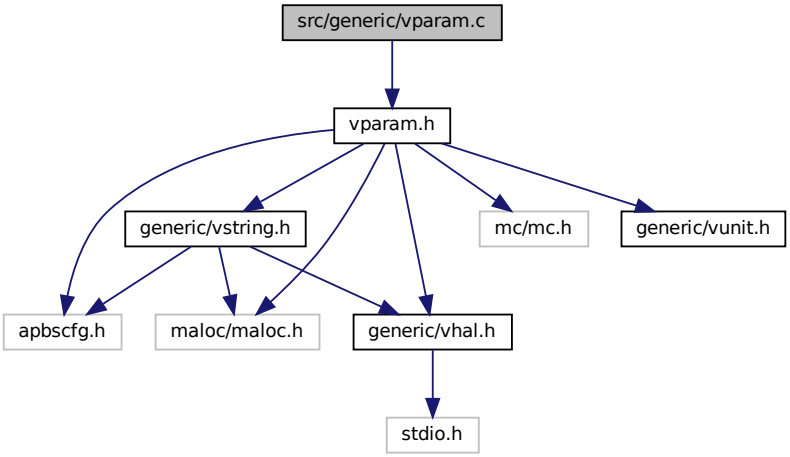
```
00001
00061 #ifndef _VMATRIX_H_
00062 #define _VMATRIX_H_
00063
00064 #define MAT2(mat, dx, dy) \
00065     int dx_##mat = dx;    \
00066     int dy_##mat = dy
00067
00068 #define RAT2(mat, x, y) \
00069     &VAT2(mat, x, y)
00070
00071 #define VAT2(mat, x, y) \
00072     mat[(y - 1) * dx_##mat + (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_ */
```

9.75 src/generic/vparam.c File Reference

Class Vparam methods.

```
#include "vparam.h"
```

Include dependency graph for vparam.c:



Functions

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

  Read a single line of the flat file database.
- VPRIVATE int readXMLFileAtom (Vio *sock, Vparam_AtomData *atom)

  Read atom information from an XML file.
- VPUBLIC unsigned long int Vparam_memChk (Vparam *thee)

  Get number of bytes in this object and its members.
- VPUBLIC Vparam_AtomData * Vparam_AtomData_ctor ()

  Construct the object.
- VPUBLIC int Vparam_AtomData_ctor2 (Vparam_AtomData *thee)

  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

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) 2010-2020 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) 2002-2020, 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 "vparam.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061
00065 VPRIVATE char *MCwhiteChars = " =,;\t\n\r";
00066
00071 VPRIVATE char *MCcommChars  = "#%";
00072
00077 VPRIVATE char *MCxmlwhiteChars = " =,;\t\n\r<>";
00078
00087 VPRIVATE int readFlatFileLine(Vio *sock, Vparam_AtomData *atom);
00088
00097 VPRIVATE int readXMLFileAtom(Vio *sock, Vparam_AtomData *atom);
00098
00099
```

```
00100 #if !defined(VINLINE_VPARAM)
00101
00102 VPUBLIC unsigned long int Vparam_memChk(Vparam *thee) {
00103     if (thee == VNULL) return 0;
00104     return Vmem_bytes(thee->vmem);
00105 }
00106
00107 #endif /* if !defined(VINLINE_VPARAM) */
00108
00109 VPUBLIC Vparam_AtomData* Vparam_AtomData_ctor() {
00110
00111     Vparam_AtomData *thee = VNULL;
00112
00113     /* Set up the structure */
00114     thee = (Vparam_AtomData*)Vmem_malloc(VNULL, 1, sizeof(Vparam_AtomData) );
00115     VASSERT(thee != VNULL);
00116     VASSERT(Vparam_AtomData_ctor2(thee));
00117
00118     return thee;
00119 }
00120
00121 VPUBLIC int Vparam_AtomData_ctor2(Vparam_AtomData *thee) { return 1; }
00122
00123 VPUBLIC void Vparam_AtomData_dtor(Vparam_AtomData **thee) {
00124
00125     if ((*thee) != VNULL) {
00126         Vparam_AtomData_dtor2(*thee);
00127         Vmem_free(VNULL, 1, sizeof(Vparam_AtomData), (void **)thee);
00128         (*thee) = VNULL;
00129     }
00130
00131 }
00132
00133 VPUBLIC void Vparam_AtomData_dtor2(Vparam_AtomData *thee) { ; }
00134
00135 VPUBLIC Vparam_ResData* Vparam_ResData_ctor(Vmem *mem) {
00136
00137     Vparam_ResData *thee = VNULL;
00138
00139     /* Set up the structure */
00140     thee = (Vparam_ResData*)Vmem_malloc(mem, 1, sizeof(Vparam_ResData) );
00141     VASSERT(thee != VNULL);
00142     VASSERT(Vparam_ResData_ctor2(thee, mem));
00143
00144     return thee;
00145 }
00146
00147 VPUBLIC int Vparam_ResData_ctor2(Vparam_ResData *thee, Vmem *mem) {
00148
00149     if (thee == VNULL) {
00150         Vnm_print(2, "Vparam_ResData_ctor2:  Got VNULL thee!\n");
00151         return 0;
00152     }
00153     thee->vmem = mem;
00154     thee->nAtomData = 0;
00155     thee->atomData = VNULL;
00156
00157     return 1;
00158 }
00159
00160 VPUBLIC void Vparam_ResData_dtor(Vparam_ResData **thee) {
00161
00162     if ((*thee) != VNULL) {
00163         Vparam_ResData_dtor2(*thee);
00164         Vmem_free((*thee)->vmem, 1, sizeof(Vparam_ResData), (void **)thee);
00165         (*thee) = VNULL;
00166     }
00167
00168 }
00169
00170 VPUBLIC void Vparam_ResData_dtor2(Vparam_ResData *thee) {
00171
00172     if (thee == VNULL) return;
00173     if (thee->nAtomData > 0) {
00174         Vmem_free(thee->vmem, thee->nAtomData, sizeof(Vparam_AtomData),
00175           (void **)&(thee->atomData));
00176     }
00177     thee->nAtomData = 0;
00178     thee->atomData = VNULL;
00179 }
00180
```

```
00181 VPUBLIC Vparam* Vparam_ctor() {
00182
00183     Vparam *thee = VNULL;
00184
00185     /* Set up the structure */
00186     thee = (Vparam*)Vmem_malloc(VNULL, 1, sizeof(Vparam) );
00187     VASSERT(thee != VNULL);
00188     VASSERT(Vparam_ctor2(thee));
00189
00190     return thee;
00191 }
00192
00193 VPUBLIC int Vparam_ctor2(Vparam *thee) {
00194
00195     if (thee == VNULL) {
00196         Vnm_print(2, "Vparam_ctor2: got VNULL thee!\n");
00197         return 0;
00198     }
00199
00200     thee->vmem = VNULL;
00201     thee->vmem = Vmem_ctor("APBS:VPARAM");
00202     if (thee->vmem == VNULL) {
00203         Vnm_print(2, "Vparam_ctor2:  failed to init Vmem!\n");
00204         return 0;
00205     }
00206
00207     thee->nResData = 0;
00208     thee->resData = VNULL;
00209
00210     return 1;
00211 }
00212
00213 VPUBLIC void Vparam_dtor(Vparam **thee) {
00214
00215     if ((*thee) != VNULL) {
00216         Vparam_dtor2(*thee);
00217         Vmem_free(VNULL, 1, sizeof(Vparam), (void **)thee);
00218         (*thee) = VNULL;
00219     }
00220
00221 }
00222
00223 VPUBLIC void Vparam_dtor2(Vparam *thee) {
00224
00225     int i;
00226
00227     if (thee == VNULL) return;
00228
00229     /* Destroy the residue data */
00230     for (i=0; i<thee->nResData; i++) Vparam_ResData_dtor2(&(thee->resData[i]));
00231     if (thee->nResData > 0) Vmem_free(thee->vmem, thee->nResData,
00232       sizeof(Vparam_ResData), (void **)&(thee->resData));
00233     thee->nResData = 0;
00234     thee->resData = VNULL;
00235
00236     if (thee->vmem != VNULL) Vmem_dtor(&(thee->vmem));
00237     thee->vmem = VNULL;
00238
00239 }
00240
00241 VPUBLIC Vparam_ResData* Vparam_getResData(Vparam *thee,
00242   char resName[VMAX_ARGLEN]) {
00243
00244     int i;
00245     Vparam_ResData *res = VNULL;
00246
00247     VASSERT(thee != VNULL);
00248
00249     if ((thee->nResData == 0) || (thee->resData == VNULL)) {
00250         res = VNULL;
00251         return res;
00252     }
00253
00254     /* Look for the matching residue */
00255     for (i=0; i<thee->nResData; i++) {
00256         res = &(thee->resData[i]);
00257         if (Vstring_strcasecmp(resName, res->name) == 0) return res;
00258
00259     }
00260
00261     /* Didn't find a matching residue */
```

```
00262     res = VNULL;
00263     Vnm_print(2, "Vparam_getResData:  unable to find res=%s\n", resName);
00264     return res;
00265 }
00266
00267 VPUBLIC Vparam_AtomData* Vparam_getAtomData(Vparam *thee,
00268   char resName[VMAX_ARGLEN], char atomName[VMAX_ARGLEN]) {
00269
00270     int i;
00271     Vparam_ResData *res = VNULL;
00272     Vparam_AtomData *atom = VNULL;
00273
00274     VASSERT(thee != VNULL);
00275
00276     if ((thee->nResData == 0) || (thee->resData == VNULL)) {
00277         atom = VNULL;
00278         return atom;
00279     }
00280
00281     /* Look for the matching residue */
00282     res = Vparam_getResData(thee, resName);
00283     if (res == VNULL) {
00284         atom = VNULL;
00285         Vnm_print(2, "Vparam_getAtomData:  Unable to find residue %s!\n", resName);
00286         return atom;
00287     }
00288     for (i=0; i<res->nAtomData; i++) {
00289         atom = &(res->atomData[i]);
00290         if (atom == VNULL) {
00291             Vnm_print(2, "Vparam_getAtomData:  got NULL atom!\n");
00292             return VNULL;
00293         }
00294         if (Vstring_strcasecmp(atomName, atom->atomName) == 0) {
00295             return atom;
00296         }
00297     }
00298
00299     /* Didn't find a matching atom/residue */
00300     atom = VNULL;
00301     Vnm_print(2, "Vparam_getAtomData:  unable to find atom '%s', res '%s'\n",
00302       atomName, resName);
00303     return atom;
00304 }
00305
00306 VPUBLIC int Vparam_readXMLFile(Vparam *thee, const char *iodev,
00307   const char *iofmt, const char *thost, const char *fname) {
00308
00309     int i, ires, natoms, nalloc, ralloc;
00310     Vparam_AtomData *atoms = VNULL;
00311     Vparam_AtomData *tatoms = VNULL;
00312     Vparam_AtomData *atom = VNULL;
00313     Vparam_ResData *res = VNULL;
00314     Vparam_ResData *residues = VNULL;
00315     Vparam_ResData *tresidues = VNULL;
00316     Vio *sock = VNULL;
00317     char currResName[VMAX_ARGLEN];
00318     char tok[VMAX_ARGLEN];
00319     char endtag[VMAX_ARGLEN];
00320
00321     VASSERT(thee != VNULL);
00322
00323     /* Setup communication */
00324     sock = Vio_ctor(iodev,iofmt,thost,fname,"r");
00325     if (sock == VNULL) {
00326         Vnm_print(2, "Vparam_readXMLFile: Problem opening virtual socket %s\n",
00327           fname);
00328         return 0;
00329     }
00330     if (Vio_accept(sock, 0) < 0) {
00331         Vnm_print(2, "Vparam_readXMLFile: Problem accepting virtual socket %s\n",
00332           fname);
00333         return 0;
00334     }
00335     Vio_setWhiteChars(sock, MCxmlwhiteChars);
00336     Vio_setCommChars(sock, MCcommChars);
00337
00338     /* Clear existing parameters */
00339     if (thee->nResData > 0) {
00340         Vnm_print(2, "WARNING -- CLEARING PARAMETER DATABASE!\n");
00341         for (i=0; i<thee->nResData; i++) {
00342             Vparam_ResData_dtor2(&(thee->resData[i]));
```

```
00343         }
00344         Vmem_free(thee->vmem, thee->nResData,
00345           sizeof(Vparam_ResData), (void **)&(thee->resData));
00346     }
00347
00348     strcpy(endtag,"/");
00349
00350     /* Set up temporary residue list */
00351
00352     ralloc = 50;
00353     residues = (Vparam_ResData*)Vmem_malloc(thee->vmem, ralloc, sizeof(Vparam_ResData));
00354
00355     /* Read until we run out of entries, allocating space as needed */
00356     while (1) {
00357
00358         VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00359
00360         /* The first token should be the start tag */
00361
00362         if (Vstring_strcasecmp(endtag, "/") == 0) strcat(endtag, tok);
00363
00364         if (Vstring_strcasecmp(tok, "residue") == 0) {
00365           if (thee->nResData >= ralloc) {
00366                 tresidues = (Vparam_ResData*)Vmem_malloc(thee->vmem, 2*ralloc, sizeof(Vparam_ResData));
00367                 VASSERT(tresidues != VNULL);
00368                 for (i=0; i<thee->nResData; i++) {
00369                     Vparam_ResData_copyTo(&(residues[i]), &(tresidues[i]));
00370                 }
00371                 Vmem_free(thee->vmem, ralloc, sizeof(Vparam_ResData),
00372                           (void **)&(residues));
00373                 residues = tresidues;
00374                 tresidues = VNULL;
00375                 ralloc = 2*ralloc;
00376             }
00377
00378           /* Initial space for this residue's atoms */
00379           nalloc = 20;
00380           natoms = 0;
00381           atoms = (Vparam_AtomData*)Vmem_malloc(thee->vmem, nalloc, sizeof(Vparam_AtomData));
00382
00383         } else if (Vstring_strcasecmp(tok, "name") == 0) {
00384             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);  /* value */
00385             strcpy(currResName, tok);
00386             VJMPERR1(Vio_scanf(sock, "%s", tok) == 1); /* </name> */
00387         } else if (Vstring_strcasecmp(tok, "atom") == 0) {
00388             if (natoms >= nalloc) {
00389                 tatoms = (Vparam_AtomData*)Vmem_malloc(thee->vmem, 2*nalloc, sizeof(Vparam_AtomData));
00390                 VASSERT(tatoms != VNULL);
00391                 for (i=0; i<natoms; i++) {
00392                     Vparam_AtomData_copyTo(&(atoms[i]), &(tatoms[i]));
00393                 }
00394                 Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData),
00395                           (void **)&(atoms));
00396                 atoms = tatoms;
00397                 tatoms = VNULL;
00398                 nalloc = 2*nalloc;
00399             }
00400             atom = &(atoms[natoms]);
00401             if (!readXMLFileAtom(sock, atom)) break;
00402             natoms++;
00403
00404         } else if (Vstring_strcasecmp(tok, "/residue") == 0) {
00405
00406           res = &(residues[thee->nResData]);
00407           Vparam_ResData_ctor2(res, thee->vmem);
00408           res->atomData = (Vparam_AtomData*)Vmem_malloc(thee->vmem, natoms,
00409                                       sizeof(Vparam_AtomData));
00410           res->nAtomData = natoms;
00411           strcpy(res->name, currResName);
00412           for (i=0; i<natoms; i++) {
00413               strcpy(atoms[i].resName, currResName);
00414               Vparam_AtomData_copyTo(&(atoms[i]), &(res->atomData[i]));
00415           }
00416           Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData), (void **)&(atoms));
00417           (thee->nResData)++;
00418
00419         } else if (Vstring_strcasecmp(tok, endtag) == 0) break;
00420     }
00421
00422     /* Initialize and copy the residues into the Vparam object */
00423
```

```
00424     thee->resData = (Vparam_ResData*)Vmem_malloc(thee->vmem, thee->nResData,
00425                                 sizeof(Vparam_ResData));
00426     for (ires=0; ires<thee->nResData; ires++) {
00427         Vparam_ResData_copyTo(&(residues[ires]), &(thee->resData[ires]));
00428     }
00429
00430     /* Destroy temporary atom space */
00431     Vmem_free(thee->vmem, ralloc, sizeof(Vparam_ResData), (void **)&(residues));
00432
00433     /* Shut down communication */
00434     Vio_acceptFree(sock);
00435     Vio_dtor(&sock);
00436
00437     return 1;
00438
00439 VERROR1:
00440     Vnm_print(2, "Vparam_readXMLFile: Got unexpected EOF reading parameter file!\n");
00441     return 0;
00442
00443 }
00444
00445 VPUBLIC int Vparam_readFlatFile(Vparam *thee, const char *iodev,
00446   const char *iofmt, const char *thost, const char *fname) {
00447
00448     int i, iatom, jatom, ires, natoms, nalloc;
00449     Vparam_AtomData *atoms = VNULL;
00450     Vparam_AtomData *tatoms = VNULL;
00451     Vparam_AtomData *atom = VNULL;
00452     Vparam_ResData *res = VNULL;
00453     Vio *sock = VNULL;
00454     char currResName[VMAX_ARGLEN];
00455
00456     VASSERT(thee != VNULL);
00457
00458     /* Setup communication */
00459     sock = Vio_ctor(iodev,iofmt,thost,fname,"r");
00460     if (sock == VNULL) {
00461         Vnm_print(2, "Vparam_readFlatFile: Problem opening virtual socket %s\n",
00462           fname);
00463         return 0;
00464     }
00465     if (Vio_accept(sock, 0) < 0) {
00466         Vnm_print(2, "Vparam_readFlatFile: Problem accepting virtual socket %s\n",
00467           fname);
00468         return 0;
00469     }
00470     Vio_setWhiteChars(sock, MCwhiteChars);
00471     Vio_setCommChars(sock, MCcommChars);
00472
00473     /* Clear existing parameters */
00474     if (thee->nResData > 0) {
00475         Vnm_print(2, "WARNING -- CLEARING PARAMETER DATABASE!\n");
00476         for (i=0; i<thee->nResData; i++) {
00477             Vparam_ResData_dtor2(&(thee->resData[i]));
00478         }
00479         Vmem_free(thee->vmem, thee->nResData,
00480           sizeof(Vparam_ResData), (void **)&(thee->resData));
00481     }
00482
00483     /* Initial space for atoms */
00484     nalloc = 200;
00485     natoms = 0;
00486     atoms = (Vparam_AtomData*)Vmem_malloc(thee->vmem, nalloc, sizeof(Vparam_AtomData));
00487
00488     /* Read until we run out of entries, allocating space as needed */
00489     while (1) {
00490         if (natoms >= nalloc) {
00491             tatoms = (Vparam_AtomData*)Vmem_malloc(thee->vmem, 2*nalloc, sizeof(Vparam_AtomData));
00492             VASSERT(tatoms != VNULL);
00493             for (i=0; i<natoms; i++) {
00494                 Vparam_AtomData_copyTo(&(atoms[i]), &(tatoms[i]));
00495             }
00496             Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData),
00497               (void **)&(atoms));
00498             atoms = tatoms;
00499             tatoms = VNULL;
00500             nalloc = 2*nalloc;
00501         }
00502         atom = &(atoms[natoms]);
00503         if (!readFlatFileLine(sock, atom)) break;
00504         natoms++;
```

```
00505     }
00506     if (natoms == 0) return 0;
00507
00508     /* Count the number of residues */
00509     thee->nResData = 1;
00510     strcpy(currResName, atoms[0].resName);
00511     for (i=1; i<natoms; i++) {
00512         if (Vstring_strcasecmp(atoms[i].resName, currResName) != 0) {
00513             strcpy(currResName, atoms[i].resName);
00514             (thee->nResData)++;
00515         }
00516     }
00517
00518     /* Create the residues */
00519     thee->resData = (Vparam_ResData*)Vmem_malloc(thee->vmem, thee->nResData,
00520       sizeof(Vparam_ResData));
00521     VASSERT(thee->resData != VNULL);
00522     for (i=0; i<(thee->nResData); i++) {
00523         res = &(thee->resData[i]);
00524         Vparam_ResData_ctor2(res, thee->vmem);
00525     }
00526
00527     /* Count the number of atoms per residue */
00528     ires = 0;
00529     res = &(thee->resData[ires]);
00530     res->nAtomData = 1;
00531     strcpy(res->name, atoms[0].resName);
00532     for (i=1; i<natoms; i++) {
00533         if (Vstring_strcasecmp(atoms[i].resName, res->name) != 0) {
00534             (ires)++;
00535             res = &(thee->resData[ires]);
00536             res->nAtomData = 1;
00537             strcpy(res->name, atoms[i].resName);
00538         } else (res->nAtomData)++;
00539     }
00540
00541     /* Allocate per-residue space for atoms */
00542     for (ires=0; ires<thee->nResData; ires++) {
00543         res = &(thee->resData[ires]);
00544         res->atomData = (Vparam_AtomData*)Vmem_malloc(thee->vmem, res->nAtomData,
00545           sizeof(Vparam_AtomData));
00546     }
00547
00548     /* Copy atoms into residues */
00549     iatom = 0;
00550     Vparam_AtomData_copyTo(&(atoms[0]), &(res->atomData[iatom]));
00551     for (ires=0; ires<thee->nResData; ires++) {
00552         res = &(thee->resData[ires]);
00553         for (jatom=0; jatom<res->nAtomData; jatom++) {
00554             Vparam_AtomData_copyTo(&(atoms[iatom]), &(res->atomData[jatom]));
00555             iatom++;
00556         }
00557     }
00558
00559
00560     /* Shut down communication */
00561     Vio_acceptFree(sock);
00562     Vio_dtor(&sock);
00563
00564     /* Destroy temporary atom space */
00565     Vmem_free(thee->vmem, nalloc, sizeof(Vparam_AtomData), (void **)&(atoms));
00566
00567     return 1;
00568
00569 }
00570
00571 VEXTERNC void Vparam_AtomData_copyTo(Vparam_AtomData *thee,
00572   Vparam_AtomData *dest) {
00573
00574     VASSERT(thee != VNULL);
00575     VASSERT(dest != VNULL);
00576
00577     strcpy(dest->atomName, thee->atomName);
00578     strcpy(dest->resName, thee->resName);
00579     dest->charge = thee->charge;
00580     dest->radius = thee->radius;
00581     dest->epsilon = thee->epsilon;
00582
00583 }
00584
00585 VEXTERNC void Vparam_ResData_copyTo(Vparam_ResData *thee,
```

```
00586   Vparam_ResData *dest) {
00587
00588     int i;
00589
00590     VASSERT(thee != VNULL);
00591     VASSERT(dest != VNULL);
00592
00593     strcpy(dest->name, thee->name);
00594     dest->vmem = thee->vmem;
00595     dest->nAtomData = thee->nAtomData;
00596
00597     dest->atomData = (Vparam_AtomData*)Vmem_malloc(thee->vmem, dest->nAtomData,
00598                                 sizeof(Vparam_AtomData));
00599
00600     for (i=0; i<dest->nAtomData; i++) {
00601       Vparam_AtomData_copyTo(&(thee->atomData[i]), &(dest->atomData[i]));
00602     }
00603     Vmem_free(thee->vmem, thee->nAtomData, sizeof(Vparam_AtomData),
00604               (void **)&(thee->atomData));
00605 }
00606
00607 VEXTERNC void Vparam_AtomData_copyFrom(Vparam_AtomData *thee,
00608   Vparam_AtomData *src) {  Vparam_AtomData_copyTo(src, thee); }
00609
00610 VPRIVATE int readXMLFileAtom(Vio *sock, Vparam_AtomData *atom) {
00611
00612     double dtmp;
00613     char tok[VMAX_BUFSIZE];
00614     int chgflag, radflag, nameflag;
00615
00616     VASSERT(atom != VNULL);
00617
00618     if (Vio_scanf(sock, "%s", tok) != 1) return 0;
00619
00620     chgflag = 0;
00621     radflag = 0;
00622     nameflag = 0;
00623
00624     while (1)
00625       {
00626           if (Vstring_strcasecmp(tok, "name") == 0) {
00627               VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00628               if (strlen(tok) > VMAX_ARGLEN) {
00629                   Vnm_print(2, "Vparam_readXMLFileAtom:  string (%s) too long \
00630 (%d)!\n", tok, strlen(tok));
00631                   return 0;
00632               }
00633               nameflag = 1;
00634               strcpy(atom->atomName, tok);
00635           } else if (Vstring_strcasecmp(tok, "charge") == 0) {
00636               VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00637               if (sscanf(tok, "%lf", &dtmp) != 1) {
00638                   Vnm_print(2, "Vparam_readXMLFileAtom:  Unexpected token (%s) while \
00639 parsing charge!\n", tok);
00640                   return 0;
00641               }
00642               chgflag = 1;
00643               atom->charge = dtmp;
00644           }  else if (Vstring_strcasecmp(tok, "radius") == 0) {
00645               VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00646               if (sscanf(tok, "%lf", &dtmp) != 1) {
00647                   Vnm_print(2, "Vparam_readXMLFileAtom:  Unexpected token (%s) while \
00648 parsing radius!\n", tok);
00649                   return 0;
00650               }
00651               radflag = 1;
00652               atom->radius = dtmp;
00653           }  else if (Vstring_strcasecmp(tok, "epsilon") == 0) {
00654               VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00655               if (sscanf(tok, "%lf", &dtmp) != 1) {
00656                   Vnm_print(2, "Vparam_readXMLFileAtom:  Unexpected token (%s) while \
00657 parsing epsilon!\n", tok);
00658                   return 0;
00659               }
00660               atom->epsilon = dtmp;
00661           } else if ((Vstring_strcasecmp(tok, "/atom") == 0) ||
00662                      (Vstring_strcasecmp(tok, "atom") == 0)){
00663                 if (chgflag && radflag && nameflag) return 1;
00664                 else if (!chgflag) {
00665                   Vnm_print(2, "Vparam_readXMLFileAtom: Reached end of atom without \
00666 setting the charge!\n");
```

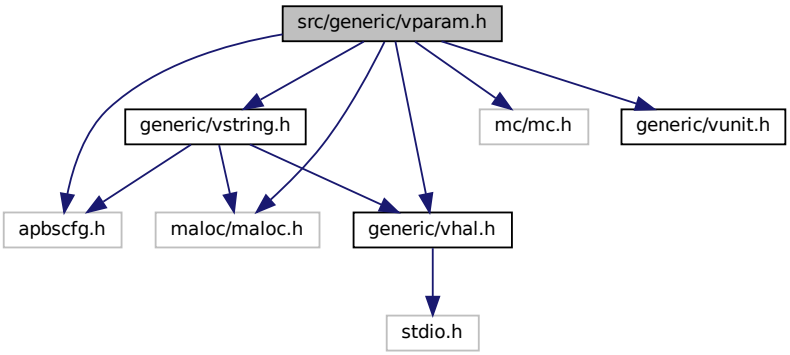
```
00667                   return 0;
00668                 } else if (!radflag) {
00669                   Vnm_print(2, "Vparam_readXMLFileAtom: Reached end of atom without \
00670 setting the radius!\n");
00671                   return 0;
00672                 } else if (!nameflag) {
00673                   Vnm_print(2, "Vparam_readXMLFileAtom: Reached end of atom without \
00674 setting the name!\n");
00675                   return 0;
00676                 }
00677           }
00678           VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00679       }
00680
00681     /* If we get here something wrong has happened */
00682
00683     VJMPERR1(1);
00684
00685 VERROR1:
00686     Vnm_print(2, "Vparam_readXMLFileAtom: Got unexpected EOF reading parameter file!\n");
00687     return 0;
00688
00689 }
00690
00691 VPRIVATE int readFlatFileLine(Vio *sock, Vparam_AtomData *atom) {
00692
00693     double dtmp;
00694     char tok[VMAX_BUFSIZE];
00695
00696     VASSERT(atom != VNULL);
00697
00698     if (Vio_scanf(sock, "%s", tok) != 1) return 0;
00699     if (strlen(tok) > VMAX_ARGLEN) {
00700         Vnm_print(2, "Vparam_readFlatFile:  string (%s) too long (%d)!\n",
00701           tok, strlen(tok));
00702         return 0;
00703     }
00704     strcpy(atom->resName, tok);
00705     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00706     if (strlen(tok) > VMAX_ARGLEN) {
00707         Vnm_print(2, "Vparam_readFlatFile:  string (%s) too long (%d)!\n",
00708           tok, strlen(tok));
00709         return 0;
00710     }
00711     strcpy(atom->atomName, tok);
00712     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00713     if (sscanf(tok, "%lf", &dtmp) != 1) {
00714         Vnm_print(2, "Vparam_readFlatFile:  Unexpected token (%s) while \
00715 parsing charge!\n", tok);
00716         return 0;
00717     }
00718     atom->charge = dtmp;
00719     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00720     if (sscanf(tok, "%lf", &dtmp) != 1) {
00721         Vnm_print(2, "Vparam_readFlatFile:  Unexpected token (%s) while \
00722 parsing radius!\n", tok);
00723         return 0;
00724     }
00725     atom->radius = dtmp;
00726     VJMPERR1(Vio_scanf(sock, "%s", tok) == 1);
00727     if (sscanf(tok, "%lf", &dtmp) != 1) {
00728         Vnm_print(2, "Vparam_readFlatFile:  Unexpected token (%s) while \
00729 parsing radius!\n", tok);
00730         return 0;
00731     }
00732     atom->epsilon = dtmp;
00733
00734     return 1;
00735
00736 VERROR1:
00737     Vnm_print(2, "Vparam_readFlatFile: Got unexpected EOF reading parameter file!\n");
00738     return 0;
00739 }
```

9.77 src/generic/vparam.h File Reference

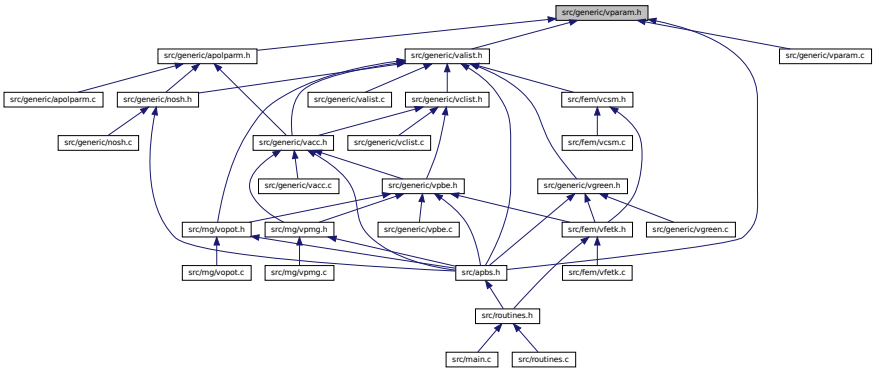
Contains declarations for class Vparam.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "mc/mc.h"
#include "generic/vhal.h"
#include "generic/vunit.h"
#include "generic/vstring.h"
```

Include dependency graph for vparam.h:



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



Data Structures

- struct sVparam_AtomData

  AtomData sub-class; stores atom data.

- struct Vparam_ResData

  ResData sub-class; stores residue data.

- struct Vparam

  Reads and assigns charge/radii parameters.

Typedefs

- typedef struct sVparam_AtomData Vparam_AtomData

  Declaration of the Vparam_AtomData class as the sVparam_AtomData structure.
- typedef struct Vparam_ResData Vparam_ResData

  Declaration of the Vparam_ResData class as the Vparam_ResData structure.
- typedef struct Vparam Vparam

  Declaration of the Vparam class as the Vparam structure.

Functions

- VEXTERNC unsigned long int Vparam_memChk (Vparam *thee)

  Get number of bytes in this object and its members.
- VEXTERNC Vparam_AtomData * Vparam_AtomData_ctor ()

  Construct the object.
- VEXTERNC int Vparam_AtomData_ctor2 (Vparam_AtomData *thee)

  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.

9.77.1 Detailed Description

Contains declarations for class Vparam.

Version

Id

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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.

9.78 vparam.h

```
00001
00062 #ifndef _VPARAM_H_
00063 #define _VPARAM_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "maloc/maloc.h"
00068 #if defined(HAVE_MC_H)
00069 #include "mc/mc.h"
00070 #endif
00071
00072 #include "generic/vhal.h"
00073 #include "generic/vunit.h"
00074 #include "generic/vstring.h"
00075
00092 struct sVparam_AtomData {
00093     char atomName[VMAX_ARGLEN];
00094     char resName[VMAX_ARGLEN];
00095     double charge;
00096     double radius;
00097     double epsilon;
00099 };
00100
00106 typedef struct sVparam_AtomData Vparam_AtomData;
00107
00114 struct Vparam_ResData {
00115     Vmem *vmem;
00116     char name[VMAX_ARGLEN];
00117     int nAtomData;
00119     Vparam_AtomData *atomData;
00120 };
00121
00127 typedef struct Vparam_ResData Vparam_ResData;
00128
00135 struct Vparam {
00136
00137   Vmem *vmem;
00138   int nResData;
00140   Vparam_ResData *resData;
00141 };
00142
00147 typedef struct Vparam Vparam;
00148
00149 /* ///////////////////////////////////////////////////////////////////////////
00150 // Class Vparam: Inlineable methods (vparam.c)
00152
00153 #if !defined(VINLINE_VPARAM)
00154
00161     VEXTERNC unsigned long int Vparam_memChk(Vparam *thee);
00162
00163 #else /* if defined(VINLINE_VPARAM) */
00164
00165 #   define Vparam_memChk(thee) (Vmem_bytes((thee)->vmem))
00166
00167 #endif /* if !defined(VINLINE_VPARAM) */
00168
00169 /* ///////////////////////////////////////////////////////////////////////////
00170 // Class Vparam: Non-Inlineable methods (vparam.c)
00172
00177 VEXTERNC Vparam_AtomData* Vparam_AtomData_ctor();
00178
00184 VEXTERNC int Vparam_AtomData_ctor2(Vparam_AtomData *thee);
00185
00190 VEXTERNC void Vparam_AtomData_dtor(Vparam_AtomData **thee);
00191
00196 VEXTERNC void Vparam_AtomData_dtor2(Vparam_AtomData *thee);
00197
00205 VEXTERNC void Vparam_AtomData_copyTo(Vparam_AtomData *thee,
00206   Vparam_AtomData *dest);
```

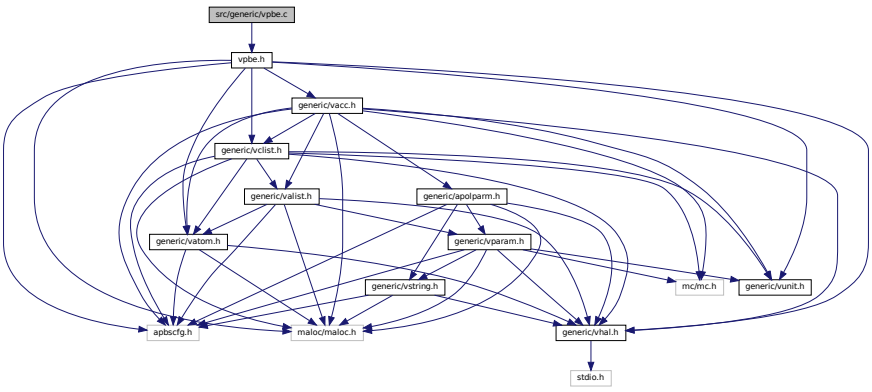
```
00207
00215 VEXTERNC void Vparam_ResData_copyTo(Vparam_ResData *thee,
00216   Vparam_ResData *dest);
00217
00225 VEXTERNC void Vparam_AtomData_copyFrom(Vparam_AtomData *thee,
00226   Vparam_AtomData *src);
00227
00233 VEXTERNC Vparam_ResData* Vparam_ResData_ctor(Vmem *mem);
00234
00241 VEXTERNC int Vparam_ResData_ctor2(Vparam_ResData *thee, Vmem *mem);
00242
00247 VEXTERNC void Vparam_ResData_dtor(Vparam_ResData **thee);
00248
00253 VEXTERNC void Vparam_ResData_dtor2(Vparam_ResData *thee);
00254
00259 VEXTERNC Vparam* Vparam_ctor();
00260
00266 VEXTERNC int Vparam_ctor2(Vparam *thee);
00267
00272 VEXTERNC void Vparam_dtor(Vparam **thee);
00273
00278 VEXTERNC void Vparam_dtor2(Vparam *thee);
00279
00290 VEXTERNC Vparam_ResData* Vparam_getResData(Vparam *thee,
00291   char resName[VMAX_ARGLEN]);
00292
00304 VEXTERNC Vparam_AtomData* Vparam_getAtomData(Vparam *thee,
00305   char resName[VMAX_ARGLEN], char atomName[VMAX_ARGLEN]);
00306
00335 VEXTERNC int Vparam_readFlatFile(Vparam *thee, const char *iodev,
00336   const char *iofmt, const char *thost, const char *fname);
00337
00348 VEXTERNC int Vparam_readXMLFile(Vparam *thee, const char *iodev,
00349   const char *iofmt, const char *thost, const char *fname);
00350
00351 #endif    /* ifndef _VPARAM_H_ */
```

9.79 src/generic/vpbe.c File Reference

Class Vpbe methods.

`#include "vpbe.h"`

Include dependency graph for vpbe.c:



Macros

- #define **MAX_SPLINE_WINDOW** 0.5

Functions

- VPUBLIC Valist * Vpbe_getValist (Vpbe *thee)

  Get atom list.
- VPUBLIC Vacc * Vpbe_getVacc (Vpbe *thee)

  Get accessibility oracle.
- VPUBLIC double Vpbe_getBulkIonicStrength (Vpbe *thee)

  Get bulk ionic strength.
- VPUBLIC double Vpbe_getTemperature (Vpbe *thee)

  Get temperature.
- VPUBLIC double Vpbe_getSoluteDiel (Vpbe *thee)

  Get solute dielectric constant.
- VPUBLIC double * Vpbe_getSoluteCenter (Vpbe *thee)

  Get coordinates of solute center.
- VPUBLIC double Vpbe_getSolventDiel (Vpbe *thee)

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

  Get solvent molecule radius.
- VPUBLIC double Vpbe_getMaxIonRadius (Vpbe *thee)

  Get maximum radius of ion species.
- VPUBLIC double Vpbe_getXkappa (Vpbe *thee)

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

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

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

  Get charge scaling factor.
- VPUBLIC double Vpbe_getSoluteRadius (Vpbe *thee)

  Get sphere radius which bounds biomolecule.
- VPUBLIC double Vpbe_getSoluteXlen (Vpbe *thee)

  Get length of solute in x dimension.
- VPUBLIC double Vpbe_getSoluteYlen (Vpbe *thee)

  Get length of solute in y dimension.
- VPUBLIC double Vpbe_getSoluteZlen (Vpbe *thee)

  Get length of solute in z dimension.
- VPUBLIC double Vpbe_getSoluteCharge (Vpbe *thee)

  Get total solute charge.
- VPUBLIC double Vpbe_getzmem (Vpbe *thee)

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

  Get length of the membrane (A)
  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.

9.79.1 Detailed Description

Class Vpbe methods.

Author

Nathan Baker

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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) 2002-2020, 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.

9.80 vpbe.c

```
00001
00057 #include "vpbe.h"
00058
00059 /* ////////////////////////////////////////////////////////////////////////// */
00060 // Class Vpbe: Private method declaration
00062 #define MAX_SPLINE_WINDOW 0.5
00063
00064 /* ////////////////////////////////////////////////////////////////////////// */
00065 // Class Vpbe: Inlineable methods
00067 #if !defined(VINLINE_VPBE)
00068
00069 VPUBLIC Valist* Vpbe_getValist(Vpbe *thee) {
00070
00071    VASSERT(thee != VNULL);
00072    return thee->alist;
00073
00074 }
00075
00076 VPUBLIC Vacc* Vpbe_getVacc(Vpbe *thee) {
00077
00078    VASSERT(thee != VNULL);
00079    VASSERT(thee->paramFlag);
00080    return thee->acc;
00081
00082 }
00083
00084 VPUBLIC double Vpbe_getBulkIonicStrength(Vpbe *thee) {
00085
00086    VASSERT(thee != VNULL);
00087    VASSERT(thee->paramFlag);
00088    return thee->bulkIonicStrength;
00089 }
00090
00091 VPUBLIC double Vpbe_getTemperature(Vpbe *thee) {
00092
00093    VASSERT(thee != VNULL);
00094    VASSERT(thee->paramFlag);
00095    return thee->T;
00096
00097 }
00098
00099 VPUBLIC double Vpbe_getSoluteDiel(Vpbe *thee) {
00100
00101    VASSERT(thee != VNULL);
00102    VASSERT(thee->paramFlag);
00103    return thee->soluteDiel;
00104
00105 }
```

```
00106
00107 VPUBLIC double* Vpbe_getSoluteCenter(Vpbe *thee) {
00108
00109    VASSERT(thee != VNULL);
00110    return thee->soluteCenter;
00111 }
00112
00113 VPUBLIC double Vpbe_getSolventDiel(Vpbe *thee) {
00114
00115    VASSERT(thee != VNULL);
00116    VASSERT(thee->paramFlag);
00117    return thee->solventDiel;
00118 }
00119
00120 VPUBLIC double Vpbe_getSolventRadius(Vpbe *thee) {
00121
00122    VASSERT(thee != VNULL);
00123    VASSERT(thee->paramFlag);
00124    return thee->solventRadius;
00125 }
00126
00127 VPUBLIC double Vpbe_getMaxIonRadius(Vpbe *thee) {
00128
00129    VASSERT(thee != VNULL);
00130    VASSERT(thee->paramFlag);
00131    return thee->maxIonRadius;
00132 }
00133
00134 VPUBLIC double Vpbe_getXkappa(Vpbe *thee) {
00135
00136    VASSERT(thee != VNULL);
00137    VASSERT(thee->paramFlag);
00138    return thee->xkappa;
00139 }
00140
00141 VPUBLIC double Vpbe_getDeblen(Vpbe *thee) {
00142
00143    VASSERT(thee != VNULL);
00144    VASSERT(thee->paramFlag);
00145    return thee->deblen;
00146 }
00147
00148 VPUBLIC double Vpbe_getZkappa2(Vpbe *thee) {
00149
00150    VASSERT(thee != VNULL);
00151    VASSERT(thee->paramFlag);
00152    return thee->zkappa2;
00153 }
00154
00155 VPUBLIC double Vpbe_getZmagic(Vpbe *thee) {
00156
00157    VASSERT(thee != VNULL);
00158    VASSERT(thee->paramFlag);
00159    return thee->zmagic;
00160 }
00161
00162 VPUBLIC double Vpbe_getSoluteRadius(Vpbe *thee) {
00163
00164    VASSERT(thee != VNULL);
00165    return thee->soluteRadius;
00166 }
00167
00168 VPUBLIC double Vpbe_getSoluteXlen(Vpbe *thee) {
00169
00170    VASSERT(thee != VNULL);
00171    return thee->soluteXlen;
00172 }
00173
00174 VPUBLIC double Vpbe_getSoluteYlen(Vpbe *thee) {
00175
00176    VASSERT(thee != VNULL);
00177    return thee->soluteYlen;
00178 }
00179
00180 VPUBLIC double Vpbe_getSoluteZlen(Vpbe *thee) {
00181
00182    VASSERT(thee != VNULL);
00183    return thee->soluteZlen;
00184 }
00185
00186 VPUBLIC double Vpbe_getSoluteCharge(Vpbe *thee) {
```

```
00187
00188    VASSERT(thee != VNULL);
00189    return thee->soluteCharge;
00190 }
00191
00192 /* ///////////////////////////////////////////////////////////////////////////
00193  // Routine:  Vpbe_getzmem
00194  // Purpose: This routine returns values stored in the structure thee.
00195  // Author:  Michael Grabe
00197 VPUBLIC double Vpbe_getzmem(Vpbe *thee) {
00198
00199     VASSERT(thee != VNULL);
00200     VASSERT(thee->param2Flag);
00201     return thee->z_mem;
00202 }
00203
00204 /* ///////////////////////////////////////////////////////////////////////////
00205  // Routine:  Vpbe_getLmem
00206  // Purpose: This routine returns values stored in the structure thee.
00207  // Author:  Michael Grabe
00209 VPUBLIC double Vpbe_getLmem(Vpbe *thee) {
00210
00211     VASSERT(thee != VNULL);
00212     VASSERT(thee->param2Flag);
00213     return thee->L;
00214 }
00215
00216 /* ///////////////////////////////////////////////////////////////////////////
00217  // Routine:  Vpbe_getmembraneDiel
00218  // Purpose: This routine returns values stored in the structure thee.
00219  // Author:  Michael Grabe
00221 VPUBLIC double Vpbe_getmembraneDiel(Vpbe *thee) {
00222
00223     VASSERT(thee != VNULL);
00224     VASSERT(thee->param2Flag);
00225     return thee->membraneDiel;
00226 }
00227
00228 /* ///////////////////////////////////////////////////////////////////////////
00229  // Routine:  Vpbe_getmemv
00230  // Purpose: This routine returns values stored in the structure thee.
00231  // Author:  Michael Grabe
00233 VPUBLIC double Vpbe_getmemv(Vpbe *thee) {
00234
00235     VASSERT(thee != VNULL);
00236     VASSERT(thee->param2Flag);
00237     return thee->V;
00238 }
00239
00240 #endif /* if !defined(VINLINE_VPBE) */
00241
00242 /* ///////////////////////////////////////////////////////////////////////////
00243 // Class Vpbe: Non-inlineable methods
00245
00246 VPUBLIC Vpbe* Vpbe_ctor(Valist *alist, int ionNum, double *ionConc,
00247                         double *ionRadii, double *ionQ, double T,
00248                         double soluteDiel, double solventDiel,
00249                         double solventRadius, int focusFlag, double sdens,
00250                         double z_mem, double L, double membraneDiel, double V ) {
00251
00252     /* Set up the structure */
00253     Vpbe *thee = VNULL;
00254     thee = (Vpbe*)Vmem_malloc(VNULL, 1, sizeof(Vpbe) );
00255     VASSERT( thee != VNULL);
00256     VASSERT( Vpbe_ctor2(thee, alist, ionNum, ionConc, ionRadii, ionQ,
00257                         T, soluteDiel, solventDiel, solventRadius, focusFlag, sdens,
00258                         z_mem, L, membraneDiel, V) );
00259
00260     return thee;
00261 }
00262
00263
00264 VPUBLIC int Vpbe_ctor2(Vpbe *thee, Valist *alist, int ionNum,
00265                        double *ionConc, double *ionRadii,
00266                        double *ionQ, double T, double soluteDiel,
00267                        double solventDiel, double solventRadius, int focusFlag,
00268                        double sdens, double z_mem, double L, double membraneDiel,
00269                        double V) {
00270
00271     int i, iatom, inhash[3];
00272     double atomRadius;
```

```
00273     Vatom *atom;
00274     double center[3] = {0.0, 0.0, 0.0};
00275     double lower_corner[3] = {0.0, 0.0, 0.0};
00276     double upper_corner[3] = {0.0, 0.0, 0.0};
00277     double disp[3], dist, radius, charge, xmin, xmax, ymin, ymax, zmin, zmax;
00278     double x, y, z, netCharge;
00279     double nhash[3];
00280     const double N_A = 6.022045000e+23;
00281     const double e_c = 4.803242384e-10;
00282     const double k_B = 1.380662000e-16;
00283     const double pi  = 4. * VATAN(1.);
00284
00285     /* Set up memory management object */
00286     thee->vmem = Vmem_ctor("APBS::VPBE");
00287
00288     VASSERT(thee != VNULL);
00289     if (alist == VNULL) {
00290         Vnm_print(2, "Vpbe_ctor2: Got null pointer to Valist object!\n");
00291         return 0;
00292     }
00293
00294     /* **** STUFF THAT GETS DONE FOR EVERYONE **** */
00295     /* Set pointers */
00296     thee->alist = alist;
00297     thee->paramFlag = 0;
00298
00299     /* Determine solute center */
00300     center[0] = thee->alist->center[0];
00301     center[1] = thee->alist->center[1];
00302     center[2] = thee->alist->center[2];
00303     thee->soluteCenter[0] = center[0];
00304     thee->soluteCenter[1] = center[1];
00305     thee->soluteCenter[2] = center[2];
00306
00307     /* Determine solute length and charge*/
00308     radius = 0;
00309     atom = Valist_getAtom(thee->alist, 0);
00310     xmin = Vatom_getPosition(atom)[0];
00311     xmax = Vatom_getPosition(atom)[0];
00312     ymin = Vatom_getPosition(atom)[1];
00313     ymax = Vatom_getPosition(atom)[1];
00314     zmin = Vatom_getPosition(atom)[2];
00315     zmax = Vatom_getPosition(atom)[2];
00316     charge = 0;
00317     for (iatom=0; iatom<Valist_getNumberAtoms(thee->alist); iatom++) {
00318         atom = Valist_getAtom(thee->alist, iatom);
00319         atomRadius = Vatom_getRadius(atom);
00320         x = Vatom_getPosition(atom)[0];
00321         y = Vatom_getPosition(atom)[1];
00322         z = Vatom_getPosition(atom)[2];
00323         if ((x+atomRadius) > xmax) xmax = x + atomRadius;
00324         if ((x-atomRadius) < xmin) xmin = x - atomRadius;
00325         if ((y+atomRadius) > ymax) ymax = y + atomRadius;
00326         if ((y-atomRadius) < ymin) ymin = y - atomRadius;
00327         if ((z+atomRadius) > zmax) zmax = z + atomRadius;
00328         if ((z-atomRadius) < zmin) zmin = z - atomRadius;
00329         disp[0] = (x - center[0]);
00330         disp[1] = (y - center[1]);
00331         disp[2] = (z - center[2]);
00332         dist = (disp[0]*disp[0]) + (disp[1]*disp[1]) + (disp[2]*disp[2]);
00333         dist = VSQRT(dist) + atomRadius;
00334         if (dist > radius) radius = dist;
00335         charge += Vatom_getCharge(Valist_getAtom(thee->alist, iatom));
00336     }
00337     thee->soluteRadius = radius;
00338     Vnm_print(0, "Vpbe_ctor2:  solute radius = %g\n", radius);
00339     thee->soluteXlen = xmax - xmin;
00340     thee->soluteYlen = ymax - ymin;
00341     thee->soluteZlen = zmax - zmin;
00342     Vnm_print(0, "Vpbe_ctor2:  solute dimensions = %g x %g x %g\n",
00343             thee->soluteXlen, thee->soluteYlen, thee->soluteZlen);
00344     thee->soluteCharge = charge;
00345     Vnm_print(0, "Vpbe_ctor2:  solute charge = %g\n", charge);
00346
00347     /* Set parameters */
00348     thee->numIon = ionNum;
00349     if (thee->numIon >= MAXION) {
00350         Vnm_print(2, "Vpbe_ctor2:  Too many ion species (MAX = %d)!\n",
00351           MAXION);
00352         return 0;
00353     }
```

```
00354     thee->bulkIonicStrength = 0.0;
00355     thee->maxIonRadius = 0.0;
00356     netCharge = 0.0;
00357     for (i=0; i<thee->numIon; i++) {
00358         thee->ionConc[i] = ionConc[i];
00359         thee->ionRadii[i] = ionRadii[i];
00360         if (ionRadii[i] > thee->maxIonRadius) thee->maxIonRadius = ionRadii[i];
00361         thee->ionQ[i] = ionQ[i];
00362         thee->bulkIonicStrength += (0.5*ionConc[i]*VSQR(ionQ[i]));
00363         netCharge += (ionConc[i]*ionQ[i]);
00364     }
00365 #ifndef VAPBSQUIET
00366     Vnm_print(1, "  Vpbe_ctor:  Using max ion radius (%g A) for exclusion \
00367 function\n", thee->maxIonRadius);
00368 #endif
00369     if (VABS(netCharge) > VSMALL) {
00370         Vnm_print(2, "Vpbe_ctor2:  You have a counterion charge imbalance!\n");
00371         Vnm_print(2, "Vpbe_ctor2:  Net charge conc. = %g M\n", netCharge);
00372         return 0;
00373     }
00374     thee->T = T;
00375     thee->soluteDiel = soluteDiel;
00376     thee->solventDiel = solventDiel;
00377     thee->solventRadius = solventRadius;
00378
00379     /* Compute parameters:
00380      *
00381      * kappa^2 = (8 pi N_A e_c^2) I_s / (1000 eps_w k_B T)
00382      * kappa   = 0.325567 * I_s^{1/2}   angstroms^{-1}
00383      * deblen  = 1 / kappa
00384      *         = 3.071564378 * I_s^{1/2}   angstroms
00385      * \bar{kappa}^2 = eps_w * kappa^2
00386      * zmagic  = (4 * pi * e_c^2) / (k_B T)   (we scale the diagonal later)
00387      *         = 7046.528838
00388      */
00389     if (thee->T == 0.0) {
00390         Vnm_print(2, "Vpbe_ctor2:  You set the temperature to 0 K.\n");
00391         Vnm_print(2, "Vpbe_ctor2:  That violates the 3rd Law of Thermo.!");
00392         return 0;
00393     }
00394     if (thee->bulkIonicStrength == 0.) {
00395         thee->xkappa  = 0.;
00396         thee->deblen  = 0.;
00397         thee->zkappa2 = 0.;
00398     } else {
00399         thee->xkappa  = VSQRT( thee->bulkIonicStrength * 1.0e-16 *
00400             ((8.0 * pi * N_A * e_c*e_c) /
00401             (1000.0 * thee->solventDiel * k_B * T))
00402         );
00403         thee->deblen  = 1. / thee->xkappa;
00404         thee->zkappa2 = thee->solventDiel * VSQR(thee->xkappa);
00405     }
00406     Vnm_print(0, "Vpbe_ctor2:  bulk ionic strength = %g\n",
00407             thee->bulkIonicStrength);
00408     Vnm_print(0, "Vpbe_ctor2:  xkappa = %g\n", thee->xkappa);
00409     Vnm_print(0, "Vpbe_ctor2:  Debye length = %g\n", thee->deblen);
00410     Vnm_print(0, "Vpbe_ctor2:  zkappa2 = %g\n", thee->zkappa2);
00411     thee->zmagic  = ((4.0 * pi * e_c*e_c) / (k_B * thee->T)) * 1.0e+8;
00412     Vnm_print(0, "Vpbe_ctor2:  zmagic = %g\n", thee->zmagic);
00413
00414     /* Compute accessibility objects:
00415      *   - Allow for extra room in the case of spline windowing
00416      *   - Place some limits on the size of the hash table in the case of very
00417      *     large molecules
00418      */
00419     if (thee->maxIonRadius > thee->solventRadius)
00420         radius = thee->maxIonRadius + MAX_SPLINE_WINDOW;
00421     else radius = thee->solventRadius + MAX_SPLINE_WINDOW;
00422
00423     nhash[0] = (thee->soluteXlen)/0.5;
00424     nhash[1] = (thee->soluteYlen)/0.5;
00425     nhash[2] = (thee->soluteZlen)/0.5;
00426     for (i=0; i<3; i++) inhash[i] = (int)(nhash[i]);
00427
00428     for (i=0;i<3;i++){
00429         if (inhash[i] < 3) inhash[i] = 3;
00430         if (inhash[i] > MAX_HASH_DIM) inhash[i] = MAX_HASH_DIM;
00431     }
00432     Vnm_print(0, "Vpbe_ctor2:  Constructing Vclist with %d x %d x %d table\n",
00433             inhash[0], inhash[1], inhash[2]);
00434
```

```
00435     thee->clist = Vclist_ctor(thee->alist, radius, inhash,
00436             CLIST_AUTO_DOMAIN, lower_corner, upper_corner);
00437
00438     VASSERT(thee->clist != VNULL);
00439     thee->acc = Vacc_ctor(thee->alist, thee->clist, sdens);
00440
00441     VASSERT(thee->acc != VNULL);
00442
00443     /* SMPBE Added */
00444     thee->smsize = 0.0;
00445     thee->smvolume = 0.0;
00446     thee->ipkey = 0;
00447
00448     thee->paramFlag = 1;
00449
00450     /*-----------------------------------------------------------*/
00451     /* added by Michael Grabe                                    */
00452     /*-----------------------------------------------------------*/
00453
00454     thee->z_mem = z_mem;
00455     thee->L = L;
00456     thee->membraneDiel = membraneDiel;
00457     thee->V = V;
00458
00459        if (V != 0.0) thee->param2Flag = 1;
00460        else thee->param2Flag = 0;
00461
00462     /*-----------------------------------------------------------*/
00463
00464     return 1;
00465 }
00466
00467 VPUBLIC void Vpbe_dtor(Vpbe **thee) {
00468     if ((*thee) != VNULL) {
00469         Vpbe_dtor2(*thee);
00470         Vmem_free(VNULL, 1, sizeof(Vpbe), (void **)thee);
00471         (*thee) = VNULL;
00472     }
00473 }
00474
00475 VPUBLIC void Vpbe_dtor2(Vpbe *thee) {
00476     Vclist_dtor(&(thee->clist));
00477     Vacc_dtor(&(thee->acc));
00478     Vmem_dtor(&(thee->vmem));
00479 }
00480
00481 VPUBLIC double Vpbe_getCoulombEnergy1(Vpbe *thee) {
00482
00483     int i, j, k, natoms;
00484
00485     double dist, *ipos, *jpos, icharge, jcharge;
00486     double energy = 0.0;
00487     double eps, T;
00488     Vatom *iatom, *jatom;
00489     Valist *alist;
00490
00491     VASSERT(thee != VNULL);
00492     alist = Vpbe_getValist(thee);
00493     VASSERT(alist != VNULL);
00494     natoms = Valist_getNumberAtoms(alist);
00495
00496     /* Do the sum */
00497     for (i=0; i<natoms; i++) {
00498         iatom = Valist_getAtom(alist,i);
00499         icharge = Vatom_getCharge(iatom);
00500         ipos = Vatom_getPosition(iatom);
00501         for (j=i+1; j<natoms; j++) {
00502             jatom = Valist_getAtom(alist,j);
00503             jcharge = Vatom_getCharge(jatom);
00504             jpos = Vatom_getPosition(jatom);
00505             dist = 0;
00506             for (k=0; k<3; k++) dist += ((ipos[k]-jpos[k])*(ipos[k]-jpos[k]));
00507             dist = VSQRT(dist);
00508             energy = energy + icharge*jcharge/dist;
00509         }
00510     }
00511
00512     /* Convert the result to J */
00513     T = Vpbe_getTemperature(thee);
00514     eps = Vpbe_getSoluteDiel(thee);
00515     energy = energy*Vunit_ec*Vunit_ec/(4*Vunit_pi*Vunit_eps0*eps*(1.0e-10));
```

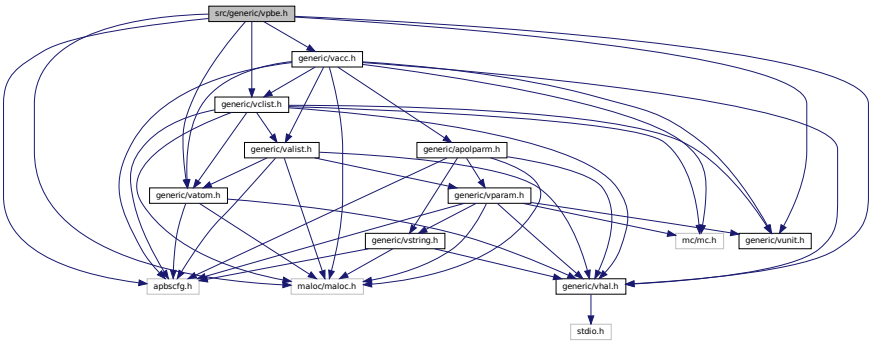
```
00516
00517     /* Scale by Boltzmann energy */
00518     energy = energy/(Vunit_kb*T);
00519
00520     return energy;
00521 }
00522
00523 VPUBLIC unsigned long int Vpbe_memChk(Vpbe *thee) {
00524
00525     unsigned long int memUse = 0;
00526
00527     if (thee == VNULL) return 0;
00528
00529     memUse = memUse + sizeof(Vpbe);
00530     memUse = memUse + (unsigned long int)Vacc_memChk(thee->acc);
00531
00532     return memUse;
00533 }
00534
00535 VPUBLIC int Vpbe_getIons(Vpbe *thee, int *nion, double ionConc[MAXION],
00536   double ionRadii[MAXION], double ionQ[MAXION]) {
00537
00538     int i;
00539
00540     VASSERT(thee != VNULL);
00541
00542     *nion = thee->numIon;
00543     for (i=0; i<(*nion); i++) {
00544         ionConc[i] = thee->ionConc[i];
00545         ionRadii[i] = thee->ionRadii[i];
00546         ionQ[i] = thee->ionQ[i];
00547     }
00548
00549     return *nion;
00550 }
```

9.81 src/generic/vpbe.h File Reference

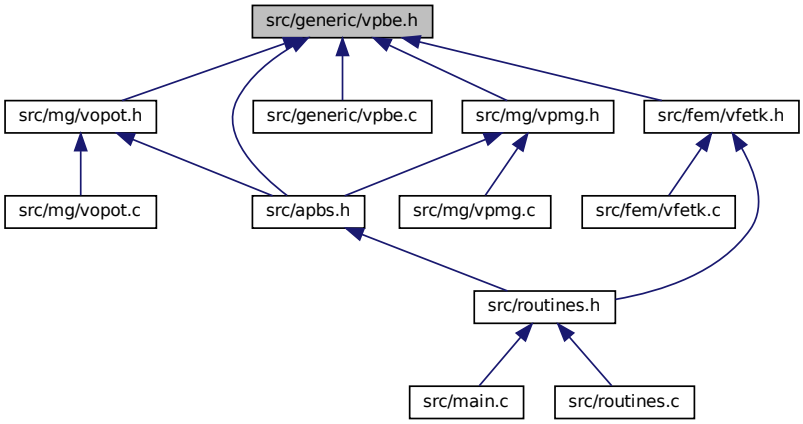
Contains declarations for class Vpbe.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/vunit.h"
#include "generic/vatom.h"
#include "generic/vacc.h"
#include "generic/vclist.h"
```

Include dependency graph for vpbe.h:



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



Data Structures

- struct sVpbe

  Contains public data members for Vpbe class/module.

Typedefs

- typedef struct sVpbe Vpbe

  Declaration of the Vpbe class as the Vpbe structure.

Functions

- VEXTERNC Valist * Vpbe_getValist (Vpbe *thee)

  Get atom list.
- VEXTERNC Vacc * Vpbe_getVacc (Vpbe *thee)

  Get accessibility oracle.
- VEXTERNC double Vpbe_getBulkIonicStrength (Vpbe *thee)

  Get bulk ionic strength.
- VEXTERNC double Vpbe_getMaxIonRadius (Vpbe *thee)

  Get maximum radius of ion species.
- VEXTERNC double Vpbe_getTemperature (Vpbe *thee)

  Get temperature.
- VEXTERNC double Vpbe_getSoluteDiel (Vpbe *thee)

  Get solute dielectric constant.
- VEXTERNC double Vpbe_getGamma (Vpbe *thee)

  Get apolar coefficient.
- VEXTERNC double Vpbe_getSoluteRadius (Vpbe *thee)

  Get sphere radius which bounds biomolecule.
- VEXTERNC double Vpbe_getSoluteXlen (Vpbe *thee)

  Get length of solute in x dimension.

- VEXTERNC double Vpbe_getSoluteYlen (Vpbe *thee)

  Get length of solute in y dimension.
- VEXTERNC double Vpbe_getSoluteZlen (Vpbe *thee)

  Get length of solute in z dimension.
- VEXTERNC double * Vpbe_getSoluteCenter (Vpbe *thee)

  Get coordinates of solute center.
- VEXTERNC double Vpbe_getSoluteCharge (Vpbe *thee)

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

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

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

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

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

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

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

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

  Get length of the membrane (A)
  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.

9.81.1 Detailed Description

Contains declarations for class Vpbe.

Version

Id

Author

Nathan A. Baker

Attention

Definition in file vpbe.h.

9.82 vpbe.h

```
00001
```

```
00066 #ifndef _VPBE_H_
00067 #define _VPBE_H_
00068
00069 #include "apbscfg.h"
00070
00071 #include "maloc/maloc.h"
00072
00073 #include "generic/vhal.h"
00074 #include "generic/vunit.h"
00075 #include "generic/vatom.h"
00076 #include "generic/vacc.h"
00077 #include "generic/vclist.h"
00078
00084 struct sVpbe {
00085
00086     Vmem *vmem;
00088     Valist *alist;
00089     Vclist *clist;
00090     Vacc *acc;
00092     double T;
00093     double soluteDiel;
00094     double solventDiel;
00095     double solventRadius;
00099     double bulkIonicStrength;
00100     double maxIonRadius;
00103     int    numIon;
00104     double ionConc[MAXION];
00105     double ionRadii[MAXION];
00106     double ionQ[MAXION];
00108     double xkappa;
00109     double deblen;
00110     double zkappa2;
00111     double zmagic;
00113     double soluteCenter[3];
00114     double soluteRadius;
00115     double soluteXlen;
00116     double soluteYlen;
00117     double soluteZlen;
00118     double soluteCharge;
00120     double smvolume;
00121     double smsize;
00122     int ipkey;
00125     int paramFlag;
00127     /*-----------------------------------------------------------*/
00128     /* Added by Michael Grabe                                    */
00129     /*-----------------------------------------------------------*/
00130
00131     double z_mem;
00132     double L;
00133     double membraneDiel;
00134     double V;
00135     int param2Flag;
00136     /*-----------------------------------------------------------*/
00137
00138 };
00139
00144 typedef struct sVpbe Vpbe;
00145
00146 /* ///////////////////////////////////////////////////////////////////////////
00147    // Class Vpbe: Inlineable methods (vpbe.c)
00149
00150 #if !defined(VINLINE_VPBE)
00151
00158 VEXTERNC Valist* Vpbe_getValist(Vpbe *thee);
00159
00166 VEXTERNC Vacc*   Vpbe_getVacc(Vpbe *thee);
00167
00174 VEXTERNC double  Vpbe_getBulkIonicStrength(Vpbe *thee);
00175
00182 VEXTERNC double  Vpbe_getMaxIonRadius(Vpbe *thee);
00183
00190 VEXTERNC double  Vpbe_getTemperature(Vpbe *thee);
00191
00198 VEXTERNC double  Vpbe_getSoluteDiel(Vpbe *thee);
00199
00206 VEXTERNC double  Vpbe_getGamma(Vpbe *thee);
00207
00214 VEXTERNC double  Vpbe_getSoluteRadius(Vpbe *thee);
00215
00222 VEXTERNC double  Vpbe_getSoluteXlen(Vpbe *thee);
00223
```

```
00230 VEXTERNC double  Vpbe_getSoluteYlen(Vpbe *thee);
00231
00238 VEXTERNC double  Vpbe_getSoluteZlen(Vpbe *thee);
00239
00246 VEXTERNC double* Vpbe_getSoluteCenter(Vpbe *thee);
00247
00254 VEXTERNC double  Vpbe_getSoluteCharge(Vpbe *thee);
00255
00262 VEXTERNC double  Vpbe_getSolventDiel(Vpbe *thee);
00263
00270 VEXTERNC double  Vpbe_getSolventRadius(Vpbe *thee);
00271
00278 VEXTERNC double  Vpbe_getXkappa(Vpbe *thee);
00279
00286 VEXTERNC double  Vpbe_getDeblen(Vpbe *thee);
00287
00294 VEXTERNC double  Vpbe_getZkappa2(Vpbe *thee);
00295
00302 VEXTERNC double  Vpbe_getZmagic(Vpbe *thee);
00303
00304 /*-----------------------------------------------------------------*/
00305 /* Added by Michael Grabe                                          */
00306 /*-----------------------------------------------------------------*/
00307
00314 VEXTERNC double  Vpbe_getzmem(Vpbe *thee);
00315
00322 VEXTERNC double  Vpbe_getLmem(Vpbe *thee);
00323
00330 VEXTERNC double  Vpbe_getmembraneDiel(Vpbe *thee);
00331
00337 VEXTERNC double  Vpbe_getmemv(Vpbe *thee);
00338
00339 /*-----------------------------------------------------------------*/
00340
00341 #else /* if defined(VINLINE_VPBE) */
00342 #   define Vpbe_getValist(thee) ((thee)->alist)
00343 #   define Vpbe_getVacc(thee) ((thee)->acc)
00344 #   define Vpbe_getBulkIonicStrength(thee) ((thee)->bulkIonicStrength)
00345 #   define Vpbe_getTemperature(thee) ((thee)->T)
00346 #   define Vpbe_getSoluteDiel(thee) ((thee)->soluteDiel)
00347 #   define Vpbe_getSoluteCenter(thee) ((thee)->soluteCenter)
00348 #   define Vpbe_getSoluteRadius(thee) ((thee)->soluteRadius)
00349 #   define Vpbe_getSoluteXlen(thee) ((thee)->soluteXlen)
00350 #   define Vpbe_getSoluteYlen(thee) ((thee)->soluteYlen)
00351 #   define Vpbe_getSoluteZlen(thee) ((thee)->soluteZlen)
00352 #   define Vpbe_getSoluteCharge(thee) ((thee)->soluteCharge)
00353 #   define Vpbe_getSolventDiel(thee) ((thee)->solventDiel)
00354 #   define Vpbe_getSolventRadius(thee) ((thee)->solventRadius)
00355 #   define Vpbe_getMaxIonRadius(thee) ((thee)->maxIonRadius)
00356 #   define Vpbe_getXkappa(thee) ((thee)->xkappa)
00357 #   define Vpbe_getDeblen(thee) ((thee)->deblen)
00358 #   define Vpbe_getZkappa2(thee) ((thee)->zkappa2)
00359 #   define Vpbe_getZmagic(thee) ((thee)->zmagic)
00360
00361 /*-----------------------------------------------------------------*/
00362 /* Added by Michael Grabe                                          */
00363 /*-----------------------------------------------------------------*/
00364
00365 #   define Vpbe_getzmem(thee) ((thee)->z_mem)
00366 #   define Vpbe_getLmem(thee) ((thee)->L)
00367 #   define Vpbe_getmembraneDiel(thee) ((thee)->membraneDiel)
00368 #   define Vpbe_getmemv(thee) ((thee)->V)
00369
00370 /*-----------------------------------------------------------------*/
00371
00372
00373 #endif /* if !defined(VINLINE_VPBE) */
00374
00375 /* ///////////////////////////////////////////////////////////////////////
00376    // Class Vpbe: Non-Inlineable methods (vpbe.c)
00378
00399 VEXTERNC Vpbe*   Vpbe_ctor(
00400                            Valist *alist, /**< Atom list */
00401                            int ionNum,
00402                            double *ionConc,
00403                            double *ionRadii,
00404                            double *ionQ,
00405                            double T,
00406                            double soluteDiel,
00407                            double solventDiel,
00408                            double solventRadius,
```

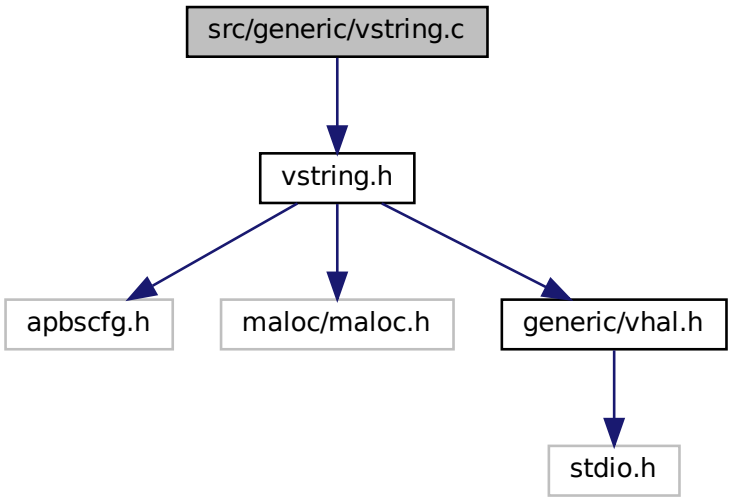
```
00409                             int focusFlag,
00410                             double sdens,
00411                             double z_mem,
00412                             double L,
00413                             double membraneDiel,
00414                             double V
00415                             );
00416
00437 VEXTERNC int    Vpbe_ctor2(
00438                             Vpbe *thee,
00439                             Valist *alist,
00440                             int ionNum,
00441                             double *ionConc,
00442                             double *ionRadii,
00443                             double *ionQ,
00444                             double T,
00445                             double soluteDiel,
00446                             double solventDiel,
00447                             double solventRadius,
00448                             int focusFlag,
00449                             double sdens,
00450                             double z_mem,
00451                             double L,
00452                             double membraneDiel,
00453                             double V
00454                             );
00455
00466 VEXTERNC int     Vpbe_getIons(Vpbe *thee, int *nion, double ionConc[MAXION],
00467                               double ionRadii[MAXION], double ionQ[MAXION]);
00468
00474 VEXTERNC void    Vpbe_dtor(Vpbe **thee);
00475
00481 VEXTERNC void    Vpbe_dtor2(Vpbe *thee);
00482
00497 VEXTERNC double  Vpbe_getCoulombEnergy1(Vpbe *thee);
00498
00506 VEXTERNC unsigned long int Vpbe_memChk(Vpbe *thee);
00507
00508 #endif /* ifndef _VPBE_H_ */
```

9.83 src/generic/vstring.c File Reference

Class Vstring methods.

```
#include "vstring.h"
```

Include dependency graph for vstring.c:



Functions

- VPUBLIC int Vstring_strcasecmp (const char *s1, const char *s2)

  Case-insensitive string comparison (BSD standard)
- VPUBLIC int Vstring_isdigit (const char *tok)

  A modified sscanf that examines the complete string.
- char * Vstring_wrappedtext (const char *str, int right_margin, int left_padding)

9.83.1 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.
```

Definition in file vstring.c.

9.84 vstring.c

```
00001
00057 #include "vstring.h"
00058
00059 /* ////////////////////////////////////////////////////////////////////////// */
00060 // Routine:  Vstring_strcasecmp
00061 //
00062 //           Copyright (c) 1988-1993 The Regents of the University of
00063 //                       California.
00064 //           Copyright (c) 1995-1996 Sun Microsystems, Inc.
00066 VPUBLIC int Vstring_strcasecmp(const char *s1, const char *s2) {
00067
00068 #if !defined(HAVE_STRCASECMP)
00069     unsigned char charmap[] = {
00070       0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
00071       0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
00072       0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
00073       0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f,
00074       0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
00075       0x28, 0x29, 0x2a, 0x2b, 0x2c, 0x2d, 0x2e, 0x2f,
00076       0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
00077       0x38, 0x39, 0x3a, 0x3b, 0x3c, 0x3d, 0x3e, 0x3f,
00078       0x40, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67,
00079       0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f,
00080       0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77,
00081       0x78, 0x79, 0x7a, 0x5b, 0x5c, 0x5d, 0x5e, 0x5f,
00082       0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67,
00083       0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f,
```

```
00084       0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77,
00085       0x78, 0x79, 0x7a, 0x7b, 0x7c, 0x7d, 0x7e, 0x7f,
00086       0x80, 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x87,
00087       0x88, 0x89, 0x8a, 0x8b, 0x8c, 0x8d, 0x8e, 0x8f,
00088       0x90, 0x91, 0x92, 0x93, 0x94, 0x95, 0x96, 0x97,
00089       0x98, 0x99, 0x9a, 0x9b, 0x9c, 0x9d, 0x9e, 0x9f,
00090       0xa0, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7,
00091       0xa8, 0xa9, 0xaa, 0xab, 0xac, 0xad, 0xae, 0xaf,
00092       0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7,
00093       0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0xbd, 0xbe, 0xbf,
00094       0xc0, 0xe1, 0xe2, 0xe3, 0xe4, 0xc5, 0xe6, 0xe7,
00095       0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef,
00096       0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
00097       0xf8, 0xf9, 0xfa, 0xdb, 0xdc, 0xdd, 0xde, 0xdf,
00098       0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7,
00099       0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef,
00100       0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
00101       0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff,
00102     };
00103
00104     unsigned char u1, u2;
00105
00106     for ( ; ; s1++, s2++) {
00107     u1 = (unsigned char) *s1;
00108     u2 = (unsigned char) *s2;
00109     if ((u1 == '\0') || (charmap[u1] != charmap[u2])) {
00110         break;
00111     }
00112     }
00113     return charmap[u1] - charmap[u2];
00114
00115 #else
00116
00117     return strcasecmp(s1, s2);
00118
00119 #endif
00120
00121 }
00122
00123 /* ////////////////////////////////////////////////////////////////////////
00124 // Routine:  Vstring_isdigit
00125 //
00126 //           Improves upon sscanf to see if a token is an int or not
00127 //
00128 //           Returns isdigit: 1 if a digit, 0 otherwise
00130 VPUBLIC int Vstring_isdigit(const char *tok) {
00131     int i, isdigit, ti;
00132     char checkchar[1];
00133     char name[VMAX_BUFSIZE];
00134     strcpy(name,tok);
00135     isdigit = 1;
00136     for(i=0; ; i++){
00137       checkchar[0] =  name[i];
00138       if (name[i] == '\0'){
00139         break;
00140       }
00141       if (sscanf(checkchar, "%d", &ti) != 1){
00142         isdigit = 0;
00143         break;
00144       }
00145     }
00146     return isdigit;
00147 }
00148
00149
00155 char* Vstring_wrappedtext(const char* str, int right_margin, int left_padding)
00156 {
00157     int span = right_margin - left_padding;
00158     int i = 0;
00159     int k = 0;
00160     int j = 0;
00161     int line_len = 0;
00162     int hyphenate = 0;
00163     char* wrap_str;
00164     int   wrap_len;
00165     int len = strlen( str );
00166
00167     if( len == 0 )
00168         return VNULL;
00169
00170     wrap_str = (char*)malloc( len * sizeof(char) );
```

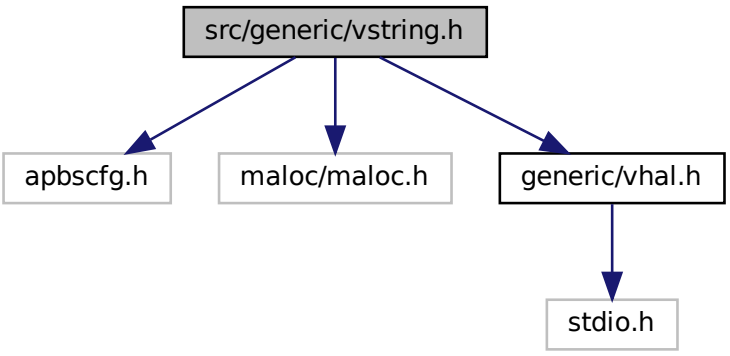
```
00171     wrap_len = len;
00172
00173     do
00174     {
00175         if( str[i] == ' ' )
00176         {
00177             i++;
00178         }
00179         else
00180         {
00184             if( k + right_margin + 2 > wrap_len )
00185             {
00186                 wrap_len += right_margin + 2;
00187                 wrap_str = (char*)realloc( wrap_str, wrap_len * sizeof( char ) );
00188             }
00189
00190
00191             if( i + span >= len )
00192             {
00193                 hyphenate = 0;
00194                 line_len = len - i;
00195             }
00196             else
00197             {
00198                 j = span;
00199                 do
00200                 {
00201                     if( str[ i + j ] == ' ' )
00202                     {
00203                         hyphenate = 0;
00204                         line_len = j;
00205                         break;
00206                     }
00207                     else if( j == 0 )
00208                     {
00209                         hyphenate = 1;
00210                         line_len = span - 1;
00211                         break;
00212                     }
00213                     else
00214                     {
00215                         j--;
00216                     }
00217                 } while( 1 );
00218             }
00219
00220             memset( wrap_str + k, ' ', left_padding * sizeof( char ) );
00221             k += left_padding;
00222
00223             memcpy( wrap_str + k, str + i, line_len * sizeof( char ) );
00224             k += line_len;
00225             i += line_len;
00226
00227             if( hyphenate )
00228                 wrap_str[k++] = '-';
00229
00230             wrap_str[k++] = '\n';
00231
00232             wrap_str[k] = '\0';
00233         }
00234     } while( i < len );
00235
00236     return wrap_str;
00237 }
```

9.85 src/generic/vstring.h File Reference

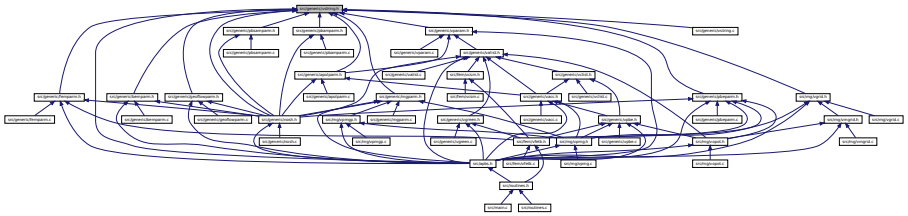
Contains declarations for class Vstring.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/vhal.h"
```

Include dependency graph for vstring.h:



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



Functions

- VEXTERNC int Vstring_strcasecmp (const char *s1, const char *s2)

  Case-insensitive string comparison (BSD standard)
- VEXTERNC int Vstring_isdigit (const char *tok)

  A modified sscanf that examines the complete string.
- char * Vstring_wrappedtext (const char *str, int right_margin, int left_padding)

9.85.1 Detailed Description

Contains declarations for class Vstring.

Version

Id

Author

Nathan A. Baker

Attention

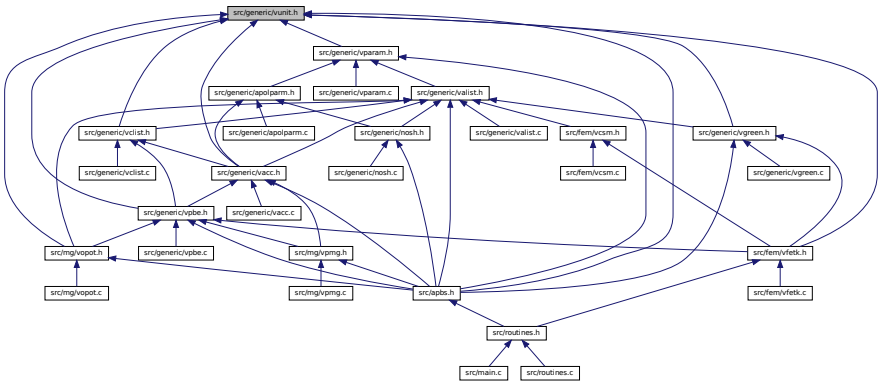
Definition in file vstring.h.

9.86 vstring.h

```
00001
00078 #ifndef _VSTRING_H_
00079 #define _VSTRING_H_
00080
00081 #include "apbscfg.h"
00082
00083 #include "maloc/maloc.h"
00084
00085 #include "generic/vhal.h"
00086
00099 VEXTERNC int Vstring_strcasecmp(const char *s1, const char *s2);
00100
00107 VEXTERNC int Vstring_isdigit(const char *tok);
00108
00114 VEXTERNC char* Vstring_wrappedtext(
00115         const char* str,
00116         int right_margin,
00117         int left_padding
00118         );
00119
00120 #endif    /* ifndef _VSTRING_H_ */
```

9.87 src/generic/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*

9.87.1 Detailed Description

Contains a collection of useful constants and conversion factors.

Author

Nathan Baker

Nathan A. Baker

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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.

9.88 vunit.h

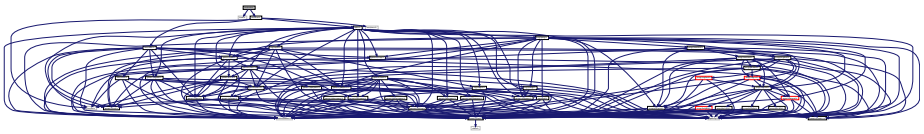
```
00001
00063 #ifndef _VUNIT_H_
00064 #define _VUNIT_H_
00065
00068 #define Vunit_J_to_cal  4.1840000e+00
00069
00072 #define Vunit_cal_to_J  2.3900574e-01
00073
00076 #define Vunit_amu_to_kg  1.6605402e-27
00077
00080 #define Vunit_kg_to_amu  6.0221367e+26
00081
00084 #define Vunit_ec_to_C  1.6021773e-19
00085
00088 #define Vunit_C_to_ec  6.2415065e+18
00089
00092 #define Vunit_ec  1.6021773e-19
00093
00096 #define Vunit_kb  1.3806581e-23
00097
00100 #define Vunit_Na  6.0221367e+23
00101
00104 #define Vunit_pi  VPI
00105
00108 #define Vunit_eps0  8.8541878e-12
00109
00112 #define Vunit_esu_ec2A  3.3206364e+02
00113
00116 #define Vunit_esu_kb            1.9871913e-03
00117
00118 #endif /* ifndef _VUNIT_H_ */
```

9.89 src/main.c File Reference

APBS "front end" program using formatted input files.

```
#include <time.h>
#include "routines.h"
```

Include dependency graph for main.c:



Functions

- int main (int argc, char **argv)

  The main APBS function.

9.89.1 Detailed Description

APBS "front end" program using formatted input files.

Author

Nathan Baker

```
This driver program represents a mish-mash of
instructions for
calculating electrostatic potentials, as well as free energies of
binding and solvation.  It is invoked as:

    apbs apbs.in

where apbs.in is a formatted input file (see documentation and
examples).
```

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
```

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

Definition in file main.c.

9.90 main.c

```
00001
00068 #include <time.h>
00069
00070 #include "routines.h"
00071
00072 VEMBED(rcsid="$Id$")
00073
00074
00080 int main(
00081          int argc,
00082          char **argv
00083          )
00084 {
00085     // PCE: Adding below variables temporarily
00086     clock_t ts, te;
00087     // End PCE
00088
00089     NOsh *nosh = VNULL;
00090
00091     MGparm *mgparm = VNULL;
00092     FEMparm *feparm = VNULL;
00093 #ifdef ENABLE_BEM
00094     BEMparm *bemparm = VNULL;
00095 #endif
00096     GEOFLOWparm *geoflowparm = VNULL;
00097     PBEparm *pbeparm = VNULL;
00098     APOLparm *apolparm = VNULL;
00099     Vparam *param = VNULL;
00100 #if defined(ENABLE_PBAM) || defined(ENABLE_PBSAM)
00101     PBAMparm *pbamparm = VNULL;
00102 #endif
00103
00104 #ifdef ENABLE_PBSAM
00105     PBSAMparm *pbsamparm = VNULL;
00106 #endif
00107
00108     Vmem *mem = VNULL;
00109     Vcom *com = VNULL;
00110     Vio *sock = VNULL;
00111 #ifdef HAVE_MC_H
00112     Vfetk *fetk[NOSH_MAXCALC];
00113     Gem *gm[NOSH_MAXMOL];
00114     int isolve;
00115 #else
00116     void *fetk[NOSH_MAXCALC];
00117     void *gm[NOSH_MAXMOL];
00118 #endif
00119     Vpmg *pmg[NOSH_MAXCALC];
00120     Vpmgp *pmgp[NOSH_MAXCALC];
00121     Vpbe *pbe[NOSH_MAXCALC];
00122     Valist *alist[NOSH_MAXMOL];
00123     Vgrid *dielXMap[NOSH_MAXMOL],
00124           *dielYMap[NOSH_MAXMOL],
00125           *dielZMap[NOSH_MAXMOL],
00126           *kappaMap[NOSH_MAXMOL],
00127           *potMap[NOSH_MAXMOL],
00128           *chargeMap[NOSH_MAXMOL];
00129     char *input_path = VNULL,
00130          *output_path = VNULL;
00131     int i,
00132         rank,   // proc id
```

```
00133         size,   // total num of procs
00134         k;
00135     size_t bytesTotal,
00136            highWater;
00137     Voutput_Format outputformat;
00138
00139     int rc = 0;
00140
00141     /* The energy double arrays below store energies from various calculations. */
00142     double qfEnergy[NOSH_MAXCALC],
00143            qmEnergy[NOSH_MAXCALC];
00144     double dielEnergy[NOSH_MAXCALC],
00145            totEnergy[NOSH_MAXCALC];
00146     double *atomEnergy[NOSH_MAXCALC];
00147     AtomForce *atomForce[NOSH_MAXCALC]; /* Stores forces from various calculations. */
00148     int nenergy[NOSH_MAXCALC], /* Stores either a flag (0,1) displaying whether
00149                                 * energies were calculated, or, if PCE_COMPS
00150                                 * was used, the number of atom energies stored
00151                                 * for the given calculation. */
00152         nforce[NOSH_MAXCALC]; /* Stores an integer which either says no
00153                                * calculation was performed (0) or gives the
00154                                * number of entries in the force array for each
00155                                * calculation. */
00156
00157     /* The real partition centers */
00158     double realCenter[3];
00159
00160     /* Instructions: */
00161     char header[] = {"\n\n\
00162 -----------------------------------------------------------------------\n\
00163     APBS -- Adaptive Poisson-Boltzmann Solver\n\
00164     Version " PACKAGE_STRING "\n\
00165     \n\
00166     Nathan A. Baker (nathan.baker@pnnl.gov)\n\
00167     Pacific Northwest National Laboratory\n\
00168     \n\
00169     Additional contributing authors listed in the code documentation.\n\
00170     \n\
00171     Copyright (c) 2010-2020 Battelle Memorial Institute. Developed at the Pacific\n\
00172     Northwest National Laboratory, operated by Battelle Memorial Institute, Pacific\n\
00173     Northwest Division for the U.S. Department of Energy.\n\
00174     \n\
00175     Portions Copyright (c) 2002-2010, Washington University in St. Louis.\n\
00176     Portions Copyright (c) 2002-2020, Nathan A. Baker.\n\
00177     Portions Copyright (c) 1999-2002, The Regents of the University of California.\n\
00178     Portions Copyright (c) 1995, Michael Holst.\n\
00179     All rights reserved.\n\
00180     \n\
00181     Redistribution and use in source and binary forms, with or without\n\
00182     modification, are permitted provided that the following conditions are met:\n\
00183     \n\
00184     * Redistributions of source code must retain the above copyright notice, this\n\
00185       list of conditions and the following disclaimer.\n\
00186     \n\
00187     * Redistributions in binary form must reproduce the above copyright notice,\n\
00188       this list of conditions and the following disclaimer in the documentation\n\
00189       and/or other materials provided with the distribution.\n\
00190     \n\
00191     * Neither the name of the developer nor the names of its contributors may be\n\
00192       used to endorse or promote products derived from this software without\n\
00193       specific prior written permission.\n\
00194     \n\
00195     THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS \"AS IS\" AND\n\
00196     ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED\n\
00197     WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE\n\
00198     DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR\n\
00199     ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES\n\
00200     (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;\n\
00201     LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND\n\
00202     ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT\n\
00203     (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS\n\
00204     SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.\n\
00205 -----------------------------------------------------------------------\n\
00206     APBS uses FETK (the Finite Element ToolKit) to solve the\n\
00207     Poisson-Boltzmann equation numerically.  FETK is a portable collection\n\
00208     of finite element modeling class libraries developed by the Michael Holst\n\
00209     research group and written in an object-oriented form of C.  FEtk is\n\
00210     designed to solve general coupled systems of nonlinear partial differential\n\
00211     equations using adaptive finite element methods, inexact Newton methods,\n\
00212     and algebraic multilevel methods.  More information about FEtk may be found\n\
00213     at <http://www.FEtk.ORG>.\n\
```

```
00214 -------------------------------------------------------------------------\n\
00215     APBS also uses Aqua to solve the Poisson-Boltzmann equation numerically.  \n\
00216     Aqua is a modified form of the Holst group PMG library <http://www.FEtk.ORG>\n\
00217     which has been modified by Patrice Koehl\n\
00218     <http://koehllab.genomecenter.ucdavis.edu/> for improved efficiency and\n\
00219     memory usage when solving the Poisson-Boltzmann equation.\n\
00220 -------------------------------------------------------------------------\n\
00221     Please cite your use of APBS as:\n\n\
00222     Baker NA, Sept D, Joseph S, Holst MJ, McCammon JA. Electrostatics of\n\
00223     nanosystems: application to microtubules and the ribosome. Proc.\n\
00224     Natl. Acad. Sci. USA 98, 10037-10041 2001.\n\
00225     \n\n"};
00226     char *usage =
00227 {"\n\n\
00228 -------------------------------------------------------------------------\n\
00229     This driver program calculates electrostatic potentials, energies,\n\
00230     and forces using both multigrid and finite element methods.\n\
00231         It is invoked as:\n\n\
00232     apbs [options] apbs.in\n\n\
00233     where apbs.in is a formatted input file and [options] are:\n\n\
00234 --output-file=<name>     Enables output logging to the path\n\
00235     listed in <name>.  Uses flat-file\n\
00236     format is --output-format is not used.\n\
00237 --output-format=<type>   Specifies format for logging.  Options\n\
00238     for type are either \"xml\" or \"flat\".\n\
00239 --help                   Display this help information.\n\
00240 --version                Display the current APBS version.\n\
00241 -------------------------------------------------------------------------\n\n"};
00242
00243     /* ************** CHECK PARALLEL STATUS ************** */
00244     VASSERT(Vcom_init(&argc, &argv));
00245     com = Vcom_ctor(1);
00246     rank = Vcom_rank(com);
00247     size = Vcom_size(com);
00248     startVio();
00249     Vnm_setIoTag(rank, size);
00250     Vnm_tprint( 0, "Hello world from PE %d\n", rank);
00251
00252     /* A bit of array/pointer initialization */
00253     mem = Vmem_ctor("MAIN");
00254     for (i=0; i<NOSH_MAXCALC; i++) {
00255         pmg[i] = VNULL;
00256         pmgp[i] = VNULL;
00257         fetk[i] = VNULL;
00258         pbe[i] = VNULL;
00259         qfEnergy[i] = 0;
00260         qmEnergy[i] = 0;
00261         dielEnergy[i] = 0;
00262         totEnergy[i] = 0;
00263         atomForce[i] = VNULL;
00264         nenergy[i] = 0;
00265         nforce[i] = 0;
00266     }
00267     for (i=0; i<NOSH_MAXMOL; i++) {
00268         alist[i] = VNULL;
00269         dielXMap[i] = VNULL;
00270         dielYMap[i] = VNULL;
00271         dielZMap[i] = VNULL;
00272         kappaMap[i] = VNULL;
00273         potMap[i] = VNULL;
00274         chargeMap[i] = VNULL;
00275     }
00276
00277     /* ********* CHECK INVOCATION AND OPTIONS ************* */
00278     Vnm_tstart(APBS_TIMER_WALL_CLOCK, "APBS WALL CLOCK");
00279     Vnm_tprint( 1, "%s", header);
00280
00281 #ifdef APBS_FAST
00282     Vnm_tprint(, 2"WARNING: APBS was compiled with the --enable-fast option.\n"
00283            "WARNING: This mode is experimental and subject to change in future releases.\n"
00284            "WARNING: The fast mode enables: Gauess-Seidel Smoothing and \n"
00285            "WARNING:   Conjugate Gradient Multigrid methods.\n\n");
00286 #endif
00287
00288     Vnm_tprint( 1, "This executable compiled on %s at %s\n\n", __DATE__, __TIME__);
00289
00290 #if defined(WITH_TINKER)
00291     Vnm_tprint( 2, "This executable was compiled with TINKER support and is not intended for stand-alone
      execution.\n");
00292     Vnm_tprint( 2, "Please compile another version without TINKER support.\n");
00293     exit(2);
```

```
00294 #endif
00295
00296     /* Process program arguments */
00297     i=0;
00298     outputformat = OUTPUT_NULL;
00299     while (i<argc){
00300         if (strncmp(argv[i], "--", 2) == 0) {
00301
00302             /* Long Options */
00303             if (Vstring_strcasecmp("--version", argv[i]) == 0){
00304                 Vnm_tprint(2, "%s\n", PACKAGE_STRING);
00305                 VJMPERR1(0);
00306             } else if (Vstring_strcasecmp("--help", argv[i]) == 0){
00307                 Vnm_tprint(2, "%s\n", usage);
00308                 VJMPERR1(0);
00309             } else if (strncmp(argv[i], "--output-format", 15) == 0) {
00310                 if (strstr(argv[i], "xml") != NULL) {
00311                     Vnm_tprint(2, "XML output format is now deprecated, please use --output-format=flat
      instead!\n\n");
00312                     VJMPERR1(0);
00313                 }
00314                 else if (strstr(argv[i], "flat") != NULL) {
00315                     outputformat = OUTPUT_FLAT;
00316                 } else {
00317                     Vnm_tprint(2, "Invalid output-format type!\n");
00318                     VJMPERR1(0);
00319                 }
00320             } else if (strncmp(argv[i], "--output-file=", 14) == 0){
00321                 output_path = strstr(argv[i], "=");
00322                 ++output_path;
00323                 if (outputformat == OUTPUT_NULL) outputformat = OUTPUT_FLAT;
00324             } else {
00325                 Vnm_tprint(2, "UNRECOGNIZED COMMAND LINE OPTION %s!\n", argv[i]);
00326                 Vnm_tprint(2, "%s\n", usage);
00327                 VJMPERR1(0);
00328             }
00329         } else {
00330
00331             /* Set the path to the input file */
00332             if ((input_path == VNULL) && (i != 0))
00333                 input_path = argv[i];
00334             else if (i != 0) {
00335                 Vnm_tprint(2, "ERROR -- CALLED WITH TOO MANY ARGUMENTS!\n", \
00336                         argc);
00337                 Vnm_tprint(2, "%s\n", usage);
00338                 VJMPERR1(0);
00339             }
00340         }
00341         i++;
00342     }
00343
00344     /* If we set an output format but no path, error. */
00345     if ((outputformat != 0) && (output_path == NULL)) {
00346         Vnm_tprint(2, "The --output-path variable must be set when using --output-format!\n");
00347         VJMPERR1(0);
00348     }
00349
00350     /* If we failed to specify an input file, error. */
00351     if (input_path == NULL) {
00352         Vnm_tprint(2, "ERROR -- APBS input file not specified!\n", argc);
00353         Vnm_tprint(2, "%s\n", usage);
00354         VJMPERR1(0);
00355     }
00356
00357     /* Append rank info if a parallel run */
00358     if ((size > 1) && (output_path != NULL))
00359         printf(output_path, "%s_%d", output_path, rank);
00360
00361     /* *************** PARSE INPUT FILE ****************** */
00362     nosh = NOsh_ctor(rank, size);
00363     Vnm_tprint( 1, "Parsing input file %s...\n", input_path);
00364     Vnm_tprint( 1, "rank %d size %d...\n", rank, size);
00365     sock = Vio_ctor("FILE", "ASC", VNULL, input_path, "r");
00366     if (sock == VNULL) {
00367         Vnm_tprint(2, "Error while opening input file %s!\n", input_path);
00368         VJMPERR1(0);
00369     }
00370     if (!NOsh_parseInput(nosh, sock)) {
00371         Vnm_tprint( 2, "Error while parsing input file.\n");
00372         VJMPERR1(0);
00373     }
```

```
00374     else
00375         Vnm_tprint( 1, "Parsed input file.\n");
00376     Vio_dtor(&sock);
00377
00378     /* *************** LOAD PARAMETERS AND MOLECULES ****************** */
00379     param = loadParameter(nosh);
00380     if (loadMolecules(nosh, param, alist) != 1) {
00381         Vnm_tprint(2, "Error reading molecules!\n");
00382         VJMPERR1(0);
00383     }
00384
00385     /* *************** SETUP CALCULATIONS *************** */
00386     if (NOsh_setupElecCalc(nosh, alist) != 1) {
00387         Vnm_tprint(2, "Error setting up ELEC calculations\n");
00388         VJMPERR1(0);
00389     }
00390
00391     if ((rc = NOsh_setupApolCalc(nosh, alist)) == ACD_ERROR) {
00392         Vnm_tprint(2, "Error setting up APOL calculations\n");
00393         VJMPERR1(0);
00394     }
00395
00396     /* ****************** CHECK APOL********************** */
00397     /* if((nosh->gotparm == 0) && (rc == ACD_YES)){
00398         Vnm_print(1,"\nError you must provide a parameter file if you\n" \
00399                     "     are performing an APOLAR calculation\n");
00400         VJMPERR1(0);
00401     } */
00402
00403 #if defined(DEBUG_MAC_OSX_OCL)
00404 #include "mach_chud.h"
00405 #include <stdint.h>
00406     uint64_t mbeg;
00407     machm_(&mbeg);
00408
00409     if(clFinish != NULL)
00410     {
00411         int ret = initOpenCL();
00412         printf("OpenCL runtime present - initialized = %i\n",ret);
00413     }
00414     else
00415     {
00416         setkOpenCLAvailable_(0);
00417         printf("OpenCL is not present!\n");
00418     }
00419 #endif
00420
00421 #if defined(DEBUG_MAC_OSX_STANDARD)
00422 #include "mach_chud.h"
00423 #include <stdint.h>
00424     uint64_t mbeg;
00425     machm_(&mbeg);
00426 #endif
00427
00428     /* *************** LOAD MAPS ****************** */
00429     if (loadDielMaps(nosh, dielXMap, dielYMap, dielZMap) != 1) {
00430         Vnm_tprint(2, "Error reading dielectric maps!\n");
00431         VJMPERR1(0);
00432     }
00433     if (loadKappaMaps(nosh, kappaMap) != 1) {
00434         Vnm_tprint(2, "Error reading kappa maps!\n");
00435         VJMPERR1(0);
00436     }
00437     if (loadPotMaps(nosh, potMap) != 1) {
00438         Vnm_tprint(2, "Error reading potential maps!\n");
00439         VJMPERR1(0);
00440     }
00441     if (loadChargeMaps(nosh, chargeMap) != 1) {
00442         Vnm_tprint(2, "Error reading charge maps!\n");
00443         VJMPERR1(0);
00444     }
00445
00446     /* *************** DO THE CALCULATIONS ****************** */
00447     Vnm_tprint( 1, "Preparing to run %d PBE calculations.\n",
00448                 nosh->ncalc);
00449     for (i=0; i<nosh->ncalc; i++) {
00450         Vnm_tprint( 1, "----------------------------------------\n");
00451
00452         switch (nosh->calc[i]->calctype) {
00453             /* Multigrid */
00454             case NCT_MG:
```

```
00455                 /* What is this?  This seems like a very awkward way to find
00456                 the right ELEC statement... */
00457                 for (k=0; k<nosh->nelec; k++) {
00458                     if (nosh->elec2calc[k] >= i) {
00459                         break;
00460                     }
00461                 }
00462                 if (Vstring_strcasecmp(nosh->elecname[k], "") == 0) {
00463                     Vnm_tprint( 1, "CALCULATION #%d: MULTIGRID\n", i+1);
00464                 } else {
00465                     Vnm_tprint( 1, "CALCULATION #%d (%s): MULTIGRID\n",
00466                                 i+1, nosh->elecname[k]);
00467                 }
00468                 /* Useful local variables */
00469                 mgparm = nosh->calc[i]->mgparm;
00470                 pbeparm = nosh->calc[i]->pbeparm;
00471 
00472                 /* Set up problem */
00473                 Vnm_tprint( 1, "  Setting up problem...\n");
00474 
00475                 if (!initMG(i, nosh, mgparm, pbeparm, realCenter, pbe,
00476                             alist, dielXMap, dielYMap, dielZMap, kappaMap,
00477                             chargeMap, pmgp, pmg, potMap)) {
00478                     Vnm_tprint( 2, "Error setting up MG calculation!\n");
00479                     VJMPERR1(0);
00480                 }
00481 
00482                 /* Print problem parameters */
00483                 printMGPARM(mgparm, realCenter);
00484                 printPBEPARM(pbeparm);
00485 
00486                 /* Solve PDE */
00487                 if (solveMG(nosh, pmg[i], mgparm->type) != 1) {
00488                     Vnm_tprint(2, "Error solving PDE!\n");
00489                     VJMPERR1(0);
00490                 }
00491 
00492                 /* Set partition information for observables and I/O */
00493                 if (setPartMG(nosh, mgparm, pmg[i]) != 1) {
00494                     Vnm_tprint(2, "Error setting partition info!\n");
00495                     VJMPERR1(0);
00496                 }
00497 
00498                 /* Write out energies */
00499                 energyMG(nosh, i, pmg[i],
00500                         &(nenergy[i]), &(totEnergy[i]), &(qfEnergy[i]),
00501                         &(qmEnergy[i]), &(dielEnergy[i]));
00502 
00503                 /* Write out forces */
00504                 forceMG(mem, nosh, pbeparm, mgparm, pmg[i], &(nforce[i]),
00505                         &(atomForce[i]), alist);
00506 
00507                 /* Write out data folks might want */
00508                 writedataMG(rank, nosh, pbeparm, pmg[i]);
00509 
00510                 /* Write matrix */
00511                 writematMG(rank, nosh, pbeparm, pmg[i]);
00512 
00513                 /* If needed, cache atom energies */
00514                 nenergy[i] = 0;
00515                 if ((pbeparm->calcenergy == PCE_COMPS) && (outputformat != OUTPUT_NULL)){
00516                     storeAtomEnergy(pmg[i], i, &(atomEnergy[i]), &(nenergy[i]));
00517                 }
00518 
00519                 fflush(stdout);
00520                 fflush(stderr);
00521 
00522                 break;
00523 
00524                 /* ***** Do FEM calculation ***** */
00525             case NCT_FEM:
00526 #ifdef HAVE_MC_H
00527                 for (k=0; k<nosh->nelec; k++) {
00528                     if (nosh->elec2calc[k] >= i) break;
00529                 }
00530                 if (Vstring_strcasecmp(nosh->elecname[i+1], "") == 0) {
00531                     Vnm_tprint( 1, "CALCULATION #%d: FINITE ELEMENT\n", i+1);
00532                 } else {
00533                     Vnm_tprint( 1, "CALCULATION #%d (%s): FINITE ELEMENT\n", i+1, nosh->elecname[k+1]);
00534                 }
00535 
```

```
00536                 /* Useful local variables */
00537                 feparm = nosh->calc[i]->femparm;
00538                 pbeparm = nosh->calc[i]->pbeparm;
00539 
00540                 /* Warn the user about some things */
00541                 Vnm_tprint(2, "#################### WARNING ###################\n");
00542                 Vnm_tprint(2, "## FE support is currently very experimental! ##\n");
00543                 Vnm_tprint(2, "#################### WARNING ###################\n");
00544 
00545                 /* Set up problem */
00546                 Vnm_tprint( 1, "  Setting up problem...\n");
00547                 /* Attempt to initialize and do an initial refinement of the mesh data.  The mesh data
00548                  * will be stored in the Vfetk object fetk, which contains the appropriate geometry
00549                  * manager (Gem) object and Vcsm object describing the mesh structure.  The mesh will
00550                  * either be loaded from an external source or generated from scratch. */
00551                 if (initFE(i, nosh, feparm, pbeparm, pbe, alist, fetk) != VRC_SUCCESS) {
00552                     Vnm_tprint( 2, "Error setting up FE calculation!\n");
00553                     VJMPERR1(0);
00554                 }
00555 
00556                     /* Print problem parameters */
00557                     printFEPARM(i, nosh, feparm, fetk);
00558                 printPBEPARM(pbeparm);
00559 
00560                 /* Refine mesh - this continues to run the AM_markRefine procedure already run
00561                  * in initFE() to arrive at some initial refinement, but does checks of the
00562                  * simplices so that it refines until the error or size tolerances are reached.
00563                  * Once this is done, we have a mesh that has been refined to the point where
00564                  * we can attempt to solve - further refinement may be needed in the loop
00565                  * below. */
00566                 if (!preRefineFE(i, feparm, fetk)) {
00567                     Vnm_tprint( 2, "Error pre-refining mesh!\n");
00568                     VJMPERR1(0);
00569                 }
00570 
00571                 /* Solve-estimate-refine */
00572                 Vnm_tprint(2, "\n\nWARNING!  DO NOT EXPECT PERFORMANCE OUT OF THE APBS/FEtk\n");
00573                 Vnm_tprint(2, "INTERFACE AT THIS TIME.  THE FINITE ELEMENT SOLVER IS\n");
00574                 Vnm_tprint(2, "CURRENTLY NOT OPTIMIZED FOR THE PB EQUATION.  IF YOU WANT\n");
00575                 Vnm_tprint(2, "PERFORMANCE, PLEASE USE THE MULTIGRID-BASED METHODS, E.G.\n");
00576                 Vnm_tprint(2, "MG-AUTO, MG-PARA, and MG-MANUAL (SEE DOCS.)\n\n");
00577                 Vnm_tprint(1, "  Beginning solve-estimate-refine cycle:\n");
00578 
00579                 for (isolve=0; isolve<feparm->maxsolve; isolve++) {
00580                     Vnm_tprint(1, "    Solve #%d...\n", isolve);
00581 
00582                     /* Attempt to solve the mesh by using one of MC's solver types. */
00583                     if (!solveFE(i, pbeparm, feparm, fetk)) {
00584                         Vnm_tprint(2, "ERROR SOLVING EQUATION!\n");
00585                         VJMPERR1(0);
00586                     }
00587 
00588                     /* Calculate the total electrostatic energy. */
00589                     if (!energyFE(nosh, i, fetk, &(nenergy[i]),
00590                                   &(totEnergy[i]), &(qfEnergy[i]),
00591                                   &(qmEnergy[i]), &(dielEnergy[i]))) {
00592                         Vnm_tprint(2, "ERROR SOLVING EQUATION!\n");
00593                         VJMPERR1(0);
00594                     }
00595 
00596                     /* We're not going to refine if we've hit the max number
00597                         * of solves */
00598                     if (isolve < (feparm->maxsolve)-1) {
00599                         /* Do a final error estimation and mesh refinement. */
00600                         if (!postRefineFE(i, feparm, fetk)) {
00601                             break;
00602                     }
00603                     }
00604                     bytesTotal = Vmem_bytesTotal();
00605                     highWater = Vmem_highWaterTotal();
00606                     Vnm_tprint(1, "      Current memory use:     %g MB\n",
00607                                ((double)bytesTotal/(1024.)/(1024.)));
00608                     Vnm_tprint(1, "      High-water memory use:  %g MB\n",
00609                                ((double)highWater/(1024.)/(1024.)));
00610                 }
00611 
00612                     Vnm_tprint(1, "  Writing FEM data to files.\n");
00613 
00614                 /* Save data. */
00615                 if (!writedataFE(rank, nosh, pbeparm, fetk[i])) {
00616                     Vnm_tprint(2, "  Error while writing FEM data!\n");
```

```
00617                 }
00618 #else /* ifdef HAVE_MC_H */
00619                     Vnm_print(2, "Error!  APBS not compiled with FEtk!\n");
00620                 exit(2);
00621 #endif /* ifdef HAVE_MC_H */
00622                 break;
00623 
00624             /* Do an apolar calculation */
00625             case NCT_APOL:
00626                 /* Copied from NCT_MG. See the note above (top of loop) for
00627                    information about this loop.
00628                 */
00629                 for (k=0; k<nosh->napol; k++) {
00630                     if (nosh->apol2calc[k] >= i) {
00631                         break;
00632                     }
00633                 }
00634 
00635                 if (Vstring_strcasecmp(nosh->apolname[k], "") == 0) {
00636                     Vnm_tprint( 1, "CALCULATION #%d: APOLAR\n", i+1);
00637                 } else {
00638                     Vnm_tprint( 1, "CALCULATION #%d (%s): APOLAR\n",
00639                                 i+1, nosh->apolname[k]);
00640                 }
00641 
00642                 apolparm = nosh->calc[i]->apolparm;
00643                 // poor man's execution timer.
00644                 ts = clock();
00645                 rc = initAPOL(nosh, mem, param, apolparm, &(nforce[i]), &(atomForce[i]),
00646                       alist[(apolparm->molid)-1]);
00647                 Vnm_print(0, "initAPOL: Time elapsed: %f\n", ((double)clock() - ts) / CLOCKS_PER_SEC);
00648                 if(rc == 0) {
00649                     Vnm_tprint(2, "Error calculating apolar solvation quantities!\n");
00650                     VJMPERR1(0);
00651                 }
00652                 break;
00653 
00654         /* Boundary Element (tabi) */
00655             case NCT_BEM:
00656 #ifdef ENABLE_BEM
00657                 /* What is this?  This seems like a very awkward way to find
00658                 the right ELEC statement... */
00659                 for (k=0; k<nosh->nelec; k++) {
00660                     if (nosh->elec2calc[k] >= i) {
00661                         break;
00662                     }
00663                 }
00664                 if (Vstring_strcasecmp(nosh->elecname[k], "") == 0) {
00665                     Vnm_tprint( 1, "CALCULATION #%d: BOUNDARY ELEMENT\n", i+1);
00666                 } else {
00667                     Vnm_tprint( 1, "CALCULATION #%d (%s): BOUNDARY ELEMENT\n",
00668                                 i+1, nosh->elecname[k]);
00669                 }
00670                 /* Useful local variables */
00671                 bemparm = nosh->calc[i]->bemparm;
00672                 pbeparm = nosh->calc[i]->pbeparm;
00673 
00674                 /* Set up problem */
00675                 Vnm_tprint( 1, "  Setting up problem...\n");
00676 
00677                 if (!initBEM(i,nosh, bemparm, pbeparm, pbe)) {
00678                     Vnm_tprint( 2, "Error setting up BEM calculation!\n");
00679                     VJMPERR1(0);
00680                 }
00681 
00682                 /* Print problem parameters */
00683                 printBEMPARM(bemparm);
00684                 printPBEPARM(pbeparm);
00685 
00686                 /* Solve PDE */
00687                 if (solveBEM(alist, nosh, pbeparm, bemparm, bemparm->type) != 1) {
00688                     Vnm_tprint(2, "Error solving PDE!\n");
00689                     VJMPERR1(0);
00690                 }
00691 
00692                 /* Write out energies */
00693                 energyBEM(nosh, i,
00694                         &(nenergy[i]), &(totEnergy[i]), &(qfEnergy[i]),
00695                         &(qmEnergy[i]), &(dielEnergy[i]));
00696 
00697                 /* Write out forces */
```

```
00698                 forceBEM(nosh, pbeparm, bemparm, &(nforce[i]),
00699                         &(atomForce[i]), alist);
00700
00701                 /* Write out data folks might want */
00702                 writedataBEM(rank, nosh, pbeparm);
00703
00704                 /* Write matrix */
00705                 writematBEM(rank, nosh, pbeparm);
00706
00707                 /* If needed, cache atom energies */
00708                 nenergy[i] = 0;
00709                 if ((pbeparm->calcenergy == PCE_COMPS) && (outputformat != OUTPUT_NULL)){
00710                     storeAtomEnergy(pmg[i], i, &(atomEnergy[i]), &(nenergy[i]));
00711                 }
00712
00713                 fflush(stdout);
00714                 fflush(stderr);
00715 #else /* ifdef ENABLE_BEM */
00716                     Vnm_print(2, "Error!  APBS not compiled with BEM!\n");
00717                 exit(2);
00718 #endif
00719                 break;
00720
00721    /* geometric flow */
00722             case NCT_GEOFLOW:
00723 #ifdef ENABLE_GEOFLOW
00724                 /* What is this?  This seems like a very awkward way to find
00725                 the right ELEC statement... */
00726                 for (k=0; k<nosh->nelec; k++) {
00727                     if (nosh->elec2calc[k] >= i) {
00728                         break;
00729                     }
00730                 }
00731                 if (Vstring_strcasecmp(nosh->elecname[k], "") == 0) {
00732                     Vnm_tprint( 1, "CALCULATION #%d: GEOMETRIC FLOW\n", i+1);
00733                 } else {
00734                     Vnm_tprint( 1, "CALCULATION #%d (%s): GEOMETRIC FLOW\n",
00735                                 i+1, nosh->elecname[k]);
00736                 }
00737                 /* Useful local variables */
00738                 geoflowparm = nosh->calc[i]->geoflowparm;
00739                 apolparm = nosh->calc[i]->apolparm;
00740                 pbeparm = nosh->calc[i]->pbeparm;
00741
00742                 /* Set up problem */
00743                 Vnm_tprint( 1, "  Setting up problem...\n");
00744
00745
00746                 /* Solve PDE */
00747                 if (solveGeometricFlow(alist, nosh, pbeparm, apolparm, geoflowparm) != 1) {
00748                     Vnm_tprint(2, "Error solving GEOFLOW!\n");
00749                     VJMPERR1(0);
00750                 }
00751
00752                 fflush(stdout);
00753                 fflush(stderr);
00754                 break;
00755 #else /* ifdef ENABLE_GEOFLOW*/
00756                     Vnm_print(2, "Error!  APBS not compiled with GEOFLOW!\n");
00757                 exit(2);
00758 #endif
00759
00760            /* Poisson-boltzmann analytical method */
00761             case NCT_PBAM:
00762 #ifdef ENABLE_PBAM
00763                 /* What is this?  This seems like a very awkward way to find
00764                 the right ELEC statement... */
00765                 //Vnm_tprint( 1, "Made it to start\n");
00766                 for (k=0; k<nosh->nelec; k++) {
00767                     if (nosh->elec2calc[k] >= i) {
00768                         break;
00769                     }
00770                 }
00771                 if (Vstring_strcasecmp(nosh->elecname[k], "") == 0) {
00772                     Vnm_tprint( 1, "CALCULATION #%d: PBAM\n", i+1);
00773                 } else {
00774                     Vnm_tprint( 1, "CALCULATION #%d (%s): PBAM\n",
00775                                 i+1, nosh->elecname[k]);
00776                 }
00777                 /* Useful local variables */
00778                 pbamparm = nosh->calc[i]->pbamparm;
```

```
00779                 pbeparm = nosh->calc[i]->pbeparm;
00780
00781                 /* Set up problem */
00782                 Vnm_tprint( 1, "  Setting up problem...\n");
00783
00784                 /* Solve LPBE with PBAM method */
00785     if (solvePBAM(alist, nosh, pbeparm, pbamparm) != 1) {
00786                     Vnm_tprint(2, "Error solving PBAM!\n");
00787                     VJMPERR1(0);
00788                 }
00789
00790                 fflush(stdout);
00791                 fflush(stderr);
00792                 break;
00793 #else /* ifdef ENABLE_PBAM*/
00794                     Vnm_print(2, "Error!  APBS not compiled with PBAM!\n");
00795                 exit(2);
00796 #endif
00797
00798             case NCT_PBSAM:
00799 #ifdef ENABLE_PBSAM
00800                 Vnm_tprint( 1, "Made it to start\n");
00801                 for (k=0; k<nosh->nelec; k++) {
00802                     if (nosh->elec2calc[k] >= i) {
00803                         break;
00804                     }
00805                 }
00806                 if (Vstring_strcasecmp(nosh->elecname[k], "") == 0) {
00807                     Vnm_tprint( 1, "CALCULATION #%d: PBSAM\n", i+1);
00808                 } else {
00809                     Vnm_tprint( 1, "CALCULATION #%d (%s): PBSAM\n",
00810                                 i+1, nosh->elecname[k]);
00811                 }
00812                 /* Useful local variables */
00813                 pbamparm = nosh->calc[i]->pbamparm;
00814                 pbsamparm = nosh->calc[i]->pbsamparm;
00815                 pbeparm = nosh->calc[i]->pbeparm;
00816
00817                 /* Set up problem */
00818                 Vnm_tprint( 1, "  Setting up problem...\n");
00819
00820
00821                 /* Solve LPBE with PBSAM method */
00822                 if (solvePBSAM(alist, nosh, pbeparm, pbamparm, pbsamparm) != 1) {
00823                     Vnm_tprint(2, "Error solving PBSAM!\n");
00824                     VJMPERR1(0);
00825                 }
00826
00827                 fflush(stdout);
00828                 fflush(stderr);
00829                 break;
00830 #else /* ifdef ENABLE_PBSAM*/
00831                     Vnm_print(2, "Error!  APBS not compiled with PBSAM!\n");
00832                 exit(2);
00833 #endif
00834
00835             default:
00836                 Vnm_tprint(2, "  Unknown calculation type (%d)!\n", nosh->calc[i]->calctype);
00837                 exit(2);
00838                 break;
00839             }
00840
00841     }
00842
00843     //Clear out the parameter file memory
00844     if(param != VNULL) Vparam_dtor(&param);
00845
00846     /* *************** HANDLE PRINT STATEMENTS ****************** */
00847     if (nosh->nprint > 0) {
00848         Vnm_tprint( 1, "----------------------------------------\n");
00849         Vnm_tprint( 1, "PRINT STATEMENTS\n");
00850     }
00851     for (i=0; i<nosh->nprint; i++) {
00852         /* Print energy */
00853         if (nosh->printwhat[i] == NPT_ENERGY) {
00854             printEnergy(com, nosh, totEnergy, i);
00855             /* Print force */
00856         } else if (nosh->printwhat[i] == NPT_FORCE) {
00857             printForce(com, nosh, nforce, atomForce, i);
00858         } else if (nosh->printwhat[i] == NPT_ELECENERGY) {
00859             printElecEnergy(com, nosh, totEnergy, i);
```

```
00860         } else if (nosh->printwhat[i] == NPT_ELECFORCE) {
00861             printElecForce(com, nosh, nforce, atomForce, i);
00862         } else if (nosh->printwhat[i] == NPT_APOLENERGY) {
00863             printApolEnergy(nosh, i);
00864         } else if (nosh->printwhat[i] == NPT_APOLFORCE) {
00865             printApolForce(com, nosh, nforce, atomForce, i);
00866         } else {
00867             Vnm_tprint( 2, "Undefined PRINT keyword!\n");
00868             break;
00869         }
00870     }
00871     Vnm_tprint( 1, "----------------------------------------\n");
00872
00873     /* *************** HANDLE LOGGING ********************** */
00874
00875     if (outputformat == OUTPUT_FLAT) {
00876         Vnm_tprint(2," Writing data to flat file %s...\n\n", output_path);
00877         writedataFlat(nosh, com, output_path, totEnergy, qfEnergy, qmEnergy,
00878                       dielEnergy, nenergy, atomEnergy, nforce, atomForce);
00879     }
00880
00881     /* Destroy energy arrays if they still exist */
00882
00883     for (i=0; i<nosh->ncalc; i++) {
00884         if (nenergy[i] > 0) Vmem_free(mem, nenergy[i], sizeof(double),
00885                                       (void **)&(atomEnergy[i]));
00886     }
00887
00888     /* *************** GARBAGE COLLECTION ****************** */
00889
00890     Vnm_tprint( 1, "CLEANING UP AND SHUTTING DOWN...\n");
00891     /* Clean up APBS structures */
00892     killForce(mem, nosh, nforce, atomForce);
00893     killEnergy();
00894     killMG(nosh, pbe, pmgp, pmg);
00895 #ifdef HAVE_MC_H
00896     killFE(nosh, pbe, fetk, gm);
00897 #endif
00898     killChargeMaps(nosh, chargeMap);
00899     killKappaMaps(nosh, kappaMap);
00900     killDielMaps(nosh, dielXMap, dielYMap, dielZMap);
00901     killMolecules(nosh, alist);
00902     NOsh_dtor(&nosh);
00903
00904     /* Memory statistics */
00905     bytesTotal = Vmem_bytesTotal();
00906     highWater = Vmem_highWaterTotal();
00907     Vnm_tprint( 1, "Final memory usage:  %4.3f MB total, %4.3f MB high water\n",
00908                 (double)(bytesTotal)/(1024.*1024.),
00909                 (double)(highWater)/(1024.*1024.));
00910
00911     /* Clean up MALOC structures */
00912     Vcom_dtor(&com);
00913     Vmem_dtor(&mem);
00914
00915     /* And now it's time to so "so long"... */
00916     Vnm_tprint(1, "\n\n");
00917     Vnm_tprint( 1, "Thanks for using APBS!\n\n");
00918
00919 #if defined(DEBUG_MAC_OSX_OCL)
00920     mets_(&mbeg, "Main Program CL");
00921 #endif
00922 #if defined(DEBUG_MAC_OSX_STANDARD)
00923     mets_(&mbeg, "Main Program Standard");
00924 #endif
00925
00926     /* This should be last */
00927     Vnm_tstop(APBS_TIMER_WALL_CLOCK, "APBS WALL CLOCK");
00928     Vnm_flush(1);
00929     Vnm_flush(2);
00930     Vcom_finalize();
00931
00932     fflush(NULL);
00933
00934     return 0;
00935
00936     VERROR1:
00937     Vcom_finalize();
00938     Vcom_dtor(&com);
00939     Vmem_dtor(&mem);
00940     return APBSRC;
```

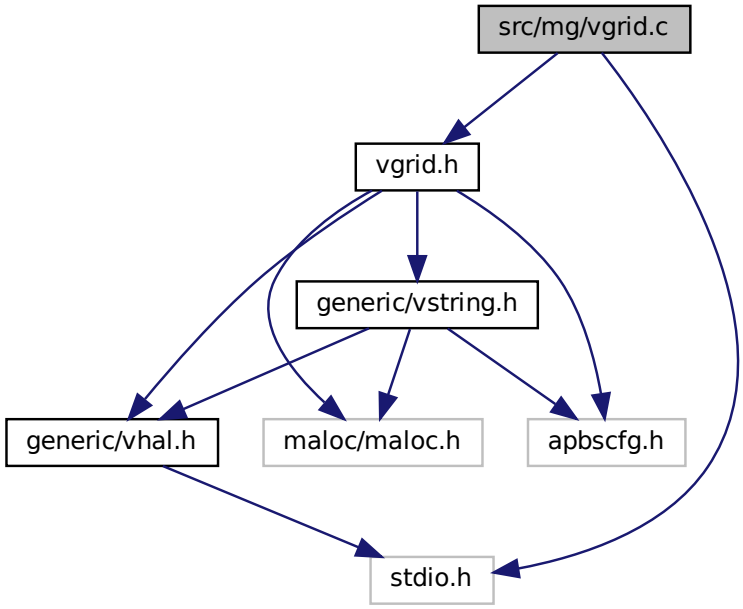
```
00941 }
```

9.91 src/mg/vgrid.c File Reference

Class Vgrid methods.

```
#include "vgrid.h"
#include <stdio.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 int Vgrid_readDXBIN (Vgrid *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname)

  Read in binary 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_writeDXBIN (Vgrid *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname, char *title, double *pvec)

  Write out the binary 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:
- VPUBLIC double Vgrid_normL2 (Vgrid *thee)

  Get the L_2 norm of the data. This returns the integral:
- VPUBLIC double Vgrid_seminormH1 (Vgrid *thee)

  Get the H_1 semi-norm of the data. This returns the integral:
- VPUBLIC double Vgrid_normH1 (Vgrid *thee)

  Get the H_1 norm (or energy norm) of the data. This returns the integral:
- VPUBLIC double Vgrid_normLinf (Vgrid *thee)

  Get the L_∞ norm of the data. This returns the integral:

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

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) 2010-2020 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) 2002-2020, 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 Vgrid_writeGZ()

```
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 1011 of file vgrid.c.

9.92 vgrid.c

```
00001
00057 #include "vgrid.h"
00058 #include <stdio.h>
00059
00060 VEMBED(rcsid="$Id$")
00061
00062 #if !defined(VINLINE_VGRID)
00063     VPUBLIC unsigned long int Vgrid_memChk(Vgrid *thee) {
00064         return Vmem_bytes(thee->mem);
00065     }
00066 #endif
00067 #define IJK(i,j,k)  (((k)*(nx)*(ny))+((j)*(nx))+(i))
00068
00069 #if defined(_WIN32) && (_MSC_VER < 1800)
00070 #include <float.h>
00071 int isnan(double d)
00072 {
00073     return _isnan(d);
00074 }
00075 #endif
00076
00077 VPRIVATE char *MCwhiteChars = " =,;\t\n";
00078 VPRIVATE char *MCcommChars  = "#%";
00079 VPRIVATE double Vcompare;
00080 VPRIVATE char Vprecision[26];
00081
00082 /* ////////////////////////////////////////////////////////////////////////////
00083 // Routine:  Vgrid_ctor
00084 // Author:   Nathan Baker
00086 VPUBLIC Vgrid* Vgrid_ctor(int nx,
00087                           int ny,
00088                           int nz,
00089                           double hx,
00090                           double hy,
00091                           double hzed,
```

```
00092                           double xmin,
00093                           double ymin,
00094                           double zmin,
00095                           double *data
00096                          ) {
00097
00098     Vgrid *thee = VNULL;
00099
00100     thee = (Vgrid*)Vmem_malloc(VNULL, 1, sizeof(Vgrid));
00101     VASSERT(thee != VNULL);
00102     VASSERT(Vgrid_ctor2(thee, nx, ny, nz, hx, hy, hzed,
00103                   xmin, ymin, zmin, data));
00104
00105     return thee;
00106 }
00107
00108 /* ///////////////////////////////////////////////////////////////////////// */
00109 // Routine:  Vgrid_ctor2
00110 // Author:   Nathan Baker
00112 VPUBLIC int Vgrid_ctor2(Vgrid *thee, int nx, int ny, int nz,
00113                   double hx, double hy, double hzed,
00114                   double xmin, double ymin, double zmin,
00115                   double *data) {
00116
00117     if (thee == VNULL) return 0;
00118     thee->nx = nx;
00119     thee->ny = ny;
00120     thee->nz = nz;
00121     thee->hx = hx;
00122     thee->hy = hy;
00123     thee->hzed = hzed;
00124     thee->xmin = xmin;
00125     thee->xmax = xmin + (nx-1)*hx;
00126     thee->ymin = ymin;
00127     thee->ymax = ymin + (ny-1)*hy;
00128     thee->zmin = zmin;
00129     thee->zmax = zmin + (nz-1)*hzed;
00130     if (data == VNULL) {
00131         thee->ctordata = 0;
00132         thee->readdata = 0;
00133     } else {
00134         thee->ctordata = 1;
00135         thee->readdata = 0;
00136         thee->data = data;
00137     }
00138
00139     thee->mem = Vmem_ctor("APBS:VGRID");
00140
00141     Vcompare = pow(10,-1*(VGRID_DIGITS - 2));
00142     sprintf(Vprecision,"%%12.%de %%12.%de %%12.%de", VGRID_DIGITS,
00143             VGRID_DIGITS, VGRID_DIGITS);
00144
00145     return 1;
00146 }
00147
00148 /* ///////////////////////////////////////////////////////////////////////// */
00149 // Routine:  Vgrid_dtor
00150 // Author:   Nathan Baker
00152 VPUBLIC void Vgrid_dtor(Vgrid **thee) {
00153
00154     if ((*thee) != VNULL) {
00155         Vgrid_dtor2(*thee);
00156         Vmem_free(VNULL, 1, sizeof(Vgrid), (void **)thee);
00157         (*thee) = VNULL;
00158     }
00159 }
00160
00161 /* ///////////////////////////////////////////////////////////////////////// */
00162 // Routine:  Vgrid_dtor2
00163 // Author:   Nathan Baker
00165 VPUBLIC void Vgrid_dtor2(Vgrid *thee) {
00166
00167     if (thee->readdata) {
00168         Vmem_free(thee->mem, (thee->nx*thee->ny*thee->nz), sizeof(double),
00169           (void **)&(thee->data));
00170     }
00171     Vmem_dtor(&(thee->mem));
00172
00173 }
00174
00175 /* ///////////////////////////////////////////////////////////////////////// */
```

```
00176 // Routine:  Vgrid_value
00177 // Author:   Nathan Baker
00179 VPUBLIC int Vgrid_value(Vgrid *thee, double pt[3], double *value) {
00180
00181     int nx, ny, nz;
00182     size_t ihi, jhi, khi, ilo, jlo, klo;
00183     double hx, hy, hzed, xmin, ymin, zmin, ifloat, jfloat, kfloat;
00184     double xmax, ymax, zmax;
00185     double u, dx, dy, dz;
00186
00187     if (thee == VNULL) {
00188         Vnm_print(2, "Vgrid_value:  Error -- got VNULL thee!\n");
00189         VASSERT(0);
00190     }
00191     if (!(thee->ctordata || thee->readdata)) {
00192         Vnm_print(2, "Vgrid_value:  Error -- no data available!\n");
00193         VASSERT(0);
00194     }
00195
00196     nx = thee->nx;
00197     ny = thee->ny;
00198     nz = thee->nz;
00199     hx = thee->hx;
00200     hy = thee->hy;
00201     hzed = thee->hzed;
00202     xmin = thee->xmin;
00203     ymin = thee->ymin;
00204     zmin = thee->zmin;
00205     xmax = thee->xmax;
00206     ymax = thee->ymax;
00207     zmax = thee->zmax;
00208
00209     u = 0;
00210
00211     ifloat = (pt[0] - xmin)/hx;
00212     jfloat = (pt[1] - ymin)/hy;
00213     kfloat = (pt[2] - zmin)/hzed;
00214
00215     ihi = (int)ceil(ifloat);
00216     jhi = (int)ceil(jfloat);
00217     khi = (int)ceil(kfloat);
00218     ilo = (int)floor(ifloat);
00219     jlo = (int)floor(jfloat);
00220     klo = (int)floor(kfloat);
00221     if (VABS(pt[0] - xmin) < Vcompare) ilo = 0;
00222     if (VABS(pt[1] - ymin) < Vcompare) jlo = 0;
00223     if (VABS(pt[2] - zmin) < Vcompare) klo = 0;
00224     if (VABS(pt[0] - xmax) < Vcompare) ihi = nx-1;
00225     if (VABS(pt[1] - ymax) < Vcompare) jhi = ny-1;
00226     if (VABS(pt[2] - zmax) < Vcompare) khi = nz-1;
00227
00228     /* See if we're on the mesh */
00229     /*the condions starting with ilo>=0 seem unnecessary since they are of type size_t*/
00230     if ((ihi<nx) && (jhi<ny) && (khi<nz) /*&&
00231         (ilo>=0) && (jlo>=0) && (klo>=0)*/) {
00232
00233         dx = ifloat - (double)(ilo);
00234         dy = jfloat - (double)(jlo);
00235         dz = kfloat - (double)(klo);
00236         u = dx      *dy      *dz      *(thee->data[IJK(ihi,jhi,khi)])
00237           + dx      *(1.0-dy)*dz      *(thee->data[IJK(ihi,jlo,khi)])
00238           + dx      *dy      *(1.0-dz)*(thee->data[IJK(ihi,jhi,klo)])
00239           + dx      *(1.0-dy)*(1.0-dz)*(thee->data[IJK(ihi,jlo,klo)])
00240           + (1.0-dx)*dy      *dz      *(thee->data[IJK(ilo,jhi,khi)])
00241           + (1.0-dx)*(1.0-dy)*dz      *(thee->data[IJK(ilo,jlo,khi)])
00242           + (1.0-dx)*dy      *(1.0-dz)*(thee->data[IJK(ilo,jhi,klo)])
00243           + (1.0-dx)*(1.0-dy)*(1.0-dz)*(thee->data[IJK(ilo,jlo,klo)]);
00244
00245         *value = u;
00246
00247         if (isnan(u)) {
00248             Vnm_print(2, "Vgrid_value:  Got NaN!\n");
00249             Vnm_print(2, "Vgrid_value:  (x, y, z) = (%4.3f, %4.3f, %4.3f)\n",
00250                       pt[0], pt[1], pt[2]);
00251             Vnm_print(2, "Vgrid_value:  (ihi, jhi, khi) = (%d, %d, %d)\n",
00252                       ihi, jhi, khi);
00253             Vnm_print(2, "Vgrid_value:  (ilo, jlo, klo) = (%d, %d, %d)\n",
00254                       ilo, jlo, klo);
00255             Vnm_print(2, "Vgrid_value:  (nx, ny, nz) = (%d, %d, %d)\n",
00256                       nx, ny, nz);
00257             Vnm_print(2, "Vgrid_value:  (dx, dy, dz) = (%4.3f, %4.3f, %4.3f)\n",
```

```
00258                   dx, dy, dz);
00259           Vnm_print(2, "Vgrid_value:  data[IJK(ihi,jhi,khi)] = %g\n",
00260                   thee->data[IJK(ihi,jhi,khi)]);
00261           Vnm_print(2, "Vgrid_value:  data[IJK(ihi,jlo,khi)] = %g\n",
00262                   thee->data[IJK(ihi,jlo,khi)]);
00263           Vnm_print(2, "Vgrid_value:  data[IJK(ihi,jhi,klo)] = %g\n",
00264                   thee->data[IJK(ihi,jhi,klo)]);
00265           Vnm_print(2, "Vgrid_value:  data[IJK(ihi,jlo,klo)] = %g\n",
00266                   thee->data[IJK(ihi,jlo,klo)]);
00267           Vnm_print(2, "Vgrid_value:  data[IJK(ilo,jhi,khi)] = %g\n",
00268                   thee->data[IJK(ilo,jhi,khi)]);
00269           Vnm_print(2, "Vgrid_value:  data[IJK(ilo,jlo,khi)] = %g\n",
00270                   thee->data[IJK(ilo,jlo,khi)]);
00271           Vnm_print(2, "Vgrid_value:  data[IJK(ilo,jhi,klo)] = %g\n",
00272                   thee->data[IJK(ilo,jhi,klo)]);
00273           Vnm_print(2, "Vgrid_value:  data[IJK(ilo,jlo,klo)] = %g\n",
00274                   thee->data[IJK(ilo,jlo,klo)]);
00275       }
00276       return 1;
00277
00278   } else {
00279
00280       *value = 0;
00281       return 0;
00282
00283   }
00284
00285   return 0;
00286
00287 }
00288
00289 /* ///////////////////////////////////////////////////////////////////////////
00290 // Routine:  Vgrid_curvature
00291 //
00292 //   Notes:  cflag=0 ==> Reduced Maximal Curvature
00293 //           cflag=1 ==> Mean Curvature (Laplace)
00294 //           cflag=2 ==> Gauss Curvature
00295 //           cflag=3 ==> True Maximal Curvature
00296 //
00297 // Authors:  Stephen Bond and Nathan Baker
00299 VPUBLIC int Vgrid_curvature(Vgrid *thee, double pt[3], int cflag,
00300   double *value) {
00301
00302     double hx, hy, hzed, curv;
00303     double dxx, dyy, dzz;
00304     double uleft, umid, uright, testpt[3];
00305
00306     if (thee == VNULL) {
00307         Vnm_print(2, "Vgrid_curvature:  Error -- got VNULL thee!\n");
00308         VASSERT(0);
00309     }
00310     if (!(thee->ctordata || thee->readdata)) {
00311         Vnm_print(2, "Vgrid_curvature:  Error -- no data available!\n");
00312         VASSERT(0);
00313     }
00314
00315     hx = thee->hx;
00316     hy = thee->hy;
00317     hzed = thee->hzed;
00318
00319     curv = 0.0;
00320
00321     testpt[0] = pt[0];
00322     testpt[1] = pt[1];
00323     testpt[2] = pt[2];
00324
00325     /* Compute 2nd derivative in the x-direction */
00326     VJMPERR1(Vgrid_value( thee, testpt, &umid));
00327     testpt[0] = pt[0] - hx;
00328     VJMPERR1(Vgrid_value( thee, testpt, &uleft));
00329     testpt[0] = pt[0] + hx;
00330     VJMPERR1(Vgrid_value( thee, testpt, &uright));
00331     testpt[0] = pt[0];
00332
00333     dxx = (uright - 2*umid + uleft)/(hx*hx);
00334
00335     /* Compute 2nd derivative in the y-direction */
00336     VJMPERR1(Vgrid_value( thee, testpt, &umid));
00337     testpt[1] = pt[1] - hy;
00338     VJMPERR1(Vgrid_value( thee, testpt, &uleft));
00339     testpt[1] = pt[1] + hy;
```

```
00340     VJMPERR1(Vgrid_value( thee, testpt, &uright));
00341     testpt[1] = pt[1];
00342
00343     dyy = (uright - 2*umid + uleft)/(hy*hy);
00344
00345     /* Compute 2nd derivative in the z-direction */
00346     VJMPERR1(Vgrid_value( thee, testpt, &umid));
00347     testpt[2] = pt[2] - hzed;
00348     VJMPERR1(Vgrid_value( thee, testpt, &uleft));
00349     testpt[2] = pt[2] + hzed;
00350     VJMPERR1(Vgrid_value( thee, testpt, &uright));
00351
00352     dzz = (uright - 2*umid + uleft)/(hzed*hzed);
00353
00354
00355     if ( cflag == 0 ) {
00356         curv = fabs(dxx);
00357         curv = ( curv > fabs(dyy) ) ? curv : fabs(dyy);
00358         curv = ( curv > fabs(dzz) ) ? curv : fabs(dzz);
00359     } else if ( cflag == 1 ) {
00360         curv = (dxx + dyy + dzz)/3.0;
00361     } else {
00362         Vnm_print(2, "Vgrid_curvature:  support for cflag = %d not available!\n", cflag);
00363         VASSERT( 0 ); /* Feature Not Coded Yet! */
00364     }
00365
00366     *value = curv;
00367     return 1;
00368
00369     VERROR1:
00370         return 0;
00371
00372 }
00373
00374 /* ///////////////////////////////////////////////////////////////////////////
00375 // Routine:  Vgrid_gradient
00376 //
00377 // Authors:  Nathan Baker and Stephen Bond
00379 VPUBLIC int Vgrid_gradient(Vgrid *thee, double pt[3], double grad[3]) {
00380
00381     double hx, hy, hzed;
00382     double uleft, umid, uright, testpt[3];
00383     int haveleft, haveright;
00384
00385     if (thee == VNULL) {
00386         Vnm_print(2, "Vgrid_gradient:  Error -- got VNULL thee!\n");
00387         VASSERT(0);
00388     }
00389     if (!(thee->ctordata || thee->readdata)) {
00390         Vnm_print(2, "Vgrid_gradient:  Error -- no data available!\n");
00391         VASSERT(0);
00392     }
00393
00394     hx = thee->hx;
00395     hy = thee->hy;
00396     hzed = thee->hzed;
00397
00398     /* Compute derivative in the x-direction */
00399     testpt[0] = pt[0];
00400     testpt[1] = pt[1];
00401     testpt[2] = pt[2];
00402     VJMPERR1( Vgrid_value( thee, testpt, &umid));
00403     testpt[0] = pt[0] - hx;
00404     if (Vgrid_value( thee, testpt, &uleft)) haveleft = 1;
00405     else haveleft = 0;
00406     testpt[0] = pt[0] + hx;
00407     if (Vgrid_value( thee, testpt, &uright)) haveright = 1;
00408     else haveright = 0;
00409     if (haveright && haveleft) grad[0] = (uright - uleft)/(2*hx);
00410     else if (haveright) grad[0] = (uright - umid)/hx;
00411     else if (haveleft) grad[0] = (umid - uleft)/hx;
00412     else VJMPERR1(0);
00413
00414     /* Compute derivative in the y-direction */
00415     testpt[0] = pt[0];
00416     testpt[1] = pt[1];
00417     testpt[2] = pt[2];
00418     VJMPERR1(Vgrid_value(thee, testpt, &umid));
00419     testpt[1] = pt[1] - hy;
00420     if (Vgrid_value( thee, testpt, &uleft)) haveleft = 1;
00421     else haveleft = 0;
```

```
00422     testpt[1] = pt[1] + hy;
00423     if (Vgrid_value( thee, testpt, &uright)) haveright = 1;
00424     else haveright = 0;
00425     if (haveright && haveleft) grad[1] = (uright - uleft)/(2*hy);
00426     else if (haveright) grad[1] = (uright - umid)/hy;
00427     else if (haveleft) grad[1] = (umid - uleft)/hy;
00428     else VJMPERR1(0);
00429
00430     /* Compute derivative in the z-direction */
00431     testpt[0] = pt[0];
00432     testpt[1] = pt[1];
00433     testpt[2] = pt[2];
00434     VJMPERR1(Vgrid_value(thee, testpt, &umid));
00435     testpt[2] = pt[2] - hzed;
00436     if (Vgrid_value( thee, testpt, &uleft)) haveleft = 1;
00437     else haveleft = 0;
00438     testpt[2] = pt[2] + hzed;
00439     if (Vgrid_value( thee, testpt, &uright)) haveright = 1;
00440     else haveright = 0;
00441     if (haveright && haveleft) grad[2] = (uright - uleft)/(2*hzed);
00442     else if (haveright) grad[2] = (uright - umid)/hzed;
00443     else if (haveleft) grad[2] = (umid - uleft)/hzed;
00444     else VJMPERR1(0);
00445
00446     return 1;
00447
00448     VERROR1:
00449         return 0;
00450
00451 }
00452
00453 /* ///////////////////////////////////////////////////////////////////////////
00454  // Routine:  Vgrid_readGZ
00455  //
00456  // Author:   David Gohara
00458 #ifdef HAVE_ZLIB
00459 #define off_t long
00460 #include "zlib.h"
00461 #endif
00462 VPUBLIC int Vgrid_readGZ(Vgrid *thee, const char *fname) {
00463
00464 #ifdef HAVE_ZLIB
00465     size_t i, j, k, u;
00466     size_t len; // Temporary counter variable for loop conditionals
00467     size_t header, incr;
00468     double *temp;
00469     double dtmp1, dtmp2, dtmp3;
00470     gzFile infile;
00471     char line[VMAX_ARGLEN];
00472
00473     header = 0;
00474
00475     /* Check to see if the existing data is null and, if not, clear it out */
00476     if (thee->data != VNULL) {
00477         Vnm_print(1, "%s:  destroying existing data!\n", __func__);
00478         Vmem_free(thee->mem, thee->nx * thee->ny * thee->nz, sizeof(double),
00479                   (void **)&(thee->data));
00480         }
00481
00482     thee->readdata = 1;
00483     thee->ctordata = 0;
00484
00485     infile = gzopen(fname, "rb");
00486     if (infile == Z_NULL) {
00487         Vnm_print(2, "%s:  Problem opening compressed file %s\n", __func__, fname);
00488         return VRC_FAILURE;
00489     }
00490
00491     thee->hx = 0.0;
00492     thee->hy = 0.0;
00493     thee->hzed = 0.0;
00494
00495     //read data here
00496     while (header < 7) {
00497         if(gzgets(infile, line, VMAX_ARGLEN) == Z_NULL){
00498             return VRC_FAILURE;
00499         }
00500
00501         // Skip comments and newlines
00502         if(strncmp(line, "#", 1) == 0) continue;
00503         if(line[0] == '\n') continue;
```

```
00504
00505         switch (header) {
00506             case 0:
00507                 sscanf(line, "object 1 class gridpositions counts %d %d %d",
00508                       &(thee->nx),&(thee->ny),&(thee->nz));
00509                 break;
00510             case 1:
00511                 sscanf(line, "origin %lf %lf %lf",
00512                       &(thee->xmin),&(thee->ymin),&(thee->zmin));
00513                 break;
00514             case 2:
00515             case 3:
00516             case 4:
00517                 sscanf(line, "delta %lf %lf %lf",&dtmp1,&dtmp2,&dtmp3);
00518                 thee->hx += dtmp1;
00519                 thee->hy += dtmp2;
00520                 thee->hzed += dtmp3;
00521                 break;
00522             default:
00523                 break;
00524         }
00525
00526         header++;
00527     }
00528
00529     /* Allocate space for the data */
00530     Vnm_print(0, "%s:  allocating %d x %d x %d doubles for storage\n",
00531         __func__, thee->nx, thee->ny, thee->nz);
00532     len = thee->nx * thee->ny * thee->nz;
00533
00534     thee->data = VNULL;
00535     thee->data = Vmem_malloc(thee->mem, len, sizeof(double));
00536     if (thee->data == VNULL) {
00537         Vnm_print(2, "%s:  Unable to allocate space for data!\n", __func__);
00538         return 0;
00539     }
00540
00541     /* Allocate a temporary buffer to store the compressed
00542      * data into (column major order). Add 2 to ensure the buffer is
00543      * big enough to take extra data on the final read loop.
00544      */
00545     temp = (double *)malloc(len * (2 * sizeof(double)));
00546
00547     for (i = 0; i < len; i += 3){
00548         memset(&line, 0, sizeof(line));
00549         gzgets(infile, line, VMAX_ARGLEN);
00550         sscanf(line, "%lf %lf %lf", &temp[i], &temp[i+1], &temp[i+2]);
00551     }
00552
00553     /* Now move the data to row major order */
00554     incr = 0;
00555     for (i=0; i<thee->nx; i++) {
00556         for (j=0; j<thee->ny; j++) {
00557             for (k=0; k<thee->nz; k++) {
00558                 u = k*(thee->nx)*(thee->ny)+j*(thee->nx)+i;
00559                 (thee->data)[u] = temp[incr++];
00560             }
00561         }
00562     }
00563
00564     /* calculate grid maxima */
00565     thee->xmax = thee->xmin + (thee->nx-1)*thee->hx;
00566     thee->ymax = thee->ymin + (thee->ny-1)*thee->hy;
00567     thee->zmax = thee->zmin + (thee->nz-1)*thee->hzed;
00568
00569     /* Close off the socket */
00570     gzclose(infile);
00571     free(temp);
00572 #else
00573
00574     Vnm_print(0, "WARNING\n");
00575     Vnm_print(0, "Vgrid_readGZ:  gzip read/write support is disabled in this build\n");
00576     Vnm_print(0, "Vgrid_readGZ:  configure and compile without the --disable-zlib flag.\n");
00577     Vnm_print(0, "WARNING\n");
00578 #endif
00579     return VRC_SUCCESS;
00580 }
00581
00586 VPUBLIC int Vgrid_readDX(Vgrid *thee,
00587                          const char *iodev,
00588                          const char *iofmt,
```

```
00589                           const char *thost,
00590                           const char *fname
00591                         ) {
00592
00593     size_t i, j, k, itmp, u;
00594     double dtmp;
00595     char tok[VMAX_BUFSIZE];
00596     Vio *sock;
00597
00598     /* Check to see if the existing data is null and, if not, clear it out */
00599     if (thee->data != VNULL) {
00600         Vnm_print(1, "Vgrid_readDX:  destroying existing data!\n");
00601     Vmem_free(thee->mem, (thee->nx*thee->ny*thee->nz), sizeof(double),
00602           (void **)&(thee->data)); }
00603     thee->readdata = 1;
00604     thee->ctordata = 0;
00605
00606     /* Set up the virtual socket */
00607     sock = Vio_ctor(iodev,iofmt,thost,fname,"r");
00608     if (sock == VNULL) {
00609         Vnm_print(2, "Vgrid_readDX: Problem opening virtual socket %s\n",
00610           fname);
00611         return 0;
00612     }
00613     if (Vio_accept(sock, 0) < 0) {
00614         Vnm_print(2, "Vgrid_readDX: Problem accepting virtual socket %s\n",
00615           fname);
00616         return 0;
00617     }
00618
00619     Vio_setWhiteChars(sock, MCwhiteChars);
00620     Vio_setCommChars(sock, MCcommChars);
00621
00622     /* Read in the DX regular positions */
00623     /* Get "object" */
00624     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00625     VJMPERR1(!strcmp(tok, "object"));
00626     /* Get "1" */
00627     VJMPERR2(1 == Vio_scanf(sock, "%d", &itmp));
00628     /* Get "class" */
00629     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00630     VJMPERR1(!strcmp(tok, "class"));
00631     /* Get "gridpositions" */
00632     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00633     VJMPERR1(!strcmp(tok, "gridpositions"));
00634     /* Get "counts" */
00635     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00636     VJMPERR1(!strcmp(tok, "counts"));
00637     /* Get nx */
00638     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00639     VJMPERR1(1 == sscanf(tok, "%d", &(thee->nx)));
00640     /* Get ny */
00641     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00642     VJMPERR1(1 == sscanf(tok, "%d", &(thee->ny)));
00643     /* Get nz */
00644     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00645     VJMPERR1(1 == sscanf(tok, "%d", &(thee->nz)));
00646     Vnm_print(0, "Vgrid_readDX:  Grid dimensions %d x %d x %d grid\n",
00647      thee->nx, thee->ny, thee->nz);
00648     /* Get "origin" */
00649     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00650     VJMPERR1(!strcmp(tok, "origin"));
00651     /* Get xmin */
00652     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00653     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->xmin)));
00654     /* Get ymin */
00655     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00656     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->ymin)));
00657     /* Get zmin */
00658     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00659     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->zmin)));
00660     Vnm_print(0, "Vgrid_readDX:  Grid origin = (%g, %g, %g)\n",
00661       thee->xmin, thee->ymin, thee->zmin);
00662     /* Get "delta" */
00663     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00664     VJMPERR1(!strcmp(tok, "delta"));
00665     /* Get hx */
00666     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00667     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->hx)));
00668     /* Get 0.0 */
00669     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
```

```
00670     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00671     VJMPERR1(dtmp == 0.0);
00672     /* Get 0.0 */
00673     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00674     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00675     VJMPERR1(dtmp == 0.0);
00676     /* Get "delta" */
00677     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00678     VJMPERR1(!strcmp(tok, "delta"));
00679     /* Get 0.0 */
00680     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00681     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00682     VJMPERR1(dtmp == 0.0);
00683     /* Get hy */
00684     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00685     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->hy)));
00686     /* Get 0.0 */
00687     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00688     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00689     VJMPERR1(dtmp == 0.0);
00690     /* Get "delta" */
00691     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00692     VJMPERR1(!strcmp(tok, "delta"));
00693     /* Get 0.0 */
00694     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00695     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00696     VJMPERR1(dtmp == 0.0);
00697     /* Get 0.0 */
00698     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00699     VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00700     VJMPERR1(dtmp == 0.0);
00701     /* Get hz */
00702     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00703     VJMPERR1(1 == sscanf(tok, "%lf", &(thee->hzed)));
00704     Vnm_print(0, "Vgrid_readDX:  Grid spacings = (%g, %g, %g)\n",
00705       thee->hx, thee->hy, thee->hzed);
00706     /* Get "object" */
00707     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00708     VJMPERR1(!strcmp(tok, "object"));
00709     /* Get "2" */
00710     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00711     /* Get "class" */
00712     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00713     VJMPERR1(!strcmp(tok, "class"));
00714     /* Get "gridconnections" */
00715     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00716     VJMPERR1(!strcmp(tok, "gridconnections"));
00717     /* Get "counts" */
00718     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00719     VJMPERR1(!strcmp(tok, "counts"));
00720     /* Get the dimensions again */
00721     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00722     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00723     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00724     /* Get "object" */
00725     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00726     VJMPERR1(!strcmp(tok, "object"));
00727     /* Get # */
00728     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00729     /* Get "class" */
00730     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00731     VJMPERR1(!strcmp(tok, "class"));
00732     /* Get "array" */
00733     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00734     VJMPERR1(!strcmp(tok, "array"));
00735     /* Get "type" */
00736     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00737     VJMPERR1(!strcmp(tok, "type"));
00738     /* Get "double" */
00739     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00740     VJMPERR1(!strcmp(tok, "double"));
00741     /* Get "rank" */
00742     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00743     VJMPERR1(!strcmp(tok, "rank"));
00744     /* Get # */
00745     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00746     /* Get "items" */
00747     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00748     VJMPERR1(!strcmp(tok, "items"));
00749     /* Get # */
00750     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
```

```
00751     VJMPERR1(1 == sscanf(tok, "%lu", &itmp));
00752     u = (size_t)thee->nx * thee->ny * thee->nz;
00753     VJMPERR1(u == itmp);
00754     /* Get "data" */
00755     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00756     VJMPERR1(!strcmp(tok, "data"));
00757     /* Get "follows" */
00758     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00759     VJMPERR1(!strcmp(tok, "follows"));
00760
00761     /* Allocate space for the data */
00762     Vnm_print(0, "Vgrid_readDX:  allocating %d x %d x %d doubles for storage\n",
00763       thee->nx, thee->ny, thee->nz);
00764     thee->data = VNULL;
00765     thee->data = (double*)Vmem_malloc(thee->mem, u, sizeof(double));
00766     if (thee->data == VNULL) {
00767         Vnm_print(2, "Vgrid_readDX:  Unable to allocate space for data!\n");
00768         return 0;
00769     }
00770
00771     for (i=0; i<thee->nx; i++) {
00772         for (j=0; j<thee->ny; j++) {
00773             for (k=0; k<thee->nz; k++) {
00774                 u = k*(thee->nx)*(thee->ny)+j*(thee->nx)+i;
00775                 VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00776                 VJMPERR1(1 == sscanf(tok, "%lf", &dtmp));
00777                 (thee->data)[u] = dtmp;
00778             }
00779         }
00780     }
00781
00782     /* calculate grid maxima */
00783     thee->xmax = thee->xmin + (thee->nx-1)*thee->hx;
00784     thee->ymax = thee->ymin + (thee->ny-1)*thee->hy;
00785     thee->zmax = thee->zmin + (thee->nz-1)*thee->hzed;
00786
00787     /* Close off the socket */
00788     Vio_acceptFree(sock);
00789     Vio_dtor(&sock);
00790
00791     return 1;
00792
00793   VERROR1:
00794     Vio_dtor(&sock);
00795     Vnm_print(2, "Vgrid_readDX:  Format problem with input file <%s>\n",
00796       fname);
00797     return 0;
00798
00799   VERROR2:
00800     Vio_dtor(&sock);
00801     Vnm_print(2, "Vgrid_readDX:  I/O problem with input file <%s>\n",
00802       fname);
00803     return 0;
00804 }
00805
00810 VPUBLIC int Vgrid_readDXBIN(Vgrid *thee, const char *iodev, const char *iofmt,
00811                             const char *thost, const char *fname) {
00812
00813  size_t i, j, k, itmp, u;
00814  double dtmp, dtmp2;
00815  char tok[VMAX_BUFSIZE];
00816  int isBinary = 0;
00817  //Vio *sock;
00818
00819  /* Check to see if the existing data is null and, if not, clear it out */
00820  if (thee->data != VNULL) {
00821   Vnm_print(1, "Vgrid_readDXBIN: destroying existing data!\n");
00822  Vmem_free(thee->mem, (thee->nx*thee->ny*thee->nz), sizeof(double),
00823     (void **)&(thee->data)); }
00824  thee->readdata = 1;
00825  thee->ctordata = 0;
00826
00827  /*Opend file fd for binary reading*/
00828  FILE *fd = fopen(fname,"rb");
00829  if(fd == NULL){
00830   printf("Vgrid_readDXBIN: Problem opening file %s\n",fname);
00831   fclose(fd);
00832   return 0;
00833  }
00834
00835      /* Set up the virtual socket */
```

```
00836 //     sock = Vio_ctor(iodev,iofmt,thost,fname,"r");
00837 //     if (sock == VNULL) {
00838 //         Vnm_print(2, "Vgrid_readDX: Problem opening virtual socket %s\n",
00839 //           fname);
00840 //         return 0;
00841 //     }
00842 //     if (Vio_accept(sock, 0) < 0) {
00843 //         Vnm_print(2, "Vgrid_readDX: Problem accepting virtual socket %s\n",
00844 //           fname);
00845 //         return 0;
00846 //     }
00847 //
00848 //     Vio_setWhiteChars(sock, MCwhiteChars);
00849 //     Vio_setCommChars(sock, MCcommChars);
00850
00851  //skip comments
00852  do{
00853   fgets(tok, VMAX_BUFSIZE, fd);
00854  }
00855  while(tok[0]=='#');
00856
00857  //get counts
00858  if(sscanf(tok,"object 1 class gridpositions counts %i %i %i\n",&(thee->nx),&(thee->ny),&(thee->nz))!= 3){
00859   printf("Vgrid_readDXBIN: Failed to read dimensions.\n");
00860   fclose(fd);
00861   return 0;
00862  }
00863  printf("Vgrid_readDXBIN: Grid dimensions %d x %d x %d grid\n",thee->nx, thee->ny, thee->nz);
00864
00865      /* Read in the DX regular positions */
00866
00867   /* Get "object" */
00868 //     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00869 //     VJMPERR1(!strcmp(tok, "object"));
00870 //     /* Get "1" */
00871 //     VJMPERR2(1 == Vio_scanf(sock, "%d", &itmp));
00872 //     /* Get "class" */
00873 //     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00874 //     VJMPERR1(!strcmp(tok, "class"));
00875 //     /* Get "gridpositions" */
00876 //     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00877 //     VJMPERR1(!strcmp(tok, "gridpositions"));
00878 //     /* Get "counts" */
00879 //     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00880 //     VJMPERR1(!strcmp(tok, "counts"));
00881 //     /* Get nx */
00882 //     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00883 //     VJMPERR1(1 == sscanf(tok, "%d", &(thee->nx)));
00884 //     /* Get ny */
00885 //     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00886 //     VJMPERR1(1 == sscanf(tok, "%d", &(thee->ny)));
00887 //     /* Get nz */
00888 //     VJMPERR2(1 == Vio_scanf(sock, "%s", tok));
00889 //     VJMPERR1(1 == sscanf(tok, "%d", &(thee->nz)));
00890 //     Vnm_print(0, "Vgrid_readDX:  Grid dimensions %d x %d x %d grid\n",
00891 //      thee->nx, thee->ny, thee->nz);
00892
00893  if(fgets(tok,VMAX_BUFSIZE, fd) == NULL){
00894   printf("Vgrid_readDXBIN: unexpected end of file.\n");
00895   fclose(fd);
00896   return 0;
00897  }
00898  if(sscanf(tok,"origin %lf %lf %lf",&(thee->xmin),&(thee->ymin),&(thee->zmin))!=3){
00899   printf("Vgrid_readDXBIN: Failed to read origin cell data.\n");
00900   fclose(fd);
00901   return 0;
00902  }
00903  printf("Vgrid_readDXBIN: Grid origin = (%g %g %g)\n",thee->xmin, thee->ymin, thee->zmin);
00904
00905  //get Delta x
00906  if(fgets(tok,VMAX_BUFSIZE, fd) == NULL){
00907   printf("Vgrid_readDXBIN: unexpected end of file.\n");
00908   fclose(fd);
00909   return 0;
00910  }
00911  if(sscanf(tok,"delta %lf %lf %lf",&(thee->hx),&dtmp,&dtmp2)!=3){
00912   printf("Vgrid_readDXBIN: Failed to read delta x data.\n");
00913   fclose(fd);
00914   return 0;
00915  }
00916  //get Delta y
```

```
00917  if(fgets(tok,VMAX_BUFSIZE, fd) == NULL){
00918   printf("Vgrid_readDXBIN: Unexpected end of file\n");
00919   fclose(fd);
00920   return 0;
00921  }
00922  if(sscanf(tok,"delta %lf %lf %lf",&dtmp,&(thee->hy),&dtmp2)!=3){
00923   printf("Vgrid_readDXBIN: Failed to read delta y data.\n");
00924   fclose(fd);
00925   return 0;
00926  }
00927  //get Delta z
00928  if(fgets(tok,VMAX_BUFSIZE, fd) == NULL){
00929   printf("Vgrid_readDXBIN: Unexpected end of file.\n");
00930   fclose(fd);
00931   return 0;
00932  }
00933  if(sscanf(tok,"delta %lf %lf %lf",&dtmp,&dtmp2,&(thee->hzed))!=3){
00934   printf("Vgrid_readDXBIN: Failed to read delta z data.\n");
00935   fclose(fd);
00936   return 0;
00937  }
00938  printf("Vgrid_readDXBIN: Grid spacings = (%g, %g, %g)\n",thee->hx, thee->hy, thee->hzed);
00939
00940  //skip a line
00941  if(fgets(tok,VMAX_BUFSIZE, fd) == NULL){
00942   printf("Vgrid_readDXBIN: Unexpected end of file.\n");
00943   fclose(fd);
00944   return 0;
00945  }
00946
00947  //scan the buffer for the word binary
00948  if(fgets(tok,VMAX_BUFSIZE, fd) == NULL){
00949   printf("Vgrid_readDXBIN: Unexpected end of file.\n");
00950   fclose(fd);
00951   return 0;
00952  }
00953  if(strstr(tok,"binary")){
00954   isBinary = 1;
00955  }
00956  else{
00957   printf("Vgrid_readDXBIN: Binary tag not found. Will continue to try to read binary data.");
00958  }
00959
00960  u = (size_t)thee->nx * thee->ny * thee->nz;
00961  int tot = thee->nx * thee->ny * thee->nz;
00962
00963  /*Allocate space for the data*/
00964  printf("Vgrid_readDXBIN: allocating %d x %d x %d doubled for storage\n", thee->nx, thee->ny, thee->nz);
00965  thee->data = NULL;
00966  thee->data = (double *)malloc(tot*sizeof(double));
00967
00968  if(thee->data == NULL){
00969   printf("Vgrid_readDXBIN: Unable to allocate space for data!\n");
00970   fclose(fd);
00971   return 0;
00972  }
00973
00974  int counter = 0, r;
00975
00976  for (i=0; i<thee->nx; i++) {
00977   for (j=0; j<thee->ny; j++) {
00978    for (k=0; k<thee->nz; k++) {
00979     u = k*(thee->nx)*(thee->ny)+j*(thee->nx)+i;
00980     r = fread(&dtmp,sizeof(double),1,fd);
00981     (thee->data)[u] = dtmp;
00982     if(r!= 1){
00983      printf("Vgrid_readDXBIN: Failed to read doubles.\n");
00984      return 0;
00985     }
00986     counter++;
00987    }
00988   }
00989  }
00990
00991  if(counter!=tot){
00992   printf("Vgrid_readDXBIN: Read double = %d not equal to items = %d\n",counter, tot);
00993  }
00994
00995  /* calculate grid maxima */
00996  thee->xmax = thee->xmin + (thee->nx-1)*thee->hx;
00997  thee->ymax = thee->ymin + (thee->ny-1)*thee->hy;
```

```
00998  thee->zmax = thee->zmin + (thee->nz-1)*thee->hzed;
00999
01000  fclose(fd);
01001
01002  return 1;
01003 }
01004
01005
01006 /* ///////////////////////////////////////////////////////////////////////////
01007  // Routine:  Vgrid_writeGZ
01008  //
01009  // Author:   Nathan Baker
01011 VPUBLIC void Vgrid_writeGZ(Vgrid *thee, const char *iodev, const char *iofmt,
01012                            const char *thost, const char *fname, char *title, double *pvec) {
01013
01014 #ifdef HAVE_ZLIB
01015     double xmin, ymin, zmin, hx, hy, hzed;
01016
01017     int nx, ny, nz, nxPART, nyPART, nzPART;
01018     int usepart, gotit;
01019     size_t icol, i, j, k, u;
01020     double x, y, z, xminPART, yminPART, zminPART;
01021
01022     size_t txyz;
01023     double txmin, tymin, tzmin;
01024
01025     char header[8196];
01026     char footer[8196];
01027     char line[80];
01028     char newline[] = "\n";
01029     gzFile outfile;
01030     char precFormat[VMAX_BUFSIZE];
01031
01032     if (thee == VNULL) {
01033         Vnm_print(2, "Vgrid_writeGZ:  Error -- got VNULL thee!\n");
01034         VASSERT(0);
01035     }
01036     if (!(thee->ctordata || thee->readdata)) {
01037         Vnm_print(2, "Vgrid_writeGZ:  Error -- no data available!\n");
01038         VASSERT(0);
01039     }
01040
01041     hx = thee->hx;
01042     hy = thee->hy;
01043     hzed = thee->hzed;
01044     nx = thee->nx;
01045     ny = thee->ny;
01046     nz = thee->nz;
01047     xmin = thee->xmin;
01048     ymin = thee->ymin;
01049     zmin = thee->zmin;
01050
01051     if (pvec == VNULL) usepart = 0;
01052     else usepart = 1;
01053
01054     /* Set up the virtual socket */
01055     Vnm_print(0, "Vgrid_writeGZ:  Opening file...\n");
01056     outfile = gzopen(fname, "wb");
01057
01058     if (usepart) {
01059         /* Get the lower corner and number of grid points for the local
01060          * partition */
01061         xminPART = VLARGE;
01062         yminPART = VLARGE;
01063         zminPART = VLARGE;
01064         nxPART = 0;
01065         nyPART = 0;
01066         nzPART = 0;
01067         /* First, search for the lower corner */
01068         for (k=0; k<nz; k++) {
01069             z = k*hzed + zmin;
01070             for (j=0; j<ny; j++) {
01071                 y = j*hy + ymin;
01072                 for (i=0; i<nx; i++) {
01073                     x = i*hx + xmin;
01074                     if (pvec[IJK(i,j,k)] > 0.0) {
01075                         if (x < xminPART) xminPART = x;
01076                         if (y < yminPART) yminPART = y;
01077                         if (z < zminPART) zminPART = z;
01078                     }
01079                 }
```

```
01080             }
01081         }
01082         /* Now search for the number of grid points in the z direction */
01083         for (k=0; k<nz; k++) {
01084             gotit = 0;
01085             for (j=0; j<ny; j++) {
01086                 for (i=0; i<nx; i++) {
01087                     if (pvec[IJK(i,j,k)] > 0.0) {
01088                         gotit = 1;
01089                         break;
01090                     }
01091                 }
01092                 if (gotit) break;
01093             }
01094             if (gotit) nzPART++;
01095         }
01096         /* Now search for the number of grid points in the y direction */
01097         for (j=0; j<ny; j++) {
01098             gotit = 0;
01099             for (k=0; k<nz; k++) {
01100                 for (i=0; i<nx; i++) {
01101                     if (pvec[IJK(i,j,k)] > 0.0) {
01102                         gotit = 1;
01103                         break;
01104                     }
01105                 }
01106                 if (gotit) break;
01107             }
01108             if (gotit) nyPART++;
01109         }
01110         /* Now search for the number of grid points in the x direction */
01111         for (i=0; i<nx; i++) {
01112             gotit = 0;
01113             for (k=0; k<nz; k++) {
01114                 for (j=0; j<ny; j++) {
01115                     if (pvec[IJK(i,j,k)] > 0.0) {
01116                         gotit = 1;
01117                         break;
01118                     }
01119                 }
01120                 if (gotit) break;
01121             }
01122             if (gotit) nxPART++;
01123         }
01124 
01125         if ((nxPART != nx) || (nyPART != ny) || (nzPART != nz)) {
01126             Vnm_print(0, "Vgrid_writeGZ:  printing only subset of domain\n");
01127         }
01128 
01129         txyz = (nxPART*nyPART*nzPART);
01130         txmin = xminPART;
01131         tymin = yminPART;
01132         tzmin = zminPART;
01133 
01134     }else {
01135 
01136         txyz = (nx*ny*nz);
01137         txmin = xmin;
01138         tymin = ymin;
01139         tzmin = zmin;
01140 
01141     }
01142 
01143     /* Write off the title (if we're not XDR) */
01144     sprintf(header,
01145             "# Data from %s\n" \
01146             "# \n"       \
01147             "# %s\n"    \
01148             "# \n"       \
01149             "object 1 class gridpositions counts %i %i %i\n" \
01150             "origin %12.6e %12.6e %12.6e\n" \
01151             "delta %12.6e 0.000000e+00 0.000000e+00\n"  \
01152             "delta 0.000000e+00 %12.6e 0.000000e+00\n"  \
01153             "delta 0.000000e+00 0.000000e+00 %12.6e\n"  \
01154             "object 2 class gridconnections counts %i %i %i\n"\
01155             "object 3 class array type double rank 0 items %lu data follows\n",
01156             PACKAGE_STRING,title,nx,ny,nz,txmin,tymin,tzmin,
01157             hx,hy,hzed,nx,ny,nz,txyz);
01158     gzwrite(outfile, header, strlen(header)*sizeof(char));
01159 
01160     /* Now write the data */
```

```
01161     icol = 0;
01162     for (i=0; i<nx; i++) {
01163         for (j=0; j<ny; j++) {
01164             for (k=0; k<nz; k++) {
01165                 u = k*(nx)*(ny)+j*(nx)+i;
01166                 if (pvec[u] > 0.0) {
01167                     sprintf(line, "%12.6e ", thee->data[u]);
01168                     gzwrite(outfile, line, strlen(line)*sizeof(char));
01169                     icol++;
01170                     if (icol == 3) {
01171                         icol = 0;
01172                         gzwrite(outfile, newline, strlen(newline)*sizeof(char));
01173                     }
01174                 }
01175             }
01176         }
01177     }
01178     if(icol < 3){
01179         char newline[] = "\n";
01180         gzwrite(outfile, newline, strlen(newline)*sizeof(char));
01181     }
01182
01183     /* Create the field */
01184     sprintf(footer, "attribute \"dep\" string \"positions\"\n" \
01185             "object \"regular positions regular connections\" class field\n" \
01186             "component \"positions\" value 1\n" \
01187             "component \"connections\" value 2\n" \
01188             "component \"data\" value 3\n");
01189     gzwrite(outfile, footer, strlen(footer)*sizeof(char));
01190
01191     gzclose(outfile);
01192 #else
01193
01194     Vnm_print(0, "WARNING\n");
01195     Vnm_print(0, "Vgrid_readGZ:  gzip read/write support is disabled in this build\n");
01196     Vnm_print(0, "Vgrid_readGZ:  configure and compile without the --disable-zlib flag.\n");
01197     Vnm_print(0, "WARNING\n");
01198 #endif
01199 }
01200
01201 /* ///////////////////////////////////////////////////////////////////////////
01202 // Routine:  Vgrid_writeDX
01203 //
01204 // Author:   Nathan Baker
01206 VPUBLIC void Vgrid_writeDX(Vgrid *thee, const char *iodev, const char *iofmt,
01207   const char *thost, const char *fname, char *title, double *pvec) {
01208
01209     double xmin, ymin, zmin, hx, hy, hzed;
01210     int nx, ny, nz, nxPART, nyPART, nzPART;
01211     int usepart, gotit;
01212     size_t icol, i, j, k, u;
01213     double x, y, z, xminPART, yminPART, zminPART;
01214     Vio *sock;
01215     char precFormat[VMAX_BUFSIZE];
01216
01217     if (thee == VNULL) {
01218         Vnm_print(2, "Vgrid_writeDX:  Error -- got VNULL thee!\n");
01219         VASSERT(0);
01220     }
01221     if (!(thee->ctordata || thee->readdata)) {
01222         Vnm_print(2, "Vgrid_writeDX:  Error -- no data available!\n");
01223         VASSERT(0);
01224     }
01225
01226     hx = thee->hx;
01227     hy = thee->hy;
01228     hzed = thee->hzed;
01229     nx = thee->nx;
01230     ny = thee->ny;
01231     nz = thee->nz;
01232     xmin = thee->xmin;
01233     ymin = thee->ymin;
01234     zmin = thee->zmin;
01235
01236     if (pvec == VNULL) usepart = 0;
01237     else usepart = 1;
01238
01239     /* Set up the virtual socket */
01240     Vnm_print(0, "Vgrid_writeDX:  Opening virtual socket...\n");
01241     sock = Vio_ctor(iodev,iofmt,thost,fname,"w");
01242     if (sock == VNULL) {
```

```
01243         Vnm_print(2, "Vgrid_writeDX:  Problem opening virtual socket %s\n",
01244           fname);
01245         return;
01246     }
01247     if (Vio_connect(sock, 0) < 0) {
01248         Vnm_print(2, "Vgrid_writeDX: Problem connecting virtual socket %s\n",
01249           fname);
01250         return;
01251     }
01252 
01253     Vio_setWhiteChars(sock, MCwhiteChars);
01254     Vio_setCommChars(sock, MCcommChars);
01255 
01256     Vnm_print(0, "Vgrid_writeDX:  Writing to virtual socket...\n");
01257 
01258     if (usepart) {
01259         /* Get the lower corner and number of grid points for the local
01260          * partition */
01261         xminPART = VLARGE;
01262         yminPART = VLARGE;
01263         zminPART = VLARGE;
01264         nxPART = 0;
01265         nyPART = 0;
01266         nzPART = 0;
01267         /* First, search for the lower corner */
01268         for (k=0; k<nz; k++) {
01269             z = k*hzed + zmin;
01270             for (j=0; j<ny; j++) {
01271                 y = j*hy + ymin;
01272                 for (i=0; i<nx; i++) {
01273                     x = i*hx + xmin;
01274                     if (pvec[IJK(i,j,k)] > 0.0) {
01275                         if (x < xminPART) xminPART = x;
01276                         if (y < yminPART) yminPART = y;
01277                         if (z < zminPART) zminPART = z;
01278                     }
01279                 }
01280             }
01281         }
01282         /* Now search for the number of grid points in the z direction */
01283         for (k=0; k<nz; k++) {
01284             gotit = 0;
01285             for (j=0; j<ny; j++) {
01286                 for (i=0; i<nx; i++) {
01287                     if (pvec[IJK(i,j,k)] > 0.0) {
01288                         gotit = 1;
01289                         break;
01290                     }
01291                 }
01292                 if (gotit) break;
01293             }
01294             if (gotit) nzPART++;
01295         }
01296         /* Now search for the number of grid points in the y direction */
01297         for (j=0; j<ny; j++) {
01298             gotit = 0;
01299             for (k=0; k<nz; k++) {
01300                 for (i=0; i<nx; i++) {
01301                     if (pvec[IJK(i,j,k)] > 0.0) {
01302                         gotit = 1;
01303                         break;
01304                     }
01305                 }
01306                 if (gotit) break;
01307             }
01308             if (gotit) nyPART++;
01309         }
01310         /* Now search for the number of grid points in the x direction */
01311         for (i=0; i<nx; i++) {
01312             gotit = 0;
01313             for (k=0; k<nz; k++) {
01314                 for (j=0; j<ny; j++) {
01315                     if (pvec[IJK(i,j,k)] > 0.0) {
01316                         gotit = 1;
01317                         break;
01318                     }
01319                 }
01320                 if (gotit) break;
01321             }
01322             if (gotit) nxPART++;
01323         }
```

```
01324
01325         if ((nxPART != nx) || (nyPART != ny) || (nzPART != nz)) {
01326             Vnm_print(0, "Vgrid_writeDX:  printing only subset of domain\n");
01327         }
01328
01329
01330         /* Write off the title (if we're not XDR) */
01331         if (Vstring_strcasecmp(iofmt, "XDR") == 0) {
01332             Vnm_print(0, "Vgrid_writeDX:  Skipping comments for XDR format.\n");
01333         } else {
01334             Vnm_print(0, "Vgrid_writeDX:  Writing comments for %s format.\n",
01335               iofmt);
01336             Vio_printf(sock, "# Data from %s\n", PACKAGE_STRING);
01337             Vio_printf(sock, "# \n");
01338             Vio_printf(sock, "# %s\n", title);
01339             Vio_printf(sock, "# \n");
01340         }
01341
01342         /* Write off the DX regular positions */
01343         Vio_printf(sock, "object 1 class gridpositions counts %d %d %d\n",
01344           nxPART, nyPART, nzPART);
01345
01346         sprintf(precFormat, Vprecision, xminPART, yminPART, zminPART);
01347         Vio_printf(sock, "origin %s\n", precFormat);
01348         sprintf(precFormat, Vprecision, hx, 0.0, 0.0);
01349         Vio_printf(sock, "delta %s\n", precFormat);
01350         sprintf(precFormat, Vprecision, 0.0, hy, 0.0);
01351         Vio_printf(sock, "delta %s\n", precFormat);
01352         sprintf(precFormat, Vprecision, 0.0, 0.0, hzed);
01353         Vio_printf(sock, "delta %s\n", precFormat);
01354
01355         /* Write off the DX regular connections */
01356         Vio_printf(sock, "object 2 class gridconnections counts %d %d %d\n",
01357           nxPART, nyPART, nzPART);
01358
01359         /* Write off the DX data */
01360         Vio_printf(sock, "object 3 class array type double rank 0 items %lu \
01361 data follows\n", (nxPART*nyPART*nzPART));
01362         icol = 0;
01363         for (i=0; i<nx; i++) {
01364             for (j=0; j<ny; j++) {
01365                 for (k=0; k<nz; k++) {
01366                     u = k*(nx)*(ny)+j*(nx)+i;
01367                     if (pvec[u] > 0.0) {
01368                         Vio_printf(sock, "%12.6e ", thee->data[u]);
01369                         icol++;
01370                         if (icol == 3) {
01371                             icol = 0;
01372                             Vio_printf(sock, "\n");
01373                         }
01374                     }
01375                 }
01376             }
01377         }
01378
01379         if (icol != 0) Vio_printf(sock, "\n");
01380
01381         /* Create the field */
01382         Vio_printf(sock, "attribute \"dep\" string \"positions\"\n");
01383         Vio_printf(sock, "object \"regular positions regular connections\" \
01384 class field\n");
01385         Vio_printf(sock, "component \"positions\" value 1\n");
01386         Vio_printf(sock, "component \"connections\" value 2\n");
01387         Vio_printf(sock, "component \"data\" value 3\n");
01388
01389     } else {
01390         /* Write off the title (if we're not XDR) */
01391         if (Vstring_strcasecmp(iofmt, "XDR") == 0) {
01392             Vnm_print(0, "Vgrid_writeDX:  Skipping comments for XDR format.\n");
01393         } else {
01394             Vnm_print(0, "Vgrid_writeDX:  Writing comments for %s format.\n",
01395               iofmt);
01396             Vio_printf(sock, "# Data from %s\n", PACKAGE_STRING);
01397             Vio_printf(sock, "# \n");
01398             Vio_printf(sock, "# %s\n", title);
01399             Vio_printf(sock, "# \n");
01400         }
01401
01402
01403         /* Write off the DX regular positions */
01404         Vio_printf(sock, "object 1 class gridpositions counts %d %d %d\n",
```

```
01405           nx, ny, nz);
01406
01407         sprintf(precFormat, Vprecision, xmin, ymin, zmin);
01408         Vio_printf(sock, "origin %s\n", precFormat);
01409         sprintf(precFormat, Vprecision, hx, 0.0, 0.0);
01410         Vio_printf(sock, "delta %s\n", precFormat);
01411         sprintf(precFormat, Vprecision, 0.0, hy, 0.0);
01412         Vio_printf(sock, "delta %s\n", precFormat);
01413         sprintf(precFormat, Vprecision, 0.0, 0.0, hzed);
01414         Vio_printf(sock, "delta %s\n", precFormat);
01415
01416         /* Write off the DX regular connections */
01417         Vio_printf(sock, "object 2 class gridconnections counts %d %d %d\n",
01418           nx, ny, nz);
01419
01420         /* Write off the DX data */
01421         Vio_printf(sock, "object 3 class array type double rank 0 items %lu \
01422 data follows\n", (nx*ny*nz));
01423         icol = 0;
01424         for (i=0; i<nx; i++) {
01425             for (j=0; j<ny; j++) {
01426                 for (k=0; k<nz; k++) {
01427                     u = k*(nx)*(ny)+j*(nx)+i;
01428                     Vio_printf(sock, "%12.6e ", thee->data[u]);
01429                     icol++;
01430                     if (icol == 3) {
01431                         icol = 0;
01432                         Vio_printf(sock, "\n");
01433                     }
01434                 }
01435             }
01436         }
01437         if (icol != 0) Vio_printf(sock, "\n");
01438
01439         /* Create the field */
01440         Vio_printf(sock, "attribute \"dep\" string \"positions\"\n");
01441         Vio_printf(sock, "object \"regular positions regular connections\" \
01442 class field\n");
01443         Vio_printf(sock, "component \"positions\" value 1\n");
01444         Vio_printf(sock, "component \"connections\" value 2\n");
01445         Vio_printf(sock, "component \"data\" value 3\n");
01446     }
01447
01448     /* Close off the socket */
01449     Vio_connectFree(sock);
01450     Vio_dtor(&sock);
01451 }
01452
01453 /* ///////////////////////////////////////////////////////////////////////////
01454 // Routine:  Vgrid_writeDXBIN
01455 //
01456 // Author:   Juan Brandi
01458 VPUBLIC void Vgrid_writeDXBIN(Vgrid *thee, const char *iodev, const char *iofmt,
01459   const char *thost, const char *fname, char *title, double *pvec){
01460
01461  double xmin, ymin, zmin, hx, hy, hzed;
01462  int nx, ny, nz, nxPART, nyPART, nzPART;
01463  int usepart, gotit;
01464  size_t icol, i, j, k, u;
01465  double x, y, z, xminPART, yminPART, zminPART;
01466  //Vio *sock;
01467  char precFormat[VMAX_BUFSIZE];
01468
01469  if (thee == VNULL) {
01470   Vnm_print(2, "Vgrid_writeDXBIN:  Error -- got VNULL thee!\n");
01471    VASSERT(0);
01472   }
01473   if (!(thee->ctordata || thee->readdata)) {
01474    Vnm_print(2, "Vgrid_writeDXBIN:  Error -- no data available!\n");
01475    VASSERT(0);
01476   }
01477
01478
01479   hx = thee->hx;
01480   hy = thee->hy;
01481   hzed = thee->hzed;
01482   nx = thee->nx;
01483   ny = thee->ny;
01484   nz = thee->nz;
01485   xmin = thee->xmin;
01486   ymin = thee->ymin;
```

```
01487   zmin = thee->zmin;
01488
01489   if (pvec == VNULL) usepart = 0;
01490   else usepart = 1;
01491
01492   /*will not use vio methods to try to avoid using malloc.*/
01493   FILE *fd = fopen(fname,"wb");
01494
01495   //check to se if the file was created/open successfully.
01496   if(fd == NULL){
01497    printf("Vgrid_writeDXBIN: Problem opening file %s for writing.\n", fname);
01498    return;
01499   }
01500
01501   printf("Vgrid_writeDXBIN: Writing to file...\n");
01502
01503   if (usepart) {
01504    /* Get the lower corner and number of grid points for the local
01505     * partition */
01506    xminPART = VLARGE;
01507    yminPART = VLARGE;
01508    zminPART = VLARGE;
01509    nxPART = 0;
01510    nyPART = 0;
01511    nzPART = 0;
01512    /* First, search for the lower corner */
01513    for (k=0; k<nz; k++) {
01514     z = k*hzed + zmin;
01515     for (j=0; j<ny; j++) {
01516      y = j*hy + ymin;
01517      for (i=0; i<nx; i++) {
01518       x = i*hx + xmin;
01519       if (pvec[IJK(i,j,k)] > 0.0) {
01520        if (x < xminPART) xminPART = x;
01521        if (y < yminPART) yminPART = y;
01522        if (z < zminPART) zminPART = z;
01523       }
01524      }
01525     }
01526    }
01527    /* Now search for the number of grid points in the z direction */
01528    for (k=0; k<nz; k++) {
01529     gotit = 0;
01530     for (j=0; j<ny; j++) {
01531      for (i=0; i<nx; i++) {
01532       if (pvec[IJK(i,j,k)] > 0.0) {
01533        gotit = 1;
01534        break;
01535       }
01536      }
01537      if (gotit) break;
01538     }
01539     if (gotit) nzPART++;
01540    }
01541    /* Now search for the number of grid points in the y direction */
01542    for (j=0; j<ny; j++) {
01543     gotit = 0;
01544     for (k=0; k<nz; k++) {
01545      for (i=0; i<nx; i++) {
01546       if (pvec[IJK(i,j,k)] > 0.0) {
01547        gotit = 1;
01548        break;
01549       }
01550      }
01551      if (gotit) break;
01552     }
01553     if (gotit) nyPART++;
01554    }
01555    /* Now search for the number of grid points in the x direction */
01556    for (i=0; i<nx; i++) {
01557     gotit = 0;
01558     for (k=0; k<nz; k++) {
01559      for (j=0; j<ny; j++) {
01560       if (pvec[IJK(i,j,k)] > 0.0) {
01561        gotit = 1;
01562        break;
01563       }
01564      }
01565      if (gotit) break;
01566     }
01567     if (gotit) nxPART++;
```

```
01568    }
01569
01570    if ((nxPART != nx) || (nyPART != ny) || (nzPART != nz)) {
01571     Vnm_print(0, "Vgrid_writeDXBIN:  printing only subset of domain\n");
01572    }
01573
01574    /* Write title (we're in XDR and "wb") */
01575    //Vnm_print(0, "Vgrid_writeDXBIN:  Writing comments for dxbin format.\n");
01576    printf("Vgrid_writeDXBIN: Writing comments for dxbin format\n");
01577    fprintf(fd, "# Data from %s\n",PACKAGE_STRING);
01578    fprintf(fd, "# \n");
01579    fprintf(fd, "# %s\n",title);
01580    fprintf(fd, "# \n");
01581
01582    /* Write off the DX regular positions */
01583    fprintf(fd, "object 1 class gridpositions counts %d %d %d\n",nxPART, nyPART, nzPART);
01584
01585    sprintf(precFormat, Vprecision, xminPART, yminPART, zminPART);
01586    fprintf(fd, "origin %s\n",precFormat);
01587
01588    sprintf(precFormat, Vprecision, hx, 0.0, 0.0);
01589    fprintf(fd, "delta %s\n",precFormat);
01590
01591    sprintf(precFormat, Vprecision, 0.0, hy, 0.0);
01592    fprintf(fd, "delta %s\n", precFormat);
01593
01594    sprintf(precFormat, Vprecision, 0.0, 0.0, hzed);
01595    fprintf(fd, "delta %s\n", precFormat);
01596
01597    /* Write off the DX regular connections */
01598    fprintf(fd, "object 2 class gridconnections counts %d %d %d\n", nxPART, nyPART, nzPART);
01599
01600    /* Write off the DX data */
01601    fprintf(fd, "object 3 class array type double rank 0 items %d binary data
       follows\n",(nxPART*nyPART*nzPART));
01602
01603    icol = 0;
01604    for (i=0; i<nx; i++) {
01605     for (j=0; j<ny; j++) {
01606      for (k=0; k<nz; k++) {
01607       u = k*(nx)*(ny)+j*(nx)+i;
01608       if (pvec[u] > 0.0) {
01609        fwrite(&(thee->data)[u],sizeof(double),1,fd);
01610        icol++;
01611        /*don't need the column formating to write binary doubles.*/
01612        if (icol == 3) {
01613         icol = 0;
01614        }
01615       }
01616      }
01617     }
01618    }
01619
01620    fprintf(fd,"\n");
01621
01622    /* Create the field */
01623    fprintf(fd, "attribute \"dep\" string \"positions\"\n");
01624    fprintf(fd, "object \"regular positions regular connections\" class field\n");
01625    fprintf(fd, "component \"positions\" value 1\n");
01626    fprintf(fd, "component \"connections\" value 2\n");
01627    fprintf(fd, "component \"data\" value 3\n");
01628
01629    fclose(fd);
01630
01631   } else {
01632    /*write dx format title*/
01633    printf("Vgrid_writeDXBIN: Writing comments for %s format.\n",iofmt);
01634    fprintf(fd,"# Data from %s\n", PACKAGE_STRING);
01635    fprintf(fd,"# \n");
01636    fprintf(fd, "# %s\n", title);
01637    fprintf(fd, "# \n");
01638
01639    /* Write off the DX regular positions */
01640    fprintf(fd, "object 1 class gridpositions counts %d %d %d\n", nx, ny, nz);
01641
01642    sprintf(precFormat, Vprecision, xmin, ymin, zmin);
01643    fprintf(fd, "origin %s\n", precFormat);
01644
01645    sprintf(precFormat, Vprecision, hx, 0.0, 0.0);
01646    fprintf(fd, "delta %s\n", precFormat);
01647
```

```
01648    sprintf(precFormat, Vprecision, 0.0, hy, 0.0);
01649    fprintf(fd, "delta %s\n", precFormat);
01650
01651    sprintf(precFormat, Vprecision, 0.0, 0.0, hzed);
01652    fprintf(fd, "delta %s\n", precFormat);
01653
01654    /* Write off the DX regular connections */
01655    fprintf(fd, "object 2 class gridconnections counts %d %d %d\n", nx, ny, nz);
01656
01657    /* Write off the DX data */
01658    fprintf(fd, "object 3 class array type double rank 0 items %d binary data follows\n", (nx*ny*nz));
01659
01660    icol = 0;
01661    for (i=0; i<nx; i++) {
01662     for (j=0; j<ny; j++) {
01663      for (k=0; k<nz; k++) {
01664       u = k*(nx)*(ny)+j*(nx)+i;
01665       fwrite(&(thee->data)[u],sizeof(double),1,fd);
01666       icol++;
01667       if (icol == 3) {
01668        icol = 0;
01669       }
01670      }
01671     }
01672    }
01673
01674    fprintf(fd, "\n");
01675
01676    /* Create the field */
01677    fprintf(fd, "attribute \"dep\" string \"positions\"\n");
01678    fprintf(fd, "object \"regular positions regular connections\" class field\n");
01679    fprintf(fd, "component \"positions\" value 1\n");
01680    fprintf(fd, "component \"connections\" value 2\n");
01681    fprintf(fd, "component \"data\" value 3\n");
01682
01683    fclose(fd);
01684   }
01685 }
01686
01687
01688 /* ///////////////////////////////////////////////////////////////////////////
01689 // Routine:  Vgrid_writeUHBD
01690 // Author:   Nathan Baker
01692 VPUBLIC void Vgrid_writeUHBD(Vgrid *thee, const char *iodev, const char *iofmt,
01693   const char *thost, const char *fname, char *title, double *pvec) {
01694
01695     size_t u, icol, i, j, k;
01696     size_t gotit, nx, ny, nz;
01697     double xmin, ymin, zmin, hzed, hy, hx;
01698     Vio *sock;
01699
01700     if (thee == VNULL) {
01701         Vnm_print(2, "Vgrid_writeUHBD:  Error -- got VNULL thee!\n");
01702         VASSERT(0);
01703     }
01704     if (!(thee->ctordata || thee->readdata)) {
01705         Vnm_print(2, "Vgrid_writeUHBD:  Error -- no data available!\n");
01706         VASSERT(0);
01707     }
01708
01709     if ((thee->hx!=thee->hy) || (thee->hy!=thee->hzed)
01710       || (thee->hx!=thee->hzed)) {
01711         Vnm_print(2, "Vgrid_writeUHBD: can't write UHBD mesh with non-uniform \
01712 spacing\n");
01713         return;
01714     }
01715
01716     /* Set up the virtual socket */
01717     sock = Vio_ctor(iodev,iofmt,thost,fname,"w");
01718     if (sock == VNULL) {
01719         Vnm_print(2, "Vgrid_writeUHBD: Problem opening virtual socket %s\n",
01720           fname);
01721         return;
01722     }
01723     if (Vio_connect(sock, 0) < 0) {
01724         Vnm_print(2, "Vgrid_writeUHBD: Problem connecting virtual socket %s\n",
01725           fname);
01726         return;
01727     }
01728
01729     /* Get the lower corner and number of grid points for the local
```

```
01730      * partition */
01731     hx = thee->hx;
01732     hy = thee->hy;
01733     hzed = thee->hzed;
01734     nx = thee->nx;
01735     ny = thee->ny;
01736     nz = thee->nz;
01737     xmin = thee->xmin;
01738     ymin = thee->ymin;
01739     zmin = thee->zmin;
01740
01741     /* Let interested folks know that partition information is ignored */
01742     if (pvec != VNULL) {
01743         gotit = 0;
01744         for (i=0; i<(nx*ny*nz); i++) {
01745             if (pvec[i] == 0) {
01746                 gotit = 1;
01747                 break;
01748             }
01749         }
01750         if (gotit) {
01751             Vnm_print(2, "Vgrid_writeUHBD:  IGNORING PARTITION INFORMATION!\n");
01752             Vnm_print(2, "Vgrid_writeUHBD:  This means I/O from parallel runs \
01753 will have significant overlap.\n");
01754         }
01755     }
01756
01757     /* Write out the header */
01758     Vio_printf(sock, "%72s\n", title);
01759     Vio_printf(sock, "%12.5e%12.5e%7d%7d%7d%7d%7d\n", 1.0, 0.0, -1, 0,
01760       nz, 1, nz);
01761     Vio_printf(sock, "%7d%7d%7d%12.5e%12.5e%12.5e%12.5e\n", nx, ny, nz,
01762       hx, (xmin-hx), (ymin-hx), (zmin-hx));
01763     Vio_printf(sock, "%12.5e%12.5e%12.5e%12.5e\n", 0.0, 0.0, 0.0, 0.0);
01764     Vio_printf(sock, "%12.5e%12.5e%7d%7d", 0.0, 0.0, 0, 0);
01765
01766     /* Write out the entries */
01767     icol = 0;
01768     for (k=0; k<nz; k++) {
01769         Vio_printf(sock, "\n%7d%7d%7d\n", k+1, thee->nx, thee->ny);
01770         icol = 0;
01771         for (j=0; j<ny; j++) {
01772             for (i=0; i<nx; i++) {
01773                 u = k*(nx)*(ny)+j*(nx)+i;
01774                 icol++;
01775                 Vio_printf(sock, " %12.5e", thee->data[u]);
01776                 if (icol == 6) {
01777                     icol = 0;
01778                     Vio_printf(sock, "\n");
01779                 }
01780             }
01781         }
01782     }
01783     if (icol != 0) Vio_printf(sock, "\n");
01784
01785     /* Close off the socket */
01786     Vio_connectFree(sock);
01787     Vio_dtor(&sock);
01788 }
01789
01790 VPUBLIC double Vgrid_integrate(Vgrid *thee) {
01791
01792     size_t i, j, k;
01793     int nx, ny, nz;
01794     double sum, w;
01795
01796     if (thee == VNULL) {
01797         Vnm_print(2, "Vgrid_integrate:  Got VNULL thee!\n");
01798         VASSERT(0);
01799     }
01800
01801     nx = thee->nx;
01802     ny = thee->ny;
01803     nz = thee->nz;
01804
01805     sum = 0.0;
01806
01807     for (k=0; k<nz; k++) {
01808         w = 1.0;
01809         if ((k==0) || (k==(nz-1))) w = w * 0.5;
01810         for (j=0; j<ny; j++) {
```

```
01811             w = 1.0;
01812             if ((j==0) || (j==(ny-1))) w = w * 0.5;
01813             for (i=0; i<nx; i++) {
01814                 w = 1.0;
01815                 if ((i==0) || (i==(nx-1))) w = w * 0.5;
01816                 sum = sum + w*(thee->data[IJK(i,j,k)]);
01817             }
01818         }
01819     }
01820
01821     sum = sum*(thee->hx)*(thee->hy)*(thee->hzed);
01822
01823     return sum;
01824
01825 }
01826
01827
01828 VPUBLIC double Vgrid_normL1(Vgrid *thee) {
01829
01830     size_t i, j, k;
01831     int nx, ny, nz;
01832     double sum;
01833
01834     if (thee == VNULL) {
01835         Vnm_print(2, "Vgrid_normL1:  Got VNULL thee!\n");
01836         VASSERT(0);
01837     }
01838
01839     nx = thee->nx;
01840     ny = thee->ny;
01841     nz = thee->nz;
01842
01843     sum = 0.0;
01844     for (k=0; k<nz; k++) {
01845         for (j=0; j<ny; j++) {
01846             for (i=0; i<nx; i++) {
01847                 sum = sum + VABS(thee->data[IJK(i,j,k)]);
01848             }
01849         }
01850     }
01851
01852     sum = sum*(thee->hx)*(thee->hy)*(thee->hzed);
01853
01854     return sum;
01855
01856 }
01857
01858 VPUBLIC double Vgrid_normL2(Vgrid *thee) {
01859
01860     size_t i, j, k;
01861     int nx, ny, nz;
01862     double sum;
01863
01864     if (thee == VNULL) {
01865         Vnm_print(2, "Vgrid_normL2:  Got VNULL thee!\n");
01866         VASSERT(0);
01867     }
01868
01869     nx = thee->nx;
01870     ny = thee->ny;
01871     nz = thee->nz;
01872
01873     sum = 0.0;
01874     for (k=0; k<nz; k++) {
01875         for (j=0; j<ny; j++) {
01876             for (i=0; i<nx; i++) {
01877                 sum = sum + VSQR(thee->data[IJK(i,j,k)]);
01878             }
01879         }
01880     }
01881
01882     sum = sum*(thee->hx)*(thee->hy)*(thee->hzed);
01883
01884     return VSQRT(sum);
01885
01886 }
01887
01888 VPUBLIC double Vgrid_seminormH1(Vgrid *thee) {
01889
01890     size_t i, j, k;
01891     int nx, ny, nz, d;
```

```
01892     double pt[3], grad[3], sum, hx, hy, hzed, xmin, ymin, zmin;
01893
01894     if (thee == VNULL) {
01895         Vnm_print(2, "Vgrid_seminormH1:  Got VNULL thee!\n");
01896         VASSERT(0);
01897     }
01898
01899     nx = thee->nx;
01900     ny = thee->ny;
01901     nz = thee->nz;
01902     hx = thee->hx;
01903     hy = thee->hy;
01904     hzed = thee->hzed;
01905     xmin = thee->xmin;
01906     ymin = thee->ymin;
01907     zmin = thee->zmin;
01908
01909     sum = 0.0;
01910     for (k=0; k<nz; k++) {
01911         pt[2] = k*hzed + zmin;
01912         for (j=0; j<ny; j++) {
01913             pt[1] = j*hy + ymin;
01914             for (i=0; i<nx; i++) {
01915                 pt[0] = i*hx + xmin;
01916                 VASSERT(Vgrid_gradient(thee, pt, grad));
01917                 for (d=0; d<3; d++) sum = sum + VSQR(grad[d]);
01918             }
01919         }
01920     }
01921
01922     sum = sum*(hx)*(hy)*(hzed);
01923
01924     if (VABS(sum) < VSMALL) sum = 0.0;
01925     else sum = VSQRT(sum);
01926
01927     return sum;
01928
01929 }
01930
01931 VPUBLIC double Vgrid_normH1(Vgrid *thee) {
01932
01933     double sum = 0.0;
01934
01935     if (thee == VNULL) {
01936         Vnm_print(2, "Vgrid_normH1:  Got VNULL thee!\n");
01937         VASSERT(0);
01938     }
01939
01940     sum = VSQR(Vgrid_seminormH1(thee)) + VSQR(Vgrid_normL2(thee));
01941
01942     return VSQRT(sum);
01943
01944 }
01945
01946 VPUBLIC double Vgrid_normLinf(Vgrid *thee) {
01947
01948     size_t i, j, k;
01949     int nx, ny, nz, gotval;
01950     double sum, val;
01951
01952     if (thee == VNULL) {
01953         Vnm_print(2, "Vgrid_normLinf:  Got VNULL thee!\n");
01954         VASSERT(0);
01955     }
01956
01957     nx = thee->nx;
01958     ny = thee->ny;
01959     nz = thee->nz;
01960
01961     sum = 0.0;
01962     gotval = 0;
01963     for (k=0; k<nz; k++) {
01964         for (j=0; j<ny; j++) {
01965             for (i=0; i<nx; i++) {
01966                 val = VABS(thee->data[IJK(i,j,k)]);
01967                 if (!gotval) {
01968                     gotval = 1;
01969                     sum = val;
01970                 }
01971                 if (val > sum) sum = val;
01972             }
```

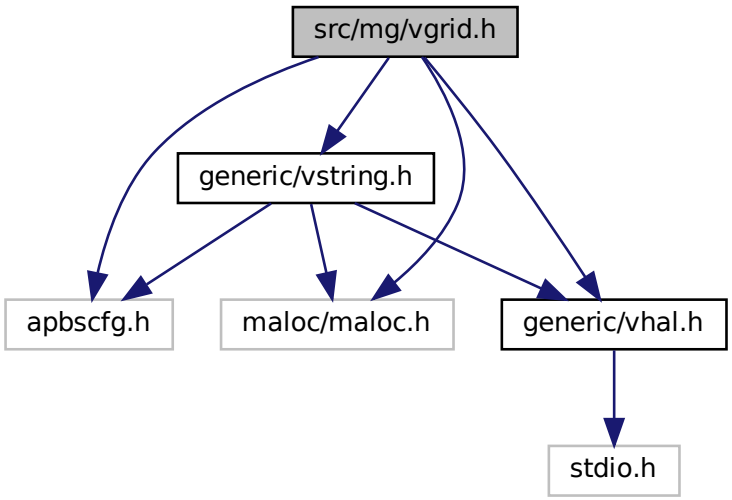
```
01973         }
01974     }
01975
01976     return sum;
01977
01978 }
```

9.93 src/mg/vgrid.h File Reference

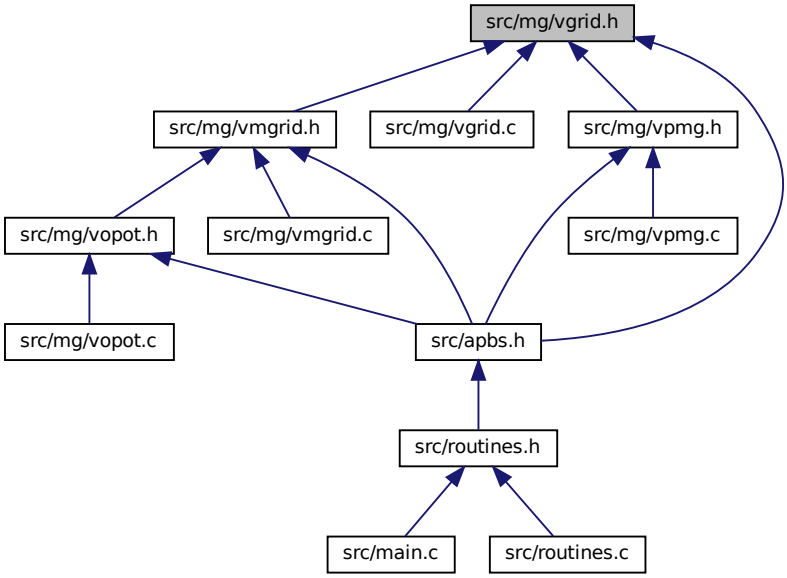
Potential oracle for Cartesian mesh data.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/vstring.h"
```

Include dependency graph for vgrid.h:



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



Data Structures

- struct sVgrid

  Electrostatic potential oracle for Cartesian mesh data.

Macros

- #define VGRID_DIGITS 6

  Number of decimal places for comparisons and formatting.

Typedefs

- typedef struct sVgrid Vgrid

  Declaration of the Vgrid class as the sVgrid structure.

Functions

- VEXTERNC unsigned long int Vgrid_memChk (Vgrid *thee)

  Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC Vgrid * Vgrid_ctor (int nx, int ny, int nz, double hx, double hy, double hzed, double xmin, double ymin, double zmin, double *data)

  Construct Vgrid object with values obtained from Vpmg_readDX (for example)
- VEXTERNC int Vgrid_ctor2 (Vgrid *thee, int nx, int ny, int nz, double hx, double hy, double hzed, double xmin, double ymin, double zmin, double *data)

  Initialize Vgrid object with values obtained from Vpmg_readDX (for example)
- VEXTERNC int Vgrid_value (Vgrid *thee, double x[3], double *value)

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

- VEXTERNC void Vgrid_dtor (Vgrid **thee)

  Object destructor.
- VEXTERNC void Vgrid_dtor2 (Vgrid *thee)

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

  Write out the binary data in OpenDX grid format.
- VEXTERNC int Vgrid_readDXBIN (Vgrid *thee, const char *iodev, const char *iofmt, const char *thost, const char *fname)

  Read in binary 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:
- VEXTERNC double Vgrid_normL2 (Vgrid *thee)

  Get the L_2 norm of the data. This returns the integral:
- VEXTERNC double Vgrid_normLinf (Vgrid *thee)

  Get the L_∞ norm of the data. This returns the integral:
- VEXTERNC double Vgrid_seminormH1 (Vgrid *thee)

  Get the H_1 semi-norm of the data. This returns the integral:
- VEXTERNC double Vgrid_normH1 (Vgrid *thee)

  Get the H_1 norm (or energy norm) of the data. This returns the integral:

9.93.1 Detailed Description

Potential oracle for Cartesian mesh data.

Author

Nathan Baker and Steve Bond

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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.

9.93.2 Function Documentation

9.93.2.1 Vgrid_writeGZ()

```
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 1011 of file vgrid.c.

9.94 vgrid.h

```
00001
00062 #ifndef _VGRID_H_
00063 #define _VGRID_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "maloc/maloc.h"
00068
00069 #include "generic/vhal.h"
00070 #include "generic/vstring.h"
00071
00074 #define VGRID_DIGITS 6
00075
00081 struct sVgrid {
00082
00083     int nx;
00084     int ny;
00085     int nz;
00086     double hx;
00087     double hy;
00088     double hzed;
00089     double xmin;
00090     double ymin;
00091     double zmin;
00092     double xmax;
00093     double ymax;
00094     double zmax;
00095     double *data;
00096     int readdata;
00097     int ctordata;
00099     Vmem *mem;
00100 };
00101
00106 typedef struct sVgrid Vgrid;
00107
00108 #if !defined(VINLINE_VGRID)
00109
00117     VEXTERNC unsigned long int Vgrid_memChk(Vgrid *thee);
00118
00119 #else /* if defined(VINLINE_VGRID) */
00120
00128 #define Vgrid_memChk(thee) (Vmem_bytes((thee)->vmem))
00129
```

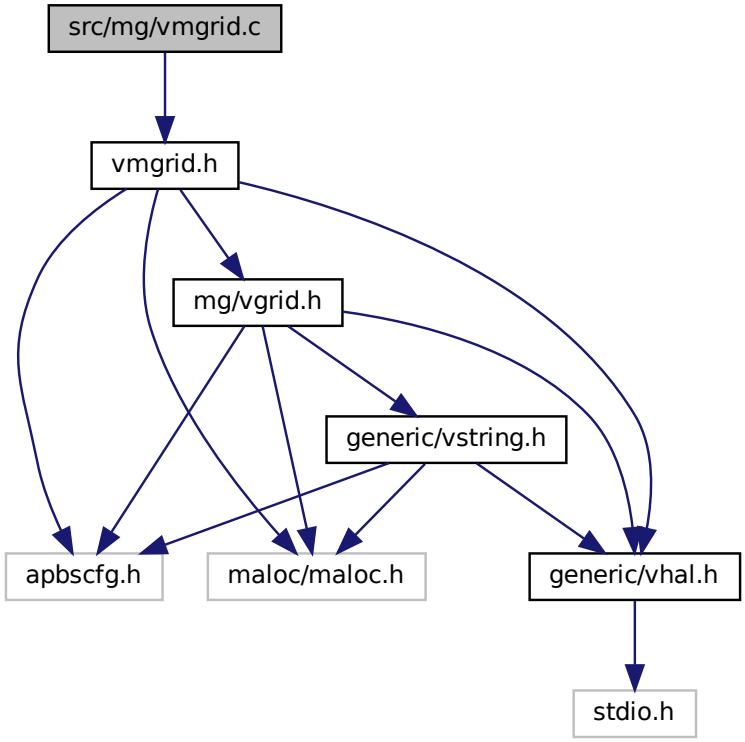
```
00130 #endif /* if !defined(VINLINE_VPMG) */
00131
00149 VEXTERNC Vgrid*  Vgrid_ctor(int nx, int ny, int nz,
00150                   double hx, double hy, double hzed,
00151                   double xmin, double ymin, double zmin,
00152                   double *data);
00153
00172 VEXTERNC int Vgrid_ctor2(Vgrid *thee, int nx, int ny, int nz,
00173                   double hx, double hy, double hzed,
00174                   double xmin, double ymin, double zmin,
00175                   double *data);
00176
00185 VEXTERNC int Vgrid_value(Vgrid *thee, double x[3], double *value);
00186
00192 VEXTERNC void Vgrid_dtor(Vgrid **thee);
00193
00199 VEXTERNC void Vgrid_dtor2(Vgrid *thee);
00200
00214 VEXTERNC int Vgrid_curvature(Vgrid *thee, double pt[3], int cflag,
00215   double *curv);
00216
00225 VEXTERNC int Vgrid_gradient(Vgrid *thee, double pt[3], double grad[3] );
00226
00231 VEXTERNC int Vgrid_readGZ(
00232                           Vgrid *thee,
00233                           const char *fname
00234                           );
00235
00239 VEXTERNC void Vgrid_writeGZ(
00240                             Vgrid *thee,
00241                             const char *iodev,
00242                             const char *iofmt,
00243                             const char *thost,
00244                             const char *fname,
00245                             char *title,
00246                             double *pvec
00247                             );
00248
00266 VEXTERNC void Vgrid_writeUHBD(Vgrid *thee, const char *iodev,
00267   const char *iofmt, const char *thost, const char *fname, char *title,
00268   double *pvec);
00269
00284 VEXTERNC void Vgrid_writeDX(Vgrid *thee, const char *iodev,
00285   const char *iofmt,  const char *thost, const char *fname, char *title,
00286   double *pvec);
00287
00299 VEXTERNC int Vgrid_readDX(Vgrid *thee, const char *iodev, const char *iofmt,
00300   const char *thost, const char *fname);
00301
00316 VEXTERNC void Vgrid_writeDXBIN(Vgrid *thee, const char *iodev,
00317   const char *iofmt,  const char *thost, const char *fname, char *title,
00318   double *pvec);
00319
00320
00332 VEXTERNC int Vgrid_readDXBIN(Vgrid *thee, const char *iodev, const char *iofmt,
00333    const char *thost, const char *fname);
00334
00341 VEXTERNC double Vgrid_integrate(Vgrid *thee);
00342
00351 VEXTERNC double Vgrid_normL1(Vgrid *thee);
00352
00361 VEXTERNC double Vgrid_normL2(Vgrid *thee);
00362
00371 VEXTERNC double Vgrid_normLinf(Vgrid *thee);
00372
00382 VEXTERNC double Vgrid_seminormH1(Vgrid *thee);
00383
00394 VEXTERNC double Vgrid_normH1(Vgrid *thee);
00395
00396 #endif
```

9.95 src/mg/vmgrid.c File Reference

Class Vmgrid methods.

```
#include "vmgrid.h"
```

Include dependency graph for vmgrid.c:



Functions

- VPUBLIC Vmgrid * Vmgrid_ctor ()

  Construct Vmgrid object.
- VPUBLIC int Vmgrid_ctor2 (Vmgrid *thee)

  Initialize Vmgrid object.
- VPUBLIC void Vmgrid_dtor (Vmgrid **thee)

  Object destructor.
- VPUBLIC void Vmgrid_dtor2 (Vmgrid *thee)

  FORTRAN stub object destructor.
- VPUBLIC int Vmgrid_value (Vmgrid *thee, double pt[3], double *value)

  Get potential value (from mesh or approximation) at a point.
- VPUBLIC int Vmgrid_curvature (Vmgrid *thee, double pt[3], int cflag, double *value)

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

  Get first derivative values at a point.
- VPUBLIC int Vmgrid_addGrid (Vmgrid *thee, Vgrid *grid)

  Add a grid to the hierarchy.

9.95.1 Detailed Description

Class Vmgrid methods.

Author

Nathan Baker

Version

Id

vmgrid.c 1615 2010-10-20 19:16:35Z sobolevnrm

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
* Nathan A. Baker (nathan.baker@pnl.gov)
* Pacific Northwest National Laboratory
*
* Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010, Pacific Northwest National Laboratory.  Portions Copyright (c) 2002-2020, Washington Un
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* -  Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* - Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* - Neither the name of Washington University in St. Louis nor the names of its
* contributors may be used to endorse or promote products derived from this
* software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*
```

Definition in file vmgrid.c.

9.96 vmgrid.c

```
00001
00049 #include "vmgrid.h"
00050
00051 VEMBED(rcsid="$Id: vmgrid.c 1615 2010-10-20 19:16:35Z sobolevnrm $")
```

```
00052
00053 /* ////////////////////////////////////////////////////////////////////////
00054 // Routine:  Vmgrid_ctor
00055 // Author:   Nathan Baker
00057 VPUBLIC Vmgrid* Vmgrid_ctor() {
00058
00059     Vmgrid *thee = VNULL;
00060
00061     thee = Vmem_malloc(VNULL, 1, sizeof(Vmgrid));
00062     VASSERT(thee != VNULL);
00063     VASSERT(Vmgrid_ctor2(thee));
00064
00065     return thee;
00066 }
00067
00068 /* ////////////////////////////////////////////////////////////////////////
00069 // Routine:  Vmgrid_ctor2
00070 // Author:   Nathan Baker
00072 VPUBLIC int Vmgrid_ctor2(Vmgrid *thee) {
00073
00074     int i;
00075
00076     if (thee == VNULL) return 0;
00077
00078     thee->ngrids = 0;
00079     for (i=0; i<VMGRIDMAX; i++) thee->grids[i] = VNULL;
00080
00081     return 1;
00082 }
00083
00084 /* ////////////////////////////////////////////////////////////////////////
00085 // Routine:  Vmgrid_dtor
00086 // Author:   Nathan Baker
00088 VPUBLIC void Vmgrid_dtor(Vmgrid **thee) {
00089
00090     if ((*thee) != VNULL) {
00091         Vmgrid_dtor2(*thee);
00092         Vmem_free(VNULL, 1, sizeof(Vmgrid), (void **)thee);
00093         (*thee) = VNULL;
00094     }
00095 }
00096
00097 /* ////////////////////////////////////////////////////////////////////////
00098 // Routine:  Vmgrid_dtor2
00099 // Author:   Nathan Baker
00101 VPUBLIC void Vmgrid_dtor2(Vmgrid *thee) { ; }
00102
00103 /* ////////////////////////////////////////////////////////////////////////
00104 // Routine:  Vmgrid_value
00105 // Author:   Nathan Baker
00107 VPUBLIC int Vmgrid_value(Vmgrid *thee, double pt[3], double *value) {
00108
00109     int i, rc;
00110     double tvalue;
00111
00112     VASSERT(thee != VNULL);
00113
00114     for (i=0; i<thee->ngrids; i++) {
00115         rc = Vgrid_value(thee->grids[i], pt, &tvalue);
00116         if (rc) {
00117             *value = tvalue;
00118             return 1;
00119         }
00120     }
00121
00122     Vnm_print(2, "Vmgrid_value:  Point (%g, %g, %g) not found in \
00123 hiearchy!\n", pt[0], pt[1], pt[2]);
00124
00125     return 0;
00126 }
00127
00128 /* ////////////////////////////////////////////////////////////////////////
00129 // Routine:  Vmgrid_curvature
00130 //
00131 //   Notes:  cflag=0 ==> Reduced Maximal Curvature
00132 //           cflag=1 ==> Mean Curvature (Laplace)
00133 //           cflag=2 ==> Gauss Curvature
00134 //           cflag=3 ==> True Maximal Curvature
00135 //
00136 // Authors:  Nathan Baker
00138 VPUBLIC int Vmgrid_curvature(Vmgrid *thee, double pt[3], int cflag,
```

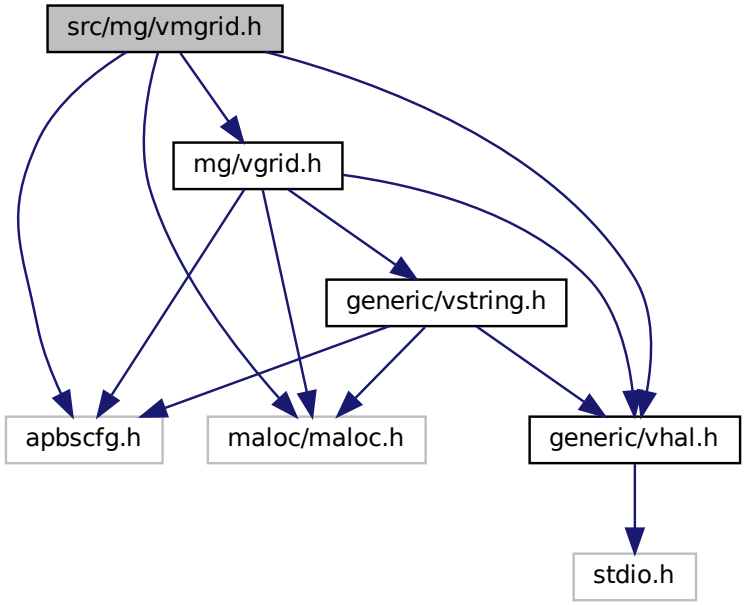
```
00139   double *value) {
00140
00141     int i, rc;
00142     double tvalue;
00143
00144     VASSERT(thee != VNULL);
00145
00146     for (i=0; i<thee->ngrids; i++) {
00147         rc = Vgrid_curvature(thee->grids[i], pt, cflag, &tvalue);
00148         if (rc) {
00149             *value = tvalue;
00150             return 1;
00151         }
00152     }
00153
00154     Vnm_print(2, "Vmgrid_curvature:  Point (%g, %g, %g) not found in \
00155 hiearchy!\n", pt[0], pt[1], pt[2]);
00156
00157     return 0;
00158
00159
00160 }
00161
00162 /* ///////////////////////////////////////////////////////////////////////////
00163 // Routine:  Vmgrid_gradient
00164 //
00165 // Authors:  Nathan Baker
00167 VPUBLIC int Vmgrid_gradient(Vmgrid *thee, double pt[3], double grad[3]) {
00168
00169     int i, j, rc;
00170     double tgrad[3];
00171
00172     VASSERT(thee != VNULL);
00173
00174     for (i=0; i<thee->ngrids; i++) {
00175         rc = Vgrid_gradient(thee->grids[i], pt, tgrad);
00176         if (rc) {
00177             for (j=0; j<3; j++) grad[j] = tgrad[j];
00178             return 1;
00179         }
00180     }
00181
00182     Vnm_print(2, "Vmgrid_gradient:  Point (%g, %g, %g) not found in \
00183 hiearchy!\n", pt[0], pt[1], pt[2]);
00184
00185     return 0;
00186
00187
00188 }
00189
00190 /* ///////////////////////////////////////////////////////////////////////////
00191 // Routine:  Vmgrid_addGrid
00192 //
00193 // Authors:  Nathan Baker
00195 VPUBLIC int Vmgrid_addGrid(Vmgrid *thee, Vgrid *grid) {
00196
00197     int i, j, rc;
00198     double tgrad[3];
00199
00200     VASSERT(thee != VNULL);
00201
00202     if (grid == VNULL) {
00203         Vnm_print(2, "Vmgrid_addGrid:  Not adding VNULL grid!\n");
00204         return 0;
00205     }
00206
00207     if (thee->ngrids >= VMGRIDMAX) {
00208         Vnm_print(2, "Vmgrid_addGrid:  Too many grids in hierarchy (max = \
00209 %d)!\n", VMGRIDMAX);
00210         Vnm_print(2, "Vmgrid_addGrid:  Not adding grid!\n");
00211         return 0;
00212     }
00213
00214     thee->grids[thee->ngrids] = grid;
00215     (thee->ngrids)++;
00216
00217     return 1;
00218
00219 }
```

9.97 src/mg/vmgrid.h File Reference

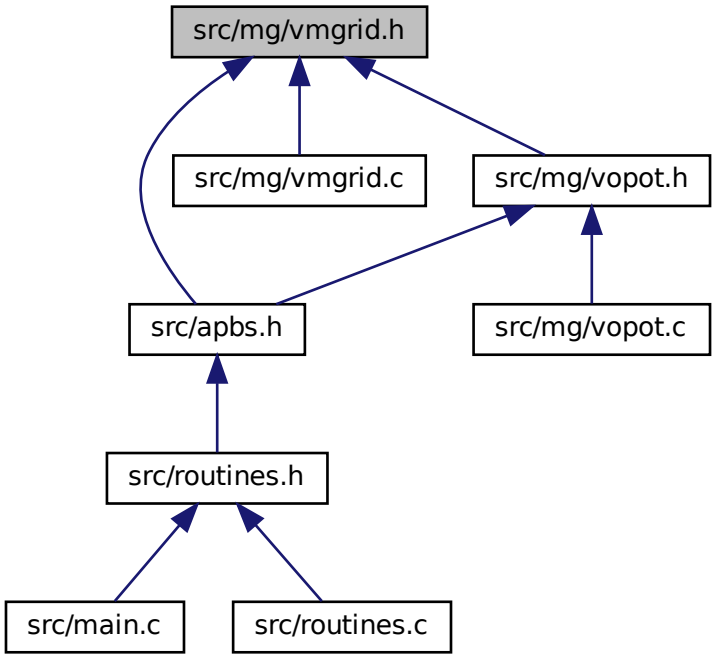
Multiresolution oracle for Cartesian mesh data.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "mg/vgrid.h"
```

Include dependency graph for vmgrid.h:



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



Data Structures

- struct sVmgrid

  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.

9.97.1 Detailed Description

Multiresolution oracle for Cartesian mesh data.

Author

Nathan Baker

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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.

9.98 vmgrid.h

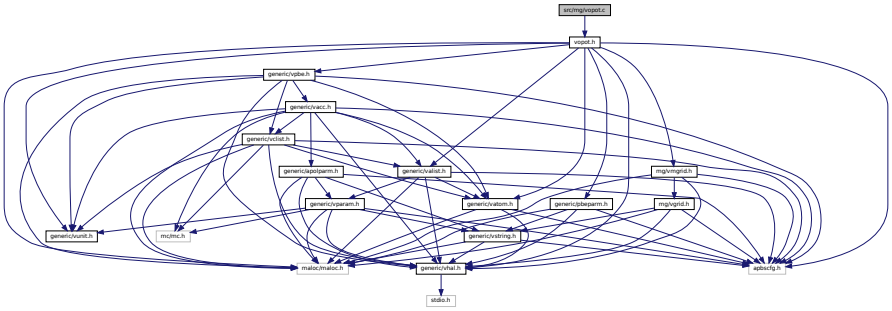
```
00001
00062 #ifndef _VMGRID_H_
00063 #define _VMGRID_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "maloc/maloc.h"
00068
00069 #include "generic/vhal.h"
00070 #include "mg/vgrid.h"
00071
00076 #define VMGRIDMAX 20
00077
00078
00084 struct sVmgrid {
00085
00086     int ngrids;
00087     Vgrid *grids[VMGRIDMAX];
00092 };
00093
00098 typedef struct sVmgrid Vmgrid;
00099
00105 VEXTERNC Vmgrid*  Vmgrid_ctor();
00106
00113 VEXTERNC int Vmgrid_ctor2(Vmgrid *thee);
00114
00123 VEXTERNC int Vmgrid_value(Vmgrid *thee, double x[3], double *value);
00124
00130 VEXTERNC void Vmgrid_dtor(Vmgrid **thee);
00131
00137 VEXTERNC void Vmgrid_dtor2(Vmgrid *thee);
00138
00151 VEXTERNC int Vmgrid_addGrid(Vmgrid *thee, Vgrid *grid);
00152
00153
00167 VEXTERNC int Vmgrid_curvature(Vmgrid *thee, double pt[3], int cflag,
00168   double *curv);
00169
00178 VEXTERNC int Vmgrid_gradient(Vmgrid *thee, double pt[3], double grad[3] );
00179
00187 VEXTERNC Vgrid* Vmgrid_getGridByNum(Vmgrid *thee, int num);
00188
00196 VEXTERNC Vgrid* Vmgrid_getGridByPoint(Vmgrid *thee, double pt[3]);
00197
00198 #endif
00199
```

9.99 src/mg/vopot.c File Reference

Class Vopot methods.

```
#include "vopot.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.

9.99.1 Detailed Description

Class Vopot methods.

Author

Nathan Baker

Version

Id

Attention

Definition in file vopot.c.

9.100 vopot.c

```
00001
00057 #include "vopot.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 /* ////////////////////////////////////////////////////////////////////////
00062 // Routine:  Vopot_ctor
00063 // Author:   Nathan Baker
00065 VPUBLIC Vopot* Vopot_ctor(Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl) {
00066
00067     Vopot *thee = VNULL;
00068
00069     thee = Vmem_malloc(VNULL, 1, sizeof(Vopot));
00070     VASSERT(thee != VNULL);
00071     VASSERT(Vopot_ctor2(thee, mgrid, pbe, bcfl));
00072
00073     return thee;
```

```
00074 }
00075
00076 /* ///////////////////////////////////////////////////////////////////////////
00077 // Routine:  Vopot_ctor2
00078 // Author:   Nathan Baker
00080 VPUBLIC int Vopot_ctor2(Vopot *thee, Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl) {
00081
00082     if (thee == VNULL) return 0;
00083     thee->bcfl = bcfl;
00084     thee->mgrid = mgrid;
00085     thee->pbe = pbe;
00086
00087     return 1;
00088 }
00089
00090 /* ///////////////////////////////////////////////////////////////////////////
00091 // Routine:  Vopot_dtor
00092 // Author:   Nathan Baker
00094 VPUBLIC void Vopot_dtor(Vopot **thee) {
00095
00096     if ((*thee) != VNULL) {
00097         Vopot_dtor2(*thee);
00098         Vmem_free(VNULL, 1, sizeof(Vopot), (void **)thee);
00099         (*thee) = VNULL;
00100     }
00101 }
00102
00103 /* ///////////////////////////////////////////////////////////////////////////
00104 // Routine:  Vopot_dtor2
00105 // Author:   Nathan Baker
00107 VPUBLIC void Vopot_dtor2(Vopot *thee) { return; }
00108
00109 /* ///////////////////////////////////////////////////////////////////////////
00110 // Routine:  Vopot_pot
00111 // Author:   Nathan Baker
00113 #define IJK(i,j,k)  (((k)*(nx)*(ny))+((j)*(nx))+(i))
00114 VPUBLIC int Vopot_pot(Vopot *thee, double pt[3], double *value) {
00115
00116     Vatom *atom;
00117     int i, iatom;
00118     double u, T, charge, eps_w, xkappa, dist, size, val, *position;
00119     Valist *alist;
00120
00121     VASSERT(thee != VNULL);
00122
00123     eps_w = Vpbe_getSolventDiel(thee->pbe);
00124     xkappa = (1.0e10)*Vpbe_getXkappa(thee->pbe);
00125     T = Vpbe_getTemperature(thee->pbe);
00126     alist = Vpbe_getValist(thee->pbe);
00127
00128     u = 0;
00129
00130     /* See if we're on the mesh */
00131     if (Vmgrid_value(thee->mgrid, pt, &u)) {
00132
00133         *value = u;
00134
00135     } else {
00136
00137         switch (thee->bcfl) {
00138
00139             case BCFL_ZERO:
00140                 u = 0;
00141                 break;
00142
00143             case BCFL_SDH:
00144                 size = (1.0e-10)*Vpbe_getSoluteRadius(thee->pbe);
00145                 position = Vpbe_getSoluteCenter(thee->pbe);
00146                 charge = Vunit_ec*Vpbe_getSoluteCharge(thee->pbe);
00147                 dist = 0;
00148                 for (i=0; i<3; i++)
00149                   dist += VSQR(position[i] - pt[i]);
00150                 dist = (1.0e-10)*VSQRT(dist);
00151                 val = (charge)/(4*VPI*Vunit_eps0*eps_w*dist);
00152                 if (xkappa != 0.0)
00153                   val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00154                 val = val*Vunit_ec/(Vunit_kb*T);
00155                 u = val;
00156                 break;
00157
00158             case BCFL_MDH:
```

```
00159                 u = 0;
00160                 for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
00161                     atom = Valist_getAtom(alist, iatom);
00162                     position = Vatom_getPosition(atom);
00163                     charge = Vunit_ec*Vatom_getCharge(atom);
00164                     size = (1e-10)*Vatom_getRadius(atom);
00165                     dist = 0;
00166                     for (i=0; i<3; i++)
00167                       dist += VSQR(position[i] - pt[i]);
00168                     dist = (1.0e-10)*VSQRT(dist);
00169                     val = (charge)/(4*VPI*Vunit_eps0*eps_w*dist);
00170                     if (xkappa != 0.0)
00171                       val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00172                     val = val*Vunit_ec/(Vunit_kb*T);
00173                     u = u + val;
00174                 }
00175                 break;
00176 
00177             case BCFL_UNUSED:
00178                 Vnm_print(2, "Vopot_pot:  Invalid bcfl flag (%d)!\n",
00179                   thee->bcfl);
00180                 return 0;
00181 
00182             case BCFL_FOCUS:
00183                 Vnm_print(2, "Vopot_pot:  Invalid bcfl flag (%d)!\n",
00184                   thee->bcfl);
00185                 return 0;
00186 
00187             default:
00188                 Vnm_print(2, "Vopot_pot:  Bogus thee->bcfl flag (%d)!\n",
00189                   thee->bcfl);
00190                 return 0;
00191                 break;
00192         }
00193 
00194         *value = u;
00195 
00196     }
00197 
00198     return 1;
00199 
00200 }
00201 
00202 /* ///////////////////////////////////////////////////////////////////////////
00203 // Routine:  Vopot_curvature
00204 //
00205 //   Notes:  cflag=0 ==> Reduced Maximal Curvature
00206 //           cflag=1 ==> Mean Curvature (Laplace)
00207 //           cflag=2 ==> Gauss Curvature
00208 //           cflag=3 ==> True Maximal Curvature
00209 //   If we are off the grid, we can still evaluate the Laplacian; assuming, we
00210 //   are away from the molecular surface, it is simply equal to the DH factor.
00211 //
00212 // Authors:  Nathan Baker
00214 VPUBLIC int Vopot_curvature(Vopot *thee, double pt[3], int cflag,
00215   double *value) {
00216 
00217     Vatom *atom;
00218     int i, iatom;
00219     double u, T, charge, eps_w, xkappa, dist, size, val, *position, zkappa2;
00220     Valist *alist;
00221 
00222     VASSERT(thee != VNULL);
00223 
00224     eps_w = Vpbe_getSolventDiel(thee->pbe);
00225     xkappa = (1.0e10)*Vpbe_getXkappa(thee->pbe);
00226     zkappa2 = Vpbe_getZkappa2(thee->pbe);
00227     T = Vpbe_getTemperature(thee->pbe);
00228     alist = Vpbe_getValist(thee->pbe);
00229 
00230     u = 0;
00231 
00232     if (Vmgrid_curvature(thee->mgrid, pt, cflag, value)) return 1;
00233     else if (cflag != 1) {
00234         Vnm_print(2, "Vopot_curvature:  Off mesh!\n");
00235         return 1;
00236     } else {
00237 
00238         switch (thee->bcfl) {
00239 
00240             case BCFL_ZERO:
```

```
00241                 u = 0;
00242                 break;
00243
00244             case BCFL_SDH:
00245                 size = (1.0e-10)*Vpbe_getSoluteRadius(thee->pbe);
00246                 position = Vpbe_getSoluteCenter(thee->pbe);
00247                 charge = Vunit_ec*Vpbe_getSoluteCharge(thee->pbe);
00248                 dist = 0;
00249                 for (i=0; i<3; i++)
00250                   dist += VSQR(position[i] - pt[i]);
00251                 dist = (1.0e-10)*VSQRT(dist);
00252                 if (xkappa != 0.0)
00253                   u = zkappa2*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00254                 break;
00255
00256             case BCFL_MDH:
00257                 u = 0;
00258                 for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
00259                     atom = Valist_getAtom(alist, iatom);
00260                     position = Vatom_getPosition(atom);
00261                     charge = Vunit_ec*Vatom_getCharge(atom);
00262                     size = (1e-10)*Vatom_getRadius(atom);
00263                     dist = 0;
00264                     for (i=0; i<3; i++)
00265                       dist += VSQR(position[i] - pt[i]);
00266                     dist = (1.0e-10)*VSQRT(dist);
00267                     if (xkappa != 0.0)
00268                       val = zkappa2*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00269                     u = u + val;
00270                 }
00271                 break;
00272
00273             case BCFL_UNUSED:
00274                 Vnm_print(2, "Vopot_pot:  Invlid bcfl (%d)!\n", thee->bcfl);
00275                 return 0;
00276
00277             case BCFL_FOCUS:
00278                 Vnm_print(2, "Vopot_pot:  Invlid bcfl (%d)!\n", thee->bcfl);
00279                 return 0;
00280
00281             default:
00282                 Vnm_print(2, "Vopot_pot:  Bogus thee->bcfl flag (%d)!\n",
00283                   thee->bcfl);
00284                 return 0;
00285                 break;
00286         }
00287
00288         *value = u;
00289     }
00290
00291     return 1;
00292
00293 }
00294
00295 /* ///////////////////////////////////////////////////////////////////////////
00296 // Routine:  Vopot_gradient
00297 //
00298 // Authors:  Nathan Baker
00300 VPUBLIC int Vopot_gradient(Vopot *thee, double pt[3], double grad[3]) {
00301
00302     Vatom *atom;
00303     int iatom;
00304     double T, charge, eps_w, xkappa, size, val, *position;
00305     double dx, dy, dz, dist;
00306     Valist *alist;
00307
00308     VASSERT(thee != VNULL);
00309
00310     eps_w = Vpbe_getSolventDiel(thee->pbe);
00311     xkappa = (1.0e10)*Vpbe_getXkappa(thee->pbe);
00312     T = Vpbe_getTemperature(thee->pbe);
00313     alist = Vpbe_getValist(thee->pbe);
00314
00315
00316     if (!Vmgrid_gradient(thee->mgrid, pt, grad)) {
00317
00318         switch (thee->bcfl) {
00319
00320             case BCFL_ZERO:
00321                 grad[0] = 0.0;
00322                 grad[1] = 0.0;
```

```
00323                 grad[2] = 0.0;
00324                 break;
00325
00326             case BCFL_SDH:
00327                 grad[0] = 0.0;
00328                 grad[1] = 0.0;
00329                 grad[2] = 0.0;
00330                 size = (1.0e-10)*Vpbe_getSoluteRadius(thee->pbe);
00331                 position = Vpbe_getSoluteCenter(thee->pbe);
00332                 charge = Vunit_ec*Vpbe_getSoluteCharge(thee->pbe);
00333                 dx = position[0] - pt[0];
00334                 dy = position[1] - pt[1];
00335                 dz = position[2] - pt[2];
00336                 dist = VSQR(dx) + VSQR(dy) + VSQR(dz);
00337                 dist = (1.0e-10)*VSQRT(dist);
00338                 val = (charge)/(4*VPI*Vunit_eps0*eps_w);
00339                 if (xkappa != 0.0)
00340                   val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00341                 val = val*Vunit_ec/(Vunit_kb*T);
00342                 grad[0] = val*dx/dist*(-1.0/dist/dist + xkappa/dist);
00343                 grad[1] = val*dy/dist*(-1.0/dist/dist + xkappa/dist);
00344                 grad[2] = val*dz/dist*(-1.0/dist/dist + xkappa/dist);
00345                 break;
00346
00347             case BCFL_MDH:
00348                 grad[0] = 0.0;
00349                 grad[1] = 0.0;
00350                 grad[2] = 0.0;
00351                 for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
00352                     atom = Valist_getAtom(alist, iatom);
00353                     position = Vatom_getPosition(atom);
00354                     charge = Vunit_ec*Vatom_getCharge(atom);
00355                     size = (1e-10)*Vatom_getRadius(atom);
00356                     dx = position[0] - pt[0];
00357                     dy = position[1] - pt[1];
00358                     dz = position[2] - pt[2];
00359                     dist = VSQR(dx) + VSQR(dy) + VSQR(dz);
00360                     dist = (1.0e-10)*VSQRT(dist);
00361                     val = (charge)/(4*VPI*Vunit_eps0*eps_w);
00362                     if (xkappa != 0.0)
00363                       val = val*(exp(-xkappa*(dist-size))/(1+xkappa*size));
00364                     val = val*Vunit_ec/(Vunit_kb*T);
00365                     grad[0] += (val*dx/dist*(-1.0/dist/dist + xkappa/dist));
00366                     grad[1] += (val*dy/dist*(-1.0/dist/dist + xkappa/dist));
00367                     grad[2] += (val*dz/dist*(-1.0/dist/dist + xkappa/dist));
00368                 }
00369                 break;
00370
00371             case BCFL_UNUSED:
00372                 Vnm_print(2, "Vopot:  Invalid bcfl (%d)!\n", thee->bcfl);
00373                 return 0;
00374
00375             case BCFL_FOCUS:
00376                 Vnm_print(2, "Vopot:  Invalid bcfl (%d)!\n", thee->bcfl);
00377                 return 0;
00378
00379             default:
00380                 Vnm_print(2, "Vopot_pot:  Bogus thee->bcfl flag (%d)!\n",
00381                   thee->bcfl);
00382                 return 0;
00383                 break;
00384         }
00385
00386         return 1;
00387     }
00388
00389     return 1;
00390
00391 }
```

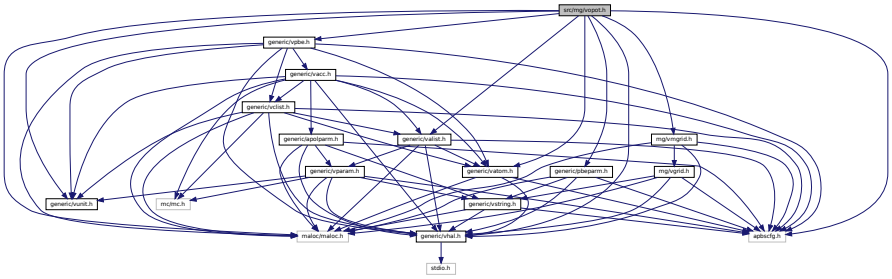
9.101 src/mg/vopot.h File Reference

Potential oracle for Cartesian mesh data.

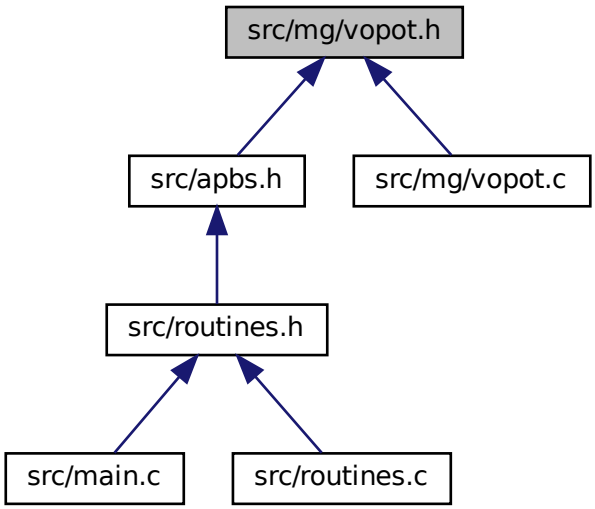
```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/vhal.h"
```

```
#include "generic/pbeparm.h"
#include "generic/vatom.h"
#include "generic/valist.h"
#include "generic/vunit.h"
#include "generic/vpbe.h"
#include "mg/vmgrid.h"
```

Include dependency graph for vopot.h:



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



Data Structures

- struct sVopot

  Electrostatic potential oracle for Cartesian mesh data.

Typedefs

- typedef struct sVopot Vopot

Declaration of the Vopot class as the Vopot structure.

Functions

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

  Construct Vopot object with values obtained from Vpmg_readDX (for example)
- VEXTERNC int Vopot_ctor2 (Vopot *thee, Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl)

  Initialize Vopot object with values obtained from Vpmg_readDX (for example)
- VEXTERNC int Vopot_pot (Vopot *thee, double x[3], double *pot)

  Get potential value (from mesh or approximation) at a point.
- VEXTERNC void Vopot_dtor (Vopot **thee)

  Object destructor.
- VEXTERNC void Vopot_dtor2 (Vopot *thee)

  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.

9.101.1 Detailed Description

Potential oracle for Cartesian mesh data.

Author

Nathan Baker

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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.

9.102 vopot.h

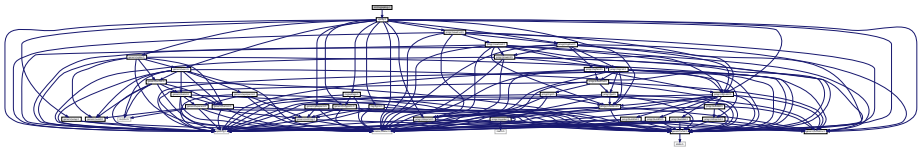
```
00001
00062 #ifndef _VOPOT_H_
00063 #define _VOPOT_H_
00064
00065 #include "apbscfg.h"
00066
00067 #include "maloc/maloc.h"
00068
00069 #include "generic/vhal.h"
00070 #include "generic/pbeparm.h"
00071 #include "generic/vatom.h"
00072 #include "generic/valist.h"
00073 #include "generic/vunit.h"
00074 #include "generic/vpbe.h"
00075 #include "generic/pbeparm.h"
00076 #include "mg/vmgrid.h"
00077
00083 struct sVopot {
00084
00085     Vmgrid *mgrid;
00087     Vpbe   *pbe;
00088     Vbcfl bcfl;
00090 };
00091
00096 typedef struct sVopot Vopot;
00097
00108 VEXTERNC Vopot*  Vopot_ctor(Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl);
00109
00121 VEXTERNC int Vopot_ctor2(Vopot *thee, Vmgrid *mgrid, Vpbe *pbe, Vbcfl bcfl);
00122
00131 VEXTERNC int Vopot_pot(Vopot *thee, double x[3], double *pot);
00132
00138 VEXTERNC void Vopot_dtor(Vopot **thee);
00139
00145 VEXTERNC void Vopot_dtor2(Vopot *thee);
00146
00160 VEXTERNC int Vopot_curvature(Vopot *thee, double pt[3], int cflag, double
00161   *curv);
00162
00171 VEXTERNC int Vopot_gradient(Vopot *thee, double pt[3], double grad[3] );
00172
00173
00174 #endif
```

9.103 src/mg/vpmg.c File Reference

Class Vpmg methods.

```
#include "vpmg.h"
```

Include dependency graph for vpmg.c:



Functions

- VPUBLIC unsigned long int Vpmg_memChk (Vpmg *thee)

  Return the memory used by this structure (and its contents) in bytes.
- VPUBLIC void Vpmg_printColComp (Vpmg *thee, char path[72], char title[72], char mxtype[3], int flag)

  Print out a column-compressed sparse matrix in Harwell-Boeing format.
- VPUBLIC Vpmg * Vpmg_ctor (Vpmgp *pmgp, Vpbe *pbe, int focusFlag, Vpmg *pmgOLD, MGparm *mgparm, PBEparm_calcEnergy energyFlag)

  Constructor for the Vpmg class (allocates new memory)
- VPUBLIC int Vpmg_ctor2 (Vpmg *thee, Vpmgp *pmgp, Vpbe *pbe, int focusFlag, Vpmg *pmgOLD, MGparm *mgparm, PBEparm_calcEnergy energyFlag)

  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 **bcfl1new** (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 use↩ KappaMap, 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.

9.103.1 Detailed Description

Class Vpmg methods.

Author

Nathan Baker

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
```

Definition in file vpmg.c.

9.103.2 Function Documentation

9.103.2.1 bcCalc()

```
VPRIVATE void bcCalc (
            Vpmg * thee )
```

Fill boundary condition arrays.

Author

Nathan Baker

Definition at line 4382 of file vpmg.c.

9.103.2.2 bcfl1()

```
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 2564 of file vpmg.c.

9.103.2.3 bspline2()

```
VPRIVATE double bspline2 (
            double x )
```

Evaluate a cubic B-spline.

Author

Nathan Baker

Returns

Cubic B-spline value

Parameters

x	Position

Definition at line 5496 of file vpmg.c.

9.103.2.4 bspline4()

```
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 7136 of file vpmg.c.

9.103.2.5 d2bspline4()

```
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 7202 of file vpmg.c.

9.103.2.6 d3bspline4()

```
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 7229 of file vpmg.c.

9.103.2.7 dbspline2()

```
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 5512 of file vpmg.c.

9.103.2.8 dbspline4()

```
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 7170 of file vpmg.c.

9.103.2.9 fillcoCharge()

```
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 5287 of file vpmg.c.

9.103.2.10 fillcoChargeMap()

```
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 5343 of file vpmg.c.

9.103.2.11 fillcoChargeSpline1()

```
VPRIVATE void fillcoChargeSpline1 (
            Vpmg * thee )
```

Fill source term charge array from linear interpolation.

Author

Nathan Baker

Definition at line 5391 of file vpmg.c.

9.103.2.12 fillcoChargeSpline2()

```
VPRIVATE void fillcoChargeSpline2 (
            Vpmg * thee )
```

Fill source term charge array from cubic spline interpolation.

Author

Nathan Baker

Definition at line 5528 of file vpmg.c.

9.103.2.13 fillcoCoef()

```
VPRIVATE void fillcoCoef (
            Vpmg * thee )
```

Top-level driver to fill all operator coefficient arrays.

Author

Nathan Baker

Definition at line 5247 of file vpmg.c.

9.103.2.14 fillcoCoefMap()

```
VPRIVATE void fillcoCoefMap (
            Vpmg * thee )
```

Fill operator coefficient arrays from pre-calculated maps.

Author

Nathan Baker

Definition at line 4489 of file vpmg.c.

9.103.2.15 fillcoCoefMol()

```
VPRIVATE void fillcoCoefMol (
            Vpmg * thee )
```

Fill operator coefficient arrays from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4612 of file vpmg.c.

9.103.2.16 fillcoCoefMolDiel()

```
VPRIVATE void fillcoCoefMolDiel (
            Vpmg * thee )
```

Fill differential operator coefficient arrays from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4726 of file vpmg.c.

9.103.2.17 fillcoCoefMolDielNoSmooth()

```
VPRIVATE void fillcoCoefMolDielNoSmooth (
            Vpmg * thee )
```

Fill differential operator coefficient arrays from a molecular surface calculation without smoothing.

Author

Nathan Baker

Definition at line 4737 of file vpmg.c.

9.103.2.18 fillcoCoefMolDielSmooth()

```
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 4891 of file vpmg.c.

9.103.2.19 fillcoCoefMolIon()

```
VPRIVATE void fillcoCoefMolIon (
            Vpmg * thee )
```

Fill ion (nonlinear) operator coefficient array from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4628 of file vpmg.c.

9.103.2.20 fillcoCoefSpline()

```
VPRIVATE void fillcoCoefSpline (
            Vpmg * thee )
```

Fill operator coefficient arrays from a spline-based surface calculation.

Author

Nathan Baker

Definition at line 5022 of file vpmg.c.

9.103.2.21 fillcoCoefSpline3()

```
VPRIVATE void fillcoCoefSpline3 (
            Vpmg * thee )
```

Fill operator coefficient arrays from a 5th order polynomial based surface calculation.

Author

Michael Schnieders

Definition at line 10430 of file vpmg.c.

9.103.2.22 fillcoCoefSpline4()

```
VPRIVATE void fillcoCoefSpline4 (
            Vpmg * thee )
```

Fill operator coefficient arrays from a 7th order polynomial based surface calculation.

Author

Michael Schnieders

Definition at line 9939 of file vpmg.c.

9.103.2.23 fillcoPermanentMultipole()

```
VPUBLIC void fillcoPermanentMultipole (
            Vpmg * thee )
```

Fill source term charge array for the use of permanent multipoles.

Author

Michael Schnieders

Definition at line 7240 of file vpmg.c.

9.103.2.24 markSphere()

```
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 6849 of file vpmg.c.

9.103.2.25 multipolebc()

```
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 3487 of file vpmg.c.

9.103.2.26 qfForceSpline1()

```
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 6311 of file vpmg.c.

9.103.2.27 qfForceSpline2()

```
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 6448 of file vpmg.c.

9.103.2.28 qfForceSpline4()

```
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 6561 of file vpmg.c.

9.103.2.29 VFCHI4()

```
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 7132 of file vpmg.c.

9.103.2.30 Vpmg_polarizEnergy()

```
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 1148 of file vpmg.c.

9.103.2.31 Vpmg_qfEnergyPoint()

```
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 1704 of file vpmg.c.

9.103.2.32 Vpmg_qfEnergyVolume()

```
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 1861 of file vpmg.c.

9.103.2.33 Vpmg_qmEnergySMPBE()

```
VPUBLIC double Vpmg_qmEnergySMPBE (
            Vpmg * thee,
            int extFlag )
```

Vpmg_qmEnergy for SMPBE.

Author

Vincent Chu

Definition at line 1490 of file vpmg.c.

9.103.2.34 Vpmg_splineSelect()

```
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 1893 of file vpmg.c.

9.103.2.35 zlapSolve()

```
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 6898 of file vpmg.c.

9.104 vpmg.c

```
00001
00073 #include "vpmg.h"
00074
00075 VEMBED(rcsid="$Id$")
00076
00077 #if !defined(VINLINE_VPMG)
00078
00079 VPUBLIC unsigned long int Vpmg_memChk(Vpmg *thee) {
00080     if (thee == VNULL) return 0;
00081     return Vmem_bytes(thee->vmem);
00082 }
00083
00084 #endif /* if !defined(VINLINE_VPMG) */
00085
```

```
00086
00087 VPUBLIC void Vpmg_printColComp(Vpmg *thee, char path[72], char title[72],
00088   char mxtype[3], int flag) {
00089
00090     int nn, nxm2, nym2, nzm2, ncol, nrow, nonz;
00091     double *nzval;
00092     int *colptr, *rowind;
00093
00094     /* Calculate the total number of unknowns */
00095     nxm2 = thee->pmgp->nx - 2;
00096     nym2 = thee->pmgp->ny - 2;
00097     nzm2 = thee->pmgp->nz - 2;
00098     nn = nxm2*nym2*nzm2;
00099     ncol = nn;
00100     nrow = nn;
00101
00102     /* Calculate the number of non-zero matrix entries:
00103      *    nn       nonzeros on diagonal
00104      *    nn-1     nonzeros on first off-diagonal
00105      *    nn-nx    nonzeros on second off-diagonal
00106      *    nn-nx*ny nonzeros on third off-diagonal
00107      *
00108      *    7*nn-2*nx*ny-2*nx-2 TOTAL non-zeros
00109      */
00110     nonz = 7*nn - 2*nxm2*nym2 - 2*nxm2 - 2;
00111     nzval  = (double*)Vmem_malloc(thee->vmem, nonz, sizeof(double));
00112     rowind = (int*)Vmem_malloc(thee->vmem, nonz, sizeof(int));
00113     colptr = (int*)Vmem_malloc(thee->vmem, (ncol+1), sizeof(int));
00114
00115 #ifndef VAPBSQUIET
00116     Vnm_print(1, "Vpmg_printColComp:  Allocated space for %d nonzeros\n",
00117       nonz);
00118 #endif
00119
00120     bcolcomp(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00121       nzval, rowind, colptr, &flag);
00122
00123
00124 #if 0
00125     for (i=0; i<nn; i++) {
00126         Vnm_print(1, "nnz(%d) = %g\n", i, nzval[i]);
00127     }
00128 #endif
00129
00130     /* I do not understand why I need to pass nzval in this way, but it
00131      * works... */
00132     pcolcomp(&nrow, &ncol, &nonz, &(nzval[0]), rowind, colptr, path, title,
00133       mxtype);
00134
00135     Vmem_free(thee->vmem, (ncol+1), sizeof(int), (void **)&colptr);
00136     Vmem_free(thee->vmem, nonz, sizeof(int), (void **)&rowind);
00137     Vmem_free(thee->vmem, nonz, sizeof(double), (void **)&nzval);
00138
00139 }
00140
00141 VPUBLIC Vpmg* Vpmg_ctor(Vpmgp *pmgp, Vpbe *pbe, int focusFlag,
00142         Vpmg *pmgOLD, MGparm *mgparm, PBEparm_calcEnergy energyFlag) {
00143
00144     Vpmg *thee = VNULL;
00145
00146     thee = (Vpmg*)Vmem_malloc(VNULL, 1, sizeof(Vpmg) );
00147     VASSERT(thee != VNULL);
00148     VASSERT( Vpmg_ctor2(thee, pmgp, pbe, focusFlag, pmgOLD, mgparm,
00149                 energyFlag) );
00150     return thee;
00151 }
00152
00153 VPUBLIC int Vpmg_ctor2(Vpmg *thee, Vpmgp *pmgp, Vpbe *pbe, int focusFlag,
00154                        Vpmg *pmgOLD, MGparm *mgparm, PBEparm_calcEnergy energyFlag) {
00155
00156     int i, j, nion;
00157     double ionConc[MAXION], ionQ[MAXION], ionRadii[MAXION], zkappa2, zks2;
00158     double ionstr, partMin[3], partMax[3];
00159  size_t size;
00160
00161     /* Get the parameters */
00162     VASSERT(pmgp != VNULL);
00163     VASSERT(pbe != VNULL);
00164     thee->pmgp = pmgp;
00165     thee->pbe = pbe;
00166
```

```
00167     /* Set up the memory */
00168     thee->vmem = Vmem_ctor("APBS:VPMG");
00169
00170
00171
00173     /* Initialize ion concentrations and valencies in PMG routines */
00174     zkappa2 = Vpbe_getZkappa2(thee->pbe);
00175     ionstr = Vpbe_getBulkIonicStrength(thee->pbe);
00176     if (ionstr > 0.0) zks2 = 0.5/ionstr;
00177     else zks2 = 0.0;
00178     Vpbe_getIons(thee->pbe, &nion, ionConc, ionRadii, ionQ);
00179
00180     /* TEMPORARY USEAQUA */
00181         /* Calculate storage requirements */
00182     if(mgparm->useAqua == 0){
00183             Vpmgp_size(thee->pmgp);
00184     }else{
00185         VABORT_MSG0("Aqua is currently disabled");
00186     }
00187
00188     /* We need some additional storage if: nonlinear & newton OR cgmg */
00189     /* SMPBE Added - nonlin = 2 added since it mimics NPBE */
00190     if ( ( ((thee->pmgp->nonlin == NONLIN_NPBE) || (thee->pmgp->nonlin == NONLIN_SMPBE))
00191            && (thee->pmgp->meth == VSOL_Newton) ) || (thee->pmgp->meth == VSOL_CGMG) )
00192     {
00193         thee->pmgp->nrwk += (2*(thee->pmgp->nf));
00194     }
00195
00196
00197         if (thee->pmgp->iinfo > 1) {
00198             Vnm_print(2, "Vpmg_ctor2:  PMG chose nx = %d, ny = %d, nz = %d\n",
00199                               thee->pmgp->nx, thee->pmgp->ny, thee->pmgp->nz);
00200             Vnm_print(2, "Vpmg_ctor2:  PMG chose nlev = %d\n",
00201                               thee->pmgp->nlev);
00202             Vnm_print(2, "Vpmg_ctor2:  PMG chose nxc = %d, nyc = %d, nzc = %d\n",
00203                               thee->pmgp->nxc, thee->pmgp->nyc, thee->pmgp->nzc);
00204             Vnm_print(2, "Vpmg_ctor2:  PMG chose nf = %d, nc = %d\n",
00205                               thee->pmgp->nf, thee->pmgp->nc);
00206             Vnm_print(2, "Vpmg_ctor2:  PMG chose narr = %d, narrc = %d\n",
00207                               thee->pmgp->narr, thee->pmgp->narrc);
00208             Vnm_print(2, "Vpmg_ctor2:  PMG chose n_rpc = %d, n_iz = %d, n_ipc = %d\n",
00209                               thee->pmgp->n_rpc, thee->pmgp->n_iz, thee->pmgp->n_ipc);
00210             Vnm_print(2, "Vpmg_ctor2:  PMG chose nrwk = %d, niwk = %d\n",
00211                               thee->pmgp->nrwk, thee->pmgp->niwk);
00212         }
00213
00214
00215
00216     /* Allocate boundary storage */
00217     thee->gxcf = (double *)Vmem_malloc(
00218         thee->vmem,
00219         10*(thee->pmgp->ny)*(thee->pmgp->nz),
00220         sizeof(double)
00221         );
00222
00223     thee->gycf = (double *)Vmem_malloc(
00224         thee->vmem,
00225         10*(thee->pmgp->nx)*(thee->pmgp->nz),
00226         sizeof(double)
00227         );
00228
00229     thee->gzcf = (double *)Vmem_malloc(
00230         thee->vmem,
00231         10*(thee->pmgp->nx)*(thee->pmgp->ny),
00232         sizeof(double)
00233         );
00234
00235
00236
00237     /* Warn users if they are using BCFL_MAP that
00238        we do not include external energies */
00239     if (thee->pmgp->bcfl == BCFL_MAP)
00240         Vnm_print(2,"Vpmg_ctor2: \nWarning: External energies are not used in BCFL_MAP calculations!\n");
00241
00242     if (focusFlag) {
00243
00244         /* Overwrite any default or user-specified boundary condition
00245         * arguments; we are now committed to a calculation via focusing */
00246         if (thee->pmgp->bcfl != BCFL_FOCUS) {
00247             Vnm_print(2,
00248                       "Vpmg_ctor2: reset boundary condition flag to BCFL_FOCUS!\n");
```

```
00249             thee->pmgp->bcfl = BCFL_FOCUS;
00250         }
00251
00252         /* Fill boundaries */
00253         Vnm_print(0, "Vpmg_ctor2:  Filling boundary with old solution!\n");
00254         focusFillBound(thee, pmgOLD);
00255
00256         /* Calculate energetic contributions from region outside focusing
00257             * domain */
00258         if (energyFlag != PCE_NO) {
00259
00260             if (mgparm->type == MCT_PARALLEL) {
00261
00262                 for (j=0; j<3; j++) {
00263                     partMin[j] = mgparm->partDisjCenter[j]
00264                     - 0.5*mgparm->partDisjLength[j];
00265                     partMax[j] = mgparm->partDisjCenter[j]
00266                         + 0.5*mgparm->partDisjLength[j];
00267                 }
00268
00269             } else {
00270                 for (j=0; j<3; j++) {
00271                     partMin[j] = mgparm->center[j] - 0.5*mgparm->glen[j];
00272                     partMax[j] = mgparm->center[j] + 0.5*mgparm->glen[j];
00273                 }
00274             }
00275             extEnergy(thee, pmgOLD, energyFlag, partMin, partMax,
00276                       mgparm->partDisjOwnSide);
00277         }
00278
00279     } else {
00280
00281         /* Ignore external energy contributions */
00282         thee->extQmEnergy = 0;
00283         thee->extDiEnergy = 0;
00284         thee->extQfEnergy = 0;
00285     }
00286
00287     /* Allocate partition vector storage */
00288     size = (thee->pmgp->nx)*(thee->pmgp->ny)*(thee->pmgp->nz);
00289     thee->pvec = (double *)Vmem_malloc(
00290         thee->vmem,
00291         size,
00292         sizeof(double)
00293         );
00294
00295     /* Allocate remaining storage */
00296     thee->iparm  = (   int *)Vmem_malloc(thee->vmem,               100, sizeof(   int));
00297     thee->rparm  = (double *)Vmem_malloc(thee->vmem,               100, sizeof(double));
00298     thee->iwork  = (   int *)Vmem_malloc(thee->vmem,   thee->pmgp->niwk, sizeof(   int));
00299     thee->rwork  = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->nrwk, sizeof(double));
00300     thee->charge = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
00301     thee->kappa  = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
00302     thee->pot    = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
00303     thee->epsx   = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
00304     thee->epsy   = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
00305     thee->epsz   = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
00306     thee->a1cf   = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
00307     thee->a2cf   = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
00308     thee->a3cf   = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
00309     thee->ccf    = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
00310     thee->fcf    = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
00311     thee->tcf    = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
00312     thee->u      = (double *)Vmem_malloc(thee->vmem,   thee->pmgp->narr, sizeof(double));
00313     thee->xf     = (double *)Vmem_malloc(thee->vmem, 5*(thee->pmgp->nx), sizeof(double));
00314     thee->yf     = (double *)Vmem_malloc(thee->vmem, 5*(thee->pmgp->ny), sizeof(double));
00315     thee->zf     = (double *)Vmem_malloc(thee->vmem, 5*(thee->pmgp->nz), sizeof(double));
00316
00317
00318
00319     /* Packs parameters into the iparm and rparm arrays */
00320     Vpackmg(thee->iparm, thee->rparm, &(thee->pmgp->nrwk), &(thee->pmgp->niwk),
00321             &(thee->pmgp->nx), &(thee->pmgp->ny), &(thee->pmgp->nz),
00322             &(thee->pmgp->nlev), &(thee->pmgp->nu1), &(thee->pmgp->nu2),
00323             &(thee->pmgp->mgkey), &(thee->pmgp->itmax), &(thee->pmgp->istop),
00324             &(thee->pmgp->ipcon), &(thee->pmgp->nonlin), &(thee->pmgp->mgsmoo),
00325             &(thee->pmgp->mgprol), &(thee->pmgp->mgcoar), &(thee->pmgp->mgsolv),
00326             &(thee->pmgp->mgdisc), &(thee->pmgp->iinfo), &(thee->pmgp->errtol),
00327             &(thee->pmgp->ipkey), &(thee->pmgp->omegal), &(thee->pmgp->omegan),
00328             &(thee->pmgp->irite), &(thee->pmgp->iperf));
00329
```

```
00330
00331
00332     /* Currently for SMPBE type calculations we do not want to apply a scale
00333         factor to the ionConc */
00340     switch(pmgp->ipkey){
00341
00342         case IPKEY_SMPBE:
00343
00344                         Vmypdefinitsmpbe(&nion, ionQ, ionConc, &pbe->smvolume, &pbe->smsize);
00345             break;
00346
00347
00348
00349         case IPKEY_NPBE:
00350
00351             /* Else adjust the inoConc by scaling factor zks2 */
00352             for (i=0; i<nion; i++)
00353                 ionConc[i] = zks2 * ionConc[i];
00354
00355                         Vmypdefinitnpbe(&nion, ionQ, ionConc);
00356             break;
00357
00358
00359
00360         case IPKEY_LPBE:
00361
00362             /* Else adjust the inoConc by scaling factor zks2 */
00363             for (i=0; i<nion; i++)
00364                             ionConc[i] = zks2 * ionConc[i];
00365
00366             Vmypdefinitlpbe(&nion, ionQ, ionConc);
00367             break;
00368
00369
00370
00371         default:
00372             Vnm_print(2, "PMG: Warning: PBE structure not initialized!\n");
00373             /* Else adjust the inoConc by scaling factor zks2 */
00374             for (i=0; i<nion; i++)
00375                 ionConc[i] = zks2 * ionConc[i];
00376             break;
00377     }
00378
00379     /* Set the default chargeSrc for 5th order splines */
00380     thee->chargeSrc = mgparm->chgs;
00381
00382     /* Turn off restriction of observable calculations to a specific
00383     * partition */
00384     Vpmg_unsetPart(thee);
00385
00386     /* The coefficient arrays have not been filled */
00387     thee->filled = 0;
00388
00389
00390     /*
00391      * TODO: Move the dtor out of here. The current ctor is done in routines.c,
00392      *       This was originally moved out to kill a memory leak. The dtor has
00393      *       has been removed from initMG and placed back here to keep memory
00394      *       usage low. killMG has been modified accordingly.
00395      */
00396     Vpmg_dtor(&pmgOLD);
00397
00398     return 1;
00399 }
00400
00401 VPUBLIC int Vpmg_solve(Vpmg *thee) {
00402
00403     int i,
00404         nx,
00405         ny,
00406         nz,
00407         n;
00408     double zkappa2;
00409
00410     nx = thee->pmgp->nx;
00411     ny = thee->pmgp->ny;
00412     nz = thee->pmgp->nz;
00413     n = nx*ny*nz;
00414
00415     if (!(thee->filled)) {
00416         Vnm_print(2, "Vpmg_solve:  Need to call Vpmg_fillco()!\n");
```

```
00417         return 0;
00418     }
00419
00420     /* Fill the "true solution" array */
00421     for (i=0; i<n; i++) {
00422         thee->tcf[i] = 0.0;
00423     }
00424
00425     /* Fill the RHS array */
00426     for (i=0; i<n; i++) {
00427         thee->fcf[i] = thee->charge[i];
00428     }
00429
00430     /* Fill the operator coefficient array. */
00431     for (i=0; i<n; i++) {
00432         thee->a1cf[i] = thee->epsx[i];
00433         thee->a2cf[i] = thee->epsy[i];
00434         thee->a3cf[i] = thee->epsz[i];
00435     }
00436
00437     /* Fill the nonlinear coefficient array by multiplying the kappa
00438      * accessibility array (containing values between 0 and 1) by zkappa2. */
00439     zkappa2 = Vpbe_getZkappa2(thee->pbe);
00440     if (zkappa2 > VPMGSMALL) {
00441         for (i=0; i<n; i++) {
00442             thee->ccf[i] = zkappa2*thee->kappa[i];
00443         }
00444     } else {
00445         for (i=0; i<n; i++) {
00446             thee->ccf[i] = 0.0;
00447         }
00448     }
00449
00450     switch(thee->pmgp->meth) {
00451         /* CGMG (linear) */
00452         case VSOL_CGMG:
00453
00454             if (thee->pmgp->iinfo > 1)
00455                 Vnm_print(2, "Driving with CGMGDRIV\n");
00456
00457             VABORT_MSG0("CGMGDRIV is not currently supported");
00458             break;
00459
00460         /* Newton (nonlinear) */
00461         case VSOL_Newton:
00462
00463             if (thee->pmgp->iinfo > 1)
00464                 Vnm_print(2, "Driving with NEWDRIV\n");
00465
00466             Vnewdriv
00467                       (thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00468                        thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00469                        thee->gzcf, thee->a1cf, thee->a2cf, thee->a3cf, thee->ccf,
00470                        thee->fcf, thee->tcf);
00471             break;
00472
00473         /* MG (linear/nonlinear) */
00474         case VSOL_MG:
00475
00476             if (thee->pmgp->iinfo > 1)
00477                 Vnm_print(2, "Driving with MGDRIV\n");
00478
00479             Vmgdriv(thee->iparm, thee->rparm, thee->iwork, thee->rwork,
00480                                      thee->u, thee->xf, thee->yf, thee->zf, thee->gxcf, thee->gycf,
00481                                      thee->gzcf, thee->a1cf, thee->a2cf, thee->a3cf, thee->ccf,
00482                                      thee->fcf, thee->tcf);
00483             break;
00484
00485         /* CGHS (linear/nonlinear) */
00486         case VSOL_CG:
00487
00488             if (thee->pmgp->iinfo > 1)
00489                 Vnm_print(2, "Driving with NCGHSDRIV\n");
00490
00491             VABORT_MSG0("NCGHSDRIV is not currently supported");
00492             break;
00493
00494         /* SOR (linear/nonlinear) */
00495         case VSOL_SOR:
00496
00497             if (thee->pmgp->iinfo > 1)
```

```
00498                 Vnm_print(2, "Driving with NSORDRIV\n");
00499
00500             VABORT_MSG0("NSORDRIV is not currently supported");
00501             break;
00502
00503         /* GSRB (linear/nonlinear) */
00504         case VSOL_RBGS:
00505
00506             if (thee->pmgp->iinfo > 1)
00507                 Vnm_print(2, "Driving with NGSRBDRIV\n");
00508
00509             VABORT_MSG0("NGSRBDRIV is not currently supported");
00510             break;
00511
00512         /* WJAC (linear/nonlinear) */
00513         case VSOL_WJ:
00514
00515             if (thee->pmgp->iinfo > 1)
00516                 Vnm_print(2, "Driving with NWJACDRIV\n");
00517
00518             VABORT_MSG0("NWJACDRIV is not currently supported");
00519             break;
00520
00521         /* RICH (linear/nonlinear) */
00522         case VSOL_Richardson:
00523
00524             if (thee->pmgp->iinfo > 1)
00525                 Vnm_print(2, "Driving with NRICHDRIV\n");
00526
00527             VABORT_MSG0("NRICHDRIV is not currently supported");
00528             break;
00529
00530         /* CGMG (linear) TEMPORARY USEAQUA */
00531         case VSOL_CGMGAqua:
00532
00533             if (thee->pmgp->iinfo > 1)
00534                 Vnm_print(2, "Driving with CGMGDRIVAQUA\n");
00535
00536             VABORT_MSG0("CGMGDRIVAQUA is not currently supported");
00537             break;
00538
00539         /* Newton (nonlinear) TEMPORARY USEAQUA */
00540         case VSOL_NewtonAqua:
00541
00542             if (thee->pmgp->iinfo > 1)
00543                 Vnm_print(2, "Driving with NEWDRIVAQUA\n");
00544
00545             VABORT_MSG0("NEWDRIVAQUA is not currently supported");
00546             break;
00547
00548         /* Error handling */
00549         default:
00550             Vnm_print(2, "Vpmg_solve: invalid solver method key (%d)\n",
00551               thee->pmgp->key);
00552             return 0;
00553             break;
00554     }
00555
00556     return 1;
00557
00558 }
00559
00560
00561 VPUBLIC void Vpmg_dtor(Vpmg **thee) {
00562
00563     if ((*thee) != VNULL) {
00564         Vpmg_dtor2(*thee);
00565         Vmem_free(VNULL, 1, sizeof(Vpmg), (void **)thee);
00566         (*thee) = VNULL;
00567     }
00568
00569 }
00570
00571 VPUBLIC void Vpmg_dtor2(Vpmg *thee) {
00572
00573     /* Clean up the storage */
00574
00575     Vmem_free(thee->vmem,              100, sizeof(int),
00576       (void **)&(thee->iparm));
00577     Vmem_free(thee->vmem,              100, sizeof(double),
00578         (void **)&(thee->rparm));
```

```
00579     Vmem_free(thee->vmem, thee->pmgp->niwk, sizeof(int),
00580       (void **)&(thee->iwork));
00581     Vmem_free(thee->vmem, thee->pmgp->nrwk, sizeof(double),
00582       (void **)&(thee->rwork));
00583     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00584       (void **)&(thee->charge));
00585     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00586       (void **)&(thee->kappa));
00587     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00588               (void **)&(thee->pot));
00589     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00590       (void **)&(thee->epsx));
00591     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00592       (void **)&(thee->epsy));
00593     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00594       (void **)&(thee->epsz));
00595     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00596       (void **)&(thee->a1cf));
00597     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00598       (void **)&(thee->a2cf));
00599     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00600       (void **)&(thee->a3cf));
00601     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00602       (void **)&(thee->ccf));
00603     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00604       (void **)&(thee->fcf));
00605     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00606       (void **)&(thee->tcf));
00607     Vmem_free(thee->vmem, thee->pmgp->narr, sizeof(double),
00608       (void **)&(thee->u));
00609     Vmem_free(thee->vmem, 5*(thee->pmgp->nx), sizeof(double),
00610       (void **)&(thee->xf));
00611     Vmem_free(thee->vmem, 5*(thee->pmgp->ny), sizeof(double),
00612       (void **)&(thee->yf));
00613     Vmem_free(thee->vmem, 5*(thee->pmgp->nz), sizeof(double),
00614       (void **)&(thee->zf));
00615     Vmem_free(thee->vmem, 10*(thee->pmgp->ny)*(thee->pmgp->nz), sizeof(double),
00616       (void **)&(thee->gxcf));
00617     Vmem_free(thee->vmem, 10*(thee->pmgp->nx)*(thee->pmgp->nz), sizeof(double),
00618       (void **)&(thee->gycf));
00619     Vmem_free(thee->vmem, 10*(thee->pmgp->nx)*(thee->pmgp->ny), sizeof(double),
00620       (void **)&(thee->gzcf));
00621     Vmem_free(thee->vmem, (thee->pmgp->nx)*(thee->pmgp->ny)*(thee->pmgp->nz),
00622       sizeof(double), (void **)&(thee->pvec));
00623
00624     Vmem_dtor(&(thee->vmem));
00625 }
00626
00627 VPUBLIC void Vpmg_setPart(Vpmg *thee, double lowerCorner[3],
00628         double upperCorner[3], int bflags[6]) {
00629
00630     Valist *alist;
00631     Vatom *atom;
00632     int i, j, k, nx, ny, nz;
00633     double xmin, ymin, zmin, x, y, z, hx, hy, hzed, xok, yok, zok;
00634     double x0,x1,y0,y1,z0,z1;
00635
00636     nx = thee->pmgp->nx;
00637     ny = thee->pmgp->ny;
00638     nz = thee->pmgp->nz;
00639     hx = thee->pmgp->hx;
00640     hy = thee->pmgp->hy;
00641     hzed = thee->pmgp->hzed;
00642     xmin = thee->pmgp->xcent - 0.5*hx*(nx-1);
00643     ymin = thee->pmgp->ycent - 0.5*hy*(ny-1);
00644     zmin = thee->pmgp->zcent - 0.5*hzed*(nz-1);
00645
00646     xok = 0;
00647     yok = 0;
00648     zok = 0;
00649
00650     /* We need have called Vpmg_fillco first */
00651
00652     alist = thee->pbe->alist;
00653
00654     Vnm_print(0, "Vpmg_setPart:  lower corner = (%g, %g, %g)\n",
00655       lowerCorner[0], lowerCorner[1], lowerCorner[2]);
00656     Vnm_print(0, "Vpmg_setPart:  upper corner = (%g, %g, %g)\n",
00657       upperCorner[0], upperCorner[1], upperCorner[2]);
00658     Vnm_print(0, "Vpmg_setPart:  actual minima = (%g, %g, %g)\n",
00659       xmin, ymin, zmin);
```

```
00660     Vnm_print(0, "Vpmg_setPart:  actual maxima = (%g, %g, %g)\n",
00661       xmin+hx*(nx-1), ymin+hy*(ny-1), zmin+hzed*(nz-1));
00662     Vnm_print(0, "Vpmg_setPart:  bflag[FRONT] = %d\n",
00663       bflags[VAPBS_FRONT]);
00664     Vnm_print(0, "Vpmg_setPart:  bflag[BACK] = %d\n",
00665       bflags[VAPBS_BACK]);
00666     Vnm_print(0, "Vpmg_setPart:  bflag[LEFT] = %d\n",
00667       bflags[VAPBS_LEFT]);
00668     Vnm_print(0, "Vpmg_setPart:  bflag[RIGHT] = %d\n",
00669       bflags[VAPBS_RIGHT]);
00670     Vnm_print(0, "Vpmg_setPart:  bflag[UP] = %d\n",
00671       bflags[VAPBS_UP]);
00672     Vnm_print(0, "Vpmg_setPart:  bflag[DOWN] = %d\n",
00673       bflags[VAPBS_DOWN]);
00674
00675     /* Identify atoms as inside, outside, or on the border
00676        If on the border, use the bflags to determine if there
00677        is an adjacent processor - if so, this atom should be equally
00678        shared. */
00679
00680     for (i=0; i<Valist_getNumberAtoms(alist); i++) {
00681         atom = Valist_getAtom(alist, i);
00682
00683         if ((atom->position[0] < upperCorner[0]) &&
00684             (atom->position[0] > lowerCorner[0])) xok = 1;
00685         else {
00686             if ((VABS(atom->position[0] - lowerCorner[0]) < VPMGSMALL) &&
00687                 (bflags[VAPBS_LEFT] == 0)) xok = 1;
00688             else if ((VABS(atom->position[0] - lowerCorner[0]) < VPMGSMALL) &&
00689                 (bflags[VAPBS_LEFT] == 1)) xok = 0.5;
00690             else if ((VABS(atom->position[0] - upperCorner[0]) < VPMGSMALL) &&
00691                 (bflags[VAPBS_RIGHT] == 0)) xok = 1;
00692             else if ((VABS(atom->position[0] - upperCorner[0]) < VPMGSMALL) &&
00693                 (bflags[VAPBS_RIGHT] == 1)) xok = 0.5;
00694             else xok = 0;
00695         }
00696         if ((atom->position[1] < upperCorner[1]) &&
00697             (atom->position[1] > lowerCorner[1])) yok = 1;
00698         else {
00699             if ((VABS(atom->position[1] - lowerCorner[1]) < VPMGSMALL) &&
00700                 (bflags[VAPBS_BACK] == 0)) yok = 1;
00701             else if ((VABS(atom->position[1] - lowerCorner[1]) < VPMGSMALL) &&
00702                 (bflags[VAPBS_BACK] == 1)) yok = 0.5;
00703             else if ((VABS(atom->position[1] - upperCorner[1]) < VPMGSMALL) &&
00704                 (bflags[VAPBS_FRONT] == 0)) yok = 1;
00705             else if ((VABS(atom->position[1] - upperCorner[1]) < VPMGSMALL) &&
00706                 (bflags[VAPBS_FRONT] == 1)) yok = 0.5;
00707             else yok = 0;
00708         }
00709         if ((atom->position[2] < upperCorner[2]) &&
00710             (atom->position[2] > lowerCorner[2])) zok = 1;
00711         else {
00712             if ((VABS(atom->position[2] - lowerCorner[2]) < VPMGSMALL) &&
00713                 (bflags[VAPBS_DOWN] == 0)) zok = 1;
00714             else if ((VABS(atom->position[2] - lowerCorner[2]) < VPMGSMALL) &&
00715                 (bflags[VAPBS_DOWN] == 1)) zok = 0.5;
00716             else if ((VABS(atom->position[2] - upperCorner[2]) < VPMGSMALL) &&
00717                 (bflags[VAPBS_UP] == 0)) zok = 1;
00718             else if ((VABS(atom->position[2] - upperCorner[2]) < VPMGSMALL) &&
00719                 (bflags[VAPBS_UP] == 1)) zok = 0.5;
00720             else zok = 0;
00721         }
00722
00723         atom->partID = xok*yok*zok;
00724         /*
00725         Vnm_print(1, "DEBUG (%s, %d):  atom->position[0] - upperCorner[0] = %g\n",
00726                   __FILE__, __LINE__, atom->position[0] - upperCorner[0]);
00727         Vnm_print(1, "DEBUG (%s, %d):  atom->position[0] - lowerCorner[0] = %g\n",
00728                   __FILE__, __LINE__, atom->position[0] - lowerCorner[0]);
00729         Vnm_print(1, "DEBUG (%s, %d):  atom->position[1] - upperCorner[1] = %g\n",
00730                   __FILE__, __LINE__, atom->position[1] - upperCorner[1]);
00731         Vnm_print(1, "DEBUG (%s, %d):  atom->position[1] - lowerCorner[1] = %g\n",
00732                   __FILE__, __LINE__, atom->position[1] - lowerCorner[1]);
00733         Vnm_print(1, "DEBUG (%s, %d):  atom->position[2] - upperCorner[2] = %g\n",
00734                   __FILE__, __LINE__, atom->position[2] - upperCorner[2]);
00735         Vnm_print(1, "DEBUG (%s, %d):  atom->position[2] - lowerCorner[0] = %g\n",
00736                   __FILE__, __LINE__, atom->position[2] - lowerCorner[2]);
00737         Vnm_print(1, "DEBUG (%s, %d):  xok = %g, yok = %g, zok = %g\n",
00738                   __FILE__, __LINE__, xok, yok, zok);
00739          */
00740
```

```
00741     }
00742
00743     /* Load up pvec -
00744        For all points within h{axis}/2 of a border - use a gradient
00745        to determine the pvec weight.
00746        Points on the boundary depend on the presence of an adjacent
00747        processor. */
00748
00749     for (i=0; i<(nx*ny*nz); i++) thee->pvec[i] = 0.0;
00750
00751     for (i=0; i<nx; i++) {
00752         xok = 0.0;
00753         x = i*hx + xmin;
00754         if ( (x < (upperCorner[0]-hx/2)) &&
00755              (x > (lowerCorner[0]+hx/2))
00756            ) xok = 1.0;
00757         else if ( (VABS(x - lowerCorner[0]) < VPMGSMALL) &&
00758                   (bflags[VAPBS_LEFT] == 0)) xok = 1.0;
00759         else if ((VABS(x - lowerCorner[0]) < VPMGSMALL) &&
00760                  (bflags[VAPBS_LEFT] == 1)) xok = 0.5;
00761         else if ((VABS(x - upperCorner[0]) < VPMGSMALL) &&
00762                  (bflags[VAPBS_RIGHT] == 0)) xok = 1.0;
00763         else if ((VABS(x - upperCorner[0]) < VPMGSMALL) &&
00764                  (bflags[VAPBS_RIGHT] == 1)) xok = 0.5;
00765         else if ((x > (upperCorner[0] + hx/2)) || (x < (lowerCorner[0] - hx/2))) xok = 0.0;
00766         else if ((x < (upperCorner[0] + hx/2)) || (x > (lowerCorner[0] - hx/2))) {
00767             x0 = VMAX2(x - hx/2, lowerCorner[0]);
00768             x1 = VMIN2(x + hx/2, upperCorner[0]);
00769             xok = VABS(x1-x0)/hx;
00770
00771             if (xok < 0.0) {
00772                 if (VABS(xok) < VPMGSMALL) xok = 0.0;
00773                 else {
00774                     Vnm_print(2, "Vpmg_setPart:  fell off x-interval (%1.12E)!\n",
00775                               xok);
00776                     VASSERT(0);
00777                 }
00778             }
00779             if (xok > 1.0) {
00780                 if (VABS(xok - 1.0) < VPMGSMALL) xok = 1.0;
00781                 else {
00782                     Vnm_print(2, "Vpmg_setPart:  fell off x-interval (%1.12E)!\n",
00783                               xok);
00784                     VASSERT(0);
00785                 }
00786             }
00787
00788         } else xok = 0.0;
00789
00790         for (j=0; j<ny; j++) {
00791             yok = 0.0;
00792             y = j*hy + ymin;
00793             if ((y < (upperCorner[1]-hy/2)) && (y > (lowerCorner[1]+hy/2))) yok = 1.0;
00794             else if ((VABS(y - lowerCorner[1]) < VPMGSMALL) &&
00795                      (bflags[VAPBS_BACK] == 0)) yok = 1.0;
00796             else if ((VABS(y - lowerCorner[1]) < VPMGSMALL) &&
00797                      (bflags[VAPBS_BACK] == 1)) yok = 0.5;
00798             else if ((VABS(y - upperCorner[1]) < VPMGSMALL) &&
00799                      (bflags[VAPBS_FRONT] == 0)) yok = 1.0;
00800             else if ((VABS(y - upperCorner[1]) < VPMGSMALL) &&
00801                      (bflags[VAPBS_FRONT] == 1)) yok = 0.5;
00802             else if ((y > (upperCorner[1] + hy/2)) || (y < (lowerCorner[1] - hy/2))) yok=0.0;
00803             else if ((y < (upperCorner[1] + hy/2)) || (y > (lowerCorner[1] - hy/2))){
00804                 y0 = VMAX2(y - hy/2, lowerCorner[1]);
00805                 y1 = VMIN2(y + hy/2, upperCorner[1]);
00806                 yok = VABS(y1-y0)/hy;
00807
00808                 if (yok < 0.0) {
00809                     if (VABS(yok) < VPMGSMALL) yok = 0.0;
00810                     else {
00811                         Vnm_print(2, "Vpmg_setPart:  fell off y-interval (%1.12E)!\n",
00812                                   yok);
00813                         VASSERT(0);
00814                     }
00815                 }
00816                 if (yok > 1.0) {
00817                     if (VABS(yok - 1.0) < VPMGSMALL) yok = 1.0;
00818                     else {
00819                         Vnm_print(2, "Vpmg_setPart:  fell off y-interval (%1.12E)!\n",
00820                                   yok);
00821                         VASSERT(0);
```

```
00822                     }
00823                 }
00824             }
00825             else yok=0.0;
00826
00827             for (k=0; k<nz; k++) {
00828                 zok = 0.0;
00829                 z = k*hzed + zmin;
00830                 if ((z < (upperCorner[2]-hzed/2)) && (z > (lowerCorner[2]+hzed/2))) zok = 1.0;
00831                 else if ((VABS(z - lowerCorner[2]) < VPMGSMALL) &&
00832                          (bflags[VAPBS_DOWN] == 0)) zok = 1.0;
00833                 else if ((VABS(z - lowerCorner[2]) < VPMGSMALL) &&
00834                          (bflags[VAPBS_DOWN] == 1)) zok = 0.5;
00835                 else if ((VABS(z - upperCorner[2]) < VPMGSMALL) &&
00836                          (bflags[VAPBS_UP] == 0)) zok = 1.0;
00837                 else if ((VABS(z - upperCorner[2]) < VPMGSMALL) &&
00838                          (bflags[VAPBS_UP] == 1)) zok = 0.5;
00839                 else if ((z > (upperCorner[2] + hzed/2)) || (z < (lowerCorner[2] - hzed/2))) zok=0.0;
00840                 else if ((z < (upperCorner[2] + hzed/2)) || (z > (lowerCorner[2] - hzed/2))){
00841                     z0 = VMAX2(z - hzed/2, lowerCorner[2]);
00842                     z1 = VMIN2(z + hzed/2, upperCorner[2]);
00843                     zok = VABS(z1-z0)/hzed;
00844
00845                     if (zok < 0.0) {
00846                         if (VABS(zok) < VPMGSMALL) zok = 0.0;
00847                         else {
00848                             Vnm_print(2, "Vpmg_setPart:  fell off z-interval (%1.12E)!\n",
00849                                       zok);
00850                             VASSERT(0);
00851                         }
00852                     }
00853                     if (zok > 1.0) {
00854                         if (VABS(zok - 1.0) < VPMGSMALL) zok = 1.0;
00855                         else {
00856                             Vnm_print(2, "Vpmg_setPart:  fell off z-interval (%1.12E)!\n",
00857                                       zok);
00858                             VASSERT(0);
00859                         }
00860                     }
00861                 }
00862                 else zok = 0.0;
00863
00864                 if (VABS(xok*yok*zok) < VPMGSMALL) thee->pvec[IJK(i,j,k)] = 0.0;
00865                 else thee->pvec[IJK(i,j,k)] = xok*yok*zok;
00866
00867             }
00868         }
00869     }
00870 }
00871
00872 VPUBLIC void Vpmg_unsetPart(Vpmg *thee) {
00873
00874     int i, nx, ny, nz;
00875     Vatom *atom;
00876     Valist *alist;
00877
00878     VASSERT(thee != VNULL);
00879
00880     nx = thee->pmgp->nx;
00881     ny = thee->pmgp->ny;
00882     nz = thee->pmgp->nz;
00883     alist = thee->pbe->alist;
00884
00885     for (i=0; i<(nx*ny*nz); i++) thee->pvec[i] = 1;
00886     for (i=0; i<Valist_getNumberAtoms(alist); i++) {
00887         atom = Valist_getAtom(alist, i);
00888         atom->partID = 1;
00889     }
00890 }
00891
00892 VPUBLIC int Vpmg_fillArray(Vpmg *thee, double *vec, Vdata_Type type,
00893   double parm, Vhal_PBEType pbetype, PBEparm *pbeparm) {
00894
00895     Vacc *acc = VNULL;
00896     Vpbe *pbe = VNULL;
00897     Vgrid *grid = VNULL;
00898     Vatom *atoms = VNULL;
00899     Valist *alist = VNULL;
00900     double position[3], hx, hy, hzed, xmin, ymin, zmin;
00901     double grad[3], eps, epsp, epss, zmagic, u;
00902     int i, j, k, l, nx, ny, nz, ichop;
```

```
00903
00904     pbe = thee->pbe;
00905     acc = Vpbe_getVacc(pbe);
00906     nx = thee->pmgp->nx;
00907     ny = thee->pmgp->ny;
00908     nz = thee->pmgp->nz;
00909     hx = thee->pmgp->hx;
00910     hy = thee->pmgp->hy;
00911     hzed = thee->pmgp->hzed;
00912     xmin = thee->pmgp->xmin;
00913     ymin = thee->pmgp->ymin;
00914     zmin = thee->pmgp->zmin;
00915     epsp = Vpbe_getSoluteDiel(pbe);
00916     epss = Vpbe_getSolventDiel(pbe);
00917     zmagic = Vpbe_getZmagic(pbe);
00918
00919     if (!(thee->filled)) {
00920         Vnm_print(2, "Vpmg_fillArray:  need to call Vpmg_fillco first!\n");
00921         return 0;
00922     }
00923
00924     switch (type) {
00925
00926         case VDT_CHARGE:
00927
00928             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->charge[i]/zmagic;
00929             break;
00930
00931         case VDT_DIELX:
00932
00933             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->epsx[i];
00934             break;
00935
00936         case VDT_DIELY:
00937
00938             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->epsy[i];
00939             break;
00940
00941         case VDT_DIELZ:
00942
00943             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->epsz[i];
00944             break;
00945
00946         case VDT_KAPPA:
00947
00948             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->kappa[i];
00949             break;
00950
00951         case VDT_POT:
00952
00953             for (i=0; i<nx*ny*nz; i++) vec[i] = thee->u[i];
00954             break;
00955
00956         case VDT_ATOMPOT:
00957             alist = thee->pbe->alist;
00958             atoms = alist[pbeparm->molid-1].atoms;
00959             grid = Vgrid_ctor(nx, ny, nz, hx, hy,
00960                               hzed, xmin, ymin, zmin,thee->u);
00961             for (i=0; i<alist[pbeparm->molid-1].number;i++) {
00962                 position[0] = atoms[i].position[0];
00963                 position[1] = atoms[i].position[1];
00964                 position[2] = atoms[i].position[2];
00965
00966                 Vgrid_value(grid, position, &vec[i]);
00967             }
00968             Vgrid_dtor(&grid);
00969             break;
00970
00971         case VDT_SMOL:
00972
00973             for (k=0; k<nz; k++) {
00974                 for (j=0; j<ny; j++) {
00975                     for (i=0; i<nx; i++) {
00976
00977                         position[0] = i*hx + xmin;
00978                         position[1] = j*hy + ymin;
00979                         position[2] = k*hzed + zmin;
00980
00981                         vec[IJK(i,j,k)] = (Vacc_molAcc(acc,position,parm));
00982                     }
00983                 }
```

```
00984             }
00985             break;
00986
00987         case VDT_SSPL:
00988
00989             for (k=0; k<nz; k++) {
00990                 for (j=0; j<ny; j++) {
00991                     for (i=0; i<nx; i++) {
00992
00993                         position[0] = i*hx + xmin;
00994                         position[1] = j*hy + ymin;
00995                         position[2] = k*hzed + zmin;
00996
00997                         vec[IJK(i,j,k)] = Vacc_splineAcc(acc,position,parm,0);
00998                     }
00999                 }
01000             }
01001             break;
01002
01003         case VDT_VDW:
01004
01005             for (k=0; k<nz; k++) {
01006                 for (j=0; j<ny; j++) {
01007                     for (i=0; i<nx; i++) {
01008
01009                         position[0] = i*hx + xmin;
01010                         position[1] = j*hy + ymin;
01011                         position[2] = k*hzed + zmin;
01012
01013                         vec[IJK(i,j,k)] = Vacc_vdwAcc(acc,position);
01014                     }
01015                 }
01016             }
01017             break;
01018
01019         case VDT_IVDW:
01020
01021             for (k=0; k<nz; k++) {
01022                 for (j=0; j<ny; j++) {
01023                     for (i=0; i<nx; i++) {
01024
01025                         position[0] = i*hx + xmin;
01026                         position[1] = j*hy + ymin;
01027                         position[2] = k*hzed + zmin;
01028
01029                         vec[IJK(i,j,k)] = Vacc_ivdwAcc(acc,position,parm);
01030                     }
01031                 }
01032             }
01033             break;
01034
01035         case VDT_LAP:
01036
01037             grid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin,
01038               thee->u);
01039             for (k=0; k<nz; k++) {
01040                 for (j=0; j<ny; j++) {
01041                     for (i=0; i<nx; i++) {
01042
01043                         if ((k==0) || (k==(nz-1)) ||
01044                             (j==0) || (j==(ny-1)) ||
01045                             (i==0) || (i==(nx-1))) {
01046
01047                             vec[IJK(i,j,k)] = 0;
01048
01049                         } else {
01050                                 position[0] = i*hx + xmin;
01051                                 position[1] = j*hy + ymin;
01052                                 position[2] = k*hzed + zmin;
01053                                 VASSERT(Vgrid_curvature(grid,position, 1,
01054                                   &(vec[IJK(i,j,k)])));
01055                         }
01056                     }
01057                 }
01058             }
01059             Vgrid_dtor(&grid);
01060             break;
01061
01062         case VDT_EDENS:
01063
01064             grid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin,
```

```
01065               thee->u);
01066           for (k=0; k<nz; k++) {
01067               for (j=0; j<ny; j++) {
01068                   for (i=0; i<nx; i++) {
01069
01070                       position[0] = i*hx + xmin;
01071                       position[1] = j*hy + ymin;
01072                       position[2] = k*hzed + zmin;
01073                       VASSERT(Vgrid_gradient(grid, position, grad));
01074                       eps = epsp + (epss-epsp)*Vacc_molAcc(acc, position,
01075                         pbe->solventRadius);
01076                       vec[IJK(i,j,k)] = 0.0;
01077                       for (l=0; l<3; l++)
01078                         vec[IJK(i,j,k)] += eps*VSQR(grad[l]);
01079                   }
01080               }
01081           }
01082           Vgrid_dtor(&grid);
01083           break;
01084
01085       case VDT_NDENS:
01086
01087           for (k=0; k<nz; k++) {
01088               for (j=0; j<ny; j++) {
01089                   for (i=0; i<nx; i++) {
01090
01091                       position[0] = i*hx + xmin;
01092                       position[1] = j*hy + ymin;
01093                       position[2] = k*hzed + zmin;
01094                       vec[IJK(i,j,k)] = 0.0;
01095                       u = thee->u[IJK(i,j,k)];
01096                       if ( VABS(Vacc_ivdwAcc(acc,
01097                                position, pbe->maxIonRadius) - 1.0) < VSMALL) {
01098                           for (l=0; l<pbe->numIon; l++) {
01099                               double q = pbe->ionQ[l];
01100                               if (pbetype == PBE_NPBE || pbetype == PBE_SMPBE /*  SMPBE Added */) {
01101                                   vec[IJK(i,j,k)] += pbe->ionConc[l]*Vcap_exp(-q*u, &ichop);
01102                               } else if (pbetype == PBE_LPBE){
01103                                   vec[IJK(i,j,k)] += pbe->ionConc[l]*(1 - q*u + 0.5*q*q*u*u);
01104                               }
01105                           }
01106                       }
01107                   }
01108               }
01109           }
01110           break;
01111
01112       case VDT_QDENS:
01113
01114           for (k=0; k<nz; k++) {
01115           for (j=0; j<ny; j++) {
01116           for (i=0; i<nx; i++) {
01117               position[0] = i*hx + xmin;
01118               position[1] = j*hy + ymin;
01119               position[2] = k*hzed + zmin;
01120               vec[IJK(i,j,k)] = 0.0;
01121               u = thee->u[IJK(i,j,k)];
01122               if ( VABS(Vacc_ivdwAcc(acc,
01123                                position, pbe->maxIonRadius) - 1.0) < VSMALL) {
01124                   for (l=0; l<pbe->numIon; l++) {
01125                       double q = pbe->ionQ[l];
01126                       if (pbetype == PBE_NPBE || pbetype == PBE_SMPBE /*  SMPBE Added */) {
01127                           vec[IJK(i,j,k)] += pbe->ionConc[l]*q*Vcap_exp(-q*u, &ichop);
01128                       } else if (pbetype == PBE_LPBE) {
01129                           vec[IJK(i,j,k)] += pbe->ionConc[l]*q*(1 - q*u + 0.5*q*q*u*u);
01130                       }
01131                   }
01132               }
01133           }}}
01134           break;
01135
01136       default:
01137
01138           Vnm_print(2, "main:  Bogus data type (%d)!\n", type);
01139           return 0;
01140           break;
01141
01142   }
01143
01144   return 1;
01145
```

```
01146 }
01147
01148 VPRIVATE double Vpmg_polarizEnergy(Vpmg *thee,
01149                                    int extFlag
01150                                   ) {
01151
01152     int i,
01153         j,
01154         k,
01155         ijk,
01156         nx,
01157         ny,
01158         nz,
01159         iatom;
01160     double xmin,
01161            ymin,
01162            zmin,
01163            //x, // gcc: not used
01164            //y,
01165            //z,
01166            hx,
01167            hy,
01168            hzed,
01169            epsp,
01170            lap,
01171            pt[3],
01172            T,
01173            pre,
01174            polq,
01175            dist2,
01176            dist,
01177            energy,
01178            q,
01179            *charge,
01180            *pos,
01181            eps_w;
01182     Vgrid *potgrid;
01183     Vpbe *pbe;
01184     Valist *alist;
01185     Vatom *atom;
01186
01187     xmin = thee->pmgp->xmin;
01188     ymin = thee->pmgp->ymin;
01189     zmin = thee->pmgp->ymin;
01190     hx = thee->pmgp->hx;
01191     hy = thee->pmgp->hy;
01192     hzed = thee->pmgp->hzed;
01193     nx = thee->pmgp->nx;
01194     ny = thee->pmgp->ny;
01195     nz = thee->pmgp->nz;
01196     pbe = thee->pbe;
01197     epsp = Vpbe_getSoluteDiel(pbe);
01198     eps_w = Vpbe_getSolventDiel(pbe);
01199     alist = pbe->alist;
01200     charge = thee->charge;
01201
01202     /* Calculate the prefactor for Coulombic calculations */
01203     T = Vpbe_getTemperature(pbe);
01204     pre = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T);
01205     pre = pre*(1.0e10);
01206
01207     /* Set up Vgrid object with solution */
01208     potgrid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin, thee->u);
01209
01210     /* Calculate polarization charge */
01211     energy = 0.0;
01212     for (i=1; i<(nx-1); i++) {
01213         pt[0] = xmin + hx*i;
01214         for (j=1; j<(ny-1); j++) {
01215             pt[1] = ymin + hy*j;
01216             for (k=1; k<(nz-1); k++) {
01217                 pt[2] = zmin + hzed*k;
01218
01219                 /* Calculate polarization charge */
01220                 VASSERT(Vgrid_curvature(potgrid, pt, 1, &lap));
01221                 ijk = IJK(i,j,k);
01222                 polq = charge[ijk] + epsp*lap*3.0;
01223
01224                 /* Calculate interaction energy with atoms */
01225                 if (VABS(polq) > VSMALL) {
01226                     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
```

```
01227                         atom = Valist_getAtom(alist, iatom);
01228                         q = Vatom_getCharge(atom);
01229                         pos = Vatom_getPosition(atom);
01230                         dist2 = VSQR(pos[0]-pt[0]) + VSQR(pos[1]-pt[1]) \
01231                                 + VSQR(pos[2]-pt[2]);
01232                         dist = VSQRT(dist2);
01233 
01234                         if (dist < VSMALL) {
01235                             Vnm_print(2, "Vpmg_polarizEnergy:  atom on grid point; ignoring!\n");
01236                         } else {
01237                             energy = energy + polq*q/dist;
01238                         }
01239                     }
01240                 }
01241             }
01242         }
01243     }
01244 
01245     return pre*energy;
01246 }
01247 
01248 VPUBLIC double Vpmg_energy(Vpmg *thee,
01249                            int extFlag
01250                           ) {
01251 
01252     double totEnergy = 0.0,
01253            dielEnergy = 0.0,
01254            qmEnergy = 0.0,
01255            qfEnergy = 0.0;
01256 
01257     VASSERT(thee != VNULL);
01258 
01259     if ((thee->pmgp->nonlin) && (Vpbe_getBulkIonicStrength(thee->pbe) > 0.)) {
01260         Vnm_print(0, "Vpmg_energy:  calculating full PBE energy\n");
01261         qmEnergy = Vpmg_qmEnergy(thee, extFlag);
01262         Vnm_print(0, "Vpmg_energy:  qmEnergy = %1.12E kT\n", qmEnergy);
01263         qfEnergy = Vpmg_qfEnergy(thee, extFlag);
01264         Vnm_print(0, "Vpmg_energy:  qfEnergy = %1.12E kT\n", qfEnergy);
01265         dielEnergy = Vpmg_dielEnergy(thee, extFlag);
01266         Vnm_print(0, "Vpmg_energy:  dielEnergy = %1.12E kT\n", dielEnergy);
01267         totEnergy = qfEnergy - dielEnergy - qmEnergy;
01268     } else {
01269         Vnm_print(0, "Vpmg_energy:  calculating only q-phi energy\n");
01270         qfEnergy = Vpmg_qfEnergy(thee, extFlag);
01271         Vnm_print(0, "Vpmg_energy:  qfEnergy = %1.12E kT\n", qfEnergy);
01272         totEnergy = 0.5*qfEnergy;
01273     }
01274 
01275     return totEnergy;
01276 
01277 }
01278 
01279 VPUBLIC double Vpmg_dielEnergy(Vpmg *thee,
01280                                int extFlag
01281                               ) {
01282 
01283     double hx,
01284            hy,
01285            hzed,
01286            energy,
01287            nrgx,
01288            nrgy,
01289            nrgz,
01290            pvecx,
01291            pvecy,
01292            pvecz;
01293     int i,
01294         j,
01295         k,
01296         nx,
01297         ny,
01298         nz;
01299 
01300     VASSERT(thee != VNULL);
01301 
01302     /* Get the mesh information */
01303     nx = thee->pmgp->nx;
01304     ny = thee->pmgp->ny;
01305     nz = thee->pmgp->nz;
01306     hx = thee->pmgp->hx;
01307     hy = thee->pmgp->hy;
```

```
01308     hzed = thee->pmgp->hzed;
01309
01310     energy = 0.0;
01311
01312     if (!thee->filled) {
01313         Vnm_print(2, "Vpmg_dielEnergy:  Need to call Vpmg_fillco!\n");
01314         VASSERT(0);
01315     }
01316
01317     for (k=0; k<(nz-1); k++) {
01318         for (j=0; j<(ny-1); j++) {
01319             for (i=0; i<(nx-1); i++) {
01320                 pvecx = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i+1,j,k)]);
01321                 pvecy = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j+1,k)]);
01322                 pvecz = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j,k+1)]);
01323                 nrgx = thee->epsx[IJK(i,j,k)]*pvecx
01324                   * VSQR((thee->u[IJK(i,j,k)]-thee->u[IJK(i+1,j,k)])/hx);
01325                 nrgy = thee->epsy[IJK(i,j,k)]*pvecy
01326                   * VSQR((thee->u[IJK(i,j,k)]-thee->u[IJK(i,j+1,k)])/hy);
01327                 nrgz = thee->epsz[IJK(i,j,k)]*pvecz
01328                   * VSQR((thee->u[IJK(i,j,k)]-thee->u[IJK(i,j,k+1)])/hzed);
01329                 energy += (nrgx + nrgy + nrgz);
01330             }
01331         }
01332     }
01333
01334     energy = 0.5*energy*hx*hy*hzed;
01335     energy = energy/Vpbe_getZmagic(thee->pbe);
01336
01337     if (extFlag == 1) energy += (thee->extDiEnergy);
01338
01339     return energy;
01340 }
01341
01342 VPUBLIC double Vpmg_dielGradNorm(Vpmg *thee) {
01343
01344     double hx, hy, hzed, energy, nrgx, nrgy, nrgz, pvecx, pvecy, pvecz;
01345     int i, j, k, nx, ny, nz;
01346
01347     VASSERT(thee != VNULL);
01348
01349     /* Get the mesh information */
01350     nx = thee->pmgp->nx;
01351     ny = thee->pmgp->ny;
01352     nz = thee->pmgp->nz;
01353     hx = thee->pmgp->hx;
01354     hy = thee->pmgp->hy;
01355     hzed = thee->pmgp->hzed;
01356
01357     energy = 0.0;
01358
01359     if (!thee->filled) {
01360         Vnm_print(2, "Vpmg_dielGradNorm:  Need to call Vpmg_fillco!\n");
01361         VASSERT(0);
01362     }
01363
01364     for (k=1; k<nz; k++) {
01365         for (j=1; j<ny; j++) {
01366             for (i=1; i<nx; i++) {
01367                 pvecx = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i-1,j,k)]);
01368                 pvecy = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j-1,k)]);
01369                 pvecz = 0.5*(thee->pvec[IJK(i,j,k)]+thee->pvec[IJK(i,j,k-1)]);
01370                 nrgx = pvecx
01371                  * VSQR((thee->epsx[IJK(i,j,k)]-thee->epsx[IJK(i-1,j,k)])/hx);
01372                 nrgy = pvecy
01373                  * VSQR((thee->epsy[IJK(i,j,k)]-thee->epsy[IJK(i,j-1,k)])/hy);
01374                 nrgz = pvecz
01375                  * VSQR((thee->epsz[IJK(i,j,k)]-thee->epsz[IJK(i,j,k-1)])/hzed);
01376                 energy += VSQRT(nrgx + nrgy + nrgz);
01377             }
01378         }
01379     }
01380
01381     energy = energy*hx*hy*hzed;
01382
01383     return energy;
01384 }
01385
01386 VPUBLIC double Vpmg_qmEnergy(Vpmg *thee,
01387                              int extFlag
01388                             ) {
```

```
01389
01390     double energy;
01391
01392     if(thee->pbe->ipkey == IPKEY_SMPBE){
01393         energy = Vpmg_qmEnergySMPBE(thee,extFlag);
01394     }else{
01395         energy = Vpmg_qmEnergyNONLIN(thee,extFlag);
01396     }
01397
01398     return energy;
01399 }
01400
01401 VPRIVATE double Vpmg_qmEnergyNONLIN(Vpmg *thee,
01402                                     int extFlag
01403                                    ) {
01404
01405     double hx,
01406            hy,
01407            hzed,
01408            energy,
01409            ionConc[MAXION],
01410            ionRadii[MAXION],
01411            ionQ[MAXION],
01412            zkappa2,
01413            ionstr,
01414            zks2;
01415     int i, /* Loop variable */
01416         j,
01417         nx,
01418         ny,
01419         nz,
01420         nion,
01421         ichop,
01422         nchop,
01423         len; /* Stores number of iterations for loops to avoid multiple recalculations */
01424
01425     VASSERT(thee != VNULL);
01426
01427     /* Get the mesh information */
01428     nx = thee->pmgp->nx;
01429     ny = thee->pmgp->ny;
01430     nz = thee->pmgp->nz;
01431     hx = thee->pmgp->hx;
01432     hy = thee->pmgp->hy;
01433     hzed = thee->pmgp->hzed;
01434     zkappa2 = Vpbe_getZkappa2(thee->pbe);
01435     ionstr = Vpbe_getBulkIonicStrength(thee->pbe);
01436
01437     /* Bail if we're at zero ionic strength */
01438     if (zkappa2 < VSMALL) {
01439
01440 #ifndef VAPBSQUIET
01441         Vnm_print(0, "Vpmg_qmEnergy:  Zero energy for zero ionic strength!\n");
01442 #endif
01443
01444         return 0.0;
01445     }
01446     zks2 = 0.5*zkappa2/ionstr;
01447
01448     if (!thee->filled) {
01449         Vnm_print(2, "Vpmg_qmEnergy:  Need to call Vpmg_fillco()!\n");
01450         VASSERT(0);
01451     }
01452
01453     energy = 0.0;
01454     nchop = 0;
01455     Vpbe_getIons(thee->pbe, &nion, ionConc, ionRadii, ionQ);
01456     if (thee->pmgp->nonlin) {
01457         Vnm_print(0, "Vpmg_qmEnergy:  Calculating nonlinear energy\n");
01458         for (i=0, len=nx*ny*nz; i<len; i++) {
01459             if (thee->pvec[i]*thee->kappa[i] > VSMALL) {
01460                 for (j=0; j<nion; j++) {
01461                     energy += (thee->pvec[i]*thee->kappa[i]*zks2
01462                       * ionConc[j]
01463                       * (Vcap_exp(-ionQ[j]*thee->u[i], &ichop)-1.0));
01464                     nchop += ichop;
01465                 }
01466             }
01467         }
01468         if (nchop > 0){
01469             Vnm_print(2, "Vpmg_qmEnergy:  Chopped EXP %d times!\n",nchop);
```

```
01470             Vnm_print(2, "\nERROR!  Detected large potential values in energy evaluation! \nERROR!  This
      calculation failed -- please report to the APBS developers!\n\n");
01471             VASSERT(0);
01472         }
01473     } else {
01474         /* Zkappa2 OK here b/c LPBE approx */
01475         Vnm_print(0, "Vpmg_qmEnergy:  Calculating linear energy\n");
01476         for (i=0, len=nx*ny*nz; i<len; i++) {
01477             if (thee->pvec[i]*thee->kappa[i] > VSMALL)
01478               energy += (thee->pvec[i]*zkappa2*thee->kappa[i]*VSQR(thee->u[i]));
01479         }
01480         energy = 0.5*energy;
01481     }
01482     energy = energy*hx*hy*hzed;
01483     energy = energy/Vpbe_getZmagic(thee->pbe);
01484 
01485     if (extFlag == 1) energy += thee->extQmEnergy;
01486 
01487     return energy;
01488 }
01489 
01490 VPUBLIC double Vpmg_qmEnergySMPBE(Vpmg *thee,
01491                                   int extFlag
01492                                  ) {
01493 
01494     double hx,
01495            hy,
01496            hzed,
01497            energy,
01498            ionConc[MAXION],
01499            ionRadii[MAXION],
01500            ionQ[MAXION],
01501            zkappa2,
01502            ionstr,
01503            zks2;
01504     int i,
01505         //j, // gcc: not used
01506         nx,
01507         ny,
01508         nz,
01509         nion,
01510         //ichop, // gcc: not used
01511         nchop,
01512         len; /* Loop variable */
01513 
01514     /* SMPB Modification (vchu, 09/21/06)*/
01515     /* variable declarations for SMPB energy terms */
01516     double a,
01517            k,
01518            z1,
01519            z2,
01520            z3,
01521            cb1,
01522            cb2,
01523            cb3,
01524            a1,
01525            a2,
01526            a3,
01527            c1,
01528            c2,
01529            c3,
01530            currEnergy,
01531            fracOccA,
01532            fracOccB,
01533            fracOccC,
01534            phi,
01535            gpark,
01536            denom;
01537            // Na; /**< @todo remove if no conflicts are caused - This constant is already defined in
      vpde.h.  no need to redefine. */
01538     int ichop1,
01539         ichop2,
01540         ichop3;
01541 
01542     VASSERT(thee != VNULL);
01543 
01544     /* Get the mesh information */
01545     nx = thee->pmgp->nx;
01546     ny = thee->pmgp->ny;
01547     nz = thee->pmgp->nz;
01548     hx = thee->pmgp->hx;
```

```
01549     hy = thee->pmgp->hy;
01550     hzed = thee->pmgp->hzed;
01551     zkappa2 = Vpbe_getZkappa2(thee->pbe);
01552     ionstr = Vpbe_getBulkIonicStrength(thee->pbe);
01553 
01554     /* Bail if we're at zero ionic strength */
01555     if (zkappa2 < VSMALL) {
01556 
01557 #ifndef VAPBSQUIET
01558         Vnm_print(0, "Vpmg_qmEnergySMPBE:  Zero energy for zero ionic strength!\n");
01559 #endif
01560 
01561         return 0.0;
01562     }
01563     zks2 = 0.5*zkappa2/ionstr;
01564 
01565     if (!thee->filled) {
01566         Vnm_print(2, "Vpmg_qmEnergySMPBE:  Need to call Vpmg_fillco()!\n");
01567         VASSERT(0);
01568     }
01569 
01570     energy = 0.0;
01571     nchop = 0;
01572     Vpbe_getIons(thee->pbe, &nion, ionConc, ionRadii, ionQ);
01573 
01574     /* SMPB Modification (vchu, 09/21/06) */
01575     /* Extensive modification to the first part of the if statement
01576         where that handles the thee->pmgp->nonlin part. Basically, I've
01577         deleted all of the original code and written my own code that computes
01578         the electrostatic free energy in the SMPB framework. Definitely really hacky
01579         at this stage of the game, but gets the job done. The second part of the
01580         if statement (the part that handles linear poisson-boltzmann) has been deleted
01581         because there will be no linearized SMPB energy.. */
01582 
01583     z1 = ionQ[0];
01584     z2 = ionQ[1];
01585     z3 = ionQ[2];
01586     cb1 = ionConc[0];
01587     cb2 = ionConc[1];
01588     cb3 = ionConc[2];
01589     a  = thee->pbe->smvolume;
01590     k  = thee->pbe->smsize;
01591 
01593     // This constant is defined in vpde.h  Do not need to redefine
01594     //Na = 6.022045000e-04; /* Converts from Molar to N/A^3 */
01595 
01596     fracOccA = Na*cb1*VCUB(a);
01597     fracOccB = Na*cb2*VCUB(a);
01598     fracOccC = Na*cb3*VCUB(a);
01599 
01600     phi = (fracOccA/k) + fracOccB + fracOccC;
01601 
01602     if (thee->pmgp->nonlin) {
01603         Vnm_print(0, "Vpmg_qmEnergySMPBE:  Calculating nonlinear energy using SMPB functional!\n");
01604         for (i=0, len=nx*ny*nz; i<len; i++) {
01605             if (((k-1) > VSMALL) && (thee->pvec[i]*thee->kappa[i] > VSMALL)) {
01606 
01607                 a1 = Vcap_exp(-1.0*z1*thee->u[i], &ichop1);
01608                 a2 = Vcap_exp(-1.0*z2*thee->u[i], &ichop2);
01609                 a3 = Vcap_exp(-1.0*z3*thee->u[i], &ichop3);
01610 
01611                 nchop += ichop1 + ichop2 + ichop3;
01612 
01613                 gpark = (1 - phi + (fracOccA/k)*a1);
01614                 denom = VPOW(gpark, k) + VPOW(1-fracOccB-fracOccC, k-1)*(fracOccB*a2+fracOccC*a3);
01615 
01616                 if (cb1 > VSMALL) {
01617                     c1 = Na*cb1*VPOW(gpark, k-1)*a1/denom;
01618                     if(c1 != c1) c1 = 0.;
01619                 } else c1 = 0.;
01620 
01621                 if (cb2 > VSMALL) {
01622                     c2 = Na*cb2*VPOW(1-fracOccB-fracOccC,k-1)*a2/denom;
01623                     if(c2 != c2) c2 = 0.;
01624                 } else c2 = 0.;
01625 
01626                 if (cb3 > VSMALL) {
01627                     c3 = Na*cb3*VPOW(1-fracOccB-fracOccC,k-1)*a3/denom;
01628                     if(c3 != c3) c3 = 0.;
01629                 } else c3 = 0.;
01630 
```

```
01631                 currEnergy = k*VLOG((1-(c1*VCUB(a)/k)-c2*VCUB(a)-c3*VCUB(a))/(1-phi))
01632                     -(k-1)*VLOG((1-c2*VCUB(a)-c3*VCUB(a))/(1-phi+(fracOccA/k)));
01633
01634                 energy += thee->pvec[i]*thee->kappa[i]*currEnergy;
01635
01636             } else if (thee->pvec[i]*thee->kappa[i] > VSMALL){
01637
01638                 a1 = Vcap_exp(-1.0*z1*thee->u[i], &ichop1);
01639                 a2 = Vcap_exp(-1.0*z2*thee->u[i], &ichop2);
01640                 a3 = Vcap_exp(-1.0*z3*thee->u[i], &ichop3);
01641
01642                 nchop += ichop1 + ichop2 + ichop3;
01643
01644                 gpark = (1 - phi + (fracOccA)*a1);
01645                 denom = gpark + (fracOccB*a2+fracOccC*a3);
01646
01647                 if (cb1 > VSMALL) {
01648                     c1 = Na*cb1*a1/denom;
01649                     if(c1 != c1) c1 = 0.;
01650                 } else c1 = 0.;
01651
01652                 if (cb2 > VSMALL) {
01653                     c2 = Na*cb2*a2/denom;
01654                     if(c2 != c2) c2 = 0.;
01655                 } else c2 = 0.;
01656
01657                 if (cb3 > VSMALL) {
01658                     c3 = Na*cb3*a3/denom;
01659                     if(c3 != c3) c3 = 0.;
01660                 } else c3 = 0.;
01661
01662                 currEnergy = VLOG((1-c1*VCUB(a)-c2*VCUB(a)-c3*VCUB(a))/(1-fracOccA-fracOccB-fracOccC));
01663
01664                 energy += thee->pvec[i]*thee->kappa[i]*currEnergy;
01665             }
01666         }
01667
01668         energy = -energy/VCUB(a);
01669
01670         if (nchop > 0) Vnm_print(2, "Vpmg_qmEnergySMPBE:  Chopped EXP %d times!\n",
01671                                  nchop);
01672
01673     } else {
01674         /* Zkappa2 OK here b/c LPBE approx */
01675         Vnm_print(0, "Vpmg_qmEnergySMPBE:  ERROR: NO LINEAR ENERGY!! Returning 0!\n");
01676
01677         energy = 0.0;
01678
01679     }
01680     energy = energy*hx*hy*hzed;
01681
01682     if (extFlag == 1) energy += thee->extQmEnergy;
01683
01684     return energy;
01685 }
01686
01687 VPUBLIC double Vpmg_qfEnergy(Vpmg *thee,
01688                              int extFlag
01689                             ) {
01690
01691     double energy = 0.0;
01692
01693     VASSERT(thee != VNULL);
01694
01695     if ((thee->useChargeMap) || (thee->chargeMeth == VCM_BSPL2)) {
01696         energy = Vpmg_qfEnergyVolume(thee, extFlag);
01697     } else {
01698         energy = Vpmg_qfEnergyPoint(thee, extFlag);
01699     }
01700
01701     return energy;
01702 }
01703
01704 VPRIVATE double Vpmg_qfEnergyPoint(Vpmg *thee,
01705                                    int extFlag
01706                                   ) {
01707
01708     int iatom, nx, ny, nz, ihi, ilo, jhi, jlo, khi, klo;
01709     double xmax, ymax, zmax, xmin, ymin, zmin, hx, hy, hzed, ifloat, jfloat;
01710     double charge, kfloat, dx, dy, dz, energy, uval, *position;
01711     double *u;
```

```
01712     double *pvec;
01713     Valist *alist;
01714     Vatom *atom;
01715     Vpbe *pbe;
01716
01717     pbe = thee->pbe;
01718     alist = pbe->alist;
01719     VASSERT(alist != VNULL);
01720
01721     /* Get the mesh information */
01722     nx = thee->pmgp->nx;
01723     ny = thee->pmgp->ny;
01724     nz = thee->pmgp->nz;
01725     hx = thee->pmgp->hx;
01726     hy = thee->pmgp->hy;
01727     hzed = thee->pmgp->hzed;
01728     xmax = thee->pmgp->xmax;
01729     ymax = thee->pmgp->ymax;
01730     zmax = thee->pmgp->zmax;
01731     xmin = thee->pmgp->xmin;
01732     ymin = thee->pmgp->ymin;
01733     zmin = thee->pmgp->zmin;
01734
01735     u = thee->u;
01736     pvec = thee->pvec;
01737
01738     energy = 0.0;
01739
01740     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
01741
01742         /* Get atomic information */
01743         atom = Valist_getAtom(alist, iatom);
01744
01745         position = Vatom_getPosition(atom);
01746         charge = Vatom_getCharge(atom);
01747
01748         /* Figure out which vertices we're next to */
01749         ifloat = (position[0] - xmin)/hx;
01750         jfloat = (position[1] - ymin)/hy;
01751         kfloat = (position[2] - zmin)/hzed;
01752         ihi = (int)ceil(ifloat);
01753         ilo = (int)floor(ifloat);
01754         jhi = (int)ceil(jfloat);
01755         jlo = (int)floor(jfloat);
01756         khi = (int)ceil(kfloat);
01757         klo = (int)floor(kfloat);
01758
01759         if (atom->partID > 0) {
01760
01761             if ((ihi<nx) && (jhi<ny) && (khi<nz) &&
01762                 (ilo>=0) && (jlo>=0) && (klo>=0)) {
01763
01764                 /* Now get trilinear interpolation constants */
01765                 dx = ifloat - (double)(ilo);
01766                 dy = jfloat - (double)(jlo);
01767                 dz = kfloat - (double)(klo);
01768                 uval =
01769                   dx*dy*dz*u[IJK(ihi,jhi,khi)]
01770                 + dx*(1.0-dy)*dz*u[IJK(ihi,jlo,khi)]
01771                 + dx*dy*(1.0-dz)*u[IJK(ihi,jhi,klo)]
01772                 + dx*(1.0-dy)*(1.0-dz)*u[IJK(ihi,jlo,klo)]
01773                 + (1.0-dx)*dy*dz*u[IJK(ilo,jhi,khi)]
01774                 + (1.0-dx)*(1.0-dy)*dz*u[IJK(ilo,jlo,khi)]
01775                 + (1.0-dx)*dy*(1.0-dz)*u[IJK(ilo,jhi,klo)]
01776                 + (1.0-dx)*(1.0-dy)*(1.0-dz)*u[IJK(ilo,jlo,klo)];
01777                 energy += (uval*charge*atom->partID);
01778             } else if (thee->pmgp->bcfl != BCFL_FOCUS) {
01779                 Vnm_print(2, "Vpmg_qfEnergy:  Atom #%d at (%4.3f, %4.3f, \
01780 %4.3f) is off the mesh (ignoring)!\n",
01781                 iatom, position[0], position[1], position[2]);
01782             }
01783         }
01784     }
01785
01786     if (extFlag) energy += thee->extQfEnergy;
01787
01788     return energy;
01789 }
01790
01791 VPUBLIC double Vpmg_qfAtomEnergy(Vpmg *thee, Vatom *atom) {
01792
```

```
01793     int nx, ny, nz, ihi, ilo, jhi, jlo, khi, klo;
01794     double xmax, xmin, ymax, ymin, zmax, zmin, hx, hy, hzed, ifloat, jfloat;
01795     double charge, kfloat, dx, dy, dz, energy, uval, *position;
01796     double *u;
01797
01798
01799     /* Get the mesh information */
01800     nx = thee->pmgp->nx;
01801     ny = thee->pmgp->ny;
01802     nz = thee->pmgp->nz;
01803     hx = thee->pmgp->hx;
01804     hy = thee->pmgp->hy;
01805     hzed = thee->pmgp->hzed;
01806     xmax = thee->xf[nx-1];
01807     ymax = thee->yf[ny-1];
01808     zmax = thee->zf[nz-1];
01809     xmin = thee->xf[0];
01810     ymin = thee->yf[0];
01811     zmin = thee->zf[0];
01812
01813     u = thee->u;
01814
01815     energy = 0.0;
01816
01817
01818     position = Vatom_getPosition(atom);
01819     charge = Vatom_getCharge(atom);
01820
01821     /* Figure out which vertices we're next to */
01822     ifloat = (position[0] - xmin)/hx;
01823     jfloat = (position[1] - ymin)/hy;
01824     kfloat = (position[2] - zmin)/hzed;
01825     ihi = (int)ceil(ifloat);
01826     ilo = (int)floor(ifloat);
01827     jhi = (int)ceil(jfloat);
01828     jlo = (int)floor(jfloat);
01829     khi = (int)ceil(kfloat);
01830     klo = (int)floor(kfloat);
01831
01832     if (atom->partID > 0) {
01833
01834         if ((ihi<nx) && (jhi<ny) && (khi<nz) &&
01835             (ilo>=0) && (jlo>=0) && (klo>=0)) {
01836
01837             /* Now get trilinear interpolation constants */
01838             dx = ifloat - (double)(ilo);
01839             dy = jfloat - (double)(jlo);
01840             dz = kfloat - (double)(klo);
01841             uval =
01842               dx*dy*dz*u[IJK(ihi,jhi,khi)]
01843             + dx*(1.0-dy)*dz*u[IJK(ihi,jlo,khi)]
01844             + dx*dy*(1.0-dz)*u[IJK(ihi,jhi,klo)]
01845             + dx*(1.0-dy)*(1.0-dz)*u[IJK(ihi,jlo,klo)]
01846             + (1.0-dx)*dy*dz*u[IJK(ilo,jhi,khi)]
01847             + (1.0-dx)*(1.0-dy)*dz*u[IJK(ilo,jlo,khi)]
01848             + (1.0-dx)*dy*(1.0-dz)*u[IJK(ilo,jhi,klo)]
01849             + (1.0-dx)*(1.0-dy)*(1.0-dz)*u[IJK(ilo,jlo,klo)];
01850             energy += (uval*charge*atom->partID);
01851         } else if (thee->pmgp->bcfl != BCFL_FOCUS) {
01852             Vnm_print(2, "Vpmg_qfAtomEnergy:  Atom at (%4.3f, %4.3f, \
01853 %4.3f) is off the mesh (ignoring)!\n",
01854             position[0], position[1], position[2]);
01855         }
01856     }
01857
01858     return energy;
01859 }
01860
01861 VPRIVATE double Vpmg_qfEnergyVolume(Vpmg *thee, int extFlag) {
01862
01863     double hx, hy, hzed, energy;
01864     int i, nx, ny, nz;
01865
01866     VASSERT(thee != VNULL);
01867
01868     /* Get the mesh information */
01869     nx = thee->pmgp->nx;
01870     ny = thee->pmgp->ny;
01871     nz = thee->pmgp->nz;
01872     hx = thee->pmgp->hx;
01873     hy = thee->pmgp->hy;
```

```
01874     hzed = thee->pmgp->hzed;
01875
01876     if (!thee->filled) {
01877         Vnm_print(2, "Vpmg_qfEnergyVolume:  need to call Vpmg_fillco!\n");
01878         VASSERT(0);
01879     }
01880
01881     energy = 0.0;
01882     Vnm_print(0, "Vpmg_qfEnergyVolume:  Calculating energy\n");
01883     for (i=0; i<(nx*ny*nz); i++) {
01884         energy += (thee->pvec[i]*thee->u[i]*thee->charge[i]);
01885     }
01886     energy = energy*hx*hy*hzed/Vpbe_getZmagic(thee->pbe);
01887
01888     if (extFlag == 1) energy += thee->extQfEnergy;
01889
01890     return energy;
01891 }
01892
01893 VPRIVATE void Vpmg_splineSelect(int srfm,Vacc *acc,double *gpos,double win,
01894                                 double infrad,Vatom *atom,double *force){
01895
01896     switch (srfm) {
01897         case VSM_SPLINE :
01898             Vacc_splineAccGradAtomNorm(acc, gpos, win, infrad, atom, force);
01899             break;
01900         case VSM_SPLINE3:
01901             Vacc_splineAccGradAtomNorm3(acc, gpos, win, infrad, atom, force);
01902             break;
01903         case VSM_SPLINE4 :
01904             Vacc_splineAccGradAtomNorm4(acc, gpos, win, infrad, atom, force);
01905             break;
01906         default:
01907             Vnm_print(2, "Vpmg_dbnbForce: Unknown surface method.\n");
01908             return;
01909     }
01910
01911     return;
01912 }
01913
01914 VPRIVATE void focusFillBound(Vpmg *thee,
01915                              Vpmg *pmgOLD
01916                             ) {
01917
01918     Vpbe *pbe;
01919     double hxOLD,
01920            hyOLD,
01921            hzOLD,
01922            xminOLD,
01923            yminOLD,
01924            zminOLD,
01925            xmaxOLD,
01926            ymaxOLD,
01927            zmaxOLD,
01928            hxNEW,
01929            hyNEW,
01930            hzNEW,
01931            xminNEW,
01932            yminNEW,
01933            zminNEW,
01934            xmaxNEW,
01935            ymaxNEW,
01936            zmaxNEW,
01937            x,
01938            y,
01939            z,
01940            dx,
01941            dy,
01942            dz,
01943            ifloat,
01944            jfloat,
01945            kfloat,
01946            uval,
01947            eps_w,
01948            T,
01949            pre1,
01950            xkappa,
01951            size,
01952            *apos,
01953            charge,
01954            //pos[3], // gcc: not used
```

```
01955           uvalMin,
01956           uvalMax,
01957           *data;
01958       int nxOLD,
01959           nyOLD,
01960           nzOLD,
01961           nxNEW,
01962           nyNEW,
01963           nzNEW,
01964           i,
01965           j,
01966           k,
01967           ihi,
01968           ilo,
01969           jhi,
01970           jlo,
01971           khi,
01972           klo,
01973           nx,
01974           ny,
01975           nz;
01976
01977       /* Calculate new problem dimensions */
01978       hxNEW = thee->pmgp->hx;
01979       hyNEW = thee->pmgp->hy;
01980       hzNEW = thee->pmgp->hzed;
01981       nx =  thee->pmgp->nx;
01982       ny =  thee->pmgp->ny;
01983       nz =  thee->pmgp->nz;
01984       nxNEW = thee->pmgp->nx;
01985       nyNEW = thee->pmgp->ny;
01986       nzNEW = thee->pmgp->nz;
01987       xminNEW = thee->pmgp->xcent - ((double)(nxNEW-1)*hxNEW)/2.0;
01988       xmaxNEW = thee->pmgp->xcent + ((double)(nxNEW-1)*hxNEW)/2.0;
01989       yminNEW = thee->pmgp->ycent - ((double)(nyNEW-1)*hyNEW)/2.0;
01990       ymaxNEW = thee->pmgp->ycent + ((double)(nyNEW-1)*hyNEW)/2.0;
01991       zminNEW = thee->pmgp->zcent - ((double)(nzNEW-1)*hzNEW)/2.0;
01992       zmaxNEW = thee->pmgp->zcent + ((double)(nzNEW-1)*hzNEW)/2.0;
01993
01994       if(pmgOLD != VNULL){
01995           /* Relevant old problem parameters */
01996           hxOLD = pmgOLD->pmgp->hx;
01997           hyOLD = pmgOLD->pmgp->hy;
01998           hzOLD = pmgOLD->pmgp->hzed;
01999           nxOLD = pmgOLD->pmgp->nx;
02000           nyOLD = pmgOLD->pmgp->ny;
02001           nzOLD = pmgOLD->pmgp->nz;
02002           xminOLD = pmgOLD->pmgp->xcent - ((double)(nxOLD-1)*hxOLD)/2.0;
02003           xmaxOLD = pmgOLD->pmgp->xcent + ((double)(nxOLD-1)*hxOLD)/2.0;
02004           yminOLD = pmgOLD->pmgp->ycent - ((double)(nyOLD-1)*hyOLD)/2.0;
02005           ymaxOLD = pmgOLD->pmgp->ycent + ((double)(nyOLD-1)*hyOLD)/2.0;
02006           zminOLD = pmgOLD->pmgp->zcent - ((double)(nzOLD-1)*hzOLD)/2.0;
02007           zmaxOLD = pmgOLD->pmgp->zcent + ((double)(nzOLD-1)*hzOLD)/2.0;
02008
02009           data = pmgOLD->u;
02010       }else{
02011           /* Relevant old problem parameters */
02012           hxOLD = thee->potMap->hx;
02013           hyOLD = thee->potMap->hy;
02014           hzOLD = thee->potMap->hzed;
02015           nxOLD = thee->potMap->nx;
02016           nyOLD = thee->potMap->ny;
02017           nzOLD = thee->potMap->nz;
02018           xminOLD = thee->potMap->xmin;
02019           xmaxOLD = thee->potMap->xmax;
02020           yminOLD = thee->potMap->ymin;
02021           ymaxOLD = thee->potMap->ymax;
02022           zminOLD = thee->potMap->zmin;
02023           zmaxOLD = thee->potMap->zmax;
02024
02025           data = thee->potMap->data;
02026       }
02027       /* BOUNDARY CONDITION SETUP FOR POINTS OFF OLD MESH:
02028        * For each "atom" (only one for bcfl=1), we use the following formula to
02029        * calculate the boundary conditions:
02030        *    g(x) = \frac{q e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
02031        *           * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
02032        *           * 1/d
02033        * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
02034        * We only need to evaluate some of these prefactors once:
02035        *    pre1 = \frac{e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
```

```
02036      * which gives the potential as
02037      *    g(x) = pre1 * q/d * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
02038      */
02039     pbe = thee->pbe;
02040     eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
02041     T = Vpbe_getTemperature(pbe);               /* K             */
02042     pre1 = (Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T);
02043
02044     /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
02045      * m/A, then we will only need to deal with distances and sizes in
02046      * Angstroms rather than meters.                                    */
02047     xkappa = Vpbe_getXkappa(pbe);              /* A^{-1}        */
02048     pre1 = pre1*(1.0e10);
02049     size = Vpbe_getSoluteRadius(pbe);
02050     apos = Vpbe_getSoluteCenter(pbe);
02051     charge = Vunit_ec*Vpbe_getSoluteCharge(pbe);
02052
02053     /* Check for rounding error */
02054     if (VABS(xminOLD-xminNEW) < VSMALL) xminNEW = xminOLD;
02055     if (VABS(xmaxOLD-xmaxNEW) < VSMALL) xmaxNEW = xmaxOLD;
02056     if (VABS(yminOLD-yminNEW) < VSMALL) yminNEW = yminOLD;
02057     if (VABS(ymaxOLD-ymaxNEW) < VSMALL) ymaxNEW = ymaxOLD;
02058     if (VABS(zminOLD-zminNEW) < VSMALL) zminNEW = zminOLD;
02059     if (VABS(zmaxOLD-zmaxNEW) < VSMALL) zmaxNEW = zmaxOLD;
02060
02061
02062     /* Sanity check: make sure we're within the old mesh */
02063     Vnm_print(0, "VPMG::focusFillBound -- New mesh mins = %g, %g, %g\n",
02064               xminNEW, yminNEW, zminNEW);
02065     Vnm_print(0, "VPMG::focusFillBound -- New mesh maxs = %g, %g, %g\n",
02066               xmaxNEW, ymaxNEW, zmaxNEW);
02067     Vnm_print(0, "VPMG::focusFillBound -- Old mesh mins = %g, %g, %g\n",
02068               xminOLD, yminOLD, zminOLD);
02069     Vnm_print(0, "VPMG::focusFillBound -- Old mesh maxs = %g, %g, %g\n",
02070               xmaxOLD, ymaxOLD, zmaxOLD);
02071
02072     /* The following is obsolete; we'll substitute analytical boundary
02073      * condition values when the new mesh falls outside the old */
02074     if ((xmaxNEW>xmaxOLD) || (ymaxNEW>ymaxOLD) || (zmaxNEW>zmaxOLD) ||
02075         (xminOLD>xminNEW) || (yminOLD>yminNEW) || (zminOLD>zminNEW)) {
02076
02077         Vnm_print(2, "Vpmg::focusFillBound -- new mesh not contained in old!\n");
02078         Vnm_print(2, "Vpmg::focusFillBound -- old mesh min = (%g, %g, %g)\n",
02079                   xminOLD, yminOLD, zminOLD);
02080         Vnm_print(2, "Vpmg::focusFillBound -- old mesh max = (%g, %g, %g)\n",
02081                   xmaxOLD, ymaxOLD, zmaxOLD);
02082         Vnm_print(2, "Vpmg::focusFillBound -- new mesh min = (%g, %g, %g)\n",
02083                   xminNEW, yminNEW, zminNEW);
02084         Vnm_print(2, "Vpmg::focusFillBound -- new mesh max = (%g, %g, %g)\n",
02085                   xmaxNEW, ymaxNEW, zmaxNEW);
02086         fflush(stderr);
02087         VASSERT(0);
02088     }
02089
02090     uvalMin = VPMGSMALL;
02091     uvalMax = -VPMGSMALL;
02092
02093     /* Fill the "i" boundaries (dirichlet) */
02094     for (k=0; k<nzNEW; k++) {
02095         for (j=0; j<nyNEW; j++) {
02096             /* Low X face */
02097             x = xminNEW;
02098             y = yminNEW + j*hyNEW;
02099             z = zminNEW + k*hzNEW;
02100             if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&
02101                 (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
02102                 ifloat = (x - xminOLD)/hxOLD;
02103                 jfloat = (y - yminOLD)/hyOLD;
02104                 kfloat = (z - zminOLD)/hzOLD;
02105                 ihi = (int)ceil(ifloat);
02106                 if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02107                 ilo = (int)floor(ifloat);
02108                 if (ilo < 0) ilo = 0;
02109                 jhi = (int)ceil(jfloat);
02110                 if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02111                 jlo = (int)floor(jfloat);
02112                 if (jlo < 0) jlo = 0;
02113                 khi = (int)ceil(kfloat);
02114                 if (khi > (nzOLD-1)) khi = nzOLD-1;
02115                 klo = (int)floor(kfloat);
02116                 if (klo < 0) klo = 0;
```

```
02117                 dx = ifloat - (double)(ilo);
02118                 dy = jfloat - (double)(jlo);
02119                 dz = kfloat - (double)(klo);
02120                 nx = nxOLD; ny = nyOLD; nz = nzOLD;
02121                 uval =  dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02122                 + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02123                 + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02124                 + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02125                 + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02126                 + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02127                 + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02128                 + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02129                 nx = nxNEW; ny = nyNEW; nz = nzNEW;
02130             } else {
02131                 Vnm_print(2, "focusFillBound (%s, %d):  Off old mesh at %g, %g \
02132                           %g!\n", __FILE__, __LINE__, x, y, z);
02133                 Vnm_print(2, "focusFillBound (%s, %d):  old mesh lower corner at \
02134                           %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02135                 Vnm_print(2, "focusFillBound (%s, %d):  old mesh upper corner at \
02136                           %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02137                 VASSERT(0);
02138             }
02139             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02140             thee->gxcf[IJKx(j,k,0)] = uval;
02141             if(uval < uvalMin) uvalMin = uval;
02142             if(uval > uvalMax) uvalMax = uval;
02143 
02144             /* High X face */
02145             x = xmaxNEW;
02146             if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&
02147                 (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
02148                 ifloat = (x - xminOLD)/hxOLD;
02149                 jfloat = (y - yminOLD)/hyOLD;
02150                 kfloat = (z - zminOLD)/hzOLD;
02151                 ihi = (int)ceil(ifloat);
02152                 if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02153                 ilo = (int)floor(ifloat);
02154                 if (ilo < 0) ilo = 0;
02155                 jhi = (int)ceil(jfloat);
02156                 if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02157                 jlo = (int)floor(jfloat);
02158                 if (jlo < 0) jlo = 0;
02159                 khi = (int)ceil(kfloat);
02160                 if (khi > (nzOLD-1)) khi = nzOLD-1;
02161                 klo = (int)floor(kfloat);
02162                 if (klo < 0) klo = 0;
02163                 dx = ifloat - (double)(ilo);
02164                 dy = jfloat - (double)(jlo);
02165                 dz = kfloat - (double)(klo);
02166                 nx = nxOLD; ny = nyOLD; nz = nzOLD;
02167                 uval =  dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02168                 + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02169                 + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02170                 + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02171                 + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02172                 + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02173                 + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02174                 + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02175                 nx = nxNEW; ny = nyNEW; nz = nzNEW;
02176             } else {
02177                 Vnm_print(2, "focusFillBound (%s, %d):  Off old mesh at %g, %g \
02178                           %g!\n", __FILE__, __LINE__, x, y, z);
02179                 Vnm_print(2, "focusFillBound (%s, %d):  old mesh lower corner at \
02180                           %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02181                 Vnm_print(2, "focusFillBound (%s, %d):  old mesh upper corner at \
02182                           %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02183                 VASSERT(0);
02184             }
02185             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02186             thee->gxcf[IJKx(j,k,1)] = uval;
02187             if(uval < uvalMin) uvalMin = uval;
02188             if(uval > uvalMax) uvalMax = uval;
02189 
02190             /* Zero Neumann conditions */
02191             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02192             thee->gxcf[IJKx(j,k,2)] = 0.0;
02193             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02194             thee->gxcf[IJKx(j,k,3)] = 0.0;
02195         }
02196     }
02197 
```

```
02198     /* Fill the "j" boundaries (dirichlet) */
02199     for (k=0; k<nzNEW; k++) {
02200         for (i=0; i<nxNEW; i++) {
02201             /* Low Y face */
02202             x = xminNEW + i*hxNEW;
02203             y = yminNEW;
02204             z = zminNEW + k*hzNEW;
02205             if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&
02206                 (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
02207                 ifloat = (x - xminOLD)/hxOLD;
02208                 jfloat = (y - yminOLD)/hyOLD;
02209                 kfloat = (z - zminOLD)/hzOLD;
02210                 ihi = (int)ceil(ifloat);
02211                 if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02212                 ilo = (int)floor(ifloat);
02213                 if (ilo < 0) ilo = 0;
02214                 jhi = (int)ceil(jfloat);
02215                 if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02216                 jlo = (int)floor(jfloat);
02217                 if (jlo < 0) jlo = 0;
02218                 khi = (int)ceil(kfloat);
02219                 if (khi > (nzOLD-1)) khi = nzOLD-1;
02220                 klo = (int)floor(kfloat);
02221                 if (klo < 0) klo = 0;
02222                 dx = ifloat - (double)(ilo);
02223                 dy = jfloat - (double)(jlo);
02224                 dz = kfloat - (double)(klo);
02225                 nx = nxOLD; ny = nyOLD; nz = nzOLD;
02226                 uval =  dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02227                 + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02228                 + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02229                 + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02230                 + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02231                 + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02232                 + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02233                 + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02234                 nx = nxNEW; ny = nyNEW; nz = nzNEW;
02235             } else {
02236                 Vnm_print(2, "focusFillBound (%s, %d):  Off old mesh at %g, %g \
02237                           %g!\n", __FILE__, __LINE__, x, y, z);
02238                 Vnm_print(2, "focusFillBound (%s, %d):  old mesh lower corner at \
02239                           %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02240                 Vnm_print(2, "focusFillBound (%s, %d):  old mesh upper corner at \
02241                           %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02242                 VASSERT(0);
02243             }
02244             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02245             thee->gycf[IJKy(i,k,0)] = uval;
02246             if(uval < uvalMin) uvalMin = uval;
02247             if(uval > uvalMax) uvalMax = uval;
02248
02249             /* High Y face */
02250             y = ymaxNEW;
02251             if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&
02252                 (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
02253                 ifloat = (x - xminOLD)/hxOLD;
02254                 jfloat = (y - yminOLD)/hyOLD;
02255                 kfloat = (z - zminOLD)/hzOLD;
02256                 ihi = (int)ceil(ifloat);
02257                 if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02258                 ilo = (int)floor(ifloat);
02259                 if (ilo < 0) ilo = 0;
02260                 jhi = (int)ceil(jfloat);
02261                 if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02262                 jlo = (int)floor(jfloat);
02263                 if (jlo < 0) jlo = 0;
02264                 khi = (int)ceil(kfloat);
02265                 if (khi > (nzOLD-1)) khi = nzOLD-1;
02266                 klo = (int)floor(kfloat);
02267                 if (klo < 0) klo = 0;
02268                 dx = ifloat - (double)(ilo);
02269                 dy = jfloat - (double)(jlo);
02270                 dz = kfloat - (double)(klo);
02271                 nx = nxOLD; ny = nyOLD; nz = nzOLD;
02272                 uval =  dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02273                 + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02274                 + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02275                 + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02276                 + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02277                 + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02278                 + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
```

```
02279                 + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02280                 nx = nxNEW; ny = nyNEW; nz = nzNEW;
02281             } else {
02282                 Vnm_print(2, "focusFillBound (%s, %d):  Off old mesh at %g, %g \
02283                           %g!\n", __FILE__, __LINE__, x, y, z);
02284                 Vnm_print(2, "focusFillBound (%s, %d):  old mesh lower corner at \
02285                           %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02286                 Vnm_print(2, "focusFillBound (%s, %d):  old mesh upper corner at \
02287                           %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02288                 VASSERT(0);
02289             }
02290             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02291             thee->gycf[IJKy(i,k,1)] = uval;
02292             if(uval < uvalMin) uvalMin = uval;
02293             if(uval > uvalMax) uvalMax = uval;
02294 
02295             /* Zero Neumann conditions */
02296             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02297             thee->gycf[IJKy(i,k,2)] = 0.0;
02298             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02299             thee->gycf[IJKy(i,k,3)] = 0.0;
02300         }
02301     }
02302 
02303     /* Fill the "k" boundaries (dirichlet) */
02304     for (j=0; j<nyNEW; j++) {
02305         for (i=0; i<nxNEW; i++) {
02306             /* Low Z face */
02307             x = xminNEW + i*hxNEW;
02308             y = yminNEW + j*hyNEW;
02309             z = zminNEW;
02310             if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&
02311                 (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
02312                 ifloat = (x - xminOLD)/hxOLD;
02313                 jfloat = (y - yminOLD)/hyOLD;
02314                 kfloat = (z - zminOLD)/hzOLD;
02315                 ihi = (int)ceil(ifloat);
02316                 if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02317                 ilo = (int)floor(ifloat);
02318                 if (ilo < 0) ilo = 0;
02319                 jhi = (int)ceil(jfloat);
02320                 if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02321                 jlo = (int)floor(jfloat);
02322                 if (jlo < 0) jlo = 0;
02323                 khi = (int)ceil(kfloat);
02324                 if (khi > (nzOLD-1)) khi = nzOLD-1;
02325                 klo = (int)floor(kfloat);
02326                 if (klo < 0) klo = 0;
02327                 dx = ifloat - (double)(ilo);
02328                 dy = jfloat - (double)(jlo);
02329                 dz = kfloat - (double)(klo);
02330                 nx = nxOLD; ny = nyOLD; nz = nzOLD;
02331                 uval =  dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02332                 + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02333                 + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02334                 + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02335                 + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02336                 + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02337                 + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02338                 + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02339                 nx = nxNEW; ny = nyNEW; nz = nzNEW;
02340             } else {
02341                 Vnm_print(2, "focusFillBound (%s, %d):  Off old mesh at %g, %g \
02342                           %g!\n", __FILE__, __LINE__, x, y, z);
02343                 Vnm_print(2, "focusFillBound (%s, %d):  old mesh lower corner at \
02344                           %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02345                 Vnm_print(2, "focusFillBound (%s, %d):  old mesh upper corner at \
02346                           %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02347                 VASSERT(0);
02348             }
02349             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02350             thee->gzcf[IJKz(i,j,0)] = uval;
02351             if(uval < uvalMin) uvalMin = uval;
02352             if(uval > uvalMax) uvalMax = uval;
02353 
02354             /* High Z face */
02355             z = zmaxNEW;
02356             if ((x >= (xminOLD-VSMALL)) && (y >= (yminOLD-VSMALL)) && (z >= (zminOLD-VSMALL)) &&
02357                 (x <= (xmaxOLD+VSMALL)) && (y <= (ymaxOLD+VSMALL)) && (z <= (zmaxOLD+VSMALL))) {
02358                 ifloat = (x - xminOLD)/hxOLD;
02359                 jfloat = (y - yminOLD)/hyOLD;
```

```
02360                 kfloat = (z - zminOLD)/hzOLD;
02361                 ihi = (int)ceil(ifloat);
02362                 if (ihi > (nxOLD-1)) ihi = nxOLD-1;
02363                 ilo = (int)floor(ifloat);
02364                 if (ilo < 0) ilo = 0;
02365                 jhi = (int)ceil(jfloat);
02366                 if (jhi > (nyOLD-1)) jhi = nyOLD-1;
02367                 jlo = (int)floor(jfloat);
02368                 if (jlo < 0) jlo = 0;
02369                 khi = (int)ceil(kfloat);
02370                 if (khi > (nzOLD-1)) khi = nzOLD-1;
02371                 klo = (int)floor(kfloat);
02372                 if (klo < 0) klo = 0;
02373                 dx = ifloat - (double)(ilo);
02374                 dy = jfloat - (double)(jlo);
02375                 dz = kfloat - (double)(klo);
02376                 nx = nxOLD; ny = nyOLD; nz = nzOLD;
02377                 uval =  dx*dy*dz*(data[IJK(ihi,jhi,khi)])
02378                 + dx*(1.0-dy)*dz*(data[IJK(ihi,jlo,khi)])
02379                 + dx*dy*(1.0-dz)*(data[IJK(ihi,jhi,klo)])
02380                 + dx*(1.0-dy)*(1.0-dz)*(data[IJK(ihi,jlo,klo)])
02381                 + (1.0-dx)*dy*dz*(data[IJK(ilo,jhi,khi)])
02382                 + (1.0-dx)*(1.0-dy)*dz*(data[IJK(ilo,jlo,khi)])
02383                 + (1.0-dx)*dy*(1.0-dz)*(data[IJK(ilo,jhi,klo)])
02384                 + (1.0-dx)*(1.0-dy)*(1.0-dz)*(data[IJK(ilo,jlo,klo)]);
02385                 nx = nxNEW; ny = nyNEW; nz = nzNEW;
02386             } else {
02387                 Vnm_print(2, "focusFillBound (%s, %d):  Off old mesh at %g, %g \
02388                           %g!\n", __FILE__, __LINE__, x, y, z);
02389                 Vnm_print(2, "focusFillBound (%s, %d):  old mesh lower corner at \
02390                           %g %g %g.\n", __FILE__, __LINE__, xminOLD, yminOLD, zminOLD);
02391                 Vnm_print(2, "focusFillBound (%s, %d):  old mesh upper corner at \
02392                           %g %g %g.\n", __FILE__, __LINE__, xmaxOLD, ymaxOLD, zmaxOLD);
02393                 VASSERT(0);
02394             }
02395             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02396             thee->gzcf[IJKz(i,j,1)] = uval;
02397             if(uval < uvalMin) uvalMin = uval;
02398             if(uval > uvalMax) uvalMax = uval;
02399 
02400             /* Zero Neumann conditions */
02401             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02402             thee->gzcf[IJKz(i,j,2)] = 0.0;
02403             nx = nxNEW; ny = nyNEW; nz = nzNEW;
02404             thee->gzcf[IJKz(i,j,3)] = 0.0;
02405         }
02406     }
02407 
02408     VWARN_MSG0(
02409         uvalMin >= SINH_MIN && uvalMax <= SINH_MAX,
02410         "Unusually large potential values\n"
02411         "    detected on the focusing boundary!\n"
02412         "    Convergence not guaranteed for NPBE/NRPBE calculations!"
02413         );
02414 }
02415 
02416 VPRIVATE void extEnergy(Vpmg *thee, Vpmg *pmgOLD, PBEparm_calcEnergy extFlag,
02417                         double partMin[3], double partMax[3], int bflags[6]) {
02418 
02419     Vatom *atom;
02420     double hxNEW, hyNEW, hzNEW;
02421     double lowerCorner[3], upperCorner[3];
02422     int nxNEW, nyNEW, nzNEW;
02423     int nxOLD, nyOLD, nzOLD;
02424     int i,j,k;
02425     double xmin, xmax, ymin, ymax, zmin, zmax;
02426     double hxOLD, hyOLD, hzOLD;
02427     double xval, yval, zval;
02428     double x,y,z;
02429     int nx, ny, nz;
02430 
02431     /* Set the new external energy contribution to zero.  Any external
02432      * contributions from higher levels will be included in the appropriate
02433      * energy function call. */
02434     thee->extQmEnergy = 0;
02435     thee->extQfEnergy = 0;
02436     thee->extDiEnergy = 0;
02437 
02438     /* New problem dimensions */
02439     hxNEW = thee->pmgp->hx;
02440     hyNEW = thee->pmgp->hy;
```

```
02441     hzNEW = thee->pmgp->hzed;
02442     nxNEW = thee->pmgp->nx;
02443     nyNEW = thee->pmgp->ny;
02444     nzNEW = thee->pmgp->nz;
02445     lowerCorner[0] = thee->pmgp->xcent - ((double)(nxNEW-1)*hxNEW)/2.0;
02446     upperCorner[0] = thee->pmgp->xcent + ((double)(nxNEW-1)*hxNEW)/2.0;
02447     lowerCorner[1] = thee->pmgp->ycent - ((double)(nyNEW-1)*hyNEW)/2.0;
02448     upperCorner[1] = thee->pmgp->ycent + ((double)(nyNEW-1)*hyNEW)/2.0;
02449     lowerCorner[2] = thee->pmgp->zcent - ((double)(nzNEW-1)*hzNEW)/2.0;
02450     upperCorner[2] = thee->pmgp->zcent + ((double)(nzNEW-1)*hzNEW)/2.0;
02451
02452     Vnm_print(0, "VPMG::extEnergy:  energy flag = %d\n", extFlag);
02453
02454     /* Old problem dimensions */
02455     nxOLD = pmgOLD->pmgp->nx;
02456     nyOLD = pmgOLD->pmgp->ny;
02457     nzOLD = pmgOLD->pmgp->nz;
02458
02459     /* Create a partition based on the new problem dimensions */
02460     /* Vnm_print(1, "DEBUG (%s, %d):  extEnergy calling Vpmg_setPart for old PMG.\n",
02461      __FILE__, __LINE__); */
02462     Vpmg_setPart(pmgOLD, lowerCorner, upperCorner, bflags);
02463
02464
02465     Vnm_print(0,"VPMG::extEnergy:   Finding extEnergy dimensions...\n");
02466     Vnm_print(0,"VPMG::extEnergy    Disj part lower corner = (%g, %g, %g)\n",
02467               partMin[0], partMin[1], partMin[2]);
02468     Vnm_print(0,"VPMG::extEnergy    Disj part upper corner = (%g, %g, %g)\n",
02469               partMax[0], partMax[1], partMax[2]);
02470
02471     /* Find the old dimensions */
02472
02473     hxOLD = pmgOLD->pmgp->hx;
02474     hyOLD = pmgOLD->pmgp->hy;
02475     hzOLD = pmgOLD->pmgp->hzed;
02476     xmin =  pmgOLD->pmgp->xcent - 0.5*hxOLD*(nxOLD-1);
02477     ymin =  pmgOLD->pmgp->ycent - 0.5*hyOLD*(nyOLD-1);
02478     zmin =  pmgOLD->pmgp->zcent - 0.5*hzOLD*(nzOLD-1);
02479     xmax =  xmin+hxOLD*(nxOLD-1);
02480     ymax =  ymin+hyOLD*(nyOLD-1);
02481     zmax =  zmin+hzOLD*(nzOLD-1);
02482
02483     Vnm_print(0,"VPMG::extEnergy    Old lower corner = (%g, %g, %g)\n",
02484               xmin, ymin, zmin);
02485     Vnm_print(0,"VPMG::extEnergy    Old upper corner = (%g, %g, %g)\n",
02486               xmax, ymax, zmax);
02487
02488     /* Flip the partition, but do not include any points that will
02489      be included by another processor */
02490
02491     nx = nxOLD;
02492     ny = nyOLD;
02493     nz = nzOLD;
02494
02495     for(i=0; i<nx; i++) {
02496         xval = 1;
02497         x = i*hxOLD + xmin;
02498         if (x < partMin[0] && bflags[VAPBS_LEFT] == 1) xval = 0;
02499         else if (x > partMax[0] && bflags[VAPBS_RIGHT] == 1) xval = 0;
02500
02501         for(j=0; j<ny; j++) {
02502             yval = 1;
02503             y = j*hyOLD + ymin;
02504             if (y < partMin[1] && bflags[VAPBS_BACK] == 1) yval = 0;
02505             else if (y > partMax[1] && bflags[VAPBS_FRONT] == 1) yval = 0;
02506
02507             for(k=0; k<nz; k++) {
02508                 zval = 1;
02509                 z = k*hzOLD + zmin;
02510                 if (z < partMin[2] && bflags[VAPBS_DOWN] == 1) zval = 0;
02511                 else if (z > partMax[2] && bflags[VAPBS_UP] == 1) zval = 0;
02512
02513                 if (pmgOLD->pvec[IJK(i,j,k)] > VSMALL) pmgOLD->pvec[IJK(i,j,k)] = 1.0;
02514                 pmgOLD->pvec[IJK(i,j,k)] = (1 - (pmgOLD->pvec[IJK(i,j,k)])) * (xval*yval*zval);
02515             }
02516         }
02517     }
02518
02519     for (i=0; i<Valist_getNumberAtoms(thee->pbe->alist); i++) {
02520         xval=1;
02521         yval=1;
```

```
02522         zval=1;
02523         atom = Valist_getAtom(thee->pbe->alist, i);
02524         x = atom->position[0];
02525         y = atom->position[1];
02526         z = atom->position[2];
02527         if (x < partMin[0] && bflags[VAPBS_LEFT] == 1) xval = 0;
02528         else if (x > partMax[0] && bflags[VAPBS_RIGHT] == 1) xval = 0;
02529         if (y < partMin[1] && bflags[VAPBS_BACK] == 1) yval = 0;
02530         else if (y > partMax[1] && bflags[VAPBS_FRONT] == 1) yval = 0;
02531         if (z < partMin[2] && bflags[VAPBS_DOWN] == 1) zval = 0;
02532         else if (z > partMax[2] && bflags[VAPBS_UP] == 1) zval = 0;
02533         if (atom->partID > VSMALL) atom->partID = 1.0;
02534         atom->partID = (1 - atom->partID) * (xval*yval*zval);
02535     }
02536
02537     /* Now calculate the energy on inverted subset of the domain */
02538     thee->extQmEnergy = Vpmg_qmEnergy(pmgOLD, 1);
02539     Vnm_print(0, "VPMG::extEnergy: extQmEnergy = %g kT\n", thee->extQmEnergy);
02540     thee->extQfEnergy = Vpmg_qfEnergy(pmgOLD, 1);
02541     Vnm_print(0, "VPMG::extEnergy: extQfEnergy = %g kT\n", thee->extQfEnergy);
02542     thee->extDiEnergy = Vpmg_dielEnergy(pmgOLD, 1);
02543     Vnm_print(0, "VPMG::extEnergy: extDiEnergy = %g kT\n", thee->extDiEnergy);
02544     Vpmg_unsetPart(pmgOLD);
02545 }
02546
02547 VPRIVATE double bcfl1sp(double size, double *apos, double charge,
02548                         double xkappa, double pre1, double *pos) {
02549
02550     double dist, val;
02551
02552     dist = VSQRT(VSQR(pos[0]-apos[0]) + VSQR(pos[1]-apos[1])
02553                  + VSQR(pos[2]-apos[2]));
02554     if (xkappa > VSMALL) {
02555         val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02556         / (1+xkappa*size);
02557     } else {
02558         val = pre1*(charge/dist);
02559     }
02560
02561     return val;
02562 }
02563
02564 VPRIVATE void bcfl1(double size, double *apos, double charge,
02565                     double xkappa, double pre1, double *gxcf, double *gycf, double *gzcf,
02566                     double *xf, double *yf, double *zf, int nx, int ny, int nz) {
02567
02568     int i, j, k;
02569     double dist, val;
02570     double gpos[3];
02571
02572     /* the "i" boundaries (dirichlet) */
02573     for (k=0; k<nz; k++) {
02574         gpos[2] = zf[k];
02575         for (j=0; j<ny; j++) {
02576             gpos[1] = yf[j];
02577             gpos[0] = xf[0];
02578             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02579                          + VSQR(gpos[2]-apos[2]));
02580             if (xkappa > VSMALL) {
02581                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02582                 / (1+xkappa*size);
02583             } else {
02584                 val = pre1*(charge/dist);
02585             }
02586             gxcf[IJKx(j,k,0)] += val;
02587             gpos[0] = xf[nx-1];
02588             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02589                          + VSQR(gpos[2]-apos[2]));
02590             if (xkappa > VSMALL) {
02591                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02592                 / (1+xkappa*size);
02593             } else {
02594                 val = pre1*(charge/dist);
02595             }
02596             gxcf[IJKx(j,k,1)] += val;
02597         }
02598     }
02599
02600     /* the "j" boundaries (dirichlet) */
02601     for (k=0; k<nz; k++) {
02602         gpos[2] = zf[k];
```

```
02603         for (i=0; i<nx; i++) {
02604             gpos[0] = xf[i];
02605             gpos[1] = yf[0];
02606             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02607                          + VSQR(gpos[2]-apos[2]));
02608             if (xkappa > VSMALL) {
02609                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02610                 / (1+xkappa*size);
02611             } else {
02612                 val = pre1*(charge/dist);
02613             }
02614             gycf[IJKy(i,k,0)] += val;
02615             gpos[1] = yf[ny-1];
02616             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02617                          + VSQR(gpos[2]-apos[2]));
02618             if (xkappa > VSMALL) {
02619                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02620                 / (1+xkappa*size);
02621             } else {
02622                 val = pre1*(charge/dist);
02623             }
02624             gycf[IJKy(i,k,1)] += val;
02625         }
02626     }
02627
02628     /* the "k" boundaries (dirichlet) */
02629     for (j=0; j<ny; j++) {
02630         gpos[1] = yf[j];
02631         for (i=0; i<nx; i++) {
02632             gpos[0] = xf[i];
02633             gpos[2] = zf[0];
02634             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02635                          + VSQR(gpos[2]-apos[2]));
02636             if (xkappa > VSMALL) {
02637                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02638                 / (1+xkappa*size);
02639             } else {
02640                 val = pre1*(charge/dist);
02641             }
02642             gzcf[IJKz(i,j,0)] += val;
02643             gpos[2] = zf[nz-1];
02644             dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
02645                          + VSQR(gpos[2]-apos[2]));
02646             if (xkappa > VSMALL) {
02647                 val = pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
02648                 / (1+xkappa*size);
02649             } else {
02650                 val = pre1*(charge/dist);
02651             }
02652             gzcf[IJKz(i,j,1)] += val;
02653         }
02654     }
02655 }
02656
02657 VPRIVATE void bcfl2(double size, double *apos,
02658                     double charge, double *dipole, double *quad,
02659                     double xkappa, double eps_p, double eps_w, double T,
02660                     double *gxcf, double *gycf, double *gzcf,
02661                     double *xf, double *yf, double *zf,
02662                     int nx, int ny, int nz) {
02663
02664     int i, j, k;
02665     double val;
02666     double gpos[3],tensor[3];
02667     double ux,uy,uz,xr,yr,zr;
02668     double qxx,qxy,qxz,qyx,qyy,qyz,qzx,qzy,qzz;
02669     double dist, pre;
02670
02671     VASSERT(dipole != VNULL);
02672     ux = dipole[0];
02673     uy = dipole[1];
02674     uz = dipole[2];
02675     if (quad != VNULL) {
02676         /* The factor of 1/3 results from using a
02677          traceless quadrupole definition. See, for example,
02678          "The Theory of Intermolecular Forces" by A.J. Stone,
02679          Chapter 3. */
02680         qxx = quad[0] / 3.0;
02681         qxy = quad[1] / 3.0;
02682         qxz = quad[2] / 3.0;
02683         qyx = quad[3] / 3.0;
```

```
02684         qyy = quad[4] / 3.0;
02685         qyz = quad[5] / 3.0;
02686         qzx = quad[6] / 3.0;
02687         qzy = quad[7] / 3.0;
02688         qzz = quad[8] / 3.0;
02689     } else {
02690         qxx = 0.0;
02691         qxy = 0.0;
02692         qxz = 0.0;
02693         qyx = 0.0;
02694         qyy = 0.0;
02695         qyz = 0.0;
02696         qzx = 0.0;
02697         qzy = 0.0;
02698         qzz = 0.0;
02699     }
02700
02701     pre = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*Vunit_kb*T);
02702     pre = pre*(1.0e10);
02703
02704     /* the "i" boundaries (dirichlet) */
02705     for (k=0; k<nz; k++) {
02706         gpos[2] = zf[k];
02707         for (j=0; j<ny; j++) {
02708             gpos[1] = yf[j];
02709             gpos[0] = xf[0];
02710             xr = gpos[0] - apos[0];
02711             yr = gpos[1] - apos[1];
02712             zr = gpos[2] - apos[2];
02713             dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02714             multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02715             val = pre*charge*tensor[0];
02716             val -= pre*ux*xr*tensor[1];
02717             val -= pre*uy*yr*tensor[1];
02718             val -= pre*uz*zr*tensor[1];
02719             val += pre*qxx*xr*xr*tensor[2];
02720             val += pre*qyy*yr*yr*tensor[2];
02721             val += pre*qzz*zr*zr*tensor[2];
02722             val += pre*2.0*qxy*xr*yr*tensor[2];
02723             val += pre*2.0*qxz*xr*zr*tensor[2];
02724             val += pre*2.0*qyz*yr*zr*tensor[2];
02725             gxcf[IJKx(j,k,0)] += val;
02726
02727             gpos[0] = xf[nx-1];
02728             xr = gpos[0] - apos[0];
02729             dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02730             multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02731             val = pre*charge*tensor[0];
02732             val -= pre*ux*xr*tensor[1];
02733             val -= pre*uy*yr*tensor[1];
02734             val -= pre*uz*zr*tensor[1];
02735             val += pre*qxx*xr*xr*tensor[2];
02736             val += pre*qyy*yr*yr*tensor[2];
02737             val += pre*qzz*zr*zr*tensor[2];
02738             val += pre*2.0*qxy*xr*yr*tensor[2];
02739             val += pre*2.0*qxz*xr*zr*tensor[2];
02740             val += pre*2.0*qyz*yr*zr*tensor[2];
02741             gxcf[IJKx(j,k,1)] += val;
02742         }
02743     }
02744
02745     /* the "j" boundaries (dirichlet) */
02746     for (k=0; k<nz; k++) {
02747         gpos[2] = zf[k];
02748         for (i=0; i<nx; i++) {
02749             gpos[0] = xf[i];
02750             gpos[1] = yf[0];
02751             xr = gpos[0] - apos[0];
02752             yr = gpos[1] - apos[1];
02753             zr = gpos[2] - apos[2];
02754             dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02755             multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02756             val = pre*charge*tensor[0];
02757             val -= pre*ux*xr*tensor[1];
02758             val -= pre*uy*yr*tensor[1];
02759             val -= pre*uz*zr*tensor[1];
02760             val += pre*qxx*xr*xr*tensor[2];
02761             val += pre*qyy*yr*yr*tensor[2];
02762             val += pre*qzz*zr*zr*tensor[2];
02763             val += pre*2.0*qxy*xr*yr*tensor[2];
02764             val += pre*2.0*qxz*xr*zr*tensor[2];
```

```
02765             val += pre*2.0*qyz*yr*zr*tensor[2];
02766             gycf[IJKy(i,k,0)] += val;
02767
02768             gpos[1] = yf[ny-1];
02769             yr = gpos[1] - apos[1];
02770             dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02771             multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02772             val = pre*charge*tensor[0];
02773             val -= pre*ux*xr*tensor[1];
02774             val -= pre*uy*yr*tensor[1];
02775             val -= pre*uz*zr*tensor[1];
02776             val += pre*qxx*xr*xr*tensor[2];
02777             val += pre*qyy*yr*yr*tensor[2];
02778             val += pre*qzz*zr*zr*tensor[2];
02779             val += pre*2.0*qxy*xr*yr*tensor[2];
02780             val += pre*2.0*qxz*xr*zr*tensor[2];
02781             val += pre*2.0*qyz*yr*zr*tensor[2];
02782             gycf[IJKy(i,k,1)] += val;
02783         }
02784     }
02785
02786     /* the "k" boundaries (dirichlet) */
02787     for (j=0; j<ny; j++) {
02788         gpos[1] = yf[j];
02789         for (i=0; i<nx; i++) {
02790             gpos[0] = xf[i];
02791             gpos[2] = zf[0];
02792             xr = gpos[0] - apos[0];
02793             yr = gpos[1] - apos[1];
02794             zr = gpos[2] - apos[2];
02795             dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02796             multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02797             val = pre*charge*tensor[0];
02798             val -= pre*ux*xr*tensor[1];
02799             val -= pre*uy*yr*tensor[1];
02800             val -= pre*uz*zr*tensor[1];
02801             val += pre*qxx*xr*xr*tensor[2];
02802             val += pre*qyy*yr*yr*tensor[2];
02803             val += pre*qzz*zr*zr*tensor[2];
02804             val += pre*2.0*qxy*xr*yr*tensor[2];
02805             val += pre*2.0*qxz*xr*zr*tensor[2];
02806             val += pre*2.0*qyz*yr*zr*tensor[2];
02807             gzcf[IJKz(i,j,0)] += val;
02808
02809             gpos[2] = zf[nz-1];
02810             zr = gpos[2] - apos[2];
02811             dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
02812             multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
02813             val = pre*charge*tensor[0];
02814             val -= pre*ux*xr*tensor[1];
02815             val -= pre*uy*yr*tensor[1];
02816             val -= pre*uz*zr*tensor[1];
02817             val += pre*qxx*xr*xr*tensor[2];
02818             val += pre*qyy*yr*yr*tensor[2];
02819             val += pre*qzz*zr*zr*tensor[2];
02820             val += pre*2.0*qxy*xr*yr*tensor[2];
02821             val += pre*2.0*qxz*xr*zr*tensor[2];
02822             val += pre*2.0*qyz*yr*zr*tensor[2];
02823             gzcf[IJKz(i,j,1)] += val;
02824         }
02825     }
02826 }
02827
02828 VPRIVATE void bcCalcOrig(Vpmg *thee) {
02829
02830     int nx, ny, nz;
02831     double size, *position, charge, xkappa, eps_w, T, pre1;
02832     double *dipole, *quadrupole, debye, eps_p;
02833     double xr,yr,zr,qave,*apos;
02834     double sdhcharge, sdhdipole[3], traced[9], sdhquadrupole[9];
02835     int i, j, k, iatom;
02836     Vpbe *pbe;
02837     Vatom *atom;
02838     Valist *alist;
02839
02840     pbe = thee->pbe;
02841     alist = thee->pbe->alist;
02842     nx = thee->pmgp->nx;
02843     ny = thee->pmgp->ny;
02844     nz = thee->pmgp->nz;
02845
```

```
02846     /* Zero out the boundaries */
02847     /* the "i" boundaries (dirichlet) */
02848     for (k=0; k<nz; k++) {
02849         for (j=0; j<ny; j++) {
02850             thee->gxcf[IJKx(j,k,0)] = 0.0;
02851             thee->gxcf[IJKx(j,k,1)] = 0.0;
02852             thee->gxcf[IJKx(j,k,2)] = 0.0;
02853             thee->gxcf[IJKx(j,k,3)] = 0.0;
02854         }
02855     }
02856
02857     /* the "j" boundaries (dirichlet) */
02858     for (k=0; k<nz; k++) {
02859         for (i=0; i<nx; i++) {
02860             thee->gycf[IJKy(i,k,0)] = 0.0;
02861             thee->gycf[IJKy(i,k,1)] = 0.0;
02862             thee->gycf[IJKy(i,k,2)] = 0.0;
02863             thee->gycf[IJKy(i,k,3)] = 0.0;
02864         }
02865     }
02866
02867     /* the "k" boundaries (dirichlet) */
02868     for (j=0; j<ny; j++) {
02869         for (i=0; i<nx; i++) {
02870             thee->gzcf[IJKz(i,j,0)] = 0.0;
02871             thee->gzcf[IJKz(i,j,1)] = 0.0;
02872             thee->gzcf[IJKz(i,j,2)] = 0.0;
02873             thee->gzcf[IJKz(i,j,3)] = 0.0;
02874         }
02875     }
02876
02877     /* For each "atom" (only one for bcfl=1), we use the following formula to
02878     * calculate the boundary conditions:
02879     *    g(x) = \frac{q e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
02880     *          * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
02881     *          * 1/d
02882     * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
02883     * We only need to evaluate some of these prefactors once:
02884     *    pre1 = \frac{e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
02885     * which gives the potential as
02886     *    g(x) = pre1 * q/d * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
02887     */
02888     eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
02889     eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
02890     T = Vpbe_getTemperature(pbe);               /* K             */
02891     pre1 = (Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T);
02892
02893     /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
02894         * m/A, then we will only need to deal with distances and sizes in
02895         * Angstroms rather than meters.                                    */
02896     xkappa = Vpbe_getXkappa(pbe);              /* A^{-1}        */
02897     pre1 = pre1*(1.0e10);
02898
02899     switch (thee->pmgp->bcfl) {
02900         /*  If we have zero boundary conditions, we're done */
02901         case BCFL_ZERO:
02902             return;
02903
02904             /*  For single DH sphere BC's, we only have one "atom" to deal with;
02905             *  get its information and */
02906         case BCFL_SDH:
02907             size = Vpbe_getSoluteRadius(pbe);
02908             position = Vpbe_getSoluteCenter(pbe);
02909
02910             /*
02911              For AMOEBA SDH boundary conditions, we need to find the
02912              total monopole, dipole and traceless quadrupole moments
02913              of either the permanent multipoles, induced dipoles or
02914              non-local induced dipoles.
02915              */
02916
02917             sdhcharge = 0.0;
02918             for (i=0; i<3; i++) sdhdipole[i] = 0.0;
02919             for (i=0; i<9; i++) sdhquadrupole[i] = 0.0;
02920
02921             for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
02922                 atom = Valist_getAtom(alist, iatom);
02923                 apos = Vatom_getPosition(atom);
02924                 xr = apos[0] - position[0];
02925                 yr = apos[1] - position[1];
02926                 zr = apos[2] - position[2];
```

```
02927                 switch (thee->chargeSrc) {
02928                     case VCM_CHARGE:
02929                         charge = Vatom_getCharge(atom);
02930                         sdhcharge += charge;
02931                         sdhdipole[0] += xr * charge;
02932                         sdhdipole[1] += yr * charge;
02933                         sdhdipole[2] += zr * charge;
02934                         traced[0] = xr*xr*charge;
02935                         traced[1] = xr*yr*charge;
02936                         traced[2] = xr*zr*charge;
02937                         traced[3] = yr*xr*charge;
02938                         traced[4] = yr*yr*charge;
02939                         traced[5] = yr*zr*charge;
02940                         traced[6] = zr*xr*charge;
02941                         traced[7] = zr*yr*charge;
02942                         traced[8] = zr*zr*charge;
02943                         qave = (traced[0] + traced[4] + traced[8]) / 3.0;
02944                         sdhquadrupole[0] += 1.5*(traced[0] - qave);
02945                         sdhquadrupole[1] += 1.5*(traced[1]);
02946                         sdhquadrupole[2] += 1.5*(traced[2]);
02947                         sdhquadrupole[3] += 1.5*(traced[3]);
02948                         sdhquadrupole[4] += 1.5*(traced[4] - qave);
02949                         sdhquadrupole[5] += 1.5*(traced[5]);
02950                         sdhquadrupole[6] += 1.5*(traced[6]);
02951                         sdhquadrupole[7] += 1.5*(traced[7]);
02952                         sdhquadrupole[8] += 1.5*(traced[8] - qave);
02953 #if defined(WITH_TINKER)
02954                     case VCM_PERMANENT:
02955                         charge = Vatom_getCharge(atom);
02956                         dipole = Vatom_getDipole(atom);
02957                         quadrupole = Vatom_getQuadrupole(atom);
02958                         sdhcharge += charge;
02959                         sdhdipole[0] += xr * charge;
02960                         sdhdipole[1] += yr * charge;
02961                         sdhdipole[2] += zr * charge;
02962                         traced[0] = xr*xr*charge;
02963                         traced[1] = xr*yr*charge;
02964                         traced[2] = xr*zr*charge;
02965                         traced[3] = yr*xr*charge;
02966                         traced[4] = yr*yr*charge;
02967                         traced[5] = yr*zr*charge;
02968                         traced[6] = zr*xr*charge;
02969                         traced[7] = zr*yr*charge;
02970                         traced[8] = zr*zr*charge;
02971                         sdhdipole[0] += dipole[0];
02972                         sdhdipole[1] += dipole[1];
02973                         sdhdipole[2] += dipole[2];
02974                         traced[0] += 2.0*xr*dipole[0];
02975                         traced[1] += xr*dipole[1] + yr*dipole[0];
02976                         traced[2] += xr*dipole[2] + zr*dipole[0];
02977                         traced[3] += yr*dipole[0] + xr*dipole[1];
02978                         traced[4] += 2.0*yr*dipole[1];
02979                         traced[5] += yr*dipole[2] + zr*dipole[1];
02980                         traced[6] += zr*dipole[0] + xr*dipole[2];
02981                         traced[7] += zr*dipole[1] + yr*dipole[2];
02982                         traced[8] += 2.0*zr*dipole[2];
02983                         qave = (traced[0] + traced[4] + traced[8]) / 3.0;
02984                         sdhquadrupole[0] += 1.5*(traced[0] - qave);
02985                         sdhquadrupole[1] += 1.5*(traced[1]);
02986                         sdhquadrupole[2] += 1.5*(traced[2]);
02987                         sdhquadrupole[3] += 1.5*(traced[3]);
02988                         sdhquadrupole[4] += 1.5*(traced[4] - qave);
02989                         sdhquadrupole[5] += 1.5*(traced[5]);
02990                         sdhquadrupole[6] += 1.5*(traced[6]);
02991                         sdhquadrupole[7] += 1.5*(traced[7]);
02992                         sdhquadrupole[8] += 1.5*(traced[8] - qave);
02993                         sdhquadrupole[0] += quadrupole[0];
02994                         sdhquadrupole[1] += quadrupole[1];
02995                         sdhquadrupole[2] += quadrupole[2];
02996                         sdhquadrupole[3] += quadrupole[3];
02997                         sdhquadrupole[4] += quadrupole[4];
02998                         sdhquadrupole[5] += quadrupole[5];
02999                         sdhquadrupole[6] += quadrupole[6];
03000                         sdhquadrupole[7] += quadrupole[7];
03001                         sdhquadrupole[8] += quadrupole[8];
03002                     case VCM_INDUCED:
03003                         dipole = Vatom_getInducedDipole(atom);
03004                         sdhdipole[0] += dipole[0];
03005                         sdhdipole[1] += dipole[1];
03006                         sdhdipole[2] += dipole[2];
03007                         traced[0] = 2.0*xr*dipole[0];
```

```
03008                       traced[1] = xr*dipole[1] + yr*dipole[0];
03009                       traced[2] = xr*dipole[2] + zr*dipole[0];
03010                       traced[3] = yr*dipole[0] + xr*dipole[1];
03011                       traced[4] = 2.0*yr*dipole[1];
03012                       traced[5] = yr*dipole[2] + zr*dipole[1];
03013                       traced[6] = zr*dipole[0] + xr*dipole[2];
03014                       traced[7] = zr*dipole[1] + yr*dipole[2];
03015                       traced[8] = 2.0*zr*dipole[2];
03016                       qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03017                       sdhquadrupole[0] += 1.5*(traced[0] - qave);
03018                       sdhquadrupole[1] += 1.5*(traced[1]);
03019                       sdhquadrupole[2] += 1.5*(traced[2]);
03020                       sdhquadrupole[3] += 1.5*(traced[3]);
03021                       sdhquadrupole[4] += 1.5*(traced[4] - qave);
03022                       sdhquadrupole[5] += 1.5*(traced[5]);
03023                       sdhquadrupole[6] += 1.5*(traced[6]);
03024                       sdhquadrupole[7] += 1.5*(traced[7]);
03025                       sdhquadrupole[8] += 1.5*(traced[8] - qave);
03026                   case VCM_NLINDUCED:
03027                       dipole = Vatom_getNLInducedDipole(atom);
03028                       sdhdipole[0] += dipole[0];
03029                       sdhdipole[1] += dipole[1];
03030                       sdhdipole[2] += dipole[2];
03031                       traced[0] = 2.0*xr*dipole[0];
03032                       traced[1] = xr*dipole[1] + yr*dipole[0];
03033                       traced[2] = xr*dipole[2] + zr*dipole[0];
03034                       traced[3] = yr*dipole[0] + xr*dipole[1];
03035                       traced[4] = 2.0*yr*dipole[1];
03036                       traced[5] = yr*dipole[2] + zr*dipole[1];
03037                       traced[6] = zr*dipole[0] + xr*dipole[2];
03038                       traced[7] = zr*dipole[1] + yr*dipole[2];
03039                       traced[8] = 2.0*zr*dipole[2];
03040                       qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03041                       sdhquadrupole[0] += 1.5*(traced[0] - qave);
03042                       sdhquadrupole[1] += 1.5*(traced[1]);
03043                       sdhquadrupole[2] += 1.5*(traced[2]);
03044                       sdhquadrupole[3] += 1.5*(traced[3]);
03045                       sdhquadrupole[4] += 1.5*(traced[4] - qave);
03046                       sdhquadrupole[5] += 1.5*(traced[5]);
03047                       sdhquadrupole[6] += 1.5*(traced[6]);
03048                       sdhquadrupole[7] += 1.5*(traced[7]);
03049                       sdhquadrupole[8] += 1.5*(traced[8] - qave);
03050                       /*Added the else to kill a warning when building with clang*/
03051 #else
03052                   case VCM_PERMANENT:;
03053                   case VCM_INDUCED:;
03054                   case VCM_NLINDUCED:;
03055 #endif /* if defined(WITH_TINKER) */
03056                   }
03057               }
03058               /* SDH dipole and traceless quadrupole values
03059                were checked against similar routines in TINKER
03060                for large proteins.
03061
03062                debye=4.8033324;
03063                printf("%6.3f, %6.3f, %6.3f\n", sdhdipole[0]*debye,
03064                sdhdipole[1]*debye, sdhdipole[2]*debye);
03065                printf("%6.3f\n", sdhquadrupole[0]*debye);
03066                printf("%6.3f %6.3f\n", sdhquadrupole[3]*debye,
03067                sdhquadrupole[4]*debye);
03068                printf("%6.3f %6.3f %6.3f\n", sdhquadrupole[6]*debye,
03069                sdhquadrupole[7]*debye, sdhquadrupole[8]*debye);
03070                */
03071
03072               bcfl2(size, position, sdhcharge, sdhdipole, sdhquadrupole,
03073                     xkappa, eps_p, eps_w, T, thee->gxcf, thee->gycf,
03074                     thee->gzcf, thee->xf, thee->yf, thee->zf, nx, ny, nz);
03075               break;
03076
03077           case BCFL_MDH:
03078               for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
03079                   atom = Valist_getAtom(alist, iatom);
03080                   position = Vatom_getPosition(atom);
03081                   charge = Vunit_ec*Vatom_getCharge(atom);
03082                   dipole = VNULL;
03083                   quadrupole = VNULL;
03084                   size = Vatom_getRadius(atom);
03085                   switch (thee->chargeSrc)
03086                   {
03087                       case VCM_CHARGE:
03088                           ;
```

```
03089 #if  defined(WITH_TINKER)
03090                       case VCM_PERMANENT:
03091                           dipole = Vatom_getDipole(atom);
03092                           quadrupole = Vatom_getQuadrupole(atom);
03093 
03094                       case VCM_INDUCED:
03095                           dipole = Vatom_getInducedDipole(atom);
03096 
03097                       case VCM_NLINDUCED:
03098                           dipole = Vatom_getNLInducedDipole(atom);
03099 /*added this to kill a warning when building with clang (by Juan Brandi).*/
03100 #else
03101                       case VCM_PERMANENT:;
03102                       case VCM_INDUCED:;
03103                       case VCM_NLINDUCED:;
03104 #endif
03105                   }
03106                   bcfl1(size, position, charge, xkappa, pre1,
03107                         thee->gxcf, thee->gycf, thee->gzcf,
03108                         thee->xf, thee->yf, thee->zf, nx, ny, nz);
03109               }
03110               break;
03111 
03112           case BCFL_UNUSED:
03113               Vnm_print(2, "bcCalc:  Invalid bcfl (%d)!\n", thee->pmgp->bcfl);
03114               VASSERT(0);
03115 
03116           case BCFL_FOCUS:
03117               Vnm_print(2, "VPMG::bcCalc -- not appropriate for focusing!\n");
03118               VASSERT(0);
03119 
03120           default:
03121               Vnm_print(2, "VPMG::bcCalc -- invalid boundary condition \
03122 flag (%d)!\n", thee->pmgp->bcfl);
03123               VASSERT(0);
03124       }
03125 }
03126 
03127 /*
03128  Used by bcflnew
03129  */
03130 VPRIVATE int gridPointIsValid(int i, int j, int k, int nx, int ny, int nz){
03131 
03132     int isValid = 0;
03133 
03134     if((k==0) || (k==nz-1)){
03135         isValid = 1;
03136     }else if((j==0) || (j==ny-1)){
03137         isValid = 1;
03138     }else if((i==0) || (i==nx-1)){
03139         isValid = 1;
03140     }
03141 
03142     return isValid;
03143 }
03144 
03145 /*
03146  Used by bcflnew
03147  */
03148 #ifdef DEBUG_MAC_OSX_OCL
03149 #include "mach_chud.h"
03150 VPRIVATE void packAtomsOpenCL(float *ax, float *ay, float *az,
03151                           float *charge, float *size, Vpmg *thee){
03152 
03153     int i;
03154     int natoms;
03155 
03156     Vatom *atom = VNULL;
03157     Valist *alist = VNULL;
03158 
03159     alist = thee->pbe->alist;
03160     natoms = Valist_getNumberAtoms(alist);
03161 
03162     for(i=0;i<natoms;i++){
03163         atom = &(alist->atoms[i]);
03164         charge[i] = Vunit_ec*atom->charge;
03165         ax[i] = atom->position[0];
03166         ay[i] = atom->position[1];
03167         az[i] = atom->position[2];
03168         size[i] = atom->radius;
03169     }
```

```
03170 }
03171
03172 /*
03173  Used by bcflnew
03174  */
03175 VPRIVATE void packUnpackOpenCL(int nx, int ny, int nz, int ngrid,
03176                           float *gx, float *gy, float *gz, float *value,
03177                           Vpmg *thee, int pack){
03178
03179     int i,j,k,igrid;
03180     int x0,x1,y0,y1,z0,z1;
03181
03182     float gpos[3];
03183     double *xf, *yf, *zf;
03184     double *gxcf, *gycf, *gzcf;
03185
03186     xf = thee->xf;
03187     yf = thee->yf;
03188     zf = thee->zf;
03189
03190     gxcf = thee->gxcf;
03191     gycf = thee->gycf;
03192     gzcf = thee->gzcf;
03193
03194     igrid = 0;
03195     for(k=0;k<nz;k++){
03196         gpos[2] = zf[k];
03197         for(j=0;j<ny;j++){
03198             gpos[1] = yf[j];
03199             for(i=0;i<nx;i++){
03200                 gpos[0] = xf[i];
03201                 if(gridPointIsValid(i, j, k, nx, ny, nz)){
03202                     if(pack != 0){
03203                         gx[igrid] = gpos[0];
03204                         gy[igrid] = gpos[1];
03205                         gz[igrid] = gpos[2];
03206
03207                         value[igrid] = 0.0;
03208                     }else{
03209                         x0 = IJKx(j,k,0);
03210                         x1 = IJKx(j,k,1);
03211                         y0 = IJKy(i,k,0);
03212                         y1 = IJKy(i,k,1);
03213                         z0 = IJKz(i,j,0);
03214                         z1 = IJKz(i,j,1);
03215
03216                         if(i==0){
03217                             gxcf[x0] += value[igrid];
03218                         }
03219                         if(i==nx-1){
03220                             gxcf[x1] += value[igrid];
03221                         }
03222                         if(j==0){
03223                             gycf[y0] += value[igrid];
03224                         }
03225                         if(j==ny-1){
03226                             gycf[y1] += value[igrid];
03227                         }
03228                         if(k==0){
03229                             gzcf[z0] += value[igrid];
03230                         }
03231                         if(k==nz-1){
03232                             gzcf[z1] += value[igrid];
03233                         }
03234                     }
03235
03236                     igrid++;
03237                 } //end is valid point
03238             } //end i
03239         } //end j
03240     } //end k
03241
03242 }
03243
03244 /*
03245  bcflnew is an optimized replacement for bcfl1. bcfl1 is still used when TINKER
03246  support is compiled in.
03247  bcflnew uses: packUnpack, packAtoms, gridPointIsValid
03248  */
03249 VPRIVATE void bcflnewOpenCL(Vpmg *thee){
03250
```

```
03251     int i,j,k, iatom, igrid;
03252     int x0, x1, y0, y1, z0, z1;
03253
03254     int nx, ny, nz;
03255     int natoms, ngrid, ngadj;
03256
03257     float dist, pre1, eps_w, eps_p, T, xkappa;
03258
03259     float *ax, *ay, *az;
03260     float *charge, *size, *val;
03261
03262     float *gx, *gy, *gz;
03263
03264     Vpbe *pbe = thee->pbe;
03265
03266     nx = thee->pmgp->nx;
03267     ny = thee->pmgp->ny;
03268     nz = thee->pmgp->nz;
03269
03270     eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
03271     eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
03272     T = Vpbe_getTemperature(pbe);               /* K             */
03273     pre1 = ((Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T))*(1.0e10);
03274     xkappa = Vpbe_getXkappa(pbe);
03275
03276     natoms = Valist_getNumberAtoms(thee->pbe->alist);
03277     ngrid = 2*((nx*ny) + (ny*nz) + (nx*nz));
03278     ngadj = ngrid + (512 - (ngrid & 511));
03279
03280     ax = (float*)malloc(natoms * sizeof(float));
03281     ay = (float*)malloc(natoms * sizeof(float));
03282     az = (float*)malloc(natoms * sizeof(float));
03283
03284     charge = (float*)malloc(natoms * sizeof(float));
03285     size = (float*)malloc(natoms * sizeof(float));
03286
03287     gx = (float*)malloc(ngrid * sizeof(float));
03288     gy = (float*)malloc(ngrid * sizeof(float));
03289     gz = (float*)malloc(ngrid * sizeof(float));
03290
03291     val = (float*)malloc(ngrid * sizeof(float));
03292
03293     packAtomsOpenCL(ax,ay,az,charge,size,thee);
03294     packUnpackOpenCL(nx,ny,nz,ngrid,gx,gy,gz,val,thee,1);
03295
03296     runMDHCL(ngrid,natoms,ngadj,ax,ay,az,gx,gy,gz,charge,size,xkappa,pre1,val);
03297
03298     packUnpackOpenCL(nx,ny,nz,ngrid,gx,gy,gz,val,thee,0);
03299
03300     free(ax);
03301     free(ay);
03302     free(az);
03303     free(charge);
03304     free(size);
03305
03306     free(gx);
03307     free(gy);
03308     free(gz);
03309     free(val);
03310 }
03311 #endif
03312
03313 VPRIVATE void packAtoms(double *ax, double *ay, double *az,
03314                         double *charge, double *size, Vpmg *thee){
03315
03316     int i;
03317     int natoms;
03318
03319     Vatom *atom = VNULL;
03320     Valist *alist = VNULL;
03321
03322     alist = thee->pbe->alist;
03323     natoms = Valist_getNumberAtoms(alist);
03324
03325     for(i=0;i<natoms;i++){
03326         atom = &(alist->atoms[i]);
03327         charge[i] = Vunit_ec*atom->charge;
03328         ax[i] = atom->position[0];
03329         ay[i] = atom->position[1];
03330         az[i] = atom->position[2];
03331         size[i] = atom->radius;
```

```
03332     }
03333 }
03334
03335 /*
03336  Used by bcflnew
03337  */
03338 VPRIVATE void packUnpack(int nx, int ny, int nz, int ngrid,
03339                          double *gx, double *gy, double *gz, double *value,
03340                          Vpmg *thee, int pack){
03341
03342     int i,j,k,igrid;
03343     int x0,x1,y0,y1,z0,z1;
03344
03345     double gpos[3];
03346     double *xf, *yf, *zf;
03347     double *gxcf, *gycf, *gzcf;
03348
03349     xf = thee->xf;
03350     yf = thee->yf;
03351     zf = thee->zf;
03352
03353     gxcf = thee->gxcf;
03354     gycf = thee->gycf;
03355     gzcf = thee->gzcf;
03356
03357     igrid = 0;
03358     for(k=0;k<nz;k++){
03359         gpos[2] = zf[k];
03360         for(j=0;j<ny;j++){
03361             gpos[1] = yf[j];
03362             for(i=0;i<nx;i++){
03363                 gpos[0] = xf[i];
03364                 if(gridPointIsValid(i, j, k, nx, ny, nz)){
03365                     if(pack != 0){
03366                         gx[igrid] = gpos[0];
03367                         gy[igrid] = gpos[1];
03368                         gz[igrid] = gpos[2];
03369
03370                         value[igrid] = 0.0;
03371                     }else{
03372                         x0 = IJKx(j,k,0);
03373                         x1 = IJKx(j,k,1);
03374                         y0 = IJKy(i,k,0);
03375                         y1 = IJKy(i,k,1);
03376                         z0 = IJKz(i,j,0);
03377                         z1 = IJKz(i,j,1);
03378
03379                         if(i==0){
03380                             gxcf[x0] += value[igrid];
03381                         }
03382                         if(i==nx-1){
03383                             gxcf[x1] += value[igrid];
03384                         }
03385                         if(j==0){
03386                             gycf[y0] += value[igrid];
03387                         }
03388                         if(j==ny-1){
03389                             gycf[y1] += value[igrid];
03390                         }
03391                         if(k==0){
03392                             gzcf[z0] += value[igrid];
03393                         }
03394                         if(k==nz-1){
03395                             gzcf[z1] += value[igrid];
03396                         }
03397                     }
03398
03399                     igrid++;
03400                 } //end is valid point
03401             } //end i
03402         } //end j
03403     } //end k
03404
03405 }
03406
03407 VPRIVATE void bcflnew(Vpmg *thee){
03408
03409     int i,j,k, iatom, igrid;
03410     int x0, x1, y0, y1, z0, z1;
03411
03412     int nx, ny, nz;
```

```
03413     int natoms, ngrid;
03414
03415     double dist, pre1, eps_w, eps_p, T, xkappa;
03416
03417     double *ax, *ay, *az;
03418     double *charge, *size, *val;
03419
03420     double *gx, *gy, *gz;
03421
03422     Vpbe *pbe = thee->pbe;
03423
03424     nx = thee->pmgp->nx;
03425     ny = thee->pmgp->ny;
03426     nz = thee->pmgp->nz;
03427
03428     eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
03429     eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
03430     T = Vpbe_getTemperature(pbe);               /* K             */
03431     pre1 = ((Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T))*(1.0e10);
03432     xkappa = Vpbe_getXkappa(pbe);
03433
03434     natoms = Valist_getNumberAtoms(thee->pbe->alist);
03435     ngrid = 2*((nx*ny) + (ny*nz) + (nx*nz));
03436
03437     ax = (double*)malloc(natoms * sizeof(double));
03438     ay = (double*)malloc(natoms * sizeof(double));
03439     az = (double*)malloc(natoms * sizeof(double));
03440
03441     charge = (double*)malloc(natoms * sizeof(double));
03442     size = (double*)malloc(natoms * sizeof(double));
03443
03444     gx = (double*)malloc(ngrid * sizeof(double));
03445     gy = (double*)malloc(ngrid * sizeof(double));
03446     gz = (double*)malloc(ngrid * sizeof(double));
03447
03448     val = (double*)malloc(ngrid * sizeof(double));
03449
03450     packAtoms(ax,ay,az,charge,size,thee);
03451     packUnpack(nx,ny,nz,ngrid,gx,gy,gz,val,thee,1);
03452
03453     if(xkappa > VSMALL){
03454 #pragma omp parallel for default(shared) private(igrid,iatom,dist)
03455         for(igrid=0;igrid<ngrid;igrid++){
03456             for(iatom=0; iatom<natoms; iatom++){
03457                 dist = VSQRT(VSQR(gx[igrid]-ax[iatom]) + VSQR(gy[igrid]-ay[iatom])
03458                              + VSQR(gz[igrid]-az[iatom]));
03459                 val[igrid] += pre1*(charge[iatom]/dist)*VEXP(-xkappa*(dist-size[iatom]))
03460                 / (1+xkappa*size[iatom]);
03461             }
03462         }
03463     }else{
03464 #pragma omp parallel for default(shared) private(igrid,iatom,dist)
03465         for(igrid=0;igrid<ngrid;igrid++){
03466             for(iatom=0; iatom<natoms; iatom++){
03467                 dist = VSQRT(VSQR(gx[igrid]-ax[iatom]) + VSQR(gy[igrid]-ay[iatom])
03468                              + VSQR(gz[igrid]-az[iatom]));
03469                 val[igrid] += pre1*(charge[iatom]/dist);
03470             }
03471         }
03472     }
03473     packUnpack(nx,ny,nz,ngrid,gx,gy,gz,val,thee,0);
03474
03475     free(ax);
03476     free(ay);
03477     free(az);
03478     free(charge);
03479     free(size);
03480
03481     free(gx);
03482     free(gy);
03483     free(gz);
03484     free(val);
03485 }
03486
03487 VPRIVATE void multipolebc(double r, double kappa, double eps_p,
03488                           double eps_w, double rad, double tsr[3]) {
03489     double r2,r3,r5;
03490     double eps_r;
03491     double ka,ka2,ka3;
03492     double kr,kr2,kr3;
03493
```

```
03494     /*
03495      Below an attempt is made to explain the potential outside of a
03496      multipole located at the center of spherical cavity of dieletric
03497      eps_p, with dielectric eps_w outside (and possibly kappa > 0).
03498
03499
03500      First, eps_p = 1.0
03501      eps_w = 1.0
03502      kappa = 0.0
03503
03504      The general form for the potential of a traceless multipole tensor
03505      of rank n in vacuum is:
03506
03507      V(r) = (-1)^n * u . n . Del^n (1/r)
03508
03509      where
03510      u                    is a multipole of order n (3^n components)
03511      u . n. Del^n (1/r)    is the contraction of u with the nth
03512      derivative of 1/r
03513
03514      for example, if n = 1, the dipole potential is
03515      V_vac(r) = (-1)*[ux*x + uy*y + uz*z]/r^3
03516
03517      This function returns the parts of V(r) for multipoles of
03518      order 0, 1 and 2 that are independent of the contraction.
03519
03520      For the vacuum example, this would be 1/r, -1/r^3 and 3/r^5
03521      respectively.
03522
03523      *** Note that this requires that the quadrupole is
03524      traceless. If not, the diagonal quadrupole potential changes
03525      from
03526      qaa *  3*a^2/r^5
03527      to
03528      qaa * (3*a^2/r^5 - 1/r^3a )
03529      where we sum over the trace; a = x, y and z.
03530
03531      (In other words, the -1/r^3 term cancels for a traceless quadrupole.
03532      qxx + qyy + qzz = 0
03533      such that
03534      -(qxx + qyy + qzz)/r^3 = 0
03535
03536      For quadrupole with trace:
03537      qxx + qyy + qzz != 0
03538      such that
03539      -(qxx + qyy + qzz)/r^3 != 0
03540      )
03541
03542      ========================================================================
03543
03544      eps_p != 1 or eps_w != 1
03545      kappa = 0.0
03546
03547      If the multipole is placed at the center of a sphere with
03548      dieletric eps_p in a solvent of dielectric eps_w, the potential
03549      outside the sphere is the solution to the Laplace equation:
03550
03551      V(r) = 1/eps_w * (2*n+1)*eps_r/(n+(n+1)*eps_r)
03552      * (-1)^n * u . n . Del^n (1/r)
03553      where
03554      eps_r = eps_w / eps_p
03555      is the ratio of solvent to solute dielectric
03556
03557      ========================================================================
03558
03559      kappa > 0
03560
03561      Finally, if the region outside the sphere is treated by the linearized
03562      PB equation with Debye-Huckel parameter kappa, the solution is:
03563
03564      V(r) = kappa/eps_w * alpha_n(kappa*a) * K_n(kappa*r) * r^(n+1)/a^n
03565      * (-1)^n * u . n . Del^n (1/r)
03566      where
03567      alpha_n(x) is [(2n + 1) / x] / [(n*K_n(x)/eps_r) - x*K_n'(x)]
03568      K_n(x) are modified spherical Bessel functions of the third kind.
03569      K_n'(x) is the derivative of K_n(x)
03570      */
03571
03572     eps_r = eps_w/eps_p;
03573     r2 = r*r;
03574     r3 = r2*r;
```

```
03575     r5 = r3*r2;
03576     tsr[0] = (1.0/eps_w)/r;
03577     tsr[1] = (1.0/eps_w)*(-1.0)/r3;
03578     tsr[2] = (1.0/eps_w)*(3.0)/r5;
03579     if (kappa < VSMALL) {
03580         tsr[1] = (3.0*eps_r)/(1.0 + 2.0*eps_r)*tsr[1];
03581         tsr[2] = (5.0*eps_r)/(2.0 + 3.0*eps_r)*tsr[2];
03582     } else {
03583         ka = kappa*rad;
03584         ka2 = ka*ka;
03585         ka3 = ka2*ka;
03586         kr = kappa*r;
03587         kr2 = kr*kr;
03588         kr3 = kr2*kr;
03589         tsr[0] = exp(ka-kr) / (1.0 + ka) * tsr[0];
03590         tsr[1] = 3.0*eps_r*exp(ka-kr)*(1.0 + kr) * tsr[1];
03591         tsr[1] = tsr[1] / (1.0 + ka + eps_r*(2.0 + 2.0*ka + ka2));
03592         tsr[2] = 5.0*eps_r*exp(ka-kr)*(3.0 + 3.0*kr + kr2) * tsr[2];
03593         tsr[2] = tsr[2]/(6.0+6.0*ka+2.0*ka2+eps_r*(9.0+9.0*ka+4.0*ka2+ka3));
03594     }
03595 }
03596
03597 VPRIVATE void bcfl_sdh(Vpmg *thee){
03598
03599     int i,j,k,iatom;
03600     int nx, ny, nz;
03601
03602     double size, *position, charge, xkappa, eps_w, eps_p, T, pre, dist;
03603     double sdhcharge, sdhdipole[3], traced[9], sdhquadrupole[9];
03604     double *dipole, *quadrupole;
03605
03606     double val, *apos, gpos[3], tensor[3], qave;
03607     double ux, uy, uz, xr, yr, zr;
03608     double qxx,qxy,qxz,qyx,qyy,qyz,qzx,qzy,qzz;
03609
03610     double *xf, *yf, *zf;
03611     double *gxcf, *gycf, *gzcf;
03612
03613     Vpbe *pbe;
03614     Vatom *atom;
03615     Valist *alist;
03616
03617     pbe = thee->pbe;
03618     alist = thee->pbe->alist;
03619     nx = thee->pmgp->nx;
03620     ny = thee->pmgp->ny;
03621     nz = thee->pmgp->nz;
03622
03623     xf = thee->xf;
03624     yf = thee->yf;
03625     zf = thee->zf;
03626
03627     gxcf = thee->gxcf;
03628     gycf = thee->gycf;
03629     gzcf = thee->gzcf;
03630
03631     /* For each "atom" (only one for bcfl=1), we use the following formula to
03632      * calculate the boundary conditions:
03633      *    g(x) = \frac{q e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
03634      *          * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
03635      *          * 1/d
03636      * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
03637      * We only need to evaluate some of these prefactors once:
03638      *    pre1 = \frac{e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
03639      * which gives the potential as
03640      *    g(x) = pre1 * q/d * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
03641      */
03642     eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
03643     eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
03644     T = Vpbe_getTemperature(pbe);               /* K             */
03645
03646     pre = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*Vunit_kb*T);
03647     pre = pre*(1.0e10);
03648
03649     /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
03650      * m/A, then we will only need to deal with distances and sizes in
03651      * Angstroms rather than meters.                                    */
03652     xkappa = Vpbe_getXkappa(pbe);              /* A^{-1}        */
03653
03654     /* Solute size and position */
03655     size = Vpbe_getSoluteRadius(pbe);
```

```
03656     position = Vpbe_getSoluteCenter(pbe);
03657
03658     sdhcharge = 0.0;
03659     for (i=0; i<3; i++) sdhdipole[i] = 0.0;
03660     for (i=0; i<9; i++) sdhquadrupole[i] = 0.0;
03661
03662     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
03663         atom = Valist_getAtom(alist, iatom);
03664         apos = Vatom_getPosition(atom);
03665         xr = apos[0] - position[0];
03666         yr = apos[1] - position[1];
03667         zr = apos[2] - position[2];
03668         switch (thee->chargeSrc) {
03669             case VCM_CHARGE:
03670                 charge = Vatom_getCharge(atom);
03671                 sdhcharge += charge;
03672                 sdhdipole[0] += xr * charge;
03673                 sdhdipole[1] += yr * charge;
03674                 sdhdipole[2] += zr * charge;
03675                 traced[0] = xr*xr*charge;
03676                 traced[1] = xr*yr*charge;
03677                 traced[2] = xr*zr*charge;
03678                 traced[3] = yr*xr*charge;
03679                 traced[4] = yr*yr*charge;
03680                 traced[5] = yr*zr*charge;
03681                 traced[6] = zr*xr*charge;
03682                 traced[7] = zr*yr*charge;
03683                 traced[8] = zr*zr*charge;
03684                 qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03685                 sdhquadrupole[0] += 1.5*(traced[0] - qave);
03686                 sdhquadrupole[1] += 1.5*(traced[1]);
03687                 sdhquadrupole[2] += 1.5*(traced[2]);
03688                 sdhquadrupole[3] += 1.5*(traced[3]);
03689                 sdhquadrupole[4] += 1.5*(traced[4] - qave);
03690                 sdhquadrupole[5] += 1.5*(traced[5]);
03691                 sdhquadrupole[6] += 1.5*(traced[6]);
03692                 sdhquadrupole[7] += 1.5*(traced[7]);
03693                 sdhquadrupole[8] += 1.5*(traced[8] - qave);
03694 #if defined(WITH_TINKER)
03695             case VCM_PERMANENT:
03696                 charge = Vatom_getCharge(atom);
03697                 dipole = Vatom_getDipole(atom);
03698                 quadrupole = Vatom_getQuadrupole(atom);
03699                 sdhcharge += charge;
03700                 sdhdipole[0] += xr * charge;
03701                 sdhdipole[1] += yr * charge;
03702                 sdhdipole[2] += zr * charge;
03703                 traced[0] = xr*xr*charge;
03704                 traced[1] = xr*yr*charge;
03705                 traced[2] = xr*zr*charge;
03706                 traced[3] = yr*xr*charge;
03707                 traced[4] = yr*yr*charge;
03708                 traced[5] = yr*zr*charge;
03709                 traced[6] = zr*xr*charge;
03710                 traced[7] = zr*yr*charge;
03711                 traced[8] = zr*zr*charge;
03712                 sdhdipole[0] += dipole[0];
03713                 sdhdipole[1] += dipole[1];
03714                 sdhdipole[2] += dipole[2];
03715                 traced[0] += 2.0*xr*dipole[0];
03716                 traced[1] += xr*dipole[1] + yr*dipole[0];
03717                 traced[2] += xr*dipole[2] + zr*dipole[0];
03718                 traced[3] += yr*dipole[0] + xr*dipole[1];
03719                 traced[4] += 2.0*yr*dipole[1];
03720                 traced[5] += yr*dipole[2] + zr*dipole[1];
03721                 traced[6] += zr*dipole[0] + xr*dipole[2];
03722                 traced[7] += zr*dipole[1] + yr*dipole[2];
03723                 traced[8] += 2.0*zr*dipole[2];
03724                 qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03725                 sdhquadrupole[0] += 1.5*(traced[0] - qave);
03726                 sdhquadrupole[1] += 1.5*(traced[1]);
03727                 sdhquadrupole[2] += 1.5*(traced[2]);
03728                 sdhquadrupole[3] += 1.5*(traced[3]);
03729                 sdhquadrupole[4] += 1.5*(traced[4] - qave);
03730                 sdhquadrupole[5] += 1.5*(traced[5]);
03731                 sdhquadrupole[6] += 1.5*(traced[6]);
03732                 sdhquadrupole[7] += 1.5*(traced[7]);
03733                 sdhquadrupole[8] += 1.5*(traced[8] - qave);
03734                 sdhquadrupole[0] += quadrupole[0];
03735                 sdhquadrupole[1] += quadrupole[1];
03736                 sdhquadrupole[2] += quadrupole[2];
```

```
03737                 sdhquadrupole[3] += quadrupole[3];
03738                 sdhquadrupole[4] += quadrupole[4];
03739                 sdhquadrupole[5] += quadrupole[5];
03740                 sdhquadrupole[6] += quadrupole[6];
03741                 sdhquadrupole[7] += quadrupole[7];
03742                 sdhquadrupole[8] += quadrupole[8];
03743             case VCM_INDUCED:
03744                 dipole = Vatom_getInducedDipole(atom);
03745                 sdhdipole[0] += dipole[0];
03746                 sdhdipole[1] += dipole[1];
03747                 sdhdipole[2] += dipole[2];
03748                 traced[0] = 2.0*xr*dipole[0];
03749                 traced[1] = xr*dipole[1] + yr*dipole[0];
03750                 traced[2] = xr*dipole[2] + zr*dipole[0];
03751                 traced[3] = yr*dipole[0] + xr*dipole[1];
03752                 traced[4] = 2.0*yr*dipole[1];
03753                 traced[5] = yr*dipole[2] + zr*dipole[1];
03754                 traced[6] = zr*dipole[0] + xr*dipole[2];
03755                 traced[7] = zr*dipole[1] + yr*dipole[2];
03756                 traced[8] = 2.0*zr*dipole[2];
03757                 qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03758                 sdhquadrupole[0] += 1.5*(traced[0] - qave);
03759                 sdhquadrupole[1] += 1.5*(traced[1]);
03760                 sdhquadrupole[2] += 1.5*(traced[2]);
03761                 sdhquadrupole[3] += 1.5*(traced[3]);
03762                 sdhquadrupole[4] += 1.5*(traced[4] - qave);
03763                 sdhquadrupole[5] += 1.5*(traced[5]);
03764                 sdhquadrupole[6] += 1.5*(traced[6]);
03765                 sdhquadrupole[7] += 1.5*(traced[7]);
03766                 sdhquadrupole[8] += 1.5*(traced[8] - qave);
03767             case VCM_NLINDUCED:
03768                 dipole = Vatom_getNLInducedDipole(atom);
03769                 sdhdipole[0] += dipole[0];
03770                 sdhdipole[1] += dipole[1];
03771                 sdhdipole[2] += dipole[2];
03772                 traced[0] = 2.0*xr*dipole[0];
03773                 traced[1] = xr*dipole[1] + yr*dipole[0];
03774                 traced[2] = xr*dipole[2] + zr*dipole[0];
03775                 traced[3] = yr*dipole[0] + xr*dipole[1];
03776                 traced[4] = 2.0*yr*dipole[1];
03777                 traced[5] = yr*dipole[2] + zr*dipole[1];
03778                 traced[6] = zr*dipole[0] + xr*dipole[2];
03779                 traced[7] = zr*dipole[1] + yr*dipole[2];
03780                 traced[8] = 2.0*zr*dipole[2];
03781                 qave = (traced[0] + traced[4] + traced[8]) / 3.0;
03782                 sdhquadrupole[0] += 1.5*(traced[0] - qave);
03783                 sdhquadrupole[1] += 1.5*(traced[1]);
03784                 sdhquadrupole[2] += 1.5*(traced[2]);
03785                 sdhquadrupole[3] += 1.5*(traced[3]);
03786                 sdhquadrupole[4] += 1.5*(traced[4] - qave);
03787                 sdhquadrupole[5] += 1.5*(traced[5]);
03788                 sdhquadrupole[6] += 1.5*(traced[6]);
03789                 sdhquadrupole[7] += 1.5*(traced[7]);
03790                 sdhquadrupole[8] += 1.5*(traced[8] - qave);
03791 /*added this to kill a warning when building with clang (by Juan Brandi)*/
03792 #else
03793                     case VCM_PERMANENT:;
03794                     case VCM_INDUCED:;
03795                     case VCM_NLINDUCED:;
03796 #endif /* if defined(WITH_TINKER) */
03797         }
03798     }
03799
03800     ux = sdhdipole[0];
03801     uy = sdhdipole[1];
03802     uz = sdhdipole[2];
03803
03804     /* The factor of 1/3 results from using a
03805      traceless quadrupole definition. See, for example,
03806      "The Theory of Intermolecular Forces" by A.J. Stone,
03807      Chapter 3. */
03808     qxx = sdhquadrupole[0] / 3.0;
03809     qxy = sdhquadrupole[1] / 3.0;
03810     qxz = sdhquadrupole[2] / 3.0;
03811     qyx = sdhquadrupole[3] / 3.0;
03812     qyy = sdhquadrupole[4] / 3.0;
03813     qyz = sdhquadrupole[5] / 3.0;
03814     qzx = sdhquadrupole[6] / 3.0;
03815     qzy = sdhquadrupole[7] / 3.0;
03816     qzz = sdhquadrupole[8] / 3.0;
03817
```

```
03818     for(k=0;k<nz;k++){
03819         gpos[2] = zf[k];
03820         for(j=0;j<ny;j++){
03821             gpos[1] = yf[j];
03822             for(i=0;i<nx;i++){
03823                 gpos[0] = xf[i];
03824                 if(gridPointIsValid(i, j, k, nx, ny, nz)){
03825                     xr = gpos[0] - position[0];
03826                     yr = gpos[1] - position[1];
03827                     zr = gpos[2] - position[2];
03828
03829                     dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
03830                     multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
03831
03832                     val = pre*sdhcharge*tensor[0];
03833                     val -= pre*ux*xr*tensor[1];
03834                     val -= pre*uy*yr*tensor[1];
03835                     val -= pre*uz*zr*tensor[1];
03836                     val += pre*qxx*xr*xr*tensor[2];
03837                     val += pre*qyy*yr*yr*tensor[2];
03838                     val += pre*qzz*zr*zr*tensor[2];
03839                     val += pre*2.0*qxy*xr*yr*tensor[2];
03840                     val += pre*2.0*qxz*xr*zr*tensor[2];
03841                     val += pre*2.0*qyz*yr*zr*tensor[2];
03842
03843                     if(i==0){
03844                         gxcf[IJKx(j,k,0)] = val;
03845                     }
03846                     if(i==nx-1){
03847                         gxcf[IJKx(j,k,1)] = val;
03848                     }
03849                     if(j==0){
03850                         gycf[IJKy(i,k,0)] = val;
03851                     }
03852                     if(j==ny-1){
03853                         gycf[IJKy(i,k,1)] = val;
03854                     }
03855                     if(k==0){
03856                         gzcf[IJKz(i,j,0)] = val;
03857                     }
03858                     if(k==nz-1){
03859                         gzcf[IJKz(i,j,1)] = val;
03860                     }
03861                 } /* End grid point is valid */
03862             } /* End i loop */
03863         } /* End j loop */
03864     } /* End k loop */
03865
03866 }
03867
03868 VPRIVATE void bcfl_mdh(Vpmg *thee){
03869
03870     int i,j,k,iatom;
03871     int nx, ny, nz;
03872
03873     double val, *apos, gpos[3];
03874     double *dipole, *quadrupole;
03875     double size, charge, xkappa, eps_w, eps_p, T, pre1, dist;
03876
03877     double *xf, *yf, *zf;
03878     double *gxcf, *gycf, *gzcf;
03879
03880     Vpbe *pbe;
03881     Vatom *atom;
03882     Valist *alist;
03883
03884     pbe = thee->pbe;
03885     alist = thee->pbe->alist;
03886     nx = thee->pmgp->nx;
03887     ny = thee->pmgp->ny;
03888     nz = thee->pmgp->nz;
03889
03890     xf = thee->xf;
03891     yf = thee->yf;
03892     zf = thee->zf;
03893
03894     gxcf = thee->gxcf;
03895     gycf = thee->gycf;
03896     gzcf = thee->gzcf;
03897
03898     /* For each "atom" (only one for bcfl=1), we use the following formula to
```

```
03899      * calculate the boundary conditions:
03900      *    g(x) = \frac{q e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
03901      *          * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
03902      *          * 1/d
03903      * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
03904      * We only need to evaluate some of these prefactors once:
03905      *    pre1 = \frac{e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
03906      * which gives the potential as
03907      *    g(x) = pre1 * q/d * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
03908      */
03909     eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
03910     eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
03911     T = Vpbe_getTemperature(pbe);               /* K             */
03912     pre1 = (Vunit_ec)/(4*VPI*Vunit_eps0*eps_w*Vunit_kb*T);
03913
03914     /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
03915      * m/A, then we will only need to deal with distances and sizes in
03916      * Angstroms rather than meters.                                    */
03917     xkappa = Vpbe_getXkappa(pbe);              /* A^{-1}        */
03918     pre1 = pre1*(1.0e10);
03919
03920     /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
03921      * m/A, then we will only need to deal with distances and sizes in
03922      * Angstroms rather than meters.                                    */
03923     xkappa = Vpbe_getXkappa(pbe);              /* A^{-1}        */
03924
03925     for(k=0;k<nz;k++){
03926         gpos[2] = zf[k];
03927         for(j=0;j<ny;j++){
03928             gpos[1] = yf[j];
03929             for(i=0;i<nx;i++){
03930                 gpos[0] = xf[i];
03931                 if(gridPointIsValid(i, j, k, nx, ny, nz)){
03932
03933                     val = 0.0;
03934
03935                     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
03936                         atom = Valist_getAtom(alist, iatom);
03937                         apos = Vatom_getPosition(atom);
03938                         charge = Vunit_ec*Vatom_getCharge(atom);
03939                         size = Vatom_getRadius(atom);
03940
03941                         dist = VSQRT(VSQR(gpos[0]-apos[0]) + VSQR(gpos[1]-apos[1])
03942                                      + VSQR(gpos[2]-apos[2]));
03943                         if (xkappa > VSMALL) {
03944                             val += pre1*(charge/dist)*VEXP(-xkappa*(dist-size))
03945                             / (1+xkappa*size);
03946                         } else {
03947                             val += pre1*(charge/dist);
03948                         }
03949
03950                     }
03951
03952                     if(i==0){
03953                         gxcf[IJKx(j,k,0)] = val;
03954                     }
03955                     if(i==nx-1){
03956                         gxcf[IJKx(j,k,1)] = val;
03957                     }
03958                     if(j==0){
03959                         gycf[IJKy(i,k,0)] = val;
03960                     }
03961                     if(j==ny-1){
03962                         gycf[IJKy(i,k,1)] = val;
03963                     }
03964                     if(k==0){
03965                         gzcf[IJKz(i,j,0)] = val;
03966                     }
03967                     if(k==nz-1){
03968                         gzcf[IJKz(i,j,1)] = val;
03969                     }
03970                 } /* End grid point is valid */
03971             } /* End i loop */
03972         } /* End j loop */
03973     } /* End k loop */
03974
03975 }
03976
03977 /* ////////////////////////////////////////////////////////////////////////// */
03978  // Routine:  bcfl_mem
03979  //
```

```
03980  // Purpose:  Increment all the boundary points by the
03981  //           analytic expression for a membrane system in
03982  //           the presence of a membrane potential. This
03983  //           Boundary flag should only be used for systems
03984  //           that explicitly have membranes in the dielectric
03985  //           and solvent maps.
03986  //
03987  //           There should be several input variables add to this
03988  //           function such as membrane potential, membrane thickness
03989  //           and height.
03990  //
03991  // Args:     apos is a 3-vector
03992  //
03993  // Author: Michael Grabe
03995 VPRIVATE void bcfl_mem(double zmem, double L, double eps_m, double eps_w,
03996                        double V, double xkappa, double *gxcf, double *gycf, double *gzcf,
03997                        double *xf, double *yf, double *zf, int nx, int ny, int nz) {
03998
04000     /* some definitions                                  */
04001     /* L = total length of the membrane                  */
04002     /* xkappa = inverse Debeye length                    */
04003     /* zmem = z value of membrane bottom (Cytoplasm)     */
04004     /* V = electrical potential inside the cell          */
04006     int i, j, k;
04007     double dist, val, z_low, z_high, z_shift;
04008     double A, B, C, D, edge_L, l;
04009     double G, z_0, z_rel;
04010     double gpos[3];
04011
04012     Vnm_print(0, "Here is the value of kappa: %f\n",xkappa);
04013     Vnm_print(0, "Here is the value of L: %f\n",L);
04014     Vnm_print(0, "Here is the value of zmem: %f\n",zmem);
04015     Vnm_print(0, "Here is the value of mdie: %f\n",eps_m);
04016     Vnm_print(0, "Here is the value of memv: %f\n",V);
04017
04018     /* no salt symmetric BC's at +/- infinity */
04019     // B=V/(edge_L - l*(1-eps_w/eps_m));
04020     // A=V + B*edge_L;
04021     // D=eps_w/eps_m*B;
04022     z_low = zmem;     /* this defines the bottom of the membrane */
04023     z_high = zmem + L;  /* this is the top of the membrane */
04024
04025     /*******************************************************/
04026     /* proper boundary conditions for V = 0 extracellular */
04027     /* and psi=-V cytoplasm.                               */
04028     /* Implicit in this formulation is that the membrane  */
04029     /* center be at z = 0                                  */
04030     /*******************************************************/
04031
04032     l=L/2;                    /* half of the membrane length */
04033     z_0 = z_low + l;          /* center of the membrane      */
04034     G=l*eps_w/eps_m*xkappa;
04035     A=-V/2*(1/(G+1))*exp(xkappa*l);
04036     B=V/2;
04037     C=-V/2*eps_w/eps_m*xkappa*(1/(G+1));
04038     D=-A;
04039     /* The analytic expression for the boundary conditions      */
04040     /* had the cytoplasmic surface of the membrane set to zero. */
04041     /* This requires an off-set of the BC equations.            */
04042
04043     /* the "i" boundaries (dirichlet) */
04044     for (k=0; k<nz; k++) {
04045         gpos[2] = zf[k];
04046         z_rel = gpos[2] - z_0;    /* relative position for BCs */
04047
04048         for (j=0; j<ny; j++) {
04049
04050             if (gpos[2] <= z_low) {                      /* cytoplasmic */
04051
04052                 val = A*exp(xkappa*z_rel) + V;
04053                 gxcf[IJKx(j,k,0)] += val;    /* assign low side BC */
04054                 gxcf[IJKx(j,k,1)] += val;    /* assign high side BC */
04055
04056             }
04057
04058             else if (gpos[2] > z_low && gpos[2] <= z_high) {  /* in membrane */
04059
04060                 val = B + C*z_rel;
04061                 gxcf[IJKx(j,k,0)] += val;    /* assign low side BC */
04062                 gxcf[IJKx(j,k,1)] += val;    /* assign high side BC */
04063
```

```
04064             }
04065
04066             else if (gpos[2] > z_high)  {                  /* extracellular */
04067
04068                 val = D*exp(-xkappa*z_rel);
04069                 gxcf[IJKx(j,k,0)] += val;    /* assign low side BC */
04070                 gxcf[IJKx(j,k,1)] += val;    /* assign high side BC */
04071
04072             }
04073
04074         }
04075     }
04076
04077     /* the "j" boundaries (dirichlet) */
04078     for (k=0; k<nz; k++) {
04079         gpos[2] = zf[k];
04080         z_rel = gpos[2] - z_0;
04081         for (i=0; i<nx; i++) {
04082
04083             if (gpos[2] <= z_low) {                       /* cytoplasmic */
04084
04085                 val = A*exp(xkappa*z_rel) + V;
04086                 gycf[IJKy(i,k,0)] += val;    /* assign low side BC */
04087                 gycf[IJKy(i,k,1)] += val;    /* assign high side BC */
04088                 //printf("%f \n",val);
04089
04090             }
04091
04092             else if (gpos[2] > z_low && gpos[2] <= z_high) {  /* in membrane */
04093
04094                 val = B + C*z_rel;
04095                 gycf[IJKy(i,k,0)] += val;    /* assign low side BC */
04096                 gycf[IJKy(i,k,1)] += val;    /* assign high side BC */
04097                 //printf("%f \n",val);
04098
04099             }
04100             else if (gpos[2] > z_high)  {                  /* extracellular */
04101
04102                 val = D*exp(-xkappa*z_rel);
04103                 gycf[IJKy(i,k,0)] += val;    /* assign low side BC */
04104                 gycf[IJKy(i,k,1)] += val;    /* assign high side BC */
04105                 //printf("%f \n",val);
04106
04107             }
04108
04109         }
04110     }
04111
04112     /* the "k" boundaries (dirichlet) */
04113     for (j=0; j<ny; j++) {
04114         for (i=0; i<nx; i++) {
04115
04116             /* first assign the bottom boundary */
04117
04118             gpos[2] = zf[0];
04119             z_rel = gpos[2] - z_0;
04120
04121             if (gpos[2] <= z_low) {                       /* cytoplasmic */
04122
04123                 val = A*exp(xkappa*z_rel) + V;
04124                 gzcf[IJKz(i,j,0)] += val;    /* assign low side BC */
04125                 //printf("%f \n",val);
04126
04127             }
04128
04129             else if (gpos[2] > z_low && gpos[2] <= z_high) {  /* in membrane */
04130
04131                 val = B + C*z_rel;
04132                 gzcf[IJKz(i,j,0)] += val;    /* assign low side BC */
04133
04134             }
04135
04136             else if (gpos[2] > z_high)  {                  /* extracellular */
04137
04138                 val = D*exp(-xkappa*z_rel);
04139                 gzcf[IJKz(i,j,0)] += val;    /* assign low side BC */
04140
04141             }
04142
04143             /* now assign the top boundary */
04144
```

```
04145             gpos[2] = zf[nz-1];
04146             z_rel = gpos[2] - z_0;
04147 
04148             if (gpos[2] <= z_low) {                       /* cytoplasmic */
04149 
04150                 val = A*exp(xkappa*z_rel) + V;
04151                 gzcf[IJKz(i,j,1)] += val;    /* assign high side BC */
04152 
04153             }
04154 
04155             else if (gpos[2] > z_low && gpos[2] <= z_high) {  /* in membrane */
04156 
04157                 val = B + C*z_rel;
04158                 gzcf[IJKz(i,j,1)] += val;    /* assign high side BC */
04159 
04160             }
04161 
04162             else if (gpos[2] > z_high)  {                  /* extracellular */
04163 
04164                 val = D*exp(-xkappa*z_rel);
04165                 gzcf[IJKz(i,j,1)] += val;    /* assign high side BC */
04166                 //printf("%f \n",val);
04167 
04168             }
04169 
04170         }
04171     }
04172 }
04173 
04174 VPRIVATE void bcfl_map(Vpmg *thee){
04175 
04176     Vpbe *pbe;
04177     double position[3], pot, hx, hy, hzed;
04178     int i, j, k, nx, ny, nz, rc;
04179 
04180 
04181     VASSERT(thee != VNULL);
04182 
04183     /* Mesh info */
04184     nx = thee->pmgp->nx;
04185     ny = thee->pmgp->ny;
04186     nz = thee->pmgp->nz;
04187     hx = thee->pmgp->hx;
04188     hy = thee->pmgp->hy;
04189     hzed = thee->pmgp->hzed;
04190 
04191     /* Reset the potential array */
04192     for (i=0; i<(nx*ny*nz); i++) thee->pot[i] = 0.0;
04193 
04194     /* Fill in the source term (atomic potentials) */
04195     Vnm_print(0, "Vpmg_fillco:  filling in source term.\n");
04196     for (k=0; k<nz; k++) {
04197         for (j=0; j<ny; j++) {
04198             for (i=0; i<nx; i++) {
04199                 position[0] = thee->xf[i];
04200                 position[1] = thee->yf[j];
04201                 position[2] = thee->zf[k];
04202                 rc = Vgrid_value(thee->potMap, position, &pot);
04203                 if (!rc) {
04204                     Vnm_print(2, "fillcoChargeMap:  Error -- fell off of potential map at (%g, %g,
      %g)!\n",
04205                               position[0], position[1], position[2]);
04206                     VASSERT(0);
04207                 }
04208                 thee->pot[IJK(i,j,k)] = pot;
04209             }
04210         }
04211     }
04212 
04213 }
04214 
04215 #if  defined(WITH_TINKER)
04216 VPRIVATE void bcfl_mdh_tinker(Vpmg *thee){
04217 
04218     int i,j,k,iatom;
04219     int nx, ny, nz;
04220 
04221     double val, *apos, gpos[3], tensor[9];
04222     double *dipole, *quadrupole;
04223     double size, charge, xkappa, eps_w, eps_p, T, pre1, dist;
04224 
```

```
04225     double ux,uy,uz,xr,yr,zr;
04226     double qxx,qxy,qxz,qyx,qyy,qyz,qzx,qzy,qzz;
04227
04228     double *xf, *yf, *zf;
04229     double *gxcf, *gycf, *gzcf;
04230
04231     Vpbe *pbe;
04232     Vatom *atom;
04233     Valist *alist;
04234
04235     pbe = thee->pbe;
04236     alist = thee->pbe->alist;
04237     nx = thee->pmgp->nx;
04238     ny = thee->pmgp->ny;
04239     nz = thee->pmgp->nz;
04240
04241     xf = thee->xf;
04242     yf = thee->yf;
04243     zf = thee->zf;
04244
04245     gxcf = thee->gxcf;
04246     gycf = thee->gycf;
04247     gzcf = thee->gzcf;
04248
04249     /* For each "atom" (only one for bcfl=1), we use the following formula to
04250      * calculate the boundary conditions:
04251      *    g(x) = \frac{q e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
04252      *          * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
04253      *          * 1/d
04254      * where d = ||x - x_0|| (in m) and a is the size of the atom (in m).
04255      * We only need to evaluate some of these prefactors once:
04256      *    pre1 = \frac{e_c}{4*\pi*\eps_0*\eps_w*k_b*T}
04257      * which gives the potential as
04258      *    g(x) = pre1 * q/d * \frac{exp(-xkappa*(d - a))}{1+xkappa*a}
04259      */
04260     eps_w = Vpbe_getSolventDiel(pbe);           /* Dimensionless */
04261     eps_p = Vpbe_getSoluteDiel(pbe);           /* Dimensionless */
04262     T = Vpbe_getTemperature(pbe);               /* K             */
04263     pre1 = (Vunit_ec*Vunit_ec)/(4*VPI*Vunit_eps0*Vunit_kb*T);
04264
04265     /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
04266      * m/A, then we will only need to deal with distances and sizes in
04267      * Angstroms rather than meters.                                    */
04268     xkappa = Vpbe_getXkappa(pbe);              /* A^{-1}        */
04269     pre1 = pre1*(1.0e10);
04270
04271     /* Finally, if we convert keep xkappa in A^{-1} and scale pre1 by
04272      * m/A, then we will only need to deal with distances and sizes in
04273      * Angstroms rather than meters.                                    */
04274     xkappa = Vpbe_getXkappa(pbe);              /* A^{-1}        */
04275
04276     for(k=0;k<nz;k++){
04277         gpos[2] = zf[k];
04278         for(j=0;j<ny;j++){
04279             gpos[1] = yf[j];
04280             for(i=0;i<nx;i++){
04281                 gpos[0] = xf[i];
04282                 if(gridPointIsValid(i, j, k, nx, ny, nz)){
04283
04284                     val = 0.0;
04285
04286                     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
04287                         atom = Valist_getAtom(alist, iatom);
04288                         apos = Vatom_getPosition(atom);
04289                         size = Vatom_getRadius(atom);
04290
04291                         charge = 0.0;
04292
04293                         dipole = VNULL;
04294                         quadrupole = VNULL;
04295
04296                         if (thee->chargeSrc == VCM_PERMANENT) {
04297                             charge = Vatom_getCharge(atom);
04298                             dipole = Vatom_getDipole(atom);
04299                             quadrupole = Vatom_getQuadrupole(atom);
04300                         } else if (thee->chargeSrc == VCM_INDUCED) {
04301                             dipole = Vatom_getInducedDipole(atom);
04302                         } else {
04303                             dipole = Vatom_getNLInducedDipole(atom);
04304                         }
04305
```

```
04306                         ux = dipole[0];
04307                         uy = dipole[1];
04308                         uz = dipole[2];
04309
04310                         if (quadrupole != VNULL) {
04311                             /* The factor of 1/3 results from using a
04312                              traceless quadrupole definition. See, for example,
04313                              "The Theory of Intermolecular Forces" by A.J. Stone,
04314                              Chapter 3. */
04315                             qxx = quadrupole[0] / 3.0;
04316                             qxy = quadrupole[1] / 3.0;
04317                             qxz = quadrupole[2] / 3.0;
04318                             qyx = quadrupole[3] / 3.0;
04319                             qyy = quadrupole[4] / 3.0;
04320                             qyz = quadrupole[5] / 3.0;
04321                             qzx = quadrupole[6] / 3.0;
04322                             qzy = quadrupole[7] / 3.0;
04323                             qzz = quadrupole[8] / 3.0;
04324                         } else {
04325                             qxx = 0.0;
04326                             qxy = 0.0;
04327                             qxz = 0.0;
04328                             qyx = 0.0;
04329                             qyy = 0.0;
04330                             qyz = 0.0;
04331                             qzx = 0.0;
04332                             qzy = 0.0;
04333                             qzz = 0.0;
04334                         }
04335
04336                         xr = gpos[0] - apos[0];
04337                         yr = gpos[1] - apos[1];
04338                         zr = gpos[2] - apos[2];
04339
04340                         dist = VSQRT(VSQR(xr) + VSQR(yr) + VSQR(zr));
04341                         multipolebc(dist, xkappa, eps_p, eps_w, size, tensor);
04342
04343                         val += pre1*charge*tensor[0];
04344                         val -= pre1*ux*xr*tensor[1];
04345                         val -= pre1*uy*yr*tensor[1];
04346                         val -= pre1*uz*zr*tensor[1];
04347                         val += pre1*qxx*xr*xr*tensor[2];
04348                         val += pre1*qyy*yr*yr*tensor[2];
04349                         val += pre1*qzz*zr*zr*tensor[2];
04350                         val += pre1*2.0*qxy*xr*yr*tensor[2];
04351                         val += pre1*2.0*qxz*xr*zr*tensor[2];
04352                         val += pre1*2.0*qyz*yr*zr*tensor[2];
04353
04354                     }
04355
04356                     if(i==0){
04357                         gxcf[IJKx(j,k,0)] = val;
04358                     }
04359                     if(i==nx-1){
04360                         gxcf[IJKx(j,k,1)] = val;
04361                     }
04362                     if(j==0){
04363                         gycf[IJKy(i,k,0)] = val;
04364                     }
04365                     if(j==ny-1){
04366                         gycf[IJKy(i,k,1)] = val;
04367                     }
04368                     if(k==0){
04369                         gzcf[IJKz(i,j,0)] = val;
04370                     }
04371                     if(k==nz-1){
04372                         gzcf[IJKz(i,j,1)] = val;
04373                     }
04374                 } /* End grid point is valid */
04375             } /* End i loop */
04376         } /* End j loop */
04377     } /* End k loop */
04378
04379 }
04380 #endif
04381
04382 VPRIVATE void bcCalc(Vpmg *thee){
04383
04384     int i, j, k;
04385     int nx, ny, nz;
04386
```

```
04387     double zmem, eps_m, Lmem, memv, eps_w, xkappa;
04388
04389     nx = thee->pmgp->nx;
04390     ny = thee->pmgp->ny;
04391     nz = thee->pmgp->nz;
04392
04393     /* Zero out the boundaries */
04394     /* the "i" boundaries (dirichlet) */
04395     for (k=0; k<nz; k++) {
04396         for (j=0; j<ny; j++) {
04397             thee->gxcf[IJKx(j,k,0)] = 0.0;
04398             thee->gxcf[IJKx(j,k,1)] = 0.0;
04399             thee->gxcf[IJKx(j,k,2)] = 0.0;
04400             thee->gxcf[IJKx(j,k,3)] = 0.0;
04401         }
04402     }
04403
04404     /* the "j" boundaries (dirichlet) */
04405     for (k=0; k<nz; k++) {
04406         for (i=0; i<nx; i++) {
04407             thee->gycf[IJKy(i,k,0)] = 0.0;
04408             thee->gycf[IJKy(i,k,1)] = 0.0;
04409             thee->gycf[IJKy(i,k,2)] = 0.0;
04410             thee->gycf[IJKy(i,k,3)] = 0.0;
04411         }
04412     }
04413
04414     /* the "k" boundaries (dirichlet) */
04415     for (j=0; j<ny; j++) {
04416         for (i=0; i<nx; i++) {
04417             thee->gzcf[IJKz(i,j,0)] = 0.0;
04418             thee->gzcf[IJKz(i,j,1)] = 0.0;
04419             thee->gzcf[IJKz(i,j,2)] = 0.0;
04420             thee->gzcf[IJKz(i,j,3)] = 0.0;
04421         }
04422     }
04423
04424     switch (thee->pmgp->bcfl) {
04425             /*  If we have zero boundary conditions, we're done */
04426         case BCFL_ZERO:
04427             return;
04428         case BCFL_SDH:
04429             bcfl_sdh(thee);
04430             break;
04431         case BCFL_MDH:
04432 #if defined(WITH_TINKER)
04433             bcfl_mdh_tinker(thee);
04434 #else
04435
04436 #ifdef DEBUG_MAC_OSX_OCL
04437 #include "mach_chud.h"
04438             uint64_t mbeg = mach_absolute_time();
04439
04440             /*
04441              * If OpenCL is available we use it, otherwise fall back to
04442              * normal route (CPU multithreaded w/ OpenMP)
04443              */
04444             if (kOpenCLAvailable == 1) bcflnewOpenCL(thee);
04445             else bcflnew(thee);
04446
04447             mets_(&mbeg, "MDH");
04448 #else
04449             /* bcfl_mdh(thee); */
04450             bcflnew(thee);
04451 #endif /* DEBUG_MAC_OSX_OCL */
04452
04453 #endif /* WITH_TINKER */
04454             break;
04455         case BCFL_MEM:
04456
04457             zmem  = Vpbe_getzmem(thee->pbe);
04458             Lmem  = Vpbe_getLmem(thee->pbe);
04459             eps_m = Vpbe_getmembraneDiel(thee->pbe);
04460             memv =  Vpbe_getmemv(thee->pbe);
04461
04462             eps_w = Vpbe_getSolventDiel(thee->pbe);
04463             xkappa = Vpbe_getXkappa(thee->pbe);
04464
04465             bcfl_mem(zmem, Lmem, eps_m, eps_w, memv, xkappa,
04466                   thee->gxcf, thee->gycf, thee->gzcf,
04467                   thee->xf, thee->yf, thee->zf, nx, ny, nz);
```

```
04468             break;
04469         case BCFL_UNUSED:
04470             Vnm_print(2, "bcCalc:  Invalid bcfl (%d)!\n", thee->pmgp->bcfl);
04471             VASSERT(0);
04472             break;
04473         case BCFL_FOCUS:
04474             Vnm_print(2, "VPMG::bcCalc -- not appropriate for focusing!\n");
04475             VASSERT(0);
04476             break;
04477         case BCFL_MAP:
04478             bcfl_map(thee);
04479             focusFillBound(thee,VNULL);
04480             break;
04481         default:
04482             Vnm_print(2, "VPMG::bcCalc -- invalid boundary condition \
04483                       flag (%d)!\n", thee->pmgp->bcfl);
04484             VASSERT(0);
04485             break;
04486     }
04487 }
04488
04489 VPRIVATE void fillcoCoefMap(Vpmg *thee) {
04490
04491     Vpbe *pbe;
04492     double ionstr, position[3], tkappa, eps, pot, hx, hy, hzed;
04493     int i, j, k, nx, ny, nz;
04494     double kappamax;
04495     VASSERT(thee != VNULL);
04496
04497     /* Get PBE info */
04498     pbe = thee->pbe;
04499     ionstr = Vpbe_getBulkIonicStrength(pbe);
04500
04501     /* Mesh info */
04502     nx = thee->pmgp->nx;
04503     ny = thee->pmgp->ny;
04504     nz = thee->pmgp->nz;
04505     hx = thee->pmgp->hx;
04506     hy = thee->pmgp->hy;
04507     hzed = thee->pmgp->hzed;
04508
04509     if ((!thee->useDielXMap) || (!thee->useDielYMap)
04510         || (!thee->useDielZMap) || ((!thee->useKappaMap) && (ionstr>VPMGSMALL))) {
04511
04512         Vnm_print(2, "fillcoCoefMap:  You need to use all coefficient maps!\n");
04513         VASSERT(0);
04514
04515     }
04516
04517     /* Scale the kappa map to values between 0 and 1
04518        Thus get the maximum value in the map - this
04519        is theoretically unnecessary, but a good check.*/
04520     kappamax = -1.00;
04521     for (k=0; k<nz; k++) {
04522         for (j=0; j<ny; j++) {
04523             for (i=0; i<nx; i++) {
04524                 if (ionstr > VPMGSMALL) {
04525                     position[0] = thee->xf[i];
04526                     position[1] = thee->yf[j];
04527                     position[2] = thee->zf[k];
04528                     if (!Vgrid_value(thee->kappaMap, position, &tkappa)) {
04529                         Vnm_print(2, "Vpmg_fillco:  Off kappaMap at:\n");
04530                         Vnm_print(2, "Vpmg_fillco:  (x,y,z) = (%g,%g %g)\n",
04531                                   position[0], position[1], position[2]);
04532                         VASSERT(0);
04533                     }
04534                     if (tkappa > kappamax) {
04535                         kappamax = tkappa;
04536                     }
04537                     if (tkappa < 0.0){
04538                       Vnm_print(2, "Vpmg_fillcoCoefMap: Kappa map less than 0\n");
04539                       Vnm_print(2, "Vpmg_fillcoCoefMap: at (x,y,z) = (%g,%g %g)\n",
04540                                 position[0], position[1], position[2]);
04541                       VASSERT(0);
04542                     }
04543                 }
04544             }
04545         }
04546     }
04547
04548     if (kappamax > 1.0){
```

```
04549       Vnm_print(2, "Vpmg_fillcoCoefMap:  Maximum Kappa value\n");
04550       Vnm_print(2, "%g is greater than 1 - will scale appropriately!\n",
04551                 kappamax);
04552     }
04553     else {
04554       kappamax = 1.0;
04555     }
04556 
04557     for (k=0; k<nz; k++) {
04558         for (j=0; j<ny; j++) {
04559             for (i=0; i<nx; i++) {
04560 
04561                 if (ionstr > VPMGSMALL) {
04562                      position[0] = thee->xf[i];
04563                      position[1] = thee->yf[j];
04564                      position[2] = thee->zf[k];
04565                      if (!Vgrid_value(thee->kappaMap, position, &tkappa)) {
04566                          Vnm_print(2, "Vpmg_fillco:  Off kappaMap at:\n");
04567                          Vnm_print(2, "Vpmg_fillco:  (x,y,z) = (%g,%g %g)\n",
04568                            position[0], position[1], position[2]);
04569                          VASSERT(0);
04570                      }
04571                      if (tkappa < VPMGSMALL) tkappa = 0.0;
04572                      thee->kappa[IJK(i,j,k)] = (tkappa / kappamax);
04573                 }
04574 
04575                 position[0] = thee->xf[i] + 0.5*hx;
04576                 position[1] = thee->yf[j];
04577                 position[2] = thee->zf[k];
04578                 if (!Vgrid_value(thee->dielXMap, position, &eps)) {
04579                     Vnm_print(2, "Vpmg_fillco:  Off dielXMap at:\n");
04580                     Vnm_print(2, "Vpmg_fillco:  (x,y,z) = (%g,%g %g)\n",
04581                       position[0], position[1], position[2]);
04582                     VASSERT(0);
04583                  }
04584                  thee->epsx[IJK(i,j,k)] = eps;
04585 
04586                  position[0] = thee->xf[i];
04587                  position[1] = thee->yf[j] + 0.5*hy;
04588                  position[2] = thee->zf[k];
04589                  if (!Vgrid_value(thee->dielYMap, position, &eps)) {
04590                     Vnm_print(2, "Vpmg_fillco:  Off dielYMap at:\n");
04591                     Vnm_print(2, "Vpmg_fillco:  (x,y,z) = (%g,%g %g)\n",
04592                       position[0], position[1], position[2]);
04593                     VASSERT(0);
04594                  }
04595                  thee->epsy[IJK(i,j,k)] = eps;
04596 
04597                  position[0] = thee->xf[i];
04598                  position[1] = thee->yf[j];
04599                  position[2] = thee->zf[k] + 0.5*hzed;
04600                  if (!Vgrid_value(thee->dielZMap, position, &eps)) {
04601                     Vnm_print(2, "Vpmg_fillco:  Off dielZMap at:\n");
04602                     Vnm_print(2, "Vpmg_fillco:  (x,y,z) = (%g,%g %g)\n",
04603                       position[0], position[1], position[2]);
04604                     VASSERT(0);
04605                  }
04606                  thee->epsz[IJK(i,j,k)] = eps;
04607             }
04608         }
04609     }
04610 }
04611 
04612 VPRIVATE void fillcoCoefMol(Vpmg *thee) {
04613 
04614     if (thee->useDielXMap || thee->useDielYMap || thee->useDielZMap ||
04615       thee->useKappaMap)  {
04616 
04617         fillcoCoefMap(thee);
04618 
04619     } else {
04620 
04621         fillcoCoefMolDiel(thee);
04622         fillcoCoefMolIon(thee);
04623 
04624     }
04625 
04626 }
04627 
04628 VPRIVATE void fillcoCoefMolIon(Vpmg *thee) {
04629 
```

```
04630     Vacc *acc;
04631     Valist *alist;
04632     Vpbe *pbe;
04633     Vatom *atom;
04634     double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr;
04635     double xlen, ylen, zlen, irad;
04636     double hx, hy, hzed, *apos, arad;
04637     int i, nx, ny, nz, iatom;
04638     Vsurf_Meth surfMeth;
04639
04640     VASSERT(thee != VNULL);
04641     surfMeth = thee->surfMeth;
04642
04643     /* Get PBE info */
04644     pbe = thee->pbe;
04645     acc = pbe->acc;
04646     alist = pbe->alist;
04647     irad = Vpbe_getMaxIonRadius(pbe);
04648     ionstr = Vpbe_getBulkIonicStrength(pbe);
04649
04650     /* Mesh info */
04651     nx = thee->pmgp->nx;
04652     ny = thee->pmgp->ny;
04653     nz = thee->pmgp->nz;
04654     hx = thee->pmgp->hx;
04655     hy = thee->pmgp->hy;
04656     hzed = thee->pmgp->hzed;
04657
04658     /* Define the total domain size */
04659     xlen = thee->pmgp->xlen;
04660     ylen = thee->pmgp->ylen;
04661     zlen = thee->pmgp->zlen;
04662
04663     /* Define the min/max dimensions */
04664     xmin = thee->pmgp->xcent - (xlen/2.0);
04665     ymin = thee->pmgp->ycent - (ylen/2.0);
04666     zmin = thee->pmgp->zcent - (zlen/2.0);
04667     xmax = thee->pmgp->xcent + (xlen/2.0);
04668     ymax = thee->pmgp->ycent + (ylen/2.0);
04669     zmax = thee->pmgp->zcent + (zlen/2.0);
04670
04671     /* This is a floating point parameter related to the non-zero nature of the
04672      * bulk ionic strength.  If the ionic strength is greater than zero; this
04673      * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
04674      * Otherwise, this parameter is set to 0.0 */
04675     if (ionstr > VPMGSMALL) ionmask = 1.0;
04676     else ionmask = 0.0;
04677
04678     /* Reset the kappa array, marking everything accessible */
04679     for (i=0; i<(nx*ny*nz); i++) thee->kappa[i] = ionmask;
04680
04681     if (ionstr < VPMGSMALL) return;
04682
04683     /* Loop through the atoms and set kappa = 0.0 (inaccessible) if a point
04684      * is inside the ion-inflated van der Waals radii */
04685     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
04686
04687         atom = Valist_getAtom(alist, iatom);
04688         apos = Vatom_getPosition(atom);
04689         arad = Vatom_getRadius(atom);
04690
04691         if (arad > VSMALL) {
04692
04693             /* Make sure we're on the grid */
04694             if ((apos[0]<(xmin-irad-arad)) || (apos[0]>(xmax+irad+arad))  || \
04695                 (apos[1]<(ymin-irad-arad)) || (apos[1]>(ymax+irad+arad))  || \
04696                 (apos[2]<(zmin-irad-arad)) || (apos[2]>(zmax+irad+arad))) {
04697                 if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
04698                     (thee->pmgp->bcfl != BCFL_MAP)) {
04699                     Vnm_print(2,
04700     "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):\n",
04701                       iatom, apos[0], apos[1], apos[2]);
04702                     Vnm_print(2, "Vpmg_fillco:  xmin = %g, xmax = %g\n",
04703                       xmin, xmax);
04704                     Vnm_print(2, "Vpmg_fillco:  ymin = %g, ymax = %g\n",
04705                       ymin, ymax);
04706                     Vnm_print(2, "Vpmg_fillco:  zmin = %g, zmax = %g\n",
04707                       zmin, zmax);
04708                 }
04709                 fflush(stderr);
04710
```

```
04711             } else { /* if we're on the mesh */
04712
04713                 /* Mark ions */
04714                 markSphere((irad+arad), apos,
04715                         nx, ny, nz,
04716                         hx, hy, hzed,
04717                         xmin, ymin, zmin,
04718                         thee->kappa, 0.0);
04719
04720             } /* endif (on the mesh) */
04721         }
04722     } /* endfor (over all atoms) */
04723
04724 }
04725
04726 VPRIVATE void fillcoCoefMolDiel(Vpmg *thee) {
04727
04728     /* Always call NoSmooth to fill the epsilon arrays */
04729     fillcoCoefMolDielNoSmooth(thee);
04730
04731     /* Call the smoothing algorithm as needed */
04732     if (thee->surfMeth == VSM_MOLSMOOTH) {
04733         fillcoCoefMolDielSmooth(thee);
04734     }
04735 }
04736
04737 VPRIVATE void fillcoCoefMolDielNoSmooth(Vpmg *thee) {
04738
04739     Vacc *acc;
04740     VaccSurf *asurf;
04741     Valist *alist;
04742     Vpbe *pbe;
04743     Vatom *atom;
04744     double xmin, xmax, ymin, ymax, zmin, zmax;
04745     double xlen, ylen, zlen, position[3];
04746     double srad, epsw, epsp, deps, area;
04747     double hx, hy, hzed, *apos, arad;
04748     int i, nx, ny, nz, ntot, iatom, ipt;
04749
04750     /* Get PBE info */
04751     pbe = thee->pbe;
04752     acc = pbe->acc;
04753     alist = pbe->alist;
04754     srad = Vpbe_getSolventRadius(pbe);
04755     epsw = Vpbe_getSolventDiel(pbe);
04756     epsp = Vpbe_getSoluteDiel(pbe);
04757
04758     /* Mesh info */
04759     nx = thee->pmgp->nx;
04760     ny = thee->pmgp->ny;
04761     nz = thee->pmgp->nz;
04762     hx = thee->pmgp->hx;
04763     hy = thee->pmgp->hy;
04764     hzed = thee->pmgp->hzed;
04765
04766     /* Define the total domain size */
04767     xlen = thee->pmgp->xlen;
04768     ylen = thee->pmgp->ylen;
04769     zlen = thee->pmgp->zlen;
04770
04771     /* Define the min/max dimensions */
04772     xmin = thee->pmgp->xcent - (xlen/2.0);
04773     ymin = thee->pmgp->ycent - (ylen/2.0);
04774     zmin = thee->pmgp->zcent - (zlen/2.0);
04775     xmax = thee->pmgp->xcent + (xlen/2.0);
04776     ymax = thee->pmgp->ycent + (ylen/2.0);
04777     zmax = thee->pmgp->zcent + (zlen/2.0);
04778
04779     /* Reset the arrays */
04780     ntot = nx*ny*nz;
04781     for (i=0; i<ntot; i++) {
04782         thee->epsx[i] = epsw;
04783         thee->epsy[i] = epsw;
04784         thee->epsz[i] = epsw;
04785     }
04786
04787     /* Loop through the atoms and set a{123}cf = 0.0 (inaccessible)
04788      * if a point is inside the solvent-inflated van der Waals radii */
04789 #pragma omp parallel for default(shared) private(iatom,atom,apos,arad)
04790     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
04791
```

```
04792         atom = Valist_getAtom(alist, iatom);
04793         apos = Vatom_getPosition(atom);
04794         arad = Vatom_getRadius(atom);
04795 
04796         /* Make sure we're on the grid */
04797         if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
04798             (apos[1]<=ymin) || (apos[1]>=ymax)  || \
04799             (apos[2]<=zmin) || (apos[2]>=zmax)) {
04800             if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
04801                 (thee->pmgp->bcfl != BCFL_MAP)) {
04802                 Vnm_print(2, "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f,\
04803  %4.3f) is off the mesh (ignoring):\n",
04804                   iatom, apos[0], apos[1], apos[2]);
04805                 Vnm_print(2, "Vpmg_fillco:  xmin = %g, xmax = %g\n",
04806                   xmin, xmax);
04807                 Vnm_print(2, "Vpmg_fillco:  ymin = %g, ymax = %g\n",
04808                   ymin, ymax);
04809                 Vnm_print(2, "Vpmg_fillco:  zmin = %g, zmax = %g\n",
04810                   zmin, zmax);
04811             }
04812             fflush(stderr);
04813 
04814         } else { /* if we're on the mesh */
04815 
04816             if (arad > VSMALL) {
04817                 /* Mark x-shifted dielectric */
04818                 markSphere((arad+srad), apos,
04819                         nx, ny, nz,
04820                         hx, hy, hzed,
04821                         (xmin+0.5*hx), ymin, zmin,
04822                         thee->epsx, epsp);
04823 
04824                 /* Mark y-shifted dielectric */
04825                 markSphere((arad+srad), apos,
04826                         nx, ny, nz,
04827                         hx, hy, hzed,
04828                         xmin, (ymin+0.5*hy), zmin,
04829                         thee->epsy, epsp);
04830 
04831                 /* Mark z-shifted dielectric */
04832                 markSphere((arad+srad), apos,
04833                         nx, ny, nz,
04834                         hx, hy, hzed,
04835                         xmin, ymin, (zmin+0.5*hzed),
04836                         thee->epsz, epsp);
04837             }
04838 
04839         } /* endif (on the mesh) */
04840     } /* endfor (over all atoms) */
04841 
04842     area = Vacc_SASA(acc, srad);
04843 
04844     /* We only need to do the next step for non-zero solvent radii */
04845     if (srad > VSMALL) {
04846 
04847         /* Now loop over the solvent accessible surface points */
04848 
04849 #pragma omp parallel for default(shared) private(iatom,atom,area,asurf,ipt,position)
04850         for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
04851             atom = Valist_getAtom(alist, iatom);
04852             area = Vacc_atomSASA(acc, srad, atom);
04853             if (area > 0.0 ) {
04854                 asurf = Vacc_atomSASPoints(acc, srad, atom);
04855 
04856                 /* Use each point on the SAS to reset the solvent accessibility */
04857                 /* TODO:  Make sure we're not still wasting time here. */
04858                 for (ipt=0; ipt<(asurf->npts); ipt++) {
04859 
04860                     position[0] = asurf->xpts[ipt];
04861                     position[1] = asurf->ypts[ipt];
04862                     position[2] = asurf->zpts[ipt];
04863 
04864                     /* Mark x-shifted dielectric */
04865                     markSphere(srad, position,
04866                                nx, ny, nz,
04867                                hx, hy, hzed,
04868                                (xmin+0.5*hx), ymin, zmin,
04869                                thee->epsx, epsw);
04870 
04871                     /* Mark y-shifted dielectric */
04872                     markSphere(srad, position,
```

```
04873                                 nx, ny, nz,
04874                                 hx, hy, hzed,
04875                                 xmin, (ymin+0.5*hy), zmin,
04876                                 thee->epsy, epsw);
04877
04878                     /* Mark z-shifted dielectric */
04879                     markSphere(srad, position,
04880                                 nx, ny, nz,
04881                                 hx, hy, hzed,
04882                                 xmin, ymin, (zmin+0.5*hzed),
04883                                 thee->epsz, epsw);
04884
04885                 }
04886             }
04887         }
04888     }
04889 }
04890
04891 VPRIVATE void fillcoCoefMolDielSmooth(Vpmg *thee) {
04892
04893   /* This function smoothes using a 9 point method based on
04894      Bruccoleri, et al. J Comput Chem 18 268-276 (1997).  The nine points
04895      used are the shifted grid point and the 8 points that are 1/sqrt(2)
04896      grid spacings away.  The harmonic mean of the 9 points is then used to
04897      find the overall dielectric value for the point in question. The use of
04898      this function assumes that the non-smoothed values were placed in the
04899      dielectric arrays by the fillcoCoefMolDielNoSmooth function.*/
04900
04901     Vpbe *pbe;
04902     double frac, epsw;
04903     int i, j, k, nx, ny, nz, numpts;
04904
04905     /* Mesh info */
04906     nx = thee->pmgp->nx;
04907     ny = thee->pmgp->ny;
04908     nz = thee->pmgp->nz;
04909
04910     pbe = thee->pbe;
04911     epsw = Vpbe_getSolventDiel(pbe);
04912
04913     /* Copy the existing diel arrays to work arrays */
04914     for (i=0; i<(nx*ny*nz); i++) {
04915         thee->a1cf[i] = thee->epsx[i];
04916         thee->a2cf[i] = thee->epsy[i];
04917         thee->a3cf[i] = thee->epsz[i];
04918         thee->epsx[i] = epsw;
04919         thee->epsy[i] = epsw;
04920         thee->epsz[i] = epsw;
04921     }
04922
04923     /* Smooth the dielectric values */
04924     for (i=0; i<nx; i++) {
04925         for (j=0; j<ny; j++) {
04926             for (k=0; k<nz; k++) {
04927
04928                 /* Get the 8 points that are 1/sqrt(2) grid spacings away */
04929
04930                 /* Points for the X-shifted array */
04931                 frac = 1.0/thee->a1cf[IJK(i,j,k)];
04932                 frac += 1.0/thee->a2cf[IJK(i,j,k)];
04933                 frac += 1.0/thee->a3cf[IJK(i,j,k)];
04934                 numpts = 3;
04935
04936                 if (j > 0) {
04937                     frac += 1.0/thee->a2cf[IJK(i,j-1,k)];
04938                     numpts += 1;
04939                 }
04940                 if (k > 0) {
04941                     frac += 1.0/thee->a3cf[IJK(i,j,k-1)];
04942                     numpts += 1;
04943                 }
04944                 if (i < (nx-1)){
04945                     frac += 1.0/thee->a2cf[IJK(i+1,j,k)];
04946                     frac += 1.0/thee->a3cf[IJK(i+1,j,k)];
04947                     numpts += 2;
04948                     if (j > 0) {
04949                         frac += 1.0/thee->a2cf[IJK(i+1,j-1,k)];
04950                         numpts += 1;
04951                     }
04952                     if (k > 0) {
04953                         frac += 1.0/thee->a3cf[IJK(i+1,j,k-1)];
```

```
04954                         numpts += 1;
04955                     }
04956                 }
04957                 thee->epsx[IJK(i,j,k)] = numpts/frac;
04958
04959                 /* Points for the Y-shifted array */
04960                 frac = 1.0/thee->a2cf[IJK(i,j,k)];
04961                 frac += 1.0/thee->a1cf[IJK(i,j,k)];
04962                 frac += 1.0/thee->a3cf[IJK(i,j,k)];
04963                 numpts = 3;
04964
04965                 if (i > 0) {
04966                     frac += 1.0/thee->a1cf[IJK(i-1,j,k)];
04967                     numpts += 1;
04968                 }
04969                 if (k > 0) {
04970                     frac += 1.0/thee->a3cf[IJK(i,j,k-1)];
04971                     numpts += 1;
04972                 }
04973                 if (j < (ny-1)){
04974                     frac += 1.0/thee->a1cf[IJK(i,j+1,k)];
04975                     frac += 1.0/thee->a3cf[IJK(i,j+1,k)];
04976                     numpts += 2;
04977                     if (i > 0) {
04978                         frac += 1.0/thee->a1cf[IJK(i-1,j+1,k)];
04979                         numpts += 1;
04980                     }
04981                     if (k > 0) {
04982                         frac += 1.0/thee->a3cf[IJK(i,j+1,k-1)];
04983                         numpts += 1;
04984                     }
04985                 }
04986                 thee->epsy[IJK(i,j,k)] = numpts/frac;
04987
04988                 /* Points for the Z-shifted array */
04989                 frac = 1.0/thee->a3cf[IJK(i,j,k)];
04990                 frac += 1.0/thee->a1cf[IJK(i,j,k)];
04991                 frac += 1.0/thee->a2cf[IJK(i,j,k)];
04992                 numpts = 3;
04993
04994                 if (i > 0) {
04995                     frac += 1.0/thee->a1cf[IJK(i-1,j,k)];
04996                     numpts += 1;
04997                 }
04998                 if (j > 0) {
04999                     frac += 1.0/thee->a2cf[IJK(i,j-1,k)];
05000                     numpts += 1;
05001                 }
05002                 if (k < (nz-1)){
05003                     frac += 1.0/thee->a1cf[IJK(i,j,k+1)];
05004                     frac += 1.0/thee->a2cf[IJK(i,j,k+1)];
05005                     numpts += 2;
05006                     if (i > 0) {
05007                         frac += 1.0/thee->a1cf[IJK(i-1,j,k+1)];
05008                         numpts += 1;
05009                     }
05010                     if (j > 0) {
05011                         frac += 1.0/thee->a2cf[IJK(i,j-1,k+1)];
05012                         numpts += 1;
05013                     }
05014                 }
05015                 thee->epsz[IJK(i,j,k)] = numpts/frac;
05016             }
05017         }
05018     }
05019 }
05020
05021
05022 VPRIVATE void fillcoCoefSpline(Vpmg *thee) {
05023
05024     Valist *alist;
05025     Vpbe *pbe;
05026     Vatom *atom;
05027     double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr, dist2;
05028     double xlen, ylen, zlen, position[3], itot, stot, ictot, ictot2, sctot;
05029     double irad, dx, dy, dz, epsw, epsp, w2i;
05030     double hx, hy, hzed, *apos, arad, sctot2;
05031     double dx2, dy2, dz2, stot2, itot2, rtot, rtot2, splineWin, w3i;
05032     double dist, value, sm, sm2;
05033     int i, j, k, nx, ny, nz, iatom;
05034     int imin, imax, jmin, jmax, kmin, kmax;
```

```
05035
05036     VASSERT(thee != VNULL);
05037     splineWin = thee->splineWin;
05038     w2i = 1.0/(splineWin*splineWin);
05039     w3i = 1.0/(splineWin*splineWin*splineWin);
05040
05041     /* Get PBE info */
05042     pbe = thee->pbe;
05043     alist = pbe->alist;
05044     irad = Vpbe_getMaxIonRadius(pbe);
05045     ionstr = Vpbe_getBulkIonicStrength(pbe);
05046     epsw = Vpbe_getSolventDiel(pbe);
05047     epsp = Vpbe_getSoluteDiel(pbe);
05048
05049     /* Mesh info */
05050     nx = thee->pmgp->nx;
05051     ny = thee->pmgp->ny;
05052     nz = thee->pmgp->nz;
05053     hx = thee->pmgp->hx;
05054     hy = thee->pmgp->hy;
05055     hzed = thee->pmgp->hzed;
05056
05057     /* Define the total domain size */
05058     xlen = thee->pmgp->xlen;
05059     ylen = thee->pmgp->ylen;
05060     zlen = thee->pmgp->zlen;
05061
05062     /* Define the min/max dimensions */
05063     xmin = thee->pmgp->xcent - (xlen/2.0);
05064     ymin = thee->pmgp->ycent - (ylen/2.0);
05065     zmin = thee->pmgp->zcent - (zlen/2.0);
05066     xmax = thee->pmgp->xcent + (xlen/2.0);
05067     ymax = thee->pmgp->ycent + (ylen/2.0);
05068     zmax = thee->pmgp->zcent + (zlen/2.0);
05069
05070     /* This is a floating point parameter related to the non-zero nature of the
05071      * bulk ionic strength.  If the ionic strength is greater than zero; this
05072      * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
05073      * Otherwise, this parameter is set to 0.0 */
05074     if (ionstr > VPMGSMALL) ionmask = 1.0;
05075     else ionmask = 0.0;
05076
05077     /* Reset the kappa, epsx, epsy, and epsz arrays */
05078     for (i=0; i<(nx*ny*nz); i++) {
05079         thee->kappa[i] = 1.0;
05080         thee->epsx[i] = 1.0;
05081         thee->epsy[i] = 1.0;
05082         thee->epsz[i] = 1.0;
05083     }
05084
05085     /* Loop through the atoms and do assign the dielectric */
05086     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
05087
05088         atom = Valist_getAtom(alist, iatom);
05089         apos = Vatom_getPosition(atom);
05090         arad = Vatom_getRadius(atom);
05091
05092         /* Make sure we're on the grid */
05093         if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
05094             (apos[1]<=ymin) || (apos[1]>=ymax)  || \
05095             (apos[2]<=zmin) || (apos[2]>=zmax)) {
05096             if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
05097                 (thee->pmgp->bcfl != BCFL_MAP)) {
05098                 Vnm_print(2, "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f,\
05099 %4.3f) is off the mesh (ignoring):\n",
05100                   iatom, apos[0], apos[1], apos[2]);
05101                 Vnm_print(2, "Vpmg_fillco:    xmin = %g, xmax = %g\n",
05102                   xmin, xmax);
05103                 Vnm_print(2, "Vpmg_fillco:    ymin = %g, ymax = %g\n",
05104                   ymin, ymax);
05105                 Vnm_print(2, "Vpmg_fillco:    zmin = %g, zmax = %g\n",
05106                   zmin, zmax);
05107             }
05108             fflush(stderr);
05109
05110         } else if (arad > VPMGSMALL ) { /* if we're on the mesh */
05111
05112             /* Convert the atom position to grid reference frame */
05113             position[0] = apos[0] - xmin;
05114             position[1] = apos[1] - ymin;
05115             position[2] = apos[2] - zmin;
```

```
05116
05117             /* MARK ION ACCESSIBILITY AND DIELECTRIC VALUES FOR LATER
05118              * ASSIGNMENT (Steps #1-3) */
05119             itot = irad + arad + splineWin;
05120             itot2 = VSQR(itot);
05121             ictot = VMAX2(0, (irad + arad - splineWin));
05122             ictot2 = VSQR(ictot);
05123             stot = arad + splineWin;
05124             stot2 = VSQR(stot);
05125             sctot = VMAX2(0, (arad - splineWin));
05126             sctot2 = VSQR(sctot);
05127
05128            /* We'll search over grid points which are in the greater of
05129              * these two radii */
05130             rtot = VMAX2(itot, stot);
05131             rtot2 = VMAX2(itot2, stot2);
05132             dx = rtot + 0.5*hx;
05133             dy = rtot + 0.5*hy;
05134             dz = rtot + 0.5*hzed;
05135             imin = VMAX2(0,(int)floor((position[0] - dx)/hx));
05136             imax = VMIN2(nx-1,(int)ceil((position[0] + dx)/hx));
05137             jmin = VMAX2(0,(int)floor((position[1] - dy)/hy));
05138             jmax = VMIN2(ny-1,(int)ceil((position[1] + dy)/hy));
05139             kmin = VMAX2(0,(int)floor((position[2] - dz)/hzed));
05140             kmax = VMIN2(nz-1,(int)ceil((position[2] + dz)/hzed));
05141             for (i=imin; i<=imax; i++) {
05142                 dx2 = VSQR(position[0] - hx*i);
05143                 for (j=jmin; j<=jmax; j++) {
05144                     dy2 = VSQR(position[1] - hy*j);
05145                     for (k=kmin; k<=kmax; k++) {
05146                         dz2 = VSQR(position[2] - k*hzed);
05147
05148                         /* ASSIGN CCF */
05149                         if (thee->kappa[IJK(i,j,k)] > VPMGSMALL) {
05150                             dist2 = dz2 + dy2 + dx2;
05151                             if (dist2 >= itot2) {
05152                                 ;
05153                             }
05154                             if (dist2 <= ictot2) {
05155                                 thee->kappa[IJK(i,j,k)] = 0.0;
05156                             }
05157                             if ((dist2 < itot2) && (dist2 > ictot2)) {
05158                                 dist = VSQRT(dist2);
05159                                 sm = dist - (arad + irad) + splineWin;
05160                                 sm2 = VSQR(sm);
05161                                 value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
05162                                 thee->kappa[IJK(i,j,k)] *= value;
05163                             }
05164                         }
05165
05166                         /* ASSIGN A1CF */
05167                         if (thee->epsx[IJK(i,j,k)] > VPMGSMALL) {
05168                             dist2 = dz2+dy2+VSQR(position[0]-(i+0.5)*hx);
05169                             if (dist2 >= stot2) {
05170                                 thee->epsx[IJK(i,j,k)] *= 1.0;
05171                             }
05172                             if (dist2 <= sctot2) {
05173                                 thee->epsx[IJK(i,j,k)] = 0.0;
05174                             }
05175                             if ((dist2 > sctot2) && (dist2 < stot2)) {
05176                                 dist = VSQRT(dist2);
05177                                 sm = dist - arad + splineWin;
05178                                 sm2 = VSQR(sm);
05179                                 value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
05180                                 thee->epsx[IJK(i,j,k)] *= value;
05181                             }
05182                         }
05183
05184                         /* ASSIGN A2CF */
05185                         if (thee->epsy[IJK(i,j,k)] > VPMGSMALL) {
05186                             dist2 = dz2+dx2+VSQR(position[1]-(j+0.5)*hy);
05187                             if (dist2 >= stot2) {
05188                                 thee->epsy[IJK(i,j,k)] *= 1.0;
05189                             }
05190                             if (dist2 <= sctot2) {
05191                                 thee->epsy[IJK(i,j,k)] = 0.0;
05192                             }
05193                             if ((dist2 > sctot2) && (dist2 < stot2)) {
05194                                 dist = VSQRT(dist2);
05195                                 sm = dist - arad + splineWin;
05196                                 sm2 = VSQR(sm);
```

```
05197                                 value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
05198                                 thee->epsy[IJK(i,j,k)] *= value;
05199                             }
05200                         }
05201
05202                         /* ASSIGN A3CF */
05203                         if (thee->epsz[IJK(i,j,k)] > VPMGSMALL) {
05204                             dist2 = dy2+dx2+VSQR(position[2]-(k+0.5)*hzed);
05205                             if (dist2 >= stot2) {
05206                                 thee->epsz[IJK(i,j,k)] *= 1.0;
05207                             }
05208                             if (dist2 <= sctot2) {
05209                                 thee->epsz[IJK(i,j,k)] = 0.0;
05210                             }
05211                             if ((dist2 > sctot2) && (dist2 < stot2)) {
05212                                 dist = VSQRT(dist2);
05213                                 sm = dist - arad + splineWin;
05214                                 sm2 = VSQR(sm);
05215                                 value = 0.75*sm2*w2i - 0.25*sm*sm2*w3i;
05216                                 thee->epsz[IJK(i,j,k)] *= value;
05217                             }
05218                         }
05219
05220
05221                     } /* k loop */
05222                 } /* j loop */
05223             } /* i loop */
05224         } /* endif (on the mesh) */
05225     } /* endfor (over all atoms) */
05226
05227     Vnm_print(0, "Vpmg_fillco:  filling coefficient arrays\n");
05228     /* Interpret markings and fill the coefficient arrays */
05229     for (k=0; k<nz; k++) {
05230         for (j=0; j<ny; j++) {
05231             for (i=0; i<nx; i++) {
05232
05233                 thee->kappa[IJK(i,j,k)] = ionmask*thee->kappa[IJK(i,j,k)];
05234                 thee->epsx[IJK(i,j,k)] = (epsw-epsp)*thee->epsx[IJK(i,j,k)]
05235                   + epsp;
05236                 thee->epsy[IJK(i,j,k)] = (epsw-epsp)*thee->epsy[IJK(i,j,k)]
05237                   + epsp;
05238                 thee->epsz[IJK(i,j,k)] = (epsw-epsp)*thee->epsz[IJK(i,j,k)]
05239                   + epsp;
05240
05241             } /* i loop */
05242         } /* j loop */
05243     } /* k loop */
05244
05245 }
05246
05247 VPRIVATE void fillcoCoef(Vpmg *thee) {
05248
05249     VASSERT(thee != VNULL);
05250
05251     if (thee->useDielXMap || thee->useDielYMap ||
05252         thee->useDielZMap || thee->useKappaMap) {
05253         fillcoCoefMap(thee);
05254         return;
05255     }
05256
05257     switch(thee->surfMeth) {
05258         case VSM_MOL:
05259             Vnm_print(0, "fillcoCoef:  Calling fillcoCoefMol...\n");
05260             fillcoCoefMol(thee);
05261             break;
05262         case VSM_MOLSMOOTH:
05263             Vnm_print(0, "fillcoCoef:  Calling fillcoCoefMol...\n");
05264             fillcoCoefMol(thee);
05265             break;
05266         case VSM_SPLINE:
05267             Vnm_print(0, "fillcoCoef:  Calling fillcoCoefSpline...\n");
05268             fillcoCoefSpline(thee);
05269             break;
05270         case VSM_SPLINE3:
05271             Vnm_print(0, "fillcoCoef:  Calling fillcoCoefSpline3...\n");
05272             fillcoCoefSpline3(thee);
05273             break;
05274         case VSM_SPLINE4:
05275             Vnm_print(0, "fillcoCoef:  Calling fillcoCoefSpline4...\n");
05276             fillcoCoefSpline4(thee);
05277             break;
```

```
05278         default:
05279             Vnm_print(2, "fillcoCoef:  Invalid surfMeth (%d)!\n",
05280               thee->surfMeth);
05281             VASSERT(0);
05282             break;
05283     }
05284 }
05285
05286
05287 VPRIVATE Vrc_Codes fillcoCharge(Vpmg *thee) {
05288
05289     Vrc_Codes rc;
05290
05291     VASSERT(thee != VNULL);
05292
05293     if (thee->useChargeMap) {
05294         return fillcoChargeMap(thee);
05295     }
05296
05297     switch(thee->chargeMeth) {
05298         case VCM_TRIL:
05299             Vnm_print(0, "fillcoCharge:  Calling fillcoChargeSpline1...\n");
05300             fillcoChargeSpline1(thee);
05301             break;
05302         case VCM_BSPL2:
05303             Vnm_print(0, "fillcoCharge:  Calling fillcoChargeSpline2...\n");
05304             fillcoChargeSpline2(thee);
05305             break;
05306         case VCM_BSPL4:
05307             switch (thee->chargeSrc) {
05308                 case VCM_CHARGE:
05309                     Vnm_print(0, "fillcoCharge: Calling fillcoPermanentMultipole...\n");
05310                     fillcoPermanentMultipole(thee);
05311                     break;
05312 #if defined(WITH_TINKER)
05313                 case VCM_PERMANENT:
05314                     Vnm_print(0, "fillcoCharge: Calling fillcoPermanentMultipole...\n");
05315                     fillcoPermanentMultipole(thee);
05316                     break;
05317                 case VCM_INDUCED:
05318                     Vnm_print(0, "fillcoCharge: Calling fillcoInducedDipole...\n");
05319                     fillcoInducedDipole(thee);
05320                     break;
05321                 case VCM_NLINDUCED:
05322                      Vnm_print(0, "fillcoCharge: Calling fillcoNLInducedDipole...\n");
05323                      fillcoNLInducedDipole(thee);
05324                      break;
05325 #endif /* if defined(WITH_TINKER) */
05326                 default:
05327                     Vnm_print(2, "fillcoCharge:  Invalid chargeSource (%d)!\n",
05328                       thee->chargeSrc);
05329                     return VRC_FAILURE;
05330                     break;
05331             }
05332             break;
05333         default:
05334             Vnm_print(2, "fillcoCharge:  Invalid chargeMeth (%d)!\n",
05335               thee->chargeMeth);
05336             return VRC_FAILURE;
05337             break;
05338     }
05339
05340     return VRC_SUCCESS;
05341 }
05342
05343 VPRIVATE Vrc_Codes fillcoChargeMap(Vpmg *thee) {
05344
05345     Vpbe *pbe;
05346     double position[3], charge, zmagic, hx, hy, hzed;
05347     int i, j, k, nx, ny, nz, rc;
05348
05349
05350     VASSERT(thee != VNULL);
05351
05352     /* Get PBE info */
05353     pbe = thee->pbe;
05354     zmagic = Vpbe_getZmagic(pbe);
05355
05356     /* Mesh info */
05357     nx = thee->pmgp->nx;
05358     ny = thee->pmgp->ny;
```

```
05359     nz = thee->pmgp->nz;
05360     hx = thee->pmgp->hx;
05361     hy = thee->pmgp->hy;
05362     hzed = thee->pmgp->hzed;
05363
05364     /* Reset the charge array */
05365     for (i=0; i<(nx*ny*nz); i++) thee->charge[i] = 0.0;
05366
05367     /* Fill in the source term (atomic charges) */
05368     Vnm_print(0, "Vpmg_fillco:  filling in source term.\n");
05369     for (k=0; k<nz; k++) {
05370         for (j=0; j<ny; j++) {
05371             for (i=0; i<nx; i++) {
05372                 position[0] = thee->xf[i];
05373                 position[1] = thee->yf[j];
05374                 position[2] = thee->zf[k];
05375                 rc = Vgrid_value(thee->chargeMap, position, &charge);
05376                 if (!rc) {
05377                     Vnm_print(2, "fillcoChargeMap:  Error -- fell off of charge map at (%g, %g, %g)!\n",
05378                           position[0], position[1], position[2]);
05379                     return VRC_FAILURE;
05380                 }
05381                 /* Scale the charge to internal units */
05382                 charge = charge*zmagic;
05383                 thee->charge[IJK(i,j,k)] = charge;
05384             }
05385         }
05386     }
05387
05388     return VRC_SUCCESS;
05389 }
05390
05391 VPRIVATE void fillcoChargeSpline1(Vpmg *thee) {
05392
05393     Valist *alist;
05394     Vpbe *pbe;
05395     Vatom *atom;
05396     double xmin, xmax, ymin, ymax, zmin, zmax;
05397     double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;
05398     double charge, dx, dy, dz, zmagic, hx, hy, hzed, *apos;
05399     int i, nx, ny, nz, iatom, ihi, ilo, jhi, jlo, khi, klo;
05400
05401
05402     VASSERT(thee != VNULL);
05403
05404     /* Get PBE info */
05405     pbe = thee->pbe;
05406     alist = pbe->alist;
05407     zmagic = Vpbe_getZmagic(pbe);
05408
05409     /* Mesh info */
05410     nx = thee->pmgp->nx;
05411     ny = thee->pmgp->ny;
05412     nz = thee->pmgp->nz;
05413     hx = thee->pmgp->hx;
05414     hy = thee->pmgp->hy;
05415     hzed = thee->pmgp->hzed;
05416
05417     /* Define the total domain size */
05418     xlen = thee->pmgp->xlen;
05419     ylen = thee->pmgp->ylen;
05420     zlen = thee->pmgp->zlen;
05421
05422     /* Define the min/max dimensions */
05423     xmin = thee->pmgp->xcent - (xlen/2.0);
05424     ymin = thee->pmgp->ycent - (ylen/2.0);
05425     zmin = thee->pmgp->zcent - (zlen/2.0);
05426     xmax = thee->pmgp->xcent + (xlen/2.0);
05427     ymax = thee->pmgp->ycent + (ylen/2.0);
05428     zmax = thee->pmgp->zcent + (zlen/2.0);
05429
05430     /* Reset the charge array */
05431     for (i=0; i<(nx*ny*nz); i++) thee->charge[i] = 0.0;
05432
05433     /* Fill in the source term (atomic charges) */
05434     Vnm_print(0, "Vpmg_fillco:  filling in source term.\n");
05435     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
05436
05437         atom = Valist_getAtom(alist, iatom);
05438         apos = Vatom_getPosition(atom);
05439         charge = Vatom_getCharge(atom);
```

```
05440
05441         /* Make sure we're on the grid */
05442         if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
05443             (apos[1]<=ymin) || (apos[1]>=ymax)  || \
05444             (apos[2]<=zmin) || (apos[2]>=zmax)) {
05445             if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
05446                 (thee->pmgp->bcfl != BCFL_MAP)) {
05447                 Vnm_print(2, "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f, \
05448 %4.3f) is off the mesh (ignoring):\n",
05449                   iatom, apos[0], apos[1], apos[2]);
05450                 Vnm_print(2, "Vpmg_fillco:    xmin = %g, xmax = %g\n",
05451                   xmin, xmax);
05452                 Vnm_print(2, "Vpmg_fillco:    ymin = %g, ymax = %g\n",
05453                   ymin, ymax);
05454                 Vnm_print(2, "Vpmg_fillco:    zmin = %g, zmax = %g\n",
05455                   zmin, zmax);
05456             }
05457             fflush(stderr);
05458         } else {
05459
05460             /* Convert the atom position to grid reference frame */
05461             position[0] = apos[0] - xmin;
05462             position[1] = apos[1] - ymin;
05463             position[2] = apos[2] - zmin;
05464
05465             /* Scale the charge to be a delta function */
05466             charge = charge*zmagic/(hx*hy*hzed);
05467
05468             /* Figure out which vertices we're next to */
05469             ifloat = position[0]/hx;
05470             jfloat = position[1]/hy;
05471             kfloat = position[2]/hzed;
05472
05473             ihi = (int)ceil(ifloat);
05474             ilo = (int)floor(ifloat);
05475             jhi = (int)ceil(jfloat);
05476             jlo = (int)floor(jfloat);
05477             khi = (int)ceil(kfloat);
05478             klo = (int)floor(kfloat);
05479
05480             /* Now assign fractions of the charge to the nearby verts */
05481             dx = ifloat - (double)(ilo);
05482             dy = jfloat - (double)(jlo);
05483             dz = kfloat - (double)(klo);
05484             thee->charge[IJK(ihi,jhi,khi)] += (dx*dy*dz*charge);
05485             thee->charge[IJK(ihi,jlo,khi)] += (dx*(1.0-dy)*dz*charge);
05486             thee->charge[IJK(ihi,jhi,klo)] += (dx*dy*(1.0-dz)*charge);
05487             thee->charge[IJK(ihi,jlo,klo)] += (dx*(1.0-dy)*(1.0-dz)*charge);
05488             thee->charge[IJK(ilo,jhi,khi)] += ((1.0-dx)*dy*dz *charge);
05489             thee->charge[IJK(ilo,jlo,khi)] += ((1.0-dx)*(1.0-dy)*dz *charge);
05490             thee->charge[IJK(ilo,jhi,klo)] += ((1.0-dx)*dy*(1.0-dz)*charge);
05491             thee->charge[IJK(ilo,jlo,klo)] += ((1.0-dx)*(1.0-dy)*(1.0-dz)*charge);
05492         } /* endif (on the mesh) */
05493     } /* endfor (each atom) */
05494 }
05495
05496 VPRIVATE double bspline2(double x) {
05497
05498     double m2m, m2, m3;
05499
05500     if ((x >= 0.0) && (x <= 2.0)) m2m = 1.0 - VABS(x - 1.0);
05501     else m2m = 0.0;
05502     if ((x >= 1.0) && (x <= 3.0)) m2 = 1.0 - VABS(x - 2.0);
05503     else m2 = 0.0;
05504
05505     if ((x >= 0.0) && (x <= 3.0)) m3 = 0.5*x*m2m + 0.5*(3.0-x)*m2;
05506     else m3 = 0.0;
05507
05508     return m3;
05509
05510 }
05511
05512 VPRIVATE double dbspline2(double x) {
05513
05514     double m2m, m2, dm3;
05515
05516     if ((x >= 0.0) && (x <= 2.0)) m2m = 1.0 - VABS(x - 1.0);
05517     else m2m = 0.0;
05518     if ((x >= 1.0) && (x <= 3.0)) m2 = 1.0 - VABS(x - 2.0);
05519     else m2 = 0.0;
05520
```

```
05521     dm3 = m2m - m2;
05522
05523     return dm3;
05524
05525 }
05526
05527
05528 VPRIVATE void fillcoChargeSpline2(Vpmg *thee) {
05529
05530     Valist *alist;
05531     Vpbe *pbe;
05532     Vatom *atom;
05533     double xmin, xmax, ymin, ymax, zmin, zmax, zmagic;
05534     double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;
05535     double charge, hx, hy, hzed, *apos, mx, my, mz;
05536     int i, ii, jj, kk, nx, ny, nz, iatom;
05537     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
05538
05539
05540     VASSERT(thee != VNULL);
05541
05542     /* Get PBE info */
05543     pbe = thee->pbe;
05544     alist = pbe->alist;
05545     zmagic = Vpbe_getZmagic(pbe);
05546
05547     /* Mesh info */
05548     nx = thee->pmgp->nx;
05549     ny = thee->pmgp->ny;
05550     nz = thee->pmgp->nz;
05551     hx = thee->pmgp->hx;
05552     hy = thee->pmgp->hy;
05553     hzed = thee->pmgp->hzed;
05554
05555     /* Define the total domain size */
05556     xlen = thee->pmgp->xlen;
05557     ylen = thee->pmgp->ylen;
05558     zlen = thee->pmgp->zlen;
05559
05560     /* Define the min/max dimensions */
05561     xmin = thee->pmgp->xcent - (xlen/2.0);
05562     ymin = thee->pmgp->ycent - (ylen/2.0);
05563     zmin = thee->pmgp->zcent - (zlen/2.0);
05564     xmax = thee->pmgp->xcent + (xlen/2.0);
05565     ymax = thee->pmgp->ycent + (ylen/2.0);
05566     zmax = thee->pmgp->zcent + (zlen/2.0);
05567
05568     /* Reset the charge array */
05569     for (i=0; i<(nx*ny*nz); i++) thee->charge[i] = 0.0;
05570
05571     /* Fill in the source term (atomic charges) */
05572     Vnm_print(0, "Vpmg_fillco:  filling in source term.\n");
05573     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
05574
05575         atom = Valist_getAtom(alist, iatom);
05576         apos = Vatom_getPosition(atom);
05577         charge = Vatom_getCharge(atom);
05578
05579         /* Make sure we're on the grid */
05580         if ((apos[0]<=(xmin-hx)) || (apos[0]>=(xmax+hx))  || \
05581             (apos[1]<=(ymin-hy)) || (apos[1]>=(ymax+hy))  || \
05582             (apos[2]<=(zmin-hzed)) || (apos[2]>=(zmax+hzed))) {
05583             if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
05584                 (thee->pmgp->bcfl != BCFL_MAP)) {
05585                 Vnm_print(2, "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f, \
05586 %4.3f) is off the mesh (for cubic splines!!) (ignoring this atom):\n",
05587                   iatom, apos[0], apos[1], apos[2]);
05588                 Vnm_print(2, "Vpmg_fillco:    xmin = %g, xmax = %g\n",
05589                   xmin, xmax);
05590                 Vnm_print(2, "Vpmg_fillco:    ymin = %g, ymax = %g\n",
05591                   ymin, ymax);
05592                 Vnm_print(2, "Vpmg_fillco:    zmin = %g, zmax = %g\n",
05593                   zmin, zmax);
05594             }
05595             fflush(stderr);
05596         } else {
05597
05598             /* Convert the atom position to grid reference frame */
05599             position[0] = apos[0] - xmin;
05600             position[1] = apos[1] - ymin;
05601             position[2] = apos[2] - zmin;
```

```
05602
05603             /* Scale the charge to be a delta function */
05604             charge = charge*zmagic/(hx*hy*hzed);
05605
05606             /* Figure out which vertices we're next to */
05607             ifloat = position[0]/hx;
05608             jfloat = position[1]/hy;
05609             kfloat = position[2]/hzed;
05610
05611             ip1   = (int)ceil(ifloat);
05612             ip2   = ip1 + 1;
05613             im1   = (int)floor(ifloat);
05614             im2   = im1 - 1;
05615             jp1   = (int)ceil(jfloat);
05616             jp2   = jp1 + 1;
05617             jm1   = (int)floor(jfloat);
05618             jm2   = jm1 - 1;
05619             kp1   = (int)ceil(kfloat);
05620             kp2   = kp1 + 1;
05621             km1   = (int)floor(kfloat);
05622             km2   = km1 - 1;
05623
05624             /* This step shouldn't be necessary, but it saves nasty debugging
05625              * later on if something goes wrong */
05626             ip2 = VMIN2(ip2,nx-1);
05627             ip1 = VMIN2(ip1,nx-1);
05628             im1 = VMAX2(im1,0);
05629             im2 = VMAX2(im2,0);
05630             jp2 = VMIN2(jp2,ny-1);
05631             jp1 = VMIN2(jp1,ny-1);
05632             jm1 = VMAX2(jm1,0);
05633             jm2 = VMAX2(jm2,0);
05634             kp2 = VMIN2(kp2,nz-1);
05635             kp1 = VMIN2(kp1,nz-1);
05636             km1 = VMAX2(km1,0);
05637             km2 = VMAX2(km2,0);
05638
05639             /* Now assign fractions of the charge to the nearby verts */
05640             for (ii=im2; ii<=ip2; ii++) {
05641                 mx = bspline2(VFCHI(ii,ifloat));
05642                 for (jj=jm2; jj<=jp2; jj++) {
05643                     my = bspline2(VFCHI(jj,jfloat));
05644                     for (kk=km2; kk<=kp2; kk++) {
05645                         mz = bspline2(VFCHI(kk,kfloat));
05646                         thee->charge[IJK(ii,jj,kk)] += (charge*mx*my*mz);
05647                     }
05648                 }
05649             }
05650
05651         } /* endif (on the mesh) */
05652     } /* endfor (each atom) */
05653 }
05654
05655 VPUBLIC int Vpmg_fillco(Vpmg *thee,
05656                         Vsurf_Meth surfMeth,
05657                         double splineWin,
05658                         Vchrg_Meth chargeMeth,
05659                         int useDielXMap,
05660                         Vgrid *dielXMap,
05661                         int useDielYMap,
05662                         Vgrid *dielYMap,
05663                         int useDielZMap,
05664                         Vgrid *dielZMap,
05665                         int useKappaMap,
05666                         Vgrid *kappaMap,
05667                         int usePotMap,
05668                         Vgrid *potMap,
05669                         int useChargeMap,
05670                         Vgrid *chargeMap
05671                        ) {
05672
05673     Vpbe *pbe;
05674     double xmin,
05675            xmax,
05676            ymin,
05677            ymax,
05678            zmin,
05679            zmax,
05680            xlen,
05681            ylen,
05682            zlen,
```

```
05683             hx,
05684             hy,
05685             hzed,
05686             epsw,
05687             epsp,
05688             ionstr;
05689     int i,
05690         nx,
05691         ny,
05692         nz,
05693         islap;
05694     Vrc_Codes rc;
05695
05696     if (thee == VNULL) {
05697         Vnm_print(2, "Vpmg_fillco:  got NULL thee!\n");
05698         return 0;
05699     }
05700
05701     thee->surfMeth = surfMeth;
05702     thee->splineWin = splineWin;
05703     thee->chargeMeth = chargeMeth;
05704     thee->useDielXMap = useDielXMap;
05705     if (thee->useDielXMap) thee->dielXMap = dielXMap;
05706     thee->useDielYMap = useDielYMap;
05707     if (thee->useDielYMap) thee->dielYMap = dielYMap;
05708     thee->useDielZMap = useDielZMap;
05709     if (thee->useDielZMap) thee->dielZMap = dielZMap;
05710     thee->useKappaMap = useKappaMap;
05711     if (thee->useKappaMap) thee->kappaMap = kappaMap;
05712     thee->usePotMap = usePotMap;
05713     if (thee->usePotMap) thee->potMap = potMap;
05714     thee->useChargeMap = useChargeMap;
05715     if (thee->useChargeMap) thee->chargeMap = chargeMap;
05716
05717     /* Get PBE info */
05718     pbe = thee->pbe;
05719     ionstr = Vpbe_getBulkIonicStrength(pbe);
05720     epsw = Vpbe_getSolventDiel(pbe);
05721     epsp = Vpbe_getSoluteDiel(pbe);
05722
05723     /* Mesh info */
05724     nx = thee->pmgp->nx;
05725     ny = thee->pmgp->ny;
05726     nz = thee->pmgp->nz;
05727     hx = thee->pmgp->hx;
05728     hy = thee->pmgp->hy;
05729     hzed = thee->pmgp->hzed;
05730
05731     /* Define the total domain size */
05732     xlen = thee->pmgp->xlen;
05733     ylen = thee->pmgp->ylen;
05734     zlen = thee->pmgp->zlen;
05735
05736     /* Define the min/max dimensions */
05737     xmin = thee->pmgp->xcent - (xlen/2.0);
05738     thee->pmgp->xmin = xmin;
05739     ymin = thee->pmgp->ycent - (ylen/2.0);
05740     thee->pmgp->ymin = ymin;
05741     zmin = thee->pmgp->zcent - (zlen/2.0);
05742     thee->pmgp->zmin = zmin;
05743     xmax = thee->pmgp->xcent + (xlen/2.0);
05744     thee->pmgp->xmax = xmax;
05745     ymax = thee->pmgp->ycent + (ylen/2.0);
05746     thee->pmgp->ymax = ymax;
05747     zmax = thee->pmgp->zcent + (zlen/2.0);
05748     thee->pmgp->zmax = zmax;
05749     thee->rparm[2] = xmin;
05750     thee->rparm[3] = xmax;
05751     thee->rparm[4] = ymin;
05752     thee->rparm[5] = ymax;
05753     thee->rparm[6] = zmin;
05754     thee->rparm[7] = zmax;
05755
05756     /* This is a flag that gets set if the operator is a simple Laplacian;
05757      * i.e., in the case of a homogenous dielectric and zero ionic strength
05758      * The operator cannot be a simple Laplacian if maps are read in. */
05759     if(thee->useDielXMap || thee->useDielYMap || thee->useDielZMap ||
05760        thee->useKappaMap || thee->usePotMap){
05761         islap = 0;
05762     }else if ( (ionstr < VPMGSMALL) && (VABS(epsp-epsw) < VPMGSMALL) ){
05763         islap = 1;
```

```
05764     }else{
05765         islap = 0;
05766     }
05767
05768     /* Fill the mesh point coordinate arrays */
05769     for (i=0; i<nx; i++) thee->xf[i] = xmin + i*hx;
05770     for (i=0; i<ny; i++) thee->yf[i] = ymin + i*hy;
05771     for (i=0; i<nz; i++) thee->zf[i] = zmin + i*hzed;
05772
05773     /* Reset the tcf array */
05774     for (i=0; i<(nx*ny*nz); i++) thee->tcf[i] = 0.0;
05775
05776     /* Fill in the source term (atomic charges) */
05777     Vnm_print(0, "Vpmg_fillco:  filling in source term.\n");
05778     rc = fillcoCharge(thee);
05779     switch(rc) {
05780         case VRC_SUCCESS:
05781             break;
05782         case VRC_WARNING:
05783             Vnm_print(2, "Vpmg_fillco:  non-fatal errors while filling charge map!\n");
05784             break;
05785         case VRC_FAILURE:
05786             Vnm_print(2, "Vpmg_fillco:  fatal errors while filling charge map!\n");
05787             return 0;
05788             break;
05789     }
05790
05791     /* THE FOLLOWING NEEDS TO BE DONE IF WE'RE NOT USING A SIMPLE LAPLACIAN
05792      * OPERATOR */
05793     if (!islap) {
05794         Vnm_print(0, "Vpmg_fillco:  marking ion and solvent accessibility.\n");
05795         fillcoCoef(thee);
05796         Vnm_print(0, "Vpmg_fillco:  done filling coefficient arrays\n");
05797
05798     } else { /* else (!islap) ==> It's a Laplacian operator! */
05799
05800         for (i=0; i<(nx*ny*nz); i++) {
05801             thee->kappa[i] = 0.0;
05802             thee->epsx[i] = epsp;
05803             thee->epsy[i] = epsp;
05804             thee->epsz[i] = epsp;
05805         }
05806
05807     } /* endif (!islap) */
05808
05809     /* Fill the boundary arrays (except when focusing, bcfl = 4) */
05810     if (thee->pmgp->bcfl != BCFL_FOCUS) {
05811         Vnm_print(0, "Vpmg_fillco:  filling boundary arrays\n");
05812         bcCalc(thee);
05813         Vnm_print(0, "Vpmg_fillco:  done filling boundary arrays\n");
05814     }
05815
05816     thee->filled = 1;
05817
05818     return 1;
05819 }
05820
05821
05822 VPUBLIC int Vpmg_force(Vpmg *thee, double *force, int atomID,
05823   Vsurf_Meth srfm, Vchrg_Meth chgm) {
05824
05825     int rc = 1;
05826     double qfF[3];                  /* Charge-field force */
05827     double dbF[3];                  /* Dielectric boundary force */
05828     double ibF[3];                  /* Ion boundary force */
05829     double npF[3];                  /* Non-polar boundary force */
05830
05831     VASSERT(thee != VNULL);
05832
05833     rc = rc && Vpmg_dbForce(thee, qfF, atomID, srfm);
05834     rc = rc && Vpmg_ibForce(thee, dbF, atomID, srfm);
05835     rc = rc && Vpmg_qfForce(thee, ibF, atomID, chgm);
05836
05837     force[0] = qfF[0] + dbF[0] + ibF[0];
05838     force[1] = qfF[1] + dbF[1] + ibF[1];
05839     force[2] = qfF[2] + dbF[2] + ibF[2];
05840
05841     return rc;
05842
05843 }
05844
```

```
05845 VPUBLIC int Vpmg_ibForce(Vpmg *thee, double *force, int atomID,
05846   Vsurf_Meth srfm) {
05847
05848     Valist *alist;
05849     Vacc *acc;
05850     Vpbe *pbe;
05851     Vatom *atom;
05852
05853     double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
05854     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;
05855     double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
05856     double izmagic;
05857     int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
05858
05859     /* For nonlinear forces */
05860     int ichop, nchop, nion, m;
05861     double ionConc[MAXION], ionRadii[MAXION], ionQ[MAXION], ionstr;
05862
05863     VASSERT(thee != VNULL);
05864
05865     acc = thee->pbe->acc;
05866     atom = Valist_getAtom(thee->pbe->alist, atomID);
05867     apos = Vatom_getPosition(atom);
05868     arad = Vatom_getRadius(atom);
05869
05870     /* Reset force */
05871     force[0] = 0.0;
05872     force[1] = 0.0;
05873     force[2] = 0.0;
05874
05875     /* Check surface definition */
05876     if ((srfm != VSM_SPLINE) && (srfm!=VSM_SPLINE3) && (srfm!=VSM_SPLINE4)) {
05877         Vnm_print(2, "Vpmg_ibForce:  Forces *must* be calculated with \
05878 spline-based surfaces!\n");
05879         Vnm_print(2, "Vpmg_ibForce:  Skipping ionic boundary force \
05880 calculation!\n");
05881         return 0;
05882     }
05883
05884     /* If we aren't in the current position, then we're done */
05885     if (atom->partID == 0) return 1;
05886
05887     /* Get PBE info */
05888     pbe = thee->pbe;
05889     acc = pbe->acc;
05890     alist = pbe->alist;
05891     irad = Vpbe_getMaxIonRadius(pbe);
05892     zkappa2 = Vpbe_getZkappa2(pbe);
05893     izmagic = 1.0/Vpbe_getZmagic(pbe);
05894
05895     ionstr = Vpbe_getBulkIonicStrength(pbe);
05896     Vpbe_getIons(pbe, &nion, ionConc, ionRadii, ionQ);
05897
05898     /* Mesh info */
05899     nx = thee->pmgp->nx;
05900     ny = thee->pmgp->ny;
05901     nz = thee->pmgp->nz;
05902     hx = thee->pmgp->hx;
05903     hy = thee->pmgp->hy;
05904     hzed = thee->pmgp->hzed;
05905     xlen = thee->pmgp->xlen;
05906     ylen = thee->pmgp->ylen;
05907     zlen = thee->pmgp->zlen;
05908     xmin = thee->pmgp->xmin;
05909     ymin = thee->pmgp->ymin;
05910     zmin = thee->pmgp->zmin;
05911     xmax = thee->pmgp->xmax;
05912     ymax = thee->pmgp->ymax;
05913     zmax = thee->pmgp->zmax;
05914
05915     /* Sanity check: there is no force if there is zero ionic strength */
05916     if (zkappa2 < VPMGSMALL) {
05917 #ifndef VAPBSQUIET
05918         Vnm_print(2, "Vpmg_ibForce:  No force for zero ionic strength!\n");
05919 #endif
05920         return 1;
05921     }
05922
05923     /* Make sure we're on the grid */
05924     if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
05925       (apos[1]<=ymin) || (apos[1]>=ymax)  || \
```

```
05926       (apos[2]<=zmin) || (apos[2]>=zmax)) {
05927         if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
05928             (thee->pmgp->bcfl != BCFL_MAP)) {
05929             Vnm_print(2, "Vpmg_ibForce:  Atom #%d at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):\n",
05930                   atom, apos[0], apos[1], apos[2]);
05931             Vnm_print(2, "Vpmg_ibForce:    xmin = %g, xmax = %g\n",
05932               xmin, xmax);
05933             Vnm_print(2, "Vpmg_ibForce:    ymin = %g, ymax = %g\n",
05934               ymin, ymax);
05935             Vnm_print(2, "Vpmg_ibForce:    zmin = %g, zmax = %g\n",
05936               zmin, zmax);
05937         }
05938         fflush(stderr);
05939     } else {
05940
05941         /* Convert the atom position to grid reference frame */
05942         position[0] = apos[0] - xmin;
05943         position[1] = apos[1] - ymin;
05944         position[2] = apos[2] - zmin;
05945
05946         /* Integrate over points within this atom's (inflated) radius */
05947         rtot = (irad + arad + thee->splineWin);
05948         rtot2 = VSQR(rtot);
05949         dx = rtot + 0.5*hx;
05950         imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
05951         imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
05952         for (i=imin; i<=imax; i++) {
05953             dx2 = VSQR(position[0] - hx*i);
05954             if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
05955             else dy = 0.5*hy;
05956             jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
05957             jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
05958             for (j=jmin; j<=jmax; j++) {
05959                 dy2 = VSQR(position[1] - hy*j);
05960                 if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
05961                 else dz = 0.5*hzed;
05962                 kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
05963                 kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
05964                 for (k=kmin; k<=kmax; k++) {
05965                     dz2 = VSQR(k*hzed - position[2]);
05966                     /* See if grid point is inside ivdw radius and set kappa
05967                      * accordingly (do spline assignment here) */
05968                     if ((dz2 + dy2 + dx2) <= rtot2) {
05969                         gpos[0] = i*hx + xmin;
05970                         gpos[1] = j*hy + ymin;
05971                         gpos[2] = k*hzed + zmin;
05972
05973                         /* Select the correct function based on the surface definition
05974                          * (now including the 7th order polynomial) */
05975                         Vpmg_splineSelect(srfm,acc, gpos,thee->splineWin, irad, atom, tgrad);
05976
05977                         if (thee->pmgp->nonlin) {
05978                             /* Nonlinear forces */
05979                             fmag = 0.0;
05980                             nchop = 0;
05981                             for (m=0; m<nion; m++) {
05982                                 fmag +=
(thee->kappa[IJK(i,j,k)])*ionConc[m]*(Vcap_exp(-ionQ[m]*thee->u[IJK(i,j,k)], &ichop)-1.0)/ionstr;
05983                                 nchop += ichop;
05984                             }
05985                             /*          if (nchop > 0) Vnm_print(2, "Vpmg_ibForece:  Chopped EXP %d
times!\n", nchop);*/
05986                             force[0] += (zkappa2*fmag*tgrad[0]);
05987                             force[1] += (zkappa2*fmag*tgrad[1]);
05988                             force[2] += (zkappa2*fmag*tgrad[2]);
05989                         } else {
05990                             /* Use of bulk factor (zkappa2) OK here becuase
05991                              * LPBE force approximation */
05992                             /* NAB -- did we forget a kappa factor here??? */
05993                             fmag = VSQR(thee->u[IJK(i,j,k)])*(thee->kappa[IJK(i,j,k)]);
05994                             force[0] += (zkappa2*fmag*tgrad[0]);
05995                             force[1] += (zkappa2*fmag*tgrad[1]);
05996                             force[2] += (zkappa2*fmag*tgrad[2]);
05997                         }
05998                     }
05999                 } /* k loop */
06000             } /* j loop */
06001         } /* i loop */
06002     }
06003     force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
06004     force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
```

```
06005     force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
06006
06007     return 1;
06008 }
06009
06010 VPUBLIC int Vpmg_dbForce(Vpmg *thee, double *dbForce, int atomID,
06011                          Vsurf_Meth srfm) {
06012
06013     Vacc *acc;
06014     Vpbe *pbe;
06015     Vatom *atom;
06016
06017     double *apos, position[3], arad, srad, hx, hy, hzed, izmagic, deps, depsi;
06018     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2, epsp;
06019     double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
06020     double *u, Hxijk, Hyijk, Hzijk, Hxim1jk, Hyijm1k, Hzijkm1;
06021     double dHxijk[3], dHyijk[3], dHzijk[3], dHxim1jk[3], dHyijm1k[3];
06022     double dHzijkm1[3];
06023     int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
06024
06025     VASSERT(thee != VNULL);
06026     if (!thee->filled) {
06027         Vnm_print(2, "Vpmg_dbForce:  Need to callVpmg_fillco!\n");
06028         return 0;
06029     }
06030
06031     acc = thee->pbe->acc;
06032     atom = Valist_getAtom(thee->pbe->alist, atomID);
06033     apos = Vatom_getPosition(atom);
06034     arad = Vatom_getRadius(atom);
06035     srad = Vpbe_getSolventRadius(thee->pbe);
06036
06037     /* Reset force */
06038     dbForce[0] = 0.0;
06039     dbForce[1] = 0.0;
06040     dbForce[2] = 0.0;
06041
06042     /* Check surface definition */
06043     if ((srfm != VSM_SPLINE) && (srfm!=VSM_SPLINE3) && (srfm!=VSM_SPLINE4)) {
06044         Vnm_print(2, "Vpmg_dbForce:  Forces *must* be calculated with \
06045 spline-based surfaces!\n");
06046         Vnm_print(2, "Vpmg_dbForce:  Skipping dielectric/apolar boundary \
06047 force calculation!\n");
06048         return 0;
06049     }
06050
06051
06052     /* If we aren't in the current position, then we're done */
06053     if (atom->partID == 0) return 1;
06054
06055     /* Get PBE info */
06056     pbe = thee->pbe;
06057     acc = pbe->acc;
06058     epsp = Vpbe_getSoluteDiel(pbe);
06059     epsw = Vpbe_getSolventDiel(pbe);
06060     kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*Vunit_kb;
06061     izmagic = 1.0/Vpbe_getZmagic(pbe);
06062
06063     /* Mesh info */
06064     nx = thee->pmgp->nx;
06065     ny = thee->pmgp->ny;
06066     nz = thee->pmgp->nz;
06067     hx = thee->pmgp->hx;
06068     hy = thee->pmgp->hy;
06069     hzed = thee->pmgp->hzed;
06070     xlen = thee->pmgp->xlen;
06071     ylen = thee->pmgp->ylen;
06072     zlen = thee->pmgp->zlen;
06073     xmin = thee->pmgp->xmin;
06074     ymin = thee->pmgp->ymin;
06075     zmin = thee->pmgp->zmin;
06076     xmax = thee->pmgp->xmax;
06077     ymax = thee->pmgp->ymax;
06078     zmax = thee->pmgp->zmax;
06079     u = thee->u;
06080
06081     /* Sanity check: there is no force if there is zero ionic strength */
06082     if (VABS(epsp-epsw) < VPMGSMALL) {
06083         Vnm_print(0, "Vpmg_dbForce: No force for uniform dielectric!\n");
06084         return 1;
06085     }
```

```
06086     deps = (epsw - epsp);
06087     depsi = 1.0/deps;
06088     rtot = (arad + thee->splineWin + srad);
06089
06090     /* Make sure we're on the grid */
06091     /* Grid checking modified by Matteo Rotter */
06092     if ((apos[0]<=xmin + rtot) || (apos[0]>=xmax - rtot)  || \
06093         (apos[1]<=ymin + rtot) || (apos[1]>=ymax - rtot)  || \
06094         (apos[2]<=zmin + rtot) || (apos[2]>=zmax - rtot)) {
06095         if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
06096             (thee->pmgp->bcfl != BCFL_MAP)) {
06097             Vnm_print(2, "Vpmg_dbForce:  Atom #%d at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):\n",
06098                       atomID, apos[0], apos[1], apos[2]);
06099             Vnm_print(2, "Vpmg_dbForce:    xmin = %g, xmax = %g\n",
06100                       xmin, xmax);
06101             Vnm_print(2, "Vpmg_dbForce:    ymin = %g, ymax = %g\n",
06102                       ymin, ymax);
06103             Vnm_print(2, "Vpmg_dbForce:    zmin = %g, zmax = %g\n",
06104                       zmin, zmax);
06105         }
06106         fflush(stderr);
06107     } else {
06108
06109         /* Convert the atom position to grid reference frame */
06110         position[0] = apos[0] - xmin;
06111         position[1] = apos[1] - ymin;
06112         position[2] = apos[2] - zmin;
06113
06114         /* Integrate over points within this atom's (inflated) radius */
06115         rtot2 = VSQR(rtot);
06116         dx = rtot/hx;
06117         imin = (int)floor((position[0]-rtot)/hx);
06118         if (imin < 1) {
06119             Vnm_print(2, "Vpmg_dbForce:  Atom %d off grid!\n", atomID);
06120             return 0;
06121         }
06122         imax = (int)ceil((position[0]+rtot)/hx);
06123         if (imax > (nx-2)) {
06124             Vnm_print(2, "Vpmg_dbForce:  Atom %d off grid!\n", atomID);
06125             return 0;
06126         }
06127         jmin = (int)floor((position[1]-rtot)/hy);
06128         if (jmin < 1) {
06129             Vnm_print(2, "Vpmg_dbForce:  Atom %d off grid!\n", atomID);
06130             return 0;
06131         }
06132         jmax = (int)ceil((position[1]+rtot)/hy);
06133         if (jmax > (ny-2)) {
06134             Vnm_print(2, "Vpmg_dbForce:  Atom %d off grid!\n", atomID);
06135             return 0;
06136         }
06137         kmin = (int)floor((position[2]-rtot)/hzed);
06138         if (kmin < 1) {
06139             Vnm_print(2, "Vpmg_dbForce:  Atom %d off grid!\n", atomID);
06140             return 0;
06141         }
06142         kmax = (int)ceil((position[2]+rtot)/hzed);
06143         if (kmax > (nz-2)) {
06144             Vnm_print(2, "Vpmg_dbForce:  Atom %d off grid!\n", atomID);
06145             return 0;
06146         }
06147         for (i=imin; i<=imax; i++) {
06148             for (j=jmin; j<=jmax; j++) {
06149                 for (k=kmin; k<=kmax; k++) {
06150                     /* i,j,k */
06151                     gpos[0] = (i+0.5)*hx + xmin;
06152                     gpos[1] = j*hy + ymin;
06153                     gpos[2] = k*hzed + zmin;
06154                     Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
06155
06156                     /* Select the correct function based on the surface definition
06157                         * (now including the 7th order polynomial) */
06158                     Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin, 0.,atom, dHxijk);
06159                     /*
06160                      switch (srfm) {
06161                          case VSM_SPLINE :
06162                              Vacc_splineAccGradAtomNorm(acc, gpos, thee->splineWin, 0.,
06163                                                         atom, dHxijk);
06164                              break;
06165                          case VSM_SPLINE4 :
06166                              Vacc_splineAccGradAtomNorm4(acc, gpos, thee->splineWin, 0.,
```

```
06167                                                   atom, dHxijk);
06168                 break;
06169             default:
06170                 Vnm_print(2, "Vpmg_dbnbForce: Unknown surface method.\n");
06171                 return;
06172          }
06173          */
06174         for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
06175         gpos[0] = i*hx + xmin;
06176         gpos[1] = (j+0.5)*hy + ymin;
06177         gpos[2] = k*hzed + zmin;
06178         Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
06179
06180         /* Select the correct function based on the surface definition
06181             * (now including the 7th order polynomial) */
06182         Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin, 0.,atom, dHyijk);
06183
06184         for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
06185         gpos[0] = i*hx + xmin;
06186         gpos[1] = j*hy + ymin;
06187         gpos[2] = (k+0.5)*hzed + zmin;
06188         Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
06189
06190         /* Select the correct function based on the surface definition
06191             * (now including the 7th order polynomial) */
06192         Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin, 0.,atom, dHzijk);
06193
06194         for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
06195         /* i-1,j,k */
06196         gpos[0] = (i-0.5)*hx + xmin;
06197         gpos[1] = j*hy + ymin;
06198         gpos[2] = k*hzed + zmin;
06199         Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
06200
06201         /* Select the correct function based on the surface definition
06202             * (now including the 7th order polynomial) */
06203         Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin, 0.,atom, dHxim1jk);
06204
06205         for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
06206         /* i,j-1,k */
06207         gpos[0] = i*hx + xmin;
06208         gpos[1] = (j-0.5)*hy + ymin;
06209         gpos[2] = k*hzed + zmin;
06210         Hyijm1k = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
06211
06212         /* Select the correct function based on the surface definition
06213             * (now including the 7th order polynomial) */
06214         Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin, 0.,atom, dHyijm1k);
06215
06216         for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;
06217         /* i,j,k-1 */
06218         gpos[0] = i*hx + xmin;
06219         gpos[1] = j*hy + ymin;
06220         gpos[2] = (k-0.5)*hzed + zmin;
06221         Hzijkm1 = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
06222
06223         /* Select the correct function based on the surface definition
06224             * (now including the 7th order polynomial) */
06225         Vpmg_splineSelect(srfm,acc, gpos, thee->splineWin, 0.,atom, dHzijkm1);
06226
06227         for (l=0; l<3; l++) dHzijkm1[l] *= Hzijkm1;
06228         /* *** CALCULATE DIELECTRIC BOUNDARY FORCES *** */
06229         dbFmag = u[IJK(i,j,k)];
06230         tgrad[0] =
06231             (dHxijk[0]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
06232              +  dHxim1jk[0]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
06233             + (dHyijk[0]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
06234                +  dHyijm1k[0]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
06235             + (dHzijk[0]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
06236                + dHzijkm1[0]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
06237         tgrad[1] =
06238             (dHxijk[1]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
06239              +  dHxim1jk[1]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
06240             + (dHyijk[1]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
06241                +  dHyijm1k[1]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
06242             + (dHzijk[1]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
06243                + dHzijkm1[1]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
06244         tgrad[2] =
06245             (dHxijk[2]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
06246              +  dHxim1jk[2]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
06247             + (dHyijk[2]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
```

```
06248                           +  dHyijm1k[2]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
06249                       + (dHzijk[2]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
06250                           + dHzijkm1[2]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
06251                   dbForce[0] += (dbFmag*tgrad[0]);
06252                   dbForce[1] += (dbFmag*tgrad[1]);
06253                   dbForce[2] += (dbFmag*tgrad[2]);
06254
06255               } /* k loop */
06256           } /* j loop */
06257       } /* i loop */
06258
06259       dbForce[0] = -dbForce[0]*hx*hy*hzed*deps*0.5*izmagic;
06260       dbForce[1] = -dbForce[1]*hx*hy*hzed*deps*0.5*izmagic;
06261       dbForce[2] = -dbForce[2]*hx*hy*hzed*deps*0.5*izmagic;
06262   }
06263
06264   return 1;
06265 }
06266
06267 VPUBLIC int Vpmg_qfForce(Vpmg *thee, double *force, int atomID,
06268   Vchrg_Meth chgm) {
06269
06270   double tforce[3];
06271
06272   /* Reset force */
06273   force[0] = 0.0;
06274   force[1] = 0.0;
06275   force[2] = 0.0;
06276
06277   /* Check surface definition */
06278   if (chgm != VCM_BSPL2) {
06279       Vnm_print(2, "Vpmg_qfForce:  It is recommended that forces be \
06280 calculated with the\n");
06281       Vnm_print(2, "Vpmg_qfForce:  cubic spline charge discretization \
06282 scheme\n");
06283   }
06284
06285   switch (chgm) {
06286       case VCM_TRIL:
06287           qfForceSpline1(thee, tforce, atomID);
06288           break;
06289       case VCM_BSPL2:
06290           qfForceSpline2(thee, tforce, atomID);
06291           break;
06292       case VCM_BSPL4:
06293           qfForceSpline4(thee, tforce, atomID);
06294           break;
06295       default:
06296           Vnm_print(2, "Vpmg_qfForce:  Undefined charge discretization \
06297 method (%d)!\n", chgm);
06298           Vnm_print(2, "Vpmg_qfForce:  Forces not calculated!\n");
06299           return 0;
06300   }
06301
06302   /* Assign forces */
06303   force[0] = tforce[0];
06304   force[1] = tforce[1];
06305   force[2] = tforce[2];
06306
06307   return 1;
06308 }
06309
06310
06311 VPRIVATE void qfForceSpline1(Vpmg *thee, double *force, int atomID) {
06312
06313   Vatom *atom;
06314
06315   double *apos, position[3], hx, hy, hzed;
06316   double xmin, ymin, zmin, xmax, ymax, zmax;
06317   double dx, dy, dz;
06318   double *u, charge, ifloat, jfloat, kfloat;
06319   int nx, ny, nz, ihi, ilo, jhi, jlo, khi, klo;
06320
06321   VASSERT(thee != VNULL);
06322
06323   atom = Valist_getAtom(thee->pbe->alist, atomID);
06324   apos = Vatom_getPosition(atom);
06325   charge = Vatom_getCharge(atom);
06326
06327   /* Reset force */
06328   force[0] = 0.0;
```

```
06329     force[1] = 0.0;
06330     force[2] = 0.0;
06331
06332     /* If we aren't in the current position, then we're done */
06333     if (atom->partID == 0) return;
06334
06335     /* Mesh info */
06336     nx = thee->pmgp->nx;
06337     ny = thee->pmgp->ny;
06338     nz = thee->pmgp->nz;
06339     hx = thee->pmgp->hx;
06340     hy = thee->pmgp->hy;
06341     hzed = thee->pmgp->hzed;
06342     xmin = thee->pmgp->xmin;
06343     ymin = thee->pmgp->ymin;
06344     zmin = thee->pmgp->zmin;
06345     xmax = thee->pmgp->xmax;
06346     ymax = thee->pmgp->ymax;
06347     zmax = thee->pmgp->zmax;
06348     u = thee->u;
06349
06350     /* Make sure we're on the grid */
06351     if ((apos[0]<=xmin) || (apos[0]>=xmax) || (apos[1]<=ymin) || \
06352         (apos[1]>=ymax) || (apos[2]<=zmin) || (apos[2]>=zmax)) {
06353         if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
06354             (thee->pmgp->bcfl != BCFL_MAP)) {
06355             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]);
06356             Vnm_print(2, "Vpmg_qfForce:    xmin = %g, xmax = %g\n", xmin, xmax);
06357             Vnm_print(2, "Vpmg_qfForce:    ymin = %g, ymax = %g\n", ymin, ymax);
06358             Vnm_print(2, "Vpmg_qfForce:    zmin = %g, zmax = %g\n", zmin, zmax);
06359         }
06360         fflush(stderr);
06361     } else {
06362
06363         /* Convert the atom position to grid coordinates */
06364         position[0] = apos[0] - xmin;
06365         position[1] = apos[1] - ymin;
06366         position[2] = apos[2] - zmin;
06367         ifloat = position[0]/hx;
06368         jfloat = position[1]/hy;
06369         kfloat = position[2]/hzed;
06370         ihi = (int)ceil(ifloat);
06371         ilo = (int)floor(ifloat);
06372         jhi = (int)ceil(jfloat);
06373         jlo = (int)floor(jfloat);
06374         khi = (int)ceil(kfloat);
06375         klo = (int)floor(kfloat);
06376         VASSERT((ihi < nx) && (ihi >=0));
06377         VASSERT((ilo < nx) && (ilo >=0));
06378         VASSERT((jhi < ny) && (jhi >=0));
06379         VASSERT((jlo < ny) && (jlo >=0));
06380         VASSERT((khi < nz) && (khi >=0));
06381         VASSERT((klo < nz) && (klo >=0));
06382         dx = ifloat - (double)(ilo);
06383         dy = jfloat - (double)(jlo);
06384         dz = kfloat - (double)(klo);
06385
06386
06387 #if 0
06388         Vnm_print(1, "Vpmg_qfForce: (DEBUG) u ~ %g\n",
06389           dx    *dy    *dz    *u[IJK(ihi,jhi,khi)]
06390          +dx    *dy    *(1-dz)*u[IJK(ihi,jhi,klo)]
06391          +dx    *(1-dy)*dz    *u[IJK(ihi,jlo,khi)]
06392          +dx    *(1-dy)*(1-dz)*u[IJK(ihi,jlo,klo)]
06393          +(1-dx)*dy    *dz    *u[IJK(ilo,jhi,khi)]
06394          +(1-dx)*dy    *(1-dz)*u[IJK(ilo,jhi,klo)]
06395          +(1-dx)*(1-dy)*dz    *u[IJK(ilo,jlo,khi)]
06396          +(1-dx)*(1-dy)*(1-dz)*u[IJK(ilo,jlo,klo)]);
06397 #endif
06398
06399
06400         if ((dx > VPMGSMALL) && (VABS(1.0-dx) > VPMGSMALL)) {
06401             force[0] =
06402               -charge*(dy    *dz    *u[IJK(ihi,jhi,khi)]
06403                      + dy    *(1-dz)*u[IJK(ihi,jhi,klo)]
06404                      + (1-dy)*dz    *u[IJK(ihi,jlo,khi)]
06405                      + (1-dy)*(1-dz)*u[IJK(ihi,jlo,klo)]
06406                      - dy    *dz    *u[IJK(ilo,jhi,khi)]
06407                      - dy    *(1-dz)*u[IJK(ilo,jhi,klo)]
06408                      - (1-dy)*dz    *u[IJK(ilo,jlo,khi)]
```

```
06409                       - (1-dy)*(1-dz)*u[IJK(ilo,jlo,klo)])/hx;
06410         } else {
06411             force[0] = 0;
06412             Vnm_print(0,
06413               "Vpmg_qfForce:  Atom %d on x gridline; zero x-force\n", atomID);
06414         }
06415         if ((dy > VPMGSMALL) && (VABS(1.0-dy) > VPMGSMALL)) {
06416             force[1] =
06417               -charge*(dx    *dz    *u[IJK(ihi,jhi,khi)]
06418                      + dx    *(1-dz)*u[IJK(ihi,jhi,klo)]
06419                      - dx    *dz    *u[IJK(ihi,jlo,khi)]
06420                      - dx    *(1-dz)*u[IJK(ihi,jlo,klo)]
06421                      + (1-dx)*dz    *u[IJK(ilo,jhi,khi)]
06422                      + (1-dx)*(1-dz)*u[IJK(ilo,jhi,klo)]
06423                      - (1-dx)*dz    *u[IJK(ilo,jlo,khi)]
06424                      - (1-dx)*(1-dz)*u[IJK(ilo,jlo,klo)])/hy;
06425         } else {
06426             force[1] = 0;
06427             Vnm_print(0,
06428               "Vpmg_qfForce:  Atom %d on y gridline; zero y-force\n", atomID);
06429         }
06430         if ((dz > VPMGSMALL) && (VABS(1.0-dz) > VPMGSMALL)) {
06431             force[2] =
06432               -charge*(dy    *dx    *u[IJK(ihi,jhi,khi)]
06433                      - dy    *dx    *u[IJK(ihi,jhi,klo)]
06434                      + (1-dy)*dx    *u[IJK(ihi,jlo,khi)]
06435                      - (1-dy)*dx    *u[IJK(ihi,jlo,klo)]
06436                      + dy    *(1-dx)*u[IJK(ilo,jhi,khi)]
06437                      - dy    *(1-dx)*u[IJK(ilo,jhi,klo)]
06438                      + (1-dy)*(1-dx)*u[IJK(ilo,jlo,khi)]
06439                      - (1-dy)*(1-dx)*u[IJK(ilo,jlo,klo)])/hzed;
06440         } else {
06441             force[2] = 0;
06442             Vnm_print(0,
06443               "Vpmg_qfForce:  Atom %d on z gridline; zero z-force\n", atomID);
06444         }
06445     }
06446 }
06447
06448 VPRIVATE void qfForceSpline2(Vpmg *thee, double *force, int atomID) {
06449
06450     Vatom *atom;
06451
06452     double *apos, position[3], hx, hy, hzed;
06453     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
06454     double mx, my, mz, dmx, dmy, dmz;
06455     double *u, charge, ifloat, jfloat, kfloat;
06456     int nx, ny, nz, im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1;
06457     int kp1, kp2, ii, jj, kk;
06458
06459     VASSERT(thee != VNULL);
06460
06461     atom = Valist_getAtom(thee->pbe->alist, atomID);
06462     apos = Vatom_getPosition(atom);
06463     charge = Vatom_getCharge(atom);
06464
06465     /* Reset force */
06466     force[0] = 0.0;
06467     force[1] = 0.0;
06468     force[2] = 0.0;
06469
06470     /* If we aren't in the current position, then we're done */
06471     if (atom->partID == 0) return;
06472
06473     /* Mesh info */
06474     nx = thee->pmgp->nx;
06475     ny = thee->pmgp->ny;
06476     nz = thee->pmgp->nz;
06477     hx = thee->pmgp->hx;
06478     hy = thee->pmgp->hy;
06479     hzed = thee->pmgp->hzed;
06480     xlen = thee->pmgp->xlen;
06481     ylen = thee->pmgp->ylen;
06482     zlen = thee->pmgp->zlen;
06483     xmin = thee->pmgp->xmin;
06484     ymin = thee->pmgp->ymin;
06485     zmin = thee->pmgp->zmin;
06486     xmax = thee->pmgp->xmax;
06487     ymax = thee->pmgp->ymax;
06488     zmax = thee->pmgp->zmax;
06489     u = thee->u;
```

```
06490
06491     /* Make sure we're on the grid */
06492     if ((apos[0]<=(xmin+hx))   || (apos[0]>=(xmax-hx)) \
06493      || (apos[1]<=(ymin+hy))   || (apos[1]>=(ymax-hy)) \
06494      || (apos[2]<=(zmin+hzed)) || (apos[2]>=(zmax-hzed))) {
06495         if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
06496             (thee->pmgp->bcfl != BCFL_MAP)) {
06497             Vnm_print(2, "qfForceSpline2:  Atom #%d off the mesh \
06498                 (ignoring)\n", atomID);
06499         }
06500         fflush(stderr);
06501
06502     } else {
06503
06504         /* Convert the atom position to grid coordinates */
06505         position[0] = apos[0] - xmin;
06506         position[1] = apos[1] - ymin;
06507         position[2] = apos[2] - zmin;
06508         ifloat = position[0]/hx;
06509         jfloat = position[1]/hy;
06510         kfloat = position[2]/hzed;
06511         ip1 = (int)ceil(ifloat);
06512         ip2 = ip1 + 1;
06513         im1 = (int)floor(ifloat);
06514         im2 = im1 - 1;
06515         jp1 = (int)ceil(jfloat);
06516         jp2 = jp1 + 1;
06517         jm1 = (int)floor(jfloat);
06518         jm2 = jm1 - 1;
06519         kp1 = (int)ceil(kfloat);
06520         kp2 = kp1 + 1;
06521         km1 = (int)floor(kfloat);
06522         km2 = km1 - 1;
06523
06524         /* This step shouldn't be necessary, but it saves nasty debugging
06525          * later on if something goes wrong */
06526         ip2 = VMIN2(ip2,nx-1);
06527         ip1 = VMIN2(ip1,nx-1);
06528         im1 = VMAX2(im1,0);
06529         im2 = VMAX2(im2,0);
06530         jp2 = VMIN2(jp2,ny-1);
06531         jp1 = VMIN2(jp1,ny-1);
06532         jm1 = VMAX2(jm1,0);
06533         jm2 = VMAX2(jm2,0);
06534         kp2 = VMIN2(kp2,nz-1);
06535         kp1 = VMIN2(kp1,nz-1);
06536         km1 = VMAX2(km1,0);
06537         km2 = VMAX2(km2,0);
06538
06539
06540         for (ii=im2; ii<=ip2; ii++) {
06541             mx = bspline2(VFCHI(ii,ifloat));
06542             dmx = dbspline2(VFCHI(ii,ifloat));
06543             for (jj=jm2; jj<=jp2; jj++) {
06544                 my = bspline2(VFCHI(jj,jfloat));
06545                 dmy = dbspline2(VFCHI(jj,jfloat));
06546                 for (kk=km2; kk<=kp2; kk++) {
06547                     mz = bspline2(VFCHI(kk,kfloat));
06548                     dmz = dbspline2(VFCHI(kk,kfloat));
06549
06550                     force[0] += (charge*dmx*my*mz*u[IJK(ii,jj,kk)])/hx;
06551                     force[1] += (charge*mx*dmy*mz*u[IJK(ii,jj,kk)])/hy;
06552                     force[2] += (charge*mx*my*dmz*u[IJK(ii,jj,kk)])/hzed;
06553
06554                 }
06555             }
06556         }
06557
06558     }
06559 }
06560
06561 VPRIVATE void qfForceSpline4(Vpmg *thee, double *force, int atomID) {
06562
06563     Vatom *atom;
06564     double f, c, *u, *apos, position[3];
06565
06566     /* Grid variables */
06567     int nx,ny,nz;
06568     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
06569     double hx, hy, hzed, ifloat, jfloat, kfloat;
06570
```

```
06571     /* B-spline weights */
06572     double mx, my, mz, dmx, dmy, dmz;
06573     double mi, mj, mk;
06574
06575     /* Loop indeces */
06576     int i, j, k, ii, jj, kk;
06577     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
06578
06579     /* field */
06580     double e[3];
06581
06582     VASSERT(thee != VNULL);
06583     VASSERT(thee->filled);
06584
06585     atom = Valist_getAtom(thee->pbe->alist, atomID);
06586     apos = Vatom_getPosition(atom);
06587     c = Vatom_getCharge(atom);
06588
06589     for (i=0;i<3;i++){
06590         e[i] = 0.0;
06591     }
06592
06593     /* Mesh info */
06594     nx = thee->pmgp->nx;
06595     ny = thee->pmgp->ny;
06596     nz = thee->pmgp->nz;
06597     hx = thee->pmgp->hx;
06598     hy = thee->pmgp->hy;
06599     hzed = thee->pmgp->hzed;
06600     xlen = thee->pmgp->xlen;
06601     ylen = thee->pmgp->ylen;
06602     zlen = thee->pmgp->zlen;
06603     xmin = thee->pmgp->xmin;
06604     ymin = thee->pmgp->ymin;
06605     zmin = thee->pmgp->zmin;
06606     xmax = thee->pmgp->xmax;
06607     ymax = thee->pmgp->ymax;
06608     zmax = thee->pmgp->zmax;
06609     u = thee->u;
06610
06611     /* Make sure we're on the grid */
06612     if ((apos[0]<=(xmin+2*hx))   || (apos[0]>=(xmax-2*hx)) \
06613         || (apos[1]<=(ymin+2*hy))   || (apos[1]>=(ymax-2*hy)) \
06614         || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
06615         Vnm_print(2, "qfForceSpline4:  Atom off the mesh \
06616             (ignoring) %6.3f %6.3f %6.3f\n", apos[0], apos[1], apos[2]);
06617         fflush(stderr);
06618     } else {
06619
06620         /* Convert the atom position to grid coordinates */
06621         position[0] = apos[0] - xmin;
06622         position[1] = apos[1] - ymin;
06623         position[2] = apos[2] - zmin;
06624         ifloat = position[0]/hx;
06625         jfloat = position[1]/hy;
06626         kfloat = position[2]/hzed;
06627         ip1 = (int)ceil(ifloat);
06628         ip2 = ip1 + 2;
06629         im1 = (int)floor(ifloat);
06630         im2 = im1 - 2;
06631         jp1 = (int)ceil(jfloat);
06632         jp2 = jp1 + 2;
06633         jm1 = (int)floor(jfloat);
06634         jm2 = jm1 - 2;
06635         kp1 = (int)ceil(kfloat);
06636         kp2 = kp1 + 2;
06637         km1 = (int)floor(kfloat);
06638         km2 = km1 - 2;
06639
06640         /* This step shouldn't be necessary, but it saves nasty debugging
06641          * later on if something goes wrong */
06642         ip2 = VMIN2(ip2,nx-1);
06643         ip1 = VMIN2(ip1,nx-1);
06644         im1 = VMAX2(im1,0);
06645         im2 = VMAX2(im2,0);
06646         jp2 = VMIN2(jp2,ny-1);
06647         jp1 = VMIN2(jp1,ny-1);
06648         jm1 = VMAX2(jm1,0);
06649         jm2 = VMAX2(jm2,0);
06650         kp2 = VMIN2(kp2,nz-1);
06651         kp1 = VMIN2(kp1,nz-1);
```

```
06652         km1 = VMAX2(km1,0);
06653         km2 = VMAX2(km2,0);
06654
06655         for (ii=im2; ii<=ip2; ii++) {
06656             mi = VFCHI4(ii,ifloat);
06657             mx = bspline4(mi);
06658             dmx = dbspline4(mi);
06659             for (jj=jm2; jj<=jp2; jj++) {
06660                 mj = VFCHI4(jj,jfloat);
06661                 my = bspline4(mj);
06662                 dmy = dbspline4(mj);
06663                 for (kk=km2; kk<=kp2; kk++) {
06664                     mk = VFCHI4(kk,kfloat);
06665                     mz = bspline4(mk);
06666                     dmz = dbspline4(mk);
06667                     f = u[IJK(ii,jj,kk)];
06668                     /* Field */
06669                     e[0] += f*dmx*my*mz/hx;
06670                     e[1] += f*mx*dmy*mz/hy;
06671                     e[2] += f*mx*my*dmz/hzed;
06672                 }
06673             }
06674         }
06675     }
06676
06677     /* Monopole Force */
06678     force[0] = e[0]*c;
06679     force[1] = e[1]*c;
06680     force[2] = e[2]*c;
06681
06682 }
06683
06684 VPRIVATE void markFrac(
06685         double rtot, double *tpos,
06686         int nx, int ny, int nz,
06687         double hx, double hy, double hzed,
06688         double xmin, double ymin, double zmin,
06689         double *xarray, double *yarray, double *zarray) {
06690
06691     int i, j, k, imin, imax, jmin, jmax, kmin, kmax;
06692     double dx, dx2, dy, dy2, dz, dz2, a000, a001, a010, a100, r2;
06693     double x, xp, xm, y, yp, ym, zp, z, zm, xspan, yspan, zspan;
06694     double rtot2, pos[3];
06695
06696     /* Convert to grid reference frame */
06697     pos[0] = tpos[0] - xmin;
06698     pos[1] = tpos[1] - ymin;
06699     pos[2] = tpos[2] - zmin;
06700
06701     rtot2 = VSQR(rtot);
06702
06703     xspan = rtot + 2*hx;
06704     imin = VMAX2(0, (int)ceil((pos[0] - xspan)/hx));
06705     imax = VMIN2(nx-1, (int)floor((pos[0] + xspan)/hx));
06706     for (i=imin; i<=imax; i++) {
06707         x = hx*i;
06708         dx2 = VSQR(pos[0] - x);
06709         if (rtot2 > dx2) {
06710             yspan = VSQRT(rtot2 - dx2) + 2*hy;
06711         } else {
06712             yspan = 2*hy;
06713         }
06714         jmin = VMAX2(0,(int)ceil((pos[1] - yspan)/hy));
06715         jmax = VMIN2(ny-1,(int)floor((pos[1] + yspan)/hy));
06716         for (j=jmin; j<=jmax; j++) {
06717             y = hy*j;
06718             dy2 = VSQR(pos[1] - y);
06719             if (rtot2 > (dx2+dy2)) {
06720                 zspan = VSQRT(rtot2-dx2-dy2) + 2*hzed;
06721             } else {
06722                 zspan = 2*hzed;
06723             }
06724             kmin = VMAX2(0,(int)ceil((pos[2] - zspan)/hzed));
06725             kmax = VMIN2(nz-1,(int)floor((pos[2] + zspan)/hzed));
06726             for (k=kmin; k<=kmax; k++) {
06727                 z = hzed*k;
06728                 dz2 = VSQR(pos[2] - z);
06729
06730                 r2 = dx2 + dy2 + dz2;
06731
06732                 /* We need to determine the inclusion value a000 at (i,j,k) */
```

```
06733                 if (r2 < rtot2) a000 = 1.0;
06734                 else a000 = 0.0;
06735
06736                 /* We need to evaluate the values of x which intersect the
06737                  * sphere and determine if these are in the interval
06738                  * [(i,j,k), (i+1,j,k)] */
06739                 if (r2 < (rtot2 - hx*hx)) a100 = 1.0;
06740                 else if (r2 > (rtot2 + hx*hx)) a100 = 0.0;
06741                 else if (rtot2 > (dy2 + dz2)) {
06742                     dx = VSQRT(rtot2 - dy2 - dz2);
06743                     xm = pos[0] - dx;
06744                     xp = pos[0] + dx;
06745                     if ((xm < x+hx) && (xm > x)) {
06746                         a100 = (xm - x)/hx;
06747                     } else if ((xp < x+hx) && (xp > x)) {
06748                         a100 = (xp - x)/hx;
06749                     }
06750                 } else a100 = 0.0;
06751
06752                 /* We need to evaluate the values of y which intersect the
06753                  * sphere and determine if these are in the interval
06754                  * [(i,j,k), (i,j+1,k)] */
06755                 if (r2 < (rtot2 - hy*hy)) a010 = 1.0;
06756                 else if (r2 > (rtot2 + hy*hy)) a010 = 0.0;
06757                 else if (rtot2 > (dx2 + dz2)) {
06758                     dy = VSQRT(rtot2 - dx2 - dz2);
06759                     ym = pos[1] - dy;
06760                     yp = pos[1] + dy;
06761                     if ((ym < y+hy) && (ym > y)) {
06762                         a010 = (ym - y)/hy;
06763                     } else if ((yp < y+hy) && (yp > y)) {
06764                         a010 = (yp - y)/hy;
06765                     }
06766                 } else a010 = 0.0;
06767
06768                 /* We need to evaluate the values of y which intersect the
06769                  * sphere and determine if these are in the interval
06770                  * [(i,j,k), (i,j,k+1)] */
06771                 if (r2 < (rtot2 - hzed*hzed)) a001 = 1.0;
06772                 else if (r2 > (rtot2 + hzed*hzed)) a001 = 0.0;
06773                 else if (rtot2 > (dx2 + dy2)) {
06774                     dz = VSQRT(rtot2 - dx2 - dy2);
06775                     zm = pos[2] - dz;
06776                     zp = pos[2] + dz;
06777                     if ((zm < z+hzed) && (zm > z)) {
06778                         a001 = (zm - z)/hzed;
06779                     } else if ((zp < z+hzed) && (zp > z)) {
06780                         a001 = (zp - z)/hzed;
06781                     }
06782                 } else a001 = 0.0;
06783
06784                 if (a100 < xarray[IJK(i,j,k)]) xarray[IJK(i,j,k)] = a100;
06785                 if (a010 < yarray[IJK(i,j,k)]) yarray[IJK(i,j,k)] = a010;
06786                 if (a001 < zarray[IJK(i,j,k)]) zarray[IJK(i,j,k)] = a001;
06787
06788             } /* k loop */
06789         } /* j loop */
06790     } /* i loop */
06791 }
06792
06793 /*
06794
06795  NOTE: This is the original version of the markSphere function. It's in here
06796  for reference and in case a reversion to the original code is needed.
06797  D. Gohara (2/14/08)
06798  */
06799 /*
06800 VPRIVATE void markSphere(
06801                          double rtot, double *tpos,
06802                          int nx, int ny, int nz,
06803                          double hx, double hy, double hzed,
06804                          double xmin, double ymin, double zmin,
06805                          double *array, double markVal) {
06806
06807     int i, j, k, imin, imax, jmin, jmax, kmin, kmax;
06808     double dx, dx2, dy, dy2, dz, dz2;
06809     double rtot2, pos[3];
06810
06811     // Convert to grid reference frame
06812     pos[0] = tpos[0] - xmin;
06813     pos[1] = tpos[1] - ymin;
```

```
06814     pos[2] = tpos[2] - zmin;
06815
06816     rtot2 = VSQR(rtot);
06817
06818     dx = rtot + 0.5*hx;
06819     imin = VMAX2(0,(int)ceil((pos[0] - dx)/hx));
06820     imax = VMIN2(nx-1,(int)floor((pos[0] + dx)/hx));
06821     for (i=imin; i<=imax; i++) {
06822         dx2 = VSQR(pos[0] - hx*i);
06823         if (rtot2 > dx2) {
06824             dy = VSQRT(rtot2 - dx2) + 0.5*hy;
06825         } else {
06826             dy = 0.5*hy;
06827         }
06828         jmin = VMAX2(0,(int)ceil((pos[1] - dy)/hy));
06829         jmax = VMIN2(ny-1,(int)floor((pos[1] + dy)/hy));
06830         for (j=jmin; j<=jmax; j++) {
06831             dy2 = VSQR(pos[1] - hy*j);
06832             if (rtot2 > (dx2+dy2)) {
06833                 dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
06834             } else {
06835                 dz = 0.5*hzed;
06836             }
06837             kmin = VMAX2(0,(int)ceil((pos[2] - dz)/hzed));
06838             kmax = VMIN2(nz-1,(int)floor((pos[2] + dz)/hzed));
06839             for (k=kmin; k<=kmax; k++) {
06840                 dz2 = VSQR(k*hzed - pos[2]);
06841                 if ((dz2 + dy2 + dx2) <= rtot2) {
06842                     array[IJK(i,j,k)] = markVal;
06843                 }
06844             } // k loop
06845         } // j loop
06846     } // i loop
06847 }
06848 */
06849 VPRIVATE void markSphere(double rtot, double *tpos,
06850                          int nx, int ny, int nz,
06851                          double hx, double hy, double hz,
06852                          double xmin, double ymin, double zmin,
06853                          double *array, double markVal) {
06854
06855     int i, j, k;
06856     double fi,fj,fk;
06857     int imin, imax;
06858     int jmin, jmax;
06859     int kmin, kmax;
06860     double dx2, dy2, dz2;
06861     double xrange, yrange, zrange;
06862     double rtot2, posx, posy, posz;
06863
06864     /* Convert to grid reference frame */
06865     posx = tpos[0] - xmin;
06866     posy = tpos[1] - ymin;
06867     posz = tpos[2] - zmin;
06868
06869     rtot2 = VSQR(rtot);
06870
06871     xrange = rtot + 0.5 * hx;
06872     yrange = rtot + 0.5 * hy;
06873     zrange = rtot + 0.5 * hz;
06874
06875     imin = VMAX2(0, (int)ceil((posx - xrange)/hx));
06876     jmin = VMAX2(0, (int)ceil((posy - yrange)/hy));
06877     kmin = VMAX2(0, (int)ceil((posz - zrange)/hz));
06878
06879     imax = VMIN2(nx-1, (int)floor((posx + xrange)/hx));
06880     jmax = VMIN2(ny-1, (int)floor((posy + yrange)/hy));
06881     kmax = VMIN2(nz-1, (int)floor((posz + zrange)/hz));
06882
06883     for (i=imin,fi=imin; i<=imax; i++, fi+=1.) {
06884         dx2 = VSQR(posx - hx*fi);
06885         for (j=jmin,fj=jmin; j<=jmax; j++, fj+=1.) {
06886             dy2 = VSQR(posy - hy*fj);
06887             if((dx2 + dy2) > rtot2) continue;
06888             for (k=kmin, fk=kmin; k<=kmax; k++, fk+=1.) {
06889                 dz2 = VSQR(posz - hz*fk);
06890                 if ((dz2 + dy2 + dx2) <= rtot2) {
06891                     array[IJK(i,j,k)] = markVal;
06892                 }
06893             }
06894         }
```

```
06895     }
06896 }
06897
06898 VPRIVATE void zlapSolve(
06899         Vpmg *thee,
06900         double **solution,
06901         double **source,
06902         double **work1
06903         ) {
06904
06905     /* NOTE:  this is an incredibly inefficient algorithm.  The next
06906      * improvement is to focus on only non-zero entries in the source term.
06907      * The best improvement is to use a fast sine transform */
06908
06909     int n, nx, ny, nz, i, j, k, kx, ky, kz;
06910     double hx, hy, hzed, wx, wy, wz, xlen, ylen, zlen;
06911     double phix, phixp1, phixm1, phiy, phiym1, phiyp1, phiz, phizm1, phizp1;
06912     double norm, coef, proj, eigx, eigy, eigz;
06913     double ihx2, ihy2, ihzed2;
06914     double *u, *f, *phi;
06915
06916     /* Snarf grid parameters */
06917     nx = thee->pmgp->nx;
06918     ny = thee->pmgp->ny;
06919     nz = thee->pmgp->nz;
06920     n = nx*ny*nz;
06921     hx = thee->pmgp->hx;
06922     ihx2 = 1.0/hx/hx;
06923     hy = thee->pmgp->hy;
06924     ihy2 = 1.0/hy/hy;
06925     hzed = thee->pmgp->hzed;
06926     ihzed2 = 1.0/hzed/hzed;
06927     xlen = thee->pmgp->xlen;
06928     ylen = thee->pmgp->ylen;
06929     zlen = thee->pmgp->zlen;
06930
06931     /* Set solution and source array pointers */
06932     u = *solution;
06933     f = *source;
06934     phi = *work1;
06935
06936     /* Zero out the solution vector */
06937     for (i=0; i<n; i++) thee->u[i] = 0.0;
06938
06939     /* Iterate through the wavenumbers */
06940     for (kx=1; kx<(nx-1); kx++) {
06941
06942         wx = (VPI*(double)kx)/((double)nx - 1.0);
06943         eigx = 2.0*ihx2*(1.0 - cos(wx));
06944
06945         for (ky=1; ky<(ny-1); ky++) {
06946
06947             wy = (VPI*(double)ky)/((double)ny - 1.0);
06948             eigy = 2.0*ihy2*(1.0 - cos(wy));
06949
06950             for (kz=1; kz<(nz-1); kz++) {
06951
06952                 wz = (VPI*(double)kz)/((double)nz - 1.0);
06953                 eigz = 2.0*ihzed2*(1.0 - cos(wz));
06954
06955                 /* Calculate the basis function.
06956                  * We could calculate each basis function as
06957                  *   phix(i) = sin(wx*i)
06958                  *   phiy(j) = sin(wy*j)
06959                  *   phiz(k) = sin(wz*k)
06960                  * However, this is likely to be very expensive.
06961                  * Therefore, we can use the fact that
06962                  *   phix(i+1) = (2-hx*hx*eigx)*phix(i) - phix(i-1)
06963                  * */
06964                 for (i=1; i<(nx-1); i++) {
06965                     if (i == 1) {
06966                         phix = sin(wx*(double)i);
06967                         phixm1 = 0.0;
06968                     } else {
06969                         phixp1 = (2.0-hx*hx*eigx)*phix - phixm1;
06970                         phixm1 = phix;
06971                         phix = phixp1;
06972                     }
06973                     /* phix = sin(wx*(double)i); */
06974                     for (j=1; j<(ny-1); j++) {
06975                         if (j == 1) {
```

```
06976                               phiy = sin(wy*(double)j);
06977                               phiym1 = 0.0;
06978                           } else {
06979                               phiyp1 = (2.0-hy*hy*eigy)*phiy - phiym1;
06980                               phiym1 = phiy;
06981                               phiy = phiyp1;
06982                           }
06983                           /* phiy = sin(wy*(double)j); */
06984                           for (k=1; k<(nz-1); k++) {
06985                               if (k == 1) {
06986                                   phiz = sin(wz*(double)k);
06987                                   phizm1 = 0.0;
06988                               } else {
06989                                   phizp1 = (2.0-hzed*hzed*eigz)*phiz - phizm1;
06990                                   phizm1 = phiz;
06991                                   phiz = phizp1;
06992                               }
06993                               /* phiz = sin(wz*(double)k); */
06994
06995                               phi[IJK(i,j,k)] = phix*phiy*phiz;
06996
06997                           }
06998                       }
06999                   }
07000
07001                   /* Calculate the projection of the source function on this
07002                    * basis function */
07003                   proj = 0.0;
07004                   for (i=1; i<(nx-1); i++) {
07005                       for (j=1; j<(ny-1); j++) {
07006                           for (k=1; k<(nz-1); k++) {
07007
07008                               proj += f[IJK(i,j,k)]*phi[IJK(i,j,k)];
07009
07010                           } /* k loop */
07011                       } /* j loop */
07012                   } /* i loop */
07013
07014                   /* Assemble the coefficient to weight the contribution of this
07015                    * basis function to the solution */
07016                   /* The first contribution is the projection */
07017                   coef = proj;
07018                   /* The second contribution is the eigenvalue */
07019                   coef = coef/(eigx + eigy + eigz);
07020                   /* The third contribution is the normalization factor */
07021                   coef = (8.0/xlen/ylen/zlen)*coef;
07022                   /* The fourth contribution is from scaling the diagonal */
07023                   /* coef = hx*hy*hzed*coef; */
07024
07025                   /* Evaluate the basis function at each grid point */
07026                   for (i=1; i<(nx-1); i++) {
07027                       for (j=1; j<(ny-1); j++) {
07028                           for (k=1; k<(nz-1); k++) {
07029
07030                               u[IJK(i,j,k)] += coef*phi[IJK(i,j,k)];
07031
07032                           } /* k loop */
07033                       } /* j loop */
07034                   } /* i loop */
07035
07036               } /* kz loop */
07037           } /* ky loop */
07038       } /* kx loop */
07039
07040 }
07041
07042 VPUBLIC int Vpmg_solveLaplace(Vpmg *thee) {
07043
07044     int i, j, k, ijk, nx, ny, nz, n, dilo, dihi, djlo, djhi, dklo, dkhi;
07045     double hx, hy, hzed, epsw, iepsw, scal, scalx, scaly, scalz;
07046
07047     nx = thee->pmgp->nx;
07048     ny = thee->pmgp->ny;
07049     nz = thee->pmgp->nz;
07050     n = nx*ny*nz;
07051     hx = thee->pmgp->hx;
07052     hy = thee->pmgp->hy;
07053     hzed = thee->pmgp->hzed;
07054     epsw = Vpbe_getSolventDiel(thee->pbe);
07055     iepsw = 1.0/epsw;
07056     scal = hx*hy*hzed;
```

```
07057     scalx = hx*hy/hzed;
07058     scaly = hx*hzed/hy;
07059     scalz = hx*hy/hzed;
07060
07061     if (!(thee->filled)) {
07062         Vnm_print(2, "Vpmg_solve:  Need to call Vpmg_fillco()!\n");
07063         return 0;
07064     }
07065
07066     /* Load boundary conditions into the RHS array */
07067     for (i=1; i<(nx-1); i++) {
07068
07069         if (i == 1) dilo = 1;
07070         else dilo = 0;
07071         if (i == nx-2) dihi = 1;
07072         else dihi = 0;
07073
07074         for (j=1; j<(ny-1); j++) {
07075
07076             if (j == 1) djlo = 1;
07077             else djlo = 0;
07078             if (j == ny-2) djhi = 1;
07079             else djhi = 0;
07080
07081             for (k=1; k<(nz-1); k++) {
07082
07083                 if (k == 1) dklo = 1;
07084                 else dklo = 0;
07085                 if (k == nz-2) dkhi = 1;
07086                 else dkhi = 0;
07087
07089                 thee->fcf[IJK(i,j,k)] = \
07090                       iepsw*scal*thee->charge[IJK(i,j,k)] \
07091                     + dilo*scalx*thee->gxcf[IJKx(j,k,0)] \
07092                     + dihi*scalx*thee->gxcf[IJKx(j,k,1)] \
07093                     + djlo*scaly*thee->gycf[IJKy(i,k,0)] \
07094                     + djhi*scaly*thee->gycf[IJKy(i,k,1)] \
07095                     + dklo*scalz*thee->gzcf[IJKz(i,j,0)] \
07096                     + dkhi*scalz*thee->gzcf[IJKz(i,j,1)] ;
07097
07098             }
07099         }
07100     }
07101
07102     /* Solve */
07103     zlapSolve( thee, &(thee->u), &(thee->fcf), &(thee->tcf) );
07104
07105     /* Add boundary conditions to solution */
07106     /* i faces */
07107     for (j=0; j<ny; j++) {
07108         for (k=0; k<nz; k++) {
07109             thee->u[IJK(0,j,k)] = thee->gxcf[IJKx(j,k,0)];
07110             thee->u[IJK(nx-1,j,k)] = thee->gycf[IJKx(j,k,1)];
07111         }
07112     }
07113     /* j faces */
07114     for (i=0; i<nx; i++) {
07115         for (k=0; k<nz; k++) {
07116             thee->u[IJK(i,0,k)] = thee->gycf[IJKy(i,k,0)];
07117             thee->u[IJK(i,ny-1,k)] = thee->gycf[IJKy(i,k,1)];
07118         }
07119     }
07120     /* k faces */
07121     for (i=0; i<nx; i++) {
07122         for (j=0; j<ny; j++) {
07123             thee->u[IJK(i,j,0)] = thee->gzcf[IJKz(i,j,0)];
07124             thee->u[IJK(i,j,nz-1)] = thee->gzcf[IJKz(i,j,1)];
07125         }
07126     }
07127
07128     return 1;
07129
07130 }
07131
07132 VPRIVATE double VFCHI4(int i, double f) {
07133   return (2.5+((double)(i)-(f)));
07134 }
07135
07136 VPRIVATE double bspline4(double x) {
07137
07138     double m, m2;
```

```
07139     static double one6 = 1.0/6.0;
07140     static double one8 = 1.0/8.0;
07141     static double one24 = 1.0/24.0;
07142     static double thirteen24 = 13.0/24.0;
07143     static double fourtyseven24 = 47.0/24.0;
07144     static double seventeen24 = 17.0/24.0;
07145
07146     if ((x > 0.0) && (x <= 1.0)){
07147       m = x*x;
07148       return one24*m*m;
07149     } else if ((x > 1.0) && (x <= 2.0)){
07150       m = x - 1.0;
07151       m2 = m*m;
07152       return -one8 + one6*x + m2*(0.25 + one6*m - one6*m2);
07153     } else if ((x > 2.0) && (x <= 3.0)){
07154       m = x - 2.0;
07155       m2 = m*m;
07156       return -thirteen24 + 0.5*x + m2*(-0.25 - 0.5*m + 0.25*m2);
07157     } else if ((x > 3.0) && (x <= 4.0)){
07158       m = x - 3.0;
07159       m2 = m*m;
07160       return fourtyseven24 - 0.5*x + m2*(-0.25 + 0.5*m - one6*m2);
07161     } else if ((x > 4.0) && (x <= 5.0)){
07162       m = x - 4.0;
07163       m2 = m*m;
07164       return seventeen24 - one6*x + m2*(0.25 - one6*m + one24*m2);
07165     } else {
07166       return 0.0;
07167     }
07168 }
07169
07170 VPUBLIC double dbspline4(double x) {
07171
07172     double m, m2;
07173     static double one6 = 1.0/6.0;
07174     static double one3 = 1.0/3.0;
07175     static double two3 = 2.0/3.0;
07176     static double thirteen6 = 13.0/6.0;
07177
07178     if ((x > 0.0) && (x <= 1.0)){
07179       m2 = x*x;
07180       return one6*x*m2;
07181     } else if ((x > 1.0) && (x <= 2.0)){
07182       m = x - 1.0;
07183       m2 = m*m;
07184       return -one3 + 0.5*x + m2*(0.5 - two3*m);
07185     } else if ((x > 2.0) && (x <= 3.0)){
07186       m = x - 2.0;
07187       m2 = m*m;
07188       return 1.5 - 0.5*x + m2*(-1.5 + m);
07189     } else if ((x > 3.0) && (x <= 4.0)){
07190       m = x - 3.0;
07191       m2 = m*m;
07192       return 1.0 - 0.5*x + m2*(1.5 - two3*m);
07193     } else if ((x > 4.0) && (x <= 5.0)){
07194       m = x - 4.0;
07195       m2 = m*m;
07196       return -thirteen6 + 0.5*x + m2*(-0.5 + one6*m);
07197     } else {
07198       return 0.0;
07199     }
07200 }
07201
07202 VPUBLIC double d2bspline4(double x) {
07203
07204     double m, m2;
07205
07206     if ((x > 0.0) && (x <= 1.0)){
07207       return 0.5*x*x;
07208     } else if ((x > 1.0) && (x <= 2.0)){
07209       m = x - 1.0;
07210       m2 = m*m;
07211       return -0.5 + x - 2.0*m2;
07212     } else if ((x > 2.0) && (x <= 3.0)){
07213       m = x - 2.0;
07214       m2 = m*m;
07215       return 5.5 - 3.0*x + 3.0*m2;
07216     } else if ((x > 3.0) && (x <= 4.0)){
07217       m = x - 3.0;
07218       m2 = m*m;
07219       return -9.5 + 3.0*x - 2.0*m2;
```

```
07220     } else if ((x > 4.0) && (x <= 5.0)){
07221       m = x - 4.0;
07222       m2 = m*m;
07223       return 4.5 - x + 0.5*m2;
07224     } else {
07225       return 0.0;
07226     }
07227 }
07228
07229 VPUBLIC double d3bspline4(double x) {
07230
07231     if      ((x > 0.0) && (x <= 1.0)) return x;
07232     else if ((x > 1.0) && (x <= 2.0)) return 5.0 - 4.0 * x;
07233     else if ((x > 2.0) && (x <= 3.0)) return -15.0 + 6.0 * x;
07234     else if ((x > 3.0) && (x <= 4.0)) return 15.0 - 4.0 * x;
07235     else if ((x > 4.0) && (x <= 5.0)) return x - 5.0;
07236     else                              return 0.0;
07237
07238 }
07239
07240 VPUBLIC void fillcoPermanentMultipole(Vpmg *thee) {
07241
07242     Valist *alist;
07243     Vpbe *pbe;
07244     Vatom *atom;
07245     /* Coversions */
07246     double zmagic, f;
07247     /* Grid */
07248     double xmin, xmax, ymin, ymax, zmin, zmax;
07249     double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;
07250     double hx, hy, hzed, *apos;
07251     /* Multipole */
07252     double charge, *dipole,*quad;
07253     double c,ux,uy,uz,qxx,qyx,qyy,qzx,qzy,qzz,qave;
07254     /* B-spline weights */
07255     double mx,my,mz,dmx,dmy,dmz,d2mx,d2my,d2mz;
07256     double mi,mj,mk;
07257     /* Loop variables */
07258     int i, ii, jj, kk, nx, ny, nz, iatom;
07259     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
07260
07261     /* sanity check */
07262     double mir,mjr,mkr,mr2;
07263     double debye,mc,mux,muy,muz,mqxx,mqyx,mqyy,mqzx,mqzy,mqzz;
07264
07265     VASSERT(thee != VNULL);
07266
07267     /* Get PBE info */
07268     pbe = thee->pbe;
07269     alist = pbe->alist;
07270     zmagic = Vpbe_getZmagic(pbe);
07271
07272     /* Mesh info */
07273     nx = thee->pmgp->nx;
07274     ny = thee->pmgp->ny;
07275     nz = thee->pmgp->nz;
07276     hx = thee->pmgp->hx;
07277     hy = thee->pmgp->hy;
07278     hzed = thee->pmgp->hzed;
07279
07280     /* Conversion */
07281     f = zmagic/(hx*hy*hzed);
07282
07283     /* Define the total domain size */
07284     xlen = thee->pmgp->xlen;
07285     ylen = thee->pmgp->ylen;
07286     zlen = thee->pmgp->zlen;
07287
07288     /* Define the min/max dimensions */
07289     xmin = thee->pmgp->xcent - (xlen/2.0);
07290     ymin = thee->pmgp->ycent - (ylen/2.0);
07291     zmin = thee->pmgp->zcent - (zlen/2.0);
07292     xmax = thee->pmgp->xcent + (xlen/2.0);
07293     ymax = thee->pmgp->ycent + (ylen/2.0);
07294     zmax = thee->pmgp->zcent + (zlen/2.0);
07295
07296     /* Fill in the source term (permanent atomic multipoles) */
07297     Vnm_print(0, "fillcoPermanentMultipole:  filling in source term.\n");
07298     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
07299
07300         atom = Valist_getAtom(alist, iatom);
```

```
07301         apos = Vatom_getPosition(atom);
07302
07303         c = Vatom_getCharge(atom)*f;
07304
07305 #if defined(WITH_TINKER)
07306         dipole = Vatom_getDipole(atom);
07307         ux = dipole[0]/hx*f;
07308         uy = dipole[1]/hy*f;
07309         uz = dipole[2]/hzed*f;
07310         quad = Vatom_getQuadrupole(atom);
07311         qxx = (1.0/3.0)*quad[0]/(hx*hx)*f;
07312         qyx = (2.0/3.0)*quad[3]/(hx*hy)*f;
07313         qyy = (1.0/3.0)*quad[4]/(hy*hy)*f;
07314         qzx = (2.0/3.0)*quad[6]/(hzed*hx)*f;
07315         qzy = (2.0/3.0)*quad[7]/(hzed*hy)*f;
07316         qzz = (1.0/3.0)*quad[8]/(hzed*hzed)*f;
07317 #else
07318         ux = 0.0;
07319         uy = 0.0;
07320         uz = 0.0;
07321         qxx = 0.0;
07322         qyx = 0.0;
07323         qyy = 0.0;
07324         qzx = 0.0;
07325         qzy = 0.0;
07326         qzz = 0.0;
07327 #endif /* if defined(WITH_TINKER) */
07328
07329         /* check
07330         mc = 0.0;
07331         mux = 0.0;
07332         muy = 0.0;
07333         muz = 0.0;
07334         mqxx = 0.0;
07335         mqyx = 0.0;
07336         mqyy = 0.0;
07337         mqzx = 0.0;
07338         mqzy = 0.0;
07339         mqzz = 0.0; */
07340
07341         /* Make sure we're on the grid */
07342         if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx))  || \
07343             (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy))  || \
07344             (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
07345             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]);
07346             Vnm_print(2, "fillcoPermanentMultipole: xmin = %g, xmax = %g\n", xmin, xmax);
07347             Vnm_print(2, "fillcoPermanentMultipole: ymin = %g, ymax = %g\n", ymin, ymax);
07348             Vnm_print(2, "fillcoPermanentMultipole: zmin = %g, zmax = %g\n", zmin, zmax);
07349             fflush(stderr);
07350         } else {
07351
07352             /* Convert the atom position to grid reference frame */
07353             position[0] = apos[0] - xmin;
07354             position[1] = apos[1] - ymin;
07355             position[2] = apos[2] - zmin;
07356
07357             /* Figure out which vertices we're next to */
07358             ifloat = position[0]/hx;
07359             jfloat = position[1]/hy;
07360             kfloat = position[2]/hzed;
07361
07362             ip1   = (int)ceil(ifloat);
07363             ip2   = ip1 + 2;
07364             im1   = (int)floor(ifloat);
07365             im2   = im1 - 2;
07366             jp1   = (int)ceil(jfloat);
07367             jp2   = jp1 + 2;
07368             jm1   = (int)floor(jfloat);
07369             jm2   = jm1 - 2;
07370             kp1   = (int)ceil(kfloat);
07371             kp2   = kp1 + 2;
07372             km1   = (int)floor(kfloat);
07373             km2   = km1 - 2;
07374
07375             /* This step shouldn't be necessary, but it saves nasty debugging
07376              * later on if something goes wrong */
07377             ip2 = VMIN2(ip2,nx-1);
07378             ip1 = VMIN2(ip1,nx-1);
07379             im1 = VMAX2(im1,0);
07380             im2 = VMAX2(im2,0);
```

```
07381             jp2 = VMIN2(jp2,ny-1);
07382             jp1 = VMIN2(jp1,ny-1);
07383             jm1 = VMAX2(jm1,0);
07384             jm2 = VMAX2(jm2,0);
07385             kp2 = VMIN2(kp2,nz-1);
07386             kp1 = VMIN2(kp1,nz-1);
07387             km1 = VMAX2(km1,0);
07388             km2 = VMAX2(km2,0);
07389
07390             /* Now assign fractions of the charge to the nearby verts */
07391             for (ii=im2; ii<=ip2; ii++) {
07392                 mi = VFCHI4(ii,ifloat);
07393                 mx = bspline4(mi);
07394                 dmx = dbspline4(mi);
07395                 d2mx = d2bspline4(mi);
07396                 for (jj=jm2; jj<=jp2; jj++) {
07397                     mj = VFCHI4(jj,jfloat);
07398                     my = bspline4(mj);
07399                     dmy = dbspline4(mj);
07400                     d2my = d2bspline4(mj);
07401                     for (kk=km2; kk<=kp2; kk++) {
07402                         mk = VFCHI4(kk,kfloat);
07403                         mz = bspline4(mk);
07404                         dmz = dbspline4(mk);
07405                         d2mz = d2bspline4(mk);
07406                         charge = mx*my*mz*c -
07407                          dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz +
07408                          d2mx*my*mz*qxx +
07409                          dmx*dmy*mz*qyx + mx*d2my*mz*qyy +
07410                          dmx*my*dmz*qzx + mx*dmy*dmz*qzy + mx*my*d2mz*qzz;
07411                         thee->charge[IJK(ii,jj,kk)] += charge;
07412
07413                         /* sanity check - recalculate traceless multipoles
07414                            from the grid charge distribution for this
07415                            site.
07416
07417                         mir = (mi - 2.5) * hx;
07418                         mjr = (mj - 2.5) * hy;
07419                         mkr = (mk - 2.5) * hzed;
07420                         mr2 = mir*mir+mjr*mjr+mkr*mkr;
07421                         mc += charge;
07422                         mux += mir * charge;
07423                         muy += mjr * charge;
07424                         muz += mkr * charge;
07425                         mqxx += (1.5*mir*mir - 0.5*mr2) * charge;
07426                         mqyx += 1.5*mjr*mir * charge;
07427                         mqyy += (1.5*mjr*mjr - 0.5*mr2) * charge;
07428                         mqzx += 1.5*mkr*mir * charge;
07429                         mqzy += 1.5*mkr*mjr * charge;
07430                         mqzz += (1.5*mkr*mkr - 0.5*mr2) * charge;
07431                          */
07432                     }
07433                 }
07434             }
07435         } /* endif (on the mesh) */
07436
07437         /* print out the Grid vs. Ideal Point Multipole. */
07438
07439         /*
07440         debye = 4.8033324;
07441         mc = mc/f;
07442         mux = mux/f*debye;
07443         muy = muy/f*debye;
07444         muz = muz/f*debye;
07445         mqxx = mqxx/f*debye;
07446         mqyy = mqyy/f*debye;
07447         mqzz = mqzz/f*debye;
07448         mqyx = mqyx/f*debye;
07449         mqzx = mqzx/f*debye;
07450         mqzy = mqzy/f*debye;
07451
07452         printf(" Grid v. Actual Permanent Multipole for Site %i\n",iatom);
07453         printf(" G: %10.6f\n",mc);
07454         printf(" A: %10.6f\n\n",c/f);
07455         printf(" G: %10.6f %10.6f %10.6f\n",mux,muy,muz);
07456         printf(" A: %10.6f %10.6f %10.6f\n\n",
07457                  (ux * hx / f) * debye,
07458                  (uy * hy / f) * debye,
07459                  (uz * hzed /f) * debye);
07460         printf(" G: %10.6f\n",mqxx);
07461         printf(" A: %10.6f\n",quad[0]*debye);
```

```
07462         printf(" G: %10.6f %10.6f\n",mqyx,mqyy);
07463         printf(" A: %10.6f %10.6f\n",quad[3]*debye,quad[4]*debye);
07464         printf(" G: %10.6f %10.6f %10.6f\n",mqzx,mqzy,mqzz);
07465         printf(" A: %10.6f %10.6f %10.6f\n\n",
07466                 quad[6]*debye,quad[7]*debye,quad[8]*debye);  */
07467
07468     } /* endfor (each atom) */
07469 }
07470
07471 #if defined(WITH_TINKER)
07472
07473 VPUBLIC void fillcoInducedDipole(Vpmg *thee) {
07474
07475     Valist *alist;
07476     Vpbe *pbe;
07477     Vatom *atom;
07478     /* Conversions */
07479     double zmagic, f;
07480     /* Grid */
07481     double xmin, xmax, ymin, ymax, zmin, zmax;
07482     double xlen, ylen, zlen, ifloat, jfloat, kfloat;
07483     double hx, hy, hzed, *apos, position[3];
07484     /* B-spline weights */
07485     double mx, my, mz, dmx, dmy, dmz;
07486     /* Dipole */
07487     double charge, *dipole, ux,uy,uz;
07488     double mi,mj,mk;
07489     /* Loop indeces */
07490     int i, ii, jj, kk, nx, ny, nz, iatom;
07491     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
07492
07493     double debye;
07494     double mux,muy,muz;
07495     double mir,mjr,mkr;
07496
07497     VASSERT(thee != VNULL);
07498
07499     /* Get PBE info */
07500     pbe = thee->pbe;
07501     alist = pbe->alist;
07502     zmagic = Vpbe_getZmagic(pbe);
07503
07504     /* Mesh info */
07505     nx = thee->pmgp->nx;
07506     ny = thee->pmgp->ny;
07507     nz = thee->pmgp->nz;
07508     hx = thee->pmgp->hx;
07509     hy = thee->pmgp->hy;
07510     hzed = thee->pmgp->hzed;
07511
07512     /* Conversion */
07513     f = zmagic/(hx*hy*hzed);
07514
07515     /* Define the total domain size */
07516     xlen = thee->pmgp->xlen;
07517     ylen = thee->pmgp->ylen;
07518     zlen = thee->pmgp->zlen;
07519
07520     /* Define the min/max dimensions */
07521     xmin = thee->pmgp->xcent - (xlen/2.0);
07522     ymin = thee->pmgp->ycent - (ylen/2.0);
07523     zmin = thee->pmgp->zcent - (zlen/2.0);
07524     xmax = thee->pmgp->xcent + (xlen/2.0);
07525     ymax = thee->pmgp->ycent + (ylen/2.0);
07526     zmax = thee->pmgp->zcent + (zlen/2.0);
07527
07528     /* Fill in the source term (induced dipoles) */
07529     Vnm_print(0, "fillcoInducedDipole:  filling in the source term.\n");
07530     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
07531
07532         atom = Valist_getAtom(alist, iatom);
07533         apos = Vatom_getPosition(atom);
07534
07535         dipole = Vatom_getInducedDipole(atom);
07536         ux = dipole[0]/hx*f;
07537         uy = dipole[1]/hy*f;
07538         uz = dipole[2]/hzed*f;
07539
07540         mux = 0.0;
07541         muy = 0.0;
07542         muz = 0.0;
```

```
07543
07544         /* Make sure we're on the grid */
07545         if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx))  || \
07546             (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy))  || \
07547             (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
07548             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]);
07549             Vnm_print(2, "fillcoInducedDipole: xmin = %g, xmax = %g\n", xmin, xmax);
07550             Vnm_print(2, "fillcoInducedDipole: ymin = %g, ymax = %g\n", ymin, ymax);
07551             Vnm_print(2, "fillcoInducedDipole: zmin = %g, zmax = %g\n", zmin, zmax);
07552             fflush(stderr);
07553         } else {
07554
07555             /* Convert the atom position to grid reference frame */
07556             position[0] = apos[0] - xmin;
07557             position[1] = apos[1] - ymin;
07558             position[2] = apos[2] - zmin;
07559
07560             /* Figure out which vertices we're next to */
07561             ifloat = position[0]/hx;
07562             jfloat = position[1]/hy;
07563             kfloat = position[2]/hzed;
07564
07565             ip1   = (int)ceil(ifloat);
07566             ip2   = ip1 + 2;
07567             im1   = (int)floor(ifloat);
07568             im2   = im1 - 2;
07569             jp1   = (int)ceil(jfloat);
07570             jp2   = jp1 + 2;
07571             jm1   = (int)floor(jfloat);
07572             jm2   = jm1 - 2;
07573             kp1   = (int)ceil(kfloat);
07574             kp2   = kp1 + 2;
07575             km1   = (int)floor(kfloat);
07576             km2   = km1 - 2;
07577
07578             /* This step shouldn't be necessary, but it saves nasty debugging
07579              * later on if something goes wrong */
07580             ip2 = VMIN2(ip2,nx-1);
07581             ip1 = VMIN2(ip1,nx-1);
07582             im1 = VMAX2(im1,0);
07583             im2 = VMAX2(im2,0);
07584             jp2 = VMIN2(jp2,ny-1);
07585             jp1 = VMIN2(jp1,ny-1);
07586             jm1 = VMAX2(jm1,0);
07587             jm2 = VMAX2(jm2,0);
07588             kp2 = VMIN2(kp2,nz-1);
07589             kp1 = VMIN2(kp1,nz-1);
07590             km1 = VMAX2(km1,0);
07591             km2 = VMAX2(km2,0);
07592
07593             /* Now assign fractions of the dipole to the nearby verts */
07594             for (ii=im2; ii<=ip2; ii++) {
07595                 mi = VFCHI4(ii,ifloat);
07596                 mx = bspline4(mi);
07597                 dmx = dbspline4(mi);
07598                 for (jj=jm2; jj<=jp2; jj++) {
07599                     mj = VFCHI4(jj,jfloat);
07600                     my = bspline4(mj);
07601                     dmy = dbspline4(mj);
07602                     for (kk=km2; kk<=kp2; kk++) {
07603                         mk = VFCHI4(kk,kfloat);
07604                         mz = bspline4(mk);
07605                         dmz = dbspline4(mk);
07606                         charge = -dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz;
07607                         thee->charge[IJK(ii,jj,kk)] += charge;
07608
07609                         /*
07610                         mir = (mi - 2.5) * hx;
07611                         mjr = (mj - 2.5) * hy;
07612                         mkr = (mk - 2.5) * hzed;
07613                         mux += mir * charge;
07614                         muy += mjr * charge;
07615                         muz += mkr * charge;
07616                         */
07617                     }
07618                 }
07619             }
07620         } /* endif (on the mesh) */
07621
07622         /* check
```

```
07623         debye = 4.8033324;
07624         mux = mux/f*debye;
07625         muy = muy/f*debye;
07626         muz = muz/f*debye;
07627 
07628         printf(" Grid v. Actual Induced Dipole for Site %i\n",iatom);
07629         printf(" G: %10.6f %10.6f %10.6f\n",mux,muy,muz);
07630         printf(" A: %10.6f %10.6f %10.6f\n\n",
07631                 (ux * hx / f) * debye,
07632                 (uy * hy / f) * debye,
07633                 (uz * hzed /f) * debye);
07634          */
07635 
07636     } /* endfor (each atom) */
07637 }
07638 
07639 VPUBLIC void fillcoNLInducedDipole(Vpmg *thee) {
07640 
07641     Valist *alist;
07642     Vpbe *pbe;
07643     Vatom *atom;
07644     /* Conversions */
07645     double zmagic, f;
07646     /* Grid */
07647     double xmin, xmax, ymin, ymax, zmin, zmax;
07648     double xlen, ylen, zlen, ifloat, jfloat, kfloat;
07649     double hx, hy, hzed, *apos, position[3];
07650     /* B-spline weights */
07651     double mx, my, mz, dmx, dmy, dmz;
07652     /* Dipole */
07653     double charge, *dipole, ux,uy,uz;
07654     double mi,mj,mk;
07655     /* Loop indeces */
07656     int i, ii, jj, kk, nx, ny, nz, iatom;
07657     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
07658 
07659     /* sanity check
07660     double debye;
07661     double mux,muy,muz;
07662     double mir,mjr,mkr;
07663      */
07664 
07665     VASSERT(thee != VNULL);
07666 
07667     /* Get PBE info */
07668     pbe = thee->pbe;
07669     alist = pbe->alist;
07670     zmagic = Vpbe_getZmagic(pbe);
07671 
07672     /* Mesh info */
07673     nx = thee->pmgp->nx;
07674     ny = thee->pmgp->ny;
07675     nz = thee->pmgp->nz;
07676     hx = thee->pmgp->hx;
07677     hy = thee->pmgp->hy;
07678     hzed = thee->pmgp->hzed;
07679 
07680     /* Conversion */
07681     f = zmagic/(hx*hy*hzed);
07682 
07683     /* Define the total domain size */
07684     xlen = thee->pmgp->xlen;
07685     ylen = thee->pmgp->ylen;
07686     zlen = thee->pmgp->zlen;
07687 
07688     /* Define the min/max dimensions */
07689     xmin = thee->pmgp->xcent - (xlen/2.0);
07690     ymin = thee->pmgp->ycent - (ylen/2.0);
07691     zmin = thee->pmgp->zcent - (zlen/2.0);
07692     xmax = thee->pmgp->xcent + (xlen/2.0);
07693     ymax = thee->pmgp->ycent + (ylen/2.0);
07694     zmax = thee->pmgp->zcent + (zlen/2.0);
07695 
07696     /* Fill in the source term (non-local induced dipoles) */
07697     Vnm_print(0, "fillcoNLInducedDipole:  filling in source term.\n");
07698     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
07699 
07700         atom = Valist_getAtom(alist, iatom);
07701         apos = Vatom_getPosition(atom);
07702 
07703         dipole = Vatom_getNLInducedDipole(atom);
```

```
07704         ux = dipole[0]/hx*f;
07705         uy = dipole[1]/hy*f;
07706         uz = dipole[2]/hzed*f;
07707 
07708         /*
07709         mux = 0.0;
07710         muy = 0.0;
07711         muz = 0.0;
07712          */
07713 
07714         /* Make sure we're on the grid */
07715         if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx))  || \
07716             (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy))  || \
07717             (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
07718             Vnm_print(2, "fillcoNLInducedDipole: Atom #%d at (%4.3f, %4.3f,%4.3f) is off the mesh
       (ignoring this atom):\n", iatom, apos[0], apos[1], apos[2]);
07719             Vnm_print(2, "fillcoNLInducedDipole: xmin = %g, xmax = %g\n", xmin, xmax);
07720             Vnm_print(2, "fillcoNLInducedDipole: ymin = %g, ymax = %g\n", ymin, ymax);
07721             Vnm_print(2, "fillcoNLInducedDipole: zmin = %g, zmax = %g\n", zmin, zmax);
07722             fflush(stderr);
07723         } else {
07724 
07725             /* Convert the atom position to grid reference frame */
07726             position[0] = apos[0] - xmin;
07727             position[1] = apos[1] - ymin;
07728             position[2] = apos[2] - zmin;
07729 
07730             /* Figure out which vertices we're next to */
07731             ifloat = position[0]/hx;
07732             jfloat = position[1]/hy;
07733             kfloat = position[2]/hzed;
07734 
07735             ip1   = (int)ceil(ifloat);
07736             ip2   = ip1 + 2;
07737             im1   = (int)floor(ifloat);
07738             im2   = im1 - 2;
07739             jp1   = (int)ceil(jfloat);
07740             jp2   = jp1 + 2;
07741             jm1   = (int)floor(jfloat);
07742             jm2   = jm1 - 2;
07743             kp1   = (int)ceil(kfloat);
07744             kp2   = kp1 + 2;
07745             km1   = (int)floor(kfloat);
07746             km2   = km1 - 2;
07747 
07748             /* This step shouldn't be necessary, but it saves nasty debugging
07749              * later on if something goes wrong */
07750             ip2 = VMIN2(ip2,nx-1);
07751             ip1 = VMIN2(ip1,nx-1);
07752             im1 = VMAX2(im1,0);
07753             im2 = VMAX2(im2,0);
07754             jp2 = VMIN2(jp2,ny-1);
07755             jp1 = VMIN2(jp1,ny-1);
07756             jm1 = VMAX2(jm1,0);
07757             jm2 = VMAX2(jm2,0);
07758             kp2 = VMIN2(kp2,nz-1);
07759             kp1 = VMIN2(kp1,nz-1);
07760             km1 = VMAX2(km1,0);
07761             km2 = VMAX2(km2,0);
07762 
07763             /* Now assign fractions of the non local induced dipole
07764                to the nearby verts */
07765             for (ii=im2; ii<=ip2; ii++) {
07766                 mi = VFCHI4(ii,ifloat);
07767                 mx = bspline4(mi);
07768                 dmx = dbspline4(mi);
07769                 for (jj=jm2; jj<=jp2; jj++) {
07770                     mj = VFCHI4(jj,jfloat);
07771                     my = bspline4(mj);
07772                     dmy = dbspline4(mj);
07773                     for (kk=km2; kk<=kp2; kk++) {
07774                         mk = VFCHI4(kk,kfloat);
07775                         mz = bspline4(mk);
07776                         dmz = dbspline4(mk);
07777                         charge = -dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz;
07778                         thee->charge[IJK(ii,jj,kk)] += charge;
07779 
07780                         /*
07781                         mir = (mi - 2.5) * hx;
07782                         mjr = (mj - 2.5) * hy;
07783                         mkr = (mk - 2.5) * hzed;
```

```
07784                           mux += mir * charge;
07785                           muy += mjr * charge;
07786                           muz += mkr * charge;
07787                            */
07788                       }
07789                   }
07790               }
07791           } /* endif (on the mesh) */
07792
07793           /*
07794           debye = 4.8033324;
07795           mux = mux/f*debye;
07796           muy = muy/f*debye;
07797           muz = muz/f*debye;
07798
07799           printf(" Grid v. Actual Non-Local Induced Dipole for Site %i\n",iatom);
07800           printf(" G: %10.6f %10.6f %10.6f\n",mux,muy,muz);
07801           printf(" A: %10.6f %10.6f %10.6f\n\n",
07802                    (ux * hx / f) * debye,
07803                    (uy * hy / f) * debye,
07804                    (uz * hzed /f) * debye); */
07805
07806       } /* endfor (each atom) */
07807   }
07808
07809   VPUBLIC double Vpmg_qfPermanentMultipoleEnergy(Vpmg *thee, int atomID) {
07810
07811       double *u;
07812       Vatom *atom;
07813       /* Grid variables */
07814       int nx, ny, nz;
07815       double xmax, xmin, ymax, ymin, zmax, zmin;
07816       double hx, hy, hzed, ifloat, jfloat, kfloat;
07817       double mi, mj, mk;
07818       double *position;
07819       /* B-spline weights */
07820       double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz;
07821       /* Loop indeces */
07822       int ip1,ip2,im1,im2,jp1,jp2,jm1,jm2,kp1,kp2,km1,km2;
07823       int i,j,ii,jj,kk;
07824       /* Potential, field, field gradient and multipole components */
07825       double pot, rfe[3], rfde[3][3], energy;
07826       double f, charge, *dipole, *quad;
07827       double qxx, qyx, qyy, qzx, qzy, qzz;
07828
07829
07830       VASSERT(thee != VNULL);
07831       VASSERT(thee->filled);
07832
07833       /* Get the mesh information */
07834       nx = thee->pmgp->nx;
07835       ny = thee->pmgp->ny;
07836       nz = thee->pmgp->nz;
07837       hx = thee->pmgp->hx;
07838       hy = thee->pmgp->hy;
07839       hzed = thee->pmgp->hzed;
07840       xmax = thee->xf[nx-1];
07841       ymax = thee->yf[ny-1];
07842       zmax = thee->zf[nz-1];
07843       xmin = thee->xf[0];
07844       ymin = thee->yf[0];
07845       zmin = thee->zf[0];
07846
07847       u = thee->u;
07848
07849       atom = Valist_getAtom(thee->pbe->alist, atomID);
07850
07851       /* Currently all atoms must be in the same partition. */
07852
07853       VASSERT(atom->partID != 0);
07854
07855       /* Convert the atom position to grid coordinates */
07856
07857       position = Vatom_getPosition(atom);
07858       ifloat = (position[0] - xmin)/hx;
07859       jfloat = (position[1] - ymin)/hy;
07860       kfloat = (position[2] - zmin)/hzed;
07861       ip1 = (int)ceil(ifloat);
07862       ip2 = ip1 + 2;
07863       im1 = (int)floor(ifloat);
07864       im2 = im1 - 2;
```

```
07865     jp1 = (int)ceil(jfloat);
07866     jp2 = jp1 + 2;
07867     jm1 = (int)floor(jfloat);
07868     jm2 = jm1 - 2;
07869     kp1 = (int)ceil(kfloat);
07870     kp2 = kp1 + 2;
07871     km1 = (int)floor(kfloat);
07872     km2 = km1 - 2;
07873
07874     /* This step shouldn't be necessary, but it saves nasty debugging
07875      * later on if something goes wrong */
07876     ip2 = VMIN2(ip2,nx-1);
07877     ip1 = VMIN2(ip1,nx-1);
07878     im1 = VMAX2(im1,0);
07879     im2 = VMAX2(im2,0);
07880     jp2 = VMIN2(jp2,ny-1);
07881     jp1 = VMIN2(jp1,ny-1);
07882     jm1 = VMAX2(jm1,0);
07883     jm2 = VMAX2(jm2,0);
07884     kp2 = VMIN2(kp2,nz-1);
07885     kp1 = VMIN2(kp1,nz-1);
07886     km1 = VMAX2(km1,0);
07887     km2 = VMAX2(km2,0);
07888
07889     /* Initialize observables to zero */
07890     energy = 0.0;
07891     pot = 0.0;
07892     for (i=0;i<3;i++){
07893        rfe[i] = 0.0;
07894        for (j=0;j<3;j++){
07895          rfde[i][j] = 0.0;
07896        }
07897     }
07898
07899     for (ii=im2; ii<=ip2; ii++) {
07900       mi = VFCHI4(ii,ifloat);
07901       mx = bspline4(mi);
07902       dmx = dbspline4(mi);
07903       d2mx = d2bspline4(mi);
07904       for (jj=jm2; jj<=jp2; jj++) {
07905         mj = VFCHI4(jj,jfloat);
07906         my = bspline4(mj);
07907         dmy = dbspline4(mj);
07908         d2my = d2bspline4(mj);
07909         for (kk=km2; kk<=kp2; kk++) {
07910           mk = VFCHI4(kk,kfloat);
07911           mz = bspline4(mk);
07912           dmz = dbspline4(mk);
07913           d2mz = d2bspline4(mk);
07914           f = u[IJK(ii,jj,kk)];
07915           /* potential */
07916           pot  += f*mx*my*mz;
07917           /* field */
07918           rfe[0] += f*dmx*my*mz/hx;
07919           rfe[1] += f*mx*dmy*mz/hy;
07920           rfe[2] += f*mx*my*dmz/hzed;
07921           /* field gradient */
07922           rfde[0][0] += f*d2mx*my*mz/(hx*hx);
07923           rfde[1][0] += f*dmx*dmy*mz/(hy*hx);
07924           rfde[1][1] += f*mx*d2my*mz/(hy*hy);
07925           rfde[2][0] += f*dmx*my*dmz/(hx*hzed);
07926           rfde[2][1] += f*mx*dmy*dmz/(hy*hzed);
07927           rfde[2][2] += f*mx*my*d2mz/(hzed*hzed);
07928         }
07929       }
07930     }
07931
07932     charge = Vatom_getCharge(atom);
07933     dipole = Vatom_getDipole(atom);
07934     quad = Vatom_getQuadrupole(atom);
07935     qxx = quad[0]/3.0;
07936     qyx = quad[3]/3.0;
07937     qyy = quad[4]/3.0;
07938     qzx = quad[6]/3.0;
07939     qzy = quad[7]/3.0;
07940     qzz = quad[8]/3.0;
07941
07942     energy =   pot * charge
07943              - rfe[0] * dipole[0]
07944              - rfe[1] * dipole[1]
07945              - rfe[2] * dipole[2]
```

```
07946               +     rfde[0][0]*qxx
07947               + 2.0*rfde[1][0]*qyx +     rfde[1][1]*qyy
07948               + 2.0*rfde[2][0]*qzx + 2.0*rfde[2][1]*qzy + rfde[2][2]*qzz;
07949
07950     return energy;
07951 }
07952
07953 VPUBLIC void Vpmg_fieldSpline4(Vpmg *thee, int atomID, double field[3]) {
07954
07955     Vatom *atom;
07956     double *u, f;
07957     /* Grid variables */
07958     int nx, ny, nz;
07959     double xmax, xmin, ymax, ymin, zmax, zmin;
07960     double hx, hy, hzed, ifloat, jfloat, kfloat;
07961     double *apos, position[3];
07962     /* B-Spline weights */
07963     double mx, my, mz, dmx, dmy, dmz;
07964     double mi, mj, mk;
07965     /* Loop indeces */
07966     int ip1,ip2,im1,im2,jp1,jp2,jm1,jm2,kp1,kp2,km1,km2;
07967     int i,j,ii,jj,kk;
07968
07969
07970     VASSERT (thee != VNULL);
07971
07972     /* Get the mesh information */
07973     nx = thee->pmgp->nx;
07974     ny = thee->pmgp->ny;
07975     nz = thee->pmgp->nz;
07976     hx = thee->pmgp->hx;
07977     hy = thee->pmgp->hy;
07978     hzed = thee->pmgp->hzed;
07979     xmax = thee->xf[nx-1];
07980     ymax = thee->yf[ny-1];
07981     zmax = thee->zf[nz-1];
07982     xmin = thee->xf[0];
07983     ymin = thee->yf[0];
07984     zmin = thee->zf[0];
07985
07986     u = thee->u;
07987
07988     atom = Valist_getAtom(thee->pbe->alist, atomID);
07989
07990     /* Currently all atoms must be in the same partition. */
07991
07992     VASSERT (atom->partID != 0);
07993
07994     /* Convert the atom position to grid coordinates */
07995
07996     apos = Vatom_getPosition(atom);
07997     position[0] = apos[0] - xmin;
07998     position[1] = apos[1] - ymin;
07999     position[2] = apos[2] - zmin;
08000     ifloat = position[0]/hx;
08001     jfloat = position[1]/hy;
08002     kfloat = position[2]/hzed;
08003     ip1 = (int)ceil(ifloat);
08004     ip2 = ip1 + 2;
08005     im1 = (int)floor(ifloat);
08006     im2 = im1 - 2;
08007     jp1 = (int)ceil(jfloat);
08008     jp2 = jp1 + 2;
08009     jm1 = (int)floor(jfloat);
08010     jm2 = jm1 - 2;
08011     kp1 = (int)ceil(kfloat);
08012     kp2 = kp1 + 2;
08013     km1 = (int)floor(kfloat);
08014     km2 = km1 - 2;
08015
08016     /* This step shouldn't be necessary, but it saves nasty debugging
08017      * later on if something goes wrong */
08018     ip2 = VMIN2(ip2,nx-1);
08019     ip1 = VMIN2(ip1,nx-1);
08020     im1 = VMAX2(im1,0);
08021     im2 = VMAX2(im2,0);
08022     jp2 = VMIN2(jp2,ny-1);
08023     jp1 = VMIN2(jp1,ny-1);
08024     jm1 = VMAX2(jm1,0);
08025     jm2 = VMAX2(jm2,0);
08026     kp2 = VMIN2(kp2,nz-1);
```

```
08027     kp1 = VMIN2(kp1,nz-1);
08028     km1 = VMAX2(km1,0);
08029     km2 = VMAX2(km2,0);
08030
08031     for (i=0;i<3;i++){
08032        field[i] = 0.0;
08033     }
08034
08035     for (ii=im2; ii<=ip2; ii++) {
08036       mi = VFCHI4(ii,ifloat);
08037       mx = bspline4(mi);
08038       dmx = dbspline4(mi);
08039       for (jj=jm2; jj<=jp2; jj++) {
08040         mj = VFCHI4(jj,jfloat);
08041         my = bspline4(mj);
08042         dmy = dbspline4(mj);
08043         for (kk=km2; kk<=kp2; kk++) {
08044           mk = VFCHI4(kk,kfloat);
08045           mz = bspline4(mk);
08046           dmz = dbspline4(mk);
08047           f = u[IJK(ii,jj,kk)];
08048
08049           field[0] += f*dmx*my*mz/hx;
08050           field[1] += f*mx*dmy*mz/hy;
08051           field[2] += f*mx*my*dmz/hzed;
08052         }
08053       }
08054     }
08055 }
08056
08057 VPUBLIC void Vpmg_qfPermanentMultipoleForce(Vpmg *thee, int atomID,
08058                                             double force[3], double torque[3]) {
08059
08060     Vatom *atom;
08061     double f, *u, *apos, position[3];
08062
08063     /* Grid variables */
08064     int nx,ny,nz;
08065     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
08066     double hx, hy, hzed, ifloat, jfloat, kfloat;
08067
08068     /* B-spline weights */
08069     double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz, d3mx, d3my, d3mz;
08070     double mi, mj, mk;
08071
08072     /* Loop indeces */
08073     int i, j, k, ii, jj, kk;
08074     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
08075
08076     /* Potential, field, field gradient and 2nd field gradient */
08077     double pot, e[3], de[3][3], d2e[3][3][3];
08078
08079     /* Permanent multipole components */
08080     double *dipole, *quad;
08081     double c, ux, uy, uz, qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
08082
08083     VASSERT(thee != VNULL);
08084     VASSERT(thee->filled);
08085
08086     atom = Valist_getAtom(thee->pbe->alist, atomID);
08087
08088     /* Currently all atoms must be in the same partition. */
08089
08090     VASSERT(atom->partID != 0);
08091
08092     apos = Vatom_getPosition(atom);
08093
08094     c = Vatom_getCharge(atom);
08095     dipole = Vatom_getDipole(atom);
08096     ux = dipole[0];
08097     uy = dipole[1];
08098     uz = dipole[2];
08099     quad = Vatom_getQuadrupole(atom);
08100     qxx = quad[0]/3.0;
08101     qxy = quad[1]/3.0;
08102     qxz = quad[2]/3.0;
08103     qyx = quad[3]/3.0;
08104     qyy = quad[4]/3.0;
08105     qyz = quad[5]/3.0;
08106     qzx = quad[6]/3.0;
08107     qzy = quad[7]/3.0;
```

```
08108     qzz = quad[8]/3.0;
08109
08110     /* Initialize observables */
08111     pot = 0.0;
08112     for (i=0;i<3;i++){
08113        e[i] = 0.0;
08114        for (j=0;j<3;j++){
08115           de[i][j] = 0.0;
08116           for (k=0;k<3;k++){
08117              d2e[i][j][k] = 0.0;
08118           }
08119        }
08120     }
08121
08122     /* Mesh info */
08123     nx = thee->pmgp->nx;
08124     ny = thee->pmgp->ny;
08125     nz = thee->pmgp->nz;
08126     hx = thee->pmgp->hx;
08127     hy = thee->pmgp->hy;
08128     hzed = thee->pmgp->hzed;
08129     xlen = thee->pmgp->xlen;
08130     ylen = thee->pmgp->ylen;
08131     zlen = thee->pmgp->zlen;
08132     xmin = thee->pmgp->xmin;
08133     ymin = thee->pmgp->ymin;
08134     zmin = thee->pmgp->zmin;
08135     xmax = thee->pmgp->xmax;
08136     ymax = thee->pmgp->ymax;
08137     zmax = thee->pmgp->zmax;
08138     u = thee->u;
08139
08140     /* Make sure we're on the grid */
08141     if ((apos[0]<=(xmin+2*hx))   || (apos[0]>=(xmax-2*hx)) \
08142      || (apos[1]<=(ymin+2*hy))   || (apos[1]>=(ymax-2*hy)) \
08143      || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
08144        Vnm_print(2, "qfPermanentMultipoleForce:  Atom off the mesh (ignoring) %6.3f %6.3f %6.3f\n",
      apos[0], apos[1], apos[2]);
08145        fflush(stderr);
08146     } else {
08147
08148        /* Convert the atom position to grid coordinates */
08149        position[0] = apos[0] - xmin;
08150        position[1] = apos[1] - ymin;
08151        position[2] = apos[2] - zmin;
08152        ifloat = position[0]/hx;
08153        jfloat = position[1]/hy;
08154        kfloat = position[2]/hzed;
08155        ip1 = (int)ceil(ifloat);
08156        ip2 = ip1 + 2;
08157        im1 = (int)floor(ifloat);
08158        im2 = im1 - 2;
08159        jp1 = (int)ceil(jfloat);
08160        jp2 = jp1 + 2;
08161        jm1 = (int)floor(jfloat);
08162        jm2 = jm1 - 2;
08163        kp1 = (int)ceil(kfloat);
08164        kp2 = kp1 + 2;
08165        km1 = (int)floor(kfloat);
08166        km2 = km1 - 2;
08167
08168        /* This step shouldn't be necessary, but it saves nasty debugging
08169         * later on if something goes wrong */
08170        ip2 = VMIN2(ip2,nx-1);
08171        ip1 = VMIN2(ip1,nx-1);
08172        im1 = VMAX2(im1,0);
08173        im2 = VMAX2(im2,0);
08174        jp2 = VMIN2(jp2,ny-1);
08175        jp1 = VMIN2(jp1,ny-1);
08176        jm1 = VMAX2(jm1,0);
08177        jm2 = VMAX2(jm2,0);
08178        kp2 = VMIN2(kp2,nz-1);
08179        kp1 = VMIN2(kp1,nz-1);
08180        km1 = VMAX2(km1,0);
08181        km2 = VMAX2(km2,0);
08182
08183        for (ii=im2; ii<=ip2; ii++) {
08184            mi = VFCHI4(ii,ifloat);
08185            mx = bspline4(mi);
08186            dmx = dbspline4(mi);
08187            d2mx = d2bspline4(mi);
```

```
08188             d3mx = d3bspline4(mi);
08189             for (jj=jm2; jj<=jp2; jj++) {
08190                 mj = VFCHI4(jj,jfloat);
08191                 my = bspline4(mj);
08192                 dmy = dbspline4(mj);
08193                 d2my = d2bspline4(mj);
08194                 d3my = d3bspline4(mj);
08195                 for (kk=km2; kk<=kp2; kk++) {
08196                     mk = VFCHI4(kk,kfloat);
08197                     mz = bspline4(mk);
08198                     dmz = dbspline4(mk);
08199                     d2mz = d2bspline4(mk);
08200                     d3mz = d3bspline4(mk);
08201                     f = u[IJK(ii,jj,kk)];
08202                     /* Potential */
08203                     pot  += f*mx*my*mz;
08204                     /* Field */
08205                     e[0] += f*dmx*my*mz/hx;
08206                     e[1] += f*mx*dmy*mz/hy;
08207                     e[2] += f*mx*my*dmz/hzed;
08208                     /* Field gradient */
08209                     de[0][0] += f*d2mx*my*mz/(hx*hx);
08210                     de[1][0] += f*dmx*dmy*mz/(hy*hx);
08211                     de[1][1] += f*mx*d2my*mz/(hy*hy);
08212                     de[2][0] += f*dmx*my*dmz/(hx*hzed);
08213                     de[2][1] += f*mx*dmy*dmz/(hy*hzed);
08214                     de[2][2] += f*mx*my*d2mz/(hzed*hzed);
08215                     /* 2nd Field Gradient
08216                        VxVxVa */
08217                     d2e[0][0][0] += f*d3mx*my*mz /(hx*hx*hx);
08218                     d2e[0][0][1] += f*d2mx*dmy*mz/(hx*hy*hx);
08219                     d2e[0][0][2] += f*d2mx*my*dmz/(hx*hx*hzed);
08220                     /* VyVxVa */
08221                     d2e[1][0][0] += f*d2mx*dmy*mz/(hx*hx*hy);
08222                     d2e[1][0][1] += f*dmx*d2my*mz/(hx*hy*hy);
08223                     d2e[1][0][2] += f*dmx*dmy*dmz/(hx*hy*hzed);
08224                     /* VyVyVa */
08225                     d2e[1][1][0] += f*dmx*d2my*mz/(hx*hy*hy);
08226                     d2e[1][1][1] += f*mx*d3my*mz /(hy*hy*hy);
08227                     d2e[1][1][2] += f*mx*d2my*dmz/(hy*hy*hzed);
08228                     /* VzVxVa */
08229                     d2e[2][0][0] += f*d2mx*my*dmz/(hx*hx*hzed);
08230                     d2e[2][0][1] += f*dmx*dmy*dmz/(hx*hy*hzed);
08231                     d2e[2][0][2] += f*dmx*my*d2mz/(hx*hzed*hzed);
08232                     /* VzVyVa */
08233                     d2e[2][1][0] += f*dmx*dmy*dmz/(hx*hy*hzed);
08234                     d2e[2][1][1] += f*mx*d2my*dmz/(hy*hy*hzed);
08235                     d2e[2][1][2] += f*mx*dmy*d2mz/(hy*hzed*hzed);
08236                     /* VzVzVa */
08237                     d2e[2][2][0] += f*dmx*my*d2mz/(hx*hzed*hzed);
08238                     d2e[2][2][1] += f*mx*dmy*d2mz/(hy*hzed*hzed);
08239                     d2e[2][2][2] += f*mx*my*d3mz /(hzed*hzed*hzed);
08240                 }
08241             }
08242         }
08243     }
08244
08245     /* Monopole Force */
08246     force[0] = e[0]*c;
08247     force[1] = e[1]*c;
08248     force[2] = e[2]*c;
08249
08250     /* Dipole Force */
08251     force[0] -= de[0][0]*ux+de[1][0]*uy+de[2][0]*uz;
08252     force[1] -= de[1][0]*ux+de[1][1]*uy+de[2][1]*uz;
08253     force[2] -= de[2][0]*ux+de[2][1]*uy+de[2][2]*uz;
08254
08255     /* Quadrupole Force */
08256     force[0] += d2e[0][0][0]*qxx
08257              +  d2e[1][0][0]*qyx*2.0+d2e[1][1][0]*qyy
08258              +  d2e[2][0][0]*qzx*2.0+d2e[2][1][0]*qzy*2.0+d2e[2][2][0]*qzz;
08259     force[1] += d2e[0][0][1]*qxx
08260              +  d2e[1][0][1]*qyx*2.0+d2e[1][1][1]*qyy
08261              +  d2e[2][0][1]*qzx*2.0+d2e[2][1][1]*qzy*2.0+d2e[2][2][1]*qzz;
08262     force[2] += d2e[0][0][2]*qxx
08263              +  d2e[1][0][2]*qyx*2.0+d2e[1][1][2]*qyy
08264              +  d2e[2][0][2]*qzx*2.0+d2e[2][1][2]*qzy*2.0+d2e[2][2][2]*qzz;
08265
08266     /* Dipole Torque */
08267     torque[0] = uy * e[2] - uz * e[1];
08268     torque[1] = uz * e[0] - ux * e[2];
```

```
08269     torque[2] = ux * e[1] - uy * e[0];
08270     /* Quadrupole Torque */
08271     de[0][1] = de[1][0];
08272     de[0][2] = de[2][0];
08273     de[1][2] = de[2][1];
08274     torque[0] -= 2.0*(qyx*de[0][2] + qyy*de[1][2] + qyz*de[2][2]
08275                     - qzx*de[0][1] - qzy*de[1][1] - qzz*de[2][1]);
08276     torque[1] -= 2.0*(qzx*de[0][0] + qzy*de[1][0] + qzz*de[2][0]
08277                     - qxx*de[0][2] - qxy*de[1][2] - qxz*de[2][2]);
08278     torque[2] -= 2.0*(qxx*de[0][1] + qxy*de[1][1] + qxz*de[2][1]
08279                     - qyx*de[0][0] - qyy*de[1][0] - qyz*de[2][0]);
08280
08281
08282     /* printf(" qPhi Force %f %f %f\n", force[0], force[1], force[2]);
08283        printf(" qPhi Torque %f %f %f\n", torque[0], torque[1], torque[2]); */
08284 }
08285
08286 VPUBLIC void Vpmg_ibPermanentMultipoleForce(Vpmg *thee, int atomID,
08287                                             double force[3]) {
08288
08289     Valist *alist;
08290     Vacc *acc;
08291     Vpbe *pbe;
08292     Vatom *atom;
08293     Vsurf_Meth srfm;
08294
08295     /* Grid variables */
08296     double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
08297     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;
08298     double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
08299     double izmagic;
08300     int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
08301
08302     VASSERT(thee != VNULL);
08303
08304     /* Nonlinear PBE is not implemented for AMOEBA */
08305     VASSERT(!thee->pmgp->nonlin);
08306
08307     acc = thee->pbe->acc;
08308     srfm = thee->surfMeth;
08309     atom = Valist_getAtom(thee->pbe->alist, atomID);
08310
08311     /* Currently all atoms must be in the same partition. */
08312
08313     VASSERT(atom->partID != 0);
08314     apos = Vatom_getPosition(atom);
08315     arad = Vatom_getRadius(atom);
08316
08317     /* Reset force */
08318     force[0] = 0.0;
08319     force[1] = 0.0;
08320     force[2] = 0.0;
08321
08322     /* Get PBE info */
08323     pbe = thee->pbe;
08324     acc = pbe->acc;
08325     alist = pbe->alist;
08326     irad = Vpbe_getMaxIonRadius(pbe);
08327     zkappa2 = Vpbe_getZkappa2(pbe);
08328     izmagic = 1.0/Vpbe_getZmagic(pbe);
08329
08330     /* Should be a check for this further up. */
08331     VASSERT (zkappa2 > VPMGSMALL);
08332
08333     /* Mesh info */
08334     nx = thee->pmgp->nx;
08335     ny = thee->pmgp->ny;
08336     nz = thee->pmgp->nz;
08337     hx = thee->pmgp->hx;
08338     hy = thee->pmgp->hy;
08339     hzed = thee->pmgp->hzed;
08340     xlen = thee->pmgp->xlen;
08341     ylen = thee->pmgp->ylen;
08342     zlen = thee->pmgp->zlen;
08343     xmin = thee->pmgp->xmin;
08344     ymin = thee->pmgp->ymin;
08345     zmin = thee->pmgp->zmin;
08346     xmax = thee->pmgp->xmax;
08347     ymax = thee->pmgp->ymax;
08348     zmax = thee->pmgp->zmax;
08349
```

```
08350     /* Make sure we're on the grid */
08351     if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
08352       (apos[1]<=ymin) || (apos[1]>=ymax)  || \
08353       (apos[2]<=zmin) || (apos[2]>=zmax)) {
08354         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]);
08355         Vnm_print(2, "ibPermanentMultipoleForce:  xmin = %g, xmax = %g\n", xmin, xmax);
08356         Vnm_print(2, "ibPermanentMultipoleForce:  ymin = %g, ymax = %g\n", ymin, ymax);
08357         Vnm_print(2, "ibPermanentMultipoleForce:  zmin = %g, zmax = %g\n", zmin, zmax);
08358         fflush(stderr);
08359     } else {
08360
08361         /* Convert the atom position to grid reference frame */
08362         position[0] = apos[0] - xmin;
08363         position[1] = apos[1] - ymin;
08364         position[2] = apos[2] - zmin;
08365
08366         /* Integrate over points within this atom's (inflated) radius */
08367         rtot = (irad + arad + thee->splineWin);
08368         rtot2 = VSQR(rtot);
08369         dx = rtot + 0.5*hx;
08370         imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
08371         imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
08372         for (i=imin; i<=imax; i++) {
08373             dx2 = VSQR(position[0] - hx*i);
08374             if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
08375             else dy = 0.5*hy;
08376             jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
08377             jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
08378             for (j=jmin; j<=jmax; j++) {
08379                 dy2 = VSQR(position[1] - hy*j);
08380                 if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
08381                 else dz = 0.5*hzed;
08382                 kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
08383                 kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
08384                 for (k=kmin; k<=kmax; k++) {
08385                     dz2 = VSQR(k*hzed - position[2]);
08386                     /* See if grid point is inside ivdw radius and set ccf
08387                      * accordingly (do spline assignment here) */
08388                     if ((dz2 + dy2 + dx2) <= rtot2) {
08389                         gpos[0] = i*hx + xmin;
08390                         gpos[1] = j*hy + ymin;
08391                         gpos[2] = k*hzed + zmin;
08392                         Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, irad, atom, tgrad);
08393                         fmag = VSQR(thee->u[IJK(i,j,k)])*thee->kappa[IJK(i,j,k)];
08394                         force[0] += (zkappa2*fmag*tgrad[0]);
08395                         force[1] += (zkappa2*fmag*tgrad[1]);
08396                         force[2] += (zkappa2*fmag*tgrad[2]);
08397                     }
08398                 } /* k loop */
08399             } /* j loop */
08400         } /* i loop */
08401     }
08402
08403     force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
08404     force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
08405     force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
08406
08407 }
08408
08409 VPUBLIC void Vpmg_dbPermanentMultipoleForce(Vpmg *thee, int atomID,
08410                                             double force[3]) {
08411
08412     Vacc *acc;
08413     Vpbe *pbe;
08414     Vatom *atom;
08415     Vsurf_Meth srfm;
08416
08417     double *apos, position[3], arad, hx, hy, hzed, izmagic, deps, depsi;
08418     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2, epsp;
08419     double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
08420     double *u, Hxijk, Hyijk, Hzijk, Hxim1jk, Hyijm1k, Hzijkm1;
08421     double dHxijk[3], dHyijk[3], dHzijk[3], dHxim1jk[3], dHyijm1k[3];
08422     double dHzijkm1[3];
08423     int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
08424
08425     VASSERT(thee != VNULL);
08426
08427     acc = thee->pbe->acc;
08428     srfm = thee->surfMeth;
08429     atom = Valist_getAtom(thee->pbe->alist, atomID);
```

```
08430
08431     /* Currently all atoms must be in the same partition. */
08432
08433     VASSERT(atom->partID != 0);
08434     arad = Vatom_getRadius(atom);
08435     apos = Vatom_getPosition(atom);
08436
08437     /* Reset force */
08438     force[0] = 0.0;
08439     force[1] = 0.0;
08440     force[2] = 0.0;
08441
08442     /* Get PBE info */
08443     pbe = thee->pbe;
08444     acc = pbe->acc;
08445     epsp = Vpbe_getSoluteDiel(pbe);
08446     epsw = Vpbe_getSolventDiel(pbe);
08447     kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*Vunit_kb;
08448     izmagic = 1.0/Vpbe_getZmagic(pbe);
08449
08450
08451     deps = (epsw - epsp);
08452     depsi = 1.0/deps;
08453
08454     VASSERT(VABS(deps) > VPMGSMALL);
08455
08456     /* Mesh info */
08457     nx = thee->pmgp->nx;
08458     ny = thee->pmgp->ny;
08459     nz = thee->pmgp->nz;
08460     hx = thee->pmgp->hx;
08461     hy = thee->pmgp->hy;
08462     hzed = thee->pmgp->hzed;
08463     xlen = thee->pmgp->xlen;
08464     ylen = thee->pmgp->ylen;
08465     zlen = thee->pmgp->zlen;
08466     xmin = thee->pmgp->xmin;
08467     ymin = thee->pmgp->ymin;
08468     zmin = thee->pmgp->zmin;
08469     xmax = thee->pmgp->xmax;
08470     ymax = thee->pmgp->ymax;
08471     zmax = thee->pmgp->zmax;
08472     u = thee->u;
08473
08474     /* Make sure we're on the grid */
08475     if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
08476       (apos[1]<=ymin) || (apos[1]>=ymax)  || \
08477       (apos[2]<=zmin) || (apos[2]>=zmax)) {
08478         Vnm_print(2, "dbPermanentMultipoleForce:  Atom at (%4.3f, %4.3f, %4.3f) is off the mesh
      (ignoring):\n", apos[0], apos[1], apos[2]);
08479         Vnm_print(2, "dbPermanentMultipoleForce:  xmin = %g, xmax = %g\n", xmin, xmax);
08480         Vnm_print(2, "dbPermanentMultipoleForce:  ymin = %g, ymax = %g\n", ymin, ymax);
08481         Vnm_print(2, "dbPermanentMultipoleForce:  zmin = %g, zmax = %g\n", zmin, zmax);
08482         fflush(stderr);
08483     } else {
08484
08485         /* Convert the atom position to grid reference frame */
08486         position[0] = apos[0] - xmin;
08487         position[1] = apos[1] - ymin;
08488         position[2] = apos[2] - zmin;
08489
08490         /* Integrate over points within this atom's (inflated) radius */
08491         rtot = (arad + thee->splineWin);
08492         rtot2 = VSQR(rtot);
08493         dx = rtot/hx;
08494         imin = (int)floor((position[0]-rtot)/hx);
08495         if (imin < 1) {
08496             Vnm_print(2, "dbPermanentMultipoleForce:  Atom off grid!\n");
08497             return;
08498         }
08499         imax = (int)ceil((position[0]+rtot)/hx);
08500         if (imax > (nx-2)) {
08501             Vnm_print(2, "dbPermanentMultipoleForce:  Atom off grid!\n");
08502             return;
08503         }
08504         jmin = (int)floor((position[1]-rtot)/hy);
08505         if (jmin < 1) {
08506             Vnm_print(2, "dbPermanentMultipoleForce:  Atom off grid!\n");
08507             return;
08508         }
08509         jmax = (int)ceil((position[1]+rtot)/hy);
```

```
08510         if (jmax > (ny-2)) {
08511             Vnm_print(2, "dbPermanentMultipoleForce:  Atom off grid!\n");
08512             return;
08513         }
08514         kmin = (int)floor((position[2]-rtot)/hzed);
08515         if (kmin < 1) {
08516             Vnm_print(2, "dbPermanentMultipoleForce:  Atom off grid!\n");
08517             return;
08518         }
08519         kmax = (int)ceil((position[2]+rtot)/hzed);
08520         if (kmax > (nz-2)) {
08521             Vnm_print(2, "dbPermanentMultipoleForce:  Atom off grid!\n");
08522             return;
08523         }
08524         for (i=imin; i<=imax; i++) {
08525             for (j=jmin; j<=jmax; j++) {
08526                 for (k=kmin; k<=kmax; k++) {
08527                     /* i,j,k */
08528                     gpos[0] = (i+0.5)*hx + xmin;
08529                     gpos[1] = j*hy + ymin;
08530                     gpos[2] = k*hzed + zmin;
08531                     Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
08532                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
08533                             atom, dHxijk);
08534                     for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
08535                     gpos[0] = i*hx + xmin;
08536                     gpos[1] = (j+0.5)*hy + ymin;
08537                     gpos[2] = k*hzed + zmin;
08538                     Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
08539                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
08540                             atom, dHyijk);
08541                     for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
08542                     gpos[0] = i*hx + xmin;
08543                     gpos[1] = j*hy + ymin;
08544                     gpos[2] = (k+0.5)*hzed + zmin;
08545                     Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
08546                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
08547                             atom, dHzijk);
08548                     for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
08549                     /* i-1,j,k */
08550                     gpos[0] = (i-0.5)*hx + xmin;
08551                     gpos[1] = j*hy + ymin;
08552                     gpos[2] = k*hzed + zmin;
08553                     Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
08554                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
08555                             atom, dHxim1jk);
08556                     for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
08557                     /* i,j-1,k */
08558                     gpos[0] = i*hx + xmin;
08559                     gpos[1] = (j-0.5)*hy + ymin;
08560                     gpos[2] = k*hzed + zmin;
08561                     Hyijm1k = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
08562                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
08563                             atom, dHyijm1k);
08564                     for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;
08565                     /* i,j,k-1 */
08566                     gpos[0] = i*hx + xmin;
08567                     gpos[1] = j*hy + ymin;
08568                     gpos[2] = (k-0.5)*hzed + zmin;
08569                     Hzijkm1 = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
08570                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
08571                             atom, dHzijkm1);
08572                     for (l=0; l<3; l++) dHzijkm1[l] *= Hzijkm1;
08573                     dbFmag = u[IJK(i,j,k)];
08574                     tgrad[0] =
08575                        (dHxijk[0]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
08576                      +  dHxim1jk[0]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
08577                      + (dHyijk[0]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
08578                      +  dHyijm1k[0]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
08579                      + (dHzijk[0]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
08580                      + dHzijkm1[0]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
08581                     tgrad[1] =
08582                        (dHxijk[1]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
08583                      +  dHxim1jk[1]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
08584                      + (dHyijk[1]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
08585                      +  dHyijm1k[1]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
08586                      + (dHzijk[1]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
08587                      + dHzijkm1[1]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
08588                     tgrad[2] =
08589                        (dHxijk[2]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
08590                      +  dHxim1jk[2]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
```

```
08591                     + (dHyijk[2]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
08592                     +  dHyijm1k[2]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
08593                     + (dHzijk[2]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
08594                     + dHzijkm1[2]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
08595                     force[0] += (dbFmag*tgrad[0]);
08596                     force[1] += (dbFmag*tgrad[1]);
08597                     force[2] += (dbFmag*tgrad[2]);
08598                 } /* k loop */
08599             } /* j loop */
08600         } /* i loop */
08601         force[0] = -force[0]*hx*hy*hzed*deps*0.5*izmagic;
08602         force[1] = -force[1]*hx*hy*hzed*deps*0.5*izmagic;
08603         force[2] = -force[2]*hx*hy*hzed*deps*0.5*izmagic;
08604     }
08605 }
08606
08607 VPUBLIC void Vpmg_qfDirectPolForce(Vpmg *thee, Vgrid* perm, Vgrid *induced,
08608                                    int atomID, double force[3], double torque[3]) {
08609
08610     Vatom *atom;
08611     Vpbe *pbe;
08612     double f, fp, *u, *up, *apos, position[3];
08613
08614     /* Grid variables */
08615     int nx,ny,nz;
08616     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
08617     double hx, hy, hzed, ifloat, jfloat, kfloat;
08618
08619     /* B-spline weights */
08620     double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz, d3mx, d3my, d3mz;
08621     double mi, mj, mk;
08622
08623     /* Loop indeces */
08624     int i, j, k, ii, jj, kk;
08625     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
08626
08627     /* Permanent potential, field, field gradient and 2nd field gradient */
08628     double pot, e[3], de[3][3], d2e[3][3][3];
08629     /* Induced dipole field */
08630     double dep[3][3];
08631
08632     /* Permanent multipole components */
08633     double *dipole, *quad;
08634     double c, ux, uy, uz, qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
08635     double uix, uiy, uiz;
08636
08637     VASSERT(thee != VNULL);
08638     VASSERT(induced != VNULL); /* the potential due to permanent multipoles.*/
08639     VASSERT(induced != VNULL); /* the potential due to local induced dipoles.*/
08640     VASSERT(thee->pbe != VNULL);
08641     VASSERT(thee->pbe->alist != VNULL);
08642
08643     atom = Valist_getAtom(thee->pbe->alist, atomID);
08644     VASSERT(atom->partID != 0); /* all atoms must be in the same partition.*/
08645     apos = Vatom_getPosition(atom);
08646
08647     c = Vatom_getCharge(atom);
08648     dipole = Vatom_getDipole(atom);
08649     ux = dipole[0];
08650     uy = dipole[1];
08651     uz = dipole[2];
08652     quad = Vatom_getQuadrupole(atom);
08653     qxx = quad[0]/3.0;
08654     qxy = quad[1]/3.0;
08655     qxz = quad[2]/3.0;
08656     qyx = quad[3]/3.0;
08657     qyy = quad[4]/3.0;
08658     qyz = quad[5]/3.0;
08659     qzx = quad[6]/3.0;
08660     qzy = quad[7]/3.0;
08661     qzz = quad[8]/3.0;
08662
08663     dipole = Vatom_getInducedDipole(atom);
08664     uix = dipole[0];
08665     uiy = dipole[1];
08666     uiz = dipole[2];
08667
08668     /* Reset Field Gradients */
08669     pot = 0.0;
08670     for (i=0;i<3;i++){
08671        e[i] = 0.0;
```

```
08672         for (j=0;j<3;j++){
08673            de[i][j] = 0.0;
08674            dep[i][j] = 0.0;
08675            for (k=0;k<3;k++){
08676               d2e[i][j][k] = 0.0;
08677            }
08678         }
08679      }
08680
08681      /* Mesh info */
08682      nx = thee->pmgp->nx;
08683      ny = thee->pmgp->ny;
08684      nz = thee->pmgp->nz;
08685      hx = thee->pmgp->hx;
08686      hy = thee->pmgp->hy;
08687      hzed = thee->pmgp->hzed;
08688      xlen = thee->pmgp->xlen;
08689      ylen = thee->pmgp->ylen;
08690      zlen = thee->pmgp->zlen;
08691      xmin = thee->pmgp->xmin;
08692      ymin = thee->pmgp->ymin;
08693      zmin = thee->pmgp->zmin;
08694      xmax = thee->pmgp->xmax;
08695      ymax = thee->pmgp->ymax;
08696      zmax = thee->pmgp->zmax;
08697      u = induced->data;
08698      up = perm->data;
08699
08700      /* Make sure we're on the grid */
08701      if ((apos[0]<=(xmin+2*hx))   || (apos[0]>=(xmax-2*hx)) \
08702       || (apos[1]<=(ymin+2*hy))   || (apos[1]>=(ymax-2*hy)) \
08703       || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
08704         Vnm_print(2, "qfDirectPolForce:  Atom off the mesh (ignoring) %6.3f %6.3f %6.3f\n", apos[0],
    apos[1], apos[2]);
08705         fflush(stderr);
08706
08707      } else {
08708
08709         /* Convert the atom position to grid coordinates */
08710         position[0] = apos[0] - xmin;
08711         position[1] = apos[1] - ymin;
08712         position[2] = apos[2] - zmin;
08713         ifloat = position[0]/hx;
08714         jfloat = position[1]/hy;
08715         kfloat = position[2]/hzed;
08716         ip1 = (int)ceil(ifloat);
08717         ip2 = ip1 + 2;
08718         im1 = (int)floor(ifloat);
08719         im2 = im1 - 2;
08720         jp1 = (int)ceil(jfloat);
08721         jp2 = jp1 + 2;
08722         jm1 = (int)floor(jfloat);
08723         jm2 = jm1 - 2;
08724         kp1 = (int)ceil(kfloat);
08725         kp2 = kp1 + 2;
08726         km1 = (int)floor(kfloat);
08727         km2 = km1 - 2;
08728
08729         /* This step shouldn't be necessary, but it saves nasty debugging
08730          * later on if something goes wrong */
08731         ip2 = VMIN2(ip2,nx-1);
08732         ip1 = VMIN2(ip1,nx-1);
08733         im1 = VMAX2(im1,0);
08734         im2 = VMAX2(im2,0);
08735         jp2 = VMIN2(jp2,ny-1);
08736         jp1 = VMIN2(jp1,ny-1);
08737         jm1 = VMAX2(jm1,0);
08738         jm2 = VMAX2(jm2,0);
08739         kp2 = VMIN2(kp2,nz-1);
08740         kp1 = VMIN2(kp1,nz-1);
08741         km1 = VMAX2(km1,0);
08742         km2 = VMAX2(km2,0);
08743
08744         for (ii=im2; ii<=ip2; ii++) {
08745             mi = VFCHI4(ii,ifloat);
08746             mx = bspline4(mi);
08747             dmx = dbspline4(mi);
08748             d2mx = d2bspline4(mi);
08749             d3mx = d3bspline4(mi);
08750             for (jj=jm2; jj<=jp2; jj++) {
08751                 mj = VFCHI4(jj,jfloat);
```

```
08752                 my = bspline4(mj);
08753                 dmy = dbspline4(mj);
08754                 d2my = d2bspline4(mj);
08755                 d3my = d3bspline4(mj);
08756                 for (kk=km2; kk<=kp2; kk++) {
08757                     mk = VFCHI4(kk,kfloat);
08758                     mz = bspline4(mk);
08759                     dmz = dbspline4(mk);
08760                     d2mz = d2bspline4(mk);
08761                     d3mz = d3bspline4(mk);
08762                     f = u[IJK(ii,jj,kk)];
08763                     fp = up[IJK(ii,jj,kk)];
08764                     /* The potential */
08765                     pot  += f*mx*my*mz;
08766                     /* The field */
08767                     e[0] += f*dmx*my*mz/hx;
08768                     e[1] += f*mx*dmy*mz/hy;
08769                     e[2] += f*mx*my*dmz/hzed;
08770                     /* The gradient of the field */
08771                     de[0][0] += f*d2mx*my*mz/(hx*hx);
08772                     de[1][0] += f*dmx*dmy*mz/(hy*hx);
08773                     de[1][1] += f*mx*d2my*mz/(hy*hy);
08774                     de[2][0] += f*dmx*my*dmz/(hx*hzed);
08775                     de[2][1] += f*mx*dmy*dmz/(hy*hzed);
08776                     de[2][2] += f*mx*my*d2mz/(hzed*hzed);
08777                     /* The gradient of the (permanent) field */
08778                     dep[0][0] += fp*d2mx*my*mz/(hx*hx);
08779                     dep[1][0] += fp*dmx*dmy*mz/(hy*hx);
08780                     dep[1][1] += fp*mx*d2my*mz/(hy*hy);
08781                     dep[2][0] += fp*dmx*my*dmz/(hx*hzed);
08782                     dep[2][1] += fp*mx*dmy*dmz/(hy*hzed);
08783                     dep[2][2] += fp*mx*my*d2mz/(hzed*hzed);
08784                     /* The 2nd gradient of the field
08785                        VxVxVa */
08786                     d2e[0][0][0] += f*d3mx*my*mz /(hx*hx*hx);
08787                     d2e[0][0][1] += f*d2mx*dmy*mz/(hx*hy*hx);
08788                     d2e[0][0][2] += f*d2mx*my*dmz/(hx*hx*hzed);
08789                     /* VyVxVa */
08790                     d2e[1][0][0] += f*d2mx*dmy*mz/(hx*hx*hy);
08791                     d2e[1][0][1] += f*dmx*d2my*mz/(hx*hy*hy);
08792                     d2e[1][0][2] += f*dmx*dmy*dmz/(hx*hy*hzed);
08793                     /* VyVyVa */
08794                     d2e[1][1][0] += f*dmx*d2my*mz/(hx*hy*hy);
08795                     d2e[1][1][1] += f*mx*d3my*mz /(hy*hy*hy);
08796                     d2e[1][1][2] += f*mx*d2my*dmz/(hy*hy*hzed);
08797                     /* VzVxVa */
08798                     d2e[2][0][0] += f*d2mx*my*dmz/(hx*hx*hzed);
08799                     d2e[2][0][1] += f*dmx*dmy*dmz/(hx*hy*hzed);
08800                     d2e[2][0][2] += f*dmx*my*d2mz/(hx*hzed*hzed);
08801                     /* VzVyVa */
08802                     d2e[2][1][0] += f*dmx*dmy*dmz/(hx*hy*hzed);
08803                     d2e[2][1][1] += f*mx*d2my*dmz/(hy*hy*hzed);
08804                     d2e[2][1][2] += f*mx*dmy*d2mz/(hy*hzed*hzed);
08805                     /* VzVzVa */
08806                     d2e[2][2][0] += f*dmx*my*d2mz/(hx*hzed*hzed);
08807                     d2e[2][2][1] += f*mx*dmy*d2mz/(hy*hzed*hzed);
08808                     d2e[2][2][2] += f*mx*my*d3mz /(hzed*hzed*hzed);
08809                 }
08810             }
08811         }
08812     }
08813
08814     /* force on permanent multipole due to induced reaction field */
08815
08816     /* Monopole Force */
08817     force[0] = e[0]*c;
08818     force[1] = e[1]*c;
08819     force[2] = e[2]*c;
08820
08821     /* Dipole Force */
08822     force[0] -= de[0][0]*ux+de[1][0]*uy+de[2][0]*uz;
08823     force[1] -= de[1][0]*ux+de[1][1]*uy+de[2][1]*uz;
08824     force[2] -= de[2][0]*ux+de[2][1]*uy+de[2][2]*uz;
08825
08826     /* Quadrupole Force */
08827     force[0] += d2e[0][0][0]*qxx
08828              +  d2e[1][0][0]*qyx*2.0+d2e[1][1][0]*qyy
08829              +  d2e[2][0][0]*qzx*2.0+d2e[2][1][0]*qzy*2.0+d2e[2][2][0]*qzz;
08830     force[1] += d2e[0][0][1]*qxx
08831              +  d2e[1][0][1]*qyx*2.0+d2e[1][1][1]*qyy
08832              +  d2e[2][0][1]*qzx*2.0+d2e[2][1][1]*qzy*2.0+d2e[2][2][1]*qzz;
```

```
08833     force[2] += d2e[0][0][2]*qxx
08834              +  d2e[1][0][2]*qyx*2.0+d2e[1][1][2]*qyy
08835              +  d2e[2][0][2]*qzx*2.0+d2e[2][1][2]*qzy*2.0+d2e[2][2][2]*qzz;
08836
08837     /* torque on permanent mulitpole due to induced reaction field */
08838
08839     /* Dipole Torque */
08840     torque[0] = uy * e[2] - uz * e[1];
08841     torque[1] = uz * e[0] - ux * e[2];
08842     torque[2] = ux * e[1] - uy * e[0];
08843
08844     /* Quadrupole Torque */
08845     /* Tx = -2.0*(Sum_a (Qya*dEaz) + Sum_b (Qzb*dEby))
08846        Ty = -2.0*(Sum_a (Qza*dEax) + Sum_b (Qxb*dEbz))
08847        Tz = -2.0*(Sum_a (Qxa*dEay) + Sum_b (Qyb*dEbx))  */
08848     de[0][1] = de[1][0];
08849     de[0][2] = de[2][0];
08850     de[1][2] = de[2][1];
08851     torque[0] -= 2.0*(qyx*de[0][2] + qyy*de[1][2] + qyz*de[2][2]
08852                     - qzx*de[0][1] - qzy*de[1][1] - qzz*de[2][1]);
08853     torque[1] -= 2.0*(qzx*de[0][0] + qzy*de[1][0] + qzz*de[2][0]
08854                     - qxx*de[0][2] - qxy*de[1][2] - qxz*de[2][2]);
08855     torque[2] -= 2.0*(qxx*de[0][1] + qxy*de[1][1] + qxz*de[2][1]
08856                     - qyx*de[0][0] - qyy*de[1][0] - qyz*de[2][0]);
08857
08858     /* force on induced dipole due to permanent reaction field */
08859
08860     force[0] -= dep[0][0]*uix+dep[1][0]*uiy+dep[2][0]*uiz;
08861     force[1] -= dep[1][0]*uix+dep[1][1]*uiy+dep[2][1]*uiz;
08862     force[2] -= dep[2][0]*uix+dep[2][1]*uiy+dep[2][2]*uiz;
08863
08864     force[0] = 0.5 * force[0];
08865     force[1] = 0.5 * force[1];
08866     force[2] = 0.5 * force[2];
08867     torque[0] = 0.5 * torque[0];
08868     torque[1] = 0.5 * torque[1];
08869     torque[2] = 0.5 * torque[2];
08870
08871     /* printf(" qPhi Force %f %f %f\n", force[0], force[1], force[2]);
08872       printf(" qPhi Torque %f %f %f\n", torque[0], torque[1], torque[2]); */
08873 }
08874
08875 VPUBLIC void Vpmg_qfNLDirectPolForce(Vpmg *thee, Vgrid *perm, Vgrid *nlInduced,
08876                                      int atomID, double force[3], double torque[3]) {
08877
08878     Vatom *atom;
08879     double *apos, *dipole, *quad, position[3], hx, hy, hzed;
08880     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
08881     double pot, e[3],de[3][3],dep[3][3],d2e[3][3][3];
08882     double mx, my, mz, dmx, dmy, dmz, mi, mj, mk;
08883     double d2mx, d2my, d2mz, d3mx, d3my, d3mz;
08884     double *u, *up, charge, ifloat, jfloat, kfloat;
08885     double f, fp, c, ux, uy, uz, qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy, qzz;
08886     double uix, uiy, uiz;
08887     int i,j,k,nx, ny, nz, im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1;
08888     int kp1, kp2, ii, jj, kk;
08889
08890     VASSERT(thee != VNULL);
08891     VASSERT(perm != VNULL);      /* potential due to permanent multipoles. */
08892     VASSERT(nlInduced != VNULL); /* potential due to non-local induced dipoles */
08893     VASSERT(!thee->pmgp->nonlin); /* Nonlinear PBE is not implemented for AMOEBA */
08894
08895     atom = Valist_getAtom(thee->pbe->alist, atomID);
08896     VASSERT(atom->partID != 0);   /* Currently all atoms must be in the same partition. */
08897     apos = Vatom_getPosition(atom);
08898
08899     c = Vatom_getCharge(atom);
08900     dipole = Vatom_getDipole(atom);
08901     ux = dipole[0];
08902     uy = dipole[1];
08903     uz = dipole[2];
08904     quad = Vatom_getQuadrupole(atom);
08905     qxx = quad[0]/3.0;
08906     qxy = quad[1]/3.0;
08907     qxz = quad[2]/3.0;
08908     qyx = quad[3]/3.0;
08909     qyy = quad[4]/3.0;
08910     qyz = quad[5]/3.0;
08911     qzx = quad[6]/3.0;
08912     qzy = quad[7]/3.0;
08913     qzz = quad[8]/3.0;
```

```
08914
08915     dipole = Vatom_getNLInducedDipole(atom);
08916     uix = dipole[0];
08917     uiy = dipole[1];
08918     uiz = dipole[2];
08919
08920     /* Reset Field Gradients */
08921     pot = 0.0;
08922     for (i=0;i<3;i++){
08923        e[i] = 0.0;
08924        for (j=0;j<3;j++){
08925           de[i][j] = 0.0;
08926           dep[i][j] = 0.0;
08927           for (k=0;k<3;k++){
08928              d2e[i][j][k] = 0.0;
08929           }
08930        }
08931     }
08932
08933     /* Mesh info */
08934     nx = thee->pmgp->nx;
08935     ny = thee->pmgp->ny;
08936     nz = thee->pmgp->nz;
08937     hx = thee->pmgp->hx;
08938     hy = thee->pmgp->hy;
08939     hzed = thee->pmgp->hzed;
08940     xlen = thee->pmgp->xlen;
08941     ylen = thee->pmgp->ylen;
08942     zlen = thee->pmgp->zlen;
08943     xmin = thee->pmgp->xmin;
08944     ymin = thee->pmgp->ymin;
08945     zmin = thee->pmgp->zmin;
08946     xmax = thee->pmgp->xmax;
08947     ymax = thee->pmgp->ymax;
08948     zmax = thee->pmgp->zmax;
08949     u = nlInduced->data;
08950     up = perm->data;
08951
08952
08953     /* Make sure we're on the grid */
08954     if ((apos[0]<=(xmin+2*hx))   || (apos[0]>=(xmax-2*hx)) \
08955      || (apos[1]<=(ymin+2*hy))   || (apos[1]>=(ymax-2*hy)) \
08956      || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
08957        Vnm_print(2, "qfNLDirectMultipoleForce:  Atom off the mesh (ignoring) %6.3f %6.3f %6.3f\n",
       apos[0], apos[1], apos[2]);
08958     } else {
08959
08960        /* Convert the atom position to grid coordinates */
08961        position[0] = apos[0] - xmin;
08962        position[1] = apos[1] - ymin;
08963        position[2] = apos[2] - zmin;
08964        ifloat = position[0]/hx;
08965        jfloat = position[1]/hy;
08966        kfloat = position[2]/hzed;
08967        ip1 = (int)ceil(ifloat);
08968        ip2 = ip1 + 2;
08969        im1 = (int)floor(ifloat);
08970        im2 = im1 - 2;
08971        jp1 = (int)ceil(jfloat);
08972        jp2 = jp1 + 2;
08973        jm1 = (int)floor(jfloat);
08974        jm2 = jm1 - 2;
08975        kp1 = (int)ceil(kfloat);
08976        kp2 = kp1 + 2;
08977        km1 = (int)floor(kfloat);
08978        km2 = km1 - 2;
08979
08980        /* This step shouldn't be necessary, but it saves nasty debugging
08981         * later on if something goes wrong */
08982        ip2 = VMIN2(ip2,nx-1);
08983        ip1 = VMIN2(ip1,nx-1);
08984        im1 = VMAX2(im1,0);
08985        im2 = VMAX2(im2,0);
08986        jp2 = VMIN2(jp2,ny-1);
08987        jp1 = VMIN2(jp1,ny-1);
08988        jm1 = VMAX2(jm1,0);
08989        jm2 = VMAX2(jm2,0);
08990        kp2 = VMIN2(kp2,nz-1);
08991        kp1 = VMIN2(kp1,nz-1);
08992        km1 = VMAX2(km1,0);
08993        km2 = VMAX2(km2,0);
```

```
08994
08995         for (ii=im2; ii<=ip2; ii++) {
08996             mi = VFCHI4(ii,ifloat);
08997             mx = bspline4(mi);
08998             dmx = dbspline4(mi);
08999             d2mx = d2bspline4(mi);
09000             d3mx = d3bspline4(mi);
09001             for (jj=jm2; jj<=jp2; jj++) {
09002                 mj = VFCHI4(jj,jfloat);
09003                 my = bspline4(mj);
09004                 dmy = dbspline4(mj);
09005                 d2my = d2bspline4(mj);
09006                 d3my = d3bspline4(mj);
09007                 for (kk=km2; kk<=kp2; kk++) {
09008                     mk = VFCHI4(kk,kfloat);
09009                     mz = bspline4(mk);
09010                     dmz = dbspline4(mk);
09011                     d2mz = d2bspline4(mk);
09012                     d3mz = d3bspline4(mk);
09013                     f = u[IJK(ii,jj,kk)];
09014                     fp = up[IJK(ii,jj,kk)];
09015                     /* The potential */
09016                     pot  += f*mx*my*mz;
09017                     /* The field */
09018                     e[0] += f*dmx*my*mz/hx;
09019                     e[1] += f*mx*dmy*mz/hy;
09020                     e[2] += f*mx*my*dmz/hzed;
09021                     /* The gradient of the field */
09022                     de[0][0] += f*d2mx*my*mz/(hx*hx);
09023                     de[1][0] += f*dmx*dmy*mz/(hy*hx);
09024                     de[1][1] += f*mx*d2my*mz/(hy*hy);
09025                     de[2][0] += f*dmx*my*dmz/(hx*hzed);
09026                     de[2][1] += f*mx*dmy*dmz/(hy*hzed);
09027                     de[2][2] += f*mx*my*d2mz/(hzed*hzed);
09028                     /* The gradient of the (permanent) field */
09029                     dep[0][0] += fp*d2mx*my*mz/(hx*hx);
09030                     dep[1][0] += fp*dmx*dmy*mz/(hy*hx);
09031                     dep[1][1] += fp*mx*d2my*mz/(hy*hy);
09032                     dep[2][0] += fp*dmx*my*dmz/(hx*hzed);
09033                     dep[2][1] += fp*mx*dmy*dmz/(hy*hzed);
09034                     dep[2][2] += fp*mx*my*d2mz/(hzed*hzed);
09035                     /* The 2nd gradient of the field */
09036                     /* VxVxVa */
09037                     d2e[0][0][0] += f*d3mx*my*mz /(hx*hx*hx);
09038                     d2e[0][0][1] += f*d2mx*dmy*mz/(hx*hy*hx);
09039                     d2e[0][0][2] += f*d2mx*my*dmz/(hx*hx*hzed);
09040                     /* VyVxVa */
09041                     d2e[1][0][0] += f*d2mx*dmy*mz/(hx*hx*hy);
09042                     d2e[1][0][1] += f*dmx*d2my*mz/(hx*hy*hy);
09043                     d2e[1][0][2] += f*dmx*dmy*dmz/(hx*hy*hzed);
09044                     /* VyVyVa */
09045                     d2e[1][1][0] += f*dmx*d2my*mz/(hx*hy*hy);
09046                     d2e[1][1][1] += f*mx*d3my*mz /(hy*hy*hy);
09047                     d2e[1][1][2] += f*mx*d2my*dmz/(hy*hy*hzed);
09048                     /* VzVxVa */
09049                     d2e[2][0][0] += f*d2mx*my*dmz/(hx*hx*hzed);
09050                     d2e[2][0][1] += f*dmx*dmy*dmz/(hx*hy*hzed);
09051                     d2e[2][0][2] += f*dmx*my*d2mz/(hx*hzed*hzed);
09052                     /* VzVyVa */
09053                     d2e[2][1][0] += f*dmx*dmy*dmz/(hx*hy*hzed);
09054                     d2e[2][1][1] += f*mx*d2my*dmz/(hy*hy*hzed);
09055                     d2e[2][1][2] += f*mx*dmy*d2mz/(hy*hzed*hzed);
09056                     /* VzVzVa */
09057                     d2e[2][2][0] += f*dmx*my*d2mz/(hx*hzed*hzed);
09058                     d2e[2][2][1] += f*mx*dmy*d2mz/(hy*hzed*hzed);
09059                     d2e[2][2][2] += f*mx*my*d3mz /(hzed*hzed*hzed);
09060                 }
09061             }
09062         }
09063     }
09064
09065     /* force on permanent multipole due to non-local induced reaction field */
09066
09067     /* Monopole Force */
09068     force[0] = e[0]*c;
09069     force[1] = e[1]*c;
09070     force[2] = e[2]*c;
09071
09072     /* Dipole Force */
09073     force[0] -= de[0][0]*ux+de[1][0]*uy+de[2][0]*uz;
09074     force[1] -= de[1][0]*ux+de[1][1]*uy+de[2][1]*uz;
```

```
09075     force[2] -= de[2][0]*ux+de[2][1]*uy+de[2][2]*uz;
09076
09077     /* Quadrupole Force */
09078     force[0] += d2e[0][0][0]*qxx
09079              +  d2e[1][0][0]*qyx*2.0+d2e[1][1][0]*qyy
09080              +  d2e[2][0][0]*qzx*2.0+d2e[2][1][0]*qzy*2.0+d2e[2][2][0]*qzz;
09081     force[1] += d2e[0][0][1]*qxx
09082              +  d2e[1][0][1]*qyx*2.0+d2e[1][1][1]*qyy
09083              +  d2e[2][0][1]*qzx*2.0+d2e[2][1][1]*qzy*2.0+d2e[2][2][1]*qzz;
09084     force[2] += d2e[0][0][2]*qxx
09085              +  d2e[1][0][2]*qyx*2.0+d2e[1][1][2]*qyy
09086              +  d2e[2][0][2]*qzx*2.0+d2e[2][1][2]*qzy*2.0+d2e[2][2][2]*qzz;
09087
09088     /* torque on permanent mulitpole due to non-local induced reaction field */
09089
09090     /* Dipole Torque */
09091     torque[0] = uy * e[2] - uz * e[1];
09092     torque[1] = uz * e[0] - ux * e[2];
09093     torque[2] = ux * e[1] - uy * e[0];
09094
09095     /* Quadrupole Torque */
09096     /* Tx = -2.0*(Sum_a (Qya*dEaz) + Sum_b (Qzb*dEby))
09097        Ty = -2.0*(Sum_a (Qza*dEax) + Sum_b (Qxb*dEbz))
09098        Tz = -2.0*(Sum_a (Qxa*dEay) + Sum_b (Qyb*dEbx))  */
09099     de[0][1] = de[1][0];
09100     de[0][2] = de[2][0];
09101     de[1][2] = de[2][1];
09102     torque[0] -= 2.0*(qyx*de[0][2] + qyy*de[1][2] + qyz*de[2][2]
09103                     - qzx*de[0][1] - qzy*de[1][1] - qzz*de[2][1]);
09104     torque[1] -= 2.0*(qzx*de[0][0] + qzy*de[1][0] + qzz*de[2][0]
09105                     - qxx*de[0][2] - qxy*de[1][2] - qxz*de[2][2]);
09106     torque[2] -= 2.0*(qxx*de[0][1] + qxy*de[1][1] + qxz*de[2][1]
09107                     - qyx*de[0][0] - qyy*de[1][0] - qyz*de[2][0]);
09108
09109     /* force on non-local induced dipole due to permanent reaction field */
09110
09111     force[0] -= dep[0][0]*uix+dep[1][0]*uiy+dep[2][0]*uiz;
09112     force[1] -= dep[1][0]*uix+dep[1][1]*uiy+dep[2][1]*uiz;
09113     force[2] -= dep[2][0]*uix+dep[2][1]*uiy+dep[2][2]*uiz;
09114
09115     force[0] = 0.5 * force[0];
09116     force[1] = 0.5 * force[1];
09117     force[2] = 0.5 * force[2];
09118     torque[0] = 0.5 * torque[0];
09119     torque[1] = 0.5 * torque[1];
09120     torque[2] = 0.5 * torque[2];
09121
09122     /* printf(" qPhi Force %f %f %f\n", force[0], force[1], force[2]);
09123        printf(" qPhi Torque %f %f %f\n", torque[0], torque[1], torque[2]); */
09124 }
09125
09126 VPUBLIC void Vpmg_ibDirectPolForce(Vpmg *thee, Vgrid *perm, Vgrid *induced,
09127                                    int atomID, double force[3]) {
09128
09129     Vatom *atom;
09130     Valist *alist;
09131     Vacc *acc;
09132     Vpbe *pbe;
09133     Vsurf_Meth srfm;
09134
09135     double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
09136     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;
09137     double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
09138     double izmagic;
09139     int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
09140
09141     VASSERT(thee != VNULL);
09142     VASSERT(perm != VNULL);        /* potential due to permanent multipoles.*/
09143     VASSERT(induced != VNULL);     /* potential due to induced dipoles. */
09144     VASSERT (!thee->pmgp->nonlin); /* Nonlinear PBE is not implemented for AMOEBA */
09145
09146     acc = thee->pbe->acc;
09147     srfm = thee->surfMeth;
09148     atom = Valist_getAtom(thee->pbe->alist, atomID);
09149     VASSERT(atom->partID != 0);   /* Currently all atoms must be in the same partition. */
09150     apos = Vatom_getPosition(atom);
09151     arad = Vatom_getRadius(atom);
09152
09153     /* Reset force */
09154     force[0] = 0.0;
09155     force[1] = 0.0;
```

```
09156     force[2] = 0.0;
09157
09158     /* Get PBE info */
09159     pbe = thee->pbe;
09160     acc = pbe->acc;
09161     alist = pbe->alist;
09162     irad = Vpbe_getMaxIonRadius(pbe);
09163     zkappa2 = Vpbe_getZkappa2(pbe);
09164     izmagic = 1.0/Vpbe_getZmagic(pbe);
09165
09166     VASSERT (zkappa2 > VPMGSMALL); /* It is ok to run AMOEBA with no ions, but this is checked for higher
      up in the driver. */
09167
09168     /* Mesh info */
09169     nx = induced->nx;
09170     ny = induced->ny;
09171     nz = induced->nz;
09172     hx = induced->hx;
09173     hy = induced->hy;
09174     hzed = induced->hzed;
09175     xmin = induced->xmin;
09176     ymin = induced->ymin;
09177     zmin = induced->zmin;
09178     xmax = induced->xmax;
09179     ymax = induced->ymax;
09180     zmax = induced->zmax;
09181     xlen = xmax-xmin;
09182     ylen = ymax-ymin;
09183     zlen = zmax-zmin;
09184
09185     /* Make sure we're on the grid */
09186     if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
09187       (apos[1]<=ymin) || (apos[1]>=ymax)  || \
09188       (apos[2]<=zmin) || (apos[2]>=zmax)) {
09189         Vnm_print(2, "Vpmg_ibForce:  Atom at (%4.3f, %4.3f, %4.3f) is off the mesh (ignoring):\n",
09190             apos[0], apos[1], apos[2]);
09191         Vnm_print(2, "Vpmg_ibForce:    xmin = %g, xmax = %g\n", xmin, xmax);
09192         Vnm_print(2, "Vpmg_ibForce:    ymin = %g, ymax = %g\n", ymin, ymax);
09193         Vnm_print(2, "Vpmg_ibForce:    zmin = %g, zmax = %g\n", zmin, zmax);
09194         fflush(stderr);
09195     } else {
09196
09197         /* Convert the atom position to grid reference frame */
09198         position[0] = apos[0] - xmin;
09199         position[1] = apos[1] - ymin;
09200         position[2] = apos[2] - zmin;
09201
09202         /* Integrate over points within this atom's (inflated) radius */
09203         rtot = (irad + arad + thee->splineWin);
09204         rtot2 = VSQR(rtot);
09205         dx = rtot + 0.5*hx;
09206         imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
09207         imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
09208         for (i=imin; i<=imax; i++) {
09209             dx2 = VSQR(position[0] - hx*i);
09210             if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
09211             else dy = 0.5*hy;
09212             jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
09213             jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
09214             for (j=jmin; j<=jmax; j++) {
09215                 dy2 = VSQR(position[1] - hy*j);
09216                 if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
09217                 else dz = 0.5*hzed;
09218                 kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
09219                 kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
09220                 for (k=kmin; k<=kmax; k++) {
09221                     dz2 = VSQR(k*hzed - position[2]);
09222                     /* See if grid point is inside ivdw radius and set ccf
09223                      * accordingly (do spline assignment here) */
09224                     if ((dz2 + dy2 + dx2) <= rtot2) {
09225                         gpos[0] = i*hx + xmin;
09226                         gpos[1] = j*hy + ymin;
09227                         gpos[2] = k*hzed + zmin;
09228                         Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, irad,
09229                           atom, tgrad);
09230                         fmag = induced->data[IJK(i,j,k)];
09231                         fmag *= perm->data[IJK(i,j,k)];
09232                         fmag *= thee->kappa[IJK(i,j,k)];
09233                         force[0] += (zkappa2*fmag*tgrad[0]);
09234                         force[1] += (zkappa2*fmag*tgrad[1]);
09235                         force[2] += (zkappa2*fmag*tgrad[2]);
```

```
09236                   }
09237                } /* k loop */
09238             } /* j loop */
09239          } /* i loop */
09240      }
09241
09242      force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
09243      force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
09244      force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
09245
09246 }
09247
09248 VPUBLIC void Vpmg_ibNLDirectPolForce(Vpmg *thee, Vgrid *perm, Vgrid *nlInduced,
09249                                      int atomID, double force[3]) {
09250      Vpmg_ibDirectPolForce(thee, perm, nlInduced, atomID, force);
09251 }
09252
09253 VPUBLIC void Vpmg_dbDirectPolForce(Vpmg *thee, Vgrid *perm, Vgrid *induced,
09254                                    int atomID, double force[3]) {
09255
09256      Vatom *atom;
09257      Vacc *acc;
09258      Vpbe *pbe;
09259      Vsurf_Meth srfm;
09260
09261      double *apos, position[3], arad, hx, hy, hzed, izmagic, deps, depsi;
09262      double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2, epsp;
09263      double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
09264      double *u, *up, Hxijk, Hyijk, Hzijk, Hxim1jk, Hyijm1k, Hzijkm1;
09265      double dHxijk[3], dHyijk[3], dHzijk[3], dHxim1jk[3], dHyijm1k[3];
09266      double dHzijkm1[3];
09267      int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
09268
09269      VASSERT(thee != VNULL);
09270      VASSERT(perm != VNULL);    /* permanent multipole PMG solution. */
09271      VASSERT(induced != VNULL); /* potential due to induced dipoles. */
09272
09273      acc = thee->pbe->acc;
09274      atom = Valist_getAtom(thee->pbe->alist, atomID);
09275      VASSERT (atom->partID != 0);   /* Currently all atoms must be in the same partition. */
09276      apos = Vatom_getPosition(atom);
09277      arad = Vatom_getRadius(atom);
09278
09279      /* Reset force */
09280      force[0] = 0.0;
09281      force[1] = 0.0;
09282      force[2] = 0.0;
09283
09284      /* Get PBE info */
09285      pbe = thee->pbe;
09286      acc = pbe->acc;
09287      srfm = thee->surfMeth;
09288      epsp = Vpbe_getSoluteDiel(pbe);
09289      epsw = Vpbe_getSolventDiel(pbe);
09290      kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*Vunit_kb;
09291      izmagic = 1.0/Vpbe_getZmagic(pbe);
09292
09293      deps = (epsw - epsp);
09294      depsi = 1.0/deps;
09295      VASSERT(VABS(deps) > VPMGSMALL);
09296
09297      /* Mesh info */
09298      nx = thee->pmgp->nx;
09299      ny = thee->pmgp->ny;
09300      nz = thee->pmgp->nz;
09301      hx = thee->pmgp->hx;
09302      hy = thee->pmgp->hy;
09303      hzed = thee->pmgp->hzed;
09304      xlen = thee->pmgp->xlen;
09305      ylen = thee->pmgp->ylen;
09306      zlen = thee->pmgp->zlen;
09307      xmin = thee->pmgp->xmin;
09308      ymin = thee->pmgp->ymin;
09309      zmin = thee->pmgp->zmin;
09310      xmax = thee->pmgp->xmax;
09311      ymax = thee->pmgp->ymax;
09312      zmax = thee->pmgp->zmax;
09313      /* If the permanent and induced potentials are flipped the
09314         results are exactly the same. */
09315      u = induced->data;
09316      up = perm->data;
```

```
09317
09318     /* Make sure we're on the grid */
09319     if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
09320       (apos[1]<=ymin) || (apos[1]>=ymax)  || \
09321       (apos[2]<=zmin) || (apos[2]>=zmax)) {
09322          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]);
09323          Vnm_print(2, "Vpmg_dbDirectPolForce:    xmin = %g, xmax = %g\n", xmin, xmax);
09324          Vnm_print(2, "Vpmg_dbDirectPolForce:    ymin = %g, ymax = %g\n", ymin, ymax);
09325          Vnm_print(2, "Vpmg_dbDirectPolForce:    zmin = %g, zmax = %g\n", zmin, zmax);
09326          fflush(stderr);
09327     } else {
09328
09329         /* Convert the atom position to grid reference frame */
09330         position[0] = apos[0] - xmin;
09331         position[1] = apos[1] - ymin;
09332         position[2] = apos[2] - zmin;
09333
09334         /* Integrate over points within this atom's (inflated) radius */
09335         rtot = (arad + thee->splineWin);
09336         rtot2 = VSQR(rtot);
09337         dx = rtot/hx;
09338         imin = (int)floor((position[0]-rtot)/hx);
09339         if (imin < 1) {
09340             Vnm_print(2, "Vpmg_dbDirectPolForce:  Atom %d off grid!\n", atomID);
09341             return;
09342         }
09343         imax = (int)ceil((position[0]+rtot)/hx);
09344         if (imax > (nx-2)) {
09345             Vnm_print(2, "Vpmg_dbDirectPolForce:  Atom %d off grid!\n", atomID);
09346             return;
09347         }
09348         jmin = (int)floor((position[1]-rtot)/hy);
09349         if (jmin < 1) {
09350             Vnm_print(2, "Vpmg_dbDirectPolForce:  Atom %d off grid!\n", atomID);
09351             return;
09352         }
09353         jmax = (int)ceil((position[1]+rtot)/hy);
09354         if (jmax > (ny-2)) {
09355             Vnm_print(2, "Vpmg_dbDirectPolForce:  Atom %d off grid!\n", atomID);
09356             return;
09357         }
09358         kmin = (int)floor((position[2]-rtot)/hzed);
09359         if (kmin < 1) {
09360             Vnm_print(2, "Vpmg_dbDirectPolForce:  Atom %d off grid!\n", atomID);
09361             return;
09362         }
09363         kmax = (int)ceil((position[2]+rtot)/hzed);
09364         if (kmax > (nz-2)) {
09365             Vnm_print(2, "Vpmg_dbDirectPolForce:  Atom %d off grid!\n", atomID);
09366             return;
09367         }
09368         for (i=imin; i<=imax; i++) {
09369             for (j=jmin; j<=jmax; j++) {
09370                 for (k=kmin; k<=kmax; k++) {
09371                     /* i,j,k */
09372                     gpos[0] = (i+0.5)*hx + xmin;
09373                     gpos[1] = j*hy + ymin;
09374                     gpos[2] = k*hzed + zmin;
09375                     Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
09376                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09377                             atom, dHxijk);
09378                     for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
09379                     gpos[0] = i*hx + xmin;
09380                     gpos[1] = (j+0.5)*hy + ymin;
09381                     gpos[2] = k*hzed + zmin;
09382                     Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
09383                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09384                             atom, dHyijk);
09385                     for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
09386                     gpos[0] = i*hx + xmin;
09387                     gpos[1] = j*hy + ymin;
09388                     gpos[2] = (k+0.5)*hzed + zmin;
09389                     Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
09390                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09391                             atom, dHzijk);
09392                     for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
09393                     /* i-1,j,k */
09394                     gpos[0] = (i-0.5)*hx + xmin;
09395                     gpos[1] = j*hy + ymin;
09396                     gpos[2] = k*hzed + zmin;
```

```
09397                     Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
09398                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09399                             atom, dHxim1jk);
09400                     for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
09401                     /* i,j-1,k */
09402                     gpos[0] = i*hx + xmin;
09403                     gpos[1] = (j-0.5)*hy + ymin;
09404                     gpos[2] = k*hzed + zmin;
09405                     Hyijm1k = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
09406                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09407                             atom, dHyijm1k);
09408                     for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;
09409                     /* i,j,k-1 */
09410                     gpos[0] = i*hx + xmin;
09411                     gpos[1] = j*hy + ymin;
09412                     gpos[2] = (k-0.5)*hzed + zmin;
09413                     Hzijkm1 = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
09414                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09415                             atom, dHzijkm1);
09416                     for (l=0; l<3; l++) dHzijkm1[l] *= Hzijkm1;
09417
09418                     dbFmag = up[IJK(i,j,k)];
09419                     tgrad[0] =
09420                        (dHxijk[0]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09421                      +  dHxim1jk[0]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09422                      + (dHyijk[0]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09423                      +  dHyijm1k[0]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09424                      + (dHzijk[0]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
09425                      + dHzijkm1[0]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09426                     tgrad[1] =
09427                        (dHxijk[1]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09428                      +  dHxim1jk[1]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09429                      + (dHyijk[1]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09430                      +  dHyijm1k[1]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09431                      + (dHzijk[1]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
09432                      + dHzijkm1[1]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09433                     tgrad[2] =
09434                        (dHxijk[2]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09435                      +  dHxim1jk[2]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09436                      + (dHyijk[2]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09437                      +  dHyijm1k[2]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09438                      + (dHzijk[2]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
09439                      + dHzijkm1[2]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09440                      force[0] += (dbFmag*tgrad[0]);
09441                      force[1] += (dbFmag*tgrad[1]);
09442                      force[2] += (dbFmag*tgrad[2]);
09443
09444                 } /* k loop */
09445             } /* j loop */
09446         } /* i loop */
09447
09448         force[0] = -force[0]*hx*hy*hzed*deps*0.5*izmagic;
09449         force[1] = -force[1]*hx*hy*hzed*deps*0.5*izmagic;
09450         force[2] = -force[2]*hx*hy*hzed*deps*0.5*izmagic;
09451
09452     }
09453 }
09454
09455 VPUBLIC void Vpmg_dbNLDirectPolForce(Vpmg *thee, Vgrid *perm, Vgrid *nlInduced,
09456                                      int atomID, double force[3]) {
09457      Vpmg_dbDirectPolForce(thee, perm, nlInduced, atomID, force);
09458 }
09459
09460 VPUBLIC void Vpmg_qfMutualPolForce(Vpmg *thee, Vgrid *induced,
09461                                Vgrid *nlinduced, int atomID, double force[3]) {
09462
09463     Vatom *atom;
09464     double *apos, *dipole, position[3], hx, hy, hzed;
09465     double *u, *unl;
09466     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax;
09467     double de[3][3], denl[3][3];
09468     double mx, my, mz, dmx, dmy, dmz, d2mx, d2my, d2mz, mi, mj, mk;
09469     double ifloat, jfloat, kfloat;
09470     double f, fnl, uix, uiy, uiz, uixnl, uiynl, uiznl;
09471     int i,j,k,nx, ny, nz, im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1;
09472     int kp1, kp2, ii, jj, kk;
09473
09474     VASSERT(thee != VNULL);   /* PMG object with PBE info. */
09475     VASSERT(induced != VNULL); /* potential due to induced dipoles. */
09476     VASSERT(nlinduced != VNULL); /* potential due to non-local induced dipoles. */
09477     atom = Valist_getAtom(thee->pbe->alist, atomID);
```

```
09478     VASSERT(atom->partID != 0);    /* all atoms must be in the same partition. */
09479     apos = Vatom_getPosition(atom);
09480     dipole = Vatom_getInducedDipole(atom);
09481     uix = dipole[0];
09482     uiy = dipole[1];
09483     uiz = dipole[2];
09484     dipole = Vatom_getNLInducedDipole(atom);
09485     uixnl = dipole[0];
09486     uiynl = dipole[1];
09487     uiznl = dipole[2];
09488     u = induced->data;
09489     unl = nlinduced->data;
09490
09491     for (i=0;i<3;i++){
09492        for (j=0;j<3;j++){
09493           de[i][j] = 0.0;
09494           denl[i][j] = 0.0;
09495        }
09496     }
09497
09498     /* Mesh info */
09499     nx = induced->nx;
09500     ny = induced->ny;
09501     nz = induced->nz;
09502     hx = induced->hx;
09503     hy = induced->hy;
09504     hzed = induced->hzed;
09505     xmin = induced->xmin;
09506     ymin = induced->ymin;
09507     zmin = induced->zmin;
09508     xmax = induced->xmax;
09509     ymax = induced->ymax;
09510     zmax = induced->zmax;
09511     xlen = xmax-xmin;
09512     ylen = ymax-ymin;
09513     zlen = zmax-zmin;
09514
09515     /* If we aren't in the current position, then we're done */
09516     if (atom->partID == 0) return;
09517
09518     /* Make sure we're on the grid */
09519     if ((apos[0]<=(xmin+2*hx))    || (apos[0]>=(xmax-2*hx)) \
09520      || (apos[1]<=(ymin+2*hy))    || (apos[1]>=(ymax-2*hy)) \
09521      || (apos[2]<=(zmin+2*hzed)) || (apos[2]>=(zmax-2*hzed))) {
09522        Vnm_print(2, "qfMutualPolForce:  Atom off the mesh (ignoring) %6.3f %6.3f %6.3f\n", apos[0],
      apos[1], apos[2]);
09523        fflush(stderr);
09524     } else {
09525
09526        /* Convert the atom position to grid coordinates */
09527        position[0] = apos[0] - xmin;
09528        position[1] = apos[1] - ymin;
09529        position[2] = apos[2] - zmin;
09530        ifloat = position[0]/hx;
09531        jfloat = position[1]/hy;
09532        kfloat = position[2]/hzed;
09533        ip1 = (int)ceil(ifloat);
09534        ip2 = ip1 + 2;
09535        im1 = (int)floor(ifloat);
09536        im2 = im1 - 2;
09537        jp1 = (int)ceil(jfloat);
09538        jp2 = jp1 + 2;
09539        jm1 = (int)floor(jfloat);
09540        jm2 = jm1 - 2;
09541        kp1 = (int)ceil(kfloat);
09542        kp2 = kp1 + 2;
09543        km1 = (int)floor(kfloat);
09544        km2 = km1 - 2;
09545
09546        /* This step shouldn't be necessary, but it saves nasty debugging
09547         * later on if something goes wrong */
09548        ip2 = VMIN2(ip2,nx-1);
09549        ip1 = VMIN2(ip1,nx-1);
09550        im1 = VMAX2(im1,0);
09551        im2 = VMAX2(im2,0);
09552        jp2 = VMIN2(jp2,ny-1);
09553        jp1 = VMIN2(jp1,ny-1);
09554        jm1 = VMAX2(jm1,0);
09555        jm2 = VMAX2(jm2,0);
09556        kp2 = VMIN2(kp2,nz-1);
09557        kp1 = VMIN2(kp1,nz-1);
```

```
09558         km1 = VMAX2(km1,0);
09559         km2 = VMAX2(km2,0);
09560
09561         for (ii=im2; ii<=ip2; ii++) {
09562             mi = VFCHI4(ii,ifloat);
09563             mx = bspline4(mi);
09564             dmx = dbspline4(mi);
09565             d2mx = d2bspline4(mi);
09566             for (jj=jm2; jj<=jp2; jj++) {
09567                 mj = VFCHI4(jj,jfloat);
09568                 my = bspline4(mj);
09569                 dmy = dbspline4(mj);
09570                 d2my = d2bspline4(mj);
09571                 for (kk=km2; kk<=kp2; kk++) {
09572                     mk = VFCHI4(kk,kfloat);
09573                     mz = bspline4(mk);
09574                     dmz = dbspline4(mk);
09575                     d2mz = d2bspline4(mk);
09576                     f = u[IJK(ii,jj,kk)];
09577                     fnl = unl[IJK(ii,jj,kk)];
09578
09579                     /* The gradient of the reaction field
09580                        due to induced dipoles */
09581                     de[0][0] += f*d2mx*my*mz/(hx*hx);
09582                     de[1][0] += f*dmx*dmy*mz/(hy*hx);
09583                     de[1][1] += f*mx*d2my*mz/(hy*hy);
09584                     de[2][0] += f*dmx*my*dmz/(hx*hzed);
09585                     de[2][1] += f*mx*dmy*dmz/(hy*hzed);
09586                     de[2][2] += f*mx*my*d2mz/(hzed*hzed);
09587
09588                     /* The gradient of the reaction field
09589                        due to non-local induced dipoles */
09590                     denl[0][0] += fnl*d2mx*my*mz/(hx*hx);
09591                     denl[1][0] += fnl*dmx*dmy*mz/(hy*hx);
09592                     denl[1][1] += fnl*mx*d2my*mz/(hy*hy);
09593                     denl[2][0] += fnl*dmx*my*dmz/(hx*hzed);
09594                     denl[2][1] += fnl*mx*dmy*dmz/(hy*hzed);
09595                     denl[2][2] += fnl*mx*my*d2mz/(hzed*hzed);
09596                 }
09597             }
09598         }
09599     }
09600
09601     /* mutual polarization force */
09602     force[0] = -(de[0][0]*uixnl + de[1][0]*uiynl + de[2][0]*uiznl);
09603     force[1] = -(de[1][0]*uixnl + de[1][1]*uiynl + de[2][1]*uiznl);
09604     force[2] = -(de[2][0]*uixnl + de[2][1]*uiynl + de[2][2]*uiznl);
09605     force[0] -=  denl[0][0]*uix + denl[1][0]*uiy + denl[2][0]*uiz;
09606     force[1] -=  denl[1][0]*uix + denl[1][1]*uiy + denl[2][1]*uiz;
09607     force[2] -=  denl[2][0]*uix + denl[2][1]*uiy + denl[2][2]*uiz;
09608
09609     force[0] = 0.5 * force[0];
09610     force[1] = 0.5 * force[1];
09611     force[2] = 0.5 * force[2];
09612
09613 }
09614
09615 VPUBLIC void Vpmg_ibMutualPolForce(Vpmg *thee, Vgrid *induced, Vgrid *nlinduced,
09616                                    int atomID, double force[3]) {
09617
09618     Vatom *atom;
09619     Valist *alist;
09620     Vacc *acc;
09621     Vpbe *pbe;
09622     Vsurf_Meth srfm;
09623
09624     double *apos, position[3], arad, irad, zkappa2, hx, hy, hzed;
09625     double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2;
09626     double rtot, dx, dx2, dy, dy2, dz, dz2, gpos[3], tgrad[3], fmag;
09627     double izmagic;
09628     int i, j, k, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
09629
09630     VASSERT(thee != VNULL);         /* We need a PMG object with PBE info. */
09631     VASSERT(induced != VNULL);      /* We need the potential due to induced dipoles. */
09632     VASSERT(nlinduced != VNULL);    /* We need the potential due to non-local induced dipoles. */
09633     VASSERT (!thee->pmgp->nonlin); /* Nonlinear PBE is not implemented for AMOEBA */
09634
09635     atom = Valist_getAtom(thee->pbe->alist, atomID);
09636     VASSERT (atom->partID != 0);    /* Currently all atoms must be in the same partition. */
09637
09638     acc = thee->pbe->acc;
```

```
09639     srfm = thee->surfMeth;
09640     apos = Vatom_getPosition(atom);
09641     arad = Vatom_getRadius(atom);
09642
09643     /* Reset force */
09644     force[0] = 0.0;
09645     force[1] = 0.0;
09646     force[2] = 0.0;
09647
09648     /* If we aren't in the current position, then we're done */
09649     if (atom->partID == 0) return;
09650
09651     /* Get PBE info */
09652     pbe = thee->pbe;
09653     acc = pbe->acc;
09654     alist = pbe->alist;
09655     irad = Vpbe_getMaxIonRadius(pbe);
09656     zkappa2 = Vpbe_getZkappa2(pbe);
09657     izmagic = 1.0/Vpbe_getZmagic(pbe);
09658
09659     VASSERT (zkappa2 > VPMGSMALL); /* Should be a check for this further up.*/
09660
09661     /* Mesh info */
09662     nx = induced->nx;
09663     ny = induced->ny;
09664     nz = induced->nz;
09665     hx = induced->hx;
09666     hy = induced->hy;
09667     hzed = induced->hzed;
09668     xmin = induced->xmin;
09669     ymin = induced->ymin;
09670     zmin = induced->zmin;
09671     xmax = induced->xmax;
09672     ymax = induced->ymax;
09673     zmax = induced->zmax;
09674     xlen = xmax-xmin;
09675     ylen = ymax-ymin;
09676     zlen = zmax-zmin;
09677
09678     /* Make sure we're on the grid */
09679     if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
09680       (apos[1]<=ymin) || (apos[1]>=ymax)  || \
09681       (apos[2]<=zmin) || (apos[2]>=zmax)) {
09682         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]);
09683         Vnm_print(2, "Vpmg_ibMutalPolForce:    xmin = %g, xmax = %g\n", xmin, xmax);
09684         Vnm_print(2, "Vpmg_ibMutalPolForce:    ymin = %g, ymax = %g\n", ymin, ymax);
09685         Vnm_print(2, "Vpmg_ibMutalPolForce:    zmin = %g, zmax = %g\n", zmin, zmax);
09686         fflush(stderr);
09687     } else {
09688
09689         /* Convert the atom position to grid reference frame */
09690         position[0] = apos[0] - xmin;
09691         position[1] = apos[1] - ymin;
09692         position[2] = apos[2] - zmin;
09693
09694         /* Integrate over points within this atom's (inflated) radius */
09695         rtot = (irad + arad + thee->splineWin);
09696         rtot2 = VSQR(rtot);
09697         dx = rtot + 0.5*hx;
09698         imin = VMAX2(0,(int)ceil((position[0] - dx)/hx));
09699         imax = VMIN2(nx-1,(int)floor((position[0] + dx)/hx));
09700         for (i=imin; i<=imax; i++) {
09701             dx2 = VSQR(position[0] - hx*i);
09702             if (rtot2 > dx2) dy = VSQRT(rtot2 - dx2) + 0.5*hy;
09703             else dy = 0.5*hy;
09704             jmin = VMAX2(0,(int)ceil((position[1] - dy)/hy));
09705             jmax = VMIN2(ny-1,(int)floor((position[1] + dy)/hy));
09706             for (j=jmin; j<=jmax; j++) {
09707                 dy2 = VSQR(position[1] - hy*j);
09708                 if (rtot2 > (dx2+dy2)) dz = VSQRT(rtot2-dx2-dy2)+0.5*hzed;
09709                 else dz = 0.5*hzed;
09710                 kmin = VMAX2(0,(int)ceil((position[2] - dz)/hzed));
09711                 kmax = VMIN2(nz-1,(int)floor((position[2] + dz)/hzed));
09712                 for (k=kmin; k<=kmax; k++) {
09713                     dz2 = VSQR(k*hzed - position[2]);
09714                     /* See if grid point is inside ivdw radius and set ccf
09715                      * accordingly (do spline assignment here) */
09716                     if ((dz2 + dy2 + dx2) <= rtot2) {
09717                         gpos[0] = i*hx + xmin;
09718                         gpos[1] = j*hy + ymin;
```

```
09719                         gpos[2] = k*hzed + zmin;
09720                         Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, irad,
09721                           atom, tgrad);
09722                         fmag = induced->data[IJK(i,j,k)];
09723                         fmag *= nlinduced->data[IJK(i,j,k)];
09724                         fmag *= thee->kappa[IJK(i,j,k)];
09725                         force[0] += (zkappa2*fmag*tgrad[0]);
09726                         force[1] += (zkappa2*fmag*tgrad[1]);
09727                         force[2] += (zkappa2*fmag*tgrad[2]);
09728                       }
09729                   } /* k loop */
09730               } /* j loop */
09731           } /* i loop */
09732       }
09733
09734       force[0] = force[0] * 0.5 * hx * hy * hzed * izmagic;
09735       force[1] = force[1] * 0.5 * hx * hy * hzed * izmagic;
09736       force[2] = force[2] * 0.5 * hx * hy * hzed * izmagic;
09737   }
09738
09739   VPUBLIC void Vpmg_dbMutualPolForce(Vpmg *thee, Vgrid *induced,
09740                                      Vgrid *nlinduced, int atomID,
09741                                      double force[3]) {
09742
09743       Vatom *atom;
09744       Vacc *acc;
09745       Vpbe *pbe;
09746       Vsurf_Meth srfm;
09747
09748       double *apos, position[3], arad, hx, hy, hzed, izmagic, deps, depsi;
09749       double xlen, ylen, zlen, xmin, ymin, zmin, xmax, ymax, zmax, rtot2, epsp;
09750       double rtot, dx, gpos[3], tgrad[3], dbFmag, epsw, kT;
09751       double *u, *unl, Hxijk, Hyijk, Hzijk, Hxim1jk, Hyijm1k, Hzijkm1;
09752       double dHxijk[3], dHyijk[3], dHzijk[3], dHxim1jk[3], dHyijm1k[3];
09753       double dHzijkm1[3];
09754       int i, j, k, l, nx, ny, nz, imin, imax, jmin, jmax, kmin, kmax;
09755
09756       VASSERT(thee != VNULL); /* PMG object with PBE info. */
09757       VASSERT(induced != VNULL); /* potential due to induced dipoles.*/
09758       VASSERT(nlinduced != VNULL); /* potential due to non-local induced dipoles.*/
09759
09760       acc = thee->pbe->acc;
09761       srfm = thee->surfMeth;
09762       atom = Valist_getAtom(thee->pbe->alist, atomID);
09763       VASSERT (atom->partID != 0); /* all atoms must be in the same partition.*/
09764       apos = Vatom_getPosition(atom);
09765       arad = Vatom_getRadius(atom);
09766
09767       /* Reset force */
09768       force[0] = 0.0;
09769       force[1] = 0.0;
09770       force[2] = 0.0;
09771
09772       /* Get PBE info */
09773       pbe = thee->pbe;
09774       acc = pbe->acc;
09775       epsp = Vpbe_getSoluteDiel(pbe);
09776       epsw = Vpbe_getSolventDiel(pbe);
09777       kT = Vpbe_getTemperature(pbe)*(1e-3)*Vunit_Na*Vunit_kb;
09778       izmagic = 1.0/Vpbe_getZmagic(pbe);
09779
09780       deps = (epsw - epsp);
09781       depsi = 1.0/deps;
09782       VASSERT(VABS(deps) > VPMGSMALL);
09783
09784       /* Mesh info */
09785       nx = thee->pmgp->nx;
09786       ny = thee->pmgp->ny;
09787       nz = thee->pmgp->nz;
09788       hx = thee->pmgp->hx;
09789       hy = thee->pmgp->hy;
09790       hzed = thee->pmgp->hzed;
09791       xlen = thee->pmgp->xlen;
09792       ylen = thee->pmgp->ylen;
09793       zlen = thee->pmgp->zlen;
09794       xmin = thee->pmgp->xmin;
09795       ymin = thee->pmgp->ymin;
09796       zmin = thee->pmgp->zmin;
09797       xmax = thee->pmgp->xmax;
09798       ymax = thee->pmgp->ymax;
09799       zmax = thee->pmgp->zmax;
```

```
09800     u = induced->data;
09801     unl = nlinduced->data;
09802
09803     /* Make sure we're on the grid */
09804     if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
09805       (apos[1]<=ymin) || (apos[1]>=ymax)  || \
09806       (apos[2]<=zmin) || (apos[2]>=zmax)) {
09807         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]);
09808         Vnm_print(2, "Vpmg_dbMutualPolForce:    xmin = %g, xmax = %g\n", xmin, xmax);
09809         Vnm_print(2, "Vpmg_dbMutualPolForce:    ymin = %g, ymax = %g\n", ymin, ymax);
09810         Vnm_print(2, "Vpmg_dbMutualPolForce:    zmin = %g, zmax = %g\n", zmin, zmax);
09811         fflush(stderr);
09812     } else {
09813
09814         /* Convert the atom position to grid reference frame */
09815         position[0] = apos[0] - xmin;
09816         position[1] = apos[1] - ymin;
09817         position[2] = apos[2] - zmin;
09818
09819         /* Integrate over points within this atom's (inflated) radius */
09820         rtot = (arad + thee->splineWin);
09821         rtot2 = VSQR(rtot);
09822         dx = rtot/hx;
09823         imin = (int)floor((position[0]-rtot)/hx);
09824         if (imin < 1) {
09825             Vnm_print(2, "Vpmg_dbMutualPolForce:  Atom %d off grid!\n", atomID);
09826             return;
09827         }
09828         imax = (int)ceil((position[0]+rtot)/hx);
09829         if (imax > (nx-2)) {
09830             Vnm_print(2, "Vpmg_dbMutualPolForce:  Atom %d off grid!\n", atomID);
09831             return;
09832         }
09833         jmin = (int)floor((position[1]-rtot)/hy);
09834         if (jmin < 1) {
09835             Vnm_print(2, "Vpmg_dbMutualPolForce:  Atom %d off grid!\n", atomID);
09836             return;
09837         }
09838         jmax = (int)ceil((position[1]+rtot)/hy);
09839         if (jmax > (ny-2)) {
09840             Vnm_print(2, "Vpmg_dbMutualPolForce:  Atom %d off grid!\n", atomID);
09841             return;
09842         }
09843         kmin = (int)floor((position[2]-rtot)/hzed);
09844         if (kmin < 1) {
09845             Vnm_print(2, "Vpmg_dbMutualPolForce:  Atom %d off grid!\n", atomID);
09846             return;
09847         }
09848         kmax = (int)ceil((position[2]+rtot)/hzed);
09849         if (kmax > (nz-2)) {
09850             Vnm_print(2, "Vpmg_dbMutualPolForce:  Atom %d off grid!\n", atomID);
09851             return;
09852         }
09853         for (i=imin; i<=imax; i++) {
09854             for (j=jmin; j<=jmax; j++) {
09855                 for (k=kmin; k<=kmax; k++) {
09856                     /* i,j,k */
09857                     gpos[0] = (i+0.5)*hx + xmin;
09858                     gpos[1] = j*hy + ymin;
09859                     gpos[2] = k*hzed + zmin;
09860                     Hxijk = (thee->epsx[IJK(i,j,k)] - epsp)*depsi;
09861                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09862                             atom, dHxijk);
09863                     for (l=0; l<3; l++) dHxijk[l] *= Hxijk;
09864                     gpos[0] = i*hx + xmin;
09865                     gpos[1] = (j+0.5)*hy + ymin;
09866                     gpos[2] = k*hzed + zmin;
09867                     Hyijk = (thee->epsy[IJK(i,j,k)] - epsp)*depsi;
09868                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09869                             atom, dHyijk);
09870                     for (l=0; l<3; l++) dHyijk[l] *= Hyijk;
09871                     gpos[0] = i*hx + xmin;
09872                     gpos[1] = j*hy + ymin;
09873                     gpos[2] = (k+0.5)*hzed + zmin;
09874                     Hzijk = (thee->epsz[IJK(i,j,k)] - epsp)*depsi;
09875                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09876                             atom, dHzijk);
09877                     for (l=0; l<3; l++) dHzijk[l] *= Hzijk;
09878                     /* i-1,j,k */
09879                     gpos[0] = (i-0.5)*hx + xmin;
```

```
09880                     gpos[1] = j*hy + ymin;
09881                     gpos[2] = k*hzed + zmin;
09882                     Hxim1jk = (thee->epsx[IJK(i-1,j,k)] - epsp)*depsi;
09883                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09884                             atom, dHxim1jk);
09885                     for (l=0; l<3; l++) dHxim1jk[l] *= Hxim1jk;
09886                     /* i,j-1,k */
09887                     gpos[0] = i*hx + xmin;
09888                     gpos[1] = (j-0.5)*hy + ymin;
09889                     gpos[2] = k*hzed + zmin;
09890                     Hyijm1k = (thee->epsy[IJK(i,j-1,k)] - epsp)*depsi;
09891                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09892                             atom, dHyijm1k);
09893                     for (l=0; l<3; l++) dHyijm1k[l] *= Hyijm1k;
09894                     /* i,j,k-1 */
09895                     gpos[0] = i*hx + xmin;
09896                     gpos[1] = j*hy + ymin;
09897                     gpos[2] = (k-0.5)*hzed + zmin;
09898                     Hzijkm1 = (thee->epsz[IJK(i,j,k-1)] - epsp)*depsi;
09899                     Vpmg_splineSelect(srfm, acc, gpos, thee->splineWin, 0.,
09900                             atom, dHzijkm1);
09901                     for (l=0; l<3; l++) dHzijkm1[l] *= Hzijkm1;
09902                     dbFmag = unl[IJK(i,j,k)];
09903                     tgrad[0] =
09904                        (dHxijk[0]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09905                      +  dHxim1jk[0]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09906                      + (dHyijk[0]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09907                      +  dHyijm1k[0]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09908                      + (dHzijk[0]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
09909                      + dHzijkm1[0]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09910                     tgrad[1] =
09911                        (dHxijk[1]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09912                      +  dHxim1jk[1]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09913                      + (dHyijk[1]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09914                      +  dHyijm1k[1]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09915                      + (dHzijk[1]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
09916                      + dHzijkm1[1]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09917                     tgrad[2] =
09918                        (dHxijk[2]  *(u[IJK(i+1,j,k)]-u[IJK(i,j,k)])
09919                      +  dHxim1jk[2]*(u[IJK(i-1,j,k)]-u[IJK(i,j,k)]))/VSQR(hx)
09920                      + (dHyijk[2]  *(u[IJK(i,j+1,k)]-u[IJK(i,j,k)])
09921                      +  dHyijm1k[2]*(u[IJK(i,j-1,k)]-u[IJK(i,j,k)]))/VSQR(hy)
09922                      + (dHzijk[2]  *(u[IJK(i,j,k+1)]-u[IJK(i,j,k)])
09923                      + dHzijkm1[2]*(u[IJK(i,j,k-1)]-u[IJK(i,j,k)]))/VSQR(hzed);
09924                     force[0] += (dbFmag*tgrad[0]);
09925                     force[1] += (dbFmag*tgrad[1]);
09926                     force[2] += (dbFmag*tgrad[2]);
09927                 } /* k loop */
09928             } /* j loop */
09929         } /* i loop */
09930
09931         force[0] = -force[0]*hx*hy*hzed*deps*0.5*izmagic;
09932         force[1] = -force[1]*hx*hy*hzed*deps*0.5*izmagic;
09933         force[2] = -force[2]*hx*hy*hzed*deps*0.5*izmagic;
09934     }
09935 }
09936
09937 #endif /* if defined(WITH_TINKER) */
09938
09939 VPRIVATE void fillcoCoefSpline4(Vpmg *thee) {
09940
09941     Valist *alist;
09942     Vpbe *pbe;
09943     Vatom *atom;
09944     double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr, dist2;
09945     double xlen, ylen, zlen, position[3], itot, stot, ictot, ictot2, sctot;
09946     double irad, dx, dy, dz, epsw, epsp, w2i;
09947     double hx, hy, hzed, *apos, arad, sctot2;
09948     double dx2, dy2, dz2, stot2, itot2, rtot, rtot2, splineWin;
09949     double dist, value, denom, sm, sm2, sm3, sm4, sm5, sm6, sm7;
09950     double e, e2, e3, e4, e5, e6, e7;
09951     double b, b2, b3, b4, b5, b6, b7;
09952     double c0, c1, c2, c3, c4, c5, c6, c7;
09953     double ic0, ic1, ic2, ic3, ic4, ic5, ic6, ic7;
09954     int i, j, k, nx, ny, nz, iatom;
09955     int imin, imax, jmin, jmax, kmin, kmax;
09956
09957     VASSERT(thee != VNULL);
09958     splineWin = thee->splineWin;
09959
09960     /* Get PBE info */
```

```
09961     pbe = thee->pbe;
09962     alist = pbe->alist;
09963     irad = Vpbe_getMaxIonRadius(pbe);
09964     ionstr = Vpbe_getBulkIonicStrength(pbe);
09965     epsw = Vpbe_getSolventDiel(pbe);
09966     epsp = Vpbe_getSoluteDiel(pbe);
09967
09968     /* Mesh info */
09969     nx = thee->pmgp->nx;
09970     ny = thee->pmgp->ny;
09971     nz = thee->pmgp->nz;
09972     hx = thee->pmgp->hx;
09973     hy = thee->pmgp->hy;
09974     hzed = thee->pmgp->hzed;
09975
09976     /* Define the total domain size */
09977     xlen = thee->pmgp->xlen;
09978     ylen = thee->pmgp->ylen;
09979     zlen = thee->pmgp->zlen;
09980
09981     /* Define the min/max dimensions */
09982     xmin = thee->pmgp->xcent - (xlen/2.0);
09983     ymin = thee->pmgp->ycent - (ylen/2.0);
09984     zmin = thee->pmgp->zcent - (zlen/2.0);
09985     xmax = thee->pmgp->xcent + (xlen/2.0);
09986     ymax = thee->pmgp->ycent + (ylen/2.0);
09987     zmax = thee->pmgp->zcent + (zlen/2.0);
09988
09989     /* This is a floating point parameter related to the non-zero nature of the
09990      * bulk ionic strength.  If the ionic strength is greater than zero; this
09991      * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
09992      * Otherwise, this parameter is set to 0.0 */
09993     if (ionstr > VPMGSMALL) ionmask = 1.0;
09994     else ionmask = 0.0;
09995
09996     /* Reset the kappa, epsx, epsy, and epsz arrays */
09997     for (i=0; i<(nx*ny*nz); i++) {
09998         thee->kappa[i] = 1.0;
09999         thee->epsx[i] = 1.0;
10000         thee->epsy[i] = 1.0;
10001         thee->epsz[i] = 1.0;
10002     }
10003
10004     /* Loop through the atoms and do assign the dielectric */
10005     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
10006
10007         atom = Valist_getAtom(alist, iatom);
10008         apos = Vatom_getPosition(atom);
10009         arad = Vatom_getRadius(atom);
10010
10011         b = arad - splineWin;
10012         e = arad + splineWin;
10013         e2 = e * e;
10014         e3 = e2 * e;
10015         e4 = e3 * e;
10016         e5 = e4 * e;
10017         e6 = e5 * e;
10018         e7 = e6 * e;
10019         b2 = b * b;
10020         b3 = b2 * b;
10021         b4 = b3 * b;
10022         b5 = b4 * b;
10023         b6 = b5 * b;
10024         b7 = b6 * b;
10025         denom = e7  - 7.0*b*e6 + 21.0*b2*e5 - 35.0*e4*b3
10026               + 35.0*e3*b4 - 21.0*b5*e2  + 7.0*e*b6 - b7;
10027         c0 = b4*(35.0*e3 - 21.0*b*e2 + 7*e*b2 - b3)/denom;
10028         c1 = -140.0*b3*e3/denom;
10029         c2 = 210.0*e2*b2*(e + b)/denom;
10030         c3 = -140.0*e*b*(e2 + 3.0*b*e + b2)/denom;
10031         c4 =  35.0*(e3 + 9.0*b*e2 + + 9.0*e*b2 + b3)/denom;
10032         c5 = -84.0*(e2 + 3.0*b*e + b2)/denom;
10033         c6 =  70.0*(e + b)/denom;
10034         c7 = -20.0/denom;
10035
10036         b = irad + arad - splineWin;
10037         e = irad + arad + splineWin;
10038         e2 = e * e;
10039         e3 = e2 * e;
10040         e4 = e3 * e;
10041         e5 = e4 * e;
```

```
10042         e6 = e5 * e;
10043         e7 = e6 * e;
10044         b2 = b * b;
10045         b3 = b2 * b;
10046         b4 = b3 * b;
10047         b5 = b4 * b;
10048         b6 = b5 * b;
10049         b7 = b6 * b;
10050         denom = e7  - 7.0*b*e6 + 21.0*b2*e5 - 35.0*e4*b3
10051               + 35.0*e3*b4 - 21.0*b5*e2  + 7.0*e*b6 - b7;
10052         ic0 = b4*(35.0*e3 - 21.0*b*e2 + 7*e*b2 - b3)/denom;
10053         ic1 = -140.0*b3*e3/denom;
10054         ic2 = 210.0*e2*b2*(e + b)/denom;
10055         ic3 = -140.0*e*b*(e2 + 3.0*b*e + b2)/denom;
10056         ic4 =  35.0*(e3 + 9.0*b*e2 + + 9.0*e*b2 + b3)/denom;
10057         ic5 = -84.0*(e2 + 3.0*b*e + b2)/denom;
10058         ic6 =  70.0*(e + b)/denom;
10059         ic7 = -20.0/denom;
10060 
10061         /* Make sure we're on the grid */
10062         if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
10063             (apos[1]<=ymin) || (apos[1]>=ymax)  || \
10064             (apos[2]<=zmin) || (apos[2]>=zmax)) {
10065             if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
10066                 (thee->pmgp->bcfl != BCFL_MAP)) {
10067                 Vnm_print(2, "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f,\
10068   %4.3f) is off the mesh (ignoring):\n",
10069                   iatom, apos[0], apos[1], apos[2]);
10070                 Vnm_print(2, "Vpmg_fillco:    xmin = %g, xmax = %g\n",
10071                   xmin, xmax);
10072                 Vnm_print(2, "Vpmg_fillco:    ymin = %g, ymax = %g\n",
10073                   ymin, ymax);
10074                 Vnm_print(2, "Vpmg_fillco:    zmin = %g, zmax = %g\n",
10075                   zmin, zmax);
10076             }
10077             fflush(stderr);
10078 
10079         } else if (arad > VPMGSMALL ) { /* if we're on the mesh */
10080 
10081             /* Convert the atom position to grid reference frame */
10082             position[0] = apos[0] - xmin;
10083             position[1] = apos[1] - ymin;
10084             position[2] = apos[2] - zmin;
10085 
10086             /* MARK ION ACCESSIBILITY AND DIELECTRIC VALUES FOR LATER
10087              * ASSIGNMENT (Steps #1-3) */
10088             itot = irad + arad + splineWin;
10089             itot2 = VSQR(itot);
10090             ictot = VMAX2(0, (irad + arad - splineWin));
10091             ictot2 = VSQR(ictot);
10092             stot = arad + splineWin;
10093             stot2 = VSQR(stot);
10094             sctot = VMAX2(0, (arad - splineWin));
10095             sctot2 = VSQR(sctot);
10096 
10097            /* We'll search over grid points which are in the greater of
10098             * these two radii */
10099             rtot = VMAX2(itot, stot);
10100             rtot2 = VMAX2(itot2, stot2);
10101             dx = rtot + 0.5*hx;
10102             dy = rtot + 0.5*hy;
10103             dz = rtot + 0.5*hzed;
10104             imin = VMAX2(0,(int)floor((position[0] - dx)/hx));
10105             imax = VMIN2(nx-1,(int)ceil((position[0] + dx)/hx));
10106             jmin = VMAX2(0,(int)floor((position[1] - dy)/hy));
10107             jmax = VMIN2(ny-1,(int)ceil((position[1] + dy)/hy));
10108             kmin = VMAX2(0,(int)floor((position[2] - dz)/hzed));
10109             kmax = VMIN2(nz-1,(int)ceil((position[2] + dz)/hzed));
10110             for (i=imin; i<=imax; i++) {
10111                 dx2 = VSQR(position[0] - hx*i);
10112                 for (j=jmin; j<=jmax; j++) {
10113                     dy2 = VSQR(position[1] - hy*j);
10114                     for (k=kmin; k<=kmax; k++) {
10115                         dz2 = VSQR(position[2] - k*hzed);
10116 
10117                         /* ASSIGN CCF */
10118                         if (thee->kappa[IJK(i,j,k)] > VPMGSMALL) {
10119                             dist2 = dz2 + dy2 + dx2;
10120                             if (dist2 >= itot2) {
10121                                 ;
10122                             }
```

```
10123                           if (dist2 <= ictot2) {
10124                               thee->kappa[IJK(i,j,k)] = 0.0;
10125                           }
10126                           if ((dist2 < itot2) && (dist2 > ictot2)) {
10127                               dist = VSQRT(dist2);
10128                               sm = dist;
10129                               sm2 = dist2;
10130                               sm3 = sm2 * sm;
10131                               sm4 = sm3 * sm;
10132                               sm5 = sm4 * sm;
10133                               sm6 = sm5 * sm;
10134                               sm7 = sm6 * sm;
10135                               value = ic0 + ic1*sm + ic2*sm2 + ic3*sm3
10136                                     + ic4*sm4 + ic5*sm5 + ic6*sm6 + ic7*sm7;
10137                               if (value > 1.0) {
10138                                  value = 1.0;
10139                               } else if (value < 0.0){
10140                                  value = 0.0;
10141                               }
10142                               thee->kappa[IJK(i,j,k)] *= value;
10143                           }
10144                       }
10145
10146                       /* ASSIGN A1CF */
10147                       if (thee->epsx[IJK(i,j,k)] > VPMGSMALL) {
10148                           dist2 = dz2+dy2+VSQR(position[0]-(i+0.5)*hx);
10149                           if (dist2 >= stot2) {
10150                               thee->epsx[IJK(i,j,k)] *= 1.0;
10151                           }
10152                           if (dist2 <= sctot2) {
10153                               thee->epsx[IJK(i,j,k)] = 0.0;
10154                           }
10155                           if ((dist2 > sctot2) && (dist2 < stot2)) {
10156                               dist = VSQRT(dist2);
10157                               sm = dist;
10158                               sm2 = VSQR(sm);
10159                               sm3 = sm2 * sm;
10160                               sm4 = sm3 * sm;
10161                               sm5 = sm4 * sm;
10162                               sm6 = sm5 * sm;
10163                               sm7 = sm6 * sm;
10164                               value = c0 + c1*sm + c2*sm2 + c3*sm3
10165                                     + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
10166                               if (value > 1.0) {
10167                                  value = 1.0;
10168                               } else if (value < 0.0){
10169                                  value = 0.0;
10170                               }
10171                               thee->epsx[IJK(i,j,k)] *= value;
10172                           }
10173                       }
10174
10175                       /* ASSIGN A2CF */
10176                       if (thee->epsy[IJK(i,j,k)] > VPMGSMALL) {
10177                           dist2 = dz2+dx2+VSQR(position[1]-(j+0.5)*hy);
10178                           if (dist2 >= stot2) {
10179                               thee->epsy[IJK(i,j,k)] *= 1.0;
10180                           }
10181                           if (dist2 <= sctot2) {
10182                               thee->epsy[IJK(i,j,k)] = 0.0;
10183                           }
10184                           if ((dist2 > sctot2) && (dist2 < stot2)) {
10185                               dist = VSQRT(dist2);
10186                               sm = dist;
10187                               sm2 = VSQR(sm);
10188                               sm3 = sm2 * sm;
10189                               sm4 = sm3 * sm;
10190                               sm5 = sm4 * sm;
10191                               sm6 = sm5 * sm;
10192                               sm7 = sm6 * sm;
10193                               value = c0 + c1*sm + c2*sm2 + c3*sm3
10194                                     + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
10195                               if (value > 1.0) {
10196                                  value = 1.0;
10197                               } else if (value < 0.0){
10198                                  value = 0.0;
10199                               }
10200                               thee->epsy[IJK(i,j,k)] *= value;
10201                           }
10202                       }
10203
```

```
10204                         /* ASSIGN A3CF */
10205                         if (thee->epsz[IJK(i,j,k)] > VPMGSMALL) {
10206                             dist2 = dy2+dx2+VSQR(position[2]-(k+0.5)*hzed);
10207                             if (dist2 >= stot2) {
10208                                 thee->epsz[IJK(i,j,k)] *= 1.0;
10209                             }
10210                             if (dist2 <= sctot2) {
10211                                 thee->epsz[IJK(i,j,k)] = 0.0;
10212                             }
10213                             if ((dist2 > sctot2) && (dist2 < stot2)) {
10214                                 dist = VSQRT(dist2);
10215                                 sm = dist;
10216                                 sm2 = dist2;
10217                                 sm3 = sm2 * sm;
10218                                 sm4 = sm3 * sm;
10219                                 sm5 = sm4 * sm;
10220                                 sm6 = sm5 * sm;
10221                                 sm7 = sm6 * sm;
10222                                 value = c0 + c1*sm + c2*sm2 + c3*sm3
10223                                       + c4*sm4 + c5*sm5 + c6*sm6 + c7*sm7;
10224                                 if (value > 1.0) {
10225                                    value = 1.0;
10226                                 } else if (value < 0.0){
10227                                    value = 0.0;
10228                                 }
10229                                 thee->epsz[IJK(i,j,k)] *= value;
10230                             }
10231                         }
10232 
10233 
10234                     } /* k loop */
10235                 } /* j loop */
10236             } /* i loop */
10237         } /* endif (on the mesh) */
10238     } /* endfor (over all atoms) */
10239 
10240     Vnm_print(0, "Vpmg_fillco:  filling coefficient arrays\n");
10241     /* Interpret markings and fill the coefficient arrays */
10242     for (k=0; k<nz; k++) {
10243         for (j=0; j<ny; j++) {
10244             for (i=0; i<nx; i++) {
10245 
10246                 thee->kappa[IJK(i,j,k)] = ionmask*thee->kappa[IJK(i,j,k)];
10247                 thee->epsx[IJK(i,j,k)] = (epsw-epsp)*thee->epsx[IJK(i,j,k)]
10248                   + epsp;
10249                 thee->epsy[IJK(i,j,k)] = (epsw-epsp)*thee->epsy[IJK(i,j,k)]
10250                   + epsp;
10251                 thee->epsz[IJK(i,j,k)] = (epsw-epsp)*thee->epsz[IJK(i,j,k)]
10252                   + epsp;
10253 
10254             } /* i loop */
10255         } /* j loop */
10256     } /* k loop */
10257 
10258 }
10259 
10260 VPUBLIC void fillcoPermanentInduced(Vpmg *thee) {
10261 
10262     Valist *alist;
10263     Vpbe *pbe;
10264     Vatom *atom;
10265     /* Coversions */
10266     double zmagic, f;
10267     /* Grid */
10268     double xmin, xmax, ymin, ymax, zmin, zmax;
10269     double xlen, ylen, zlen, position[3], ifloat, jfloat, kfloat;
10270     double hx, hy, hzed, *apos;
10271     /* Multipole */
10272     double charge, *dipole,*quad;
10273     double c,ux,uy,uz,qxx,qyx,qyy,qzx,qzy,qzz,qave;
10274     /* B-spline weights */
10275     double mx,my,mz,dmx,dmy,dmz,d2mx,d2my,d2mz;
10276     double mi,mj,mk;
10277     /* Loop variables */
10278     int i, ii, jj, kk, nx, ny, nz, iatom;
10279     int im2, im1, ip1, ip2, jm2, jm1, jp1, jp2, km2, km1, kp1, kp2;
10280 
10281     VASSERT(thee != VNULL);
10282 
10283     /* Get PBE info */
10284     pbe = thee->pbe;
```

```
10285     alist = pbe->alist;
10286     zmagic = Vpbe_getZmagic(pbe);
10287
10288     /* Mesh info */
10289     nx = thee->pmgp->nx;
10290     ny = thee->pmgp->ny;
10291     nz = thee->pmgp->nz;
10292     hx = thee->pmgp->hx;
10293     hy = thee->pmgp->hy;
10294     hzed = thee->pmgp->hzed;
10295
10296     /* Conversion */
10297     f = zmagic/(hx*hy*hzed);
10298
10299     /* Define the total domain size */
10300     xlen = thee->pmgp->xlen;
10301     ylen = thee->pmgp->ylen;
10302     zlen = thee->pmgp->zlen;
10303
10304     /* Define the min/max dimensions */
10305     xmin = thee->pmgp->xcent - (xlen/2.0);
10306     ymin = thee->pmgp->ycent - (ylen/2.0);
10307     zmin = thee->pmgp->zcent - (zlen/2.0);
10308     xmax = thee->pmgp->xcent + (xlen/2.0);
10309     ymax = thee->pmgp->ycent + (ylen/2.0);
10310     zmax = thee->pmgp->zcent + (zlen/2.0);
10311
10312     /* Fill in the source term (permanent atomic multipoles
10313        and induced dipoles) */
10314     Vnm_print(0, "fillcoPermanentInduced:  filling in source term.\n");
10315     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
10316
10317         atom = Valist_getAtom(alist, iatom);
10318         apos = Vatom_getPosition(atom);
10319
10320         c = Vatom_getCharge(atom)*f;
10321
10322 #if defined(WITH_TINKER)
10323         dipole = Vatom_getDipole(atom);
10324         ux = dipole[0]/hx*f;
10325         uy = dipole[1]/hy*f;
10326         uz = dipole[2]/hzed*f;
10327         dipole = Vatom_getInducedDipole(atom);
10328         ux = ux + dipole[0]/hx*f;
10329         uy = uy + dipole[1]/hy*f;
10330         uz = uz + dipole[2]/hzed*f;
10331         quad = Vatom_getQuadrupole(atom);
10332         qxx = (1.0/3.0)*quad[0]/(hx*hx)*f;
10333         qyx = (2.0/3.0)*quad[3]/(hx*hy)*f;
10334         qyy = (1.0/3.0)*quad[4]/(hy*hy)*f;
10335         qzx = (2.0/3.0)*quad[6]/(hzed*hx)*f;
10336         qzy = (2.0/3.0)*quad[7]/(hzed*hy)*f;
10337         qzz = (1.0/3.0)*quad[8]/(hzed*hzed)*f;
10338 #else
10339         ux = 0.0;
10340         uy = 0.0;
10341         uz = 0.0;
10342         qxx = 0.0;
10343         qyx = 0.0;
10344         qyy = 0.0;
10345         qzx = 0.0;
10346         qzy = 0.0;
10347         qzz = 0.0;
10348 #endif /* if defined(WITH_TINKER) */
10349
10350         /* Make sure we're on the grid */
10351         if ((apos[0]<=(xmin-2*hx)) || (apos[0]>=(xmax+2*hx))  || \
10352             (apos[1]<=(ymin-2*hy)) || (apos[1]>=(ymax+2*hy))  || \
10353             (apos[2]<=(zmin-2*hzed)) || (apos[2]>=(zmax+2*hzed))) {
10354             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]);
10355             Vnm_print(2, "fillcoPermanentMultipole: xmin = %g, xmax = %g\n", xmin, xmax);
10356             Vnm_print(2, "fillcoPermanentMultipole: ymin = %g, ymax = %g\n", ymin, ymax);
10357             Vnm_print(2, "fillcoPermanentMultipole: zmin = %g, zmax = %g\n", zmin, zmax);
10358             fflush(stderr);
10359         } else {
10360
10361             /* Convert the atom position to grid reference frame */
10362             position[0] = apos[0] - xmin;
10363             position[1] = apos[1] - ymin;
10364             position[2] = apos[2] - zmin;
```

```
10365
10366             /* Figure out which vertices we're next to */
10367             ifloat = position[0]/hx;
10368             jfloat = position[1]/hy;
10369             kfloat = position[2]/hzed;
10370
10371             ip1   = (int)ceil(ifloat);
10372             ip2   = ip1 + 2;
10373             im1   = (int)floor(ifloat);
10374             im2   = im1 - 2;
10375             jp1   = (int)ceil(jfloat);
10376             jp2   = jp1 + 2;
10377             jm1   = (int)floor(jfloat);
10378             jm2   = jm1 - 2;
10379             kp1   = (int)ceil(kfloat);
10380             kp2   = kp1 + 2;
10381             km1   = (int)floor(kfloat);
10382             km2   = km1 - 2;
10383
10384             /* This step shouldn't be necessary, but it saves nasty debugging
10385              * later on if something goes wrong */
10386             ip2 = VMIN2(ip2,nx-1);
10387             ip1 = VMIN2(ip1,nx-1);
10388             im1 = VMAX2(im1,0);
10389             im2 = VMAX2(im2,0);
10390             jp2 = VMIN2(jp2,ny-1);
10391             jp1 = VMIN2(jp1,ny-1);
10392             jm1 = VMAX2(jm1,0);
10393             jm2 = VMAX2(jm2,0);
10394             kp2 = VMIN2(kp2,nz-1);
10395             kp1 = VMIN2(kp1,nz-1);
10396             km1 = VMAX2(km1,0);
10397             km2 = VMAX2(km2,0);
10398
10399             /* Now assign fractions of the charge to the nearby verts */
10400             for (ii=im2; ii<=ip2; ii++) {
10401                 mi = VFCHI4(ii,ifloat);
10402                 mx = bspline4(mi);
10403                 dmx = dbspline4(mi);
10404                 d2mx = d2bspline4(mi);
10405                 for (jj=jm2; jj<=jp2; jj++) {
10406                     mj = VFCHI4(jj,jfloat);
10407                     my = bspline4(mj);
10408                     dmy = dbspline4(mj);
10409                     d2my = d2bspline4(mj);
10410                     for (kk=km2; kk<=kp2; kk++) {
10411                         mk = VFCHI4(kk,kfloat);
10412                         mz = bspline4(mk);
10413                         dmz = dbspline4(mk);
10414                         d2mz = d2bspline4(mk);
10415                         charge = mx*my*mz*c -
10416                          dmx*my*mz*ux - mx*dmy*mz*uy - mx*my*dmz*uz +
10417                          d2mx*my*mz*qxx +
10418                          dmx*dmy*mz*qyx + mx*d2my*mz*qyy +
10419                          dmx*my*dmz*qzx + mx*dmy*dmz*qzy + mx*my*d2mz*qzz;
10420                         thee->charge[IJK(ii,jj,kk)] += charge;
10421
10422                     }
10423                 }
10424             }
10425         } /* endif (on the mesh) */
10426
10427     } /* endfor (each atom) */
10428 }
10429
10430 VPRIVATE void fillcoCoefSpline3(Vpmg *thee) {
10431
10432     Valist *alist;
10433     Vpbe *pbe;
10434     Vatom *atom;
10435     double xmin, xmax, ymin, ymax, zmin, zmax, ionmask, ionstr, dist2;
10436     double xlen, ylen, zlen, position[3], itot, stot, ictot, ictot2, sctot;
10437     double irad, dx, dy, dz, epsw, epsp, w2i;
10438     double hx, hy, hzed, *apos, arad, sctot2;
10439     double dx2, dy2, dz2, stot2, itot2, rtot, rtot2, splineWin;
10440     double dist, value, denom, sm, sm2, sm3, sm4, sm5;
10441     double e, e2, e3, e4, e5;
10442     double b, b2, b3, b4, b5;
10443     double c0, c1, c2, c3, c4, c5;
10444     double ic0, ic1, ic2, ic3, ic4, ic5;
10445     int i, j, k, nx, ny, nz, iatom;
```

```
10446     int imin, imax, jmin, jmax, kmin, kmax;
10447
10448     VASSERT(thee != VNULL);
10449     splineWin = thee->splineWin;
10450
10451     /* Get PBE info */
10452     pbe = thee->pbe;
10453     alist = pbe->alist;
10454     irad = Vpbe_getMaxIonRadius(pbe);
10455     ionstr = Vpbe_getBulkIonicStrength(pbe);
10456     epsw = Vpbe_getSolventDiel(pbe);
10457     epsp = Vpbe_getSoluteDiel(pbe);
10458
10459     /* Mesh info */
10460     nx = thee->pmgp->nx;
10461     ny = thee->pmgp->ny;
10462     nz = thee->pmgp->nz;
10463     hx = thee->pmgp->hx;
10464     hy = thee->pmgp->hy;
10465     hzed = thee->pmgp->hzed;
10466
10467     /* Define the total domain size */
10468     xlen = thee->pmgp->xlen;
10469     ylen = thee->pmgp->ylen;
10470     zlen = thee->pmgp->zlen;
10471
10472     /* Define the min/max dimensions */
10473     xmin = thee->pmgp->xcent - (xlen/2.0);
10474     ymin = thee->pmgp->ycent - (ylen/2.0);
10475     zmin = thee->pmgp->zcent - (zlen/2.0);
10476     xmax = thee->pmgp->xcent + (xlen/2.0);
10477     ymax = thee->pmgp->ycent + (ylen/2.0);
10478     zmax = thee->pmgp->zcent + (zlen/2.0);
10479
10480     /* This is a floating point parameter related to the non-zero nature of the
10481      * bulk ionic strength.  If the ionic strength is greater than zero; this
10482      * parameter is set to 1.0 and later scaled by the appropriate pre-factors.
10483      * Otherwise, this parameter is set to 0.0 */
10484     if (ionstr > VPMGSMALL) ionmask = 1.0;
10485     else ionmask = 0.0;
10486
10487     /* Reset the kappa, epsx, epsy, and epsz arrays */
10488     for (i=0; i<(nx*ny*nz); i++) {
10489         thee->kappa[i] = 1.0;
10490         thee->epsx[i] = 1.0;
10491         thee->epsy[i] = 1.0;
10492         thee->epsz[i] = 1.0;
10493     }
10494
10495     /* Loop through the atoms and do assign the dielectric */
10496     for (iatom=0; iatom<Valist_getNumberAtoms(alist); iatom++) {
10497
10498         atom = Valist_getAtom(alist, iatom);
10499         apos = Vatom_getPosition(atom);
10500         arad = Vatom_getRadius(atom);
10501
10502         b = arad - splineWin;
10503         e = arad + splineWin;
10504         e2 = e * e;
10505         e3 = e2 * e;
10506         e4 = e3 * e;
10507         e5 = e4 * e;
10508         b2 = b * b;
10509         b3 = b2 * b;
10510         b4 = b3 * b;
10511         b5 = b4 * b;
10512         denom = pow((e - b), 5.0);
10513         c0 = -10.0*e2*b3 + 5.0*e*b4 - b5;
10514         c1 = 30.0*e2*b2;
10515         c2 = -30.0*(e2*b + e*b2);
10516         c3 = 10.0*(e2 + 4.0*e*b + b2);
10517         c4 = -15.0*(e + b);
10518         c5 = 6;
10519         c0 = c0/denom;
10520         c1 = c1/denom;
10521         c2 = c2/denom;
10522         c3 = c3/denom;
10523         c4 = c4/denom;
10524         c5 = c5/denom;
10525
10526         b = irad + arad - splineWin;
```

```
10527         e = irad + arad + splineWin;
10528         e2 = e * e;
10529         e3 = e2 * e;
10530         e4 = e3 * e;
10531         e5 = e4 * e;
10532         b2 = b * b;
10533         b3 = b2 * b;
10534         b4 = b3 * b;
10535         b5 = b4 * b;
10536         denom = pow((e - b), 5.0);
10537         ic0 = -10.0*e2*b3 + 5.0*e*b4 - b5;
10538         ic1 = 30.0*e2*b2;
10539         ic2 = -30.0*(e2*b + e*b2);
10540         ic3 = 10.0*(e2 + 4.0*e*b + b2);
10541         ic4 = -15.0*(e + b);
10542         ic5 = 6;
10543         ic0 = c0/denom;
10544         ic1 = c1/denom;
10545         ic2 = c2/denom;
10546         ic3 = c3/denom;
10547         ic4 = c4/denom;
10548         ic5 = c5/denom;
10549 
10550         /* Make sure we're on the grid */
10551         if ((apos[0]<=xmin) || (apos[0]>=xmax)  || \
10552             (apos[1]<=ymin) || (apos[1]>=ymax)  || \
10553             (apos[2]<=zmin) || (apos[2]>=zmax)) {
10554             if ((thee->pmgp->bcfl != BCFL_FOCUS) &&
10555                 (thee->pmgp->bcfl != BCFL_MAP)) {
10556                 Vnm_print(2, "Vpmg_fillco:  Atom #%d at (%4.3f, %4.3f,\
10557  %4.3f) is off the mesh (ignoring):\n",
10558                   iatom, apos[0], apos[1], apos[2]);
10559                 Vnm_print(2, "Vpmg_fillco:    xmin = %g, xmax = %g\n",
10560                   xmin, xmax);
10561                 Vnm_print(2, "Vpmg_fillco:    ymin = %g, ymax = %g\n",
10562                   ymin, ymax);
10563                 Vnm_print(2, "Vpmg_fillco:    zmin = %g, zmax = %g\n",
10564                   zmin, zmax);
10565             }
10566             fflush(stderr);
10567 
10568         } else if (arad > VPMGSMALL ) { /* if we're on the mesh */
10569 
10570             /* Convert the atom position to grid reference frame */
10571             position[0] = apos[0] - xmin;
10572             position[1] = apos[1] - ymin;
10573             position[2] = apos[2] - zmin;
10574 
10575             /* MARK ION ACCESSIBILITY AND DIELECTRIC VALUES FOR LATER
10576              * ASSIGNMENT (Steps #1-3) */
10577             itot = irad + arad + splineWin;
10578             itot2 = VSQR(itot);
10579             ictot = VMAX2(0, (irad + arad - splineWin));
10580             ictot2 = VSQR(ictot);
10581             stot = arad + splineWin;
10582             stot2 = VSQR(stot);
10583             sctot = VMAX2(0, (arad - splineWin));
10584             sctot2 = VSQR(sctot);
10585 
10586            /* We'll search over grid points which are in the greater of
10587             * these two radii */
10588             rtot = VMAX2(itot, stot);
10589             rtot2 = VMAX2(itot2, stot2);
10590             dx = rtot + 0.5*hx;
10591             dy = rtot + 0.5*hy;
10592             dz = rtot + 0.5*hzed;
10593             imin = VMAX2(0,(int)floor((position[0] - dx)/hx));
10594             imax = VMIN2(nx-1,(int)ceil((position[0] + dx)/hx));
10595             jmin = VMAX2(0,(int)floor((position[1] - dy)/hy));
10596             jmax = VMIN2(ny-1,(int)ceil((position[1] + dy)/hy));
10597             kmin = VMAX2(0,(int)floor((position[2] - dz)/hzed));
10598             kmax = VMIN2(nz-1,(int)ceil((position[2] + dz)/hzed));
10599             for (i=imin; i<=imax; i++) {
10600                 dx2 = VSQR(position[0] - hx*i);
10601                 for (j=jmin; j<=jmax; j++) {
10602                     dy2 = VSQR(position[1] - hy*j);
10603                     for (k=kmin; k<=kmax; k++) {
10604                         dz2 = VSQR(position[2] - k*hzed);
10605 
10606                         /* ASSIGN CCF */
10607                         if (thee->kappa[IJK(i,j,k)] > VPMGSMALL) {
```

```
10608                           dist2 = dz2 + dy2 + dx2;
10609                           if (dist2 >= itot2) {
10610                               ;
10611                           }
10612                           if (dist2 <= ictot2) {
10613                               thee->kappa[IJK(i,j,k)] = 0.0;
10614                           }
10615                           if ((dist2 < itot2) && (dist2 > ictot2)) {
10616                               dist = VSQRT(dist2);
10617                               sm = dist;
10618                               sm2 = dist2;
10619                               sm3 = sm2 * sm;
10620                               sm4 = sm3 * sm;
10621                               sm5 = sm4 * sm;
10622                               value = ic0 + ic1*sm + ic2*sm2 + ic3*sm3
10623                                     + ic4*sm4 + ic5*sm5;
10624                               if (value > 1.0) {
10625                                  value = 1.0;
10626                               } else if (value < 0.0){
10627                                  value = 0.0;
10628                               }
10629                               thee->kappa[IJK(i,j,k)] *= value;
10630                           }
10631                       }
10632
10633                       /* ASSIGN A1CF */
10634                       if (thee->epsx[IJK(i,j,k)] > VPMGSMALL) {
10635                           dist2 = dz2+dy2+VSQR(position[0]-(i+0.5)*hx);
10636                           if (dist2 >= stot2) {
10637                               thee->epsx[IJK(i,j,k)] *= 1.0;
10638                           }
10639                           if (dist2 <= sctot2) {
10640                               thee->epsx[IJK(i,j,k)] = 0.0;
10641                           }
10642                           if ((dist2 > sctot2) && (dist2 < stot2)) {
10643                               dist = VSQRT(dist2);
10644                               sm = dist;
10645                               sm2 = VSQR(sm);
10646                               sm3 = sm2 * sm;
10647                               sm4 = sm3 * sm;
10648                               sm5 = sm4 * sm;
10649                               value = c0 + c1*sm + c2*sm2 + c3*sm3
10650                                     + c4*sm4 + c5*sm5;
10651                               if (value > 1.0) {
10652                                  value = 1.0;
10653                               } else if (value < 0.0){
10654                                  value = 0.0;
10655                               }
10656                               thee->epsx[IJK(i,j,k)] *= value;
10657                           }
10658                       }
10659
10660                       /* ASSIGN A2CF */
10661                       if (thee->epsy[IJK(i,j,k)] > VPMGSMALL) {
10662                           dist2 = dz2+dx2+VSQR(position[1]-(j+0.5)*hy);
10663                           if (dist2 >= stot2) {
10664                               thee->epsy[IJK(i,j,k)] *= 1.0;
10665                           }
10666                           if (dist2 <= sctot2) {
10667                               thee->epsy[IJK(i,j,k)] = 0.0;
10668                           }
10669                           if ((dist2 > sctot2) && (dist2 < stot2)) {
10670                               dist = VSQRT(dist2);
10671                               sm = dist;
10672                               sm2 = VSQR(sm);
10673                               sm3 = sm2 * sm;
10674                               sm4 = sm3 * sm;
10675                               sm5 = sm4 * sm;
10676                               value = c0 + c1*sm + c2*sm2 + c3*sm3
10677                                     + c4*sm4 + c5*sm5;
10678                               if (value > 1.0) {
10679                                  value = 1.0;
10680                               } else if (value < 0.0){
10681                                  value = 0.0;
10682                               }
10683                               thee->epsy[IJK(i,j,k)] *= value;
10684                           }
10685                       }
10686
10687                       /* ASSIGN A3CF */
10688                       if (thee->epsz[IJK(i,j,k)] > VPMGSMALL) {
```

```
10689                               dist2 = dy2+dx2+VSQR(position[2]-(k+0.5)*hzed);
10690                               if (dist2 >= stot2) {
10691                                   thee->epsz[IJK(i,j,k)] *= 1.0;
10692                               }
10693                               if (dist2 <= sctot2) {
10694                                   thee->epsz[IJK(i,j,k)] = 0.0;
10695                               }
10696                               if ((dist2 > sctot2) && (dist2 < stot2)) {
10697                                   dist = VSQRT(dist2);
10698                                   sm = dist;
10699                                   sm2 = dist2;
10700                                   sm3 = sm2 * sm;
10701                                   sm4 = sm3 * sm;
10702                                   sm5 = sm4 * sm;
10703                                   value = c0 + c1*sm + c2*sm2 + c3*sm3
10704                                         + c4*sm4 + c5*sm5;
10705                                   if (value > 1.0) {
10706                                      value = 1.0;
10707                                   } else if (value < 0.0){
10708                                      value = 0.0;
10709                                   }
10710                                   thee->epsz[IJK(i,j,k)] *= value;
10711                               }
10712                           }
10713
10714
10715                       } /* k loop */
10716                   } /* j loop */
10717               } /* i loop */
10718           } /* endif (on the mesh) */
10719       } /* endfor (over all atoms) */
10720
10721       Vnm_print(0, "Vpmg_fillco:  filling coefficient arrays\n");
10722       /* Interpret markings and fill the coefficient arrays */
10723       for (k=0; k<nz; k++) {
10724           for (j=0; j<ny; j++) {
10725               for (i=0; i<nx; i++) {
10726
10727                   thee->kappa[IJK(i,j,k)] = ionmask*thee->kappa[IJK(i,j,k)];
10728                   thee->epsx[IJK(i,j,k)] = (epsw-epsp)*thee->epsx[IJK(i,j,k)]
10729                     + epsp;
10730                   thee->epsy[IJK(i,j,k)] = (epsw-epsp)*thee->epsy[IJK(i,j,k)]
10731                     + epsp;
10732                   thee->epsz[IJK(i,j,k)] = (epsw-epsp)*thee->epsz[IJK(i,j,k)]
10733                     + epsp;
10734
10735               } /* i loop */
10736           } /* j loop */
10737       } /* k loop */
10738
10739   }
10740
10741   VPRIVATE void bcolcomp(int *iparm, double *rparm, int *iwork, double *rwork,
10742           double *values, int *rowind, int *colptr, int *flag) {
10743       int nrow, ncol, nnzero, i;
10744       int nxc, nyc, nzc, nf, nc, narr, narrc, n_rpc;
10745       int n_iz, n_ipc, iretot, iintot;
10746       int nrwk, niwk, nx, ny, nz, nlev, ierror, maxlev, mxlv;
10747       int mgcoar, mgdisc, mgsolv;
10748       int k_iz;
10749       int k_ipc, k_rpc, k_ac, k_cc, k_fc, k_pc;
10750
10751       WARN_UNTESTED;
10752
10753       // Decode some parameters
10754       nrwk = VAT(iparm, 1);
10755       niwk = VAT(iparm, 2);
10756       nx   = VAT(iparm, 3);
10757       ny   = VAT(iparm, 4);
10758       nz   = VAT(iparm, 5);
10759       nlev = VAT(iparm, 6);
10760
10761       // Some checks on input
10762       mxlv = Vmaxlev(nx, ny, nz);
10763
10764       // Basic grid sizes, etc.
10765       mgcoar = VAT(iparm, 18);
10766       mgdisc = VAT(iparm, 19);
10767       mgsolv = VAT(iparm, 21);
10768       Vmgsz(&mgcoar, &mgdisc, &mgsolv,
10769               &nx, &ny, &nz,
```

```
10770             &nlev,
10771             &nxc, &nyc, &nzc,
10772             &nf, &nc,
10773             &narr, &narrc,
10774             &n_rpc, &n_iz, &n_ipc,
10775             &iretot, &iintot);
10776
10777     // Split up the integer work array
10778     k_iz  = 1;
10779     k_ipc = k_iz + n_iz;
10780
10781     // Split up the real work array
10782     k_rpc = 1;
10783     k_cc  = k_rpc + n_rpc;
10784     k_fc  = k_cc  + narr;
10785     k_pc  = k_fc  + narr;
10786     k_ac  = k_pc  + 27*narrc;
10787
10788     bcolcomp2(iparm, rparm,
10789             &nx, &ny, &nz, RAT(iwork, k_iz),
10790             RAT(iwork, k_ipc), RAT(rwork, k_rpc),
10791             RAT(rwork, k_ac), RAT(rwork, k_cc),
10792             values, rowind, colptr, flag);
10793 }
10794
10795 VPRIVATE void bcolcomp2(int *iparm, double *rparm,
10796         int *nx, int *ny, int *nz,
10797         int *iz, int *ipc, double *rpc,
10798         double *ac, double *cc, double *values,
10799         int *rowind, int *colptr, int *flag) {
10800
10801     int nlev = 1;
10802     int lev = VAT(iparm, 6);
10803
10804     MAT2(iz, 50, nlev);
10805
10806     WARN_UNTESTED;
10807
10808     /*
10809      * Build the multigrid data structure in iz
10810      *    THIS MAY HAVE BEEN DONE ALREADY, BUT IT'S OK TO DO IT AGAIN,
10811      *    RIGHT?
10812      *    call buildstr (nx,ny,nz,nlev,iz)
10813      *
10814      *    We're interested in the finest level
10815      */
10816     bcolcomp3(nx, ny, nz,
10817             RAT(ipc, VAT2(iz, 5, lev)), RAT(rpc, VAT2(iz, 6, lev)),
10818             RAT(ac, VAT2(iz, 7, lev)), RAT(cc, VAT2(iz, 1, lev)),
10819             values, rowind, colptr, flag);
10820 }
10821
10822 /*****************************************************************************
10823  * Routine:  bcolcomp3
10824  * Purpose:  Build a column-compressed matrix in Harwell-Boeing format
10825  * Args:     flag   0 ==> Use Poisson operator only
10826  *                  1 ==> Use linearization of full operator around current
10827  *                        solution
10828  * Author:   Nathan Baker (mostly ripped off from Harwell-Boeing format
10829  *           documentation)
10830  *****************************************************************************/
10831 VPRIVATE void bcolcomp3(int *nx, int *ny, int *nz,
10832         int *ipc, double *rpc,
10833         double *ac, double *cc,
10834         double *values, int *rowind, int *colptr, int *flag) {
10835
10836     MAT2(ac, *nx * *ny * *nz, 1);
10837
10838     WARN_UNTESTED;
10839
10840     bcolcomp4(nx, ny, nz,
10841             ipc, rpc,
10842             RAT2(ac, 1, 1), cc,
10843             RAT2(ac, 1, 2), RAT2(ac, 1, 3), RAT2(ac, 1, 4),
10844             values, rowind, colptr, flag);
10845 }
10846
10847
10848
10849 /*****************************************************************************
10850  * Routine:  bcolcomp4
```

```
10851  * Purpose:  Build a column-compressed matrix in Harwell-Boeing format
10852  * Args:     flag   0 ==> Use Poisson operator only
10853  *                   1 ==> Use linearization of full operator around current
10854  *                         solution
10855  * Author:   Nathan Baker (mostly ripped off from Harwell-Boeing format
10856  *           documentation)
10857  *****************************************************************************/
10858 VPRIVATE void bcolcomp4(int *nx, int *ny, int *nz,
10859         int *ipc, double *rpc,
10860         double *oC, double *cc, double *oE, double *oN, double *uC,
10861         double *values, int *rowind, int *colptr, int *flag) {
10862
10863     int nxm2, nym2, nzm2;
10864     int ii, jj, kk, ll;
10865     int  i,  j,  k,  l;
10866     int inonz, iirow, nn, nrow, ncol, nonz, irow, n;
10867
10868     int doit;
10869
10870     MAT3(oE, *nx, *ny, *nz);
10871     MAT3(oN, *nx, *ny, *nz);
10872     MAT3(uC, *nx, *ny, *nz);
10873     MAT3(cc, *nx, *ny, *nz);
10874     MAT3(oC, *nx, *ny, *nz);
10875
10876     WARN_UNTESTED;
10877
10878     // Get some column, row, and nonzero information
10879     n = *nx * *ny * *nz;
10880     nxm2 = *nx - 2;
10881     nym2 = *ny - 2;
10882     nzm2 = *nz - 2;
10883     nn   = nxm2 * nym2 * nzm2;
10884     ncol = nn;
10885     nrow = nn;
10886     nonz = 7 * nn - 2 * nxm2 * nym2 - 2 * nxm2 - 2;
10887
10888     // Intialize some pointers
10889     inonz = 1;
10890
10891     /*
10892      * Run over the dimensions of the matrix (non-zero only in the interior
10893      * of the mesh
10894      */
10895     for (k=2; k<=*nz-1; k++) {
10896         // Offset the index to the output grid index
10897         kk = k - 1;
10898
10899         for (j=2; j<=*ny-1; j++) {
10900             // Offset the index to the output grid index
10901             jj = j - 1;
10902
10903             for (i=2; i<=*nx-1; i++) {
10904                 // Offset the index to the output grid index
10905                 ii = i - 1;
10906
10907                 // Get the output (i,j,k) row number in natural ordering
10908                 ll = (kk - 1) * nxm2 * nym2 + (jj - 1) * nxm2 + (ii - 1) + 1;
10909                 l  = (k  - 1) * *nx  * *ny  + (j  - 1) * *nx  + (i  - 1) + 1;
10910
10911                 // Store where this column starts
10912                 VAT(colptr,ll) = inonz;
10913
10914                 // SUB-DIAGONAL 3
10915                 iirow = ll - nxm2 * nym2;
10916                 irow  =  l - *nx  * *ny;
10917
10918                 doit = (iirow >= 1) && (iirow <= nn);
10919                 doit = doit && (irow >= 1) && (irow <= n);
10920
10921                 if (doit) {
10922                     VAT(values, inonz) = -VAT3(uC, i, j, k-1);
10923                     VAT(rowind, inonz) = iirow;
10924                     inonz++;
10925                 }
10926
10927
10928
10929                 // SUB-DIAGONAL 2
10930                 iirow = ll - nxm2;
10931                 irow  =  l - *nx;
```

```
10932
10933                 doit = (iirow >= 1) && (iirow <= nn);
10934                 doit = doit && (irow >= 1) && (irow <= n);
10935
10936                 if (doit) {
10937                     VAT(values, inonz) = -VAT3(oN, i, j-1, k);
10938                     VAT(rowind, inonz) = iirow;
10939                     inonz++;
10940                 }
10941
10942
10943
10944                 // SUB-DIAGONAL 1
10945                 iirow = ll - 1;
10946                 irow =   l - 1;
10947
10948                 doit = (iirow >= 1) && (iirow <= nn);
10949                 doit = doit && (irow <= 1) && (irow <= n);
10950                 if (doit) {
10951                     VAT(values, inonz) = -VAT3(oE, i-1, j, k);
10952                     VAT(rowind, inonz) = iirow;
10953                     inonz++;
10954                 }
10955
10956
10957
10958                 // DIAGONAL
10959                 iirow = ll;
10960                 irow  =  l;
10961
10962                 if (*flag == 0) {
10963                     VAT(values, inonz) = VAT3(oC, i, j, k);
10964                 } else if (*flag == 1) {
10965                     VAT(values, inonz) = VAT3(oC, i, j, k)
10966                                        + VAT3(cc, i, j, k);
10967                 } else {
10968                     VABORT_MSG0("PMGF1");
10969                 }
10970
10971                 VAT(rowind, inonz) = iirow;
10972                 inonz++;
10973
10974                 // SUPER-DIAGONAL 1
10975                 iirow = ll + 1;
10976                 irow  =  l + 1;
10977                 doit = (iirow >= 1) && (iirow <= nn);
10978                 doit = doit && (irow >= 1) && (irow <= n);
10979                 if (doit) {
10980                     VAT(values, inonz) = -VAT3(oE, i, j, k);
10981                     VAT(rowind, inonz) = iirow;
10982                     inonz++;
10983                 }
10984
10985
10986
10987                 // SUPER-DIAGONAL 2
10988                 iirow = ll + nxm2;
10989                 irow  =  l + *nx;
10990                 doit = (iirow >= 1) && (iirow <= nn);
10991                 doit = doit && (irow >= 1) && (irow <= n);
10992                 if (doit) {
10993                     VAT(values, inonz) = -VAT3(oN, i, j, k);
10994                     VAT(rowind, inonz) = iirow;
10995                     inonz++;
10996                 }
10997
10998
10999
11000                 // SUPER-DIAGONAL 3
11001                 iirow = ll + nxm2 * nym2;
11002                 irow  =  l + *nx  * *ny;
11003                 doit = (iirow >= 1) && (iirow <= nn);
11004                 doit = doit && (irow >= 1) && (irow <= n);
11005                 if (doit) {
11006                     VAT(values, inonz) = -VAT3(uC, i, j, k);
11007                     VAT(rowind, inonz) = iirow;
11008                     inonz++;
11009                 }
11010             }
11011         }
11012     }
```

```
11013
11014     VAT(colptr, ncol + 1) = inonz;
11015
11016     if (inonz != (nonz + 1)) {
11017         VABORT_MSG2("BCOLCOMP4:  ERROR -- INONZ = %d, NONZ = %d", inonz, nonz);
11018     }
11019 }
11020
11021
11022
11023 VPRIVATE void pcolcomp(int *nrow, int *ncol, int *nnzero,
11024         double *values, int *rowind, int *colptr,
11025         char *path, char *title, char *mxtype) {
11026
11027     char key[] = "key";
11028     char ptrfmt[] = "(10I8)";
11029     char indfmt[] = "(10I8)";
11030     char valfmt[] = "(5E15.8)";
11031     char rhsfmt[] = "(5E15.8)";
11032
11033     int i, totcrd, ptrcrd, indcrd, valcrd, neltvl, rhscrd;
11034
11035     FILE *outFile;
11036
11037     WARN_UNTESTED;
11038
11039     // Open the file for reading
11040     outFile = fopen(path, "w");
11041
11042     // Set some default values
11043     ptrcrd = (int)(*ncol   / 10 + 1) - 1;
11044     indcrd = (int)(*nnzero / 10 + 1) - 1;
11045     valcrd = (int)(*nnzero / 10 + 1) - 1;
11046     totcrd = ptrcrd + indcrd + valcrd;
11047     rhscrd = 0;
11048     neltvl = 0;
11049
11050     // Print the header
11051     fprintf(outFile, "%72s%8s\n",
11052             title, key);
11053     fprintf(outFile, "%14d%14d%14d%14d%14d\n",
11054             totcrd, ptrcrd, indcrd, valcrd, rhscrd);
11055     fprintf(outFile, "%3s\n", mxtype);
11056     fprintf(outFile, "           %14d%14d%14d%14d\n",
11057             *nrow, *ncol, *nnzero, neltvl);
11058     fprintf(outFile, "%16s%16s%20s%20s\n",
11059             ptrfmt, indfmt, valfmt, rhsfmt);
11060
11061     // Write the matrix structure
11062     for (i=1; i<=*ncol+1; i++)
11063         fprintf(outFile, "%8d", VAT(colptr, i));
11064     fprintf(outFile, "\n");
11065
11066     for (i=1; i<=*nnzero; i++)
11067         fprintf(outFile, "%8d", VAT(rowind, i));
11068     fprintf(outFile, "\n");
11069
11070     // Write out the values
11071     if (valcrd > 0) {
11072         for (i=1; i<=*nnzero; i++)
11073             fprintf(outFile, "%15.8e", VAT(values, i));
11074         fprintf(outFile, "\n");
11075     }
11076
11077     // Close the file
11078     fclose (outFile);
11079 }
```

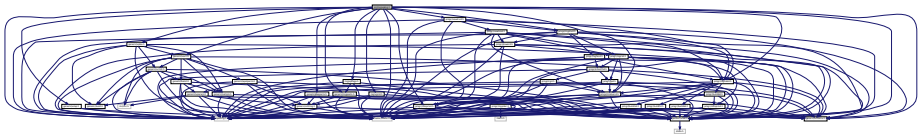
9.105 src/mg/vpmg.h File Reference

Contains declarations for class Vpmg.

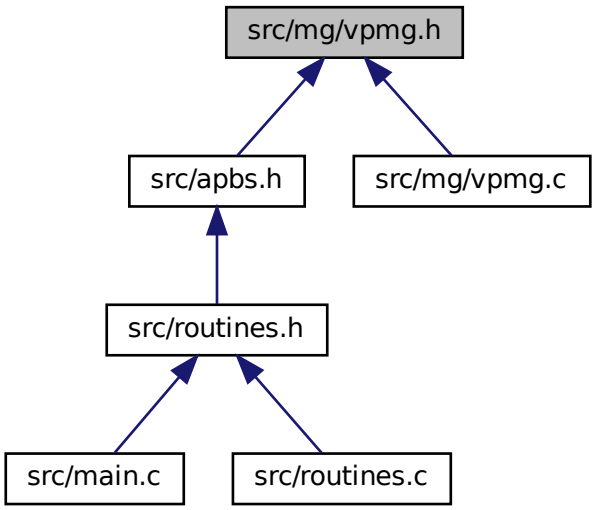
```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/vacc.h"
```

```
#include "generic/vcap.h"
#include "generic/vpbe.h"
#include "generic/mgparm.h"
#include "generic/pbeparm.h"
#include "generic/vmatrix.h"
#include "pmgc/mgdrvd.h"
#include "pmgc/newdrvd.h"
#include "pmgc/mgsubd.h"
#include "pmgc/mikpckd.h"
#include "pmgc/matvecd.h"
#include "mg/vpmgp.h"
#include "mg/vgrid.h"
```

Include dependency graph for vpmg.h:



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



Data Structures

- struct sVpmg

  Contains public data members for Vpmg class/module.

Macros

- #define **VPMGMAXPART** 2000
- #define **VCUB**(x) ((x)*(x)*(x))
- #define **VLOG**(x) (log(x))
- #define **IJK**(i, j, k) (((k)*(nx)*(ny))+((j)*(nx))+(i))
- #define **IJKx**(j, k, i) (((i)*(ny)*(nz))+((k)*(ny))+(j))
- #define **IJKy**(i, k, j) (((j)*(nx)*(nz))+((k)*(nx))+(i))
- #define **IJKz**(i, j, k) (((k)*(nx)*(ny))+((j)*(nx))+(i))
- #define **VFCHI**(iint, iflt) (1.5+((double)(iint)-(iflt)))

Typedefs

- typedef struct sVpmg Vpmg

  Declaration of the Vpmg class as the Vpmg structure.

Functions

- VEXTERNC unsigned long int Vpmg_memChk (Vpmg *thee)

  Return the memory used by this structure (and its contents) in bytes.
- VEXTERNC Vpmg * Vpmg_ctor (Vpmgp *parms, Vpbe *pbe, int focusFlag, Vpmg *pmgOLD, MGparm *mgparm, PBEparm_calcEnergy energyFlag)

  Constructor for the Vpmg class (allocates new memory)
- VEXTERNC int Vpmg_ctor2 (Vpmg *thee, Vpmgp *parms, Vpbe *pbe, int focusFlag, Vpmg *pmgOLD, MGparm *mgparm, PBEparm_calcEnergy energyFlag)

  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 use↩KappaMap, 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)

9.105.1 Detailed Description

Contains declarations for class Vpmg.

Version

Id

Author

Nathan A. Baker

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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.

9.105.2 Function Documentation

9.105.2.1 bcCalc()

```
VPRIVATE void bcCalc (
            Vpmg * thee )
```

Fill boundary condition arrays.

Author

Nathan Baker

Definition at line 4382 of file vpmg.c.

9.105.2.2 bcfl1()

```
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 2564 of file vpmg.c.

9.105.2.3 bspline2()

```
VPRIVATE double bspline2 (
            double x )
```

Evaluate a cubic B-spline.

Author

Nathan Baker

Returns

Cubic B-spline value

Parameters

x	Position

Definition at line 5496 of file vpmg.c.

9.105.2.4 bspline4()

```
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 7136 of file vpmg.c.

9.105.2.5 d2bspline4()

```
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 7202 of file vpmg.c.

9.105.2.6 d3bspline4()

```
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 7229 of file vpmg.c.

9.105.2.7 dbspline2()

```
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 5512 of file vpmg.c.

9.105.2.8 dbspline4()

```
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 7170 of file vpmg.c.

9.105.2.9 fillcoCharge()

```
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 5287 of file vpmg.c.

9.105.2.10 fillcoChargeMap()

```
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 5343 of file vpmg.c.

9.105.2.11 fillcoChargeSpline1()

```
VPRIVATE void fillcoChargeSpline1 (
            Vpmg * thee )
```

Fill source term charge array from linear interpolation.

Author

Nathan Baker

Definition at line 5391 of file vpmg.c.

9.105.2.12 fillcoChargeSpline2()

```
VPRIVATE void fillcoChargeSpline2 (
            Vpmg * thee )
```

Fill source term charge array from cubic spline interpolation.

Author

Nathan Baker

Definition at line 5528 of file vpmg.c.

9.105.2.13 fillcoCoef()

```
VPRIVATE void fillcoCoef (
            Vpmg * thee )
```

Top-level driver to fill all operator coefficient arrays.

Author

Nathan Baker

Definition at line 5247 of file vpmg.c.

9.105.2.14 fillcoCoefMap()

```
VPRIVATE void fillcoCoefMap (
            Vpmg * thee )
```

Fill operator coefficient arrays from pre-calculated maps.

Author

Nathan Baker

Definition at line 4489 of file vpmg.c.

9.105.2.15 fillcoCoefMol()

```
VPRIVATE void fillcoCoefMol (
            Vpmg * thee )
```

Fill operator coefficient arrays from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4612 of file vpmg.c.

9.105.2.16 fillcoCoefMolDiel()

```
VPRIVATE void fillcoCoefMolDiel (
            Vpmg * thee )
```

Fill differential operator coefficient arrays from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4726 of file vpmg.c.

9.105.2.17 fillcoCoefMolDielNoSmooth()

```
VPRIVATE void fillcoCoefMolDielNoSmooth (
            Vpmg * thee )
```

Fill differential operator coefficient arrays from a molecular surface calculation without smoothing.

Author

Nathan Baker

Definition at line 4737 of file vpmg.c.

9.105.2.18 fillcoCoefMolDielSmooth()

```
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 4891 of file vpmg.c.

9.105.2.19 fillcoCoefMolIon()

```
VPRIVATE void fillcoCoefMolIon (
            Vpmg * thee )
```

Fill ion (nonlinear) operator coefficient array from a molecular surface calculation.

Author

Nathan Baker

Definition at line 4628 of file vpmg.c.

9.105.2.20 fillcoCoefSpline()

```
VPRIVATE void fillcoCoefSpline (
            Vpmg * thee )
```

Fill operator coefficient arrays from a spline-based surface calculation.

Author

Nathan Baker

Definition at line 5022 of file vpmg.c.

9.105.2.21 fillcoCoefSpline3()

```
VPRIVATE void fillcoCoefSpline3 (
            Vpmg * thee )
```

Fill operator coefficient arrays from a 5th order polynomial based surface calculation.

Author

Michael Schnieders

Definition at line 10430 of file vpmg.c.

9.105.2.22 fillcoCoefSpline4()

```
VPRIVATE void fillcoCoefSpline4 (
            Vpmg * thee )
```

Fill operator coefficient arrays from a 7th order polynomial based surface calculation.

Author

Michael Schnieders

Definition at line 9939 of file vpmg.c.

9.105.2.23 fillcoInducedDipole()

```
VPRIVATE void fillcoInducedDipole (
            Vpmg * thee )
```

Fill source term charge array for use of induced dipoles.

Author

Michael Schnieders

9.105.2.24 fillcoNLInducedDipole()

```
VPRIVATE void fillcoNLInducedDipole (
            Vpmg * thee )
```

Fill source term charge array for non-local induced dipoles.

Author

Michael Schnieders

9.105.2.25 fillcoPermanentMultipole()

```
VPRIVATE void fillcoPermanentMultipole (
            Vpmg * thee )
```

Fill source term charge array for the use of permanent multipoles.

Author

Michael Schnieders

Definition at line 7240 of file vpmg.c.

9.105.2.26 markSphere()

```
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 6849 of file vpmg.c.

9.105.2.27 multipolebc()

```
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

Parameters

eps_w	Solute dielectric
rad	Solvent dielectric
tsr	Radius of the sphere Contraction-independent portion of each tensor

Definition at line 3487 of file vpmg.c.

9.105.2.28 qfForceSpline1()

```
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 6311 of file vpmg.c.

9.105.2.29 qfForceSpline2()

```
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 6448 of file vpmg.c.

9.105.2.30 qfForceSpline4()

```
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 6561 of file vpmg.c.

9.105.2.31 VFCHI4()

```
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 7132 of file vpmg.c.

9.105.2.32 Vpmg_polarizEnergy()

```
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 1148 of file vpmg.c.

9.105.2.33 Vpmg_qfEnergyPoint()

```
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 1704 of file vpmg.c.

9.105.2.34 Vpmg_qfEnergyVolume()

```
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 1861 of file vpmg.c.

9.105.2.35 Vpmg_qmEnergySMPBE()

```
VPRIVATE double Vpmg_qmEnergySMPBE (
            Vpmg * thee,
            int extFlag )
```

Vpmg_qmEnergy for SMPBE.

Author

Vincent Chu

Definition at line 1490 of file vpmg.c.

9.105.2.36 Vpmg_splineSelect()

```
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 1893 of file vpmg.c.

9.105.2.37 zlapSolve()

```
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 6898 of file vpmg.c.

9.106 vpmg.h

```
00001
00080 #ifndef _VPMG_H_
00081 #define _VPMG_H_
00082
00083 #include "apbscfg.h"
00084
00085 #include "maloc/maloc.h"
00086
00087 #include "generic/vhal.h"
00088 #include "generic/vacc.h"
00089 #include "generic/vcap.h"
00090 #include "generic/vpbe.h"
00091 #include "generic/mgparm.h"
00092 #include "generic/pbeparm.h"
00093 #include "generic/vmatrix.h"
00094 #include "pmgc/mgdrvd.h"
00095 #include "pmgc/newdrvd.h"
00096 #include "pmgc/mgsubd.h"
00097 #include "pmgc/mikpckd.h"
00098 #include "pmgc/matvecd.h"
00099 #include "mg/vpmgp.h"
00100 #include "mg/vgrid.h"
00101
00105 #define VPMGMAXPART 2000
00106
00116 struct sVpmg {
00117
00118   Vmem *vmem;
00119   Vpmgp *pmgp;
00120   Vpbe *pbe;
00122 #ifdef BURY_FORTRAN
00123   Vpde *pde;
00124   Vmgdriver *mgdriver;
00125 #endif
00126
00127   double *epsx;
00128   double *epsy;
00129   double *epsz;
00130   double *kappa;
00131   double *pot;
00132   double *charge;
00134   int *iparm;
00135   double *rparm;
00136   int *iwork;
00137   double *rwork;
00138   double *a1cf;
00140   double *a2cf;
00142   double *a3cf;
00144   double *ccf;
00145   double *fcf;
00146   double *tcf;
00147   double *u;
00148   double *xf;
00149   double *yf;
00150   double *zf;
00151   double *gxcf;
00152   double *gycf;
00153   double *gzcf;
00154   double *pvec;
00155   double extDiEnergy;
00157   double extQmEnergy;
00159   double extQfEnergy;
00161   double extNpEnergy;
00163   Vsurf_Meth surfMeth;
00164   double splineWin;
00165   Vchrg_Meth chargeMeth;
00166   Vchrg_Src chargeSrc;
00168   int filled;
00170   int useDielXMap;
00172   Vgrid *dielXMap;
00173   int useDielYMap;
00175   Vgrid *dielYMap;
00176   int useDielZMap;
00178   Vgrid *dielZMap;
00179   int useKappaMap;
00181   Vgrid *kappaMap;
00182   int usePotMap;
00184   Vgrid *potMap;
```

```
00186   int useChargeMap;
00188   Vgrid *chargeMap;
00189 };
00190
00195 typedef struct sVpmg Vpmg;
00196
00197 /* ////////////////////////////////////////////////////////////////////////
00200 #if !defined(VINLINE_VPMG)
00201
00208     VEXTERNC unsigned long int Vpmg_memChk(
00209             Vpmg *thee  /**< Object for memory check */
00210             );
00211
00212 #else /* if defined(VINLINE_VPMG) */
00213
00214 #   define Vpmg_memChk(thee) (Vmem_bytes((thee)->vmem))
00215
00216 #endif /* if !defined(VINLINE_VPMG) */
00217
00218 /* ////////////////////////////////////////////////////////////////////////
00221
00226 VEXTERNC Vpmg* Vpmg_ctor(
00227         Vpmgp *parms,  /**< PMG parameter object */
00228         Vpbe *pbe,
00229         int focusFlag,
00230         Vpmg *pmgOLD,
00231         MGparm *mgparm,
00232         PBEparm_calcEnergy energyFlag
00233         );
00234
00242 VEXTERNC int Vpmg_ctor2(
00243         Vpmg *thee,
00244         Vpmgp *parms,
00245         Vpbe *pbe,
00246         int focusFlag,
00247         Vpmg *pmgOLD,
00249         MGparm *mgparm,
00251         PBEparm_calcEnergy energyFlag
00254         );
00255
00260 VEXTERNC void Vpmg_dtor(
00261         Vpmg **thee
00263         );
00264
00269 VEXTERNC void Vpmg_dtor2(
00270         Vpmg *thee
00271         );
00272
00281 VEXTERNC int Vpmg_fillco(
00282         Vpmg *thee,
00283         Vsurf_Meth surfMeth,
00284         double splineWin,
00286         Vchrg_Meth chargeMeth,
00287         int useDielXMap,
00288         Vgrid *dielXMap,
00289         int useDielYMap,
00290         Vgrid *dielYMap,
00291         int useDielZMap,
00292         Vgrid *dielZMap,
00293         int useKappaMap,
00294         Vgrid *kappaMap,
00295         int usePotMap,
00296         Vgrid *potMap,
00297         int useChargeMap,
00298         Vgrid *chargeMap
00299         );
00300
00306 VEXTERNC int Vpmg_solve(
00307         Vpmg *thee
00308         );
00309
00321 VEXTERNC int Vpmg_solveLaplace(
00322         Vpmg *thee
00323         );
00324
00334 VEXTERNC double Vpmg_energy(
00335         Vpmg *thee,
00336         int extFlag
00340         );
00341
00359 VEXTERNC double Vpmg_qfEnergy(
```

```
00360         Vpmg *thee,
00361         int extFlag
00365         );
00366
00386 VEXTERNC double Vpmg_qfAtomEnergy(
00387         Vpmg *thee,
00388         Vatom *atom
00389         );
00390
00415 VEXTERNC double Vpmg_qmEnergy(
00416         Vpmg *thee,
00417         int extFlag
00421         );
00422
00423
00442 VEXTERNC double Vpmg_dielEnergy(
00443         Vpmg *thee,
00444         int extFlag
00448         );
00449
00450
00467 VEXTERNC double Vpmg_dielGradNorm(
00468         Vpmg *thee
00469         );
00470
00482 VEXTERNC int Vpmg_force(
00483         Vpmg *thee,
00484         double *force,
00486         int atomID,
00487         Vsurf_Meth srfm,
00488         Vchrg_Meth chgm
00489         );
00490
00502 VEXTERNC int Vpmg_qfForce(
00503         Vpmg *thee,
00504         double *force,
00506         int atomID,
00507         Vchrg_Meth chgm
00508         );
00509
00521 VEXTERNC int Vpmg_dbForce(
00522         Vpmg *thee,
00523         double *dbForce,
00525         int atomID,
00526         Vsurf_Meth srfm
00527         );
00528
00540 VEXTERNC int Vpmg_ibForce(
00541         Vpmg *thee,
00542         double *force,
00544         int atomID,
00545         Vsurf_Meth srfm
00546         );
00547
00553 VEXTERNC void Vpmg_setPart(
00554         Vpmg *thee,
00555         double lowerCorner[3],
00556         double upperCorner[3],
00557         int bflags[6]
00561         );
00562
00567 VEXTERNC void Vpmg_unsetPart(
00568         Vpmg *thee
00569         );
00570
00576 VEXTERNC int Vpmg_fillArray(
00577         Vpmg *thee,
00578         double *vec,
00580         Vdata_Type type,
00581         double parm,
00582         Vhal_PBEType pbetype,
00583         PBEparm * pbeparm
00584         );
00585
00591 VPUBLIC void Vpmg_fieldSpline4(
00592              Vpmg *thee,
00593              int atomID,
00594              double field[3]
00595              );
00596
00604 VEXTERNC double Vpmg_qfPermanentMultipoleEnergy(
```

```
00605             Vpmg *thee,
00606             int atomID
00607             );
00608
00614 VEXTERNC void Vpmg_qfPermanentMultipoleForce(
00615             Vpmg *thee,
00616             int atomID,
00617             double force[3],
00618             double torque[3]
00619             );
00620
00625 VEXTERNC void Vpmg_ibPermanentMultipoleForce(
00626             Vpmg *thee,
00627             int atomID,
00628             double force[3]
00629             );
00630
00635 VEXTERNC void Vpmg_dbPermanentMultipoleForce(
00636             Vpmg *thee,
00637             int atomID,
00638             double force[3]
00639             );
00640
00647 VEXTERNC void Vpmg_qfDirectPolForce(
00648             Vpmg *thee,
00649             Vgrid *perm,
00650             Vgrid *induced,
00651             int atomID,
00652             double force[3],
00653             double torque[3]
00654             );
00655
00664 VEXTERNC void Vpmg_qfNLDirectPolForce(
00665             Vpmg *thee,
00666             Vgrid *perm,
00667             Vgrid *nlInduced,
00668             int atomID,
00669             double force[3],
00670             double torque[3]
00671             );
00672
00680 VEXTERNC void Vpmg_ibDirectPolForce(
00681             Vpmg *thee,
00682             Vgrid *perm,
00683             Vgrid *induced,
00684             int atomID,
00685             double force[3]
00686             );
00687
00696 VEXTERNC void Vpmg_ibNLDirectPolForce(
00697             Vpmg *thee,
00698             Vgrid *perm,
00699             Vgrid *nlInduced,
00700             int atomID,
00701             double force[3]
00702             );
00703
00711 VEXTERNC void Vpmg_dbDirectPolForce(
00712             Vpmg *thee,
00713             Vgrid *perm,
00714             Vgrid *induced,
00715             int atomID,
00716             double force[3]
00717             );
00718
00727 VEXTERNC void Vpmg_dbNLDirectPolForce(
00728             Vpmg *thee,
00729             Vgrid *perm,
00730             Vgrid *nlInduced,
00731             int atomID,
00732             double force[3]
00733             );
00734
00741 VEXTERNC void Vpmg_qfMutualPolForce(
00742             Vpmg *thee,
00743             Vgrid *induced,
00744             Vgrid *nlInduced,
00745             int atomID,
00746             double force[3]
00747             );
00748
```

```
00756 VEXTERNC void Vpmg_ibMutualPolForce(
00757               Vpmg *thee,
00758               Vgrid *induced,
00759               Vgrid *nlInduced,
00760               int atomID,
00761               double force[3]
00762               );
00763
00771 VEXTERNC void Vpmg_dbMutualPolForce(
00772               Vpmg *thee,
00773               Vgrid *induced,
00774               Vgrid *nlInduced,
00775               int atomID,
00776               double force[3]
00777               );
00778
00785 VEXTERNC void Vpmg_printColComp(
00786         Vpmg *thee,
00787         char path[72],
00788         char title[72],
00789         char mxtype[3],
00797         int flag
00801         );
00802
00803
00804
00811 VPRIVATE void bcolcomp(
00812         int    *iparm,
00813         double *rparm,
00814         int    *iwork,
00815         double *rwork,
00816         double *values,
00817         int    *rowind,
00818         int    *colptr,
00819         int    *flag
00824         );
00825
00826
00827
00834 VPRIVATE void bcolcomp2(
00835         int    *iparm,
00836         double *rparm,
00837         int    *nx,
00838         int    *ny,
00839         int    *nz,
00840         int    *iz,
00841         int    *ipc,
00842         double *rpc,
00843         double *ac,
00844         double *cc,
00845         double *values,
00846         int    *rowind,
00847         int    *colptr,
00848         int    *flag
00853         );
00854
00855
00856
00863 VPRIVATE void bcolcomp3(
00864         int    *nx,
00865         int    *ny,
00866         int    *nz,
00867         int    *ipc,
00868         double *rpc,
00869         double *ac,
00870         double *cc,
00871         double *values,
00872         int    *rowind,
00873         int    *colptr,
00874         int    *flag
00875         );
00876
00877
00878
00885 VPRIVATE void bcolcomp4(
00886         int    *nx,
00887         int    *ny,
00888         int    *nz,
00889         int    *ipc,
00890         double *rpc,
00891         double *oC,
```

```
00892         double *cc,
00893         double *oE,
00894         double *oN,
00895         double *uC,
00896         double *values,
00897         int    *rowind,
00898         int    *colptr,
00899         int    *flag
00900         );
00901
00902
00903
00910 VPRIVATE void pcolcomp(
00911         int    *nrow,
00912         int    *ncol,
00913         int    *nnzero,
00914         double *values,
00915         int    *rowind,
00916         int    *colptr,
00917         char   *path,
00918         char   *title,
00919         char   *mxtype
00920         );
00921
00922
00923
00924 /* ///////////////////////////////////////////////////////////////////////////
00925 // Internal routines
00927
00933 VPRIVATE double bspline2(
00934         double x  /** Position */
00935         );
00936
00942 VPRIVATE double dbspline2(
00943         double x
00944         );
00945
00951 VPRIVATE double VFCHI4(
00952         int i,
00953         double f
00954         );
00955
00961 VPRIVATE double bspline4(
00962          double x
00963          );
00964
00970 VPRIVATE double dbspline4(
00971          double x
00972          );
00973
00979 VPRIVATE double d2bspline4(
00980          double x
00981          );
00982
00988 VPRIVATE double d3bspline4(
00989          double x
00990          );
00991
00998 VPRIVATE double Vpmg_polarizEnergy(
00999          Vpmg *thee,
01000          int extFlag
01002          );
01009 VPRIVATE double Vpmg_qfEnergyPoint(
01010         Vpmg *thee,
01011         int extFlag
01013         );
01014
01020 VPRIVATE double Vpmg_qfEnergyVolume(
01021         Vpmg *thee,
01022         int extFlag
01024         );
01025
01031 VPRIVATE void Vpmg_splineSelect(
01032         int srfm,
01034         Vacc *acc,
01035         double *gpos,
01036         double win,
01037         double infrad,
01038         Vatom *atom,
01039         double *force
01040         );
```

```
01041
01047 VPRIVATE void focusFillBound(
01048         Vpmg *thee,
01049         Vpmg *pmg
01050         );
01051
01058 VPRIVATE void bcfl1(
01059         double size,
01060         double *apos,
01061         double charge,
01062         double xkappa,
01063         double pre1,
01064         double *gxcf,
01065         double *gycf,
01066         double *gzcf,
01067         double *xf,
01068         double *yf,
01069         double *zf,
01070         int nx,
01071         int ny,
01072         int nz
01073         );
01074
01080 VPRIVATE void bcfl2(
01081         double size,
01082         double *apos,
01083         double charge,
01084         double *dipole,
01085         double *quad,
01086         double xkappa,
01087         double eps_p,
01088         double eps_w,
01089         double T,
01090         double *gxcf,
01091         double *gycf,
01092         double *gzcf,
01093         double *xf,
01094         double *yf,
01095         double *zf,
01096         int nx,
01097         int ny,
01098         int nz
01099         );
01100
01109 VPRIVATE void multipolebc(
01110         double r,
01111         double kappa,
01112         double eps_p,
01113         double eps_w,
01114         double rad,
01115         double tsr[3]
01116         );
01117
01126 VPRIVATE double bcfl1sp(
01127         double size,
01128         double *apos,
01129         double charge,
01130         double xkappa,
01131         double pre1,
01132         double *pos
01133         );
01134
01139 VPRIVATE void bcCalc(
01140         Vpmg *thee
01141         );
01142
01147 VPRIVATE void fillcoCoef(
01148         Vpmg *thee
01149         );
01150
01155 VPRIVATE void fillcoCoefMap(
01156         Vpmg *thee
01157         );
01158
01164 VPRIVATE void fillcoCoefMol(
01165         Vpmg *thee
01166         );
01167
01173 VPRIVATE void fillcoCoefMolIon(
01174         Vpmg *thee
01175         );
```

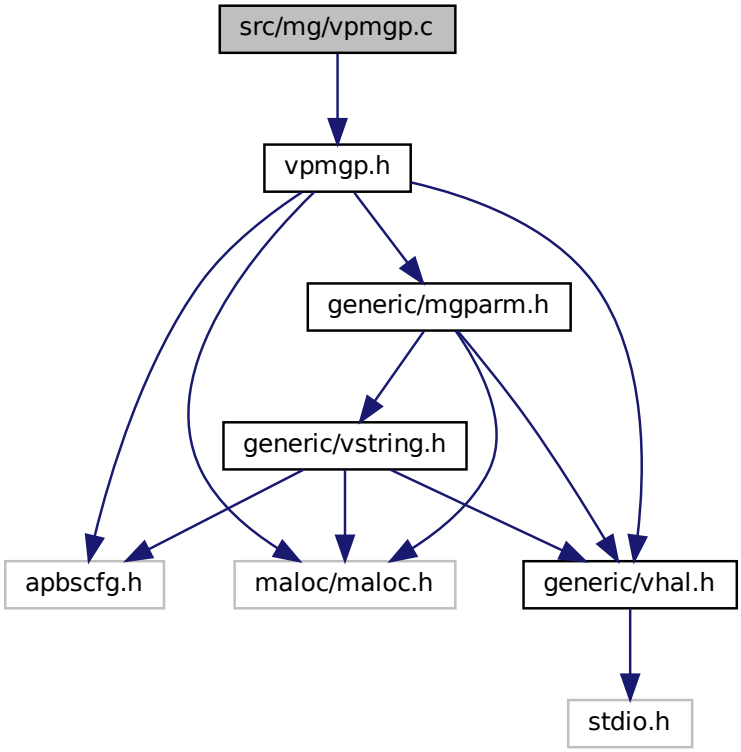
```
01176
01182 VPRIVATE void fillcoCoefMolDiel(
01183         Vpmg *thee
01184         );
01185
01191 VPRIVATE void fillcoCoefMolDielNoSmooth(
01192         Vpmg *thee
01193         );
01194
01208 VPRIVATE void fillcoCoefMolDielSmooth(
01209         Vpmg *thee
01210         );
01211
01217 VPRIVATE void fillcoCoefSpline(
01218         Vpmg *thee
01219         );
01220
01226 VPRIVATE void fillcoCoefSpline3(
01227         Vpmg *thee
01228         );
01229
01235 VPRIVATE void fillcoCoefSpline4(
01236         Vpmg *thee
01237         );
01238
01244 VPRIVATE Vrc_Codes fillcoCharge(
01245         Vpmg *thee
01246         );
01247
01253 VPRIVATE Vrc_Codes fillcoChargeMap(
01254         Vpmg *thee
01255         );
01256
01261 VPRIVATE void fillcoChargeSpline1(
01262         Vpmg *thee
01263         );
01264
01269 VPRIVATE void fillcoChargeSpline2(
01270         Vpmg *thee
01271         );
01272
01277 VPRIVATE void fillcoPermanentMultipole(
01278         Vpmg *thee
01279         );
01280
01285 VPRIVATE void fillcoInducedDipole(
01286         Vpmg *thee
01287         );
01288
01294 VPRIVATE void fillcoNLInducedDipole(
01295         Vpmg *thee
01296         );
01297
01304 VPRIVATE void extEnergy(
01305         Vpmg *thee,
01306         Vpmg *pmgOLD,
01307         PBEparm_calcEnergy extFlag,
01308         double partMin[3],
01309         double partMax[3],
01310         int bflags[6]
01311         );
01312
01317 VPRIVATE void qfForceSpline1(
01318         Vpmg *thee,
01319         double *force,
01320         int atomID
01321         );
01322
01327 VPRIVATE void qfForceSpline2(
01328         Vpmg *thee,
01329         double *force,
01330         int atomID
01331         );
01332
01337 VPRIVATE void qfForceSpline4(
01338                              Vpmg *thee,
01339                              double *force,
01340                              int atomID
01341                              );
01342
01343
```

```
01351 VPRIVATE void zlapSolve(
01352         Vpmg *thee,
01353         double **solution,
01354         double **source,
01355         double **work1
01356         );
01357
01364 VPRIVATE void markSphere(
01365         double rtot,
01366         double *tpos,
01367         int nx,
01368         int ny,
01369         int nz,
01370         double hx,
01371         double hy,
01372         double hzed,
01373         double xmin,
01374         double ymin,
01375         double zmin,
01376         double *array,
01377         double markVal
01378         );
01379
01384 VPRIVATE double Vpmg_qmEnergySMPBE(Vpmg *thee, int extFlag);
01385 VPRIVATE double Vpmg_qmEnergyNONLIN(Vpmg *thee, int extFlag);
01386
01387
01388
01389 // Additional macros and definitions.  May not be needed
01390
01391 // Added by Vincent Chu 9/13/06 for SMPB
01392 #define VCUB(x)             ((x)*(x)*(x))
01393 #define VLOG(x)             (log(x))
01394
01395 #define IJK(i,j,k)  (((k)*(nx)*(ny))+((j)*(nx))+(i))
01396 #define IJKx(j,k,i) (((i)*(ny)*(nz))+((k)*(ny))+(j))
01397 #define IJKy(i,k,j) (((j)*(nx)*(nz))+((k)*(nx))+(i))
01398 #define IJKz(i,j,k) (((k)*(nx)*(ny))+((j)*(nx))+(i))
01399 #define VFCHI(iint,iflt) (1.5+((double)(iint)-(iflt)))
01400
01401
01402 #endif    /* ifndef _VPMG_H_ */
01403
```

9.107 src/mg/vpmgp.c File Reference

Class Vpmgp methods.

```
#include "vpmgp.h"
```
Include dependency graph for vpmgp.c:



Functions

- VPUBLIC Vpmgp ∗ Vpmgp_ctor (MGparm ∗mgparm)

  Construct PMG parameter object and initialize to default values.
- VPUBLIC int Vpmgp_ctor2 (Vpmgp ∗thee, MGparm ∗mgparm)

  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.

9.107.1 Detailed Description

Class Vpmgp methods.

Author

Nathan Baker

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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) 2002-2020, 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.

9.108 vpmgp.c

```
00001
```

```
00057 #include "vpmgp.h"
00058
00059 VEMBED(rcsid="$Id$")
00060
00061 /* ////////////////////////////////////////////////////////////////////////
00062 // Class Vpmgp: Inlineable methods
00064 #if !defined(VINLINE_VACC)
00065 #endif /* if !defined(VINLINE_VACC) */
00066
00067 /* ////////////////////////////////////////////////////////////////////////
00068 // Class Vpmgp: Non-inlineable methods
00070
00071 /* ////////////////////////////////////////////////////////////////////////
00072 // Routine:  Vpmgp_ctor
00073 //
00074 // Author:   Nathan Baker
00076 VPUBLIC Vpmgp* Vpmgp_ctor(MGparm *mgparm) {
00077
00078     Vpmgp *thee = VNULL;
00079
00080     /* Set up the structure */
00081     thee = (Vpmgp*)Vmem_malloc(VNULL, 1, sizeof(Vpmgp) );
00082     VASSERT( thee != VNULL);
00083     VASSERT(Vpmgp_ctor2(thee,mgparm));
00084
00085     return thee;
00086 }
00087
00088 /* ////////////////////////////////////////////////////////////////////////
00089 // Routine:  Vpmgp_ctor2
00090 //
00091 // Author:   Nathan Baker
00093 VPUBLIC int Vpmgp_ctor2(Vpmgp *thee,MGparm *mgparm) {
00094
00095     /* Specified parameters */
00096     thee->nx = mgparm->dime[0];
00097     thee->ny = mgparm->dime[1];
00098     thee->nz = mgparm->dime[2];
00099     thee->hx = mgparm->grid[0];
00100     thee->hy = mgparm->grid[1];
00101     thee->hzed = mgparm->grid[2];
00102     thee->xlen = ((double)(mgparm->dime[0]-1))*mgparm->grid[0];
00103     thee->ylen = ((double)(mgparm->dime[1]-1))*mgparm->grid[1];
00104     thee->zlen = ((double)(mgparm->dime[2]-1))*mgparm->grid[2];
00105     thee->nlev = mgparm->nlev;
00106
00107     thee->nonlin = mgparm->nonlintype;
00108     thee->meth = mgparm->method;
00109
00110 #ifdef DEBUG_MAC_OSX_OCL
00111 #include "mach_chud.h"
00112     if(kOpenCLAvailable)
00113         thee->meth = 4;
00114 #endif
00115
00116     if (thee->nonlin == NONLIN_LPBE) thee->ipkey = IPKEY_LPBE; /* LPBE case */
00117     else if(thee->nonlin == NONLIN_SMPBE) thee->ipkey = IPKEY_SMPBE; /* SMPBE case */
00118     else thee->ipkey = IPKEY_NPBE; /* NPBE standard case */
00119
00120     /* Default parameters */
00121     if (mgparm->setetol) { /* If etol is set by the user in APBS input file, then use this custom-defined
      etol */
00122         thee->errtol = mgparm->etol;
00123         Vnm_print(1, "  Error tolerance (etol) is now set to user-defined \
00124 value: %g \n", thee->errtol);
00125         Vnm_print(0, "Error tolerance (etol) is now set to user-defined \
00126 value: %g \n", thee->errtol);
00127     } else thee->errtol = 1.0e-6;   /* Here are a few comments.  Mike had this set to
00128         * 1e-9; convential wisdom sets this at 1e-6 for
00129         * the PBE; Ray Luo sets this at 1e-3 for his
00130         * accelerated PBE (for dynamics, etc.) */
00131     thee->itmax = 200;
00132     thee->istop = 1;
00133     thee->iinfo = 1;         /* I'd recommend either 1 (for debugging LPBE) or 2 (for debugging NPBE),
      higher values give too much output */
00134
00135     thee->bcfl = BCFL_SDH;
00136     thee->key = 0;
00137     thee->iperf = 0;
00138     thee->mgcoar = 2;
00139     thee->mgkey = 0;
```

```
00140     thee->nu1 = 2;
00141     thee->nu2 = 2;
00142     thee->mgprol = 0;
00143     thee->mgdisc = 0;
00144     thee->omegal = 19.4e-1;
00145     thee->omegan = 9.0e-1;
00146     thee->ipcon = 3;
00147     thee->irite = 8;
00148     thee->xcent = 0.0;
00149     thee->ycent = 0.0;
00150     thee->zcent = 0.0;
00151
00152     /* Default value for all APBS runs */
00153     thee->mgsmoo = 1;
00154     if (thee->nonlin == NONLIN_NPBE || thee->nonlin == NONLIN_SMPBE) {
00155         /* SMPBE Added - SMPBE needs to mimic NPBE */
00156         Vnm_print(0, "Vpmp_ctor2:  Using meth = 1, mgsolv = 0\n");
00157         thee->mgsolv = 0;
00158     } else {
00159         /* Most rigorous (good for testing) */
00160         Vnm_print(0, "Vpmp_ctor2:  Using meth = 2, mgsolv = 1\n");
00161         thee->mgsolv = 1;
00162     }
00163
00164     /* TEMPORARY USEAQUA */
00165     /* If we are using aqua, our solution method is either VSOL_CGMGAqua or VSOL_NewtonAqua
00166      * so we need to temporarily override the mgsolve value and set it to 0
00167      */
00168     if(mgparm->useAqua == 1) thee->mgsolv = 0;
00169
00170     return 1;
00171 }
00172
00173 /* ///////////////////////////////////////////////////////////////////////////
00174 // Routine:  Vpmgp_dtor
00175 //
00176 // Author:   Nathan Baker
00178 VPUBLIC void Vpmgp_dtor(Vpmgp **thee) {
00179
00180     if ((*thee) != VNULL) {
00181         Vpmgp_dtor2(*thee);
00182         Vmem_free(VNULL, 1, sizeof(Vpmgp), (void **)thee);
00183         (*thee) = VNULL;
00184     }
00185
00186 }
00187
00188 /* ///////////////////////////////////////////////////////////////////////////
00189 // Routine:  Vpmgp_dtor2
00190 //
00191 // Author:   Nathan Baker
00193 VPUBLIC void Vpmgp_dtor2(Vpmgp *thee) { ; }
00194
00195
00196 VPUBLIC void Vpmgp_size(
00197     Vpmgp *thee
00198     )
00199 {
00200
00201     int num_nf = 0;
00202     int num_narr = 2;
00203     int num_narrc = 27;
00204     int nxf, nyf, nzf, level, num_nf_oper, num_narrc_oper, n_band, nc_band, num_band, iretot;
00205
00206     thee->nf = thee->nx * thee->ny * thee->nz;
00207     thee->narr = thee->nf;
00208     nxf = thee->nx;
00209     nyf = thee->ny;
00210     nzf = thee->nz;
00211     thee->nxc = thee->nx;
00212     thee->nyc = thee->ny;
00213     thee->nzc = thee->nz;
00214
00215     for (level=2; level<=thee->nlev; level++) {
00216         Vpmgp_makeCoarse(1, nxf, nyf, nzf, &(thee->nxc), &(thee->nyc), &(thee->nzc)); /* NAB TO-DO --
       implement this function and check which variables need to be passed by reference... */
00217         nxf = thee->nxc;
00218         nyf = thee->nyc;
00219         nzf = thee->nzc;
00220         thee->narr = thee->narr + (nxf * nyf * nzf);
00221     }
```

```
00222
00223     thee->nc = thee->nxc * thee->nyc * thee->nzc;
00224     thee->narrc = thee->narr - thee->nf;
00225
00226     /* Box or FEM discretization on fine grid? */
00227     switch (thee->mgdisc) { /* NAB TO-DO:  This needs to be changed into an enumeration */
00228     case 0:
00229         num_nf_oper = 4;
00230         break;
00231     case 1:
00232         num_nf_oper = 14;
00233         break;
00234     default:
00235         Vnm_print(2, "Vpmgp_size:  Invalid mgdisc value (%d)!\n", thee->mgdisc);
00236         VASSERT(0);
00237     }
00238
00239     /* Galerkin or standard coarsening? */
00240     switch (thee->mgcoar) { /* NAB TO-DO:  This needs to be changed into an enumeration */
00241     case 0:
00242         if (thee->mgdisc != 0) {
00243             Vnm_print(2, "Vpmgp_size:  Invalid mgcoar value (%d); must be used with mgdisc 0!\n",
      thee->mgcoar);
00244             VASSERT(0);
00245         }
00246         num_narrc_oper = 4;
00247         break;
00248     case 1:
00249         if (thee->mgdisc != 0) {
00250             Vnm_print(2, "Vpmgp_size:  Invalid mgcoar value (%d); must be used with mgdisc 0!\n",
      thee->mgcoar);
00251             VASSERT(0);
00252         }
00253         num_narrc_oper = 14;
00254         break;
00255     case 2:
00256         num_narrc_oper = 14;
00257         break;
00258     default:
00259         Vnm_print(2, "Vpmgp_size:  Invalid mgcoar value (%d)!\n", thee->mgcoar);
00260         VASSERT(0);
00261     }
00262
00263     /* LINPACK storage on coarse grid */
00264     switch (thee->mgsolv) { /* NAB TO-DO:  This needs to be changed into an enumeration */
00265     case 0:
00266         n_band = 0;
00267         break;
00268     case 1:
00269         if ( ( (thee->mgcoar == 0) || (thee->mgcoar == 1)) && (thee->mgdisc == 0) ) {
00270             num_band = 1 + (thee->nxc-2)*(thee->nyc-2);
00271         } else {
00272             num_band = 1 + (thee->nxc-2)*(thee->nyc-2) + (thee->nxc-2) + 1;
00273         }
00274         nc_band = (thee->nxc-2)*(thee->nyc-2)*(thee->nzc-2);
00275         n_band  = nc_band * num_band;
00276         break;
00277     default:
00278         Vnm_print(2, "Vpmgp_size:  Invalid mgsolv value (%d)!\n", thee->mgsolv);
00279         VASSERT(0);
00280     }
00281
00282     /* Real storage parameters */
00283     thee->n_rpc = 100*(thee->nlev+1);
00284
00285     /* Resulting total required for real storage */
00286     thee->nrwk = num_narr*thee->narr + (size_t)(num_nf + num_nf_oper)*thee->nf + (size_t)(num_narrc +
      num_narrc_oper)*thee->narrc + n_band + thee->n_rpc;
00287
00288     /* Integer storage parameters */
00289     thee->n_iz = 50*(thee->nlev+1);
00290     thee->n_ipc = 100*(thee->nlev+1);
00291     thee->niwk = thee->n_iz + thee->n_ipc;
00292 }
00293
00294 VPRIVATE int coarsenThis(int nOld) {
00295
00296     int nOut;
00297
00298     nOut = (nOld - 1) / 2 + 1;
00299
```

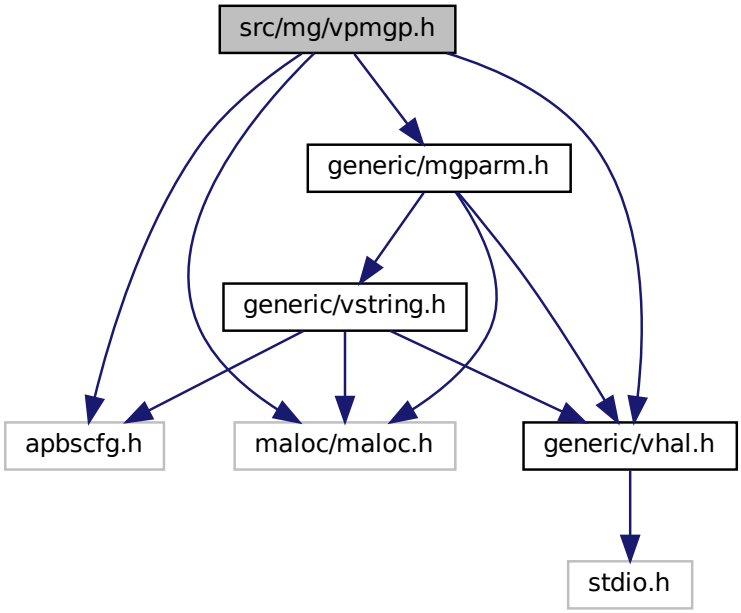
```
00300     if (((nOut-1)*2) != (nOld-1)) {
00301         Vnm_print(2, "Vpmgp_makeCoarse:  Warning!  The grid dimensions you have chosen are not consistent
      with the nlev you have specified!\n");
00302         Vnm_print(2, "Vpmgp_makeCoarse:  This calculation will only work if you are running with mg-dummy
      type.\n");
00303     }
00304     if (nOut < 1) {
00305         Vnm_print(2, "D'oh!  You coarsened the grid below zero!  How did you do that?\n");
00306         VASSERT(0);
00307     }
00308
00309     return nOut;
00310 }
00311
00312 VPUBLIC void Vpmgp_makeCoarse(
00313     int numLevel,
00314     int nxOld,
00315     int nyOld,
00316     int nzOld,
00317     int *nxNew,
00318     int *nyNew,
00319     int *nzNew
00320     )
00321 {
00322     int nxtmp, nytmp, nztmp, iLevel;
00323
00324     for (iLevel=0; iLevel<numLevel; iLevel++) {
00325         nxtmp = *nxNew;
00326         nytmp = *nyNew;
00327         nztmp = *nzNew;
00328         *nxNew = coarsenThis(nxtmp);
00329         *nyNew = coarsenThis(nytmp);
00330         *nzNew = coarsenThis(nztmp);
00331     }
00332
00333
00334 }
```

9.109 src/mg/vpmgp.h File Reference

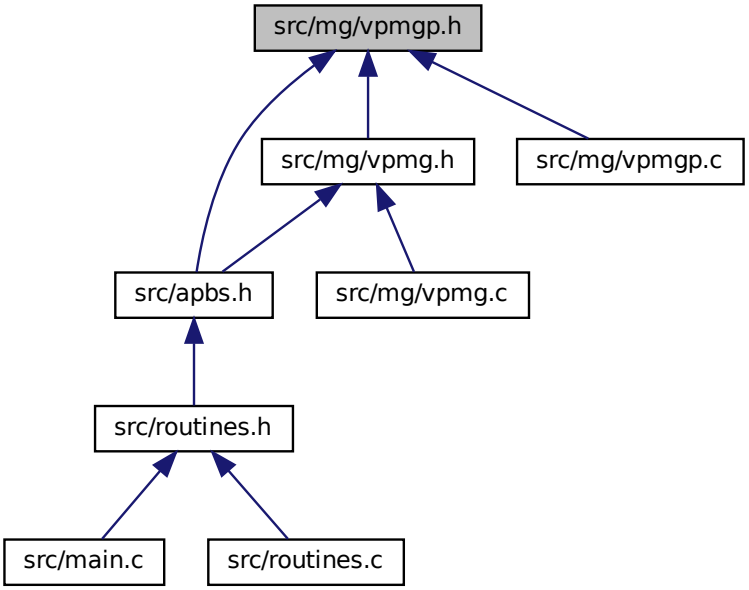
Contains declarations for class Vpmgp.

```
#include "apbscfg.h"
#include "maloc/maloc.h"
#include "generic/vhal.h"
#include "generic/mgparm.h"
```

Include dependency graph for vpmgp.h:



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



Data Structures

- struct sVpmgp

  Contains public data members for Vpmgp class/module.

Typedefs

- typedef struct sVpmgp Vpmgp

  Declaration of the Vpmgp class as the sVpmgp structure.

Functions

- VEXTERNC Vpmgp * Vpmgp_ctor (MGparm *mgparm)

  Construct PMG parameter object and initialize to default values.
- VEXTERNC int Vpmgp_ctor2 (Vpmgp *thee, MGparm *mgparm)

  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.

9.109.1 Detailed Description

Contains declarations for class Vpmgp.

Version

Id

Author

Nathan A. Baker

Note

Variables and many default values taken directly from PMG

Attention

Definition in file vpmgp.h.

9.110 vpmgp.h

```
00001
00064 #ifndef _VPMGP_H_
00065 #define _VPMGP_H_
00066
00067 #include "apbscfg.h"
00068
00069 #include "maloc/maloc.h"
00070
00071 #include "generic/vhal.h"
00072 #include "generic/mgparm.h"
00073
00080 struct sVpmgp {
00081
00082     /* ********** USER-SPECIFIED PARAMETERS ********** */
00083     int nx;
00084     int ny;
00085     int nz;
00086     int nlev;
00087     double hx;
00088     double hy;
00089     double hzed;
00090     int nonlin;
00095     /* ********** DERIVED PARAMETERS ********** */
00096     int nxc;
00097     int nyc;
00098     int nzc;
00099     int nf;
00100     int nc;
00101     int narrc;
00102     int n_rpc;
00103     int n_iz;
00104     int n_ipc;
00106     size_t nrwk;
00107     int niwk;
00108     int narr;
00109     int ipkey;
00117     /* ********** PARAMETERS WITH DEFAULT VALUES ********** */
00118     double xcent;
00119     double ycent;
00120     double zcent;
00121     double errtol;
00122     int itmax;
00123     int istop;
00130     int iinfo;
00135     Vbcfl bcfl;
00136     int key;
00139     int iperf;
00144     int meth;
00155     int mgkey;
00158     int nu1;
00159     int nu2;
00160     int mgsmoo;
00166     int mgprol;
00170     int mgcoar;
00174     int mgsolv;
00177     int mgdisc;
00180     double omegal;
00181     double omegan;
00182     int irite;
00183     int ipcon;
00189     double xlen;
00190     double ylen;
00191     double zlen;
00192     double xmin;
00193     double ymin;
00194     double zmin;
00195     double xmax;
00196     double ymax;
00197     double zmax;
00198 };
00199
00204 typedef struct sVpmgp Vpmgp;
00205
00206 /* ////////////////////////////////////////////////////////////////////////// */
00207 // Class Vpmgp: Inlineable methods (vpmgp.c)
00209
00210 #if !defined(VINLINE_VPMGP)
00211 #else /* if defined(VINLINE_VPMGP) */
```

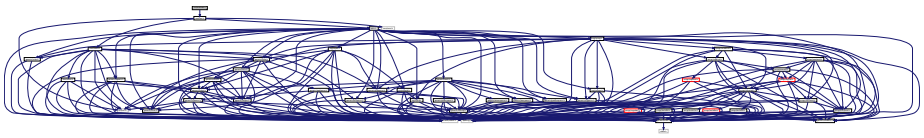
```
00212 #endif /* if !defined(VINLINE_VPMGP) */
00213
00214 /* ////////////////////////////////////////////////////////////////////////////
00215 // Class Vpmgp: Non-Inlineable methods (vpmgp.c)
00217
00224 VEXTERNC Vpmgp* Vpmgp_ctor(MGparm *mgparm);
00225
00234 VEXTERNC int Vpmgp_ctor2(Vpmgp *thee, MGparm *mgparm);
00235
00241 VEXTERNC void Vpmgp_dtor(Vpmgp **thee);
00242
00248 VEXTERNC void Vpmgp_dtor2(Vpmgp *thee);
00249
00254 VEXTERNC void Vpmgp_size(
00255     Vpmgp *thee /**< Object to be sized */
00256     );
00257
00262 VEXTERNC void Vpmgp_makeCoarse(
00263     int numLevel,
00264     int nxOld,
00265     int nyOld,
00266     int nzOld,
00267     int *nxNew,
00268     int *nyNew,
00269     int *nzNew
00270     );
00271
00272
00273
00274 #endif    /* ifndef _VPMGP_H_ */
```

9.111 src/routines.c File Reference

Supporting routines for APBS front end.

`#include "routines.h"`

Include dependency graph for routines.c:



Functions

- VPUBLIC void startVio ()

  Wrapper to start MALOC Vio layer.

- VPUBLIC Vparam * loadParameter (NOsh *nosh)

  Loads and returns parameter object.

- VPUBLIC int loadMolecules (NOsh *nosh, Vparam *param, Valist *alist[NOSH_MAXMOL])

  Load the molecules given in NOsh into atom lists.

- VPUBLIC void killMolecules (NOsh *nosh, Valist *alist[NOSH_MAXMOL])

  Destroy the loaded molecules.

- VPUBLIC int loadDielMaps (NOsh *nosh, Vgrid *dielXMap[NOSH_MAXMOL], Vgrid *dielYMap[NOSH_MAXMOL], Vgrid *dielZMap[NOSH_MAXMOL])

  Load the dielectric maps given in NOsh into grid objects.

- VPUBLIC void killDielMaps (NOsh *nosh, Vgrid *dielXMap[NOSH_MAXMOL], Vgrid *dielYMap[NOSH_MAXMOL], Vgrid *dielZMap[NOSH_MAXMOL])

  Destroy the loaded dielectric.

- VPUBLIC int loadKappaMaps (NOsh *nosh, Vgrid *map[NOSH_MAXMOL])

  Load the kappa maps given in NOsh into grid objects.

- VPUBLIC void killKappaMaps (NOsh *nosh, Vgrid *map[NOSH_MAXMOL])

  Destroy the loaded kappa maps.
- VPUBLIC int loadPotMaps (NOsh *nosh, Vgrid *map[NOSH_MAXMOL])

  Load the potential maps given in NOsh into grid objects.
- VPUBLIC void killPotMaps (NOsh *nosh, Vgrid *map[NOSH_MAXMOL])

  Destroy the loaded potential maps.
- VPUBLIC int loadChargeMaps (NOsh *nosh, Vgrid *map[NOSH_MAXMOL])

  Load the charge maps given in NOsh into grid objects.
- VPUBLIC void killChargeMaps (NOsh *nosh, Vgrid *map[NOSH_MAXMOL])

  Destroy the loaded charge maps.
- VPUBLIC void printPBEPARM (PBEparm *pbeparm)

  Print out generic PBE params loaded from input.
- VPUBLIC void printMGPARM (MGparm *mgparm, double realCenter[3])

  Print out MG-specific params loaded from input.
- VPUBLIC int initMG (int icalc, NOsh *nosh, MGparm *mgparm, PBEparm *pbeparm, double realCenter[3], Vpbe *pbe[NOSH_MAXCALC], Valist *alist[NOSH_MAXMOL], Vgrid *dielXMap[NOSH_MAXMOL], Vgrid *dielYMap[NOSH_MAXMOL], Vgrid *dielZMap[NOSH_MAXMOL], Vgrid *kappaMap[NOSH_MAXMOL], Vgrid *chargeMap[NOSH_MAXMOL], Vpmgp *pmgp[NOSH_MAXCALC], Vpmg *pmg[NOSH_MAXCALC], Vgrid *potMap[NOSH_MAXMOL])

  Initialize an MG calculation.
- VPUBLIC void killMG (NOsh *nosh, Vpbe *pbe[NOSH_MAXCALC], Vpmgp *pmgp[NOSH_MAXCALC], Vpmg *pmg[NOSH_MAXCALC])

  Kill structures initialized during an MG calculation.
- VPUBLIC int solveMG (NOsh *nosh, Vpmg *pmg, MGparm_CalcType type)

  Solve the PBE with MG.
- VPUBLIC int setPartMG (NOsh *nosh, MGparm *mgparm, Vpmg *pmg)

  Set MG partitions for calculating observables and performing I/O.
- VPUBLIC int energyMG (NOsh *nosh, int icalc, Vpmg *pmg, int *nenergy, double *totEnergy, double *qfEnergy, double *qmEnergy, double *dielEnergy)

  Calculate electrostatic energies from MG solution.
- VPUBLIC int forceMG (Vmem *mem, NOsh *nosh, PBEparm *pbeparm, MGparm *mgparm, Vpmg *pmg, int *nforce, AtomForce **atomForce, Valist *alist[NOSH_MAXMOL])

  Calculate forces from MG solution.
- VPUBLIC void killEnergy ()

  Kill arrays allocated for energies.
- VPUBLIC void killForce (Vmem *mem, NOsh *nosh, int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC])

  Free memory from MG force calculation.
- VPUBLIC int writematMG (int rank, NOsh *nosh, PBEparm *pbeparm, Vpmg *pmg)

  Write out operator matrix from MG calculation to file.
- VPUBLIC void storeAtomEnergy (Vpmg *pmg, int icalc, double **atomEnergy, int *nenergy)

  Store energy in arrays for future use.
- VPUBLIC int writedataFlat (NOsh *nosh, Vcom *com, const char *fname, double totEnergy[NOSH_MAXCALC], double qfEnergy[NOSH_MAXCALC], double qmEnergy[NOSH_MAXCALC], double dielEnergy[NOSH_MAXCALC], int nenergy[NOSH_MAXCALC], double *atomEnergy[NOSH_MAXCALC], int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC])

  Write out information to a flat file.

- VPUBLIC int writedataXML (NOsh *nosh, Vcom *com, const char *fname, double totEnergy[NOSH_MAXCALC], double qfEnergy[NOSH_MAXCALC], double qmEnergy[NOSH_MAXCALC], double dielEnergy[NOSH_MAXCALC], int nenergy[NOSH_MAXCALC], double *atomEnergy[NOSH_MAXCALC], int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC])

  Write out information to an XML file.
- VPUBLIC int writedataMG (int rank, NOsh *nosh, PBEparm *pbeparm, Vpmg *pmg)

  Write out observables from MG calculation to file.
- VPUBLIC double returnEnergy (Vcom *com, NOsh *nosh, double totEnergy[NOSH_MAXCALC], int iprint)

  Access net local energy.
- VPUBLIC int printEnergy (Vcom *com, NOsh *nosh, double totEnergy[NOSH_MAXCALC], int iprint)

  Combine and pretty-print energy data (deprecated...see printElecEnergy)
- VPUBLIC int printElecEnergy (Vcom *com, NOsh *nosh, double totEnergy[NOSH_MAXCALC], int iprint)

  Combine and pretty-print energy data.
- VPUBLIC int printApolEnergy (NOsh *nosh, int iprint)

  Combine and pretty-print energy data.
- VPUBLIC int printForce (Vcom *com, NOsh *nosh, int nforce[NOSH_MAXCALC], AtomForce *atom↩ Force[NOSH_MAXCALC], int iprint)

  Combine and pretty-print force data (deprecated...see printElecForce)
- VPUBLIC int printElecForce (Vcom *com, NOsh *nosh, int nforce[NOSH_MAXCALC], AtomForce *atom↩ Force[NOSH_MAXCALC], int iprint)

  Combine and pretty-print force data.
- VPUBLIC int printApolForce (Vcom *com, NOsh *nosh, int nforce[NOSH_MAXCALC], AtomForce *atom↩ Force[NOSH_MAXCALC], int iprint)

  Combine and pretty-print force data.
- VPUBLIC void killFE (NOsh *nosh, Vpbe *pbe[NOSH_MAXCALC], Vfetk *fetk[NOSH_MAXCALC], Gem *gm[NOSH_MAXMOL])

  Kill structures initialized during an FE calculation.
- VPUBLIC Vrc_Codes initFE (int icalc, NOsh *nosh, FEMparm *feparm, PBEparm *pbeparm, Vpbe *pbe[NOSH_MAXCALC], Valist *alist[NOSH_MAXMOL], Vfetk *fetk[NOSH_MAXCALC])

  Initialize FE solver objects.
- VPUBLIC void printFEPARM (int icalc, NOsh *nosh, FEMparm *feparm, Vfetk *fetk[NOSH_MAXCALC])

  Print out FE-specific params loaded from input.
- VPUBLIC int partFE (int icalc, NOsh *nosh, FEMparm *feparm, Vfetk *fetk[NOSH_MAXCALC])

  Partition mesh (if applicable)
- VPUBLIC int preRefineFE (int icalc, FEMparm *feparm, Vfetk *fetk[NOSH_MAXCALC])

  Pre-refine mesh before solve.
- VPUBLIC int solveFE (int icalc, PBEparm *pbeparm, FEMparm *feparm, Vfetk *fetk[NOSH_MAXCALC])

  Solve-estimate-refine.
- VPUBLIC int energyFE (NOsh *nosh, int icalc, Vfetk *fetk[NOSH_MAXCALC], int *nenergy, double *totEnergy, double *qfEnergy, double *qmEnergy, double *dielEnergy)

  Calculate electrostatic energies from FE solution.
- VPUBLIC int postRefineFE (int icalc, FEMparm *feparm, Vfetk *fetk[NOSH_MAXCALC])

  Estimate error, mark mesh, and refine mesh after solve.
- VPUBLIC int writedataFE (int rank, NOsh *nosh, PBEparm *pbeparm, Vfetk *fetk)

  Write FEM data to files.
- VPUBLIC int initAPOL (NOsh *nosh, Vmem *mem, Vparam *param, APOLparm *apolparm, int *nforce, AtomForce **atomForce, Valist *alist)

  Upperlevel routine to the non-polar energy and force routines.

- VPUBLIC int energyAPOL (APOLparm *apolparm, double sasa, double sav, double atomsasa[ ], double atomwcaEnergy[ ], int numatoms)

  Calculate non-polar energies.
- VPUBLIC int forceAPOL (Vacc *acc, Vmem *mem, APOLparm *apolparm, int *nforce, AtomForce **atomForce, Valist *alist, Vclist *clist)

  Calculate non-polar forces.

9.111.1 Detailed Description

Supporting routines for APBS front end.

Author

Nathan Baker

Version

Id

Attention

```
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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) 2002-2020, Nathan A. Baker.
* Portions Copyright (c) 1999-2002, The Regents of the University of
* California.
* Portions Copyright (c) 1995, Michael Holst.
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*
* Redistributions of source code must retain the above copyright notice, this
* list of conditions and the following disclaimer.
*
* Redistributions in binary form must reproduce the above copyright notice,
* this list of conditions and the following disclaimer in the documentation
* and/or other materials provided with the distribution.
*
* Neither the name of the developer nor the names of its contributors may be
* used to endorse or promote products derived from this software without
* specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
```

Definition in file routines.c.

9.112 routines.c

```
00001
00054 #include "routines.h"
00055
00056 VEMBED(rcsid="$Id$")
00057
00058 VPUBLIC void startVio() { Vio_start(); }
00059
00060 VPUBLIC Vparam* loadParameter(NOsh *nosh) {
00061
00062     Vparam *param = VNULL;
00063
00064     if (nosh->gotparm) {
00065         param = Vparam_ctor();
00066         switch (nosh->parmfmt) {
00067             case NPF_FLAT:
00068                 Vnm_tprint( 1, "Reading parameter data from %s.\n",
00069                             nosh->parmpath);
00070                 if (Vparam_readFlatFile(param, "FILE", "ASC", VNULL,
00071                                         nosh->parmpath) != 1) {
00072                     Vnm_tprint(2, "Error reading parameter file (%s)!\n", nosh->parmpath);
00073                     return VNULL;
00074                 }
00075                     break;
00076             case NPF_XML:
00077                 Vnm_tprint( 1, "Reading parameter data from %s.\n",
00078                             nosh->parmpath);
00079                 if (Vparam_readXMLFile(param, "FILE", "ASC", VNULL,
00080                                        nosh->parmpath) != 1) {
00081                     Vnm_tprint(2, "Error reading parameter file (%s)!\n", nosh->parmpath);
00082                     return VNULL;
00083                 }
00084                     break;
00085             default:
00086                 Vnm_tprint(2, "Error! Undefined parameter file type (%d)!\n", nosh->parmfmt);
00087                 return VNULL;
00088         } /* switch parmfmt */
00089     }
00090
00091     return param;
00092 }
00093
00094
00095 VPUBLIC int loadMolecules(NOsh *nosh, Vparam *param, Valist *alist[NOSH_MAXMOL]) {
00096
00097     int i;
00098     int use_params = 0;
00099     Vrc_Codes rc;
00100
00101     Vio *sock = VNULL;
00102
00103     Vnm_tprint( 1, "Got paths for %d molecules\n", nosh->nmol);
00104     if (nosh->nmol <= 0) {
00105         Vnm_tprint(2, "You didn't specify any molecules (correctly)!\n");
00106         Vnm_tprint(2, "Bailing out!\n");
00107         return 0;
00108     }
00109
00110     if (nosh->gotparm) {
00111         if (param == VNULL) {
00112             Vnm_tprint(2, "Error!  You don't have a valid parameter object!\n");
00113             return 0;
00114         }
00115         use_params = 1;
00116     }
00117
00118     for (i=0; i<nosh->nmol; i++) {
00119         if(alist[i] == VNULL){
00120             alist[i] = Valist_ctor();
00121         }else{
00122             alist[i] = VNULL;
00123             alist[i] = Valist_ctor();
00124         }
00125
00126         switch (nosh->molfmt[i]) {
00127             case NMF_PQR:
00128                     /* Print out a warning to the user letting them know that we are overriding PQR
00129                     values for charge, radius and epsilon */
00130                     if (use_params) {
```

```
00131                       Vnm_print(2, "\nWARNING!!  Radius/charge information from PQR file %s\n",
      nosh->molpath[i]);
00132                       Vnm_print(2, "will be replaced with data from parameter file (%s)!\n",
      nosh->parmpath);
00133                   }
00134                   Vnm_tprint( 1, "Reading PQR-format atom data from %s.\n",
00135                           nosh->molpath[i]);
00136                   sock = Vio_ctor("FILE", "ASC", VNULL, nosh->molpath[i], "r");
00137                   if (sock == VNULL) {
00138                       Vnm_print(2, "Problem opening virtual socket %s!\n",
00139                             nosh->molpath[i]);
00140                       return 0;
00141                   }
00142                   if (Vio_accept(sock, 0) < 0) {
00143                       Vnm_print(2, "Problem accepting virtual socket %s!\n",
00144                                 nosh->molpath[i]);
00145                       return 0;
00146                   }
00147                   if(use_params){
00148                       rc = Valist_readPQR(alist[i], param, sock);
00149                   }else{
00150                       rc = Valist_readPQR(alist[i], VNULL, sock);
00151                   }
00152                   if(rc == 0) return 0;
00153
00154                   Vio_acceptFree(sock);
00155                   Vio_dtor(&sock);
00156                   break;
00157           case NMF_PDB:
00158                   /* Load parameters */
00159                   if (!nosh->gotparm) {
00160                       Vnm_tprint(2, "NOsh:  Error!  Can't read PDB without specifying PARM file!\n");
00161                       return 0;
00162                   }
00163                   Vnm_tprint( 1, "Reading PDB-format atom data from %s.\n",
00164                           nosh->molpath[i]);
00165                   sock = Vio_ctor("FILE", "ASC", VNULL, nosh->molpath[i], "r");
00166                   if (sock == VNULL) {
00167                       Vnm_print(2, "Problem opening virtual socket %s!\n",
00168                             nosh->molpath[i]);
00169                       return 0;
00170                   }
00171                   if (Vio_accept(sock, 0) < 0) {
00172                       Vnm_print(2, "Problem accepting virtual socket %s!\n", nosh->molpath[i]);
00173                       return 0;
00174                   }
00175                   rc = Valist_readPDB(alist[i], param, sock);
00176                   /* If we are looking for an atom/residue that does not exist
00177                    * then abort and return 0 */
00178                   if(rc == 0)
00179                       return 0;
00180
00181                   Vio_acceptFree(sock);
00182                   Vio_dtor(&sock);
00183                   break;
00184           case NMF_XML:
00185               Vnm_tprint( 1, "Reading XML-format atom data from %s.\n",
00186                           nosh->molpath[i]);
00187               sock = Vio_ctor("FILE", "ASC", VNULL, nosh->molpath[i], "r");
00188               if (sock == VNULL) {
00189                   Vnm_print(2, "Problem opening virtual socket %s!\n",
00190                             nosh->molpath[i]);
00191                   return 0;
00192               }
00193                   if (Vio_accept(sock, 0) < 0) {
00194                       Vnm_print(2, "Problem accepting virtual socket %s!\n",
00195                                 nosh->molpath[i]);
00196                       return 0;
00197                   }
00198                   if(use_params){
00199                       rc = Valist_readXML(alist[i], param, sock);
00200                   }else{
00201                       rc = Valist_readXML(alist[i], VNULL, sock);
00202                   }
00203                   if(rc == 0)
00204                       return 0;
00205
00206               Vio_acceptFree(sock);
00207               Vio_dtor(&sock);
00208               break;
00209           default:
```

```
00210                 Vnm_tprint(2, "NOsh:  Error!  Undefined molecule file type \
00211 (%d)!\n", nosh->molfmt[i]);
00212                 return 0;
00213         } /* switch molfmt */
00214
00215         if (rc != 1) {
00216             Vnm_tprint( 2, "Error while reading molecule from %s\n",
00217                         nosh->molpath[i]);
00218             return 0;
00219         }
00220
00221         Vnm_tprint( 1, "  %d atoms\n", Valist_getNumberAtoms(alist[i]));
00222         Vnm_tprint( 1, "  Centered at (%4.3e, %4.3e, %4.3e)\n",
00223                     alist[i]->center[0], alist[i]->center[1],
00224                     alist[i]->center[2]);
00225         Vnm_tprint( 1, "  Net charge %3.2e e\n", alist[i]->charge);
00226
00227     }
00228
00229     return 1;
00230
00231 }
00232
00233 VPUBLIC void killMolecules(NOsh *nosh, Valist *alist[NOSH_MAXMOL]) {
00234
00235     int i;
00236
00237 #ifndef VAPBSQUIET
00238     Vnm_tprint( 1, "Destroying %d molecules\n", nosh->nmol);
00239 #endif
00240
00241     for (i=0; i<nosh->nmol; i++)
00242         Valist_dtor(&(alist[i]));
00243
00244 }
00245
00250 VPUBLIC int loadDielMaps(NOsh *nosh,Vgrid *dielXMap[NOSH_MAXMOL], Vgrid *dielYMap[NOSH_MAXMOL],Vgrid
      *dielZMap[NOSH_MAXMOL]){
00251
00252     int i,ii,nx,ny,nz;
00253     double sum,hx,hy,hzed,xmin,ymin,zmin;
00254
00255     // Check to be sure we have dieletric map paths; if not, return.
00256     if (nosh->ndiel > 0)
00257         Vnm_tprint( 1, "Got paths for %d dielectric map sets\n",nosh->ndiel);
00258     else
00259         return 1;
00260
00261     // For each dielectric map path, read the data and calculate needed values.
00262     for (i=0; i<nosh->ndiel; i++) {
00263         Vnm_tprint( 1, "Reading x-shifted dielectric map data from %s:\n", nosh->dielXpath[i]);
00264         dielXMap[i] = Vgrid_ctor(0, 0, 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, VNULL);
00265
00266         // Determine the format and read data if the format is valid.
00267         switch (nosh->dielfmt[i]) {
00268             // OpenDX (Data Explorer) format
00269             case VDF_DX:
00270                 if (Vgrid_readDX(dielXMap[i], "FILE", "ASC", VNULL,
00271                                  nosh->dielXpath[i]) != 1) {
00272                     Vnm_tprint( 2, "Fatal error while reading from %s\n",
00273                                 nosh->dielXpath[i]);
00274                     return 0;
00275                 }
00276
00277                 // Set grid sizes
00278                 nx = dielXMap[i]->nx;
00279                 ny = dielXMap[i]->ny;
00280                 nz = dielXMap[i]->nz;
00281
00282                 // Set spacings
00283                 hx = dielXMap[i]->hx;
00284                 hy = dielXMap[i]->hy;
00285                 hzed = dielXMap[i]->hzed;
00286
00287                 // Set minimum lower corner
00288                 xmin = dielXMap[i]->xmin;
00289                 ymin = dielXMap[i]->ymin;
00290                 zmin = dielXMap[i]->zmin;
00291                 Vnm_tprint(1, "  %d x %d x %d grid\n", nx, ny, nz);
00292                 Vnm_tprint(1, "  (%g, %g, %g) A spacings\n", hx, hy, hzed);
00293                 Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
```

```
00294                           xmin, ymin, zmin);
00295                 sum = 0;
00296                 for (ii=0; ii<(nx*ny*nz); ii++)
00297                     sum += (dielXMap[i]->data[ii]);
00298                     sum = sum*hx*hy*hzed;
00299                 Vnm_tprint(1, "  Volume integral = %3.2e A^3\n", sum);
00300                 break;
00301 
00302             //DX binary file (.dxbin)
00303             case VDF_DXBIN:
00304               //TODO: add this method and maybe change the if stmt.
00305               if (Vgrid_readDXBIN(dielXMap[i], "FILE", "ASC", VNULL,
00306                                                 nosh->dielXpath[i]) != 1) {
00307                                 Vnm_tprint( 2, "Fatal error while reading from %s\n",
00308                                                nosh->dielXpath[i]);
00309                                 return 0;
00310               }
00311 
00312               // Set grid sizes
00313         nx = dielXMap[i]->nx;
00314         ny = dielXMap[i]->ny;
00315         nz = dielXMap[i]->nz;
00316 
00317         // Set spacings
00318         hx = dielXMap[i]->hx;
00319         hy = dielXMap[i]->hy;
00320         hzed = dielXMap[i]->hzed;
00321 
00322         // Set minimum lower corner
00323         xmin = dielXMap[i]->xmin;
00324         ymin = dielXMap[i]->ymin;
00325         zmin = dielXMap[i]->zmin;
00326         Vnm_tprint(1, "  %d x %d x %d grid\n", nx, ny, nz);
00327         Vnm_tprint(1, "  (%g, %g, %g) A spacings\n", hx, hy, hzed);
00328         Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
00329                xmin, ymin, zmin);
00330         sum = 0;
00331         for (ii=0; ii<(nx*ny*nz); ii++)
00332           sum += (dielXMap[i]->data[ii]);
00333           sum = sum*hx*hy*hzed;
00334         Vnm_tprint(1, "  Volume integral = %3.2e A^3\n", sum);
00335         break;
00336 
00337             // Binary file (GZip)
00338             case VDF_GZ:
00339                 if (Vgrid_readGZ(dielXMap[i], nosh->dielXpath[i]) != 1) {
00340                     Vnm_tprint( 2, "Fatal error while reading from %s\n",
00341                                nosh->dielXpath[i]);
00342                     return 0;
00343                 }
00344 
00345                 // Set grid sizes
00346                 nx = dielXMap[i]->nx;
00347                 ny = dielXMap[i]->ny;
00348                 nz = dielXMap[i]->nz;
00349 
00350                 // Set spacings
00351                 hx = dielXMap[i]->hx;
00352                 hy = dielXMap[i]->hy;
00353                 hzed = dielXMap[i]->hzed;
00354 
00355                 // Set minimum lower corner
00356                 xmin = dielXMap[i]->xmin;
00357                 ymin = dielXMap[i]->ymin;
00358                 zmin = dielXMap[i]->zmin;
00359                 Vnm_tprint(1, "  %d x %d x %d grid\n", nx, ny, nz);
00360                 Vnm_tprint(1, "  (%g, %g, %g) A spacings\n", hx, hy, hzed);
00361                 Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
00362                            xmin, ymin, zmin);
00363                 sum = 0;
00364                 for (ii=0; ii<(nx*ny*nz); ii++)
00365                     sum += (dielXMap[i]->data[ii]);
00366                 sum = sum*hx*hy*hzed;
00367                 Vnm_tprint(1, "  Volume integral = %3.2e A^3\n", sum);
00368                 break;
00369             // UHBD format
00370             case VDF_UHBD:
00371                 Vnm_tprint( 2, "UHBD input not supported yet!\n");
00372                 return 0;
00373             // AVS UCD format
00374             case VDF_AVS:
```

```
00375                 Vnm_tprint( 2, "AVS input not supported yet!\n");
00376                 return 0;
00377             // FEtk MC Simplex Format (MCSF)
00378             case VDF_MCSF:
00379                 Vnm_tprint( 2, "MCSF input not supported yet!\n");
00380                 return 0;
00381             default:
00382                 Vnm_tprint( 2, "Invalid data format (%d)!\n",
00383                             nosh->dielfmt[i]);
00384                 return 0;
00385         }
00386         Vnm_tprint( 1, "Reading y-shifted dielectric map data from \
00387 %s:\n", nosh->dielYpath[i]);
00388         dielYMap[i] = Vgrid_ctor(0, 0, 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, VNULL);
00389
00390         // Determine the format and read data if the format is valid.
00391         switch (nosh->dielfmt[i]) {
00392             // OpenDX (Data Explorer) format
00393             case VDF_DX:
00394                 if (Vgrid_readDX(dielYMap[i], "FILE", "ASC", VNULL,
00395                                  nosh->dielYpath[i]) != 1) {
00396                     Vnm_tprint( 2, "Fatal error while reading from %s\n",
00397                                 nosh->dielYpath[i]);
00398                     return 0;
00399                 }
00400
00401                 // Read grid
00402                 nx = dielYMap[i]->nx;
00403                 ny = dielYMap[i]->ny;
00404                 nz = dielYMap[i]->nz;
00405
00406                 // Read spacings
00407                 hx = dielYMap[i]->hx;
00408                 hy = dielYMap[i]->hy;
00409                 hzed = dielYMap[i]->hzed;
00410
00411                 // Read minimum lower corner
00412                 xmin = dielYMap[i]->xmin;
00413                 ymin = dielYMap[i]->ymin;
00414                 zmin = dielYMap[i]->zmin;
00415                 Vnm_tprint(1, "  %d x %d x %d grid\n", nx, ny, nz);
00416                 Vnm_tprint(1, "  (%g, %g, %g) A spacings\n", hx, hy, hzed);
00417                 Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
00418                            xmin, ymin, zmin);
00419                 sum = 0;
00420                 for (ii=0; ii<(nx*ny*nz); ii++)
00421                     sum += (dielYMap[i]->data[ii]);
00422                     sum = sum*hx*hy*hzed;
00423                 Vnm_tprint(1, "  Volume integral = %3.2e A^3\n", sum);
00424                 break;
00425             //DX Binary file (.dxbin)
00426         case VDF_DXBIN:
00427           //TODO: add this funct/method and maybe change the if stmt.
00428           if (Vgrid_readDXBIN(dielYMap[i], "FILE", "ASC", VNULL,
00429                    nosh->dielYpath[i]) != 1) {
00430             Vnm_tprint( 2, "Fatal error while reading from %s\n",
00431                    nosh->dielYpath[i]);
00432             return 0;
00433           }
00434
00435           // Read grid
00436           nx = dielYMap[i]->nx;
00437           ny = dielYMap[i]->ny;
00438           nz = dielYMap[i]->nz;
00439
00440           // Read spacings
00441           hx = dielYMap[i]->hx;
00442           hy = dielYMap[i]->hy;
00443           hzed = dielYMap[i]->hzed;
00444
00445           // Read minimum lower corner
00446           xmin = dielYMap[i]->xmin;
00447           ymin = dielYMap[i]->ymin;
00448           zmin = dielYMap[i]->zmin;
00449           Vnm_tprint(1, "  %d x %d x %d grid\n", nx, ny, nz);
00450           Vnm_tprint(1, "  (%g, %g, %g) A spacings\n", hx, hy, hzed);
00451           Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
00452                  xmin, ymin, zmin);
00453           sum = 0;
00454           for (ii=0; ii<(nx*ny*nz); ii++)
00455             sum += (dielYMap[i]->data[ii]);
```

```
00456           sum = sum*hx*hy*hzed;
00457         Vnm_tprint(1, "  Volume integral = %3.2e A^3\n", sum);
00458         break;
00459             // Binary file (GZip) format
00460             case VDF_GZ:
00461                 if (Vgrid_readGZ(dielYMap[i], nosh->dielYpath[i]) != 1) {
00462                     Vnm_tprint( 2, "Fatal error while reading from %s\n",
00463                                 nosh->dielYpath[i]);
00464                     return 0;
00465                 }
00466
00467                 // Read grid
00468                 nx = dielYMap[i]->nx;
00469                 ny = dielYMap[i]->ny;
00470                 nz = dielYMap[i]->nz;
00471
00472                 // Read spacings
00473                 hx = dielYMap[i]->hx;
00474                 hy = dielYMap[i]->hy;
00475                 hzed = dielYMap[i]->hzed;
00476
00477                 // Read minimum lower corner
00478                 xmin = dielYMap[i]->xmin;
00479                 ymin = dielYMap[i]->ymin;
00480                 zmin = dielYMap[i]->zmin;
00481                 Vnm_tprint(1, "  %d x %d x %d grid\n", nx, ny, nz);
00482                 Vnm_tprint(1, "  (%g, %g, %g) A spacings\n", hx, hy, hzed);
00483                 Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
00484                            xmin, ymin, zmin);
00485                 sum = 0;
00486                 for (ii=0; ii<(nx*ny*nz); ii++)
00487                     sum += (dielYMap[i]->data[ii]);
00488                 sum = sum*hx*hy*hzed;
00489                 Vnm_tprint(1, "  Volume integral = %3.2e A^3\n", sum);
00490                 break;
00491             // UHBD format
00492             case VDF_UHBD:
00493                 Vnm_tprint( 2, "UHBD input not supported yet!\n");
00494                 return 0;
00495             // AVS UCD format
00496             case VDF_AVS:
00497                 Vnm_tprint( 2, "AVS input not supported yet!\n");
00498                 return 0;
00499             // FEtk MC Simplex Format (MCSF)
00500             case VDF_MCSF:
00501                 Vnm_tprint( 2, "MCSF input not supported yet!\n");
00502                 return 0;
00503             default:
00504                 Vnm_tprint( 2, "Invalid data format (%d)!\n",
00505                             nosh->dielfmt[i]);
00506                 return 0;
00507         }
00508
00509         Vnm_tprint( 1, "Reading z-shifted dielectric map data from \
00510 %s:\n", nosh->dielZpath[i]);
00511         dielZMap[i] = Vgrid_ctor(0, 0, 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, VNULL);
00512
00513         // Determine the format and read data if the format is valid.
00514         switch (nosh->dielfmt[i]) {
00515             // OpenDX (Data Explorer) format
00516             case VDF_DX:
00517                 if (Vgrid_readDX(dielZMap[i], "FILE", "ASC", VNULL,
00518                                  nosh->dielZpath[i]) != 1) {
00519                     Vnm_tprint( 2, "Fatal error while reading from %s\n",
00520                                 nosh->dielZpath[i]);
00521                     return 0;
00522                 }
00523
00524                 // Read grid
00525                 nx = dielZMap[i]->nx;
00526                 ny = dielZMap[i]->ny;
00527                 nz = dielZMap[i]->nz;
00528
00529                 // Read spacings
00530                 hx = dielZMap[i]->hx;
00531                 hy = dielZMap[i]->hy;
00532                 hzed = dielZMap[i]->hzed;
00533
00534                 // Read minimum lower corner
00535                 xmin = dielZMap[i]->xmin;
00536                 ymin = dielZMap[i]->ymin;
```

```
00537                 zmin = dielZMap[i]->zmin;
00538                 Vnm_tprint(1, "  %d x %d x %d grid\n",
00539                            nx, ny, nz);
00540                 Vnm_tprint(1, "  (%g, %g, %g) A spacings\n",
00541                            hx, hy, hzed);
00542                 Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
00543                            xmin, ymin, zmin);
00544                 sum = 0;
00545                 for (ii=0; ii<(nx*ny*nz); ii++) sum += (dielZMap[i]->data[ii]);
00546                     sum = sum*hx*hy*hzed;
00547                 Vnm_tprint(1, "  Volume integral = %3.2e A^3\n", sum);
00548                 break;
00549             //OpenDX Binary format (.dxbin)
00550             case VDF_DXBIN:
00551               //TODO: add this funct/method and maybe change the if stmt.
00552           if (Vgrid_readDXBIN(dielZMap[i], "FILE", "ASC", VNULL,
00553                    nosh->dielZpath[i]) != 1) {
00554             Vnm_tprint( 2, "Fatal error while reading from %s\n",
00555                    nosh->dielZpath[i]);
00556             return 0;
00557           }
00558
00559           // Read grid
00560           nx = dielZMap[i]->nx;
00561           ny = dielZMap[i]->ny;
00562           nz = dielZMap[i]->nz;
00563
00564           // Read spacings
00565           hx = dielZMap[i]->hx;
00566           hy = dielZMap[i]->hy;
00567           hzed = dielZMap[i]->hzed;
00568
00569           // Read minimum lower corner
00570           xmin = dielZMap[i]->xmin;
00571           ymin = dielZMap[i]->ymin;
00572           zmin = dielZMap[i]->zmin;
00573           Vnm_tprint(1, "  %d x %d x %d grid\n",
00574                  nx, ny, nz);
00575           Vnm_tprint(1, "  (%g, %g, %g) A spacings\n",
00576                  hx, hy, hzed);
00577           Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
00578                  xmin, ymin, zmin);
00579           sum = 0;
00580           for (ii=0; ii<(nx*ny*nz); ii++) sum += (dielZMap[i]->data[ii]);
00581             sum = sum*hx*hy*hzed;
00582           Vnm_tprint(1, "  Volume integral = %3.2e A^3\n", sum);
00583           break;
00584               // Binary file (GZip) format
00585               case VDF_GZ:
00586                   if (Vgrid_readGZ(dielZMap[i], nosh->dielZpath[i]) != 1) {
00587                       Vnm_tprint( 2, "Fatal error while reading from %s\n",
00588                                  nosh->dielZpath[i]);
00589                       return 0;
00590                   }
00591
00592                   // Read grid
00593                   nx = dielZMap[i]->nx;
00594                   ny = dielZMap[i]->ny;
00595                   nz = dielZMap[i]->nz;
00596
00597                   // Read spacings
00598                   hx = dielZMap[i]->hx;
00599                   hy = dielZMap[i]->hy;
00600                   hzed = dielZMap[i]->hzed;
00601
00602                   // Read minimum lower corner
00603                   xmin = dielZMap[i]->xmin;
00604                   ymin = dielZMap[i]->ymin;
00605                   zmin = dielZMap[i]->zmin;
00606                   Vnm_tprint(1, "  %d x %d x %d grid\n",
00607                              nx, ny, nz);
00608                   Vnm_tprint(1, "  (%g, %g, %g) A spacings\n",
00609                              hx, hy, hzed);
00610                   Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
00611                              xmin, ymin, zmin);
00612                   sum = 0;
00613                   for (ii=0; ii<(nx*ny*nz); ii++) sum += (dielZMap[i]->data[ii]);
00614                   sum = sum*hx*hy*hzed;
00615                   Vnm_tprint(1, "  Volume integral = %3.2e A^3\n", sum);
00616                   break;
00617               // UHBD format
```

```
00618             case VDF_UHBD:
00619                 Vnm_tprint( 2, "UHBD input not supported yet!\n");
00620                 return 0;
00621             // AVS UCD format
00622             case VDF_AVS:
00623                 Vnm_tprint( 2, "AVS input not supported yet!\n");
00624                 return 0;
00625             // FEtk MC Simplex Format (MCSF)
00626             case VDF_MCSF:
00627                 Vnm_tprint( 2, "MCSF input not supported yet!\n");
00628                 return 0;
00629             default:
00630                 Vnm_tprint( 2, "Invalid data format (%d)!\n",
00631                             nosh->dielfmt[i]);
00632                 return 0;
00633         }
00634     }
00635
00636     return 1;
00637 }
00638
00639 VPUBLIC void killDielMaps(NOsh *nosh,
00640                           Vgrid *dielXMap[NOSH_MAXMOL],
00641                           Vgrid *dielYMap[NOSH_MAXMOL],
00642                           Vgrid *dielZMap[NOSH_MAXMOL]) {
00643
00644     int i;
00645
00646     if (nosh->ndiel > 0) {
00647 #ifndef VAPBSQUIET
00648         Vnm_tprint( 1, "Destroying %d dielectric map sets\n",
00649                     nosh->ndiel);
00650 #endif
00651         for (i=0; i<nosh->ndiel; i++) {
00652             Vgrid_dtor(&(dielXMap[i]));
00653             Vgrid_dtor(&(dielYMap[i]));
00654             Vgrid_dtor(&(dielZMap[i]));
00655         }
00656     }
00657     else return;
00658
00659 }
00660
00664 VPUBLIC int loadKappaMaps(NOsh *nosh,
00665                           Vgrid *map[NOSH_MAXMOL]
00666                          ) {
00667
00668     int i,
00669         ii,
00670         len;
00671     double sum;
00672
00673     if (nosh->nkappa > 0)
00674         Vnm_tprint( 1, "Got paths for %d kappa maps\n", nosh->nkappa);
00675     else return 1;
00676
00677     for (i=0; i<nosh->nkappa; i++) {
00678         Vnm_tprint( 1, "Reading kappa map data from %s:\n",
00679                     nosh->kappapath[i]);
00680         map[i] = Vgrid_ctor(0, 0, 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, VNULL);
00681
00682         // Determine the format and read data if the format is valid.
00683         switch (nosh->kappafmt[i]) {
00684             // OpenDX (Data Explorer) format
00685             case VDF_DX:
00686                 if (Vgrid_readDX(map[i], "FILE", "ASC", VNULL,
00687                                  nosh->kappapath[i]) != 1) {
00688                     Vnm_tprint( 2, "Fatal error while reading from %s\n",
00689                                 nosh->kappapath[i]);
00690                     return 0;
00691                 }
00692                 Vnm_tprint(1, "  %d x %d x %d grid\n",
00693                            map[i]->nx, map[i]->ny, map[i]->nz);
00694                 Vnm_tprint(1, "  (%g, %g, %g) A spacings\n",
00695                            map[i]->hx, map[i]->hy, map[i]->hzed);
00696                 Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
00697                            map[i]->xmin, map[i]->ymin, map[i]->zmin);
00698                 sum = 0;
00699                 for (ii = 0, len = map[i]->nx * map[i]->ny * map[i]->nz;
00700                      ii < len;
00701                      ii++
```

```
00702                     ) {
00703                         sum += (map[i]->data[ii]);
00704                 }
00705                     sum = sum*map[i]->hx*map[i]->hy*map[i]->hzed;
00706                 Vnm_tprint(1, "  Volume integral = %3.2e A^3\n", sum);
00707                 break;
00708                 // OpenDX Binary (.dxbin) format
00709         case VDF_DXBIN:
00710           //TODO: write method and possible change if stmt.
00711           if (Vgrid_readDXBIN(map[i], "FILE", "ASC", VNULL,
00712                    nosh->kappapath[i]) != 1) {
00713             Vnm_tprint( 2, "Fatal error while reading from %s\n",
00714                   nosh->kappapath[i]);
00715             return 0;
00716           }
00717           Vnm_tprint(1, "  %d x %d x %d grid\n",
00718                  map[i]->nx, map[i]->ny, map[i]->nz);
00719           Vnm_tprint(1, "  (%g, %g, %g) A spacings\n",
00720                  map[i]->hx, map[i]->hy, map[i]->hzed);
00721           Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
00722                  map[i]->xmin, map[i]->ymin, map[i]->zmin);
00723           sum = 0;
00724           for (ii = 0, len = map[i]->nx * map[i]->ny * map[i]->nz;
00725              ii < len;
00726              ii++
00727             ) {
00728             sum += (map[i]->data[ii]);
00729           }
00730             sum = sum*map[i]->hx*map[i]->hy*map[i]->hzed;
00731           Vnm_tprint(1, "  Volume integral = %3.2e A^3\n", sum);
00732           break;
00733             // UHBD format
00734             case VDF_UHBD:
00735                 Vnm_tprint( 2, "UHBD input not supported yet!\n");
00736                 return 0;
00737             // FEtk MC Simplex Format (MCSF)
00738             case VDF_MCSF:
00739                 Vnm_tprint( 2, "MCSF input not supported yet!\n");
00740                 return 0;
00741             // AVS UCD format
00742             case VDF_AVS:
00743                 Vnm_tprint( 2, "AVS input not supported yet!\n");
00744                 return 0;
00745             // Binary file (GZip) format
00746             case VDF_GZ:
00747                 if (Vgrid_readGZ(map[i], nosh->kappapath[i]) != 1) {
00748                     Vnm_tprint( 2, "Fatal error while reading from %s\n",
00749                                 nosh->kappapath[i]);
00750                     return 0;
00751                 }
00752                 Vnm_tprint(1, "  %d x %d x %d grid\n",
00753                            map[i]->nx, map[i]->ny, map[i]->nz);
00754                 Vnm_tprint(1, "  (%g, %g, %g) A spacings\n",
00755                            map[i]->hx, map[i]->hy, map[i]->hzed);
00756                 Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
00757                            map[i]->xmin, map[i]->ymin, map[i]->zmin);
00758                 sum = 0;
00759                 for (ii=0, len=map[i]->nx*map[i]->ny*map[i]->nz; ii<len; ii++) {
00760                     sum += (map[i]->data[ii]);
00761                 }
00762                 sum = sum*map[i]->hx*map[i]->hy*map[i]->hzed;
00763                 Vnm_tprint(1, "  Volume integral = %3.2e A^3\n", sum);
00764                 break;
00765             default:
00766                 Vnm_tprint( 2, "Invalid data format (%d)!\n",
00767                             nosh->kappafmt[i]);
00768                 return 0;
00769         }
00770     }
00771
00772     return 1;
00773
00774 }
00775
00776 VPUBLIC void killKappaMaps(NOsh *nosh, Vgrid *map[NOSH_MAXMOL]) {
00777
00778     int i;
00779
00780     if (nosh->nkappa > 0) {
00781 #ifndef VAPBSQUIET
00782         Vnm_tprint( 1, "Destroying %d kappa maps\n", nosh->nkappa);
```

```
00783 #endif
00784         for (i=0; i<nosh->nkappa; i++) Vgrid_dtor(&(map[i]));
00785     }
00786     else return;
00787
00788 }
00789
00793 VPUBLIC int loadPotMaps(NOsh *nosh,
00794                         Vgrid *map[NOSH_MAXMOL]
00795                        ) {
00796
00797     int i,
00798         ii,
00799         len;
00800     double sum;
00801
00802     if (nosh->npot > 0)
00803         Vnm_tprint( 1, "Got paths for %d potential maps\n", nosh->npot);
00804     else return 1;
00805
00806     for (i=0; i<nosh->npot; i++) {
00807         Vnm_tprint( 1, "Reading potential map data from %s:\n",
00808                     nosh->potpath[i]);
00809         map[i] = Vgrid_ctor(0, 0, 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, VNULL);
00810         switch (nosh->potfmt[i]) {
00811             // OpenDX (Data Explorer) format
00812             case VDF_DX:
00813             // Binary file (GZip) format
00814             case VDF_GZ:
00815                 if (nosh->potfmt[i] == VDF_DX) {
00816                     if (Vgrid_readDX(map[i], "FILE", "ASC", VNULL,
00817                                      nosh->potpath[i]) != 1) {
00818                         Vnm_tprint( 2, "Fatal error while reading from %s\n",
00819                                     nosh->potpath[i]);
00820                         return 0;
00821                     }
00822                 }else {
00823                     if (Vgrid_readGZ(map[i], nosh->potpath[i]) != 1) {
00824                         Vnm_tprint( 2, "Fatal error while reading from %s\n",
00825                                     nosh->potpath[i]);
00826                         return 0;
00827                     }
00828                 }
00829                 Vnm_tprint(1, "  %d x %d x %d grid\n",
00830                            map[i]->nx, map[i]->ny, map[i]->nz);
00831                 Vnm_tprint(1, "  (%g, %g, %g) A spacings\n",
00832                            map[i]->hx, map[i]->hy, map[i]->hzed);
00833                 Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
00834                            map[i]->xmin, map[i]->ymin, map[i]->zmin);
00835                 sum = 0;
00836                 for (ii=0,len=map[i]->nx*map[i]->ny*map[i]->nz; ii<len; ii++) {
00837                     sum += (map[i]->data[ii]);
00838                 }
00839                 sum = sum*map[i]->hx*map[i]->hy*map[i]->hzed;
00840                 Vnm_tprint(1, "  Volume integral = %3.2e A^3\n", sum);
00841                 break;
00842             // UHBD format
00843             case VDF_UHBD:
00844                 Vnm_tprint( 2, "UHBD input not supported yet!\n");
00845                 return 0;
00846             // FEtk MC Simplex Format (MCSF)
00847             case VDF_MCSF:
00848                 Vnm_tprint( 2, "MCSF input not supported yet!\n");
00849                 return 0;
00850             // AVS UCD format
00851             case VDF_AVS:
00852                 Vnm_tprint( 2, "AVS input not supported yet!\n");
00853                 return 0;
00854             default:
00855                 Vnm_tprint( 2, "Invalid data format (%d)!\n",
00856                            nosh->potfmt[i]);
00857                 return 0;
00858         }
00859     }
00860
00861     return 1;
00862
00863 }
00864
00865 VPUBLIC void killPotMaps(NOsh *nosh,
00866                          Vgrid *map[NOSH_MAXMOL]
```

```
00867                         ) {
00868
00869     int i;
00870
00871     if (nosh->npot > 0) {
00872 #ifndef VAPBSQUIET
00873         Vnm_tprint( 1, "Destroying %d potential maps\n", nosh->npot);
00874 #endif
00875         for (i=0; i<nosh->npot; i++) Vgrid_dtor(&(map[i]));
00876     }
00877     else return;
00878
00879 }
00880
00884 VPUBLIC int loadChargeMaps(NOsh *nosh,
00885                            Vgrid *map[NOSH_MAXMOL]
00886                           ) {
00887
00888     int i,
00889         ii,
00890         len;
00891     double sum;
00892
00893     if (nosh->ncharge > 0)
00894         Vnm_tprint( 1, "Got paths for %d charge maps\n", nosh->ncharge);
00895     else return 1;
00896
00897     for (i=0; i<nosh->ncharge; i++) {
00898         Vnm_tprint( 1, "Reading charge map data from %s:\n",
00899                     nosh->chargepath[i]);
00900         map[i] = Vgrid_ctor(0, 0, 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, VNULL);
00901
00902         // Determine data format and read data
00903         switch (nosh->chargefmt[i]) {
00904             case VDF_DX:
00905                 if (Vgrid_readDX(map[i], "FILE", "ASC", VNULL,
00906                                  nosh->chargepath[i]) != 1) {
00907                     Vnm_tprint( 2, "Fatal error while reading from %s\n",
00908                                 nosh->chargepath[i]);
00909                     return 0;
00910                 }
00911                 Vnm_tprint(1, "  %d x %d x %d grid\n",
00912                            map[i]->nx, map[i]->ny, map[i]->nz);
00913                 Vnm_tprint(1, "  (%g, %g, %g) A spacings\n",
00914                            map[i]->hx, map[i]->hy, map[i]->hzed);
00915                 Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
00916                            map[i]->xmin, map[i]->ymin, map[i]->zmin);
00917                 sum = 0;
00918                 for (ii=0,len=map[i]->nx*map[i]->ny*map[i]->nz; ii<len; ii++) {
00919                     sum += (map[i]->data[ii]);
00920                 }
00921                     sum = sum*map[i]->hx*map[i]->hy*map[i]->hzed;
00922                 Vnm_tprint(1, "  Charge map integral = %3.2e e\n", sum);
00923                 break;
00924             case VDF_DXBIN:
00925               //TODO: write Vgrid_readDXBIN and possibly change if stmt.
00926         if (Vgrid_readDXBIN(map[i], "FILE", "ASC", VNULL,
00927                  nosh->chargepath[i]) != 1) {
00928           Vnm_tprint( 2, "Fatal error while reading from %s\n",
00929                  nosh->chargepath[i]);
00930           return 0;
00931         }
00932         Vnm_tprint(1, "  %d x %d x %d grid\n",
00933                map[i]->nx, map[i]->ny, map[i]->nz);
00934         Vnm_tprint(1, "  (%g, %g, %g) A spacings\n",
00935                map[i]->hx, map[i]->hy, map[i]->hzed);
00936         Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
00937                map[i]->xmin, map[i]->ymin, map[i]->zmin);
00938         sum = 0;
00939         for (ii=0,len=map[i]->nx*map[i]->ny*map[i]->nz; ii<len; ii++) {
00940           sum += (map[i]->data[ii]);
00941         }
00942           sum = sum*map[i]->hx*map[i]->hy*map[i]->hzed;
00943         Vnm_tprint(1, "  Charge map integral = %3.2e e\n", sum);
00944         break;
00945             case VDF_UHBD:
00946                 Vnm_tprint( 2, "UHBD input not supported yet!\n");
00947                 return 0;
00948             case VDF_AVS:
00949                 Vnm_tprint( 2, "AVS input not supported yet!\n");
00950                 return 0;
```

```
00951             case VDF_MCSF:
00952                 Vnm_tprint(2, "MCSF input not supported yet!\n");
00953                 return 0;
00954             case VDF_GZ:
00955                 if (Vgrid_readGZ(map[i], nosh->chargepath[i]) != 1) {
00956                     Vnm_tprint( 2, "Fatal error while reading from %s\n",
00957                                 nosh->chargepath[i]);
00958                     return 0;
00959                 }
00960                 Vnm_tprint(1, "  %d x %d x %d grid\n",
00961                            map[i]->nx, map[i]->ny, map[i]->nz);
00962                 Vnm_tprint(1, "  (%g, %g, %g) A spacings\n",
00963                            map[i]->hx, map[i]->hy, map[i]->hzed);
00964                 Vnm_tprint(1, "  (%g, %g, %g) A lower corner\n",
00965                            map[i]->xmin, map[i]->ymin, map[i]->zmin);
00966                 sum = 0;
00967                 for (ii=0,len=map[i]->nx*map[i]->ny*map[i]->nz; ii<len; ii++) {
00968                     sum += (map[i]->data[ii]);
00969                 }
00970                 sum = sum*map[i]->hx*map[i]->hy*map[i]->hzed;
00971                 Vnm_tprint(1, "  Charge map integral = %3.2e e\n", sum);
00972                 break;
00973             default:
00974                 Vnm_tprint( 2, "Invalid data format (%d)!\n",
00975                             nosh->kappafmt[i]);
00976                 return 0;
00977         }
00978     }
00979
00980     return 1;
00981
00982 }
00983
00984 VPUBLIC void killChargeMaps(NOsh *nosh,
00985                             Vgrid *map[NOSH_MAXMOL]
00986                            ) {
00987
00988     int i;
00989
00990     if (nosh->ncharge > 0) {
00991 #ifndef VAPBSQUIET
00992         Vnm_tprint( 1, "Destroying %d charge maps\n", nosh->ncharge);
00993 #endif
00994
00995         for (i=0; i<nosh->ncharge; i++) Vgrid_dtor(&(map[i]));
00996     }
00997
00998     else return;
00999
01000 }
01001
01002 VPUBLIC void printPBEPARM(PBEparm *pbeparm) {
01003
01004     int i;
01005     double ionstr = 0.0;
01006
01007     for (i=0; i<pbeparm->nion; i++)
01008         ionstr += 0.5*(VSQR(pbeparm->ionq[i])*pbeparm->ionc[i]);
01009
01010     Vnm_tprint( 1, "  Molecule ID: %d\n", pbeparm->molid);
01011     switch (pbeparm->pbetype) {
01012         case PBE_NPBE:
01013             Vnm_tprint( 1, "  Nonlinear traditional PBE\n");
01014             break;
01015         case PBE_LPBE:
01016             Vnm_tprint( 1, "  Linearized traditional PBE\n");
01017             break;
01018         case PBE_NRPBE:
01019             Vnm_tprint( 1, "  Nonlinear regularized PBE\n");
01020             Vnm_tprint( 2, "  ** Sorry, but Nathan broke the nonlinear regularized PBE implementation.
      **\n");
01021             Vnm_tprint( 2, "  ** Please let us know if you are interested in using it. **\n");
01022             VASSERT(0);
01023             break;
01024         case PBE_LRPBE:
01025             Vnm_tprint( 1, "  Linearized regularized PBE\n");
01026             break;
01027         case PBE_SMPBE: /* SMPBE Added */
01028             Vnm_tprint( 1, "  Nonlinear Size-Modified PBE\n");
01029             break;
01030         default:
```

```
01031             Vnm_tprint(2, "  Unknown PBE type (%d)!\n", pbeparm->pbetype);
01032             break;
01033     }
01034     if (pbeparm->bcfl == BCFL_ZERO) {
01035         Vnm_tprint( 1, "  Zero boundary conditions\n");
01036     } else if (pbeparm->bcfl == BCFL_SDH) {
01037         Vnm_tprint( 1, "  Single Debye-Huckel sphere boundary \
01038 conditions\n");
01039     } else if (pbeparm->bcfl == BCFL_MDH) {
01040         Vnm_tprint( 1, "  Multiple Debye-Huckel sphere boundary \
01041 conditions\n");
01042     } else if (pbeparm->bcfl == BCFL_FOCUS) {
01043         Vnm_tprint( 1, "  Boundary conditions from focusing\n");
01044     } else if (pbeparm->bcfl == BCFL_MAP) {
01045         Vnm_tprint( 1, "  Boundary conditions from potential map\n");
01046     } else if (pbeparm->bcfl == BCFL_MEM) {
01047         Vnm_tprint( 1, "  Membrane potential boundary conditions.\n");
01048     }
01049     Vnm_tprint( 1, "  %d ion species (%4.3f M ionic strength):\n",
01050                 pbeparm->nion, ionstr);
01051     for (i=0; i<pbeparm->nion; i++) {
01052         Vnm_tprint( 1, "    %4.3f A-radius, %4.3f e-charge, \
01053 %4.3f M concentration\n",
01054                     pbeparm->ionr[i], pbeparm->ionq[i], pbeparm->ionc[i]);
01055     }
01056
01057     if(pbeparm->pbetype == PBE_SMPBE){ /* SMPBE Added */
01058         Vnm_tprint( 1, "  Lattice spacing: %4.3f A (SMPBE) \n", pbeparm->smvolume);
01059         Vnm_tprint( 1, "  Relative size parameter: %4.3f  (SMPBE) \n", pbeparm->smsize);
01060     }
01061
01062     Vnm_tprint( 1, "  Solute dielectric: %4.3f\n", pbeparm->pdie);
01063     Vnm_tprint( 1, "  Solvent dielectric: %4.3f\n", pbeparm->sdie);
01064     switch (pbeparm->srfm) {
01065         case 0:
01066             Vnm_tprint( 1, "  Using \"molecular\" surface \
01067 definition; no smoothing\n");
01068             Vnm_tprint( 1, "  Solvent probe radius: %4.3f A\n",
01069                         pbeparm->srad);
01070             break;
01071         case 1:
01072             Vnm_tprint( 1, "  Using \"molecular\" surface definition;\
01073 harmonic average smoothing\n");
01074             Vnm_tprint( 1, "  Solvent probe radius: %4.3f A\n",
01075                         pbeparm->srad);
01076             break;
01077         case 2:
01078             Vnm_tprint( 1, "  Using spline-based surface definition;\
01079 window = %4.3f\n", pbeparm->swin);
01080             break;
01081         default:
01082             break;
01083     }
01084     Vnm_tprint( 1, "  Temperature:  %4.3f K\n", pbeparm->temp);
01085     if (pbeparm->calcenergy != PCE_NO) Vnm_tprint( 1, "  Electrostatic \
01086 energies will be calculated\n");
01087     if (pbeparm->calcforce == PCF_TOTAL) Vnm_tprint( 1, "  Net solvent \
01088 forces will be calculated \n");
01089     if (pbeparm->calcforce == PCF_COMPS) Vnm_tprint( 1, "  All-atom \
01090 solvent forces will be calculated\n");
01091     for (i=0; i<pbeparm->numwrite; i++) {
01092         switch (pbeparm->writetype[i]) {
01093             case VDT_CHARGE:
01094                 Vnm_tprint(1, "  Charge distribution to be written to ");
01095                 break;
01096             case VDT_POT:
01097                 Vnm_tprint(1, "  Potential to be written to ");
01098                 break;
01099             case VDT_SMOL:
01100                 Vnm_tprint(1, "  Molecular solvent accessibility \
01101 to be written to ");
01102                 break;
01103             case VDT_SSPL:
01104                 Vnm_tprint(1, "  Spline-based solvent accessibility \
01105 to be written to ");
01106                 break;
01107             case VDT_VDW:
01108                 Vnm_tprint(1, "  van der Waals solvent accessibility \
01109 to be written to ");
01110                 break;
01111             case VDT_IVDW:
```

```
01112                 Vnm_tprint(1, "  Ion accessibility to be written to ");
01113                 break;
01114             case VDT_LAP:
01115                 Vnm_tprint(1, "  Potential Laplacian to be written to ");
01116                 break;
01117             case VDT_EDENS:
01118                 Vnm_tprint(1, "  Energy density to be written to ");
01119                 break;
01120             case VDT_NDENS:
01121                 Vnm_tprint(1, "  Ion number density to be written to ");
01122                 break;
01123             case VDT_QDENS:
01124                 Vnm_tprint(1, "  Ion charge density to be written to ");
01125                 break;
01126             case VDT_DIELX:
01127                 Vnm_tprint(1, "  X-shifted dielectric map to be written \
01128 to ");
01129                 break;
01130             case VDT_DIELY:
01131                 Vnm_tprint(1, "  Y-shifted dielectric map to be written \
01132 to ");
01133                 break;
01134             case VDT_DIELZ:
01135                 Vnm_tprint(1, "  Z-shifted dielectric map to be written \
01136 to ");
01137                 break;
01138             case VDT_KAPPA:
01139                 Vnm_tprint(1, "  Kappa map to be written to ");
01140                 break;
01141             case VDT_ATOMPOT:
01142                 Vnm_tprint(1, "  Atom potentials to be written to ");
01143                 break;
01144             default:
01145                 Vnm_tprint(2, "  Invalid data type for writing!\n");
01146                 break;
01147         }
01148         switch (pbeparm->writefmt[i]) {
01149             case VDF_DX:
01150                 Vnm_tprint(1, "%s.%s\n", pbeparm->writestem[i], "dx");
01151                 break;
01152             case VDF_DXBIN:
01153         Vnm_tprint(1, "%s.%s\n", pbeparm->writestem[i], "dxbin");
01154         break;
01155             case VDF_GZ:
01156                 Vnm_tprint(1, "%s.%s\n", pbeparm->writestem[i], "dx.gz");
01157                 break;
01158             case VDF_UHBD:
01159                 Vnm_tprint(1, "%s.%s\n", pbeparm->writestem[i], "grd");
01160                 break;
01161             case VDF_AVS:
01162                 Vnm_tprint(1, "%s.%s\n", pbeparm->writestem[i], "ucd");
01163                 break;
01164             case VDF_MCSF:
01165                 Vnm_tprint(1, "%s.%s\n", pbeparm->writestem[i], "mcsf");
01166                 break;
01167             case VDF_FLAT:
01168                 Vnm_tprint(1, "%s.%s\n", pbeparm->writestem[i], "txt");
01169                 break;
01170             default:
01171                 Vnm_tprint(2, "  Invalid format for writing!\n");
01172                 break;
01173         }
01174 
01175     }
01176 
01177 }
01178 
01179 VPUBLIC void printMGPARM(MGparm *mgparm, double realCenter[3]) {
01180 
01181     switch (mgparm->chgm) {
01182         case 0:
01183             Vnm_tprint(1, "  Using linear spline charge discretization.\n");
01184             break;
01185         case 1:
01186             Vnm_tprint(1, "  Using cubic spline charge discretization.\n");
01187             break;
01188         default:
01189             break;
01190     }
01191     if (mgparm->type == MCT_PARALLEL) {
01192         Vnm_tprint( 1, "  Partition overlap fraction = %g\n",
```

```
01193                     mgparm->ofrac);
01194         Vnm_tprint( 1, "  Processor array = %d x %d x %d\n",
01195                     mgparm->pdime[0], mgparm->pdime[1], mgparm->pdime[2]);
01196     }
01197     Vnm_tprint( 1, "  Grid dimensions: %d x %d x %d\n",
01198                 mgparm->dime[0], mgparm->dime[1], mgparm->dime[2]);
01199     Vnm_tprint( 1, "  Grid spacings: %4.3f x %4.3f x %4.3f\n",
01200                 mgparm->grid[0], mgparm->grid[1], mgparm->grid[2]);
01201     Vnm_tprint( 1, "  Grid lengths: %4.3f x %4.3f x %4.3f\n",
01202                 mgparm->glen[0], mgparm->glen[1], mgparm->glen[2]);
01203     Vnm_tprint( 1, "  Grid center: (%4.3f, %4.3f, %4.3f)\n",
01204                 realCenter[0], realCenter[1], realCenter[2]);
01205     Vnm_tprint( 1, "  Multigrid levels: %d\n", mgparm->nlev);
01206
01207 }
01208
01212 VPUBLIC int initMG(int icalc,
01213                    NOsh *nosh, MGparm *mgparm,
01214                    PBEparm *pbeparm,
01215                    double realCenter[3],
01216                    Vpbe *pbe[NOSH_MAXCALC],
01217                    Valist *alist[NOSH_MAXMOL],
01218                    Vgrid *dielXMap[NOSH_MAXMOL],
01219                    Vgrid *dielYMap[NOSH_MAXMOL],
01220                    Vgrid *dielZMap[NOSH_MAXMOL],
01221                    Vgrid *kappaMap[NOSH_MAXMOL],
01222                    Vgrid *chargeMap[NOSH_MAXMOL],
01223                    Vpmgp *pmgp[NOSH_MAXCALC],
01224                    Vpmg *pmg[NOSH_MAXCALC],
01225                    Vgrid *potMap[NOSH_MAXMOL]
01226                   ) {
01227
01228     int j,
01229         focusFlag,
01230         iatom;
01231     size_t bytesTotal,
01232            highWater;
01233     double sparm,
01234            iparm,
01235            q;
01236     Vatom *atom = VNULL;
01237     Vgrid *theDielXMap = VNULL,
01238           *theDielYMap = VNULL,
01239           *theDielZMap = VNULL;
01240     Vgrid *theKappaMap = VNULL,
01241           *thePotMap = VNULL,
01242           *theChargeMap = VNULL;
01243     Valist *myalist = VNULL;
01244
01245     Vnm_tstart(APBS_TIMER_SETUP, "Setup timer");
01246
01247     /* Update the grid center */
01248     for (j=0; j<3; j++) realCenter[j] = mgparm->center[j];
01249
01250     /* Check for completely-neutral molecule */
01251     q = 0;
01252     myalist = alist[pbeparm->molid-1];
01253     for (iatom=0; iatom<Valist_getNumberAtoms(myalist); iatom++) {
01254         atom = Valist_getAtom(myalist, iatom);
01255         q += VSQR(Vatom_getCharge(atom));
01256     }
01257     /*  D. Gohara 10/22/09 - disabled
01258     if (q < (1e-6)) {
01259         Vnm_tprint(2, "Molecule #%d is uncharged!\n", pbeparm->molid);
01260         Vnm_tprint(2, "Sum square charge = %g!\n", q);
01261         return 0;
01262     }
01263     */
01264
01265     /* Set up PBE object */
01266     Vnm_tprint(0, "Setting up PBE object...\n");
01267     if (pbeparm->srfm == VSM_SPLINE) {
01268         sparm = pbeparm->swin;
01269     } else {
01270         sparm = pbeparm->srad;
01271     }
01272     if (pbeparm->nion > 0) {
01273         iparm = pbeparm->ionr[0];
01274     } else {
01275         iparm = 0.0;
01276     }
```

```
01277     if (pbeparm->bcfl == BCFL_FOCUS) {
01278         if (icalc == 0) {
01279             Vnm_tprint( 2, "Can't focus first calculation!\n");
01280             return 0;
01281         }
01282         focusFlag = 1;
01283     } else {
01284         focusFlag = 0;
01285     }
01286 
01287     // Construct Vpbe object
01288     pbe[icalc] = Vpbe_ctor(myalist, pbeparm->nion,
01289                            pbeparm->ionc, pbeparm->ionr, pbeparm->ionq,
01290                            pbeparm->temp, pbeparm->pdie,
01291                            pbeparm->sdie, sparm, focusFlag, pbeparm->sdens,
01292                            pbeparm->zmem, pbeparm->Lmem, pbeparm->mdie,
01293                            pbeparm->memv);
01294 
01295     /* Set up PDE object */
01296     Vnm_tprint(0, "Setting up PDE object...\n");
01297     switch (pbeparm->pbetype) {
01298         case PBE_NPBE:
01299             /* TEMPORARY USEAQUA */
01300             mgparm->nonlintype = NONLIN_NPBE;
01301             mgparm->method = (mgparm->useAqua == 1) ? VSOL_NewtonAqua : VSOL_Newton;
01302             pmgp[icalc] = Vpmgp_ctor(mgparm);
01303             break;
01304         case PBE_LPBE:
01305             /* TEMPORARY USEAQUA */
01306             mgparm->nonlintype = NONLIN_LPBE;
01307             mgparm->method = (mgparm->useAqua == 1) ? VSOL_CGMGAqua : VSOL_MG;
01308             pmgp[icalc] = Vpmgp_ctor(mgparm);
01309             break;
01310         case PBE_LRPBE:
01311             Vnm_tprint(2, "Sorry, LRPBE isn't supported with the MG solver!\n");
01312             return 0;
01313         case PBE_NRPBE:
01314             Vnm_tprint(2, "Sorry, NRPBE isn't supported with the MG solver!\n");
01315             return 0;
01316         case PBE_SMPBE: /* SMPBE Added */
01317          /* Due to numerical issues the SMPBE is currenty disabled. (JMB)*/
01318          Vnm_tprint(2, "  ** Sorry, due to numerical stability issues SMPBE is currently disabled. We
      apologize for the inconvenience.\n");
01319             Vnm_tprint(2, "  ** Please let us know if you would like to use it in the future.\n");
01320             return 0;
01321 
01322          /*
01323             mgparm->nonlintype = NONLIN_SMPBE;
01324             pmgp[icalc] = Vpmgp_ctor(mgparm);
01325    */
01326             /* Copy Code */
01327             /*
01328             pbe[icalc]->smsize = pbeparm->smsize;
01329             pbe[icalc]->smvolume = pbeparm->smvolume;
01330             pbe[icalc]->ipkey = pmgp[icalc]->ipkey;
01331 
01332             break;
01333             */
01334         default:
01335             Vnm_tprint(2, "Error!  Unknown PBE type (%d)!\n", pbeparm->pbetype);
01336             return 0;
01337     }
01338     Vnm_tprint(0, "Setting PDE center to local center...\n");
01339     pmgp[icalc]->bcfl = pbeparm->bcfl;
01340     pmgp[icalc]->xcent = realCenter[0];
01341     pmgp[icalc]->ycent = realCenter[1];
01342     pmgp[icalc]->zcent = realCenter[2];
01343 
01344     if (pbeparm->bcfl == BCFL_FOCUS) {
01345         if (icalc == 0) {
01346             Vnm_tprint( 2, "Can't focus first calculation!\n");
01347             return 0;
01348         }
01349         /* Focusing requires the previous calculation in order to setup the
01350         current run... */
01351         pmg[icalc] = Vpmg_ctor(pmgp[icalc], pbe[icalc], 1, pmg[icalc-1],
01352                                mgparm, pbeparm->calcenergy);
01353         /* ...however, it should be done with the previous calculation now, so
01354         we should be able to destroy it here. */
01355         /* Vpmg_dtor(&(pmg[icalc-1])); */
01356     } else {
```

```
01357         if (icalc>0) Vpmg_dtor(&(pmg[icalc-1]));
01358         pmg[icalc] = Vpmg_ctor(pmgp[icalc], pbe[icalc], 0, VNULL, mgparm, PCE_NO);
01359     }
01360     if (icalc>0) {
01361         Vpmgp_dtor(&(pmgp[icalc-1]));
01362         Vpbe_dtor(&(pbe[icalc-1]));
01363     }
01364     if (pbeparm->useDielMap) {
01365         if ((pbeparm->dielMapID-1) < nosh->ndiel) {
01366             theDielXMap = dielXMap[pbeparm->dielMapID-1];
01367         } else {
01368             Vnm_print(2, "Error!  %d is not a valid dielectric map ID!\n",
01369                       pbeparm->dielMapID);
01370             return 0;
01371         }
01372     }
01373     if (pbeparm->useDielMap) {
01374         if ((pbeparm->dielMapID-1) < nosh->ndiel) {
01375             theDielYMap = dielYMap[pbeparm->dielMapID-1];
01376         } else {
01377             Vnm_print(2, "Error!  %d is not a valid dielectric map ID!\n",
01378                       pbeparm->dielMapID);
01379             return 0;
01380         }
01381     }
01382     if (pbeparm->useDielMap) {
01383         if ((pbeparm->dielMapID-1) < nosh->ndiel) {
01384             theDielZMap = dielZMap[pbeparm->dielMapID-1];
01385         } else {
01386             Vnm_print(2, "Error!  %d is not a valid dielectric map ID!\n",
01387                       pbeparm->dielMapID);
01388             return 0;
01389         }
01390     }
01391     if (pbeparm->useKappaMap) {
01392         if ((pbeparm->kappaMapID-1) < nosh->nkappa) {
01393             theKappaMap = kappaMap[pbeparm->kappaMapID-1];
01394         } else {
01395             Vnm_print(2, "Error!  %d is not a valid kappa map ID!\n",
01396                       pbeparm->kappaMapID);
01397             return 0;
01398         }
01399     }
01400     if (pbeparm->usePotMap) {
01401         if ((pbeparm->potMapID-1) < nosh->npot) {
01402             thePotMap = potMap[pbeparm->potMapID-1];
01403         } else {
01404             Vnm_print(2, "Error!  %d is not a valid potential map ID!\n",
01405                       pbeparm->potMapID);
01406             return 0;
01407         }
01408     }
01409     if (pbeparm->useChargeMap) {
01410         if ((pbeparm->chargeMapID-1) < nosh->ncharge) {
01411             theChargeMap = chargeMap[pbeparm->chargeMapID-1];
01412         } else {
01413             Vnm_print(2, "Error!  %d is not a valid charge map ID!\n",
01414                       pbeparm->chargeMapID);
01415             return 0;
01416         }
01417     }
01418
01419     if (pbeparm->bcfl == BCFL_MAP && thePotMap == VNULL) {
01420         Vnm_print(2, "Warning: You specified 'bcfl map' in the input file, but no potential map was
      found.\n");
01421         Vnm_print(2, "         You must specify 'usemap pot' statement in the APBS input file!\n");
01422         Vnm_print(2, "Bailing out ...\n");
01423         return 0;
01424     }
01425
01426     // Initialize calculation coefficients
01427     if (!Vpmg_fillco(pmg[icalc],
01428                      pbeparm->srfm, pbeparm->swin, mgparm->chgm,
01429                      pbeparm->useDielMap, theDielXMap,
01430                      pbeparm->useDielMap, theDielYMap,
01431                      pbeparm->useDielMap, theDielZMap,
01432                      pbeparm->useKappaMap, theKappaMap,
01433                      pbeparm->usePotMap, thePotMap,
01434                      pbeparm->useChargeMap, theChargeMap)) {
01435         Vnm_print(2, "initMG:  problems setting up coefficients (fillco)!\n");
01436         return 0;
```

```
01437     }
01438
01439     /* Print a few derived parameters */
01440 #ifndef VAPBSQUIET
01441     Vnm_tprint(1, "  Debye length:  %g A\n", Vpbe_getDeblen(pbe[icalc]));
01442 #endif
01443
01444     /* Setup time statistics */
01445     Vnm_tstop(APBS_TIMER_SETUP, "Setup timer");
01446
01447     /* Memory statistics */
01448     bytesTotal = Vmem_bytesTotal();
01449     highWater = Vmem_highWaterTotal();
01450
01451 #ifndef VAPBSQUIET
01452     Vnm_tprint( 1, "  Current memory usage:  %4.3f MB total, \
01453 %4.3f MB high water\n", (double)(bytesTotal)/(1024.*1024.),
01454                 (double)(highWater)/(1024.*1024.));
01455 #endif
01456
01457     return 1;
01458
01459 }
01460
01461 VPUBLIC void killMG(NOsh *nosh, Vpbe *pbe[NOSH_MAXCALC],
01462                     Vpmgp *pmgp[NOSH_MAXCALC], Vpmg *pmg[NOSH_MAXCALC]) {
01463
01464         int i;
01465
01466 #ifndef VAPBSQUIET
01467     Vnm_tprint(1, "Destroying multigrid structures.\n");
01468 #endif
01469
01470     /*
01471        There appears to be a relationship (or this is a bug in Linux, can't tell
01472        at the moment, since Linux is the only OS that seems to be affected)
01473        between one of the three object types: Vpbe, Vpmg or Vpmgp that requires
01474        deallocations to be performed in a specific order. This results in a
01475        bug some of the time when freeing Vpmg objects below. Therefore it
01476        appears to be important to release the Vpmg structs BEFORE the Vpmgp structs .
01477     */
01478     Vpmg_dtor(&(pmg[nosh->ncalc-1]));
01479
01480     for(i=0;i<nosh->ncalc;i++){
01481         Vpbe_dtor(&(pbe[i]));
01482         Vpmgp_dtor(&(pmgp[i]));
01483     }
01484
01485 }
01486
01487 VPUBLIC int solveMG(NOsh *nosh,
01488                     Vpmg *pmg,
01489                     MGparm_CalcType type
01490                    ) {
01491
01492     int nx,
01493         ny,
01494         nz,
01495         i;
01496
01497     if (nosh != VNULL) {
01498         if (nosh->bogus) return 1;
01499     }
01500
01501     Vnm_tstart(APBS_TIMER_SOLVER, "Solver timer");
01502
01503
01504     if (type != MCT_DUMMY) {
01505         if (!Vpmg_solve(pmg)) {
01506             Vnm_print(2, "  Error during PDE solution!\n");
01507             return 0;
01508         }
01509     } else {
01510         Vnm_tprint( 1,"  Skipping solve for mg-dummy run; zeroing \
01511 solution array\n");
01512         nx = pmg->pmgp->nx;
01513         ny = pmg->pmgp->ny;
01514         nz = pmg->pmgp->nz;
01515         for (i=0; i<nx*ny*nz; i++) pmg->u[i] = 0.0;
01516     }
01517     Vnm_tstop(APBS_TIMER_SOLVER, "Solver timer");
```

```
01518
01519     return 1;
01520
01521 }
01522
01523 VPUBLIC int setPartMG(NOsh *nosh,
01524                       MGparm *mgparm,
01525                       Vpmg *pmg
01526                      ) {
01527
01528     int j;
01529     double partMin[3],
01530            partMax[3];
01531
01532     if (nosh->bogus) return 1;
01533
01534     if (mgparm->type == MCT_PARALLEL) {
01535         for (j=0; j<3; j++) {
01536             partMin[j] = mgparm->partDisjCenter[j] - 0.5*mgparm->partDisjLength[j];
01537             partMax[j] = mgparm->partDisjCenter[j] + 0.5*mgparm->partDisjLength[j];
01538         }
01539 #if 0
01540         Vnm_tprint(1, "setPartMG (%s, %d):  Disj part center = (%g, %g, %g)\n",
01541                    __FILE__, __LINE__,
01542                    mgparm->partDisjCenter[0],
01543                    mgparm->partDisjCenter[1],
01544                    mgparm->partDisjCenter[2]
01545                    );
01546         Vnm_tprint(1, "setPartMG (%s, %d):  Disj part lower corner = (%g, %g, %g)\n",
01547                    __FILE__, __LINE__, partMin[0], partMin[1], partMin[2]);
01548         Vnm_tprint(1, "setPartMG (%s, %d):  Disj part upper corner = (%g, %g, %g)\n",
01549                    __FILE__, __LINE__,
01550                    partMax[0], partMax[1], partMax[2]);
01551 #endif
01552     } else {
01553         for (j=0; j<3; j++) {
01554             partMin[j] = mgparm->center[j] - 0.5*mgparm->glen[j];
01555             partMax[j] = mgparm->center[j] + 0.5*mgparm->glen[j];
01556         }
01557     }
01558     /* Vnm_print(1, "DEBUG (%s, %d):  setPartMG calling setPart with upper corner \
01559 %g %g %g and lower corner %g %g %g\n", __FILE__,  __LINE__,
01560               partMin[0], partMin[1], partMin[2],
01561               partMax[0], partMax[1], partMax[2]); */
01562     Vpmg_setPart(pmg, partMin, partMax, mgparm->partDisjOwnSide);
01563
01564
01565     return 1;
01566
01567 }
01568
01569 VPUBLIC int energyMG(NOsh *nosh,
01570                      int icalc,
01571                      Vpmg *pmg,
01572                      int *nenergy,
01573                      double *totEnergy,
01574                      double *qfEnergy,
01575                      double *qmEnergy,
01576                      double *dielEnergy
01577                     ) {
01578
01579     Valist *alist;
01580     Vatom *atom;
01581     int i,
01582         extEnergy;
01583     double tenergy;
01584     MGparm *mgparm;
01585     PBEparm *pbeparm;
01586
01587     mgparm = nosh->calc[icalc]->mgparm;
01588     pbeparm = nosh->calc[icalc]->pbeparm;
01589
01590     Vnm_tstart(APBS_TIMER_ENERGY, "Energy timer");
01591     extEnergy = 1;
01592
01593     if (pbeparm->calcenergy == PCE_TOTAL) {
01594         *nenergy = 1;
01595         /* Some processors don't count */
01596         if (nosh->bogus == 0) {
01597             *totEnergy = Vpmg_energy(pmg, extEnergy);
01598 #ifndef VAPBSQUIET
```

```
01599             Vnm_tprint( 1, "  Total electrostatic energy = %1.12E kJ/mol\n",
01600                         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(*totEnergy));
01601 #endif
01602         } else *totEnergy = 0;
01603     } else if (pbeparm->calcenergy == PCE_COMPS) {
01604         *nenergy = 1;
01605         *totEnergy = Vpmg_energy(pmg, extEnergy);
01606         *qfEnergy = Vpmg_qfEnergy(pmg, extEnergy);
01607         *qmEnergy = Vpmg_qmEnergy(pmg, extEnergy);
01608         *dielEnergy = Vpmg_dielEnergy(pmg, extEnergy);
01609 #ifndef VAPBSQUIET
01610         Vnm_tprint( 1, "  Total electrostatic energy = %1.12E \
01611 kJ/mol\n", Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(*totEnergy));
01612         Vnm_tprint( 1, "  Fixed charge energy = %g kJ/mol\n",
01613                     0.5*Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(*qfEnergy));
01614         Vnm_tprint( 1, "  Mobile charge energy = %g kJ/mol\n",
01615                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(*qmEnergy));
01616         Vnm_tprint( 1, "  Dielectric energy = %g kJ/mol\n",
01617                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(*dielEnergy));
01618         Vnm_tprint( 1, "  Per-atom energies:\n");
01619 #endif
01620         alist = pmg->pbe->alist;
01621         for (i=0; i<Valist_getNumberAtoms(alist); i++) {
01622             atom = Valist_getAtom(alist, i);
01623             tenergy = Vpmg_qfAtomEnergy(pmg, atom);
01624 #ifndef VAPBSQUIET
01625             Vnm_tprint( 1, "      Atom %d:  %1.12E kJ/mol\n", i,
01626                         0.5*Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*tenergy);
01627 #endif
01628         }
01629     } else *nenergy = 0;
01630
01631     Vnm_tstop(APBS_TIMER_ENERGY, "Energy timer");
01632
01633     return 1;
01634 }
01635
01636 VPUBLIC int forceMG(Vmem *mem,
01637                     NOsh *nosh,
01638                     PBEparm *pbeparm,
01639                     MGparm *mgparm,
01640                     Vpmg *pmg,
01641                     int *nforce,
01642                     AtomForce **atomForce,
01643                     Valist *alist[NOSH_MAXMOL]
01644                    ) {
01645
01646     int j,
01647         k;
01648     double qfForce[3],
01649            dbForce[3],
01650            ibForce[3];
01651
01652     Vnm_tstart(APBS_TIMER_FORCE, "Force timer");
01653
01654 #ifndef VAPBSQUIET
01655     Vnm_tprint( 1,"  Calculating forces...\n");
01656 #endif
01657
01658     if (pbeparm->calcforce == PCF_TOTAL) {
01659         *nforce = 1;
01660         *atomForce = (AtomForce *)Vmem_malloc(mem, 1, sizeof(AtomForce));
01661         /* Clear out force arrays */
01662         for (j=0; j<3; j++) {
01663             (*atomForce)[0].qfForce[j] = 0;
01664             (*atomForce)[0].ibForce[j] = 0;
01665             (*atomForce)[0].dbForce[j] = 0;
01666         }
01667         for (j=0;j<Valist_getNumberAtoms(alist[pbeparm->molid-1]);j++) {
01668             if (nosh->bogus == 0) {
01669                 VASSERT(Vpmg_qfForce(pmg, qfForce, j, mgparm->chgm));
01670                 VASSERT(Vpmg_ibForce(pmg, ibForce, j, pbeparm->srfm));
01671                 VASSERT(Vpmg_dbForce(pmg, dbForce, j, pbeparm->srfm));
01672             } else {
01673                 for (k=0; k<3; k++) {
01674                     qfForce[k] = 0;
01675                     ibForce[k] = 0;
01676                     dbForce[k] = 0;
01677                 }
01678             }
01679             for (k=0; k<3; k++) {
```

```
01680                 (*atomForce)[0].qfForce[k] += qfForce[k];
01681                 (*atomForce)[0].ibForce[k] += ibForce[k];
01682                 (*atomForce)[0].dbForce[k] += dbForce[k];
01683             }
01684         }
01685 #ifndef VAPBSQUIET
01686         Vnm_tprint( 1, "  Printing net forces for molecule %d (kJ/mol/A)\n",
01687                     pbeparm->molid);
01688         Vnm_tprint( 1, "  Legend:\n");
01689         Vnm_tprint( 1, "    qf  -- fixed charge force\n");
01690         Vnm_tprint( 1, "    db  -- dielectric boundary force\n");
01691         Vnm_tprint( 1, "    ib  -- ionic boundary force\n");
01692         Vnm_tprint( 1, "  qf  %4.3e  %4.3e  %4.3e\n",
01693                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(*atomForce)[0].qfForce[0],
01694                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(*atomForce)[0].qfForce[1],
01695                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(*atomForce)[0].qfForce[2]);
01696         Vnm_tprint( 1, "  ib  %4.3e  %4.3e  %4.3e\n",
01697                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(*atomForce)[0].ibForce[0],
01698                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(*atomForce)[0].ibForce[1],
01699                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(*atomForce)[0].ibForce[2]);
01700         Vnm_tprint( 1, "  db  %4.3e  %4.3e  %4.3e\n",
01701                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(*atomForce)[0].dbForce[0],
01702                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(*atomForce)[0].dbForce[1],
01703                     Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(*atomForce)[0].dbForce[2]);
01704 #endif
01705     } else if (pbeparm->calcforce == PCF_COMPS) {
01706         *nforce = Valist_getNumberAtoms(alist[pbeparm->molid-1]);
01707         *atomForce = (AtomForce *)Vmem_malloc(mem, *nforce,
01708                                               sizeof(AtomForce));
01709 #ifndef VAPBSQUIET
01710         Vnm_tprint( 1, "  Printing per-atom forces for molecule %d (kJ/mol/A)\n",
01711                     pbeparm->molid);
01712         Vnm_tprint( 1, "  Legend:\n");
01713         Vnm_tprint( 1, "    tot n -- total force for atom n\n");
01714         Vnm_tprint( 1, "    qf  n -- fixed charge force for atom n\n");
01715         Vnm_tprint( 1, "    db  n -- dielectric boundary force for atom n\n");
01716         Vnm_tprint( 1, "    ib  n -- ionic boundary force for atom n\n");
01717 #endif
01718         for (j=0;j<Valist_getNumberAtoms(alist[pbeparm->molid-1]);j++) {
01719             if (nosh->bogus == 0) {
01720                 VASSERT(Vpmg_qfForce(pmg, (*atomForce)[j].qfForce, j,
01721                                      mgparm->chgm));
01722                 VASSERT(Vpmg_ibForce(pmg, (*atomForce)[j].ibForce, j,
01723                                      pbeparm->srfm));
01724                 VASSERT(Vpmg_dbForce(pmg, (*atomForce)[j].dbForce, j,
01725                                      pbeparm->srfm));
01726             } else {
01727                 for (k=0; k<3; k++) {
01728                     (*atomForce)[j].qfForce[k] = 0;
01729                     (*atomForce)[j].ibForce[k] = 0;
01730                     (*atomForce)[j].dbForce[k] = 0;
01731                 }
01732             }
01733 #ifndef VAPBSQUIET
01734             Vnm_tprint( 1, "mgF  tot %d  %4.3e  %4.3e  %4.3e\n", j,
01735                         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01736                         *((*atomForce)[j].qfForce[0]+(*atomForce)[j].ibForce[0]+
01737                           (*atomForce)[j].dbForce[0]),
01738                         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01739                         *((*atomForce)[j].qfForce[1]+(*atomForce)[j].ibForce[1]+
01740                           (*atomForce)[j].dbForce[1]),
01741                         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01742                         *((*atomForce)[j].qfForce[2]+(*atomForce)[j].ibForce[2]+
01743                           (*atomForce)[j].dbForce[2]));
01744             Vnm_tprint( 1, "mgF  qf  %d  %4.3e  %4.3e  %4.3e\n", j,
01745                         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01746                         *(*atomForce)[j].qfForce[0],
01747                         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01748                         *(*atomForce)[j].qfForce[1],
01749                         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01750                         *(*atomForce)[j].qfForce[2]);
01751             Vnm_tprint( 1, "mgF  ib  %d  %4.3e  %4.3e  %4.3e\n", j,
01752                         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01753                         *(*atomForce)[j].ibForce[0],
01754                         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01755                         *(*atomForce)[j].ibForce[1],
01756                         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01757                         *(*atomForce)[j].ibForce[2]);
01758             Vnm_tprint( 1, "mgF  db  %d  %4.3e  %4.3e  %4.3e\n", j,
01759                         Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01760                         *(*atomForce)[j].dbForce[0],
```

```
01761                           Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01762                           *(*atomForce)[j].dbForce[1],
01763                           Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na \
01764                           *(*atomForce)[j].dbForce[2]);
01765 #endif
01766         }
01767     } else *nforce = 0;
01768
01769     Vnm_tstop(APBS_TIMER_FORCE, "Force timer");
01770
01771     return 1;
01772 }
01773
01774 VPUBLIC void killEnergy() {
01775
01776 #ifndef VAPBSQUIET
01777     Vnm_tprint(1, "No energy arrays to destroy.\n");
01778 #endif
01779
01780 }
01781
01782 VPUBLIC void killForce(Vmem *mem, NOsh *nosh, int nforce[NOSH_MAXCALC],
01783                        AtomForce *atomForce[NOSH_MAXCALC]) {
01784
01785     int i;
01786
01787 #ifndef VAPBSQUIET
01788     Vnm_tprint(1, "Destroying force arrays.\n");
01789 #endif
01790
01791     for (i=0; i<nosh->ncalc; i++) {
01792
01793         if (nforce[i] > 0) Vmem_free(mem, nforce[i], sizeof(AtomForce),
01794                                      (void **)&(atomForce[i]));
01795
01796     }
01797 }
01798
01799 VPUBLIC int writematMG(int rank, NOsh *nosh, PBEparm *pbeparm, Vpmg *pmg) {
01800
01801     char writematstem[VMAX_ARGLEN];
01802     char outpath[VMAX_ARGLEN];
01803     char mxtype[3];
01804     int strlenmax;
01805
01806     if (nosh->bogus) return 1;
01807
01808 #ifdef HAVE_MPI_H
01809     strlenmax = VMAX_ARGLEN-14;
01810     if (strlen(pbeparm->writematstem) > strlenmax) {
01811         Vnm_tprint(2, "  Matrix name (%s) too long (%d char max)!\n",
01812                    pbeparm->writematstem, strlenmax);
01813         Vnm_tprint(2, "  Not writing matrix!\n");
01814         return 0;
01815     }
01816     sprintf(writematstem, "%s-PE%d", pbeparm->writematstem, rank);
01817 #else
01818     strlenmax = (int)(VMAX_ARGLEN)-1;
01819     if ((int)strlen(pbeparm->writematstem) > strlenmax) {
01820         Vnm_tprint(2, "  Matrix name (%s) too long (%d char max)!\n",
01821                    pbeparm->writematstem, strlenmax);
01822         Vnm_tprint(2, "  Not writing matrix!\n");
01823         return 0;
01824     }
01825     if(nosh->ispara == 1){
01826         sprintf(writematstem, "%s-PE%d", pbeparm->writematstem,nosh->proc_rank);
01827     }else{
01828         sprintf(writematstem, "%s", pbeparm->writematstem);
01829     }
01830 #endif
01831
01832     if (pbeparm->writemat == 1) {
01833         strlenmax = VMAX_ARGLEN-5;
01834         if ((int)strlen(pbeparm->writematstem) > strlenmax) {
01835             Vnm_tprint(2, "  Matrix name (%s) too long (%d char max)!\n",
01836                        pbeparm->writematstem, strlenmax);
01837             Vnm_tprint(2, "  Not writing matrix!\n");
01838             return 0;
01839         }
01840         sprintf(outpath, "%s.%s", writematstem, "mat");
01841         mxtype[0] = 'R';
```

```
01842         mxtype[1] = 'S';
01843         mxtype[2] = 'A';
01844         /* Poisson operator only */
01845         if (pbeparm->writematflag == 0) {
01846             Vnm_tprint( 1, "  Writing Poisson operator matrix \
01847 to %s...\n", outpath);
01848 
01849             /* Linearization of Poisson-Boltzmann operator around solution */
01850         } else if (pbeparm->writematflag == 1) {
01851             Vnm_tprint( 1, "  Writing linearization of full \
01852 Poisson-Boltzmann operator matrix to %s...\n", outpath);
01853 
01854         } else {
01855             Vnm_tprint( 2, "  Bogus matrix specification\
01856 (%d)!\n", pbeparm->writematflag);
01857             return 0;
01858         }
01859 
01860         Vnm_tprint(0, "  Printing operator...\n");
01861         //Vpmg_printColComp(pmg, outpath, outpath, mxtype,
01862         //          pbeparm->writematflag);
01863         return 0;
01864 
01865     }
01866 
01867     return 1;
01868 }
01869 
01870 VPUBLIC void storeAtomEnergy(Vpmg *pmg, int icalc, double **atomEnergy,
01871                              int *nenergy){
01872 
01873     Vatom *atom;
01874     Valist *alist;
01875     int i;
01876 
01877     alist = pmg->pbe->alist;
01878     *nenergy = Valist_getNumberAtoms(alist);
01879     *atomEnergy = (double *)Vmem_malloc(pmg->vmem, *nenergy, sizeof(double));
01880 
01881     for (i=0; i<*nenergy; i++) {
01882         atom = Valist_getAtom(alist, i);
01883         (*atomEnergy)[i] = Vpmg_qfAtomEnergy(pmg, atom);
01884     }
01885 }
01886 
01887 VPUBLIC int writedataFlat(
01888                           NOsh *nosh,
01889                           Vcom *com,
01890                           const char *fname,
01891                           double totEnergy[NOSH_MAXCALC],
01892                           double qfEnergy[NOSH_MAXCALC],
01893                           double qmEnergy[NOSH_MAXCALC],
01894                           double dielEnergy[NOSH_MAXCALC],
01895                           int nenergy[NOSH_MAXCALC],
01896                           double *atomEnergy[NOSH_MAXCALC],
01897                           int nforce[NOSH_MAXCALC],
01898                           AtomForce *atomForce[NOSH_MAXCALC]) {
01899 
01900     FILE *file;
01901     time_t now;
01902     int ielec, icalc, i, j;
01903     char *timestring = VNULL;
01904     PBEparm *pbeparm = VNULL;
01905     MGparm *mgparm = VNULL;
01906     double conversion, ltenergy, gtenergy, scalar;
01907 
01908     if (nosh->bogus) return 1;
01909 
01910     /* Initialize some variables */
01911 
01912     icalc = 0;
01913 
01914     file = fopen(fname, "w");
01915     if (file == VNULL) {
01916         Vnm_print(2, "writedataFlat: Problem opening virtual socket %s\n",
01917                   fname);
01918         return 0;
01919     }
01920 
01921     /* Strip the newline character from the date */
01922 
```

```
01923     now = time(VNULL);
01924     timestring = ctime(&now);
01925     fprintf(file,"%s\n", timestring);
01926
01927     for (ielec=0; ielec<nosh->nelec;ielec++) { /* elec loop */
01928
01929         /* Initialize per-elec pointers */
01930
01931         mgparm = nosh->calc[icalc]->mgparm;
01932         pbeparm = nosh->calc[icalc]->pbeparm;
01933
01934         /* Convert from kT/e to kJ/mol */
01935         conversion =  Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na;
01936
01937         fprintf(file,"elec");
01938         if (Vstring_strcasecmp(nosh->elecname[ielec], "") != 0) {
01939             fprintf(file," name %s\n", nosh->elecname[ielec]);
01940         } else fprintf(file, "\n");
01941
01942         switch (mgparm->type) {
01943             case MCT_DUMMY:
01944                 fprintf(file,"    mg-dummy\n");
01945                 break;
01946             case MCT_MANUAL:
01947                 fprintf(file,"    mg-manual\n");
01948                 break;
01949             case MCT_AUTO:
01950                 fprintf(file,"    mg-auto\n");
01951                 break;
01952             case MCT_PARALLEL:
01953                 fprintf(file,"    mg-para\n");
01954                 break;
01955             default:
01956                 break;
01957         }
01958
01959         fprintf(file,"    mol %d\n", pbeparm->molid);
01960         fprintf(file,"    dime %d %d %d\n", mgparm->dime[0], mgparm->dime[1],\
01961                 mgparm->dime[2]);
01962
01963         switch (pbeparm->pbetype) {
01964             case PBE_NPBE:
01965                 fprintf(file,"    npbe\n");
01966                 break;
01967             case PBE_LPBE:
01968                 fprintf(file,"    lpbe\n");
01969                 break;
01970             default:
01971                 break;
01972         }
01973
01974         if (pbeparm->nion > 0) {
01975             for (i=0; i<pbeparm->nion; i++) {
01976                 fprintf(file,"    ion %4.3f %4.3f %4.3f\n",
01977                         pbeparm->ionr[i], pbeparm->ionq[i], pbeparm->ionc[i]);
01978             }
01979         }
01980
01981         fprintf(file,"    pdie %4.3f\n", pbeparm->pdie);
01982         fprintf(file,"    sdie %4.3f\n", pbeparm->sdie);
01983
01984         switch (pbeparm->srfm) {
01985             case 0:
01986                 fprintf(file,"    srfm mol\n");
01987                 fprintf(file,"    srad %4.3f\n", pbeparm->srad);
01988                 break;
01989             case 1:
01990                 fprintf(file,"    srfm smol\n");
01991                 fprintf(file,"    srad %4.3f\n", pbeparm->srad);
01992                 break;
01993             case 2:
01994                 fprintf(file,"    srfm spl2\n");
01995                 fprintf(file,"    srad %4.3f\n", pbeparm->srad);
01996                 break;
01997             default:
01998                 break;
01999         }
02000
02001         switch (pbeparm->bcfl) {
02002             case BCFL_ZERO:
02003                 fprintf(file,"    bcfl zero\n");
```

```
02004                 break;
02005             case BCFL_SDH:
02006                 fprintf(file,"    bcfl sdh\n");
02007                 break;
02008             case BCFL_MDH:
02009                 fprintf(file,"    bcfl mdh\n");
02010                 break;
02011             case BCFL_FOCUS:
02012                 fprintf(file,"    bcfl focus\n");
02013                 break;
02014             case BCFL_MAP:
02015                 fprintf(file,"    bcfl map\n");
02016                 break;
02017             case BCFL_MEM:
02018                 fprintf(file,"    bcfl mem\n");
02019                 break;
02020             default:
02021                 break;
02022         }
02023
02024         fprintf(file,"    temp %4.3f\n", pbeparm->temp);
02025
02026         for (;icalc<=nosh->elec2calc[ielec];icalc++){ /* calc loop */
02027
02028             /* Reinitialize per-calc pointers */
02029             mgparm = nosh->calc[icalc]->mgparm;
02030             pbeparm = nosh->calc[icalc]->pbeparm;
02031
02032             fprintf(file,"    calc\n");
02033             fprintf(file,"        id %i\n", (icalc+1));
02034             fprintf(file,"        grid %4.3f %4.3f %4.3f\n",
02035                     mgparm->grid[0], mgparm->grid[1], mgparm->grid[2]);
02036             fprintf(file,"        glen %4.3f %4.3f %4.3f\n",
02037                     mgparm->glen[0], mgparm->glen[1], mgparm->glen[2]);
02038
02039             if (pbeparm->calcenergy == PCE_TOTAL) {
02040                 fprintf(file,"        totEnergy %1.12E kJ/mol\n",
02041                         (totEnergy[icalc]*conversion));
02042             } if (pbeparm->calcenergy == PCE_COMPS) {
02043                     fprintf(file,"        totEnergy %1.12E kJ/mol\n",
02044                         (totEnergy[icalc]*conversion));
02045                 fprintf(file,"        qfEnergy %1.12E kJ/mol\n",
02046                         (0.5*qfEnergy[icalc]*conversion));
02047                 fprintf(file,"        qmEnergy %1.12E kJ/mol\n",
02048                         (qmEnergy[icalc]*conversion));
02049                 fprintf(file,"        dielEnergy %1.12E kJ/mol\n",
02050                         (dielEnergy[icalc]*conversion));
02051                 for (i=0; i<nenergy[icalc]; i++){
02052                     fprintf(file,"        atom %i %1.12E kJ/mol\n", i,
02053                             (0.5*atomEnergy[icalc][i]*conversion));
02054
02055                 }
02056             }
02057
02058             if (pbeparm->calcforce == PCF_TOTAL) {
02059                 fprintf(file,"        qfForce %1.12E %1.12E %1.12E kJ/mol/A\n",
02060                         (atomForce[icalc][0].qfForce[0]*conversion),
02061                             (atomForce[icalc][0].qfForce[1]*conversion),
02062                             (atomForce[icalc][0].qfForce[2]*conversion));
02063                 fprintf(file,"        ibForce %1.12E %1.12E %1.12E kJ/mol/A\n",
02064                         (atomForce[icalc][0].ibForce[0]*conversion),
02065                             (atomForce[icalc][0].ibForce[1]*conversion),
02066                             (atomForce[icalc][0].ibForce[2]*conversion));
02067                 fprintf(file,"        dbForce %1.12E %1.12E %1.12E kJ/mol/A\n",
02068                         (atomForce[icalc][0].dbForce[0]*conversion),
02069                             (atomForce[icalc][0].dbForce[1]*conversion),
02070                             (atomForce[icalc][0].dbForce[2]*conversion));
02071             }
02072             fprintf(file,"    end\n");
02073         }
02074
02075         fprintf(file,"end\n");
02076     }
02077
02078 /* Handle print energy statements */
02079
02080 for (i=0; i<nosh->nprint; i++) {
02081
02082     if (nosh->printwhat[i] == NPT_ENERGY) {
02083
02084         fprintf(file,"print energy");
```

```
02085         fprintf(file," %d", nosh->printcalc[i][0]+1);
02086
02087         for (j=1; j<nosh->printnarg[i]; j++) {
02088             if (nosh->printop[i][j-1] == 0) fprintf(file," +");
02089             else if (nosh->printop[i][j-1] == 1) fprintf(file, " -");
02090             fprintf(file, " %d", nosh->printcalc[i][j]+1);
02091         }
02092
02093         fprintf(file, "\n");
02094         icalc = nosh->elec2calc[nosh->printcalc[i][0]];
02095
02096         ltenergy = Vunit_kb * (1e-3) * Vunit_Na * \
02097             nosh->calc[icalc]->pbeparm->temp * totEnergy[icalc];
02098
02099         for (j=1; j<nosh->printnarg[i]; j++) {
02100             icalc = nosh->elec2calc[nosh->printcalc[i][j]];
02101             /* Add or subtract? */
02102             if (nosh->printop[i][j-1] == 0) scalar = 1.0;
02103             else if (nosh->printop[i][j-1] == 1) scalar = -1.0;
02104             /* Accumulate */
02105             ltenergy += (scalar * Vunit_kb * (1e-3) * Vunit_Na *
02106                         nosh->calc[icalc]->pbeparm->temp * totEnergy[icalc]);
02107
02108             Vcom_reduce(com, &ltenergy, &gtenergy, 1, 2, 0);
02109
02110         }
02111         fprintf(file,"    localEnergy %1.12E kJ/mol\n", \
02112                 ltenergy);
02113         fprintf(file,"    globalEnergy %1.12E kJ/mol\nend\n", \
02114                 gtenergy);
02115     }
02116 }
02117
02118 fclose(file);
02119
02120 return 1;
02121 }
02122
02123 VPUBLIC int writedataXML(NOsh *nosh, Vcom *com, const char *fname,
02124                          double totEnergy[NOSH_MAXCALC],
02125                          double qfEnergy[NOSH_MAXCALC],
02126                          double qmEnergy[NOSH_MAXCALC],
02127                          double dielEnergy[NOSH_MAXCALC],
02128                          int nenergy[NOSH_MAXCALC],
02129                          double *atomEnergy[NOSH_MAXCALC],
02130                                     int nforce[NOSH_MAXCALC],
02131                          AtomForce *atomForce[NOSH_MAXCALC]) {
02132
02133     FILE *file;
02134     time_t now;
02135     int ielec, icalc, i, j;
02136     char *timestring = VNULL;
02137     char *c = VNULL;
02138     PBEparm *pbeparm = VNULL;
02139     MGparm *mgparm = VNULL;
02140     double conversion, ltenergy, gtenergy, scalar;
02141
02142     if (nosh->bogus) return 1;
02143
02144     /* Initialize some variables */
02145
02146     icalc = 0;
02147
02148     file = fopen(fname, "w");
02149     if (file == VNULL) {
02150         Vnm_print(2, "writedataXML: Problem opening virtual socket %s\n",
02151                   fname);
02152         return 0;
02153     }
02154
02155     fprintf(file,"<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
02156     fprintf(file,"<APBS>\n");
02157
02158     /* Strip the newline character from the date */
02159
02160     now = time(VNULL);
02161     timestring = ctime(&now);
02162     for(c = timestring; *c != '\n'; c++);
02163     *c = '\0';
02164     fprintf(file,"    <date>%s</date>\n", timestring);
02165
```

```
02166     for (ielec=0; ielec<nosh->nelec;ielec++){ /* elec loop */
02167
02168         /* Initialize per-elec pointers */
02169
02170         mgparm = nosh->calc[icalc]->mgparm;
02171         pbeparm = nosh->calc[icalc]->pbeparm;
02172
02173         /* Convert from kT/e to kJ/mol */
02174         conversion =  Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na;
02175
02176         fprintf(file,"    <elec>\n");
02177         if (Vstring_strcasecmp(nosh->elecname[ielec], "") != 0) {
02178             fprintf(file,"      <name>%s</name>\n", nosh->elecname[ielec]);
02179         }
02180
02181         switch (mgparm->type) {
02182             case MCT_DUMMY:
02183                 fprintf(file,"      <type>mg-dummy</type>\n");
02184                 break;
02185             case MCT_MANUAL:
02186                 fprintf(file,"      <type>mg-manual</type>\n");
02187                 break;
02188             case MCT_AUTO:
02189                 fprintf(file,"      <type>mg-auto</type>\n");
02190                 break;
02191             case MCT_PARALLEL:
02192                 fprintf(file,"      <type>mg-para</type>\n");
02193                 break;
02194             default:
02195                 break;
02196         }
02197
02198         fprintf(file,"      <molid>%d</molid>\n", pbeparm->molid);
02199         fprintf(file,"      <nx>%d</nx>\n", mgparm->dime[0]);
02200         fprintf(file,"      <ny>%d</ny>\n", mgparm->dime[1]);
02201         fprintf(file,"      <nz>%d</nz>\n", mgparm->dime[2]);
02202
02203         switch (pbeparm->pbetype) {
02204             case PBE_NPBE:
02205                 fprintf(file,"      <pbe>npbe</pbe>\n");
02206                 break;
02207             case PBE_LPBE:
02208                 fprintf(file,"      <pbe>lpbe</pbe>\n");
02209                 break;
02210             default:
02211                 break;
02212         }
02213
02214         if (pbeparm->nion > 0) {
02215             for (i=0; i<pbeparm->nion; i++) {
02216                 fprintf(file, "      <ion>\n");
02217                 fprintf(file,"          <radius>%4.3f A</radius>\n",
02218                         pbeparm->ionr[i]);
02219                 fprintf(file,"          <charge>%4.3f A</charge>\n",
02220                         pbeparm->ionq[i]);
02221                 fprintf(file,"          <concentration>%4.3f M</concentration>\n",
02222                         pbeparm->ionc[i]);
02223                 fprintf(file, "      </ion>\n");
02224
02225             }
02226         }
02227
02228         fprintf(file,"      <pdie>%4.3f</pdie>\n", pbeparm->pdie);
02229         fprintf(file,"      <sdie>%4.3f</sdie>\n", pbeparm->sdie);
02230
02231         switch (pbeparm->srfm) {
02232             case 0:
02233                 fprintf(file,"      <srfm>mol</srfm>\n");
02234                 fprintf(file,"      <srad>%4.3f</srad>\n", pbeparm->srad);
02235                 break;
02236             case 1:
02237                 fprintf(file,"      <srfm>smol</srfm>\n");
02238                 fprintf(file,"      <srad>%4.3f</srad>\n", pbeparm->srad);
02239                 break;
02240             case 2:
02241                 fprintf(file,"      <srfm>spl2</srfm>\n");
02242                 break;
02243             default:
02244                 break;
02245         }
02246
```

```
02247         switch (pbeparm->bcfl) {
02248             case BCFL_ZERO:
02249                 fprintf(file,"      <bcfl>zero</bcfl>\n");
02250                 break;
02251             case BCFL_SDH:
02252                 fprintf(file,"      <bcfl>sdh</bcfl>\n");
02253                 break;
02254             case BCFL_MDH:
02255                 fprintf(file,"      <bcfl>mdh</bcfl>\n");
02256                 break;
02257             case BCFL_FOCUS:
02258                 fprintf(file,"      <bcfl>focus</bcfl>\n");
02259                 break;
02260             case BCFL_MAP:
02261                 fprintf(file,"      <bcfl>map</bcfl>\n");
02262                 break;
02263             case BCFL_MEM:
02264                 fprintf(file,"      <bcfl>mem</bcfl>\n");
02265                 break;
02266             default:
02267                 break;
02268         }
02269 
02270         fprintf(file,"      <temp>%4.3f K</temp>\n", pbeparm->temp);
02271 
02272         for (;icalc<=nosh->elec2calc[ielec];icalc++){ /* calc loop */
02273 
02274             /* Reinitialize per-calc pointers */
02275             mgparm = nosh->calc[icalc]->mgparm;
02276             pbeparm = nosh->calc[icalc]->pbeparm;
02277 
02278             fprintf(file,"      <calc>\n");
02279             fprintf(file,"          <id>%i</id>\n", (icalc+1));
02280             fprintf(file,"          <hx>%4.3f A</hx>\n", mgparm->grid[0]);
02281             fprintf(file,"          <hy>%4.3f A</hy>\n", mgparm->grid[1]);
02282             fprintf(file,"          <hz>%4.3f A</hz>\n", mgparm->grid[2]);
02283             fprintf(file,"          <xlen>%4.3f A</xlen>\n", mgparm->glen[0]);
02284             fprintf(file,"          <ylen>%4.3f A</ylen>\n", mgparm->glen[1]);
02285             fprintf(file,"          <zlen>%4.3f A</zlen>\n", mgparm->glen[2]);
02286 
02287             if (pbeparm->calcenergy == PCE_TOTAL) {
02288                 fprintf(file,"          <totEnergy>%1.12E kJ/mol</totEnergy>\n",
02289                         (totEnergy[icalc]*conversion));
02290             } else if (pbeparm->calcenergy == PCE_COMPS) {
02291                         fprintf(file,"          <totEnergy>%1.12E kJ/mol</totEnergy>\n",
02292                         (totEnergy[icalc]*conversion));
02293                 fprintf(file,"          <qfEnergy>%1.12E kJ/mol</qfEnergy>\n",
02294                         (0.5*qfEnergy[icalc]*conversion));
02295                 fprintf(file,"          <qmEnergy>%1.12E kJ/mol</qmEnergy>\n",
02296                         (qmEnergy[icalc]*conversion));
02297                 fprintf(file,"          <dielEnergy>%1.12E kJ/mol</dielEnergy>\n",
02298                         (dielEnergy[icalc]*conversion));
02299                 for (i=0; i<nenergy[icalc]; i++){
02300                     fprintf(file,"          <atom>\n");
02301                     fprintf(file,"              <id>%i</id>\n", i+1);
02302                     fprintf(file,"              <energy>%1.12E kJ/mol</energy>\n",
02303                             (0.5*atomEnergy[icalc][i]*conversion));
02304                     fprintf(file,"          </atom>\n");
02305                 }
02306             }
02307 
02308 
02309             if (pbeparm->calcforce == PCF_TOTAL) {
02310                     fprintf(file,"          <qfforce_x>%1.12E</qfforce_x>\n",
02311                     atomForce[icalc][0].qfForce[0]*conversion);
02312                 fprintf(file,"          <qfforce_y>%1.12E</qfforce_y>\n",
02313                     atomForce[icalc][0].qfForce[1]*conversion);
02314                 fprintf(file,"          <qfforce_z>%1.12E</qfforce_z>\n",
02315                     atomForce[icalc][0].qfForce[2]*conversion);
02316                 fprintf(file,"          <ibforce_x>%1.12E</ibforce_x>\n",
02317                     atomForce[icalc][0].ibForce[0]*conversion);
02318                 fprintf(file,"          <ibforce_y>%1.12E</ibforce_y>\n",
02319                     atomForce[icalc][0].ibForce[1]*conversion);
02320                 fprintf(file,"          <ibforce_z>%1.12E</ibforce_z>\n",
02321                     atomForce[icalc][0].ibForce[2]*conversion);
02322                 fprintf(file,"          <dbforce_x>%1.12E</dbforce_x>\n",
02323                     atomForce[icalc][0].dbForce[0]*conversion);
02324                 fprintf(file,"          <dbforce_y>%1.12E</dbforce_y>\n",
02325                     atomForce[icalc][0].dbForce[1]*conversion);
02326                 fprintf(file,"          <dbforce_z>%1.12E</dbforce_z>\n",
02327                     atomForce[icalc][0].dbForce[2]*conversion);
```

```
02328             }
02329
02330             fprintf(file,"      </calc>\n");
02331         }
02332
02333         fprintf(file,"    </elec>\n");
02334     }
02335
02336 /* Handle print energy statements */
02337
02338 for (i=0; i<nosh->nprint; i++) {
02339
02340     if (nosh->printwhat[i] == NPT_ENERGY) {
02341
02342         fprintf(file,"    <printEnergy>\n");
02343         fprintf(file,"        <equation>%d", nosh->printcalc[i][0]+1);
02344
02345         for (j=1; j<nosh->printnarg[i]; j++) {
02346             if (nosh->printop[i][j-1] == 0) fprintf(file," +");
02347             else if (nosh->printop[i][j-1] == 1) fprintf(file, " -");
02348             fprintf(file, " %d", nosh->printcalc[i][j] +1);
02349         }
02350
02351         fprintf(file, "</equation>\n");
02352         icalc = nosh->elec2calc[nosh->printcalc[i][0]];
02353
02354         ltenergy = Vunit_kb * (1e-3) * Vunit_Na * \
02355             nosh->calc[icalc]->pbeparm->temp * totEnergy[icalc];
02356
02357         for (j=1; j<nosh->printnarg[i]; j++) {
02358             icalc = nosh->elec2calc[nosh->printcalc[i][j]];
02359             /* Add or subtract? */
02360             if (nosh->printop[i][j-1] == 0) scalar = 1.0;
02361             else if (nosh->printop[i][j-1] == 1) scalar = -1.0;
02362             /* Accumulate */
02363             ltenergy += (scalar * Vunit_kb * (1e-3) * Vunit_Na *
02364                          nosh->calc[icalc]->pbeparm->temp * totEnergy[icalc]);
02365         }
02366         Vcom_reduce(com, &ltenergy, &gtenergy, 1, 2, 0);
02367         fprintf(file,"        <localEnergy>%1.12E kJ/mol</localEnergy>\n", \
02368                 ltenergy);
02369         fprintf(file,"        <globalEnergy>%1.12E kJ/mol</globalEnergy>\n", \
02370                 gtenergy);
02371
02372         fprintf(file,"    </printEnergy>\n");
02373     }
02374 }
02375
02376 /* Add ending tags and close the file */
02377 fprintf(file,"</APBS>\n");
02378 fclose(file);
02379
02380 return 1;
02381 }
02382
02383 VPUBLIC int writedataMG(int rank,
02384                         NOsh *nosh,
02385                         PBEparm *pbeparm,
02386                         Vpmg *pmg
02387                        ) {
02388
02389     char writestem[VMAX_ARGLEN];
02390     char outpath[VMAX_ARGLEN];
02391     char title[72];
02392     int i,
02393         nx,
02394         ny,
02395         nz,
02396         natoms;
02397     double hx,
02398            hy,
02399            hzed,
02400            xcent,
02401            ycent,
02402            zcent,
02403            xmin,
02404            ymin,
02405            zmin;
02406
02407     Vgrid *grid;
02408     Vio *sock;
```

```
02409
02410     if (nosh->bogus) return 1;
02411
02412     for (i=0; i<pbeparm->numwrite; i++) {
02413
02414         nx = pmg->pmgp->nx;
02415         ny = pmg->pmgp->ny;
02416         nz = pmg->pmgp->nz;
02417         hx = pmg->pmgp->hx;
02418         hy = pmg->pmgp->hy;
02419         hzed = pmg->pmgp->hzed;
02420
02421         switch (pbeparm->writetype[i]) {
02422
02423             case VDT_CHARGE:
02424
02425                 Vnm_tprint(1, "  Writing charge distribution to ");
02426                 xcent = pmg->pmgp->xcent;
02427                 ycent = pmg->pmgp->ycent;
02428                 zcent = pmg->pmgp->zcent;
02429                 xmin = xcent - 0.5*(nx-1)*hx;
02430                 ymin = ycent - 0.5*(ny-1)*hy;
02431                 zmin = zcent - 0.5*(nz-1)*hzed;
02432                 VASSERT(Vpmg_fillArray(pmg, pmg->rwork, VDT_CHARGE, 0.0,
02433                                        pbeparm->pbetype, pbeparm));
02434                 sprintf(title, "CHARGE DISTRIBUTION (e)");
02435                 break;
02436
02437             case VDT_POT:
02438
02439                 Vnm_tprint(1, "  Writing potential to ");
02440                 xcent = pmg->pmgp->xcent;
02441                 ycent = pmg->pmgp->ycent;
02442                 zcent = pmg->pmgp->zcent;
02443                 xmin = xcent - 0.5*(nx-1)*hx;
02444                 ymin = ycent - 0.5*(ny-1)*hy;
02445                 zmin = zcent - 0.5*(nz-1)*hzed;
02446                 VASSERT(Vpmg_fillArray(pmg, pmg->rwork, VDT_POT, 0.0,
02447                                        pbeparm->pbetype, pbeparm));
02448                 sprintf(title, "POTENTIAL (kT/e)");
02449                 break;
02450
02451             case VDT_SMOL:
02452
02453                 Vnm_tprint(1, "  Writing molecular accessibility to ");
02454                 xcent = pmg->pmgp->xcent;
02455                 ycent = pmg->pmgp->ycent;
02456                 zcent = pmg->pmgp->zcent;
02457                 xmin = xcent - 0.5*(nx-1)*hx;
02458                 ymin = ycent - 0.5*(ny-1)*hy;
02459                 zmin = zcent - 0.5*(nz-1)*hzed;
02460                 VASSERT(Vpmg_fillArray(pmg, pmg->rwork, VDT_SMOL,
02461                                        pbeparm->srad, pbeparm->pbetype, pbeparm));
02462                 sprintf(title,
02463                         "SOLVENT ACCESSIBILITY -- MOLECULAR (%4.3f PROBE)",
02464                         pbeparm->srad);
02465                 break;
02466
02467             case VDT_SSPL:
02468
02469                 Vnm_tprint(1, "  Writing spline-based accessibility to ");
02470                 xcent = pmg->pmgp->xcent;
02471                 ycent = pmg->pmgp->ycent;
02472                 zcent = pmg->pmgp->zcent;
02473                 xmin = xcent - 0.5*(nx-1)*hx;
02474                 ymin = ycent - 0.5*(ny-1)*hy;
02475                 zmin = zcent - 0.5*(nz-1)*hzed;
02476                 VASSERT(Vpmg_fillArray(pmg, pmg->rwork, VDT_SSPL,
02477                                        pbeparm->swin, pbeparm->pbetype, pbeparm));
02478                 sprintf(title,
02479                         "SOLVENT ACCESSIBILITY -- SPLINE (%4.3f WINDOW)",
02480                         pbeparm->swin);
02481                 break;
02482
02483             case VDT_VDW:
02484
02485                 Vnm_tprint(1, "  Writing van der Waals accessibility to ");
02486                 xcent = pmg->pmgp->xcent;
02487                 ycent = pmg->pmgp->ycent;
02488                 zcent = pmg->pmgp->zcent;
02489                 xmin = xcent - 0.5*(nx-1)*hx;
```

```
02490                 ymin = ycent - 0.5*(ny-1)*hy;
02491                 zmin = zcent - 0.5*(nz-1)*hzed;
02492                 VASSERT(Vpmg_fillArray(pmg, pmg->rwork, VDT_VDW, 0.0,
02493                                        pbeparm->pbetype, pbeparm));
02494                 sprintf(title, "SOLVENT ACCESSIBILITY -- VAN DER WAALS");
02495                 break;
02496 
02497             case VDT_IVDW:
02498 
02499                 Vnm_tprint(1, "  Writing ion accessibility to ");
02500                 xcent = pmg->pmgp->xcent;
02501                 ycent = pmg->pmgp->ycent;
02502                 zcent = pmg->pmgp->zcent;
02503                 xmin = xcent - 0.5*(nx-1)*hx;
02504                 ymin = ycent - 0.5*(ny-1)*hy;
02505                 zmin = zcent - 0.5*(nz-1)*hzed;
02506                 VASSERT(Vpmg_fillArray(pmg, pmg->rwork, VDT_IVDW,
02507                                        pmg->pbe->maxIonRadius, pbeparm->pbetype, pbeparm));
02508                 sprintf(title,
02509                         "ION ACCESSIBILITY -- SPLINE (%4.3f RADIUS)",
02510                         pmg->pbe->maxIonRadius);
02511                 break;
02512 
02513             case VDT_LAP:
02514 
02515                 Vnm_tprint(1, "  Writing potential Laplacian to ");
02516                 xcent = pmg->pmgp->xcent;
02517                 ycent = pmg->pmgp->ycent;
02518                 zcent = pmg->pmgp->zcent;
02519                 xmin = xcent - 0.5*(nx-1)*hx;
02520                 ymin = ycent - 0.5*(ny-1)*hy;
02521                 zmin = zcent - 0.5*(nz-1)*hzed;
02522                 VASSERT(Vpmg_fillArray(pmg, pmg->rwork, VDT_LAP, 0.0,
02523                                        pbeparm->pbetype, pbeparm));
02524                 sprintf(title,
02525                         "POTENTIAL LAPLACIAN (kT/e/A^2)");
02526                 break;
02527 
02528             case VDT_EDENS:
02529 
02530                 Vnm_tprint(1, "  Writing energy density to ");
02531                 xcent = pmg->pmgp->xcent;
02532                 ycent = pmg->pmgp->ycent;
02533                 zcent = pmg->pmgp->zcent;
02534                 xmin = xcent - 0.5*(nx-1)*hx;
02535                 ymin = ycent - 0.5*(ny-1)*hy;
02536                 zmin = zcent - 0.5*(nz-1)*hzed;
02537                 VASSERT(Vpmg_fillArray(pmg, pmg->rwork, VDT_EDENS, 0.0,
02538                                        pbeparm->pbetype, pbeparm));
02539                 sprintf(title, "ENERGY DENSITY (kT/e/A)^2");
02540                 break;
02541 
02542             case VDT_NDENS:
02543 
02544                 Vnm_tprint(1, "  Writing number density to ");
02545                 xcent = pmg->pmgp->xcent;
02546                 ycent = pmg->pmgp->ycent;
02547                 zcent = pmg->pmgp->zcent;
02548                 xmin = xcent - 0.5*(nx-1)*hx;
02549                 ymin = ycent - 0.5*(ny-1)*hy;
02550                 zmin = zcent - 0.5*(nz-1)*hzed;
02551                 VASSERT(Vpmg_fillArray(pmg, pmg->rwork, VDT_NDENS, 0.0,
02552                                        pbeparm->pbetype, pbeparm));
02553                 sprintf(title,
02554                         "ION NUMBER DENSITY (M)");
02555                 break;
02556 
02557             case VDT_QDENS:
02558 
02559                 Vnm_tprint(1, "  Writing charge density to ");
02560                 xcent = pmg->pmgp->xcent;
02561                 ycent = pmg->pmgp->ycent;
02562                 zcent = pmg->pmgp->zcent;
02563                 xmin = xcent - 0.5*(nx-1)*hx;
02564                 ymin = ycent - 0.5*(ny-1)*hy;
02565                 zmin = zcent - 0.5*(nz-1)*hzed;
02566                 VASSERT(Vpmg_fillArray(pmg, pmg->rwork, VDT_QDENS, 0.0,
02567                                        pbeparm->pbetype, pbeparm));
02568                 sprintf(title,
02569                         "ION CHARGE DENSITY (e_c * M)");
02570                 break;
```

```
02571
02572             case VDT_DIELX:
02573
02574                 Vnm_tprint(1, "  Writing x-shifted dielectric map to ");
02575                 xcent = pmg->pmgp->xcent + 0.5*hx;
02576                 ycent = pmg->pmgp->ycent;
02577                 zcent = pmg->pmgp->zcent;
02578                 xmin = xcent - 0.5*(nx-1)*hx;
02579                 ymin = ycent - 0.5*(ny-1)*hy;
02580                 zmin = zcent - 0.5*(nz-1)*hzed;
02581                 VASSERT(Vpmg_fillArray(pmg, pmg->rwork, VDT_DIELX, 0.0,
02582                                        pbeparm->pbetype, pbeparm));
02583                 sprintf(title,
02584                         "X-SHIFTED DIELECTRIC MAP");
02585                 break;
02586
02587             case VDT_DIELY:
02588
02589                 Vnm_tprint(1, "  Writing y-shifted dielectric map to ");
02590                 xcent = pmg->pmgp->xcent;
02591                 ycent = pmg->pmgp->ycent + 0.5*hy;
02592                 zcent = pmg->pmgp->zcent;
02593                 xmin = xcent - 0.5*(nx-1)*hx;
02594                 ymin = ycent - 0.5*(ny-1)*hy;
02595                 zmin = zcent - 0.5*(nz-1)*hzed;
02596                 VASSERT(Vpmg_fillArray(pmg, pmg->rwork, VDT_DIELY, 0.0,
02597                                        pbeparm->pbetype, pbeparm));
02598                 sprintf(title,
02599                         "Y-SHIFTED DIELECTRIC MAP");
02600                 break;
02601
02602             case VDT_DIELZ:
02603
02604                 Vnm_tprint(1, "  Writing z-shifted dielectric map to ");
02605                 xcent = pmg->pmgp->xcent;
02606                 ycent = pmg->pmgp->ycent;
02607                 zcent = pmg->pmgp->zcent + 0.5*hzed;
02608                 xmin = xcent - 0.5*(nx-1)*hx;
02609                 ymin = ycent - 0.5*(ny-1)*hy;
02610                 zmin = zcent - 0.5*(nz-1)*hzed;
02611                 VASSERT(Vpmg_fillArray(pmg, pmg->rwork, VDT_DIELZ, 0.0,
02612                                        pbeparm->pbetype, pbeparm));
02613                 sprintf(title,
02614                         "Z-SHIFTED DIELECTRIC MAP");
02615                 break;
02616
02617             case VDT_KAPPA:
02618
02619                 Vnm_tprint(1, "  Writing kappa map to ");
02620                 xcent = pmg->pmgp->xcent;
02621                 ycent = pmg->pmgp->ycent;
02622                 zcent = pmg->pmgp->zcent;
02623                 xmin = xcent - 0.5*(nx-1)*hx;
02624                 ymin = ycent - 0.5*(ny-1)*hy;
02625                 zmin = zcent - 0.5*(nz-1)*hzed;
02626                 VASSERT(Vpmg_fillArray(pmg, pmg->rwork, VDT_KAPPA, 0.0,
02627                                        pbeparm->pbetype, pbeparm));
02628                 sprintf(title,
02629                         "KAPPA MAP");
02630                 break;
02631
02632             case VDT_ATOMPOT:
02633
02634                 Vnm_tprint(1, "  Writing atom potentials to ");
02635                 xcent = pmg->pmgp->xcent;
02636                 ycent = pmg->pmgp->ycent;
02637                 zcent = pmg->pmgp->zcent;
02638                 xmin = xcent - 0.5*(nx-1)*hx;
02639                 ymin = ycent - 0.5*(ny-1)*hy;
02640                 zmin = zcent - 0.5*(nz-1)*hzed;
02641                 VASSERT(Vpmg_fillArray(pmg, pmg->rwork, VDT_ATOMPOT, 0.0,
02642                                        pbeparm->pbetype, pbeparm));
02643                 sprintf(title,
02644                         "ATOM POTENTIALS");
02645                 break;
02646             default:
02647
02648                 Vnm_tprint(2, "Invalid data type for writing!\n");
02649                 return 0;
02650         }
02651
```

```
02652
02653 #ifdef HAVE_MPI_H
02654         sprintf(writestem, "%s-PE%d", pbeparm->writestem[i], rank);
02655 #else
02656         if(nosh->ispara){
02657             sprintf(writestem, "%s-PE%d", pbeparm->writestem[i],nosh->proc_rank);
02658         }else{
02659             sprintf(writestem, "%s", pbeparm->writestem[i]);
02660         }
02661 #endif
02662
02663         switch (pbeparm->writefmt[i]) {
02664
02665             case VDF_DX:
02666                 sprintf(outpath, "%s.%s", writestem, "dx");
02667                 Vnm_tprint(1, "%s\n", outpath);
02668                 grid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin,
02669                                   pmg->rwork);
02670                 Vgrid_writeDX(grid, "FILE", "ASC", VNULL, outpath, title,
02671                               pmg->pvec);
02672                 Vgrid_dtor(&grid);
02673                 break;
02674
02675             case VDF_DXBIN:
02676         sprintf(outpath, "%s.%s", writestem, "dxbin");
02677         Vnm_tprint(1, "%s\n", outpath);
02678         grid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin,
02679                           pmg->rwork);
02680         //TODO: write Vgrid_writeDXBIN method
02681         Vgrid_writeDXBIN(grid, "FILE", "ASC", VNULL, outpath, title,
02682                 pmg->pvec);
02683         Vgrid_dtor(&grid);
02684         break;
02685
02686             case VDF_AVS:
02687                 sprintf(outpath, "%s.%s", writestem, "ucd");
02688                 Vnm_tprint(1, "%s\n", outpath);
02689                 Vnm_tprint(2, "Sorry, AVS format isn't supported for \
02690 uniform meshes yet!\n");
02691                 break;
02692
02693             case VDF_MCSF:
02694                 sprintf(outpath, "%s.%s", writestem, "mcsf");
02695                 Vnm_tprint(1, "%s\n", outpath);
02696                 Vnm_tprint(2, "Sorry, MCSF format isn't supported for \
02697                             uniform meshes yet!\n");
02698                 break;
02699
02700             case VDF_UHBD:
02701                 sprintf(outpath, "%s.%s", writestem, "grd");
02702                 Vnm_tprint(1, "%s\n", outpath);
02703                 grid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin,
02704                                   pmg->rwork);
02705                 Vgrid_writeUHBD(grid, "FILE", "ASC", VNULL, outpath, title,
02706                                 pmg->pvec);
02707                 Vgrid_dtor(&grid);
02708                 break;
02709
02710             case VDF_GZ:
02711                 sprintf(outpath, "%s.%s", writestem, "dx.gz");
02712                 Vnm_tprint(1, "%s\n", outpath);
02713                 grid = Vgrid_ctor(nx, ny, nz, hx, hy, hzed, xmin, ymin, zmin,
02714                                   pmg->rwork);
02715                 Vgrid_writeGZ(grid, "FILE", "ASC", VNULL, outpath, title,
02716                               pmg->pvec);
02717                 Vgrid_dtor(&grid);
02718                 break;
02719             case VDF_FLAT:
02720                 sprintf(outpath, "%s.%s", writestem, "txt");
02721                 Vnm_tprint(1, "%s\n", outpath);
02722                 Vnm_print(0, "routines:  Opening virtual socket...\n");
02723                 sock = Vio_ctor("FILE","ASC",VNULL,outpath,"w");
02724                 if (sock == VNULL) {
02725                     Vnm_print(2, "routines:  Problem opening virtual socket %s\n",
02726                               outpath);
02727                     return 0;
02728                 }
02729                 if (Vio_connect(sock, 0) < 0) {
02730                     Vnm_print(2, "routines: Problem connecting virtual socket %s\n",
02731                               outpath);
02732                     return 0;
```

```
02733                 }
02734                 Vio_printf(sock, "# Data from %s\n", PACKAGE_STRING);
02735                 Vio_printf(sock, "# \n");
02736                 Vio_printf(sock, "# %s\n", title);
02737                 Vio_printf(sock, "# \n");
02738                 natoms = pmg->pbe->alist[pbeparm->molid-1].number;
02739                 for(i=0;i<natoms;i++)
02740                     Vio_printf(sock, "%12.6e\n", pmg->rwork[i]);
02741                 break;
02742             default:
02743                 Vnm_tprint(2, "Bogus data format (%d)!\n",
02744                           pbeparm->writefmt[i]);
02745                 break;
02746         }
02747
02748     }
02749
02750     return 1;
02751 }
02752
02753 VPUBLIC double returnEnergy(Vcom *com,
02754                             NOsh *nosh,
02755                             double totEnergy[NOSH_MAXCALC],
02756                             int iprint
02757                            ){
02758
02759     int iarg,
02760         calcid;
02761     double ltenergy,
02762            scalar;
02763
02764     calcid = nosh->elec2calc[nosh->printcalc[iprint][0]];
02765     if (nosh->calc[calcid]->pbeparm->calcenergy != PCE_NO) {
02766         ltenergy = Vunit_kb * (1e-3) * Vunit_Na *
02767         nosh->calc[calcid]->pbeparm->temp * totEnergy[calcid];
02768     } else {
02769         Vnm_tprint( 2, " No energy available in Calculation %d\n", calcid+1);
02770         return 0.0;
02771     }
02772     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++){
02773         calcid = nosh->elec2calc[nosh->printcalc[iprint][iarg]];
02774         /* Add or substract */
02775         if (nosh->printop[iprint][iarg-1] == 0) scalar = 1.0;
02776         else if (nosh->printop[iprint][iarg-1] == 1) scalar = -1.0;
02777         /* Accumulate */
02778         ltenergy += (scalar * Vunit_kb * (1e-3) * Vunit_Na *
02779                      nosh->calc[calcid]->pbeparm->temp * totEnergy[calcid]);
02780     }
02781
02782     return ltenergy;
02783 }
02784
02785 VPUBLIC int printEnergy(Vcom *com,
02786                         NOsh *nosh,
02787                         double totEnergy[NOSH_MAXCALC],
02788                         int iprint
02789                        ) {
02790
02791     int iarg,
02792         calcid;
02793     double ltenergy,
02794            gtenergy,
02795            scalar;
02796
02797     Vnm_tprint( 2, "Warning: The 'energy' print keyword is deprecated.\n" \
02798                    "         Use eilecEnergy for electrostatics energy calcs.\n\n");
02799
02800     if (Vstring_strcasecmp(nosh->elecname[nosh->printcalc[iprint][0]], "") == 0){
02801         Vnm_tprint( 1, "print energy %d ", nosh->printcalc[iprint][0]+1);
02802     } else {
02803         Vnm_tprint( 1, "print energy %d (%s) ", nosh->printcalc[iprint][0]+1,
02804                     nosh->elecname[nosh->printcalc[iprint][0]]);
02805     }
02806     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
02807         if (nosh->printop[iprint][iarg-1] == 0)
02808             Vnm_tprint(1, "+ ");
02809         else if (nosh->printop[iprint][iarg-1] == 1)
02810             Vnm_tprint(1, "- ");
02811         else {
02812             Vnm_tprint( 2, "Undefined PRINT operation!\n");
02813             return 0;
```

```
02814         }
02815         if (Vstring_strcasecmp(nosh->elecname[nosh->printcalc[iprint][iarg]],
02816                                "") == 0) {
02817             Vnm_tprint( 1, "%d ", nosh->printcalc[iprint][iarg]+1);
02818         } else {
02819             Vnm_tprint( 1, "%d (%s) ", nosh->printcalc[iprint][iarg]+1,
02820                         nosh->elecname[nosh->printcalc[iprint][iarg]]);
02821         }
02822     }
02823     Vnm_tprint(1, "end\n");
02824     calcid = nosh->elec2calc[nosh->printcalc[iprint][0]];
02825     if (nosh->calc[calcid]->pbeparm->calcenergy != PCE_NO) {
02826         ltenergy = Vunit_kb * (1e-3) * Vunit_Na *
02827         nosh->calc[calcid]->pbeparm->temp * totEnergy[calcid];
02828     } else {
02829         Vnm_tprint( 2, "  Didn't calculate energy in Calculation \
02830 #%d\n", calcid+1);
02831         return 0;
02832     }
02833     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
02834         calcid = nosh->elec2calc[nosh->printcalc[iprint][iarg]];
02835         /* Add or subtract? */
02836         if (nosh->printop[iprint][iarg-1] == 0) scalar = 1.0;
02837         else if (nosh->printop[iprint][iarg-1] == 1) scalar = -1.0;
02838         /* Accumulate */
02839         ltenergy += (scalar * Vunit_kb * (1e-3) * Vunit_Na *
02840                      nosh->calc[calcid]->pbeparm->temp * totEnergy[calcid]);
02841     }
02842
02843     Vnm_tprint( 1, "  Local net energy (PE %d) = %1.12E kJ/mol\n",
02844                 Vcom_rank(com), ltenergy);
02845     Vnm_tprint( 0, "printEnergy:  Performing global reduction (sum)\n");
02846     Vcom_reduce(com, &ltenergy, &gtenergy, 1, 2, 0);
02847     Vnm_tprint( 1, "  Global net ELEC energy = %1.12E kJ/mol\n", gtenergy);
02848
02849     return 1;
02850
02851 }
02852
02853 VPUBLIC int printElecEnergy(Vcom *com,
02854                             NOsh *nosh,
02855                             double totEnergy[NOSH_MAXCALC],
02856                             int iprint
02857                            ) {
02858
02859     int iarg,
02860         calcid;
02861     double ltenergy,
02862            gtenergy,
02863            scalar;
02864
02865     if (Vstring_strcasecmp(nosh->elecname[nosh->printcalc[iprint][0]], "") == 0){
02866         Vnm_tprint( 1, "\nprint energy %d ", nosh->printcalc[iprint][0]+1);
02867     } else {
02868         Vnm_tprint( 1, "\nprint energy %d (%s) ", nosh->printcalc[iprint][0]+1,
02869                     nosh->elecname[nosh->printcalc[iprint][0]]);
02870     }
02871     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
02872         if (nosh->printop[iprint][iarg-1] == 0)
02873             Vnm_tprint(1, "+ ");
02874         else if (nosh->printop[iprint][iarg-1] == 1)
02875             Vnm_tprint(1, "- ");
02876         else {
02877             Vnm_tprint( 2, "Undefined PRINT operation!\n");
02878             return 0;
02879         }
02880         if (Vstring_strcasecmp(nosh->elecname[nosh->printcalc[iprint][iarg]],
02881                                "") == 0) {
02882             Vnm_tprint( 1, "%d ", nosh->printcalc[iprint][iarg]+1);
02883         } else {
02884             Vnm_tprint( 1, "%d (%s) ", nosh->printcalc[iprint][iarg]+1,
02885                         nosh->elecname[nosh->printcalc[iprint][iarg]]);
02886         }
02887     }
02888     Vnm_tprint(1, "end\n");
02889     calcid = nosh->elec2calc[nosh->printcalc[iprint][0]];
02890     if (nosh->calc[calcid]->pbeparm->calcenergy != PCE_NO) {
02891         ltenergy = Vunit_kb * (1e-3) * Vunit_Na *
02892         nosh->calc[calcid]->pbeparm->temp * totEnergy[calcid];
02893     } else {
02894         Vnm_tprint( 2, "  Didn't calculate energy in Calculation \
```

```
02895 #%d\n", calcid+1);
02896         return 0;
02897     }
02898     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
02899         calcid = nosh->elec2calc[nosh->printcalc[iprint][iarg]];
02900         /* Add or subtract? */
02901         if (nosh->printop[iprint][iarg-1] == 0) scalar = 1.0;
02902         else if (nosh->printop[iprint][iarg-1] == 1) scalar = -1.0;
02903         /* Accumulate */
02904         ltenergy += (scalar * Vunit_kb * (1e-3) * Vunit_Na *
02905                      nosh->calc[calcid]->pbeparm->temp * totEnergy[calcid]);
02906     }
02907
02908     Vnm_tprint( 1, "  Local net energy (PE %d) = %1.12E kJ/mol\n",
02909                 Vcom_rank(com), ltenergy);
02910     Vnm_tprint( 0, "printEnergy:  Performing global reduction (sum)\n");
02911     Vcom_reduce(com, &ltenergy, &gtenergy, 1, 2, 0);
02912     Vnm_tprint( 1, "  Global net ELEC energy = %1.12E kJ/mol\n", gtenergy);
02913
02914     return 1;
02915
02916 }
02917
02918 VPUBLIC int printApolEnergy(NOsh *nosh,
02919                             int iprint
02920                            ) {
02921
02922     int iarg,
02923         calcid;
02924     double gtenergy,
02925            scalar;
02926     APOLparm *apolparm = VNULL;
02927
02928     if (Vstring_strcasecmp(nosh->apolname[nosh->printcalc[iprint][0]], "") == 0){
02929         Vnm_tprint( 1, "\nprint APOL energy %d ", nosh->printcalc[iprint][0]+1);
02930     } else {
02931         Vnm_tprint( 1, "\nprint APOL energy %d (%s) ", nosh->printcalc[iprint][0]+1,
02932                     nosh->apolname[nosh->printcalc[iprint][0]]);
02933     }
02934     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
02935         if (nosh->printop[iprint][iarg-1] == 0)
02936             Vnm_tprint(1, "+ ");
02937         else if (nosh->printop[iprint][iarg-1] == 1)
02938             Vnm_tprint(1, "- ");
02939         else {
02940             Vnm_tprint( 2, "Undefined PRINT operation!\n");
02941             return 0;
02942         }
02943         if (Vstring_strcasecmp(nosh->apolname[nosh->printcalc[iprint][iarg]],
02944                                "") == 0) {
02945             Vnm_tprint( 1, "%d ", nosh->printcalc[iprint][iarg]+1);
02946         } else {
02947             Vnm_tprint( 1, "%d (%s) ", nosh->printcalc[iprint][iarg]+1,
02948                         nosh->apolname[nosh->printcalc[iprint][iarg]]);
02949         }
02950     }
02951     Vnm_tprint(1, "end\n");
02952
02953     calcid = nosh->apol2calc[nosh->printcalc[iprint][0]];
02954     apolparm = nosh->calc[calcid]->apolparm;
02955
02956     if (apolparm->calcenergy == ACE_TOTAL) {
02957         gtenergy = ((apolparm->gamma*apolparm->sasa) + (apolparm->press*apolparm->sav) +
      (apolparm->wcaEnergy));
02958     } else {
02959         Vnm_tprint( 2, "  Didn't calculate energy in Calculation #%d\n", calcid+1);
02960         return 0;
02961     }
02962     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
02963         calcid = nosh->apol2calc[nosh->printcalc[iprint][iarg]];
02964         apolparm = nosh->calc[calcid]->apolparm;
02965
02966         /* Add or subtract? */
02967         if (nosh->printop[iprint][iarg-1] == 0) scalar = 1.0;
02968         else if (nosh->printop[iprint][iarg-1] == 1) scalar = -1.0;
02969         /* Accumulate */
02970         gtenergy += (scalar * ((apolparm->gamma*apolparm->sasa) +
02971                                (apolparm->press*apolparm->sav) +
02972                                (apolparm->wcaEnergy)));
02973
02974     }
```

```
02975
02976     Vnm_tprint( 1, "  Global net APOL energy = %1.12E kJ/mol\n", gtenergy);
02977     return 1;
02978 }
02979
02980 VPUBLIC int printForce(Vcom *com,
02981                        NOsh *nosh,
02982                        int nforce[NOSH_MAXCALC],
02983                        AtomForce *atomForce[NOSH_MAXCALC],
02984                        int iprint
02985                       ) {
02986
02987     int iarg,
02988         ifr,
02989         ivc,
02990         calcid,
02991         refnforce,
02992         refcalcforce;
02993     double temp,
02994           scalar,
02995           totforce[3];
02996     AtomForce *lforce,
02997               *gforce,
02998               *aforce;
02999
03000     Vnm_tprint( 2, "Warning: The 'force' print keyword is deprecated.\n" \
03001                    "         Use elecForce for electrostatics force calcs.\n\n");
03002
03003     if (Vstring_strcasecmp(nosh->elecname[nosh->printcalc[iprint][0]], "") == 0){
03004         Vnm_tprint( 1, "print force %d ", nosh->printcalc[iprint][0]+1);
03005     } else {
03006         Vnm_tprint( 1, "print force %d (%s) ", nosh->printcalc[iprint][0]+1,
03007                     nosh->elecname[nosh->printcalc[iprint][0]]);
03008     }
03009     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
03010         if (nosh->printop[iprint][iarg-1] == 0)
03011             Vnm_tprint(1, "+ ");
03012         else if (nosh->printop[iprint][iarg-1] == 1)
03013             Vnm_tprint(1, "- ");
03014         else {
03015             Vnm_tprint( 2, "Undefined PRINT operation!\n");
03016             return 0;
03017         }
03018         if (Vstring_strcasecmp(nosh->elecname[nosh->printcalc[iprint][iarg]],
03019                                "") == 0) {
03020             Vnm_tprint( 1, "%d ", nosh->printcalc[iprint][iarg]+1);
03021         } else {
03022             Vnm_tprint( 1, "%d (%s) ", nosh->printcalc[iprint][iarg]+1,
03023                         nosh->elecname[nosh->printcalc[iprint][iarg]]);
03024         }
03025     }
03026     Vnm_tprint(1, "end\n");
03027
03028     /* First, go through and make sure we did the same type of force
03029         * evaluation in each of the requested calculations */
03030     calcid = nosh->elec2calc[nosh->printcalc[iprint][0]];
03031     refnforce = nforce[calcid];
03032     refcalcforce = nosh->calc[calcid]->pbeparm->calcforce;
03033     if (refcalcforce == PCF_NO) {
03034         Vnm_tprint( 2, "  Didn't calculate force in calculation \
03035 #%d\n", calcid+1);
03036         return 0;
03037     }
03038     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
03039         calcid = nosh->elec2calc[nosh->printcalc[iprint][iarg]-1];
03040         if (nosh->calc[calcid]->pbeparm->calcforce != refcalcforce) {
03041             Vnm_tprint(2, "  Inconsistent calcforce declarations in \
03042 calculations %d and %d\n", nosh->elec2calc[nosh->printcalc[iprint][0]]+1,
03043                        calcid+1);
03044             return 0;
03045         }
03046         if (nforce[calcid] != refnforce) {
03047             Vnm_tprint(2, "  Inconsistent number of forces evaluated in \
03048 calculations %d and %d\n", nosh->elec2calc[nosh->printcalc[iprint][0]]+1,
03049                        calcid+1);
03050             return 0;
03051         }
03052     }
03053
03054     /* Now, allocate space to accumulate the forces */
03055     lforce = (AtomForce *)Vmem_malloc(VNULL, refnforce, sizeof(AtomForce));
```

```
03056     gforce = (AtomForce *)Vmem_malloc(VNULL, refnforce, sizeof(AtomForce));
03057
03058     /* Now, accumulate the forces */
03059     calcid = nosh->elec2calc[nosh->printcalc[iprint][0]];
03060     aforce = atomForce[calcid];
03061     temp = nosh->calc[calcid]->pbeparm->temp;
03062
03063     /* Load up the first calculation */
03064     if (refcalcforce == PCF_TOTAL) {
03065         /* Set to total force */
03066         for (ivc=0; ivc<3; ivc++) {
03067             lforce[0].qfForce[ivc] =
03068             Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].qfForce[ivc];
03069             lforce[0].ibForce[ivc] =
03070                 Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].ibForce[ivc];
03071             lforce[0].dbForce[ivc] =
03072                 Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].dbForce[ivc];
03073         }
03074     } else if (refcalcforce == PCF_COMPS) {
03075         for (ifr=0; ifr<refnforce; ifr++) {
03076             for (ivc=0; ivc<3; ivc++) {
03077                 lforce[ifr].qfForce[ivc] =
03078                 Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].qfForce[ivc];
03079                 lforce[ifr].ibForce[ivc] =
03080                     Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].ibForce[ivc];
03081                 lforce[ifr].dbForce[ivc] =
03082                     Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].dbForce[ivc];
03083             }
03084         }
03085     }
03086
03087     /* Load up the rest of the calculations */
03088     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
03089         calcid = nosh->elec2calc[nosh->printcalc[iprint][iarg]];
03090         temp = nosh->calc[calcid]->pbeparm->temp;
03091         aforce = atomForce[calcid];
03092         /* Get operation */
03093         if (nosh->printop[iprint][iarg-1] == 0) scalar = +1.0;
03094         else if (nosh->printop[iprint][iarg-1] == 1) scalar = -1.0;
03095         else scalar = 0.0;
03096         /* Accumulate */
03097         if (refcalcforce == PCF_TOTAL) {
03098             /* Set to total force */
03099             for (ivc=0; ivc<3; ivc++) {
03100                 lforce[0].qfForce[ivc] +=
03101                 (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].qfForce[ivc]);
03102                 lforce[0].ibForce[ivc] +=
03103                     (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].ibForce[ivc]);
03104                 lforce[0].dbForce[ivc] +=
03105                     (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].dbForce[ivc]);
03106             }
03107         } else if (refcalcforce == PCF_COMPS) {
03108             for (ifr=0; ifr<refnforce; ifr++) {
03109                 for (ivc=0; ivc<3; ivc++) {
03110                     lforce[ifr].qfForce[ivc] +=
03111                     (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].qfForce[ivc]);
03112                     lforce[ifr].ibForce[ivc] +=
03113                         (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].ibForce[ivc]);
03114                     lforce[ifr].dbForce[ivc] +=
03115                         (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].dbForce[ivc]);
03116                 }
03117             }
03118         }
03119     }
03120
03121     Vnm_tprint( 0, "printEnergy:  Performing global reduction (sum)\n");
03122     for (ifr=0; ifr<refnforce; ifr++) {
03123         Vcom_reduce(com, lforce[ifr].qfForce, gforce[ifr].qfForce, 3, 2, 0);
03124         Vcom_reduce(com, lforce[ifr].ibForce, gforce[ifr].ibForce, 3, 2, 0);
03125         Vcom_reduce(com, lforce[ifr].dbForce, gforce[ifr].dbForce, 3, 2, 0);
03126     }
03127
03128 #if 0
03129     if (refcalcforce == PCF_TOTAL) {
03130         Vnm_tprint( 1, "  Local net fixed charge force = \
03131 (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", lforce[0].qfForce[0],
03132                     lforce[0].qfForce[1], lforce[0].qfForce[2]);
03133         Vnm_tprint( 1, "  Local net ionic boundary force = \
03134 (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", lforce[0].ibForce[0],
03135                     lforce[0].ibForce[1], lforce[0].ibForce[2]);
03136         Vnm_tprint( 1, "  Local net dielectric boundary force = \
```

```
03137 (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", lforce[0].dbForce[0],
03138                     lforce[0].dbForce[1], lforce[0].dbForce[2]);
03139     } else if (refcalcforce == PCF_COMPS) {
03140         for (ifr=0; ifr<refnforce; ifr++) {
03141             Vnm_tprint( 1, "  Local fixed charge force \
03142 (atom %d) = (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", ifr, lforce[ifr].qfForce[0],
03143                         lforce[ifr].qfForce[1], lforce[ifr].qfForce[2]);
03144             Vnm_tprint( 1, "  Local ionic boundary force \
03145 (atom %d) = (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", ifr, lforce[ifr].ibForce[0],
03146                         lforce[ifr].ibForce[1], lforce[ifr].ibForce[2]);
03147             Vnm_tprint( 1, "  Local dielectric boundary force \
03148 (atom %d) = (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", ifr, lforce[ifr].dbForce[0],
03149                         lforce[ifr].dbForce[1], lforce[ifr].dbForce[2]);
03150         }
03151     }
03152 #endif
03153
03154     if (refcalcforce == PCF_TOTAL) {
03155         Vnm_tprint( 1, "  Printing net forces (kJ/mol/A).\n");
03156         Vnm_tprint( 1, "  Legend:\n");
03157         Vnm_tprint( 1, "    tot -- Total force\n");
03158         Vnm_tprint( 1, "    qf  -- Fixed charge force\n");
03159         Vnm_tprint( 1, "    db  -- Dielectric boundary force\n");
03160         Vnm_tprint( 1, "    ib  -- Ionic boundary force\n");
03161
03162         for (ivc=0; ivc<3; ivc++) {
03163             totforce[ivc] =
03164             gforce[0].qfForce[ivc] + gforce[0].ibForce[ivc] \
03165             + gforce[0].dbForce[ivc];
03166         }
03167
03168         Vnm_tprint( 1, "  tot %1.12E  %1.12E  %1.12E\n", totforce[0],
03169                     totforce[1], totforce[2]);
03170         Vnm_tprint( 1, "  qf  %1.12E  %1.12E  %1.12E\n", gforce[0].qfForce[0],
03171                     gforce[0].qfForce[1], gforce[0].qfForce[2]);
03172         Vnm_tprint( 1, "  ib  %1.12E  %1.12E  %1.12E\n", gforce[0].ibForce[0],
03173                     gforce[0].ibForce[1], gforce[0].ibForce[2]);
03174         Vnm_tprint( 1, "  db  %1.12E  %1.12E  %1.12E\n", gforce[0].dbForce[0],
03175                     gforce[0].dbForce[1], gforce[0].dbForce[2]);
03176
03177     } else if (refcalcforce == PCF_COMPS) {
03178
03179         Vnm_tprint( 1, "  Printing per-atom forces (kJ/mol/A).\n");
03180         Vnm_tprint( 1, "  Legend:\n");
03181         Vnm_tprint( 1, "    tot n -- Total force for atom n\n");
03182         Vnm_tprint( 1, "    qf  n -- Fixed charge force for atom n\n");
03183         Vnm_tprint( 1, "    db  n -- Dielectric boundary force for atom n\n");
03184         Vnm_tprint( 1, "    ib  n -- Ionic boundary force for atom n\n");
03185         Vnm_tprint( 1, "    tot all -- Total force for system\n");
03186
03187         totforce[0] = 0.0;
03188         totforce[1] = 0.0;
03189         totforce[2] = 0.0;
03190
03191         for (ifr=0; ifr<refnforce; ifr++) {
03192             Vnm_tprint( 1, "  qf  %d  %1.12E  %1.12E  %1.12E\n", ifr,
03193                         gforce[ifr].qfForce[0], gforce[ifr].qfForce[1],
03194                         gforce[ifr].qfForce[2]);
03195             Vnm_tprint( 1, "  ib  %d  %1.12E  %1.12E  %1.12E\n", ifr,
03196                         gforce[ifr].ibForce[0], gforce[ifr].ibForce[1],
03197                         gforce[ifr].ibForce[2]);
03198             Vnm_tprint( 1, "  db  %d  %1.12E  %1.12E  %1.12E\n", ifr,
03199                         gforce[ifr].dbForce[0], gforce[ifr].dbForce[1],
03200                         gforce[ifr].dbForce[2]);
03201             Vnm_tprint( 1, "  tot %d  %1.12E  %1.12E  %1.12E\n", ifr,
03202                         (gforce[ifr].dbForce[0] \
03203                          + gforce[ifr].ibForce[0] +
03204                          gforce[ifr].qfForce[0]),
03205                         (gforce[ifr].dbForce[1] \
03206                          + gforce[ifr].ibForce[1] +
03207                          gforce[ifr].qfForce[1]),
03208                         (gforce[ifr].dbForce[2] \
03209                          + gforce[ifr].ibForce[2] +
03210                          gforce[ifr].qfForce[2]));
03211             for (ivc=0; ivc<3; ivc++) {
03212                 totforce[ivc] += (gforce[ifr].dbForce[ivc] \
03213                                   + gforce[ifr].ibForce[ivc] \
03214                                     + gforce[ifr].qfForce[ivc]);
03215             }
03216         }
03217         Vnm_tprint( 1, "  tot all %1.12E  %1.12E  %1.12E\n", totforce[0],
```

```
03218                     totforce[1], totforce[2]);
03219     }
03220
03221     Vmem_free(VNULL, refnforce, sizeof(AtomForce), (void **)(&lforce));
03222     Vmem_free(VNULL, refnforce, sizeof(AtomForce), (void **)(&gforce));
03223
03224     return 1;
03225
03226 }
03227
03228 VPUBLIC int printElecForce(Vcom *com,
03229                            NOsh *nosh,
03230                            int nforce[NOSH_MAXCALC],
03231                            AtomForce *atomForce[NOSH_MAXCALC],
03232                            int iprint
03233                           ) {
03234
03235     int iarg,
03236         ifr,
03237         ivc,
03238         calcid,
03239         refnforce,
03240         refcalcforce;
03241     double temp,
03242            scalar,
03243            totforce[3];
03244     AtomForce *lforce, *gforce, *aforce;
03245
03246     if (Vstring_strcasecmp(nosh->elecname[nosh->printcalc[iprint][0]], "") == 0){
03247         Vnm_tprint( 1, "print force %d ", nosh->printcalc[iprint][0]+1);
03248     } else {
03249         Vnm_tprint( 1, "print force %d (%s) ", nosh->printcalc[iprint][0]+1,
03250                     nosh->elecname[nosh->printcalc[iprint][0]]);
03251     }
03252     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
03253         if (nosh->printop[iprint][iarg-1] == 0)
03254             Vnm_tprint(1, "+ ");
03255         else if (nosh->printop[iprint][iarg-1] == 1)
03256             Vnm_tprint(1, "- ");
03257         else {
03258             Vnm_tprint( 2, "Undefined PRINT operation!\n");
03259             return 0;
03260         }
03261         if (Vstring_strcasecmp(nosh->elecname[nosh->printcalc[iprint][iarg]],
03262                                "") == 0) {
03263             Vnm_tprint( 1, "%d ", nosh->printcalc[iprint][iarg]+1);
03264         } else {
03265             Vnm_tprint( 1, "%d (%s) ", nosh->printcalc[iprint][iarg]+1,
03266                         nosh->elecname[nosh->printcalc[iprint][iarg]]);
03267         }
03268     }
03269     Vnm_tprint(1, "end\n");
03270
03271     /* First, go through and make sure we did the same type of force
03272         * evaluation in each of the requested calculations */
03273     calcid = nosh->elec2calc[nosh->printcalc[iprint][0]];
03274     refnforce = nforce[calcid];
03275     refcalcforce = nosh->calc[calcid]->pbeparm->calcforce;
03276     if (refcalcforce == PCF_NO) {
03277         Vnm_tprint( 2, "  Didn't calculate force in calculation \
03278 #%d\n", calcid+1);
03279         return 0;
03280     }
03281     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
03282         calcid = nosh->elec2calc[nosh->printcalc[iprint][iarg]-1];
03283         if (nosh->calc[calcid]->pbeparm->calcforce != refcalcforce) {
03284             Vnm_tprint(2, "  Inconsistent calcforce declarations in \
03285 calculations %d and %d\n", nosh->elec2calc[nosh->printcalc[iprint][0]]+1,
03286                        calcid+1);
03287             return 0;
03288         }
03289         if (nforce[calcid] != refnforce) {
03290             Vnm_tprint(2, "  Inconsistent number of forces evaluated in \
03291 calculations %d and %d\n", nosh->elec2calc[nosh->printcalc[iprint][0]]+1,
03292                        calcid+1);
03293             return 0;
03294         }
03295     }
03296
03297     /* Now, allocate space to accumulate the forces */
03298     lforce = (AtomForce *)Vmem_malloc(VNULL, refnforce, sizeof(AtomForce));
```

```
03299     gforce = (AtomForce *)Vmem_malloc(VNULL, refnforce, sizeof(AtomForce));
03300
03301     /* Now, accumulate the forces */
03302     calcid = nosh->elec2calc[nosh->printcalc[iprint][0]];
03303     aforce = atomForce[calcid];
03304     temp = nosh->calc[calcid]->pbeparm->temp;
03305
03306     /* Load up the first calculation */
03307     if (refcalcforce == PCF_TOTAL) {
03308         /* Set to total force */
03309         for (ivc=0; ivc<3; ivc++) {
03310             lforce[0].qfForce[ivc] =
03311             Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].qfForce[ivc];
03312             lforce[0].ibForce[ivc] =
03313                 Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].ibForce[ivc];
03314             lforce[0].dbForce[ivc] =
03315                 Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].dbForce[ivc];
03316         }
03317     } else if (refcalcforce == PCF_COMPS) {
03318         for (ifr=0; ifr<refnforce; ifr++) {
03319             for (ivc=0; ivc<3; ivc++) {
03320                 lforce[ifr].qfForce[ivc] =
03321                 Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].qfForce[ivc];
03322                 lforce[ifr].ibForce[ivc] =
03323                     Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].ibForce[ivc];
03324                 lforce[ifr].dbForce[ivc] =
03325                     Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].dbForce[ivc];
03326             }
03327         }
03328     }
03329
03330     /* Load up the rest of the calculations */
03331     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
03332         calcid = nosh->elec2calc[nosh->printcalc[iprint][iarg]];
03333         temp = nosh->calc[calcid]->pbeparm->temp;
03334         aforce = atomForce[calcid];
03335         /* Get operation */
03336         if (nosh->printop[iprint][iarg-1] == 0) scalar = +1.0;
03337         else if (nosh->printop[iprint][iarg-1] == 1) scalar = -1.0;
03338         else scalar = 0.0;
03339         /* Accumulate */
03340         if (refcalcforce == PCF_TOTAL) {
03341             /* Set to total force */
03342             for (ivc=0; ivc<3; ivc++) {
03343                 lforce[0].qfForce[ivc] +=
03344                 (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].qfForce[ivc]);
03345                 lforce[0].ibForce[ivc] +=
03346                     (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].ibForce[ivc]);
03347                 lforce[0].dbForce[ivc] +=
03348                     (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[0].dbForce[ivc]);
03349             }
03350         } else if (refcalcforce == PCF_COMPS) {
03351             for (ifr=0; ifr<refnforce; ifr++) {
03352                 for (ivc=0; ivc<3; ivc++) {
03353                     lforce[ifr].qfForce[ivc] +=
03354                     (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].qfForce[ivc]);
03355                     lforce[ifr].ibForce[ivc] +=
03356                         (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].ibForce[ivc]);
03357                     lforce[ifr].dbForce[ivc] +=
03358                         (scalar*Vunit_kb*(1e-3)*Vunit_Na*temp*aforce[ifr].dbForce[ivc]);
03359                 }
03360             }
03361         }
03362     }
03363
03364     Vnm_tprint( 0, "printEnergy:  Performing global reduction (sum)\n");
03365     for (ifr=0; ifr<refnforce; ifr++) {
03366         Vcom_reduce(com, lforce[ifr].qfForce, gforce[ifr].qfForce, 3, 2, 0);
03367         Vcom_reduce(com, lforce[ifr].ibForce, gforce[ifr].ibForce, 3, 2, 0);
03368         Vcom_reduce(com, lforce[ifr].dbForce, gforce[ifr].dbForce, 3, 2, 0);
03369     }
03370
03371 #if 0
03372     if (refcalcforce == PCF_TOTAL) {
03373         Vnm_tprint( 1, "  Local net fixed charge force = \
03374 (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", lforce[0].qfForce[0],
03375                     lforce[0].qfForce[1], lforce[0].qfForce[2]);
03376         Vnm_tprint( 1, "  Local net ionic boundary force = \
03377 (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", lforce[0].ibForce[0],
03378                     lforce[0].ibForce[1], lforce[0].ibForce[2]);
03379         Vnm_tprint( 1, "  Local net dielectric boundary force = \
```

```
03380 (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", lforce[0].dbForce[0],
03381                     lforce[0].dbForce[1], lforce[0].dbForce[2]);
03382     } else if (refcalcforce == PCF_COMPS) {
03383         for (ifr=0; ifr<refnforce; ifr++) {
03384             Vnm_tprint( 1, "  Local fixed charge force \
03385 (atom %d) = (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", ifr, lforce[ifr].qfForce[0],
03386                         lforce[ifr].qfForce[1], lforce[ifr].qfForce[2]);
03387             Vnm_tprint( 1, "  Local ionic boundary force \
03388 (atom %d) = (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", ifr, lforce[ifr].ibForce[0],
03389                         lforce[ifr].ibForce[1], lforce[ifr].ibForce[2]);
03390             Vnm_tprint( 1, "  Local dielectric boundary force \
03391 (atom %d) = (%1.12E, %1.12E, %1.12E) kJ/mol/A\n", ifr, lforce[ifr].dbForce[0],
03392                         lforce[ifr].dbForce[1], lforce[ifr].dbForce[2]);
03393         }
03394     }
03395 #endif
03396
03397     if (refcalcforce == PCF_TOTAL) {
03398         Vnm_tprint( 1, "  Printing net forces (kJ/mol/A).\n");
03399         Vnm_tprint( 1, "  Legend:\n");
03400         Vnm_tprint( 1, "    tot -- Total force\n");
03401         Vnm_tprint( 1, "    qf  -- Fixed charge force\n");
03402         Vnm_tprint( 1, "    db  -- Dielectric boundary force\n");
03403         Vnm_tprint( 1, "    ib  -- Ionic boundary force\n");
03404
03405         for (ivc=0; ivc<3; ivc++) {
03406             totforce[ivc] =
03407             gforce[0].qfForce[ivc] + gforce[0].ibForce[ivc] \
03408             + gforce[0].dbForce[ivc];
03409         }
03410
03411         Vnm_tprint( 1, "  tot %1.12E  %1.12E  %1.12E\n", totforce[0],
03412                     totforce[1], totforce[2]);
03413         Vnm_tprint( 1, "  qf  %1.12E  %1.12E  %1.12E\n", gforce[0].qfForce[0],
03414                     gforce[0].qfForce[1], gforce[0].qfForce[2]);
03415         Vnm_tprint( 1, "  ib  %1.12E  %1.12E  %1.12E\n", gforce[0].ibForce[0],
03416                     gforce[0].ibForce[1], gforce[0].ibForce[2]);
03417         Vnm_tprint( 1, "  db  %1.12E  %1.12E  %1.12E\n", gforce[0].dbForce[0],
03418                     gforce[0].dbForce[1], gforce[0].dbForce[2]);
03419
03420     } else if (refcalcforce == PCF_COMPS) {
03421
03422         Vnm_tprint( 1, "  Printing per-atom forces (kJ/mol/A).\n");
03423         Vnm_tprint( 1, "  Legend:\n");
03424         Vnm_tprint( 1, "    tot n -- Total force for atom n\n");
03425         Vnm_tprint( 1, "    qf  n -- Fixed charge force for atom n\n");
03426         Vnm_tprint( 1, "    db  n -- Dielectric boundary force for atom n\n");
03427         Vnm_tprint( 1, "    ib  n -- Ionic boundary force for atom n\n");
03428         Vnm_tprint( 1, "    tot all -- Total force for system\n");
03429
03430         totforce[0] = 0.0;
03431         totforce[1] = 0.0;
03432         totforce[2] = 0.0;
03433
03434         for (ifr=0; ifr<refnforce; ifr++) {
03435             Vnm_tprint( 1, "  qf  %d  %1.12E  %1.12E  %1.12E\n", ifr,
03436                         gforce[ifr].qfForce[0], gforce[ifr].qfForce[1],
03437                         gforce[ifr].qfForce[2]);
03438             Vnm_tprint( 1, "  ib  %d  %1.12E  %1.12E  %1.12E\n", ifr,
03439                         gforce[ifr].ibForce[0], gforce[ifr].ibForce[1],
03440                         gforce[ifr].ibForce[2]);
03441             Vnm_tprint( 1, "  db  %d  %1.12E  %1.12E  %1.12E\n", ifr,
03442                         gforce[ifr].dbForce[0], gforce[ifr].dbForce[1],
03443                         gforce[ifr].dbForce[2]);
03444             Vnm_tprint( 1, "  tot %d  %1.12E  %1.12E  %1.12E\n", ifr,
03445                         (gforce[ifr].dbForce[0] \
03446                          + gforce[ifr].ibForce[0] +
03447                          gforce[ifr].qfForce[0]),
03448                         (gforce[ifr].dbForce[1] \
03449                          + gforce[ifr].ibForce[1] +
03450                          gforce[ifr].qfForce[1]),
03451                         (gforce[ifr].dbForce[2] \
03452                          + gforce[ifr].ibForce[2] +
03453                          gforce[ifr].qfForce[2]));
03454             for (ivc=0; ivc<3; ivc++) {
03455                 totforce[ivc] += (gforce[ifr].dbForce[ivc] \
03456                                   + gforce[ifr].ibForce[ivc] \
03457                                     + gforce[ifr].qfForce[ivc]);
03458             }
03459         }
03460         Vnm_tprint( 1, "  tot all %1.12E  %1.12E  %1.12E\n", totforce[0],
```

```
03461                     totforce[1], totforce[2]);
03462     }
03463
03464     Vmem_free(VNULL, refnforce, sizeof(AtomForce), (void **)(&lforce));
03465     Vmem_free(VNULL, refnforce, sizeof(AtomForce), (void **)(&gforce));
03466
03467     return 1;
03468
03469 }
03470
03471 VPUBLIC int printApolForce(Vcom *com,
03472                            NOsh *nosh,
03473                            int nforce[NOSH_MAXCALC],
03474                            AtomForce *atomForce[NOSH_MAXCALC],
03475                            int iprint
03476                           ) {
03477
03478     int iarg,
03479         ifr,
03480         ivc,
03481         calcid,
03482         refnforce,
03483         refcalcforce;
03484     double temp,
03485            scalar,
03486            totforce[3];
03487     AtomForce *lforce,
03488               *gforce,
03489               *aforce;
03490
03491     if (Vstring_strcasecmp(nosh->apolname[nosh->printcalc[iprint][0]], "") == 0){
03492         Vnm_tprint( 1, "\nprint APOL force %d ", nosh->printcalc[iprint][0]+1);
03493     } else {
03494         Vnm_tprint( 1, "\nprint APOL force %d (%s) ", nosh->printcalc[iprint][0]+1,
03495                     nosh->apolname[nosh->printcalc[iprint][0]]);
03496     }
03497     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
03498         if (nosh->printop[iprint][iarg-1] == 0)
03499             Vnm_tprint(1, "+ ");
03500         else if (nosh->printop[iprint][iarg-1] == 1)
03501             Vnm_tprint(1, "- ");
03502         else {
03503             Vnm_tprint( 2, "Undefined PRINT operation!\n");
03504             return 0;
03505         }
03506         if (Vstring_strcasecmp(nosh->apolname[nosh->printcalc[iprint][iarg]],
03507                                "") == 0) {
03508             Vnm_tprint( 1, "%d ", nosh->printcalc[iprint][iarg]+1);
03509         } else {
03510             Vnm_tprint( 1, "%d (%s) ", nosh->printcalc[iprint][iarg]+1,
03511                         nosh->apolname[nosh->printcalc[iprint][iarg]]);
03512         }
03513     }
03514     Vnm_tprint(1, "end\n");
03515
03516     /* First, go through and make sure we did the same type of force
03517        * evaluation in each of the requested calculations */
03518     calcid = nosh->apol2calc[nosh->printcalc[iprint][0]];
03519     refnforce = nforce[calcid];
03520     refcalcforce = nosh->calc[calcid]->apolparm->calcforce;
03521     if (refcalcforce == ACF_NO) {
03522         Vnm_tprint( 2, "  Didn't calculate force in calculation \
03523 #%d\n", calcid+1);
03524         return 0;
03525     }
03526     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
03527         calcid = nosh->apol2calc[nosh->printcalc[iprint][iarg]-1];
03528         if (nosh->calc[calcid]->apolparm->calcforce != refcalcforce) {
03529             Vnm_tprint(2, "  Inconsistent calcforce declarations in \
03530 calculations %d and %d\n", nosh->apol2calc[nosh->printcalc[iprint][0]]+1,
03531                        calcid+1);
03532             return 0;
03533         }
03534         if (nforce[calcid] != refnforce) {
03535             Vnm_tprint(2, "  Inconsistent number of forces evaluated in \
03536 calculations %d and %d\n", nosh->apol2calc[nosh->printcalc[iprint][0]]+1,
03537                        calcid+1);
03538             return 0;
03539         }
03540     }
03541
```

```
03542     /* Now, allocate space to accumulate the forces */
03543     lforce = (AtomForce *)Vmem_malloc(VNULL, refnforce, sizeof(AtomForce));
03544     gforce = (AtomForce *)Vmem_malloc(VNULL, refnforce, sizeof(AtomForce));
03545
03546     /* Now, accumulate the forces */
03547     calcid = nosh->apol2calc[nosh->printcalc[iprint][0]];
03548     aforce = atomForce[calcid];
03549     temp = nosh->calc[calcid]->apolparm->temp;
03550
03551     /* Load up the first calculation */
03552     if (refcalcforce == ACF_TOTAL) {
03553         /* Set to total force */
03554         for (ivc=0; ivc<3; ivc++) {
03555             lforce[0].sasaForce[ivc] = aforce[0].sasaForce[ivc];
03556             lforce[0].savForce[ivc] = aforce[0].savForce[ivc];
03557             lforce[0].wcaForce[ivc] = aforce[0].wcaForce[ivc];
03558         }
03559     } else if (refcalcforce == ACF_COMPS) {
03560         for (ifr=0; ifr<refnforce; ifr++) {
03561             for (ivc=0; ivc<3; ivc++) {
03562                 lforce[ifr].sasaForce[ivc] = aforce[ifr].sasaForce[ivc];
03563                 lforce[ifr].savForce[ivc] = aforce[ifr].savForce[ivc];
03564                 lforce[ifr].wcaForce[ivc] = aforce[ifr].wcaForce[ivc];
03565             }
03566         }
03567     }
03568
03569     /* Load up the rest of the calculations */
03570     for (iarg=1; iarg<nosh->printnarg[iprint]; iarg++) {
03571         calcid = nosh->apol2calc[nosh->printcalc[iprint][iarg]];
03572         temp = nosh->calc[calcid]->apolparm->temp;
03573         aforce = atomForce[calcid];
03574         /* Get operation */
03575         if (nosh->printop[iprint][iarg-1] == 0) scalar = +1.0;
03576         else if (nosh->printop[iprint][iarg-1] == 1) scalar = -1.0;
03577         else scalar = 0.0;
03578         /* Accumulate */
03579         if (refcalcforce == ACF_TOTAL) {
03580             /* Set to total force */
03581             for (ivc=0; ivc<3; ivc++) {
03582                 lforce[0].sasaForce[ivc] += aforce[0].sasaForce[ivc];
03583                 lforce[0].savForce[ivc] += aforce[0].savForce[ivc];
03584                 lforce[0].wcaForce[ivc] += aforce[0].wcaForce[ivc];
03585             }
03586         } else if (refcalcforce == ACF_COMPS) {
03587             for (ifr=0; ifr<refnforce; ifr++) {
03588                 for (ivc=0; ivc<3; ivc++) {
03589                     lforce[ifr].sasaForce[ivc] += aforce[ifr].sasaForce[ivc];
03590                     lforce[ifr].savForce[ivc] += aforce[ifr].savForce[ivc];
03591                     lforce[ifr].wcaForce[ivc] += aforce[ifr].wcaForce[ivc];
03592                 }
03593             }
03594         }
03595     }
03596
03597     Vnm_tprint( 0, "printForce:  Performing global reduction (sum)\n");
03598     for (ifr=0; ifr<refnforce; ifr++) {
03599         Vcom_reduce(com, lforce[ifr].sasaForce, gforce[ifr].sasaForce, 3, 2, 0);
03600         Vcom_reduce(com, lforce[ifr].savForce, gforce[ifr].savForce, 3, 2, 0);
03601         Vcom_reduce(com, lforce[ifr].wcaForce, gforce[ifr].wcaForce, 3, 2, 0);
03602     }
03603
03604     if (refcalcforce == ACF_TOTAL) {
03605         Vnm_tprint( 1, "  Printing net forces (kJ/mol/A)\n");
03606         Vnm_tprint( 1, "  Legend:\n");
03607         Vnm_tprint( 1, "    tot   -- Total force\n");
03608         Vnm_tprint( 1, "    sasa  -- SASA force\n");
03609         Vnm_tprint( 1, "    sav   -- SAV force\n");
03610         Vnm_tprint( 1, "    wca   -- WCA force\n\n");
03611
03612         for (ivc=0; ivc<3; ivc++) {
03613             totforce[ivc] =
03614             gforce[0].sasaForce[ivc] + gforce[0].savForce[ivc] \
03615             + gforce[0].wcaForce[ivc];
03616         }
03617
03618         Vnm_tprint( 1, "  tot %1.12E  %1.12E  %1.12E\n", totforce[0],
03619                     totforce[1], totforce[2]);
03620         Vnm_tprint( 1, "  sasa  %1.12E  %1.12E  %1.12E\n", gforce[0].sasaForce[0],
03621                     gforce[0].sasaForce[1], gforce[0].sasaForce[2]);
03622         Vnm_tprint( 1, "  sav  %1.12E  %1.12E  %1.12E\n", gforce[0].savForce[0],
```

```
03623                   gforce[0].savForce[1], gforce[0].savForce[2]);
03624         Vnm_tprint( 1, "  wca  %1.12E  %1.12E  %1.12E\n", gforce[0].wcaForce[0],
03625                     gforce[0].wcaForce[1], gforce[0].wcaForce[2]);
03626
03627     } else if (refcalcforce == ACF_COMPS) {
03628
03629         Vnm_tprint( 1, "  Printing per atom forces (kJ/mol/A)\n");
03630         Vnm_tprint( 1, "  Legend:\n");
03631         Vnm_tprint( 1, "    tot   n -- Total force for atom n\n");
03632         Vnm_tprint( 1, "    sasa  n -- SASA force for atom n\n");
03633         Vnm_tprint( 1, "    sav   n -- SAV force for atom n\n");
03634         Vnm_tprint( 1, "    wca   n -- WCA force for atom n\n");
03635         Vnm_tprint( 1, "    tot all -- Total force for system\n");
03636
03637         //Vnm_tprint( 1, "    gamma, pressure, bconc are: %f %f %f\n\n",
03638         //      gamma,press,bconc);
03639
03640         totforce[0] = 0.0;
03641         totforce[1] = 0.0;
03642         totforce[2] = 0.0;
03643
03644         for (ifr=0; ifr<refnforce; ifr++) {
03645             Vnm_tprint( 1, "  sasa  %d  %1.12E  %1.12E  %1.12E\n", ifr,
03646                         gforce[ifr].sasaForce[0], gforce[ifr].sasaForce[1],
03647                         gforce[ifr].sasaForce[2]);
03648             Vnm_tprint( 1, "  sav   %d  %1.12E  %1.12E  %1.12E\n", ifr,
03649                         gforce[ifr].savForce[0], gforce[ifr].savForce[1],
03650                         gforce[ifr].savForce[2]);
03651             Vnm_tprint( 1, "  wca   %d  %1.12E  %1.12E  %1.12E\n", ifr,
03652                         gforce[ifr].wcaForce[0], gforce[ifr].wcaForce[1],
03653                         gforce[ifr].wcaForce[2]);
03654             Vnm_tprint( 1, "  tot   %d  %1.12E  %1.12E  %1.12E\n", ifr,
03655                         (gforce[ifr].wcaForce[0] \
03656                          + gforce[ifr].savForce[0] +
03657                          gforce[ifr].sasaForce[0]),
03658                         (gforce[ifr].wcaForce[1] \
03659                          + gforce[ifr].savForce[1] +
03660                          gforce[ifr].sasaForce[1]),
03661                         (gforce[ifr].wcaForce[2] \
03662                          + gforce[ifr].savForce[2] +
03663                          gforce[ifr].sasaForce[2]));
03664             for (ivc=0; ivc<3; ivc++) {
03665                 totforce[ivc] += (gforce[ifr].wcaForce[ivc] \
03666                                   + gforce[ifr].savForce[ivc] \
03667                                     + gforce[ifr].sasaForce[ivc]);
03668             }
03669         }
03670         Vnm_tprint( 1, "  tot all  %1.12E  %1.12E  %1.12E\n", totforce[0],
03671                     totforce[1], totforce[2]);
03672     }
03673
03674     Vmem_free(VNULL, refnforce, sizeof(AtomForce), (void **)(&lforce));
03675     Vmem_free(VNULL, refnforce, sizeof(AtomForce), (void **)(&gforce));
03676
03677     return 1;
03678 }
03679
03680 #ifdef HAVE_MC_H
03681
03682
03683 VPUBLIC void killFE(NOsh *nosh,
03684                     Vpbe *pbe[NOSH_MAXCALC],
03685                     Vfetk *fetk[NOSH_MAXCALC],
03686                     Gem *gm[NOSH_MAXMOL]
03687                    ) {
03688
03689     int i;
03690
03691 #ifndef VAPBSQUIET
03692     Vnm_tprint(1, "Destroying finite element structures.\n");
03693 #endif
03694
03695     for(i=0;i<nosh->ncalc;i++){
03696         Vpbe_dtor(&(pbe[i]));
03697         Vfetk_dtor(&(fetk[i]));
03698     }
03699     for (i=0; i<nosh->nmesh; i++) {
03700         Gem_dtor(&(gm[i]));
03701 }
03702 }
03703
```

```
03711 VPUBLIC Vrc_Codes initFE(int icalc,
03712                           NOsh *nosh,
03713                           FEMparm *feparm,
03714                           PBEparm *pbeparm,
03715                           Vpbe *pbe[NOSH_MAXCALC],
03716                           Valist *alist[NOSH_MAXMOL],
03717                           Vfetk *fetk[NOSH_MAXCALC]
03718                          ) {
03719
03720     int iatom,
03721         imesh,
03722         i,
03723         j,
03724         theMol,
03725         focusFlag = 0;
03726     Vio *sock = VNULL;
03727     size_t bytesTotal,
03728            highWater;
03729     Vfetk_MeshLoad meshType;
03730     double length[3],
03731            center[3],
03732            sparm,
03733            q,
03734            iparm = 0.0;
03735     Vrc_Codes vrc;
03736     Valist *myalist;
03737     Vatom *atom = VNULL;
03739     Vnm_tstart(27, "Setup timer");
03740
03741     /* Print some warning messages */
03742     if (pbeparm->useDielMap)  Vnm_tprint(2, "FEM ignoring dielectric map!\n");
03743     if (pbeparm->useKappaMap)  Vnm_tprint(2, "FEM ignoring kappa map!\n");
03744     if (pbeparm->useChargeMap)  Vnm_tprint(2, "FEM ignoring charge map!\n");
03745
03746     /* Fix mesh center for "GCENT MOL #" types of declarations. */
03747     Vnm_tprint(0, "Re-centering mesh...\n");
03748     theMol = pbeparm->molid-1;
03749     myalist = alist[theMol];
03750     for (j=0; j<3; j++) {
03751         if (theMol < nosh->nmol) {
03752             center[j] = (myalist)->center[j];
03753         } else{
03754             Vnm_tprint(2, "ERROR!  Bogus molecule number (%d)!\n",
03755                        (theMol+1));
03756             return VRC_FAILURE;
03757         }
03758     }
03759
03760     /* Check for completely-neutral molecule */
03761     q = 0;
03762     for (iatom=0; iatom<Valist_getNumberAtoms(myalist); iatom++) {
03763         atom = Valist_getAtom(myalist, iatom);
03764         q += VSQR(Vatom_getCharge(atom));
03765     }
03766     /* D. Gohara 10/22/09 - disabled
03767     if (q < (1e-6)) {
03768         Vnm_tprint(2, "Molecule #%d is uncharged!\n", pbeparm->molid);
03769         Vnm_tprint(2, "Sum square charge = %g!\n", q);
03770         return VRC_FAILURE;
03771     }
03772     */
03773
03774     /* Set the femparm pkey value based on the presence of an HB solver */
03775 #ifdef USE_HB
03776     feparm->pkey = 1;
03777 #else
03778     feparm->pkey = 0;
03779 #endif
03780
03781     /* Set up PBE object */
03782     Vnm_tprint(0, "Setting up PBE object...\n");
03783     if (pbeparm->srfm == VSM_SPLINE) {
03784         sparm = pbeparm->swin;
03785     }
03786     else {
03787         sparm = pbeparm->srad;
03788     }
03789     if (pbeparm->nion > 0) {
03790         iparm = pbeparm->ionr[0];
03791     }
03792
```

```
03793     pbe[icalc] = Vpbe_ctor(myalist, pbeparm->nion,
03794                            pbeparm->ionc, pbeparm->ionr, pbeparm->ionq,
03795                            pbeparm->temp, pbeparm->pdie,
03796                            pbeparm->sdie, sparm, focusFlag, pbeparm->sdens,
03797                            pbeparm->zmem, pbeparm->Lmem, pbeparm->mdie,
03798                            pbeparm->memv);
03799
03800     /* Print a few derived parameters */
03801     Vnm_tprint(1, "  Debye length:  %g A\n", Vpbe_getDeblen(pbe[icalc]));
03802
03803     /* Set up FEtk objects */
03804     Vnm_tprint(0, "Setting up FEtk object...\n");
03805     fetk[icalc] = Vfetk_ctor(pbe[icalc], pbeparm->pbetype);
03806     Vfetk_setParameters(fetk[icalc], pbeparm, feparm);
03807
03808     /* Build mesh - this merely loads an MCSF file from an external source if one is specified or uses the
03809      * current molecule and sets center/length values based on that molecule if no external source is
03810      * specified. */
03811     Vnm_tprint(0, "Setting up mesh...\n");
03812     sock = VNULL;
03813     if (feparm->useMesh) {
03814         imesh = feparm->meshID-1;
03815         meshType = VML_EXTERNAL;
03816         for (i=0; i<3; i++) {
03817             center[i] = 0.0;
03818             length[i] = 0.0;
03819         }
03820         Vnm_print(0, "Using mesh %d (%s) in calculation.\n", imesh+1,
03821                   nosh->meshpath[imesh]);
03822         switch (nosh->meshfmt[imesh]) {
03823             case VDF_DX:
03824                 Vnm_tprint(2, "DX finite element mesh input not supported yet!\n");
03825                 return VRC_FAILURE;
03826             case VDF_DXBIN:
03827         Vnm_tprint(2, "DXBIN finite element mesh input not supported yet!\n");
03828         return VRC_FAILURE;
03829             case VDF_UHBD:
03830                 Vnm_tprint( 2, "UHBD finite element mesh input not supported!\n");
03831                 return VRC_FAILURE;
03832             case VDF_AVS:
03833                 Vnm_tprint( 2, "AVS finite element mesh input not supported!\n");
03834                 return VRC_FAILURE;
03835             case VDF_MCSF:
03836                 Vnm_tprint(1, "Reading MCSF-format input finite element mesh from %s.\n",
03837                            nosh->meshpath[imesh]);
03838                 sock = Vio_ctor("FILE", "ASC", VNULL, nosh->meshpath[imesh], "r");
03839                 if (sock == VNULL) {
03840                     Vnm_print(2, "Problem opening virtual socket %s!\n",
03841                               nosh->meshpath[imesh]);
03842                     return VRC_FAILURE;
03843                 }
03844                 if (Vio_accept(sock, 0) < 0) {
03845                     Vnm_print(2, "Problem accepting virtual socket %s!\n",
03846                               nosh->meshpath[imesh]);
03847                     return VRC_FAILURE;
03848                 }
03849                 break;
03850
03851             default:
03852                 Vnm_tprint( 2, "Invalid data format (%d)!\n",
03853                             nosh->meshfmt[imesh]);
03854                 return VRC_FAILURE;
03855         }
03856     } else { /* if (feparm->useMesh) */
03857         meshType = VML_DIRICUBE;
03858         for (i=0; i<3; i++) {
03859             center[i] = alist[theMol]->center[i];
03860             length[i] = feparm->glen[i];
03861         }
03862     }
03863
03864     /* Load the mesh with a particular center and vertex length using the provided input socket */
03865     vrc = Vfetk_loadMesh(fetk[icalc], center, length, meshType, sock);
03866     if (vrc == VRC_FAILURE) {
03867         Vnm_print(2, "Error constructing finite element mesh!\n");
03868         return VRC_FAILURE;
03869     }
03870     //Vnm_redirect(0); // Redirect output to io.mc
03871     Gem_shapeChk(fetk[icalc]->gm); // Traverse simplices and check shapes using the geometry manager.
03872     //Vnm_redirect(1);
03873
```

```
03874     /* Uniformly refine the mesh a bit */
03875     for (j=0; j<2; j++) {
03876         /* AM_* calls below are from the MC package, mc/src/nam/am.c.  Note that these calls actually are
03877          * wrappers around Aprx_* functions found in MC as well. */
03878         /* Mark the mesh for needed refinements */
03879         AM_markRefine(fetk[icalc]->am, 0, -1, 0, 0.0);
03880         /* Do actual mesh refinement */
03881         AM_refine(fetk[icalc]->am, 2, 0, feparm->pkey);
03882         //Vnm_redirect(0); // Redirect output to io.mc
03883         Gem_shapeChk(fetk[icalc]->gm); // Traverse simplices and check shapes using the geometry manager.
03884         //Vnm_redirect(1);
03885     }
03886
03887     /* Setup time statistics */
03888     Vnm_tstop(27, "Setup timer");
03889
03890     /* Memory statistics */
03891     bytesTotal = Vmem_bytesTotal();
03892     highWater = Vmem_highWaterTotal();
03893
03894 #ifndef VAPBSQUIET
03895     Vnm_tprint( 1, "  Current memory usage:  %4.3f MB total, \
03896 %4.3f MB high water\n", (double)(bytesTotal)/(1024.*1024.),
03897                 (double)(highWater)/(1024.*1024.));
03898 #endif
03899
03900     return VRC_SUCCESS;
03901 }
03902
03903 VPUBLIC void printFEPARM(int icalc,
03904                          NOsh *nosh,
03905                          FEMparm *feparm,
03906                          Vfetk *fetk[NOSH_MAXCALC]
03907                         ) {
03908
03909     Vnm_tprint(1, "  Domain size:  %g A x %g A x %g A\n",
03910                feparm->glen[0], feparm->glen[1],
03911                feparm->glen[2]);
03912     switch(feparm->ekey) {
03913         case FET_SIMP:
03914             Vnm_tprint(1, "  Per-simplex error tolerance:  %g\n", feparm->etol);
03915             break;
03916         case FET_GLOB:
03917             Vnm_tprint(1, "  Global error tolerance:  %g\n", feparm->etol);
03918             break;
03919         case FET_FRAC:
03920             Vnm_tprint(1, "  Fraction of simps to refine:  %g\n", feparm->etol);
03921             break;
03922         default:
03923             Vnm_tprint(2, "Invalid ekey (%d)!\n", feparm->ekey);
03924             VASSERT(0);
03925             break;
03926     }
03927     switch(feparm->akeyPRE) {
03928         case FRT_UNIF:
03929             Vnm_tprint(1, "  Uniform pre-solve refinement.\n");
03930             break;
03931         case FRT_GEOM:
03932             Vnm_tprint(1, "  Geometry-based pre-solve refinement.\n");
03933             break;
03934         default:
03935             Vnm_tprint(2, "Invalid akeyPRE (%d)!\n", feparm->akeyPRE);
03936             VASSERT(0);
03937             break;
03938     }
03939     switch(feparm->akeySOLVE) {
03940         case FRT_RESI:
03941             Vnm_tprint(1, "  Residual-based a posteriori refinement.\n");
03942             break;
03943         case FRT_DUAL:
03944             Vnm_tprint(1, "  Dual-based a posteriori refinement.\n");
03945             break;
03946         case FRT_LOCA:
03947             Vnm_tprint(1, "  Local-based a posteriori refinement.\n");
03948             break;
03949         default:
03950             Vnm_tprint(2, "Invalid akeySOLVE (%d)!\n", feparm->akeySOLVE);
03951             break;
03952     }
03953     Vnm_tprint(1, "  Refinement of initial mesh to ~%d vertices\n",
03954                feparm->targetNum);
```

```
03955     Vnm_tprint(1, "  Geometry-based refinment lower bound:  %g A\n",
03956                feparm->targetRes);
03957     Vnm_tprint(1, "  Maximum number of solve-estimate-refine cycles:  %d\n",
03958                feparm->maxsolve);
03959     Vnm_tprint(1, "  Maximum number of vertices in mesh:  %d\n",
03960                feparm->maxvert);
03961
03962     /* FOLLOWING IS SOLVER-RELATED; BAIL IF NOT SOLVING */
03963     if (nosh->bogus)  return;
03964 #ifdef USE_HB
03965     Vnm_tprint(1, "  HB linear solver:  AM_hPcg\n");
03966 #else
03967     Vnm_tprint(1, "  Non-HB linear solver:  ");
03968     switch (fetk[icalc]->lkey) {
03969             case VLT_SLU:
03970                 Vnm_print(1, "SLU direct\n");
03971                 break;
03972             case VLT_MG:
03973                 Vnm_print(1, "multigrid\n");
03974                 break;
03975             case VLT_CG:
03976                 Vnm_print(1, "conjugate gradient\n");
03977                 break;
03978             case VLT_BCG:
03979                 Vnm_print(1, "BiCGStab\n");
03980                 break;
03981             default:
03982                 Vnm_print(1, "???\n");
03983                 break;
03984     }
03985 #endif
03986
03987     Vnm_tprint(1, "  Linear solver tol.:  %g\n", fetk[icalc]->ltol);
03988     Vnm_tprint(1, "  Linear solver max. iters.:  %d\n", fetk[icalc]->lmax);
03989     Vnm_tprint(1, "  Linear solver preconditioner:  ");
03990     switch (fetk[icalc]->lprec) {
03991         case VPT_IDEN:
03992             Vnm_print(1, "identity\n");
03993             break;
03994         case VPT_DIAG:
03995             Vnm_print(1, "diagonal\n");
03996             break;
03997         case VPT_MG:
03998             Vnm_print(1, "multigrid\n");
03999             break;
04000         default:
04001             Vnm_print(1, "???\n");
04002             break;
04003     }
04004     Vnm_tprint(1, "  Nonlinear solver:  ");
04005     switch (fetk[icalc]->nkey) {
04006         case VNT_NEW:
04007             Vnm_print(1, "newton\n");
04008             break;
04009         case VNT_INC:
04010             Vnm_print(1, "incremental\n");
04011             break;
04012         case VNT_ARC:
04013             Vnm_print(1, "pseudo-arclength\n");
04014             break;
04015         default:
04016             Vnm_print(1, "??? ");
04017             break;
04018     }
04019     Vnm_tprint(1, "  Nonlinear solver tol.:  %g\n", fetk[icalc]->ntol);
04020     Vnm_tprint(1, "  Nonlinear solver max. iters.:  %d\n", fetk[icalc]->nmax);
04021     Vnm_tprint(1, "     Initial guess:  ");
04022     switch (fetk[icalc]->gues) {
04023         case VGT_ZERO:
04024             Vnm_tprint(1, "zero\n");
04025             break;
04026         case VGT_DIRI:
04027             Vnm_tprint(1, "boundary function\n");
04028             break;
04029         case VGT_PREV:
04030             Vnm_tprint(1, "interpolated previous solution\n");
04031             break;
04032         default:
04033             Vnm_tprint(1, "???\n");
04034             break;
04035     }
```

```
04036
04037 }
04038
04039 VPUBLIC int partFE(int icalc, NOsh *nosh, FEMparm *feparm,
04040                    Vfetk *fetk[NOSH_MAXCALC]) {
04041
04042     Vfetk_setAtomColors(fetk[icalc]);
04043     return 1;
04044 }
04045
04046 VPUBLIC int preRefineFE(int icalc, /* Calculation index */
04047                         FEMparm *feparm, /* FE-specific parameters */
04048                         Vfetk *fetk[NOSH_MAXCALC] /* Array of FE solver objects */
04049                        ) {
04050
04051     int nverts,
04052         marked;
04055     /* Based on the refinement type, alert the user to what we're tryng to refine with. */
04056     switch(feparm->akeyPRE) {
04057         case FRT_UNIF:
04058             Vnm_tprint(1, "  Commencing uniform refinement to %d verts.\n",
04059                        feparm->targetNum);
04060             break;
04061         case FRT_GEOM:
04062             Vnm_tprint(1, "  Commencing geometry-based refinement to %d \
04063 verts or %g A resolution.\n", feparm->targetNum, feparm->targetRes);
04064             break;
04065         case FRT_DUAL:
04066             Vnm_tprint(2, "What?  You can't do a posteriori error estimation \
04067 before you solve!\n");
04068             VASSERT(0);
04069             break;
04070         case FRT_RESI:
04071         case FRT_LOCA:
04072         default:
04073             VASSERT(0);
04074             break;
04075     }
04076
04082     Vnm_tprint(1, "  Initial mesh has %d vertices\n",
04083                Gem_numVV(fetk[icalc]->gm));
04084
04085     /* As long as we have simplices marked that need to be refined, run MC's
04086      * AM_markRefine against our data until we hit the error or size tolerance
04087      * for the refinement. */
04088     while (1) {
04089         nverts = Gem_numVV(fetk[icalc]->gm);
04090         if (nverts > feparm->targetNum) {
04091             Vnm_tprint(1, "  Hit vertex number limit.\n");
04092             break;
04093         }
04094         marked = AM_markRefine(fetk[icalc]->am, feparm->akeyPRE, -1,
04095                                feparm->ekey, feparm->etol);
04096         if (marked == 0) {
04097             Vnm_tprint(1, "  Marked 0 simps; hit error/size tolerance.\n");
04098             break;
04099         }
04100         Vnm_tprint(1, "    Have %d verts, marked %d.  Refining...\n", nverts,
04101                    marked);
04102         AM_refine(fetk[icalc]->am, 0, 0, feparm->pkey);
04103     }
04104
04105     nverts = Gem_numVV(fetk[icalc]->gm);
04106     Vnm_tprint(1, "  Done refining; have %d verts.\n", nverts);
04107
04108     return 1;
04109 }
04110
04111
04116 VPUBLIC int solveFE(int icalc,
04117                     PBEparm *pbeparm,
04118                     FEMparm *feparm,
04119                     Vfetk *fetk[NOSH_MAXCALC]
04120                    ) {
04121
04122     int lkeyHB = 3,
04123         meth = 2,
04124         prob = 0,
04125         prec = 0;
04127     if ((pbeparm->pbetype==PBE_NPBE) ||
04128         (pbeparm->pbetype == PBE_NRPBE) ||
```

```
04129         (pbeparm->pbetype == PBE_SMPBE)) {
04130
04131         /* Call MC's nonlinear solver - mc/src/nam/nsolv.c */
04132         AM_nSolve(
04133                   fetk[icalc]->am,
04134                   fetk[icalc]->nkey,
04135                   fetk[icalc]->nmax,
04136                   fetk[icalc]->ntol,
04137                   meth,
04138                   fetk[icalc]->lmax,
04139                   fetk[icalc]->ltol,
04140                   prec,
04141                   fetk[icalc]->gues,
04142                   fetk[icalc]->pjac
04143                   );
04144     } else if ((pbeparm->pbetype==PBE_LPBE) ||
04145                (pbeparm->pbetype==PBE_LRPBE)) {
04146         /* Note: USEHB is a compile time defined macro. The program flow
04147         is to always take the route using AM_hlSolve when the solver
04148         is linear. D. Gohara 6/29/06
04149         */
04150 #ifdef USE_HB
04151         Vnm_print(2, "SORRY!  DON'T USE HB!!!\n");
04152         VASSERT(0);
04153
04154         /* Call MC's hierarchical linear solver - mc/src/nam/lsolv.c */
04155         AM_hlSolve(fetk[icalc]->am, meth, lkeyHB, fetk[icalc]->lmax,
04156             fetk[icalc]->ltol, fetk[icalc]->gues, fetk[icalc]->pjac);
04157 #else
04158
04159         /* Call MC's linear solver - mc/src/nam/lsolv.c */
04160         AM_lSolve(
04161                   fetk[icalc]->am,
04162                   prob,
04163                   meth,
04164                   fetk[icalc]->lmax,
04165                   fetk[icalc]->ltol,
04166                   prec,
04167                   fetk[icalc]->gues,
04168                   fetk[icalc]->pjac
04169                   );
04170 #endif
04171     }
04172
04173     return 1;
04174 }
04175
04179 VPUBLIC int energyFE(NOsh *nosh,
04180                      int icalc,
04181                      Vfetk *fetk[NOSH_MAXCALC],
04182                      int *nenergy,
04183                      double *totEnergy,
04184                      double *qfEnergy,
04185                      double *qmEnergy,
04186                      double *dielEnergy
04187                     ) {
04188
04189     FEMparm *feparm = nosh->calc[icalc]->femparm;
04190     PBEparm *pbeparm = nosh->calc[icalc]->pbeparm;
04192     *nenergy = 1;
04193     *totEnergy = 0;
04194
04201     if (nosh->bogus == 0) {
04202         if ((pbeparm->pbetype==PBE_NPBE) ||
04203             (pbeparm->pbetype==PBE_NRPBE) ||
04204             (pbeparm->pbetype == PBE_SMPBE)) {
04205             *totEnergy = Vfetk_energy(fetk[icalc], -1, 1); /* Last parameter indicates NPBE */
04206         } else if ((pbeparm->pbetype==PBE_LPBE) ||
04207                    (pbeparm->pbetype==PBE_LRPBE)) {
04208             *totEnergy = Vfetk_energy(fetk[icalc], -1, 0); /* Last parameter indicates LPBE */
04209         } else {
04210             VASSERT(0);
04211         }
04212
04213 #ifndef VAPBSQUIET
04214         Vnm_tprint(1, "      Total electrostatic energy = %1.12E kJ/mol\n",
04215                    Vunit_kb*pbeparm->temp*(1e-3)*Vunit_Na*(*totEnergy));
04216         fflush(stdout);
04217 #endif
04218     }
04219
```

```
04220     if (pbeparm->calcenergy == PCE_COMPS) {
04221         Vnm_tprint(2, "Error!  Verbose energy evaluation not available for FEM yet!\n");
04222         Vnm_tprint(2, "E-mail nathan.baker@pnl.gov if you want this.\n");
04223         *qfEnergy = 0;
04224         *qmEnergy = 0;
04225         *dielEnergy = 0;
04226     } else {
04227         *nenergy = 0;
04228     }
04229     return 1;
04230 }
04231
04239 VPUBLIC int postRefineFE(int icalc,
04240                          FEMparm *feparm,
04241                          Vfetk *fetk[NOSH_MAXCALC]
04242                         ) {
04243
04244     int nverts,
04245         marked;
04247     nverts = Gem_numVV(fetk[icalc]->gm);
04248     if (nverts > feparm->maxvert) {
04249         Vnm_tprint(1, "    Current number of vertices (%d) exceeds max (%d)!\n",
04250                    nverts, feparm->maxvert);
04251         return 0;
04252     }
04253     Vnm_tprint(1, "      Mesh currently has %d vertices\n", nverts);
04254
04255     switch(feparm->akeySOLVE) {
04256         case FRT_UNIF:
04257             Vnm_tprint(1, "      Commencing uniform refinement.\n");
04258             break;
04259         case FRT_GEOM:
04260             Vnm_tprint(1, "      Commencing geometry-based refinement.\n");
04261             break;
04262         case FRT_RESI:
04263             Vnm_tprint(1, "      Commencing residual-based refinement.\n");
04264             break;
04265         case FRT_DUAL:
04266             Vnm_tprint(1, "      Commencing dual problem-based refinement.\n");
04267             break;
04268         case FRT_LOCA:
04269             Vnm_tprint(1, "      Commencing local-based refinement.\n.");
04270             break;
04271         default:
04272             Vnm_tprint(2, "      Error -- unknown refinement type (%d)!\n",
04273                        feparm->akeySOLVE);
04274             return 0;
04275             break;
04276     }
04277
04278     /* Run MC's refinement algorithm */
04279     marked = AM_markRefine(fetk[icalc]->am, feparm->akeySOLVE, -1,
04280                            feparm->ekey, feparm->etol);
04281     if (marked == 0) {
04282         Vnm_tprint(1, "      Marked 0 simps; hit error/size tolerance.\n");
04283         return 0;
04284     }
04285     Vnm_tprint(1, "      Have %d verts, marked %d.  Refining...\n", nverts,
04286                marked);
04287     AM_refine(fetk[icalc]->am, 0, 0, feparm->pkey);
04288     nverts = Gem_numVV(fetk[icalc]->gm);
04289     Vnm_tprint(1, "      Done refining; have %d verts.\n", nverts);
04290     //Vnm_redirect(0); // Redirect output to io.mc
04291     Gem_shapeChk(fetk[icalc]->gm); // Traverse simplices and check shapes using the geometry manager.
04292     //Vnm_redirect(1);
04293
04294     return 1;
04295 }
04296
04300 VPUBLIC int writedataFE(int rank,
04301                         NOsh *nosh,
04302                         PBEparm *pbeparm,
04303                         Vfetk *fetk
04304                        ) {
04305
04306     char writestem[VMAX_ARGLEN];
04307     char outpath[VMAX_ARGLEN];
04308     int i,
04309         writeit;
04310     AM *am;
04311     Bvec *vec;
```

```
04313     if (nosh->bogus) return 1;
04314
04315     am = fetk->am;
04316     vec = am->w0;
04317
04318     for (i=0; i<pbeparm->numwrite; i++) {
04319
04320         writeit = 1;
04321
04322         switch (pbeparm->writetype[i]) {
04323
04324             case VDT_CHARGE:
04325
04326                 Vnm_tprint(2, "    Sorry; can't write charge distribution for FEM!\n");
04327                 writeit = 0;
04328                 break;
04329
04330             case VDT_POT:
04331
04332                 Vnm_tprint(1, "    Writing potential to ");
04333                 Vfetk_fillArray(fetk, vec, VDT_POT);
04334                 break;
04335
04336             case VDT_SMOL:
04337
04338                 Vnm_tprint(1, "    Writing molecular accessibility to ");
04339                 Vfetk_fillArray(fetk, vec, VDT_SMOL);
04340                 break;
04341
04342             case VDT_SSPL:
04343
04344                 Vnm_tprint(1, "    Writing spline-based accessibility to ");
04345                 Vfetk_fillArray(fetk, vec, VDT_SSPL);
04346                 break;
04347
04348             case VDT_VDW:
04349
04350                 Vnm_tprint(1, "    Writing van der Waals accessibility to ");
04351                 Vfetk_fillArray(fetk, vec, VDT_VDW);
04352                 break;
04353
04354             case VDT_IVDW:
04355
04356                 Vnm_tprint(1, "    Writing ion accessibility to ");
04357                 Vfetk_fillArray(fetk, vec, VDT_IVDW);
04358                 break;
04359
04360             case VDT_LAP:
04361
04362                 Vnm_tprint(2, "    Sorry; can't write charge distribution for FEM!\n");
04363                 writeit = 0;
04364                 break;
04365
04366             case VDT_EDENS:
04367
04368                 Vnm_tprint(2, "    Sorry; can't write energy density for FEM!\n");
04369                 writeit = 0;
04370                 break;
04371
04372             case VDT_NDENS:
04373
04374                 Vnm_tprint(1, "    Writing number density to ");
04375                 Vfetk_fillArray(fetk, vec, VDT_NDENS);
04376                 break;
04377
04378             case VDT_QDENS:
04379
04380                 Vnm_tprint(1, "    Writing charge density to ");
04381                 Vfetk_fillArray(fetk, vec, VDT_QDENS);
04382                 break;
04383
04384             case VDT_DIELX:
04385
04386                 Vnm_tprint(2, "    Sorry; can't write x-shifted dielectric map for FEM!\n");
04387                 writeit = 0;
04388                 break;
04389
04390             case VDT_DIELY:
04391
04392                 Vnm_tprint(2, "    Sorry; can't write y-shifted dielectric map for FEM!\n");
04393                 writeit = 0;
```

```
04394                 break;
04395
04396             case VDT_DIELZ:
04397
04398                 Vnm_tprint(2, "    Sorry; can't write z-shifted dielectric map for FEM!\n");
04399                 writeit = 0;
04400                 break;
04401
04402             case VDT_KAPPA:
04403
04404                 Vnm_tprint(1, "    Sorry; can't write kappa map for FEM!\n");
04405                 writeit = 0;
04406                 break;
04407
04408             case VDT_ATOMPOT:
04409
04410                 Vnm_tprint(1, "    Sorry; can't write atom potentials for FEM!\n");
04411                 writeit = 0;
04412                 break;
04413
04414             default:
04415
04416                 Vnm_tprint(2, "Invalid data type for writing!\n");
04417                 writeit = 0;
04418                 return 0;
04419         }
04420
04421         if (!writeit) return 0;
04422
04423
04424 #ifdef HAVE_MPI_H
04425         sprintf(writestem, "%s-PE%d", pbeparm->writestem[i], rank);
04426 #else
04427         if(nosh->ispara){
04428             sprintf(writestem, "%s-PE%d", pbeparm->writestem[i],nosh->proc_rank);
04429         }else{
04430             sprintf(writestem, "%s", pbeparm->writestem[i]);
04431         }
04432 #endif
04433
04434         switch (pbeparm->writefmt[i]) {
04435
04436             case VDF_DX:
04437                 sprintf(outpath, "%s.%s", writestem, "dx");
04438                 Vnm_tprint(1, "%s\n", outpath);
04439                 Vfetk_write(fetk, "FILE", "ASC", VNULL, outpath, vec, VDF_DX);
04440                 break;
04441
04442             case VDF_DXBIN:
04443               //TODO: probably change some or all of below.
04444         sprintf(outpath, "%s.%s", writestem, "dxbin");
04445         Vnm_tprint(1, "%s\n", outpath);
04446         Vfetk_write(fetk, "FILE", "ASC", VNULL, outpath, vec, VDF_DXBIN);
04447         break;
04448
04449             case VDF_AVS:
04450                 sprintf(outpath, "%s.%s", writestem, "ucd");
04451                 Vnm_tprint(1, "%s\n", outpath);
04452                 Vfetk_write(fetk, "FILE", "ASC", VNULL, outpath, vec, VDF_AVS);
04453                 break;
04454
04455             case VDF_UHBD:
04456                 Vnm_tprint(2, "UHBD format not supported for FEM!\n");
04457                 break;
04458
04459             case VDF_MCSF:
04460                 Vnm_tprint(2, "MCSF format not supported yet!\n");
04461                 break;
04462
04463             default:
04464                 Vnm_tprint(2, "Bogus data format (%d)!\n",
04465                            pbeparm->writefmt[i]);
04466                 break;
04467         }
04468
04469     }
04470
04471     return 1;
04472 }
04473 #endif /* ifdef HAVE_MCX_H */
04474
```

```
04475 VPUBLIC int initAPOL(NOsh *nosh,
04476                       Vmem *mem,
04477                       Vparam *param,
04478                       APOLparm *apolparm,
04479                       int *nforce,
04480                       AtomForce **atomForce,
04481                       Valist *alist
04482                      ) {
04483     int i,
04484         natoms,
04485         len,
04486         inhash[3],
04487         rc = 0;
04489     time_t ts;
04490     Vclist *clist = VNULL;
04491     Vacc *acc = VNULL;
04492     Vatom *atom = VNULL;
04493     Vparam_AtomData *atomData = VNULL;
04495     double sasa,
04496            sav,
04497            nhash[3],
04498            sradPad,
04499            x,
04500            y,
04501            z,
04502            atomRadius,
04503            srad,
04504            *atomsasa,
04505            *atomwcaEnergy,
04506            energy = 0.0,
04507            dist,
04508            charge,
04509            xmin,
04510            xmax,
04511            ymin,
04512            ymax,
04513            zmin,
04514            zmax,
04515            disp[3],
04516            center[3],
04517            soluteXlen,
04518            soluteYlen,
04519            soluteZlen;
04521     atomsasa = (double *)Vmem_malloc(VNULL, Valist_getNumberAtoms(alist), sizeof(double));
04522     atomwcaEnergy = (double *)Vmem_malloc(VNULL, Valist_getNumberAtoms(alist), sizeof(double));
04523
04524     /* Determine solute length and charge*/
04525     atom = Valist_getAtom(alist, 0);
04526     xmin = Vatom_getPosition(atom)[0];
04527     xmax = Vatom_getPosition(atom)[0];
04528     ymin = Vatom_getPosition(atom)[1];
04529     ymax = Vatom_getPosition(atom)[1];
04530     zmin = Vatom_getPosition(atom)[2];
04531     zmax = Vatom_getPosition(atom)[2];
04532     charge = 0;
04533     natoms = Valist_getNumberAtoms(alist);
04534
04535     for (i=0; i < natoms; i++) {
04536         atom = Valist_getAtom(alist, i);
04537         atomRadius = Vatom_getRadius(atom);
04538         x = Vatom_getPosition(atom)[0];
04539         y = Vatom_getPosition(atom)[1];
04540         z = Vatom_getPosition(atom)[2];
04541         if ((x+atomRadius) > xmax) xmax = x + atomRadius;
04542         if ((x-atomRadius) < xmin) xmin = x - atomRadius;
04543         if ((y+atomRadius) > ymax) ymax = y + atomRadius;
04544         if ((y-atomRadius) < ymin) ymin = y - atomRadius;
04545         if ((z+atomRadius) > zmax) zmax = z + atomRadius;
04546         if ((z-atomRadius) < zmin) zmin = z - atomRadius;
04547         disp[0] = (x - center[0]);
04548         disp[1] = (y - center[1]);
04549         disp[2] = (z - center[2]);
04550         dist = (disp[0]*disp[0]) + (disp[1]*disp[1]) + (disp[2]*disp[2]);
04551         dist = VSQRT(dist) + atomRadius;
04552         charge += Vatom_getCharge(Valist_getAtom(alist, i));
04553     }
04554     soluteXlen = xmax - xmin;
04555     soluteYlen = ymax - ymin;
04556     soluteZlen = zmax - zmin;
04557
04558     /* Set up the hash table for the cell list */
```

```
04559     Vnm_print(0, "APOL: Setting up hash table and accessibility object...\n");
04560     nhash[0] = soluteXlen/0.5;
04561     nhash[1] = soluteYlen/0.5;
04562     nhash[2] = soluteZlen/0.5;
04563     for (i=0; i<3; i++) inhash[i] = (int)(nhash[i]);
04564
04565     for (i=0;i<3;i++){
04566         if (inhash[i] < 3) inhash[i] = 3;
04567         if (inhash[i] > MAX_HASH_DIM) inhash[i] = MAX_HASH_DIM;
04568     }
04569
04570     /* Pad the radius by 2x the maximum displacement value */
04571     srad = apolparm->srad;
04572     sradPad = srad + (2*apolparm->dpos);
04573     clist = Vclist_ctor(alist, sradPad , inhash, CLIST_AUTO_DOMAIN,
04574                                     VNULL, VNULL);
04575     acc = Vacc_ctor(alist, clist, apolparm->sdens);
04576
04577     /* Get WAT (water) LJ parameters from Vparam object */
04578     if (param == VNULL && (apolparm->bconc != 0.0)) {
04579         Vnm_tprint(2, "initAPOL:  Got NULL Vparam object!\n");
04580         Vnm_tprint(2, "initAPOL:  You are performing an apolar calculation with the van der Waals integral
    term,\n");
04581         Vnm_tprint(2, "initAPOL:  this term requires van der Waals parameters which are not available from
    the \n");
04582         Vnm_tprint(2, "initAPOL:  PQR file. Therefore, you need to supply a parameter file with the parm
    keyword,\n");
04583         Vnm_tprint(2, "initAPOL:  for example,\n");
04584         Vnm_tprint(2, "initAPOL:    read parm flat amber94.dat end\n");
04585         Vnm_tprint(2, "initAPOL:  where the relevant parameter files can be found in
    apbs/tools/conversion/param/vparam.\n");
04586         return VRC_FAILURE;
04587     }
04588
04589     if (apolparm->bconc != 0.0){
04590         atomData = Vparam_getAtomData(param, "WAT", "OW");
04591         if (atomData == VNULL) atomData = Vparam_getAtomData(param, "WAT", "O");
04592         if (atomData == VNULL) {
04593             Vnm_tprint(2, "initAPOL:  Couldn't find parameters for WAT OW or WAT O!\n");
04594             Vnm_tprint(2, "initAPOL:  These parameters must be present in your file\n");
04595             Vnm_tprint(2, "initAPOL:  for apolar calculations.\n");
04596             return VRC_FAILURE;
04597         }
04598         apolparm->watepsilon = atomData->epsilon;
04599         apolparm->watsigma = atomData->radius;
04600         apolparm->setwat = 1;
04601     }
04602
04603     /* Calculate Energy and Forces */
04604     if(apolparm->calcforce) {
04605         rc = forceAPOL(acc, mem, apolparm, nforce, atomForce, alist, clist);
04606         if(rc == VRC_FAILURE) {
04607             Vnm_print(2, "Error in apolar force calculation!\n");
04608             return VRC_FAILURE;
04609         }
04610     }
04611
04612     /* Get the SAV and SAS */
04613     sasa = 0.0;
04614     sav = 0.0;
04615
04616     if (apolparm->calcenergy) {
04617         len = Valist_getNumberAtoms(alist);
04618
04619         if (VABS(apolparm->gamma) > VSMALL) {
04620             /* Total Solvent Accessible Surface Area (SASA) */
04621             apolparm->sasa = Vacc_totalSASA(acc, srad);
04622             /* SASA for each atom */
04623             for (i = 0; i < len; i++) {
04624                 atom = Valist_getAtom(alist, i);
04625                 atomsasa[i] = Vacc_atomSASA(acc, srad, atom);
04626             }
04627         } else {
04628             /* Total Solvent Accessible Surface Area (SASA) set to zero */
04629             apolparm->sasa = 0.0;
04630             /* SASA for each atom set to zero*/
04631             for (i = 0; i < len; i++) {
04632                 atomsasa[i] = 0.0;
04633             }
04634         }
04635
```

```
04636         /* Inflated van der Waals accessibility */
04637         if (VABS(apolparm->press) > VSMALL){
04638             apolparm->sav = Vacc_totalSAV(acc, clist, apolparm, srad);
04639         } else {
04640             apolparm->sav = 0.0;
04641         }
04642 
04643         /* wcaEnergy integral code */
04644         if (VABS(apolparm->bconc) > VSMALL) {
04645             /* wcaEnergy for each atom */
04646             for (i = 0; i < len; i++) {
04647                 rc = Vacc_wcaEnergyAtom(acc, apolparm, alist, clist, i, &energy);
04648                 if (rc == 0)  {
04649                     Vnm_print(2, "Error in apolar energy calculation!\n");
04650                     return 0;
04651                 }
04652                 atomwcaEnergy[i] = energy;
04653             }
04654             /* Total WCA Energy */
04655             rc = Vacc_wcaEnergy(acc, apolparm, alist, clist);
04656             if (rc == 0) {
04657                 Vnm_print(2, "Error in apolar energy calculation!\n");
04658                 return 0;
04659             }
04660         } else {
04661             apolparm->wcaEnergy = 0.0;
04662         }
04663         energyAPOL(apolparm, apolparm->sasa, apolparm->sav, atomsasa, atomwcaEnergy,
      Valist_getNumberAtoms(alist));
04664     }
04665 
04666     Vmem_free(VNULL, Valist_getNumberAtoms(alist), sizeof(double), (void **)&(atomsasa));
04667     Vmem_free(VNULL, Valist_getNumberAtoms(alist), sizeof(double), (void **)&(atomwcaEnergy));
04668     Vclist_dtor(&clist);
04669     Vacc_dtor(&acc);
04670 
04671     return VRC_SUCCESS;
04672 }
04673 
04674 VPUBLIC int energyAPOL(APOLparm *apolparm,
04675                        double sasa,
04676                        double sav,
04677                        double atomsasa[],
04678                        double atomwcaEnergy[],
04679                        int numatoms
04680                       ){
04681 
04682     double energy = 0.0;
04683     int i = 0;
04684 
04685 #ifndef VAPBSQUIET
04686     Vnm_print(1,"\nSolvent Accessible Surface Area (SASA) for each atom:\n");
04687     for (i = 0; i < numatoms; i++) {
04688         Vnm_print(1,"  SASA for atom %i: %1.12E\n", i, atomsasa[i]);
04689     }
04690 
04691     Vnm_print(1,"\nTotal solvent accessible surface area: %g A^2\n",sasa);
04692 #endif
04693 
04694     switch(apolparm->calcenergy){
04695         case ACE_NO:
04696             break;
04697         case ACE_COMPS:
04698             Vnm_print(1,"energyAPOL: Cannot calculate component energy, skipping.\n");
04699             break;
04700         case ACE_TOTAL:
04701             energy = (apolparm->gamma*sasa) + (apolparm->press*sav)
04702                      + (apolparm->wcaEnergy);
04703 #ifndef VAPBSQUIET
04704             Vnm_print(1,"\nSurface tension*area energies (gamma * SASA) for each atom:\n");
04705             for (i = 0; i < numatoms; i++) {
04706                 Vnm_print(1,"  Surface tension*area energy for atom %i: %1.12E\n", i,
      apolparm->gamma*atomsasa[i]);
04707             }
04708 
04709             Vnm_print(1,"\nTotal surface tension energy: %g kJ/mol\n", apolparm->gamma*sasa);
04710             Vnm_print(1,"\nTotal solvent accessible volume: %g A^3\n", sav);
04711             Vnm_print(1,"\nTotal pressure*volume energy: %g kJ/mol\n", apolparm->press*sav);
04712             Vnm_print(1,"\nWCA dispersion Energies for each atom:\n");
04713             for (i = 0; i < numatoms; i++) {
04714                 Vnm_print(1,"  WCA energy for atom %i: %1.12E\n", i, atomwcaEnergy[i]);
```

```
04715             }
04716
04717             Vnm_print(1,"\nTotal WCA energy: %g kJ/mol\n", (apolparm->wcaEnergy));
04718             Vnm_print(1,"\nTotal non-polar energy = %1.12E kJ/mol\n", energy);
04719 #endif
04720             break;
04721         default:
04722             Vnm_print(2,"energyAPOL: Error in energyAPOL. Unknown option.\n");
04723             break;
04724     }
04725
04726     return VRC_SUCCESS;
04727 }
04728
04729 VPUBLIC int forceAPOL(Vacc *acc,
04730                       Vmem *mem,
04731                       APOLparm *apolparm,
04732                       int *nforce,
04733                       AtomForce **atomForce,
04734                       Valist *alist,
04735                       Vclist *clist
04736                      ) {
04737                          time_t ts, ts_main, ts_sub;
04738     int i,
04739         j,
04740         natom;
04741
04742     double srad, /* Probe radius */
04743            xF,
04744            yF,
04745            zF,  /* Individual forces */
04746            press,
04747            gamma,
04748            offset,
04749            bconc,
04750            dSASA[3],
04751            dSAV[3],
04752            force[3],
04753            *apos;
04754
04755     Vatom *atom = VNULL;
04756     ts_main = clock();
04757
04758     srad = apolparm->srad;
04759     press = apolparm->press;
04760     gamma = apolparm->gamma;
04761     offset = apolparm->dpos;
04762     bconc = apolparm->bconc;
04763
04764     natom = Valist_getNumberAtoms(alist);
04765
04766     /* Check to see if we need to build the surface */
04767     Vnm_print(0, "forceAPOL: Trying atom surf...\n");
04768     ts = clock();
04769     if (acc->surf == VNULL) {
04770         acc->surf = (VaccSurf**)Vmem_malloc(acc->mem, natom, sizeof(VaccSurf *));
04771         for (i=0; i<natom; i++) {
04772             atom = Valist_getAtom(acc->alist, i);
04773             //apos = Vatom_getPosition(atom); // apos never referenced? - Peter
04774             /* NOTE:  RIGHT NOW WE DO THIS FOR THE ENTIRE MOLECULE WHICH IS
04775              * INCREDIBLY INEFFICIENT, PARTICULARLY DURING FOCUSING!!! */
04776             acc->surf[i] = Vacc_atomSurf(acc, atom, acc->refSphere, srad);
04777         }
04778     }
04779     Vnm_print(0, "forceAPOL: atom surf: Time elapsed: %f\n", ((double)clock() - ts) / CLOCKS_PER_SEC);
04780
04781     if(apolparm->calcforce == ACF_TOTAL){
04782         Vnm_print(0, "forceAPOL: calcforce == ACF_TOTAL\n");
04783         ts = clock();
04784
04785         *nforce = 1;
04786         if(*atomForce != VNULL){
04787             Vmem_free(mem,*nforce,sizeof(AtomForce), (void **)atomForce);
04788         }
04789
04790             *atomForce = (AtomForce *)Vmem_malloc(mem, *nforce,
04791                                                   sizeof(AtomForce));
04792
04793         /* Clear out force arrays */
04794         for (j=0; j<3; j++) {
04795             (*atomForce)[0].sasaForce[j] = 0.0;
```

```
04796             (*atomForce)[0].savForce[j] = 0.0;
04797             (*atomForce)[0].wcaForce[j] = 0.0;
04798         }
04799
04800         // problem block
04801         for (i=0; i<natom; i++) {
04802             atom = Valist_getAtom(alist, i);
04803
04804             for(j=0;j<3;j++){
04805                 dSASA[j] = 0.0;
04806                 dSAV[j] = 0.0;
04807                 force[j] = 0.0;
04808             }
04809
04810             if(VABS(gamma) > VSMALL) {
04811                 Vacc_atomdSASA(acc, offset, srad, atom, dSASA);
04812             }
04813             if(VABS(press) > VSMALL) {
04814                 Vacc_atomdSAV(acc, srad, atom, dSAV);
04815             }
04816             if(VABS(bconc) > VSMALL) {
04817                 Vacc_wcaForceAtom(acc, apolparm, clist, atom, force);
04818             }
04819
04820             for(j=0;j<3;j++){
04821                 (*atomForce)[0].sasaForce[j] += dSASA[j];
04822                 (*atomForce)[0].savForce[j] += dSAV[j];
04823                 (*atomForce)[0].wcaForce[j] += force[j];
04824             }
04825         }
04826         // end block
04827
04828         Vnm_tprint( 1, "  Printing net forces (kJ/mol/A)\n");
04829         Vnm_tprint( 1, "  Legend:\n");
04830         Vnm_tprint( 1, "    sasa  -- SASA force\n");
04831         Vnm_tprint( 1, "    sav   -- SAV force\n");
04832         Vnm_tprint( 1, "    wca   -- WCA force\n\n");
04833
04834         Vnm_tprint( 1, "  sasa  %4.3e %4.3e %4.3e\n",
04835                     (*atomForce)[0].sasaForce[0],
04836                     (*atomForce)[0].sasaForce[1],
04837                     (*atomForce)[0].sasaForce[2]);
04838         Vnm_tprint( 1, "  sav   %4.3e %4.3e %4.3e\n",
04839                     (*atomForce)[0].savForce[0],
04840                     (*atomForce)[0].savForce[1],
04841                     (*atomForce)[0].savForce[2]);
04842         Vnm_tprint( 1, "  wca   %4.3e %4.3e %4.3e\n",
04843                     (*atomForce)[0].wcaForce[0],
04844                     (*atomForce)[0].wcaForce[1],
04845                     (*atomForce)[0].wcaForce[2]);
04846
04847         Vnm_print(0, "forceAPOL: calcforce == ACF_TOTAL: %f\n", ((double)clock() - ts) / CLOCKS_PER_SEC);
04848     } else if (apolparm->calcforce == ACF_COMPS ){
04849         *nforce = Valist_getNumberAtoms(alist);
04850         if(*atomForce == VNULL){
04851             *atomForce = (AtomForce *)Vmem_malloc(mem, *nforce,
04852                                                   sizeof(AtomForce));
04853         }else{
04854             Vmem_free(mem,*nforce,sizeof(AtomForce), (void **)atomForce);
04855             *atomForce = (AtomForce *)Vmem_malloc(mem, *nforce,
04856                                                   sizeof(AtomForce));
04857         }
04858
04859 #ifndef VAPBSQUIET
04860         Vnm_tprint( 1, "  Printing per atom forces (kJ/mol/A)\n");
04861         Vnm_tprint( 1, "  Legend:\n");
04862         Vnm_tprint( 1, "    tot  n -- Total force for atom n\n");
04863         Vnm_tprint( 1, "    sasa n -- SASA force for atom n\n");
04864         Vnm_tprint( 1, "    sav  n -- SAV force for atom n\n");
04865         Vnm_tprint( 1, "    wca  n -- WCA force for atom n\n\n");
04866
04867         Vnm_tprint( 1, "    gamma    %f\n" \
04868                        "    pressure %f\n" \
04869                        "    bconc    %f \n\n",
04870                             gamma,press,bconc);
04871 #endif
04872
04873         for (i=0; i<natom; i++) {
04874             atom = Valist_getAtom(alist, i);
04875
04876             for(j=0;j<3;j++){
```

```
04877                 dSASA[j] = 0.0;
04878                 dSAV[j] = 0.0;
04879                 force[j] = 0.0;
04880             }
04881
04882             /* Clear out force arrays */
04883             for (j=0; j<3; j++) {
04884                 (*atomForce)[i].sasaForce[j] = 0.0;
04885                 (*atomForce)[i].savForce[j] = 0.0;
04886                 (*atomForce)[i].wcaForce[j] = 0.0;
04887             }
04888
04889             if(VABS(gamma) > VSMALL) Vacc_atomdSASA(acc, offset, srad, atom, dSASA);
04890             if(VABS(press) > VSMALL) Vacc_atomdSAV(acc, srad, atom, dSAV);
04891             if(VABS(bconc) > VSMALL) Vacc_wcaForceAtom(acc,apolparm,clist,atom,force);
04892
04893             xF = -((gamma*dSASA[0]) + (press*dSAV[0]) + (bconc*force[0]));
04894             yF = -((gamma*dSASA[1]) + (press*dSAV[1]) + (bconc*force[1]));
04895             zF = -((gamma*dSASA[2]) + (press*dSAV[2]) + (bconc*force[2]));
04896
04897             for(j=0;j<3;j++){
04898                 (*atomForce)[i].sasaForce[j] += dSASA[j];
04899                 (*atomForce)[i].savForce[j] += dSAV[j];
04900                 (*atomForce)[i].wcaForce[j] += force[j];
04901             }
04902
04903 #ifndef VAPBSQUIET
04904             Vnm_print( 1, "  tot  %i %4.3e %4.3e %4.3e\n",
04905                        i,
04906                        xF,
04907                        yF,
04908                        zF);
04909             Vnm_print( 1, "  sasa %i %4.3e %4.3e %4.3e\n",
04910                        i,
04911                        (*atomForce)[i].sasaForce[0],
04912                        (*atomForce)[i].sasaForce[1],
04913                        (*atomForce)[i].sasaForce[2]);
04914             Vnm_print( 1, "  sav  %i %4.3e %4.3e %4.3e\n",
04915                        i,
04916                        (*atomForce)[i].savForce[0],
04917                        (*atomForce)[i].savForce[1],
04918                        (*atomForce)[i].savForce[2]);
04919             Vnm_print( 1, "  wca  %i %4.3e %4.3e %4.3e\n",
04920                        i,
04921                        (*atomForce)[i].wcaForce[0],
04922                        (*atomForce)[i].wcaForce[1],
04923                        (*atomForce)[i].wcaForce[2]);
04924 #endif
04925
04926         }
04927     } else *nforce = 0;
04928
04929 #ifndef VAPBSQUIET
04930     Vnm_print(1,"\n");
04931 #endif
04932
04933     Vnm_print(0, "forceAPOL: Time elapsed: %f\n", ((double)clock() - ts_main) / CLOCKS_PER_SEC);
04934     return VRC_SUCCESS;
04935 }
04936
04937
04938 #ifdef ENABLE_BEM
04942 VPUBLIC int initBEM(int icalc,
04943                     NOsh *nosh,
04944                     BEMparm *bemparm,
04945                     PBEparm *pbeparm,
04946                     Vpbe *pbe[NOSH_MAXCALC]
04947                    ) {
04948
04949     Vnm_tstart(APBS_TIMER_SETUP, "Setup timer");
04950
04951     /* Setup time statistics */
04952     Vnm_tstop(APBS_TIMER_SETUP, "Setup timer");
04953
04954     return 1;
04955
04956 }
04957
04958 VPUBLIC void killBEM(NOsh *nosh, Vpbe *pbe[NOSH_MAXCALC]
04959                      ) {
04960
```

```
04961         int i;
04962
04963 #ifndef VAPBSQUIET
04964     Vnm_tprint(1, "Destroying boundary element structures.\n");
04965 #endif
04966
04967
04968 }
04969
04970
04971 void apbs2tabipb_(TABIPBparm* tabiparm,
04972                   TABIPBvars* tabivars);
04973
04974 VPUBLIC int solveBEM(Valist* molecules[NOSH_MAXMOL],
04975                      NOsh *nosh,
04976                      PBEparm *pbeparm,
04977                      BEMparm *bemparm,
04978                      BEMparm_CalcType type) {
04979
04980     int nx,
04981         ny,
04982         nz,
04983         i;
04984
04985     if (nosh != VNULL) {
04986         if (nosh->bogus) return 1;
04987     }
04988
04989     Vnm_tstart(APBS_TIMER_SOLVER, "Solver timer");
04990
04991     TABIPBparm *tabiparm = (TABIPBparm*)calloc(1,sizeof(TABIPBparm));
04992
04993     sprintf(tabiparm->fpath, "");
04994     strncpy(tabiparm->fname, nosh->molpath[0],4);
04995     tabiparm->fname[4] = '\0';
04996     tabiparm->density = pbeparm->sdens;
04997     tabiparm->probe_radius = pbeparm->srad;
04998
04999     tabiparm->epsp = pbeparm->pdie;
05000     tabiparm->epsw = pbeparm->sdie;
05001     tabiparm->bulk_strength = 0.0;
05002     for (i=0; i<pbeparm->nion; i++)
05003         tabiparm->bulk_strength += pbeparm->ionc[i]
05004                                    *pbeparm->ionq[i]*pbeparm->ionq[i];
05005     tabiparm->temp = pbeparm->temp;
05006
05007     tabiparm->order = bemparm->tree_order;
05008     tabiparm->maxparnode = bemparm->tree_n0;
05009     tabiparm->theta = bemparm->mac;
05010
05011     tabiparm->mesh_flag = bemparm->mesh;
05012
05013     tabiparm->number_of_lines = Valist_getNumberAtoms(molecules[0]);
05014
05015     tabiparm->output_datafile = bemparm->outdata;
05016
05017     TABIPBvars *tabivars = (TABIPBvars*)calloc(1,sizeof(TABIPBvars));
05018     if ((tabivars->chrpos = (double *) malloc(3 * tabiparm->number_of_lines * sizeof(double))) == NULL) {
05019             printf("Error in allocating t_chrpos!\n");
05020     }
05021     if ((tabivars->atmchr = (double *) malloc(tabiparm->number_of_lines * sizeof(double))) == NULL) {
05022             printf("Error in allocating t_atmchr!\n");
05023     }
05024     if ((tabivars->atmrad = (double *) malloc(tabiparm->number_of_lines * sizeof(double))) == NULL) {
05025             printf("Error in allocating t_atmrad!\n");
05026     }
05027
05028     Vatom *atom;
05029
05030     for (i = 0; i < tabiparm->number_of_lines; i++){
05031       atom = Valist_getAtom(molecules[0], i);
05032       tabivars->chrpos[3*i] = Vatom_getPosition(atom)[0];
05033       tabivars->chrpos[3*i + 1] = Vatom_getPosition(atom)[1];
05034       tabivars->chrpos[3*i + 2] = Vatom_getPosition(atom)[2];
05035       tabivars->atmchr[i] = Vatom_getCharge(atom);
05036       tabivars->atmrad[i] = Vatom_getRadius(atom);
05037     }
05038
05039 //apbs2tabipb(TABIPBparm* tabiparm, Valist* molecules[NOSH_MAXMOL]);
05040     apbs2tabipb_(tabiparm, tabivars);
05041
```

```
05042     free(tabiparm);
05043     free(tabivars->chrpos);
05044     free(tabivars->atmchr);
05045     free(tabivars->atmrad);
05046     free(tabivars->vert_ptl); // allocate in output_potential()
05047     free(tabivars->xvct); // allocate in output_potential()
05048     free_matrix(tabivars->vert); // allocate in output_potential()
05049     free_matrix(tabivars->snrm); // allocate in output_potential()
05050     free_matrix(tabivars->face); // allocate in output_potential()
05051
05052     Vnm_tprint(1, "\n\nReturning to APBS caller...\n\n");
05053     Vnm_tprint(1, "Solvation energy and Coulombic energy in kJ/mol...\n\n");
05054     Vnm_tprint(1, "  Global net ELEC energy = %1.12E\n", tabivars->soleng);
05055     Vnm_tprint(1, "  Global net COULOMBIC energy = %1.12E\n\n", tabivars->couleng);
05056
05057     free(tabivars);
05058
05059     Vnm_tstop(APBS_TIMER_SOLVER, "Solver timer");
05060
05061     return 1;
05062
05063 }
05064
05065 VPUBLIC int setPartBEM(NOsh *nosh,
05066                        BEMparm *BEMparm
05067                       ) {
05068
05069     int j;
05070     double partMin[3],
05071            partMax[3];
05072
05073     if (nosh->bogus) return 1;
05074
05075     return 1;
05076
05077 }
05078
05079 VPUBLIC int energyBEM(NOsh *nosh,
05080                       int icalc,
05081                       int *nenergy,
05082                       double *totEnergy,
05083                       double *qfEnergy,
05084                       double *qmEnergy,
05085                       double *dielEnergy
05086                      ) {
05087
05088     Valist *alist;
05089     Vatom *atom;
05090     int i,
05091         extEnergy;
05092     double tenergy;
05093     BEMparm *bemparm;
05094     PBEparm *pbeparm;
05095
05096     bemparm = nosh->calc[icalc]->bemparm;
05097     pbeparm = nosh->calc[icalc]->pbeparm;
05098
05099     Vnm_tstart(APBS_TIMER_ENERGY, "Energy timer");
05100     Vnm_tstop(APBS_TIMER_ENERGY, "Energy timer");
05101
05102     return 1;
05103 }
05104
05105 VPUBLIC int forceBEM(
05106                      NOsh *nosh,
05107                      PBEparm *pbeparm,
05108                      BEMparm *bemparm,
05109                      int *nforce,
05110                      AtomForce **atomForce,
05111                      Valist *alist[NOSH_MAXMOL]
05112                     ) {
05113
05114     int j,
05115         k;
05116     double qfForce[3],
05117            dbForce[3],
05118            ibForce[3];
05119
05120     Vnm_tstart(APBS_TIMER_FORCE, "Force timer");
05121
05122 #ifndef VAPBSQUIET
```

```
05123     Vnm_tprint( 1,"  Calculating forces...\n");
05124 #endif
05125
05126     Vnm_tstop(APBS_TIMER_FORCE, "Force timer");
05127
05128     return 1;
05129 }
05130
05131 VPUBLIC void printBEMPARM(BEMparm *bemparm) {
05132
05133 }
05134
05135
05136 VPUBLIC int writedataBEM(int rank,
05137                          NOsh *nosh,
05138                          PBEparm *pbeparm
05139                         ) {
05140
05141     return 1;
05142 }
05143
05144
05145 VPUBLIC int writematBEM(int rank, NOsh *nosh, PBEparm *pbeparm) {
05146
05147
05148     if (nosh->bogus) return 1;
05149     return 1;
05150 }
05151
05152 #endif
05153
05154
05155 #ifdef ENABLE_GEOFLOW
05156
05160 VPUBLIC int solveGeometricFlow( Valist* molecules[NOSH_MAXMOL],
05161                                 NOsh *nosh,
05162                                 PBEparm *pbeparm,
05163                                 APOLparm *apolparm,
05164                                 GEOFLOWparm *parm )
05165 {
05166    //printf("solveGeometricFlow!!!\n");
05167    if (nosh != VNULL) {
05168       if (nosh->bogus) return 1;
05169    }
05170
05171    Vnm_tstart(APBS_TIMER_SOLVER, "Solver timer");
05172
05173    struct GeometricFlowInput geoflowIn = getGeometricFlowParams();
05174
05175    // change any of the parameters you want...
05176    geoflowIn.m_boundaryCondition = (int) pbeparm->bcfl ;
05177    geoflowIn.m_grid[0] = apolparm->grid[0];
05178    geoflowIn.m_grid[1] = apolparm->grid[1];
05179    geoflowIn.m_grid[2] = apolparm->grid[2];
05180    geoflowIn.m_gamma =  apolparm->gamma;
05181    geoflowIn.m_pdie = pbeparm->pdie ;
05182    geoflowIn.m_sdie = pbeparm->sdie ;
05183    geoflowIn.m_press = apolparm->press ;
05184    geoflowIn.m_tol = parm->etol;
05185    geoflowIn.m_bconc = apolparm->bconc ;
05186    geoflowIn.m_vdwdispersion = parm->vdw;
05187    geoflowIn.m_etolSolvation = .01 ;  // to be added?
05188
05189    // debug
05190    //printGeometricFlowStruct( geoflowIn );
05191
05192    //printf("num mols: %i\n", nosh->nmol);
05193    struct GeometricFlowOutput geoflowOut =
05194       runGeometricFlowWrapAPBS( geoflowIn, molecules[0] );
05195
05196    Vnm_tprint( 1,"  Global net energy = %1.12E\n", geoflowOut.m_totalSolvation);
05197    Vnm_tprint( 1,"  Global net ELEC energy = %1.12E\n", geoflowOut.m_elecSolvation);
05198    Vnm_tprint( 1,"  Global net APOL energy = %1.12E\n", geoflowOut.m_nonpolarSolvation);
05199
05200    Vnm_tstop(APBS_TIMER_SOLVER, "Solver timer");
05201
05202    return 1;
05203
05204 }
05205
05206 #endif
```

```
05207
05208 #ifdef ENABLE_PBAM
05209
05213 VPUBLIC int solvePBAM( Valist* molecules[NOSH_MAXMOL],
05214                                NOsh *nosh,
05215                                PBEparm *pbeparm,
05216                                PBAMparm *parm )
05217 {
05218   if (nosh != VNULL) {
05219     if (nosh->bogus) return 1;
05220   }
05221
05222   int i, j;
05223   Vnm_tstart(APBS_TIMER_SOLVER, "Solver timer");
05224   PBAMInput pbamIn = getPBAMParams();
05225
05226   pbamIn.nmol_ = nosh->nmol;
05227
05228   // change any of the parameters you want...
05229   pbamIn.temp_ =  pbeparm->temp;
05230   if (fabs(pbamIn.temp_-0.0) < 1e-3)
05231   {
05232     printf("No temperature specified. Setting to 298.15K\n");
05233     pbamIn.temp_ = 298.15;
05234   }
05235
05236   // Dielectrics
05237   pbamIn.idiel_ = pbeparm->pdie;
05238   pbamIn.sdiel_ = pbeparm->sdie;
05239
05240   // Salt conc
05241   pbamIn.salt_ = parm->salt;
05242
05243   // Runtype: can be energyforce, electrostatics etc
05244   strncpy(pbamIn.runType_, parm->runtype, CHR_MAXLEN);
05245   strncpy(pbamIn.runName_, parm->runname, CHR_MAXLEN);
05246
05247   pbamIn.randOrient_ = parm->setrandorient;
05248
05249   pbamIn.boxLen_ = parm->pbcboxlen;
05250   pbamIn.pbcType_ = parm->setpbcs;
05251
05252   pbamIn.setunits_ = parm->setunits;
05253   if(parm->setunits == 1) strncpy(pbamIn.units_, parm->units, CHR_MAXLEN);
05254
05255   // Electrostatic stuff
05256   if (parm->setgridpt) pbamIn.gridPts_ = parm->gridpt;
05257   strncpy(pbamIn.map3D_, parm->map3dname, CHR_MAXLEN);
05258   pbamIn.grid2Dct_ = parm->grid2Dct;
05259   for (i=0; i<pbamIn.grid2Dct_; i++)
05260   {
05261     strncpy(pbamIn.grid2D_[i], parm->grid2Dname[i], CHR_MAXLEN);
05262     strncpy(pbamIn.grid2Dax_[i], parm->grid2Dax[i], CHR_MAXLEN);
05263     pbamIn.grid2Dloc_[i] = parm->grid2Dloc[i];
05264   }
05265   strncpy(pbamIn.dxname_, parm->dxname, CHR_MAXLEN);
05266
05267   // Dynamics stuff
05268   pbamIn.ntraj_ = parm->ntraj;
05269   strncpy(pbamIn.termCombine_, parm->termcombine, CHR_MAXLEN);
05270
05271   pbamIn.termct_ = parm->termct;
05272   pbamIn.contct_ = parm->confilct;
05273
05274   if (strncmp(pbamIn.runType_, "dynamics", 8)== 0)
05275   {
05276     if (pbamIn.nmol_ > parm->diffct)
05277     {
05278       Vnm_tprint(2, "You need more diffusion information!\n");
05279       return 0;
05280     }
05281
05282     for (i=0; i<pbamIn.nmol_; i++)
05283     {
05284       if (parm->xyzct[i] <  parm->ntraj)
05285       {
05286         Vnm_tprint(2, "For molecule %d, you are missing trajectory!\n", i+1);
05287         return 0;
05288       } else {
05289         for (j=0; j<pbamIn.ntraj_; j++)
05290         {
```

```
05291           strncpy(pbamIn.xyzfil_[i][j], parm->xyzfil[i][j], CHR_MAXLEN);
05292         }
05293       }
05294     }
05295
05296     for (i=0; i<pbamIn.nmol_; i++)
05297     {
05298       strncpy(pbamIn.moveType_[i], parm->moveType[i], CHR_MAXLEN);
05299       pbamIn.transDiff_[i] = parm->transDiff[i];
05300       pbamIn.rotDiff_[i] = parm->rotDiff[i];
05301     }
05302
05303     for (i=0; i<pbamIn.termct_; i++)
05304     {
05305         strncpy(pbamIn.termnam_[i], parm->termnam[i], CHR_MAXLEN);
05306         pbamIn.termnu_[i][0] = parm->termnu[i][0];
05307         pbamIn.termval_[i] = parm->termVal[i];
05308     }
05309
05310     for (i=0; i<pbamIn.contct_; i++)
05311     {
05312         strncpy(pbamIn.confil_[i], parm->confil[i], CHR_MAXLEN);
05313     }
05314
05315   }
05316
05317   // debug
05318   printPBAMStruct( pbamIn );
05319
05320   // Run the darn thing
05321   PBAMOutput pbamOut = runPBAMWrapAPBS( pbamIn, molecules, nosh->nmol );
05322
05323   Vnm_tprint(1, "\n\nReturning to APBS caller...\n\n");
05324
05325   if (!(strncmp(pbamIn.runType_, "dynamics", 8) &&
05326         strncmp(pbamIn.runType_, "energyforce", 11))) {
05327
05328       if (!strncmp(pbamIn.units_, "kcalmol", 7)) {  //scale to kjmol is 4.18400
05329
05330           Vnm_tprint(1, "Interaction energy in kCal/mol...\n\n");
05331
05332           for (int i = 0; i < PBAMPARM_MAXMOL; i++) {
05333               Vnm_tprint(1, "  Molecule %d: Global net ELEC energy = %1.12E\n",
05334                          i+1, pbamOut.energies_[i]);
05335               Vnm_tprint(1, "               Force = (%1.6E, %1.6E, %1.6E)\n\n",
05336                          pbamOut.forces_[i][0], pbamOut.forces_[i][1],
05337                          pbamOut.forces_[i][2]);
05338               if (pbamOut.energies_[i+1] == 0.) break;
05339           }
05340
05341       } else if (!strncmp(pbamIn.units_, "jmol", 4)) {  //scale to kjmol is 0.001
05342
05343           Vnm_tprint(1, "Interaction energy in J/mol...\n\n");
05344
05345           for (int i = 0; i < PBAMPARM_MAXMOL; i++) {
05346               Vnm_tprint(1, "  Molecule %d: Global net ELEC energy = %1.12E\n",
05347                          i+1, pbamOut.energies_[i]);
05348               Vnm_tprint(1, "               Force = (%1.6E, %1.6E, %1.6E)\n\n",
05349                          pbamOut.forces_[i][0], pbamOut.forces_[i][1],
05350                          pbamOut.forces_[i][2]);
05351               if (pbamOut.energies_[i+1] == 0.) break;
05352           }
05353
05354       } else { // if (!strncmp(pbamIn.units_, "kT", 2)) //scale to kjmol is 2.478 @ 298K
05355                                                         // or 0.008315436242 * 298
05356
05357           Vnm_tprint(1, "Interaction energy in kT @ %6.2f K...\n\n", pbamIn.temp_);
05358
05359           for (int i = 0; i < PBAMPARM_MAXMOL; i++) {
05360               Vnm_tprint(1, "  Molecule %d: Global net ELEC energy = %1.12E\n",
05361                          i+1, pbamOut.energies_[i]);
05362               Vnm_tprint(1, "               Force = (%1.6E, %1.6E, %1.6E)\n\n",
05363                          pbamOut.forces_[i][0], pbamOut.forces_[i][1],
05364                          pbamOut.forces_[i][2]);
05365               if (pbamOut.energies_[i+1] == 0.) break;
05366           }
05367       }
05368   }
05369   Vnm_tstop(APBS_TIMER_SOLVER, "Solver timer");
05370
05371   return 1;
```

```
05372
05373 }
05374
05375 #endif
05376
05377 #ifdef ENABLE_PBSAM
05378
05382 VPUBLIC int solvePBSAM( Valist* molecules[NOSH_MAXMOL],
05383                                 NOsh *nosh,
05384                                 PBEparm *pbeparm,
05385                                 PBAMparm *parm,
05386                                 PBSAMparm *samparm )
05387 {
05388   printf("solvePBSAM!!!\n");
05389   char fname_tp[VMAX_ARGLEN];
05390   if (nosh != VNULL) {
05391     if (nosh->bogus) return 1;
05392   }
05393
05394   int i, j, k, ierr;
05395   Vnm_tstart(APBS_TIMER_SOLVER, "Solver timer");
05396   PBSAMInput pbsamIn = getPBSAMParams();
05397   PBAMInput pbamIn;// = getPBAMParams();
05398
05399   pbamIn.nmol_ = nosh->nmol;
05400
05401   // change any of the parameters you want...
05402   pbamIn.temp_ =  pbeparm->temp;
05403   if (fabs(pbamIn.temp_-0.0) < 1e-3)
05404   {
05405     printf("No temperature specified. Setting to 298.15K\n");
05406     pbamIn.temp_ = 298.15;
05407   }
05408
05409   // Dielectrics
05410   pbamIn.idiel_ = pbeparm->pdie;
05411   pbamIn.sdiel_ = pbeparm->sdie;
05412
05413   // Salt conc
05414   pbamIn.salt_ = parm->salt;
05415
05416   // Runtype: can be energyforce, electrostatics etc
05417   strncpy(pbamIn.runType_, parm->runtype, CHR_MAXLEN);
05418   strncpy(pbamIn.runName_, parm->runname, CHR_MAXLEN);
05419
05420   pbamIn.setunits_ = parm->setunits;
05421   if(parm->setunits == 1) strncpy(pbamIn.units_, parm->units, CHR_MAXLEN);
05422   pbamIn.randOrient_ = parm->setrandorient;
05423
05424   pbamIn.boxLen_ = parm->pbcboxlen;
05425   pbamIn.pbcType_ = parm->setpbcs;
05426
05427   // Electrostatic stuff
05428   if (parm->setgridpt) pbamIn.gridPts_ = parm->gridpt;
05429   strncpy(pbamIn.map3D_, parm->map3dname, CHR_MAXLEN);
05430   pbamIn.grid2Dct_ = parm->grid2Dct;
05431   for (i=0; i<pbamIn.grid2Dct_; i++)
05432   {
05433     strncpy(pbamIn.grid2D_[i], parm->grid2Dname[i], CHR_MAXLEN);
05434     strncpy(pbamIn.grid2Dax_[i], parm->grid2Dax[i], CHR_MAXLEN);
05435     pbamIn.grid2Dloc_[i] = parm->grid2Dloc[i];
05436   }
05437   strncpy(pbamIn.dxname_, parm->dxname, CHR_MAXLEN);
05438
05439   // Dynamics stuff
05440   pbamIn.ntraj_ = parm->ntraj;
05441   strncpy(pbamIn.termCombine_, parm->termcombine, CHR_MAXLEN);
05442
05443   pbamIn.termct_ = parm->termct;
05444   pbamIn.contct_ = parm->confilct;
05445
05446   if (strncmp(pbamIn.runType_, "dynamics", 8)== 0)
05447   {
05448     if (pbamIn.nmol_ > parm->diffct)
05449     {
05450       Vnm_tprint(2, "You need more diffusion information!\n");
05451       return 0;
05452     }
05453
05454     for (i=0; i<pbamIn.nmol_; i++)
05455     {
```

```
05456       if (parm->xyzct[i] <  parm->ntraj)
05457       {
05458         Vnm_tprint(2, "For molecule %d, you are missing trajectory!\n", i+1);
05459         return 0;
05460       } else {
05461         for (j=0; j<pbamIn.ntraj_; j++)
05462         {
05463           strncpy(pbamIn.xyzfil_[i][j], parm->xyzfil[i][j], CHR_MAXLEN);
05464         }
05465       }
05466     }
05467
05468     for (i=0; i<pbamIn.nmol_; i++)
05469     {
05470       strncpy(pbamIn.moveType_[i], parm->moveType[i], CHR_MAXLEN);
05471       pbamIn.transDiff_[i] = parm->transDiff[i];
05472       pbamIn.rotDiff_[i] = parm->rotDiff[i];
05473     }
05474
05475     for (i=0; i<pbamIn.termct_; i++)
05476     {
05477         strncpy(pbamIn.termnam_[i], parm->termnam[i], CHR_MAXLEN);
05478         pbamIn.termnu_[i][0] = parm->termnu[i][0];
05479         pbamIn.termval_[i] = parm->termVal[i];
05480     }
05481
05482     for (i=0; i<pbamIn.contct_; i++)
05483     {
05484         strncpy(pbamIn.confil_[i], parm->confil[i], CHR_MAXLEN);
05485     }
05486   }
05487
05488
05489   // SAM details
05490   pbsamIn.tolsp_  = samparm->tolsp;
05491   pbsamIn.imatct_ = samparm->imatct;
05492   pbsamIn.expct_  = samparm->expct;
05493   for (i=0; i<samparm->surfct; i++)
05494   {
05495       strncpy(pbsamIn.surffil_[i], samparm->surffil[i], CHR_MAXLEN);
05496   }
05497   for (i=0; i<samparm->imatct; i++)
05498   {
05499       strncpy(pbsamIn.imatfil_[i], samparm->imatfil[i], CHR_MAXLEN);
05500   }
05501   for (i=0; i<samparm->expct; i++)
05502   {
05503       strncpy(pbsamIn.expfil_[i], samparm->expfil[i], CHR_MAXLEN);
05504   }
05505
05506   // Running MSMS if the MSMS flag is used
05507   if (samparm->setmsms == 1) {
05508     for (i=0; i<pbamIn.nmol_; i++) {
05509     // find a clever way to use prefix of molecule name for msms outputs
05510     for (j=0; j < VMAX_ARGLEN; j++)
05511        if (nosh->molpath[i][j] == '\0') break;
05512
05513     // assume terminated by '.pqr' -> 4 char, want to term w/ '.xyzr'
05514  //char xyzr[j+2], surf[j+2], outname[j-4];
05515  char xyzr[VMAX_ARGLEN], surf[VMAX_ARGLEN], outname[VMAX_ARGLEN];
05516     for( k=0; k < j - 4; k++)
05517     {
05518         xyzr[k] = nosh->molpath[i][k];
05519         outname[k] = nosh->molpath[i][k];
05520         surf[k] = nosh->molpath[i][k];
05521     }
05522     outname[k] = '\0';
05523     xyzr[k]   = '.';  surf[k]   = '.';
05524     xyzr[k+1] = 'x';  surf[k+1] = 'v';
05525     xyzr[k+2] = 'y';  surf[k+2] = 'e';
05526     xyzr[k+3] = 'z';  surf[k+3] = 'r';
05527     xyzr[k+4] = 'r';  surf[k+4] = 't';
05528     xyzr[k+5] = '\0'; surf[k+5] = '\0';;
05529
05530     // write an XYZR file from xyzr data
05531     FILE *fp;
05532     fp=fopen(xyzr, "w");
05533     Vatom *atom;
05534     for(k=0; k< Valist_getNumberAtoms(molecules[i]); k++)
05535     {
05536        atom = Valist_getAtom(molecules[i],k);
```

```
05537         fprintf(fp, "%.4f  %.4f  %.4f  %.4f\n", Vatom_getPosition(atom)[0],
05538                                                 Vatom_getPosition(atom)[1],
05539                                                 Vatom_getPosition(atom)[2],
05540                                                 Vatom_getRadius(atom));
05541       }
05542       fclose(fp);
05543
05544       #ifdef _WIN32
05545          sprintf(fname_tp, "msms.exe -if %s -prob %f -dens %f -of %s",
05546                  xyzr, samparm->probe_radius,samparm->density, outname);
05547       #else
05548          sprintf(fname_tp, "msms -if %s -prob %f -dens %f -of %s",
05549                  xyzr, samparm->probe_radius,samparm->density, outname);
05550       #endif
05551
05552         printf("%s\n", fname_tp);
05553
05554         printf("Running MSMS...\n");
05555         ierr = system(fname_tp);
05556
05557         strncpy(pbsamIn.surffil_[i], surf, CHR_MAXLEN);
05558       }
05559     }
05560
05561
05562     // debug
05563     printPBSAMStruct( pbamIn, pbsamIn );
05564
05565     // Run the darn thing
05566     PBAMOutput pbamOut = runPBSAMWrapAPBS(pbamIn, pbsamIn, molecules, nosh->nmol);
05567
05568     Vnm_tprint(1, "\n\nReturning to APBS caller...\n\n");
05569
05570     if (!(strncmp(pbamIn.runType_, "dynamics", 8) &&
05571           strncmp(pbamIn.runType_, "energyforce", 11))) {
05572
05573         if (!strncmp(pbamIn.units_, "kcalmol", 7)) {  //scale to kjmol is 4.18400
05574
05575             Vnm_tprint(1, "Interaction energy in kCal/mol...\n\n");
05576
05577             for (int i = 0; i < PBAMPARM_MAXMOL; i++) {
05578                 Vnm_tprint(1, "  Molecule %d: Global net ELEC energy = %1.12E\n",
05579                            i+1, pbamOut.energies_[i]);
05580                 Vnm_tprint(1, "               Force = (%1.6E, %1.6E, %1.6E)\n\n",
05581                            pbamOut.forces_[i][0], pbamOut.forces_[i][1],
05582                            pbamOut.forces_[i][2]);
05583                 if (pbamOut.energies_[i+1] == 0.) break;
05584             }
05585
05586         } else if (!strncmp(pbamIn.units_, "jmol", 4)) {  //scale to kjmol is 0.001
05587
05588             Vnm_tprint(1, "Interaction energy in J/mol...\n\n");
05589
05590             for (int i = 0; i < PBAMPARM_MAXMOL; i++) {
05591                 Vnm_tprint(1, "  Molecule %d: Global net ELEC energy = %1.12E\n",
05592                            i+1, pbamOut.energies_[i]);
05593                 Vnm_tprint(1, "               Force = (%1.6E, %1.6E, %1.6E)\n\n",
05594                            pbamOut.forces_[i][0], pbamOut.forces_[i][1],
05595                            pbamOut.forces_[i][2]);
05596                 if (pbamOut.energies_[i+1] == 0.) break;
05597             }
05598
05599         } else { // if (!strncmp(pbamIn.units_, "kT", 2)) //scale to kjmol is 2.478 @ 298K
05600                                                           // or 0.008315436242 * 298
05601
05602             Vnm_tprint(1, "Interaction energy in kT @ %6.2f K...\n\n", pbamIn.temp_);
05603
05604             for (int i = 0; i < PBAMPARM_MAXMOL; i++) {
05605                 Vnm_tprint(1, "  Molecule %d: Global net ELEC energy = %1.12E\n",
05606                            i+1, pbamOut.energies_[i]);
05607                 Vnm_tprint(1, "               Force = (%1.6E, %1.6E, %1.6E)\n\n",
05608                            pbamOut.forces_[i][0], pbamOut.forces_[i][1],
05609                            pbamOut.forces_[i][2]);
05610                 if (pbamOut.energies_[i+1] == 0.) break;
05611             }
05612         }
05613     }
05614
05615     Vnm_tstop(APBS_TIMER_SOLVER, "Solver timer");
05616
05617     return 1;
```

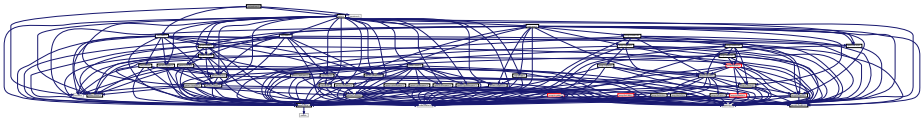
```
05618 }
05619
05620 #endif
```

9.113 src/routines.h File Reference

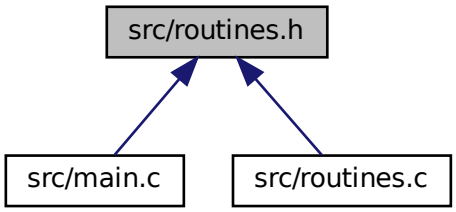
Header file for front end auxiliary routines.

```
#include "apbs.h"
#include "mc/mc.h"
#include "fem/vfetk.h"
#include "mcx/mcx.h"
```

Include dependency graph for routines.h:



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



Data Structures

- struct AtomForce

  Structure to hold atomic forces.

Macros

- #define APBSRC 13

  Return code for APBS during failure.

Typedefs

- typedef struct AtomForce AtomForce

  Define AtomForce type.

Functions

- VEXTERNC Vparam * loadParameter (NOsh *nosh)

  Loads and returns parameter object.
- VEXTERNC int loadMolecules (NOsh *nosh, Vparam *param, Valist *alist[NOSH_MAXMOL])

  Load the molecules given in NOsh into atom lists.
- VEXTERNC void killMolecules (NOsh *nosh, Valist *alist[NOSH_MAXMOL])

  Destroy the loaded molecules.
- VEXTERNC int loadDielMaps (NOsh *nosh, Vgrid *dielXMap[NOSH_MAXMOL], Vgrid *dielYMap[NOSH_MAXMOL], Vgrid *dielZMap[NOSH_MAXMOL])

  Load the dielectric maps given in NOsh into grid objects.
- VEXTERNC void killDielMaps (NOsh *nosh, Vgrid *dielXMap[NOSH_MAXMOL], Vgrid *dielYMap[NOSH_MAXMOL], Vgrid *dielZMap[NOSH_MAXMOL])

  Destroy the loaded dielectric.
- VEXTERNC int loadKappaMaps (NOsh *nosh, Vgrid *kappa[NOSH_MAXMOL])

  Load the kappa maps given in NOsh into grid objects.
- VEXTERNC void killKappaMaps (NOsh *nosh, Vgrid *kappa[NOSH_MAXMOL])

  Destroy the loaded kappa maps.
- VEXTERNC int loadPotMaps (NOsh *nosh, Vgrid *pot[NOSH_MAXMOL])

  Load the potential maps given in NOsh into grid objects.
- VEXTERNC void killPotMaps (NOsh *nosh, Vgrid *pot[NOSH_MAXMOL])

  Destroy the loaded potential maps.
- VEXTERNC int loadChargeMaps (NOsh *nosh, Vgrid *charge[NOSH_MAXMOL])

  Load the charge maps given in NOsh into grid objects.
- VEXTERNC void killChargeMaps (NOsh *nosh, Vgrid *charge[NOSH_MAXMOL])

  Destroy the loaded charge maps.
- VEXTERNC void printPBEPARM (PBEparm *pbeparm)

  Print out generic PBE params loaded from input.
- VEXTERNC void printMGPARM (MGparm *mgparm, double realCenter[3])

  Print out MG-specific params loaded from input.
- VEXTERNC int initMG (int icalc, NOsh *nosh, MGparm *mgparm, PBEparm *pbeparm, double realCenter[3], Vpbe *pbe[NOSH_MAXCALC], Valist *alist[NOSH_MAXMOL], Vgrid *dielXMap[NOSH_MAXMOL], Vgrid *dielYMap[NOSH_MAXMOL], Vgrid *dielZMap[NOSH_MAXMOL], Vgrid *kappaMap[NOSH_MAXMOL], Vgrid *chargeMap[NOSH_MAXMOL], Vpmgp *pmgp[NOSH_MAXCALC], Vpmg *pmg[NOSH_MAXCALC], Vgrid *potMap[NOSH_MAXMOL])

  Initialize an MG calculation.
- VEXTERNC void killMG (NOsh *nosh, Vpbe *pbe[NOSH_MAXCALC], Vpmgp *pmgp[NOSH_MAXCALC], Vpmg *pmg[NOSH_MAXCALC])

  Kill structures initialized during an MG calculation.
- VEXTERNC int solveMG (NOsh *nosh, Vpmg *pmg, MGparm_CalcType type)

  Solve the PBE with MG.
- VEXTERNC int setPartMG (NOsh *nosh, MGparm *mgparm, Vpmg *pmg)

  Set MG partitions for calculating observables and performing I/O.
- VEXTERNC int energyMG (NOsh *nosh, int icalc, Vpmg *pmg, int *nenergy, double *totEnergy, double *qf↩ Energy, double *qmEnergy, double *dielEnergy)

  Calculate electrostatic energies from MG solution.
- VEXTERNC void killEnergy ()

  Kill arrays allocated for energies.

- VEXTERNC int forceMG (Vmem *mem, NOsh *nosh, PBEparm *pbeparm, MGparm *mgparm, Vpmg *pmg, int *nforce, AtomForce **atomForce, Valist *alist[NOSH_MAXMOL])

  Calculate forces from MG solution.
- VEXTERNC void killForce (Vmem *mem, NOsh *nosh, int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC])

  Free memory from MG force calculation.
- VEXTERNC void storeAtomEnergy (Vpmg *pmg, int icalc, double **atomEnergy, int *nenergy)

  Store energy in arrays for future use.
- VEXTERNC int writedataFlat (NOsh *nosh, Vcom *com, const char *fname, double totEnergy[NOSH_MAXCALC], double qfEnergy[NOSH_MAXCALC], double qmEnergy[NOSH_MAXCALC], double dielEnergy[NOSH_MAXCALC], int nenergy[NOSH_MAXCALC], double *atomEnergy[NOSH_MAXCALC], int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC])

  Write out information to a flat file.
- VEXTERNC int writedataXML (NOsh *nosh, Vcom *com, const char *fname, double totEnergy[NOSH_MAXCALC], double qfEnergy[NOSH_MAXCALC], double qmEnergy[NOSH_MAXCALC], double dielEnergy[NOSH_MAXCALC], int nenergy[NOSH_MAXCALC], double *atomEnergy[NOSH_MAXCALC], int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC])

  Write out information to an XML file.
- VEXTERNC int writedataMG (int rank, NOsh *nosh, PBEparm *pbeparm, Vpmg *pmg)

  Write out observables from MG calculation to file.
- VEXTERNC int writematMG (int rank, NOsh *nosh, PBEparm *pbeparm, Vpmg *pmg)

  Write out operator matrix from MG calculation to file.
- VEXTERNC double returnEnergy (Vcom *com, NOsh *nosh, double totEnergy[NOSH_MAXCALC], int iprint)

  Access net local energy.
- VEXTERNC int printEnergy (Vcom *com, NOsh *nosh, double totEnergy[NOSH_MAXCALC], int iprint)

  Combine and pretty-print energy data (deprecated...see printElecEnergy)
- VEXTERNC int printElecEnergy (Vcom *com, NOsh *nosh, double totEnergy[NOSH_MAXCALC], int iprint)

  Combine and pretty-print energy data.
- VEXTERNC int printApolEnergy (NOsh *nosh, int iprint)

  Combine and pretty-print energy data.
- VEXTERNC int printForce (Vcom *com, NOsh *nosh, int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC], int i)

  Combine and pretty-print force data (deprecated...see printElecForce)
- VEXTERNC int printElecForce (Vcom *com, NOsh *nosh, int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC], int i)

  Combine and pretty-print force data.
- VEXTERNC int printApolForce (Vcom *com, NOsh *nosh, int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC], int i)

  Combine and pretty-print force data.
- VEXTERNC void startVio ()

  Wrapper to start MALOC Vio layer.
- VEXTERNC int energyAPOL (APOLparm *apolparm, double sasa, double sav, double atomsasa[ ], double atomwcaEnergy[ ], int numatoms)

  Calculate non-polar energies.
- VEXTERNC int forceAPOL (Vacc *acc, Vmem *mem, APOLparm *apolparm, int *nforce, AtomForce **atomForce, Valist *alist, Vclist *clist)

  Calculate non-polar forces.
- VEXTERNC int initAPOL (NOsh *nosh, Vmem *mem, Vparam *param, APOLparm *apolparm, int *nforce, AtomForce **atomForce, Valist *alist)

Upperlevel routine to the non-polar energy and force routines.

- VEXTERNC void printFEPARM (int icalc, NOsh *nosh, FEMparm *feparm, Vfetk *fetk[NOSH_MAXCALC])

  Print out FE-specific params loaded from input.
- VEXTERNC int energyFE (NOsh *nosh, int icalc, Vfetk *fetk[NOSH_MAXCALC], int *nenergy, double *totEnergy, double *qfEnergy, double *qmEnergy, double *dielEnergy)

  Calculate electrostatic energies from FE solution.
- VPUBLIC Vrc_Codes initFE (int icalc, NOsh *nosh, FEMparm *feparm, PBEparm *pbeparm, Vpbe *pbe[NOSH_MAXCALC], Valist *alist[NOSH_MAXMOL], Vfetk *fetk[NOSH_MAXCALC])

  Initialize FE solver objects.
- VEXTERNC void killFE (NOsh *nosh, Vpbe *pbe[NOSH_MAXCALC], Vfetk *fetk[NOSH_MAXCALC], Gem *gem[NOSH_MAXMOL])

  Kill structures initialized during an FE calculation.
- VEXTERNC int preRefineFE (int i, FEMparm *feparm, Vfetk *fetk[NOSH_MAXCALC])

  Pre-refine mesh before solve.
- VEXTERNC int partFE (int i, NOsh *nosh, FEMparm *feparm, Vfetk *fetk[NOSH_MAXCALC])

  Partition mesh (if applicable)
- VEXTERNC int solveFE (int i, PBEparm *pbeparm, FEMparm *feparm, Vfetk *fetk[NOSH_MAXCALC])

  Solve-estimate-refine.
- VEXTERNC int postRefineFE (int icalc, FEMparm *feparm, Vfetk *fetk[NOSH_MAXCALC])

  Estimate error, mark mesh, and refine mesh after solve.
- VEXTERNC int writedataFE (int rank, NOsh *nosh, PBEparm *pbeparm, Vfetk *fetk)

  Write FEM data to files.
- VEXTERNC Vrc_Codes loadMeshes (NOsh *nosh, Gem *gm[NOSH_MAXMOL])

  Load the meshes given in NOsh into geometry objects.
- VEXTERNC void killMeshes (NOsh *nosh, Gem *alist[NOSH_MAXMOL])

  Destroy the loaded meshes.

9.113.1 Detailed Description

Header file for front end auxiliary routines.

Author

Nathan Baker

Version

Id

Attention

```
*
* APBS -- Adaptive Poisson-Boltzmann Solver
*
*  Nathan A. Baker (nathan.baker@pnnl.gov)
*  Pacific Northwest National Laboratory
*
*  Additional contributing authors listed in the code documentation.
*
* Copyright (c) 2010-2020 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.
```

Definition in file routines.h.

9.114 routines.h

```
00001
00061 #ifndef _APBSROUTINES_H_
00062 #define _APBSROUTINES_H_
00063
00064 #include "apbs.h"
00065
00066 #ifdef HAVE_MC_H
00067 #    include "mc/mc.h"
00068 #    include "fem/vfetk.h"
00069 #endif
00070 #ifdef HAVE_MCX_H
00071 #    include "mcx/mcx.h"
00072 #endif
00073
00074 #ifdef ENABLE_BEM
00075   #include "TABIPBstruct.h"
00076 #endif
00077
00078 #ifdef ENABLE_GEOFLOW
00079   #include "GeometricFlowWrap.h"
00080 #endif
00081
00082 #if defined(ENABLE_PBAM) || defined(ENABLE_PBSAM)
00083   #include "PBAMWrap.h"
00084 #endif
00085
00086 #ifdef ENABLE_PBSAM
00087   #include "PBSAMWrap.h"
00088 #endif
00089
00093 #define APBSRC 13
00094
00099 struct AtomForce {
00100    double ibForce[3];
00101    double qfForce[3];
```

```
00102    double dbForce[3];
00103    double sasaForce[3];
00104    double savForce[3];
00105    double wcaForce[3];
00106 };
00107
00111 typedef struct AtomForce AtomForce;
00112
00118 VEXTERNC Vparam* loadParameter(
00119                                NOsh *nosh
00121                                );
00122
00128 VEXTERNC int loadMolecules(
00129                            NOsh *nosh,
00130                            Vparam *param,
00132                            Valist *alist[NOSH_MAXMOL]
00134                            );
00135
00142 VEXTERNC void killMolecules(NOsh *nosh, Valist *alist[NOSH_MAXMOL]);
00143
00153 VEXTERNC int loadDielMaps(NOsh *nosh,
00154                           Vgrid *dielXMap[NOSH_MAXMOL],
00155                           Vgrid *dielYMap[NOSH_MAXMOL],
00156                           Vgrid *dielZMap[NOSH_MAXMOL]
00157                          );
00158
00167 VEXTERNC void killDielMaps(NOsh *nosh, Vgrid *dielXMap[NOSH_MAXMOL],
00168   Vgrid *dielYMap[NOSH_MAXMOL], Vgrid *dielZMap[NOSH_MAXMOL]);
00169
00177 VEXTERNC int loadKappaMaps(NOsh *nosh, Vgrid *kappa[NOSH_MAXMOL]);
00178
00185 VEXTERNC void killKappaMaps(NOsh *nosh, Vgrid *kappa[NOSH_MAXMOL]);
00186
00194 VEXTERNC int loadPotMaps(NOsh *nosh, Vgrid *pot[NOSH_MAXMOL]);
00195
00202 VEXTERNC void killPotMaps(NOsh *nosh, Vgrid *pot[NOSH_MAXMOL]);
00203
00211 VEXTERNC int loadChargeMaps(NOsh *nosh, Vgrid *charge[NOSH_MAXMOL]);
00212
00219 VEXTERNC void killChargeMaps(NOsh *nosh, Vgrid *charge[NOSH_MAXMOL]);
00220
00226 VEXTERNC void printPBEPARM(PBEparm *pbeparm);
00227
00234 VEXTERNC void printMGPARM(MGparm *mgparm, double realCenter[3]);
00235
00241 VEXTERNC int initMG(
00242                     int icalc,
00243                     NOsh *nosh,
00244                     MGparm *mgparm,
00245                     PBEparm *pbeparm,
00246                     double realCenter[3],
00247                     Vpbe *pbe[NOSH_MAXCALC],
00248                     Valist *alist[NOSH_MAXMOL],
00249                     Vgrid *dielXMap[NOSH_MAXMOL],
00250                     Vgrid *dielYMap[NOSH_MAXMOL],
00251                     Vgrid *dielZMap[NOSH_MAXMOL],
00252                     Vgrid *kappaMap[NOSH_MAXMOL],
00253                     Vgrid *chargeMap[NOSH_MAXMOL],
00254                     Vpmgp *pmgp[NOSH_MAXCALC],
00255                     Vpmg *pmg[NOSH_MAXCALC],
00256                     Vgrid *potMap[NOSH_MAXMOL]
00257                     );
00258
00264 VEXTERNC void killMG(
00265                      NOsh *nosh,
00266                      Vpbe *pbe[NOSH_MAXCALC],
00267                      Vpmgp *pmgp[NOSH_MAXCALC],
00268                      Vpmg *pmg[NOSH_MAXCALC]
00269 );
00270
00279 VEXTERNC int solveMG(NOsh *nosh, Vpmg *pmg, MGparm_CalcType type);
00280
00289 VEXTERNC int setPartMG(NOsh *nosh, MGparm *mgparm, Vpmg *pmg);
00290
00304 VEXTERNC int energyMG(NOsh* nosh, int icalc, Vpmg *pmg,
00305   int *nenergy, double *totEnergy, double *qfEnergy, double *qmEnergy,
00306   double *dielEnergy);
00307
00312 VEXTERNC void killEnergy();
00313
00327 VEXTERNC int forceMG(Vmem *mem, NOsh *nosh, PBEparm *pbeparm,  MGparm *mgparm,
```

```
00328   Vpmg *pmg, int *nforce, AtomForce **atomForce, Valist *alist[NOSH_MAXMOL]);
00329
00338 VEXTERNC void killForce(Vmem *mem, NOsh *nosh, int nforce[NOSH_MAXCALC],
00339   AtomForce *atomForce[NOSH_MAXCALC]);
00340
00345 VEXTERNC void storeAtomEnergy(
00346         Vpmg *pmg,
00347         int icalc,
00348         double **atomEnergy,
00349         int *nenergy
00350         );
00351
00368 VEXTERNC int writedataFlat(NOsh *nosh, Vcom *com, const char *fname,
00369   double totEnergy[NOSH_MAXCALC], double qfEnergy[NOSH_MAXCALC],
00370   double qmEnergy[NOSH_MAXCALC], double dielEnergy[NOSH_MAXCALC],
00371   int nenergy[NOSH_MAXCALC], double *atomEnergy[NOSH_MAXCALC],
00372   int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC]);
00373
00390 VEXTERNC int writedataXML(NOsh *nosh, Vcom *com, const char *fname,
00391   double totEnergy[NOSH_MAXCALC], double qfEnergy[NOSH_MAXCALC],
00392   double qmEnergy[NOSH_MAXCALC], double dielEnergy[NOSH_MAXCALC],
00393   int nenergy[NOSH_MAXCALC], double *atomEnergy[NOSH_MAXCALC],
00394   int nforce[NOSH_MAXCALC], AtomForce *atomForce[NOSH_MAXCALC]);
00395
00405 VEXTERNC int writedataMG(int rank, NOsh *nosh, PBEparm *pbeparm, Vpmg *pmg);
00406
00416 VEXTERNC int writematMG(int rank, NOsh *nosh, PBEparm *pbeparm, Vpmg *pmg);
00417
00427 VEXTERNC double returnEnergy(Vcom *com, NOsh *nosh, double totEnergy[NOSH_MAXCALC], int iprint);
00428
00434 VEXTERNC int printEnergy(
00435                         Vcom *com,
00436                         NOsh *nosh,
00437                         double totEnergy[NOSH_MAXCALC],
00439                         int iprint
00440                         );
00441
00447 VEXTERNC int printElecEnergy(
00448                         Vcom *com,
00449                         NOsh *nosh,
00450                         double totEnergy[NOSH_MAXCALC],
00452                         int iprint
00453                         );
00454
00460 VEXTERNC int printApolEnergy(
00461                         NOsh *nosh,
00462                         int iprint
00463                         );
00464
00470 VEXTERNC int printForce(
00471                        Vcom *com,
00472                        NOsh *nosh,
00473                        int nforce[NOSH_MAXCALC],
00474                        AtomForce *atomForce[NOSH_MAXCALC],
00475                        int i
00476                        );
00477
00483 VEXTERNC int printElecForce(
00484                             Vcom *com,
00485                             NOsh *nosh,
00486                             int nforce[NOSH_MAXCALC],
00487                             AtomForce *atomForce[NOSH_MAXCALC],
00488                             int i
00489                             );
00490
00496 VEXTERNC int printApolForce(
00497                             Vcom *com,
00498                             NOsh *nosh,
00499                             int nforce[NOSH_MAXCALC],
00500                             AtomForce *atomForce[NOSH_MAXCALC],
00501                             int i
00502                             );
00503
00508 VEXTERNC void startVio();
00509
00515 VEXTERNC int energyAPOL(
00516                        APOLparm *apolparm,
00517                        double sasa,
00518                        double sav,
00519                        double atomsasa[],
00520                        double atomwcaEnergy[],
```

```
00521                         int numatoms
00522                         );
00523
00529 VEXTERNC int forceAPOL(
00530                         Vacc *acc,
00531                         Vmem *mem,
00532                         APOLparm *apolparm,
00534                         int *nforce,
00536                         AtomForce **atomForce,
00538                         Valist *alist,
00539                         Vclist *clist
00540                         );
00541
00547 VEXTERNC int initAPOL(
00548                        NOsh *nosh,
00549                        Vmem *mem,
00550                        Vparam *param,
00551                        APOLparm *apolparm,
00552                        int *nforce,
00553                        AtomForce **atomForce,
00554                        Valist *alist
00555                        );
00556
00557
00558 #ifdef HAVE_MC_H
00559 #include "fem/vfetk.h"
00560
00569 VEXTERNC void printFEPARM(int icalc, NOsh *nosh, FEMparm *feparm,
00570   Vfetk *fetk[NOSH_MAXCALC]);
00571
00586 VEXTERNC int energyFE(NOsh* nosh, int icalc, Vfetk *fetk[NOSH_MAXCALC],
00587   int *nenergy, double *totEnergy, double *qfEnergy, double *qmEnergy,
00588   double *dielEnergy);
00589
00597 VEXTERNC Vrc_Codes initFE(
00598                      int icalc,
00599                      NOsh *nosh,
00600                      FEMparm *feparm,
00601                      PBEparm *pbeparm,
00602                      Vpbe *pbe[NOSH_MAXCALC],
00603                      Valist *alist[NOSH_MAXMOL],
00604                      Vfetk *fetk[NOSH_MAXCALC]
00605 );
00606
00612 VEXTERNC void killFE(
00613                       NOsh *nosh,
00614                       Vpbe *pbe[NOSH_MAXCALC],
00615                       Vfetk *fetk[NOSH_MAXCALC],
00616                       Gem *gem[NOSH_MAXMOL]
00617 );
00618
00628 VEXTERNC int preRefineFE(int i,
00629                          FEMparm *feparm,
00630                          Vfetk *fetk[NOSH_MAXCALC]
00631 );
00632
00642 VEXTERNC int partFE(int i, NOsh *nosh, FEMparm *feparm,
00643   Vfetk *fetk[NOSH_MAXCALC]);
00644
00654 VEXTERNC int solveFE(int i,
00655                      PBEparm *pbeparm,
00656                      FEMparm *feparm,
00657                      Vfetk *fetk[NOSH_MAXCALC]
00658 );
00659
00671 VEXTERNC int postRefineFE(int icalc,
00672                           FEMparm *feparm,
00673                           Vfetk *fetk[NOSH_MAXCALC]
00674 );
00675
00685 VEXTERNC int writedataFE(int rank, NOsh *nosh, PBEparm *pbeparm, Vfetk *fetk);
00686
00692 VEXTERNC Vrc_Codes loadMeshes(
00693                             NOsh *nosh,
00694                             Gem *gm[NOSH_MAXMOL]
00696                            );
00697
00702 VEXTERNC void killMeshes(
00703                             NOsh *nosh,
00704                             Gem *alist[NOSH_MAXMOL]
00705                             );
```

```
00706 #endif
00707
00708 #endif
00709
00710
00711
00712 VEXTERNC void printMGPARM(MGparm *mgparm, double realCenter[3]);
00713
00714 #ifdef ENABLE_BEM
00720 VEXTERNC int initBEM(
00721                     int icalc,
00722                     NOsh *nosh,
00723                     BEMparm *bemparm,
00724                     PBEparm *pbeparm,
00725                     Vpbe *pbe[NOSH_MAXCALC]
00726                     );
00727
00733 VEXTERNC void killBEM(
00734                      NOsh *nosh,
00735                      Vpbe *pbe[NOSH_MAXCALC]
00736 );
00737
00746 VEXTERNC int solveBEM(Valist* molecules[NOSH_MAXMOL],NOsh *nosh, PBEparm *pbeparm, BEMparm *bemparm,
       BEMparm_CalcType type);
00747
00756 VEXTERNC int setPartBEM(NOsh *nosh, BEMparm *bemparm);
00757
00771 VEXTERNC int energyBEM(NOsh* nosh, int icalc,
00772   int *nenergy, double *totEnergy, double *qfEnergy, double *qmEnergy,
00773   double *dielEnergy);
00774
00788 VEXTERNC int forceBEM(NOsh *nosh, PBEparm *pbeparm,  BEMparm *bemparm,
00789   int *nforce, AtomForce **atomForce, Valist *alist[NOSH_MAXMOL]);
00790
00797 VEXTERNC void printBEMPARM(BEMparm *bemparm);
00798
00808 VEXTERNC int writedataBEM(int rank, NOsh *nosh, PBEparm *pbeparm);
00809
00819 VEXTERNC int writematBEM(int rank, NOsh *nosh, PBEparm *pbeparm);
00820 #endif
00821
00822 #ifdef ENABLE_GEOFLOW
00830 VEXTERNC int solveGeometricFlow(
00831       Valist* molecules[NOSH_MAXMOL],
00832                                 NOsh *nosh,
00833                                 PBEparm *pbeparm,
00834                                 APOLparm *apolparm,
00835                                 GEOFLOWparm *parm
00836 );
00837 #endif
00838
00839 #ifdef ENABLE_PBAM
00847 VEXTERNC int solvePBAM(
00848       Valist* molecules[NOSH_MAXMOL],
00849                                 NOsh *nosh,
00850                                 PBEparm *pbeparm,
00851                                 PBAMparm *parm
00852 );
00853 #endif
00854
00855 #ifdef ENABLE_PBSAM
00863 VEXTERNC int solvePBSAM(
00864       Valist* molecules[NOSH_MAXMOL],
00865                                 NOsh *nosh,
00866                                 PBEparm *pbeparm,
00867                                 PBAMparm *parm,
00868                                 PBSAMparm *samparm
00869 );
00870 #endif
```

Index

A
sVfetk_LocalVar, 456
a1cf
sVpmg, 475
a2cf
sVpmg, 475
a3cf
sVpmg, 475
acc
sVacc, 440
sVpbe, 468
ACD_ERROR
APOLparm class, 62
ACD_NO
APOLparm class, 62
ACD_YES
APOLparm class, 62
ACE_COMPS
APOLparm class, 62
ACE_NO
APOLparm class, 62
ACE_TOTAL
APOLparm class, 62
ACF_COMPS
APOLparm class, 62
ACF_NO
APOLparm class, 62
ACF_TOTAL
APOLparm class, 62
akeyPRE
sFEMparm, 404
akeySOLVE
sFEMparm, 404
alist
sNOsh, 419
sVacc, 441
sVclist, 447
sVcsm, 450
sVgreen, 460
sVpbe, 468
am
sVfetk, 452
apol
sNOsh, 419
apol2calc
sNOsh, 419
apolname
sNOsh, 420
apolparm
sNOsh_calc, 426
APOLparm class, 60
ACD_ERROR, 62
ACD_NO, 62
ACD_YES, 62
ACE_COMPS, 62
ACE_NO, 62
ACE_TOTAL, 62
ACF_COMPS, 62
ACF_NO, 62
ACF_TOTAL, 62
APOLparm_check, 62
APOLparm_copy, 63
APOLparm_ctor, 63
APOLparm_ctor2, 63
APOLparm_dtor, 64
APOLparm_dtor2, 64
eAPOLparm_calcEnergy, 62
eAPOLparm_calcForce, 62
eAPOLparm_doCalc, 62
APOLparm_check
APOLparm class, 62
APOLparm_copy
APOLparm class, 63
APOLparm_ctor
APOLparm class, 63
APOLparm_ctor2
APOLparm class, 63
APOLparm_dtor
APOLparm class, 64
APOLparm_dtor2
APOLparm class, 64
APOLparm_parseToken
MGparm class, 80
aprx
sVfetk, 452
area
sVaccSurf, 442
async
sMGparm, 410
atomData

Vparam_ResData, 493
atomFlags
sVacc, 441
AtomForce, 393
dbForce, 393
ibForce, 393
qfForce, 393
sasaForce, 394
savForce, 394
wcaForce, 394
atomName
sVatom, 445
sVparam_AtomData, 466
atoms
sValist, 444
sVclistCell, 449

B
sVfetk_LocalVar, 456
bcCalc
vpmg.c, 941
vpmg.h, 1096
bcfl
sPBEparm, 430
sVopot, 465
sVpmgp, 483
bcfl1
vpmg.c, 941
vpmg.h, 1096
BCFL_FOCUS
Vhal class, 186
BCFL_MAP
Vhal class, 186
BCFL_MDH
Vhal class, 186
BCFL_MEM
Vhal class, 186
BCFL_SDH
Vhal class, 186
BCFL_UNUSED
Vhal class, 186
BCFL_ZERO
Vhal class, 186
bcolcomp
Vpmg class, 243
bcolcomp2
Vpmg class, 244
bcolcomp3
Vpmg class, 245
bcolcomp4
Vpmg class, 245
bconc
sAPOLparm, 395
BCT_MANUAL
BEMparm class, 66
BCT_NONE
BEMparm class, 66
BEMparm
BEMparm class, 65
bemparm
sNOsh_calc, 426
BEMparm class, 64
BCT_MANUAL, 66
BCT_NONE, 66
BEMparm, 65
BEMparm_check, 66
BEMparm_ctor, 66
BEMparm_ctor2, 67
BEMparm_dtor, 67
BEMparm_dtor2, 67
BEMparm_parseToken, 68
eBEMparm_CalcType, 66
bemparm.c
BEMparm_copy, 579
BEMparm_check
BEMparm class, 66
BEMparm_copy
bemparm.c, 579
BEMparm_ctor
BEMparm class, 66
BEMparm_ctor2
BEMparm class, 67
BEMparm_dtor
BEMparm class, 67
BEMparm_dtor2
BEMparm class, 67
BEMparm_parseToken
BEMparm class, 68
Bmat_printHB
Vfetk class, 32
bogus
sNOsh, 420
bpts
sVaccSurf, 442
bspline2
vpmg.c, 942
vpmg.h, 1097
bspline4
vpmg.c, 943
vpmg.h, 1098
bulkIonicStrength
sVpbe, 468

C translation of Holst group PMG code, 270
HARMO2, 274
MAXIONS, 275
Vaxrand, 276
Vazeros, 277

VbuildA, 277
Vbuildband, 280
Vbuildband1_27, 282
Vbuildband1_7, 283
VbuildG, 284
VbuildG_1, 286
VbuildG_27, 289
VbuildG_7, 292
Vbuildgaler0, 295
Vbuildops, 296
VbuildP, 299
Vbuildstr, 301
Vc_vec, 302
Vc_vecpmg, 302
Vc_vecsmpbe, 303
Vcghs, 304
Vdc_vec, 306
Vdpbsl, 307
Vextrac, 309
VfboundPMG, 309
VfboundPMG00, 310
Vfmvfas, 311
Vfnewton, 314
Vgetjac, 317
Vgsrb, 318
VinterpPMG, 321
Vipower, 321
Vmatvec, 323
Vmgdriv, 326
Vmgdriv2, 328
Vmgsz, 330
Vmresid, 332
Vmvcs, 332
Vmvfas, 336
Vmypdefinitlpbe, 338
Vmypdefinitnpbe, 339
Vmypdefinitsmpbe, 339
Vnewdriv, 340
Vnewdriv2, 343
Vnewton, 345
Vnmatvec, 346
Vnmresid, 347
Vnsmooth, 348
Vpower, 349
Vprtmatd, 352
Vrestrc, 352
Vsmooth, 353
Vxaxpy, 356
Vxcopy, 356
Vxcopy_large, 358
Vxcopy_small, 359
Vxdot, 360
Vxnrm1, 360
Vxnrm2, 361
Vxscal, 361
calc
sNOsh, 420
calcenergy
sAPOLparm, 395
sPBEparm, 430
calcforce
sAPOLparm, 395
sPBEparm, 430
calctype
sNOsh_calc, 426
ccenter
sMGparm, 410
ccentmol
sMGparm, 410
ccf
sVpmg, 475
ccmeth
sMGparm, 410
cells
sVclist, 447
center
sMGparm, 410
sValist, 444
centmol
sMGparm, 411
cglen
sMGparm, 411
charge
sValist, 444
sVatom, 445
sVparam_AtomData, 466
sVpmg, 476
chargefmt
sNOsh, 420
chargeMap
sVpmg, 476
chargeMapID
sPBEparm, 430
chargeMeth
sVpmg, 476
chargepath
sNOsh, 420
chargeSrc
sVpmg, 476
chgm
sMGparm, 411
chgs
sBEMparm, 401
sMGparm, 411
clist
sVacc, 441
sVpbe, 468
CLIST_AUTO_DOMAIN

Vclist class, 168
CLIST_MANUAL_DOMAIN
Vclist class, 168
cmeth
sMGparm, 411
csm
sVfetk, 452
ctordata
sVgrid, 462

d2bspline4
vpmg.c, 943
vpmg.h, 1098
d2W
sVfetk_LocalVar, 456
d3bspline4
vpmg.c, 943
vpmg.h, 1098
data
sVgrid, 462
DB
sVfetk_LocalVar, 456
dbForce
AtomForce, 393
dbspline2
vpmg.c, 944
vpmg.h, 1099
dbspline4
vpmg.c, 944
vpmg.h, 1099
deblen
sVpbe, 469
delta
sVfetk_LocalVar, 456
dependencies, 19
DFu_wv
sVfetk_LocalVar, 456
dielfmt
sNOsh, 420
dielMapID
sPBEparm, 431
dielXMap
sVpmg, 476
dielXpath
sNOsh, 420
dielYMap
sVpmg, 476
dielYpath
sNOsh, 420
dielZMap
sVpmg, 476
dielZpath
sNOsh, 421
dime
sMGparm, 411
dpos
sAPOLparm, 395
dU
sVfetk_LocalVar, 456
dW
sVfetk_LocalVar, 456

eAPOLparm_calcEnergy
APOLparm class, 62
eAPOLparm_calcForce
APOLparm class, 62
eAPOLparm_doCalc
APOLparm class, 62
eBEMparm_CalcType
BEMparm class, 66
eFEMparm_CalcType
FEMparm class, 70
eFEMparm_EstType
FEMparm class, 70
eFEMparm_EtolType
FEMparm class, 70
eGEOFLOWparm_CalcType
GEOFLOWparm class, 74
ekey
sFEMparm, 405
elec
sNOsh, 421
elec2calc
sNOsh, 421
elecname
sNOsh, 421
eMGparm_CalcType
MGparm class, 79
eMGparm_CentMeth
MGparm class, 79
energyAPOL
High-level front-end routines, 365
energyFE
High-level front-end routines, 366
energyMG
High-level front-end routines, 367
eNOsh_CalcType
NOsh class, 91
eNOsh_MolFormat
NOsh class, 91
eNOsh_ParmFormat
NOsh class, 91
eNOsh_PrintType
NOsh class, 91
ePBAMparm_CalcType
PBAMparm class, 107
ePBEparm_calcEnergy
PBEparm class, 116

ePBEparm_calcForce
PBEparm class, 116
ePBSAMparm_CalcType
PBSAMparm class, 122
epsilon
sVatom, 446
sVparam_AtomData, 467
epsx
sVpmg, 476
epsy
sVpmg, 477
epsz
sVpmg, 477
errtol
sVpmgp, 483
etol
sFEMparm, 405
sGEOFLOWparm, 408
sMGparm, 411
eVbcfl
Vhal class, 186
eVchrg_Meth
Vhal class, 186
eVchrg_Src
Vhal class, 187
eVclist_DomainMode
Vclist class, 167
eVdata_Format
Vhal class, 187
eVdata_Type
Vhal class, 187
eVfetk_GuessType
Vfetk class, 30
eVfetk_LsolvType
Vfetk class, 30
eVfetk_MeshLoad
Vfetk class, 31
eVfetk_NsolvType
Vfetk class, 31
eVfetk_PrecType
Vfetk class, 31
eVhal_IPKEYType
Vhal class, 188
eVhal_PBEType
Vhal class, 188
eVoutput_Format
Vhal class, 189
eVrc_Codes
Vhal class, 189
eVsol_Meth
Vhal class, 189
eVsurf_Meth
Vhal class, 189
extDiEnergy
sVpmg, 477
extNpEnergy
sVpmg, 477
extQfEnergy
sVpmg, 477
extQmEnergy
sVpmg, 477

F
sVfetk_LocalVar, 457
fcenter
sMGparm, 412
fcentmol
sMGparm, 412
fcf
sVpmg, 477
fcmeth
sMGparm, 412
FCT_MANUAL
FEMparm class, 70
FCT_NONE
FEMparm class, 70
femparm
sNOsh_calc, 426
FEMparm class, 68
eFEMparm_CalcType, 70
eFEMparm_EstType, 70
eFEMparm_EtolType, 70
FCT_MANUAL, 70
FCT_NONE, 70
FEMparm_check, 71
FEMparm_copy, 71
FEMparm_ctor, 72
FEMparm_ctor2, 72
FEMparm_dtor, 72
FEMparm_dtor2, 73
FEMparm_EtolType, 69
FET_FRAC, 71
FET_GLOB, 71
FET_SIMP, 71
FRT_DUAL, 70
FRT_GEOM, 70
FRT_LOCA, 70
FRT_RESI, 70
FRT_UNIF, 70
FEMparm_check
FEMparm class, 71
FEMparm_copy
FEMparm class, 71
FEMparm_ctor
FEMparm class, 72
FEMparm_ctor2
FEMparm class, 72
FEMparm_dtor

FEMparm class, 72
FEMparm_dtor2
FEMparm class, 73
FEMparm_EtolType
FEMparm class, 69
FEMparm_parseToken
MGparm class, 80
feparm
sVfetk, 452
FET_FRAC
FEMparm class, 71
FET_GLOB
FEMparm class, 71
FET_SIMP
FEMparm class, 71
fetk
sVfetk_LocalVar, 457
fglen
sMGparm, 412
fillcoCharge
vpmg.c, 945
vpmg.h, 1099
fillcoChargeMap
vpmg.c, 945
vpmg.h, 1100
fillcoChargeSpline1
vpmg.c, 945
vpmg.h, 1100
fillcoChargeSpline2
vpmg.c, 945
vpmg.h, 1100
fillcoCoef
vpmg.c, 946
vpmg.h, 1100
fillcoCoefMap
vpmg.c, 946
vpmg.h, 1101
fillcoCoefMol
vpmg.c, 946
vpmg.h, 1101
fillcoCoefMolDiel
vpmg.c, 946
vpmg.h, 1101
fillcoCoefMolDielNoSmooth
vpmg.c, 946
vpmg.h, 1101
fillcoCoefMolDielSmooth
vpmg.c, 947
vpmg.h, 1101
fillcoCoefMolIon
vpmg.c, 947
vpmg.h, 1102
fillcoCoefSpline
vpmg.c, 947
vpmg.h, 1102
fillcoCoefSpline3
vpmg.c, 947
vpmg.h, 1102
fillcoCoefSpline4
vpmg.c, 948
vpmg.h, 1102
fillcoInducedDipole
vpmg.h, 1103
fillcoNLInducedDipole
vpmg.h, 1103
fillcoPermanentMultipole
vpmg.c, 948
vpmg.h, 1103
filled
sVpmg, 477
forceAPOL
High-level front-end routines, 367
forceMG
High-level front-end routines, 368
FRT_DUAL
FEMparm class, 70
FRT_GEOM
FEMparm class, 70
FRT_LOCA
FEMparm class, 70
FRT_RESI
FEMparm class, 70
FRT_UNIF
FEMparm class, 70
fType
sVfetk_LocalVar, 457
Fu_v
sVfetk_LocalVar, 457

gamma
sAPOLparm, 396
Gem_setExternalUpdateFunction
Vcsm class, 20
GEOFLOWparm
GEOFLOWparm class, 74
geoflowparm
sNOsh_calc, 426
GEOFLOWparm class, 73
eGEOFLOWparm_CalcType, 74
GEOFLOWparm, 74
GEOFLOWparm_check, 75
GEOFLOWparm_copy, 75
GEOFLOWparm_ctor, 75
GEOFLOWparm_ctor2, 76
GEOFLOWparm_dtor, 76
GEOFLOWparm_dtor2, 77
GEOFLOWparm_parseToken, 77
GFCT_AUTO, 75

GEOFLOWparm_check
GEOFLOWparm class, 75
GEOFLOWparm_copy
GEOFLOWparm class, 75
GEOFLOWparm_ctor
GEOFLOWparm class, 75
GEOFLOWparm_ctor2
GEOFLOWparm class, 76
GEOFLOWparm_dtor
GEOFLOWparm class, 76
GEOFLOWparm_dtor2
GEOFLOWparm class, 77
GEOFLOWparm_parseToken
GEOFLOWparm class, 77
GFCT_AUTO
GEOFLOWparm class, 75
glen
sFEMparm, 405
sMGparm, 412
gm
sVcsm, 450
sVfetk, 452
sVpee, 473
gotparm
sNOsh, 421
green
sVfetk_LocalVar, 457
grid
sAPOLparm, 396
sMGparm, 412
grids
sVmgrid, 464
gues
sVfetk, 453
gxcf
sVpmg, 478
gycf
sVpmg, 478
gzcf
sVpmg, 478

HARMO2
C translation of Holst group PMG code, 274
High-level front-end routines, 362
energyAPOL, 365
energyFE, 366
energyMG, 367
forceAPOL, 367
forceMG, 368
initAPOL, 368
initFE, 370
initMG, 371
killChargeMaps, 372
killDielMaps, 372
killEnergy, 373
killFE, 373
killForce, 373
killKappaMaps, 374
killMeshes, 374
killMG, 374
killMolecules, 375
killPotMaps, 375
loadChargeMaps, 375
loadDielMaps, 376
loadKappaMaps, 376
loadMeshes, 377
loadMolecules, 377
loadParameter, 378
loadPotMaps, 378
main, 378
partFE, 379
postRefineFE, 379
preRefineFE, 380
printApolEnergy, 381
printApolForce, 381
printElecEnergy, 382
printElecForce, 382
printEnergy, 383
printFEPARM, 383
printForce, 384
printMGPARM, 384
printPBEPARM, 385
returnEnergy, 385
setPartMG, 385
solveFE, 386
solveMG, 387
startVio, 387
storeAtomEnergy, 387
writedataFE, 388
writedataFlat, 388
writedataMG, 389
writedataXML, 390
writematMG, 391
hx
sVgrid, 462
sVpmgp, 483
hy
sVgrid, 462
sVpmgp, 483
hzed
sVgrid, 462
sVpmgp, 483

ibForce
AtomForce, 393
id
sVatom, 446
iinfo

sVpmgp, 483
initAPOL
High-level front-end routines, 368
initFE
High-level front-end routines, 370
initFlag
sVcsm, 450
initGreen
sVfetk_LocalVar, 457
initMG
High-level front-end routines, 371
ionc
sPBEparm, 431
ionConc
sVfetk_LocalVar, 457
sVpbe, 469
ionQ
sVfetk_LocalVar, 457
sVpbe, 469
ionq
sPBEparm, 431
ionr
sPBEparm, 431
ionRadii
sVfetk_LocalVar, 458
sVpbe, 469
ionstr
sVfetk_LocalVar, 458
iparm
sVpmg, 478
ipcon
sVpmgp, 483
iperf
sVpmgp, 484
ipkey
sVpbe, 469
sVpmgp, 484
IPKEY_LPBE
Vhal class, 188
IPKEY_NPBE
Vhal class, 188
IPKEY_SMPBE
Vhal class, 188
irite
sVpmgp, 484
ispara
sNOsh, 421
istop
sVpmgp, 484
itmax
sVpmgp, 485
ivdwAccExclus
vacc.c, 725
iwork
sVpmg, 478

jumpDiel
sVfetk_LocalVar, 458

kappa
sVpmg, 478
kappafmt
sNOsh, 421
kappaMap
sVpmg, 478
kappaMapID
sPBEparm, 431
kappapath
sNOsh, 422
key
sVpmgp, 485
killChargeMaps
High-level front-end routines, 372
killDielMaps
High-level front-end routines, 372
killEnergy
High-level front-end routines, 373
killFE
High-level front-end routines, 373
killFlag
sVpee, 473
killForce
High-level front-end routines, 373
killKappaMaps
High-level front-end routines, 374
killMeshes
High-level front-end routines, 374
killMG
High-level front-end routines, 374
killMolecules
High-level front-end routines, 375
killParam
sVpee, 473
killPotMaps
High-level front-end routines, 375

L
sVpbe, 469
level
sVfetk, 453
lgr_2DP1
vfetk.c, 514
lgr_2DP1x
vfetk.c, 514
lgr_2DP1y
vfetk.c, 515
lgr_2DP1z
vfetk.c, 515
lgr_3DP1

vfetk.c, 515
lgr_3DP1x
vfetk.c, 515
lgr_3DP1y
vfetk.c, 515
lgr_3DP1z
vfetk.c, 516
lkey
sVfetk, 453
lmax
sVfetk, 453
Lmem
sPBEparm, 431
loadChargeMaps
High-level front-end routines, 375
loadDielMaps
High-level front-end routines, 376
loadKappaMaps
High-level front-end routines, 376
loadMeshes
High-level front-end routines, 377
loadMolecules
High-level front-end routines, 377
loadParameter
High-level front-end routines, 378
loadPotMaps
High-level front-end routines, 378
localPartCenter
sVpee, 473
localPartID
sVpee, 473
localPartRadius
sVpee, 473
lower_corner
sVclist, 447
lprec
sVfetk, 453
ltol
sVfetk, 453

mac
sBEMparm, 401
main
High-level front-end routines, 378
markSphere
vpmg.c, 948
vpmg.h, 1103
MAT2
vmatrix.h, 833
MAT3
vmatrix.h, 833
Matrix wrapper class, 190
max_radius
sVclist, 447
maxcrd
sValist, 444
maxIonRadius
sVpbe, 469
MAXIONS
C translation of Holst group PMG code, 275
maxrad
sValist, 444
maxsolve
sFEMparm, 405
maxvert
sFEMparm, 405
MCM_FOCUS
MGparm class, 80
MCM_MOLECULE
MGparm class, 80
MCM_POINT
MGparm class, 80
MCT_AUTO
MGparm class, 79
MCT_DUMMY
MGparm class, 79
MCT_MANUAL
MGparm class, 79
MCT_NONE
MGparm class, 79
MCT_PARALLEL
MGparm class, 79
mdie
sPBEparm, 431
mem
sVacc, 441
sVaccSurf, 442
sVgrid, 463
sVpee, 474
membraneDiel
sVpbe, 469
memv
sPBEparm, 431
mesh
sBEMparm, 401
meshfmt
sNOsh, 422
meshID
sFEMparm, 405
meshpath
sNOsh, 422
meth
sVpmgp, 485
method
sMGparm, 412
mgcoar
sVpmgp, 485
mgdisc

sVpmgp, 486
mgkey
sVpmgp, 486
mgparm
sNOsh_calc, 426
MGparm class, 77
APOLparm_parseToken, 80
eMGparm_CalcType, 79
eMGparm_CentMeth, 79
FEMparm_parseToken, 80
MCM_FOCUS, 80
MCM_MOLECULE, 80
MCM_POINT, 80
MCT_AUTO, 79
MCT_DUMMY, 79
MCT_MANUAL, 79
MCT_NONE, 79
MCT_PARALLEL, 79
MGparm_check, 81
MGparm_copy, 81
MGparm_ctor, 81
MGparm_ctor2, 82
MGparm_dtor, 82
MGparm_dtor2, 82
MGparm_getCenterX, 83
MGparm_getCenterY, 83
MGparm_getCenterZ, 84
MGparm_getHx, 84
MGparm_getHy, 84
MGparm_getHz, 85
MGparm_getNx, 85
MGparm_getNy, 85
MGparm_getNz, 86
MGparm_parseToken, 86
MGparm_setCenterX, 87
MGparm_setCenterY, 87
MGparm_setCenterZ, 87
MGparm_check
MGparm class, 81
MGparm_copy
MGparm class, 81
MGparm_ctor
MGparm class, 81
MGparm_ctor2
MGparm class, 82
MGparm_dtor
MGparm class, 82
MGparm_dtor2
MGparm class, 82
MGparm_getCenterX
MGparm class, 83
MGparm_getCenterY
MGparm class, 83
MGparm_getCenterZ
MGparm class, 84
MGparm_getHx
MGparm class, 84
MGparm_getHy
MGparm class, 84
MGparm_getHz
MGparm class, 85
MGparm_getNx
MGparm class, 85
MGparm_getNy
MGparm class, 85
MGparm_getNz
MGparm class, 86
MGparm_parseToken
MGparm class, 86
MGparm_setCenterX
MGparm class, 87
MGparm_setCenterY
MGparm class, 87
MGparm_setCenterZ
MGparm class, 87
mgprol
sVpmgp, 486
mgrid
sVopot, 465
mgsmoo
sVpmgp, 486
mgsolv
sVpmgp, 487
mincrd
sValist, 444
mode
sVclist, 448
molfmt
sNOsh, 422
molid
sAPOLparm, 396
sPBEparm, 432
molpath
sNOsh, 422
msimp
sVcsm, 450
multipolebc
vpmg.c, 949
vpmg.h, 1104

n
sVclist, 448
n_ipc
sVpmgp, 487
n_iz
sVpmgp, 487
n_rpc
sVpmgp, 487

name
Vparam_ResData, 493
napol
sNOsh, 422
narr
sVpmgp, 487
narrc
sVpmgp, 487
natom
sVcsm, 450
nAtomData
Vparam_ResData, 493
natoms
sVclistCell, 449
nc
sVpmgp, 487
ncalc
sNOsh, 422
ncharge
sNOsh, 422
NCT_APOL
NOsh class, 91
NCT_BEM
NOsh class, 91
NCT_FEM
NOsh class, 91
NCT_GEOFLOW
NOsh class, 91
NCT_MG
NOsh class, 91
NCT_PBAM
NOsh class, 91
NCT_PBSAM
NOsh class, 91
ndiel
sNOsh, 423
nelec
sNOsh, 423
nf
sVpmgp, 488
ngrids
sVmgrid, 465
nion
sPBEparm, 432
sVfetk_LocalVar, 458
niwk
sVpmgp, 488
nkappa
sNOsh, 423
nkey
sVfetk, 453
nlev
sMGparm, 413
sVpmgp, 488
nmax
sVfetk, 454
nmesh
sNOsh, 423
NMF_PDB
NOsh class, 91
NMF_PQR
NOsh class, 91
NMF_XML
NOsh class, 91
nmol
sNOsh, 423
nonlin
sVpmgp, 488
nonlintype
sBEMparm, 401
sMGparm, 413
NOsh class, 88
eNOsh_CalcType, 91
eNOsh_MolFormat, 91
eNOsh_ParmFormat, 91
eNOsh_PrintType, 91
NCT_APOL, 91
NCT_BEM, 91
NCT_FEM, 91
NCT_GEOFLOW, 91
NCT_MG, 91
NCT_PBAM, 91
NCT_PBSAM, 91
NMF_PDB, 91
NMF_PQR, 91
NMF_XML, 91
NOsh_apol2calc, 92
NOsh_calc_copy, 92
NOsh_calc_ctor, 93
NOsh_calc_dtor, 93
NOsh_ctor, 93
NOsh_ctor2, 94
NOsh_dtor, 94
NOsh_dtor2, 94
NOsh_elec2calc, 95
NOsh_elecname, 95
NOsh_getCalc, 96
NOsh_getChargefmt, 96
NOsh_getChargepath, 96
NOsh_getDielfmt, 97
NOsh_getDielXpath, 97
NOsh_getDielYpath, 98
NOsh_getDielZpath, 98
NOsh_getKappafmt, 99
NOsh_getKappapath, 99
NOsh_getMolpath, 99
NOsh_getPotfmt, 100
NOsh_getPotpath, 100

NOsh_parseInput, 101
NOsh_parseInputFile, 101
NOsh_printCalc, 102
NOsh_printNarg, 102
NOsh_printOp, 103
NOsh_printWhat, 103
NOsh_setupApolCalc, 104
NOsh_setupElecCalc, 105
NPF_FLAT, 91
NPF_XML, 91
NPT_APOLENERGY, 92
NPT_APOLFORCE, 92
NPT_ELECENERGY, 92
NPT_ELECFORCE, 92
NPT_ENERGY, 92
NPT_FORCE, 92
NOsh_apol2calc
NOsh class, 92
NOsh_calc_copy
NOsh class, 92
NOsh_calc_ctor
NOsh class, 93
NOsh_calc_dtor
NOsh class, 93
NOsh_ctor
NOsh class, 93
NOsh_ctor2
NOsh class, 94
NOsh_dtor
NOsh class, 94
NOsh_dtor2
NOsh class, 94
NOsh_elec2calc
NOsh class, 95
NOsh_elecname
NOsh class, 95
NOsh_getCalc
NOsh class, 96
NOsh_getChargefmt
NOsh class, 96
NOsh_getChargepath
NOsh class, 96
NOsh_getDielfmt
NOsh class, 97
NOsh_getDielXpath
NOsh class, 97
NOsh_getDielYpath
NOsh class, 98
NOsh_getDielZpath
NOsh class, 98
NOsh_getKappafmt
NOsh class, 99
NOsh_getKappapath
NOsh class, 99
NOsh_getMolpath
NOsh class, 99
NOsh_getPotfmt
NOsh class, 100
NOsh_getPotpath
NOsh class, 100
NOsh_parseInput
NOsh class, 101
NOsh_parseInputFile
NOsh class, 101
NOsh_printCalc
NOsh class, 102
NOsh_printNarg
NOsh class, 102
NOsh_printOp
NOsh class, 103
NOsh_printWhat
NOsh class, 103
NOsh_setupApolCalc
NOsh class, 104
NOsh_setupElecCalc
NOsh class, 105
np
sVgreen, 460
NPF_FLAT
NOsh class, 91
NPF_XML
NOsh class, 91
npot
sNOsh, 423
nprint
sNOsh, 423
NPT_APOLENERGY
NOsh class, 92
NPT_APOLFORCE
NOsh class, 92
NPT_ELECENERGY
NOsh class, 92
NPT_ELECFORCE
NOsh class, 92
NPT_ENERGY
NOsh class, 92
NPT_FORCE
NOsh class, 92
npts
sVaccSurf, 442
sVclist, 448
nqsm
sVcsm, 450
nResData
Vparam, 492
nrwk
sVpmgp, 488
nsimp

sVcsm, 450
nsqm
sVcsm, 451
ntol
sVfetk, 454
nu1
sVpmgp, 488
nu2
sVpmgp, 488
number
sValist, 444
numIon
sVpbe, 470
numwrite
sPBEparm, 432
nvec
sVfetk_LocalVar, 458
nverts
sVfetk_LocalVar, 458
nx
sVgrid, 463
sVpmgp, 489
nxc
sVpmgp, 489
ny
sVgrid, 463
sVpmgp, 489
nyc
sVpmgp, 489
nz
sVgrid, 463
sVpmgp, 489
nzc
sVpmgp, 489

ofrac
sMGparm, 413
omegal
sVpmgp, 489
omegan
sVpmgp, 490
outdata
sBEMparm, 402
OUTPUT_FLAT
Vhal class, 189
OUTPUT_NULL
Vhal class, 189

param2Flag
sVpbe, 470
paramFlag
sVpbe, 470
parmfmt
sNOsh, 423
parmpath
sNOsh, 424
parsed
sAPOLparm, 396
sBEMparm, 402
sFEMparm, 405
sGEOFLOWparm, 408
sMGparm, 413
sNOsh, 424
sPBAMparm, 428
sPBEparm, 432
sPBSAMparm, 440
partDisjCenter
sMGparm, 413
partDisjLength
sMGparm, 413
partDisjOwnSide
sMGparm, 413
partFE
High-level front-end routines, 379
partID
sVatom, 446
PBAMCT_AUTO
PBAMparm class, 107
PBAMparm
PBAMparm class, 107
pbamparm
sNOsh_calc, 426
PBAMparm class, 105
ePBAMparm_CalcType, 107
PBAMCT_AUTO, 107
PBAMparm, 107
PBAMparm_check, 108
PBAMparm_copy, 108
PBAMparm_ctor, 108
PBAMparm_ctor2, 109
PBAMparm_dtor, 109
PBAMparm_dtor2, 109
PBAMparm_parse3Dmap, 110
PBAMparm_parseDiff, 110
PBAMparm_parseDX, 110
PBAMparm_parseGrid2D, 111
PBAMparm_parseGridPts, 111
PBAMparm_parsePBCS, 112
PBAMparm_parseRandorient, 112
PBAMparm_parseRunName, 112
PBAMparm_parseRunType, 113
PBAMparm_parseSalt, 113
PBAMparm_parseTermcombine, 113
PBAMparm_parseToken, 114
PBAMparm_parseUnits, 114
PBAMparm_parseXYZ, 114
PBAMparm_check
PBAMparm class, 108
PBAMparm_copy

PBAMparm class, 108
PBAMparm_ctor
PBAMparm class, 108
PBAMparm_ctor2
PBAMparm class, 109
PBAMparm_dtor
PBAMparm class, 109
PBAMparm_dtor2
PBAMparm class, 109
PBAMparm_parse3Dmap
PBAMparm class, 110
PBAMparm_parseDiff
PBAMparm class, 110
PBAMparm_parseDX
PBAMparm class, 110
PBAMparm_parseGrid2D
PBAMparm class, 111
PBAMparm_parseGridPts
PBAMparm class, 111
PBAMparm_parsePBCS
PBAMparm class, 112
PBAMparm_parseRandorient
PBAMparm class, 112
PBAMparm_parseRunName
PBAMparm class, 112
PBAMparm_parseRunType
PBAMparm class, 113
PBAMparm_parseSalt
PBAMparm class, 113
PBAMparm_parseTermcombine
PBAMparm class, 113
PBAMparm_parseToken
PBAMparm class, 114
PBAMparm_parseUnits
PBAMparm class, 114
PBAMparm_parseXYZ
PBAMparm class, 114
pbe
sVfetk, 454
sVopot, 465
sVpmg, 478
PBE_LPBE
Vhal class, 188
PBE_LRPBE
Vhal class, 188
PBE_NPBE
Vhal class, 188
PBE_SMPBE
Vhal class, 188
pbeparm
sNOsh_calc, 426
sVfetk, 454
PBEparm class, 115
ePBEparm_calcEnergy, 116
ePBEparm_calcForce, 116
PBEparm_check, 117
PBEparm_copy, 117
PBEparm_ctor, 117
PBEparm_ctor2, 118
PBEparm_dtor, 118
PBEparm_dtor2, 118
PBEparm_getIonCharge, 119
PBEparm_getIonConc, 119
PBEparm_getIonRadius, 120
PBEparm_parseToken, 120
PCE_COMPS, 116
PCE_NO, 116
PCE_TOTAL, 116
PCF_COMPS, 117
PCF_NO, 117
PCF_TOTAL, 117
PBEparm_check
PBEparm class, 117
PBEparm_copy
PBEparm class, 117
PBEparm_ctor
PBEparm class, 117
PBEparm_ctor2
PBEparm class, 118
PBEparm_dtor
PBEparm class, 118
PBEparm_dtor2
PBEparm class, 118
PBEparm_getIonCharge
PBEparm class, 119
PBEparm_getIonConc
PBEparm class, 119
PBEparm_getIonRadius
PBEparm class, 120
PBEparm_parseToken
PBEparm class, 120
pbetype
sPBEparm, 432
PBSAMCT_AUTO
PBSAMparm class, 122
PBSAMparm
PBSAMparm class, 122
pbsamparm
sNOsh_calc, 427
PBSAMparm class, 121
ePBSAMparm_CalcType, 122
PBSAMCT_AUTO, 122
PBSAMparm, 122
PBSAMparm_check, 122
PBSAMparm_copy, 123
PBSAMparm_ctor, 123
PBSAMparm_ctor2, 124
PBSAMparm_dtor, 124

PBSAMparm_dtor2, 124
PBSAMparm_parseExp, 125
PBSAMparm_parseImat, 125
PBSAMparm_parseMSMS, 125
PBSAMparm_parseSurf, 126
PBSAMparm_parseToken, 126
PBSAMparm_parseTolsp, 127
PBSAMparm_check
PBSAMparm class, 122
PBSAMparm_copy
PBSAMparm class, 123
PBSAMparm_ctor
PBSAMparm class, 123
PBSAMparm_ctor2
PBSAMparm class, 124
PBSAMparm_dtor
PBSAMparm class, 124
PBSAMparm_dtor2
PBSAMparm class, 124
PBSAMparm_parseExp
PBSAMparm class, 125
PBSAMparm_parseImat
PBSAMparm class, 125
PBSAMparm_parseMSMS
PBSAMparm class, 125
PBSAMparm_parseSurf
PBSAMparm class, 126
PBSAMparm_parseToken
PBSAMparm class, 126
PBSAMparm_parseTolsp
PBSAMparm class, 127
PCE_COMPS
PBEparm class, 116
PCE_NO
PBEparm class, 116
PCE_TOTAL
PBEparm class, 116
PCF_COMPS
PBEparm class, 117
PCF_NO
PBEparm class, 117
PCF_TOTAL
PBEparm class, 117
pcolcomp
Vpmg class, 246
pde
sVfetk, 454
pdie
sPBEparm, 432
pdime
sMGparm, 413
pjac
sVfetk, 454
pkey
sFEMparm, 405
pmgp
sVpmg, 479
position
sVatom, 446
postRefineFE
High-level front-end routines, 379
pot
sVpmg, 479
potfmt
sNOsh, 424
potMap
sVpmg, 479
potMapID
sPBEparm, 432
potpath
sNOsh, 424
preRefineFE
High-level front-end routines, 380
press
sAPOLparm, 396
PRINT_FUNC
vhal.h, 819
printApolEnergy
High-level front-end routines, 381
printApolForce
High-level front-end routines, 381
printcalc
sNOsh, 424
printElecEnergy
High-level front-end routines, 382
printElecForce
High-level front-end routines, 382
printEnergy
High-level front-end routines, 383
printFEPARM
High-level front-end routines, 383
printForce
High-level front-end routines, 384
printMGPARM
High-level front-end routines, 384
printnarg
sNOsh, 424
printop
sNOsh, 424
printPBEPARM
High-level front-end routines, 385
printwhat
sNOsh, 424
probe_radius
sVaccSurf, 443
proc_rank
sMGparm, 414
sNOsh, 425

proc_size
sMGparm, 414
sNOsh, 425
pvec
sVpmg, 479

qfForce
AtomForce, 393
qfForceSpline1
vpmg.c, 949
vpmg.h, 1105
qfForceSpline2
vpmg.c, 950
vpmg.h, 1105
qfForceSpline4
vpmg.c, 950
vpmg.h, 1105
qp
sVgreen, 461
qsm
sVcsm, 451

radius
sVatom, 446
sVparam_AtomData, 467
readdata
sVgrid, 463
readFlatFileLine
Vparam class, 192
readXMLFileAtom
Vparam class, 193
refSphere
sVacc, 441
resData
Vparam, 492
resName
sVatom, 446
sVparam_AtomData, 467
returnEnergy
High-level front-end routines, 385
rparm
sVpmg, 479
rwork
sVpmg, 479

sAPOLparm, 394
bconc, 395
calcenergy, 395
calcforce, 395
dpos, 395
gamma, 396
grid, 396
molid, 396
parsed, 396
press, 396
sasa, 396
sav, 396
sdens, 396
setbconc, 397
setcalcenergy, 397
setcalcforce, 397
setdpos, 397
setgamma, 397
setgrid, 398
setmolid, 398
setpress, 398
setsdens, 398
setsrad, 398
setsrfm, 398
setswin, 399
settemp, 399
setwat, 399
srad, 399
srfm, 399
swin, 399
temp, 400
totForce, 400
watepsilon, 400
watsigma, 400
wcaEnergy, 400
sasa
sAPOLparm, 396
sasaForce
AtomForce, 394
sav
sAPOLparm, 396
savForce
AtomForce, 394
sBEMparm, 400
chgs, 401
mac, 401
mesh, 401
nonlintype, 401
outdata, 402
parsed, 402
setmac, 402
setmesh, 402
setnonlintype, 402
setoutdata, 402
settree_n0, 403
settree_order, 403
tree_n0, 403
tree_order, 403
type, 403
sdens
sAPOLparm, 396
sPBEparm, 432
sdie
sPBEparm, 433

setakeyPRE
sFEMparm, 406
setakeySOLVE
sFEMparm, 406
setasync
sMGparm, 414
setbcfl
sPBEparm, 433
setbconc
sAPOLparm, 397
setcalcenergy
sAPOLparm, 397
sPBEparm, 433
setcalcforce
sAPOLparm, 397
sPBEparm, 433
setcgcent
sMGparm, 414
setcglen
sMGparm, 414
setchgm
sMGparm, 414
setdime
sMGparm, 415
setdpos
sAPOLparm, 397
setekey
sFEMparm, 406
setetol
sFEMparm, 406
sMGparm, 415
setfgcent
sMGparm, 415
setfglen
sMGparm, 415
setgamma
sAPOLparm, 397
setgcent
sMGparm, 415
setglen
sFEMparm, 406
sMGparm, 416
setgrid
sAPOLparm, 398
sMGparm, 416
setion
sPBEparm, 433
setLmem
sPBEparm, 433
setmac
sBEMparm, 402
setmaxsolve
sFEMparm, 406
setmaxvert
sFEMparm, 406
setmdie
sPBEparm, 434
setmemv
sPBEparm, 434
setmesh
sBEMparm, 402
setmethod
sMGparm, 416
setmolid
sAPOLparm, 398
sPBEparm, 434
setnion
sPBEparm, 434
setnlev
sMGparm, 416
setnonlintype
sBEMparm, 402
sMGparm, 416
setofrac
sMGparm, 417
setoutdata
sBEMparm, 402
setPartMG
High-level front-end routines, 385
setpbetype
sPBEparm, 434
setpdie
sPBEparm, 434
setpdime
sMGparm, 417
setpress
sAPOLparm, 398
setrank
sMGparm, 417
setsdens
sAPOLparm, 398
sPBEparm, 435
setsdie
sPBEparm, 435
setsize
sMGparm, 417
setsmsize
sPBEparm, 435
setsmvolume
sPBEparm, 435
setsrad
sAPOLparm, 398
sPBEparm, 435
setsrfm
sAPOLparm, 398
sPBEparm, 436
setswin
sAPOLparm, 399

sPBEparm, 436
settargetNum
sFEMparm, 406
settargetRes
sFEMparm, 407
settemp
sAPOLparm, 399
sPBEparm, 436
settree_n0
sBEMparm, 403
settree_order
sBEMparm, 403
settype
sFEMparm, 407
setUseAqua
sMGparm, 417
setwat
sAPOLparm, 399
setwritemat
sPBEparm, 436
setzmem
sPBEparm, 436
sFEMparm, 404
akeyPRE, 404
akeySOLVE, 404
ekey, 405
etol, 405
glen, 405
maxsolve, 405
maxvert, 405
meshID, 405
parsed, 405
pkey, 405
setakeyPRE, 406
setakeySOLVE, 406
setekey, 406
setetol, 406
setglen, 406
setmaxsolve, 406
setmaxvert, 406
settargetNum, 406
settargetRes, 407
settype, 407
targetNum, 407
targetRes, 407
type, 407
useMesh, 407
sGEOFLOWparm, 408
etol, 408
parsed, 408
type, 408
simp
sVfetk_LocalVar, 458
sMGparm, 409
async, 410
ccenter, 410
ccentmol, 410
ccmeth, 410
center, 410
centmol, 411
cglen, 411
chgm, 411
chgs, 411
cmeth, 411
dime, 411
etol, 411
fcenter, 412
fcentmol, 412
fcmeth, 412
fglen, 412
glen, 412
grid, 412
method, 412
nlev, 413
nonlintype, 413
ofrac, 413
parsed, 413
partDisjCenter, 413
partDisjLength, 413
partDisjOwnSide, 413
pdime, 413
proc_rank, 414
proc_size, 414
setasync, 414
setcgcent, 414
setcglen, 414
setchgm, 414
setdime, 415
setetol, 415
setfgcent, 415
setfglen, 415
setgcent, 415
setglen, 416
setgrid, 416
setmethod, 416
setnlev, 416
setnonlintype, 416
setofrac, 417
setpdime, 417
setrank, 417
setsize, 417
setUseAqua, 417
type, 418
useAqua, 418
smsize
sPBEparm, 437
sVpbe, 470
smvolume

sPBEparm, 437
sVpbe, 470
sNOsh, 418
alist, 419
apol, 419
apol2calc, 419
apolname, 420
bogus, 420
calc, 420
chargefmt, 420
chargepath, 420
dielfmt, 420
dielXpath, 420
dielYpath, 420
dielZpath, 421
elec, 421
elec2calc, 421
elecname, 421
gotparm, 421
ispara, 421
kappafmt, 421
kappapath, 422
meshfmt, 422
meshpath, 422
molfmt, 422
molpath, 422
napol, 422
ncalc, 422
ncharge, 422
ndiel, 423
nelec, 423
nkappa, 423
nmesh, 423
nmol, 423
npot, 423
nprint, 423
parmfmt, 423
parmpath, 424
parsed, 424
potfmt, 424
potpath, 424
printcalc, 424
printnarg, 424
printop, 424
printwhat, 424
proc_rank, 425
proc_size, 425
sNOsh_calc, 425
apolparm, 426
bemparm, 426
calctype, 426
femparm, 426
geoflowparm, 426
mgparm, 426
pbamparm, 426
pbeparm, 426
pbsamparm, 427
soluteCenter
sVpbe, 470
soluteCharge
sVpbe, 470
soluteDiel
sVpbe, 470
soluteRadius
sVpbe, 471
soluteXlen
sVpbe, 471
soluteYlen
sVpbe, 471
soluteZlen
sVpbe, 471
solveFE
High-level front-end routines, 386
solveMG
High-level front-end routines, 387
solventDiel
sVpbe, 471
solventRadius
sVpbe, 471
spacs
sVclist, 448
sPBAMparm, 427
parsed, 428
type, 428
sPBEparm, 428
bcfl, 430
calcenergy, 430
calcforce, 430
chargeMapID, 430
dielMapID, 431
ionc, 431
ionq, 431
ionr, 431
kappaMapID, 431
Lmem, 431
mdie, 431
memv, 431
molid, 432
nion, 432
numwrite, 432
parsed, 432
pbetype, 432
pdie, 432
potMapID, 432
sdens, 432
sdie, 433
setbcfl, 433
setcalcenergy, 433

setcalcforce, 433
setion, 433
setLmem, 433
setmdie, 434
setmemv, 434
setmolid, 434
setnion, 434
setpbetype, 434
setpdie, 434
setsdens, 435
setsdie, 435
setsmsize, 435
setsmvolume, 435
setsrad, 435
setsrfm, 436
setswin, 436
settemp, 436
setwritemat, 436
setzmem, 436
smsize, 437
smvolume, 437
srad, 437
srfm, 437
swin, 437
temp, 437
useChargeMap, 437
useDielMap, 437
useKappaMap, 438
usePotMap, 438
writefmt, 438
writemat, 438
writematflag, 438
writematstem, 438
writestem, 438
writetype, 439
zmem, 439
sPBSAMparm, 439
parsed, 440
type, 440
splineAcc
vacc.c, 726
splineWin
sVpmg, 479
sqm
sVcsm, 451
srad
sAPOLparm, 399
sPBEparm, 437
src/apbs.h, 495, 497
src/fem/vcsm.c, 498, 500
src/fem/vcsm.h, 506, 509
src/fem/vfetk.c, 510, 516
src/fem/vfetk.h, 546, 551
src/fem/vpee.c, 556, 558
src/fem/vpee.h, 565, 567
src/generic/apolparm.c, 568, 570
src/generic/bemparm.c, 577, 580
src/generic/femparm.c, 584, 586
src/generic/femparm.h, 591, 594
src/generic/geoflowparm.c, 595, 597
src/generic/geoflowparm.h, 600, 603
src/generic/mgparm.c, 603, 606
src/generic/mgparm.h, 618, 621
src/generic/nosh.c, 623, 626
src/generic/nosh.h, 666, 670
src/generic/pbamparm.c, 673, 676
src/generic/pbamparm.h, 684, 688
src/generic/pbeparm.c, 690, 692
src/generic/pbeparm.h, 708, 711
src/generic/pbsamparm.c, 713, 715
src/generic/pbsamparm.h, 718, 721
src/generic/vacc.c, 722, 727
src/generic/vacc.h, 750, 754
src/generic/valist.c, 757, 760
src/generic/valist.h, 771, 774
src/generic/vatom.c, 775, 778
src/generic/vatom.h, 781, 784
src/generic/vcap.c, 786, 788
src/generic/vcap.h, 788, 791
src/generic/vclist.c, 791, 793
src/generic/vclist.h, 799, 802
src/generic/vgreen.c, 803, 806
src/generic/vgreen.h, 812, 814
src/generic/vhal.h, 815, 824
src/generic/vmatrix.h, 832, 834
src/generic/vparam.c, 834, 837
src/generic/vparam.h, 845, 849
src/generic/vpbe.c, 850, 853
src/generic/vpbe.h, 859, 862
src/generic/vstring.c, 865, 867
src/generic/vstring.h, 869, 872
src/generic/vunit.h, 872, 874
src/main.c, 874, 876
src/mg/vgrid.c, 887, 890
src/mg/vgrid.h, 914, 918
src/mg/vmgrid.c, 919, 921
src/mg/vmgrid.h, 924, 927
src/mg/vopot.c, 927, 929
src/mg/vopot.h, 933, 936
src/mg/vpmg.c, 936, 953
src/mg/vpmg.h, 1089, 1109
src/mg/vpmgp.c, 1117, 1119
src/mg/vpmgp.h, 1123, 1127
src/routines.c, 1128, 1132
src/routines.h, 1200, 1204
srfm
sAPOLparm, 399
sPBEparm, 437

startVio
High-level front-end routines, 387
storeAtomEnergy
High-level front-end routines, 387
sType
sVfetk_LocalVar, 458
surf
sVacc, 441
surf_density
sVacc, 441
surfMeth
sVpmg, 479
sVacc, 440
acc, 440
alist, 441
atomFlags, 441
clist, 441
mem, 441
refSphere, 441
surf, 441
surf_density, 441
sVaccSurf, 442
area, 442
bpts, 442
mem, 442
npts, 442
probe_radius, 443
xpts, 443
ypts, 443
zpts, 443
sValist, 443
atoms, 444
center, 444
charge, 444
maxcrd, 444
maxrad, 444
mincrd, 444
number, 444
vmem, 445
sVatom, 445
atomName, 445
charge, 445
epsilon, 446
id, 446
partID, 446
position, 446
radius, 446
resName, 446
sVclist, 447
alist, 447
cells, 447
lower_corner, 447
max_radius, 447
mode, 448
n, 448
npts, 448
spacs, 448
upper_corner, 448
vmem, 448
sVclistCell, 448
atoms, 449
natoms, 449
sVcsm, 449
alist, 450
gm, 450
initFlag, 450
msimp, 450
natom, 450
nqsm, 450
nsimp, 450
nsqm, 451
qsm, 451
sqm, 451
vmem, 451
sVfetk, 451
am, 452
aprx, 452
csm, 452
feparm, 452
gm, 452
gues, 453
level, 453
lkey, 453
lmax, 453
lprec, 453
ltol, 453
nkey, 453
nmax, 454
ntol, 454
pbe, 454
pbeparm, 454
pde, 454
pjac, 454
type, 454
vmem, 454
sVfetk_LocalVar, 455
A, 456
B, 456
d2W, 456
DB, 456
delta, 456
DFu_wv, 456
dU, 456
dW, 456
F, 457
fetk, 457
fType, 457
Fu_v, 457

green, 457
initGreen, 457
ionConc, 457
ionQ, 457
ionRadii, 458
ionstr, 458
jumpDiel, 458
nion, 458
nvec, 458
nverts, 458
simp, 458
sType, 458
U, 459
u_D, 459
u_T, 459
verts, 459
vx, 459
W, 459
xq, 459
zkappa2, 459
zks2, 460
sVgreen, 460
alist, 460
np, 460
qp, 461
vmem, 461
xp, 461
yp, 461
zp, 461
sVgrid, 461
ctordata, 462
data, 462
hx, 462
hy, 462
hzed, 462
mem, 463
nx, 463
ny, 463
nz, 463
readdata, 463
xmax, 463
xmin, 463
ymax, 463
ymin, 464
zmax, 464
zmin, 464
sVmgrid, 464
grids, 464
ngrids, 465
sVopot, 465
bcfl, 465
mgrid, 465
pbe, 465
sVparam_AtomData, 466
atomName, 466
charge, 466
epsilon, 467
radius, 467
resName, 467
sVpbe, 467
acc, 468
alist, 468
bulkIonicStrength, 468
clist, 468
deblen, 469
ionConc, 469
ionQ, 469
ionRadii, 469
ipkey, 469
L, 469
maxIonRadius, 469
membraneDiel, 469
numIon, 470
param2Flag, 470
paramFlag, 470
smsize, 470
smvolume, 470
soluteCenter, 470
soluteCharge, 470
soluteDiel, 470
soluteRadius, 471
soluteXlen, 471
soluteYlen, 471
soluteZlen, 471
solventDiel, 471
solventRadius, 471
T, 471
V, 471
vmem, 472
xkappa, 472
z_mem, 472
zkappa2, 472
zmagic, 472
sVpee, 472
gm, 473
killFlag, 473
killParam, 473
localPartCenter, 473
localPartID, 473
localPartRadius, 473
mem, 474
sVpmg, 474
a1cf, 475
a2cf, 475
a3cf, 475
ccf, 475
charge, 476
chargeMap, 476

chargeMeth, 476
chargeSrc, 476
dielXMap, 476
dielYMap, 476
dielZMap, 476
epsx, 476
epsy, 477
epsz, 477
extDiEnergy, 477
extNpEnergy, 477
extQfEnergy, 477
extQmEnergy, 477
fcf, 477
filled, 477
gxcf, 478
gycf, 478
gzcf, 478
iparm, 478
iwork, 478
kappa, 478
kappaMap, 478
pbe, 478
pmgp, 479
pot, 479
potMap, 479
pvec, 479
rparm, 479
rwork, 479
splineWin, 479
surfMeth, 479
tcf, 480
u, 480
useChargeMap, 480
useDielXMap, 480
useDielYMap, 480
useDielZMap, 480
useKappaMap, 480
usePotMap, 480
vmem, 481
xf, 481
yf, 481
zf, 481
sVpmgp, 481
bcfl, 483
errtol, 483
hx, 483
hy, 483
hzed, 483
iinfo, 483
ipcon, 483
iperf, 484
ipkey, 484
irite, 484
istop, 484
itmax, 485
key, 485
meth, 485
mgcoar, 485
mgdisc, 486
mgkey, 486
mgprol, 486
mgsmoo, 486
mgsolv, 487
n_ipc, 487
n_iz, 487
n_rpc, 487
narr, 487
narrc, 487
nc, 487
nf, 488
niwk, 488
nlev, 488
nonlin, 488
nrwk, 488
nu1, 488
nu2, 488
nx, 489
nxc, 489
ny, 489
nyc, 489
nz, 489
nzc, 489
omegal, 489
omegan, 490
xcent, 490
xlen, 490
xmax, 490
xmin, 490
ycent, 490
ylen, 490
ymax, 490
ymin, 491
zcent, 491
zlen, 491
zmax, 491
zmin, 491
swin
sAPOLparm, 399
sPBEparm, 437

T
sVpbe, 471
targetNum
sFEMparm, 407
targetRes
sFEMparm, 407
tcf
sVpmg, 480

temp
sAPOLparm, 400
sPBEparm, 437
totForce
sAPOLparm, 400
tree_n0
sBEMparm, 403
tree_order
sBEMparm, 403
type
sBEMparm, 403
sFEMparm, 407
sGEOFLOWparm, 408
sMGparm, 418
sPBAMparm, 428
sPBSAMparm, 440
sVfetk, 454

U
sVfetk_LocalVar, 459
u
sVpmg, 480
u_D
sVfetk_LocalVar, 459
u_T
sVfetk_LocalVar, 459
upper_corner
sVclist, 448
useAqua
sMGparm, 418
useChargeMap
sPBEparm, 437
sVpmg, 480
useDielMap
sPBEparm, 437
useDielXMap
sVpmg, 480
useDielYMap
sVpmg, 480
useDielZMap
sVpmg, 480
useKappaMap
sPBEparm, 438
sVpmg, 480
useMesh
sFEMparm, 407
usePotMap
sPBEparm, 438
sVpmg, 480

V
sVpbe, 471
VABORT_MSG0
vhal.h, 819
VABORT_MSG1
vhal.h, 820
VABORT_MSG2
vhal.h, 820
Vacc class, 127
Vacc_atomdSASA, 129
Vacc_atomdSAV, 130
Vacc_atomSASA, 130
Vacc_atomSASPoints, 131
Vacc_atomSurf, 131
Vacc_ctor, 132
Vacc_ctor2, 132
Vacc_dtor, 133
Vacc_dtor2, 133
Vacc_fastMolAcc, 133
Vacc_ivdwAcc, 134
Vacc_memChk, 135
Vacc_molAcc, 135
Vacc_SASA, 136
Vacc_splineAcc, 136
Vacc_splineAccAtom, 137
Vacc_splineAccGrad, 137
Vacc_splineAccGradAtomNorm, 138
Vacc_splineAccGradAtomNorm3, 138
Vacc_splineAccGradAtomNorm4, 139
Vacc_splineAccGradAtomUnnorm, 139
Vacc_totalAtomdSASA, 140
Vacc_totalAtomdSAV, 140
Vacc_totalSASA, 141
Vacc_totalSAV, 141
Vacc_vdwAcc, 142
Vacc_wcaEnergy, 142
Vacc_wcaEnergyAtom, 143
Vacc_wcaForceAtom, 143
VaccSurf_ctor, 144
VaccSurf_ctor2, 144
VaccSurf_dtor, 145
VaccSurf_dtor2, 145
VaccSurf_refSphere, 146
vacc.c
ivdwAccExclus, 725
splineAcc, 726
Vacc_allocate, 726
Vacc_storeParms, 726
Vacc_allocate
vacc.c, 726
Vacc_atomdSASA
Vacc class, 129
Vacc_atomdSAV
Vacc class, 130
Vacc_atomSASA
Vacc class, 130
Vacc_atomSASPoints
Vacc class, 131
Vacc_atomSurf

Vacc class, 131
Vacc_ctor
Vacc class, 132
Vacc_ctor2
Vacc class, 132
Vacc_dtor
Vacc class, 133
Vacc_dtor2
Vacc class, 133
Vacc_fastMolAcc
Vacc class, 133
Vacc_ivdwAcc
Vacc class, 134
Vacc_memChk
Vacc class, 135
Vacc_molAcc
Vacc class, 135
Vacc_SASA
Vacc class, 136
Vacc_splineAcc
Vacc class, 136
Vacc_splineAccAtom
Vacc class, 137
Vacc_splineAccGrad
Vacc class, 137
Vacc_splineAccGradAtomNorm
Vacc class, 138
Vacc_splineAccGradAtomNorm3
Vacc class, 138
Vacc_splineAccGradAtomNorm4
Vacc class, 139
Vacc_splineAccGradAtomUnnorm
Vacc class, 139
Vacc_storeParms
vacc.c, 726
Vacc_totalAtomdSASA
Vacc class, 140
Vacc_totalAtomdSAV
Vacc class, 140
Vacc_totalSASA
Vacc class, 141
Vacc_totalSAV
Vacc class, 141
Vacc_vdwAcc
Vacc class, 142
Vacc_wcaEnergy
Vacc class, 142
Vacc_wcaEnergyAtom
Vacc class, 143
Vacc_wcaForceAtom
Vacc class, 143
VaccSurf_ctor
Vacc class, 144
VaccSurf_ctor2
Vacc class, 144
VaccSurf_dtor
Vacc class, 145
VaccSurf_dtor2
Vacc class, 145
VaccSurf_refSphere
Vacc class, 146
Valist class, 146
Valist_ctor, 147
Valist_ctor2, 148
Valist_dtor, 148
Valist_dtor2, 148
Valist_getAtom, 149
Valist_getAtomList, 149
Valist_getCenterX, 149
Valist_getCenterY, 150
Valist_getCenterZ, 150
Valist_getNumberAtoms, 151
Valist_getStatistics, 151
Valist_memChk, 151
Valist_readPDB, 152
Valist_readPQR, 152
Valist_readXML, 153
Valist_ctor
Valist class, 147
Valist_ctor2
Valist class, 148
Valist_dtor
Valist class, 148
Valist_dtor2
Valist class, 148
Valist_getAtom
Valist class, 149
Valist_getAtomList
Valist class, 149
Valist_getCenterX
Valist class, 149
Valist_getCenterY
Valist class, 150
Valist_getCenterZ
Valist class, 150
Valist_getNumberAtoms
Valist class, 151
Valist_getStatistics
Valist class, 151
Valist_memChk
Valist class, 151
Valist_readPDB
Valist class, 152
Valist_readPQR
Valist class, 152
Valist_readXML
Valist class, 153
VAPBS_BACK

Vhal class, 184
VAPBS_DOWN
Vhal class, 184
VAPBS_FRONT
Vhal class, 184
VAPBS_LEFT
Vhal class, 185
VAPBS_RIGHT
Vhal class, 185
VAPBS_UP
Vhal class, 185
VASSERT_MSG0
vhal.h, 820
VASSERT_MSG1
vhal.h, 820
VASSERT_MSG2
vhal.h, 821
VAT3
vmatrix.h, 834
Vatom class, 153
Vatom_copyFrom, 155
Vatom_copyTo, 156
Vatom_ctor, 156
Vatom_ctor2, 156
Vatom_dtor, 157
Vatom_dtor2, 157
Vatom_getAtomID, 157
Vatom_getAtomName, 158
Vatom_getCharge, 158
Vatom_getEpsilon, 158
Vatom_getPartID, 159
Vatom_getPosition, 159
Vatom_getRadius, 159
Vatom_getResName, 160
Vatom_memChk, 160
Vatom_setAtomID, 161
Vatom_setAtomName, 161
Vatom_setCharge, 161
Vatom_setEpsilon, 162
Vatom_setPartID, 162
Vatom_setPosition, 162
Vatom_setRadius, 163
Vatom_setResName, 163
VMAX_RECLEN, 155
Vatom_copyFrom
Vatom class, 155
Vatom_copyTo
Vatom class, 156
Vatom_ctor
Vatom class, 156
Vatom_ctor2
Vatom class, 156
Vatom_dtor
Vatom class, 157
Vatom_dtor2
Vatom class, 157
Vatom_getAtomID
Vatom class, 157
Vatom_getAtomName
Vatom class, 158
Vatom_getCharge
Vatom class, 158
Vatom_getEpsilon
Vatom class, 158
Vatom_getPartID
Vatom class, 159
Vatom_getPosition
Vatom class, 159
Vatom_getRadius
Vatom class, 159
Vatom_getResName
Vatom class, 160
Vatom_memChk
Vatom class, 160
Vatom_setAtomID
Vatom class, 161
Vatom_setAtomName
Vatom class, 161
Vatom_setCharge
Vatom class, 161
Vatom_setEpsilon
Vatom class, 162
Vatom_setPartID
Vatom class, 162
Vatom_setPosition
Vatom class, 162
Vatom_setRadius
Vatom class, 163
Vatom_setResName
Vatom class, 163
Vaxrand
C translation of Holst group PMG code, 276
Vazeros
C translation of Holst group PMG code, 277
VbuildA
C translation of Holst group PMG code, 277
Vbuildband
C translation of Holst group PMG code, 280
Vbuildband1_27
C translation of Holst group PMG code, 282
Vbuildband1_7
C translation of Holst group PMG code, 283
VbuildG
C translation of Holst group PMG code, 284
VbuildG_1
C translation of Holst group PMG code, 286
VbuildG_27
C translation of Holst group PMG code, 289

VbuildG_7
C translation of Holst group PMG code, 292
Vbuildgaler0
C translation of Holst group PMG code, 295
Vbuildops
C translation of Holst group PMG code, 296
VbuildP
C translation of Holst group PMG code, 299
Vbuildstr
C translation of Holst group PMG code, 301
Vc_vec
C translation of Holst group PMG code, 302
Vc_vecpmg
C translation of Holst group PMG code, 302
Vc_vecsmpbe
C translation of Holst group PMG code, 303
Vcap class, 163
Vcap_cosh, 164
Vcap_exp, 165
Vcap_sinh, 165
Vcap_cosh
Vcap class, 164
Vcap_exp
Vcap class, 165
Vcap_sinh
Vcap class, 165
Vcghs
C translation of Holst group PMG code, 304
VCHANNELEDMESSAGE0
vhal.h, 821
VCHANNELEDMESSAGE1
vhal.h, 821
VCHANNELEDMESSAGE2
vhal.h, 822
VCHANNELEDMESSAGE3
vhal.h, 822
Vclist class, 166
CLIST_AUTO_DOMAIN, 168
CLIST_MANUAL_DOMAIN, 168
eVclist_DomainMode, 167
Vclist_ctor, 168
Vclist_ctor2, 168
Vclist_dtor, 169
Vclist_dtor2, 169
Vclist_getCell, 170
Vclist_maxRadius, 170
Vclist_memChk, 170
VclistCell_ctor, 171
VclistCell_ctor2, 171
VclistCell_dtor, 172
VclistCell_dtor2, 172
Vclist_ctor
Vclist class, 168
Vclist_ctor2
Vclist class, 168
Vclist_dtor
Vclist class, 169
Vclist_dtor2
Vclist class, 169
Vclist_getCell
Vclist class, 170
Vclist_maxRadius
Vclist class, 170
Vclist_memChk
Vclist class, 170
VclistCell_ctor
Vclist class, 171
VclistCell_ctor2
Vclist class, 171
VclistCell_dtor
Vclist class, 172
VclistCell_dtor2
Vclist class, 172
VCM_BSPL2
Vhal class, 186
VCM_BSPL4
Vhal class, 186
VCM_CHARGE
Vhal class, 187
VCM_INDUCED
Vhal class, 187
VCM_NLINDUCED
Vhal class, 187
VCM_PERMANENT
Vhal class, 187
VCM_TRIL
Vhal class, 186
VCOPY
vhal.h, 822
Vcsm class, 19
Gem_setExternalUpdateFunction, 20
Vcsm_ctor, 20
Vcsm_ctor2, 21
Vcsm_dtor, 21
Vcsm_dtor2, 22
Vcsm_getAtom, 22
Vcsm_getAtomIndex, 23
Vcsm_getNumberAtoms, 23
Vcsm_getNumberSimplices, 23
Vcsm_getSimplex, 24
Vcsm_getSimplexIndex, 24
Vcsm_getValist, 25
Vcsm_init, 25
Vcsm_memChk, 26
Vcsm_update, 26
Vcsm_ctor
Vcsm class, 20
Vcsm_ctor2

Vcsm class, 21
Vcsm_dtor
Vcsm class, 21
Vcsm_dtor2
Vcsm class, 22
Vcsm_getAtom
Vcsm class, 22
Vcsm_getAtomIndex
Vcsm class, 23
Vcsm_getNumberAtoms
Vcsm class, 23
Vcsm_getNumberSimplices
Vcsm class, 23
Vcsm_getSimplex
Vcsm class, 24
Vcsm_getSimplexIndex
Vcsm class, 24
Vcsm_getValist
Vcsm class, 25
Vcsm_init
Vcsm class, 25
Vcsm_memChk
Vcsm class, 26
Vcsm_update
Vcsm class, 26
Vdc_vec
C translation of Holst group PMG code, 306
VDF_AVS
Vhal class, 187
VDF_DX
Vhal class, 187
VDF_DXBIN
Vhal class, 187
VDF_FLAT
Vhal class, 187
VDF_GZ
Vhal class, 187
VDF_MCSF
Vhal class, 187
VDF_UHBD
Vhal class, 187
Vdpbsl
C translation of Holst group PMG code, 307
VDT_ATOMPOT
Vhal class, 188
VDT_CHARGE
Vhal class, 188
VDT_DIELX
Vhal class, 188
VDT_DIELY
Vhal class, 188
VDT_DIELZ
Vhal class, 188
VDT_EDENS
Vhal class, 188
VDT_IVDW
Vhal class, 188
VDT_KAPPA
Vhal class, 188
VDT_LAP
Vhal class, 188
VDT_NDENS
Vhal class, 188
VDT_POT
Vhal class, 188
VDT_QDENS
Vhal class, 188
VDT_SMOL
Vhal class, 188
VDT_SSPL
Vhal class, 188
VDT_VDW
Vhal class, 188
VEMBED
Vhal class, 185
verts
sVfetk_LocalVar, 459
Vextrac
C translation of Holst group PMG code, 309
VfboundPMG
C translation of Holst group PMG code, 309
VfboundPMG00
C translation of Holst group PMG code, 310
VFCHI4
vpmg.c, 950
vpmg.h, 1106
Vfetk class, 26
Bmat_printHB, 32
eVfetk_GuessType, 30
eVfetk_LsolvType, 30
eVfetk_MeshLoad, 31
eVfetk_NsolvType, 31
eVfetk_PrecType, 31
Vfetk_ctor, 32
Vfetk_ctor2, 33
Vfetk_dqmEnergy, 33
Vfetk_dtor, 34
Vfetk_dtor2, 34
Vfetk_dumpLocalVar, 34
Vfetk_energy, 35
Vfetk_externalUpdateFunction, 35
Vfetk_fillArray, 36
Vfetk_genCube, 36
Vfetk_getAM, 37
Vfetk_getAtomColor, 37
Vfetk_getGem, 38
Vfetk_getSolution, 38
Vfetk_getVcsm, 39

Vfetk_getVpbe, 39
Vfetk_loadGem, 39
Vfetk_loadMesh, 40
Vfetk_memChk, 40
Vfetk_PDE_bisectEdge, 41
Vfetk_PDE_ctor, 41
Vfetk_PDE_ctor2, 42
Vfetk_PDE_delta, 42
Vfetk_PDE_DFu_wv, 43
Vfetk_PDE_dtor, 43
Vfetk_PDE_dtor2, 44
Vfetk_PDE_Fu, 44
Vfetk_PDE_Fu_v, 45
Vfetk_PDE_initAssemble, 45
Vfetk_PDE_initElement, 46
Vfetk_PDE_initFace, 46
Vfetk_PDE_initPoint, 47
Vfetk_PDE_Ju, 47
Vfetk_PDE_mapBoundary, 48
Vfetk_PDE_markSimplex, 48
Vfetk_PDE_oneChart, 49
Vfetk_PDE_simplexBasisForm, 50
Vfetk_PDE_simplexBasisInit, 50
Vfetk_PDE_u_D, 52
Vfetk_PDE_u_T, 52
Vfetk_qfEnergy, 53
Vfetk_readMesh, 54
Vfetk_setAtomColors, 54
Vfetk_setParameters, 55
Vfetk_write, 55
VGT_DIRI, 30
VGT_PREV, 30
VGT_ZERO, 30
VLT_BCG, 30
VLT_CG, 30
VLT_MG, 30
VLT_SLU, 30
VML_DIRICUBE, 31
VML_EXTERNAL, 31
VML_NEUMCUBE, 31
VNT_ARC, 31
VNT_INC, 31
VNT_NEW, 31
VPT_DIAG, 31
VPT_IDEN, 31
VPT_MG, 31
vfetk.c
lgr_2DP1, 514
lgr_2DP1x, 514
lgr_2DP1y, 515
lgr_2DP1z, 515
lgr_3DP1, 515
lgr_3DP1x, 515
lgr_3DP1y, 515
lgr_3DP1z, 516
Vfetk_ctor
Vfetk class, 32
Vfetk_ctor2
Vfetk class, 33
Vfetk_dqmEnergy
Vfetk class, 33
Vfetk_dtor
Vfetk class, 34
Vfetk_dtor2
Vfetk class, 34
Vfetk_dumpLocalVar
Vfetk class, 34
Vfetk_energy
Vfetk class, 35
Vfetk_externalUpdateFunction
Vfetk class, 35
Vfetk_fillArray
Vfetk class, 36
Vfetk_genCube
Vfetk class, 36
Vfetk_getAM
Vfetk class, 37
Vfetk_getAtomColor
Vfetk class, 37
Vfetk_getGem
Vfetk class, 38
Vfetk_getSolution
Vfetk class, 38
Vfetk_getVcsm
Vfetk class, 39
Vfetk_getVpbe
Vfetk class, 39
Vfetk_loadGem
Vfetk class, 39
Vfetk_loadMesh
Vfetk class, 40
Vfetk_memChk
Vfetk class, 40
Vfetk_PDE_bisectEdge
Vfetk class, 41
Vfetk_PDE_ctor
Vfetk class, 41
Vfetk_PDE_ctor2
Vfetk class, 42
Vfetk_PDE_delta
Vfetk class, 42
Vfetk_PDE_DFu_wv
Vfetk class, 43
Vfetk_PDE_dtor
Vfetk class, 43
Vfetk_PDE_dtor2
Vfetk class, 44
Vfetk_PDE_Fu

Vfetk class, 44
Vfetk_PDE_Fu_v
Vfetk class, 45
Vfetk_PDE_initAssemble
Vfetk class, 45
Vfetk_PDE_initElement
Vfetk class, 46
Vfetk_PDE_initFace
Vfetk class, 46
Vfetk_PDE_initPoint
Vfetk class, 47
Vfetk_PDE_Ju
Vfetk class, 47
Vfetk_PDE_mapBoundary
Vfetk class, 48
Vfetk_PDE_markSimplex
Vfetk class, 48
Vfetk_PDE_oneChart
Vfetk class, 49
Vfetk_PDE_simplexBasisForm
Vfetk class, 50
Vfetk_PDE_simplexBasisInit
Vfetk class, 50
Vfetk_PDE_u_D
Vfetk class, 52
Vfetk_PDE_u_T
Vfetk class, 52
Vfetk_qfEnergy
Vfetk class, 53
Vfetk_readMesh
Vfetk class, 54
Vfetk_setAtomColors
Vfetk class, 54
Vfetk_setParameters
Vfetk class, 55
Vfetk_write
Vfetk class, 55
VFILL
vhal.h, 822
VFLOOR
Vhal class, 185
Vfmvfas
C translation of Holst group PMG code, 311
Vfnewton
C translation of Holst group PMG code, 314
Vgetjac
C translation of Holst group PMG code, 317
Vgreen class, 172
Vgreen_coulomb, 174
Vgreen_coulomb_direct, 175
Vgreen_coulombD, 176
Vgreen_coulombD_direct, 176
Vgreen_ctor, 177
Vgreen_ctor2, 178
Vgreen_dtor, 178
Vgreen_dtor2, 178
Vgreen_getValist, 179
Vgreen_helmholtz, 179
Vgreen_helmholtzD, 180
Vgreen_memChk, 181
Vgreen_coulomb
Vgreen class, 174
Vgreen_coulomb_direct
Vgreen class, 175
Vgreen_coulombD
Vgreen class, 176
Vgreen_coulombD_direct
Vgreen class, 176
Vgreen_ctor
Vgreen class, 177
Vgreen_ctor2
Vgreen class, 178
Vgreen_dtor
Vgreen class, 178
Vgreen_dtor2
Vgreen class, 178
Vgreen_getValist
Vgreen class, 179
Vgreen_helmholtz
Vgreen class, 179
Vgreen_helmholtzD
Vgreen class, 180
Vgreen_memChk
Vgreen class, 181
Vgrid class, 219
Vgrid_ctor, 221
Vgrid_ctor2, 221
Vgrid_curvature, 222
Vgrid_dtor, 223
Vgrid_dtor2, 223
Vgrid_gradient, 223
Vgrid_integrate, 224
Vgrid_memChk, 224
Vgrid_normH1, 225
Vgrid_normL1, 225
Vgrid_normL2, 226
Vgrid_normLinf, 226
Vgrid_readDX, 226
Vgrid_readDXBIN, 227
Vgrid_readGZ, 228
Vgrid_seminormH1, 228
Vgrid_value, 229
Vgrid_writeDX, 229
Vgrid_writeDXBIN, 230
Vgrid_writeUHBD, 230
vgrid.c
Vgrid_writeGZ, 889
vgrid.h

Vgrid_writeGZ, 917
Vgrid_ctor
Vgrid class, 221
Vgrid_ctor2
Vgrid class, 221
Vgrid_curvature
Vgrid class, 222
Vgrid_dtor
Vgrid class, 223
Vgrid_dtor2
Vgrid class, 223
Vgrid_gradient
Vgrid class, 223
Vgrid_integrate
Vgrid class, 224
Vgrid_memChk
Vgrid class, 224
Vgrid_normH1
Vgrid class, 225
Vgrid_normL1
Vgrid class, 225
Vgrid_normL2
Vgrid class, 226
Vgrid_normLinf
Vgrid class, 226
Vgrid_readDX
Vgrid class, 226
Vgrid_readDXBIN
Vgrid class, 227
Vgrid_readGZ
Vgrid class, 228
Vgrid_seminormH1
Vgrid class, 228
Vgrid_value
Vgrid class, 229
Vgrid_writeDX
Vgrid class, 229
Vgrid_writeDXBIN
Vgrid class, 230
Vgrid_writeGZ
vgrid.c, 889
vgrid.h, 917
Vgrid_writeUHBD
Vgrid class, 230
Vgsrb
C translation of Holst group PMG code, 318
VGT_DIRI
Vfetk class, 30
VGT_PREV
Vfetk class, 30
VGT_ZERO
Vfetk class, 30
Vhal class, 181
BCFL_FOCUS, 186
BCFL_MAP, 186
BCFL_MDH, 186
BCFL_MEM, 186
BCFL_SDH, 186
BCFL_UNUSED, 186
BCFL_ZERO, 186
eVbcfl, 186
eVchrg_Meth, 186
eVchrg_Src, 187
eVdata_Format, 187
eVdata_Type, 187
eVhal_IPKEYType, 188
eVhal_PBEType, 188
eVoutput_Format, 189
eVrc_Codes, 189
eVsol_Meth, 189
eVsurf_Meth, 189
IPKEY_LPBE, 188
IPKEY_NPBE, 188
IPKEY_SMPBE, 188
OUTPUT_FLAT, 189
OUTPUT_NULL, 189
PBE_LPBE, 188
PBE_LRPBE, 188
PBE_NPBE, 188
PBE_SMPBE, 188
VAPBS_BACK, 184
VAPBS_DOWN, 184
VAPBS_FRONT, 184
VAPBS_LEFT, 185
VAPBS_RIGHT, 185
VAPBS_UP, 185
VCM_BSPL2, 186
VCM_BSPL4, 186
VCM_CHARGE, 187
VCM_INDUCED, 187
VCM_NLINDUCED, 187
VCM_PERMANENT, 187
VCM_TRIL, 186
VDF_AVS, 187
VDF_DX, 187
VDF_DXBIN, 187
VDF_FLAT, 187
VDF_GZ, 187
VDF_MCSF, 187
VDF_UHBD, 187
VDT_ATOMPOT, 188
VDT_CHARGE, 188
VDT_DIELX, 188
VDT_DIELY, 188
VDT_DIELZ, 188
VDT_EDENS, 188
VDT_IVDW, 188
VDT_KAPPA, 188

VDT_LAP, 188
VDT_NDENS, 188
VDT_POT, 188
VDT_QDENS, 188
VDT_SMOL, 188
VDT_SSPL, 188
VDT_VDW, 188
VEMBED, 185
VFLOOR, 185
VRC_FAILURE, 189
VRC_SUCCESS, 189
VSM_MOL, 190
VSM_MOLSMOOTH, 190
VSM_SPLINE, 190
VSM_SPLINE3, 190
VSM_SPLINE4, 190
vhal.h
PRINT_FUNC, 819
VABORT_MSG0, 819
VABORT_MSG1, 820
VABORT_MSG2, 820
VASSERT_MSG0, 820
VASSERT_MSG1, 820
VASSERT_MSG2, 821
VCHANNELEDMESSAGE0, 821
VCHANNELEDMESSAGE1, 821
VCHANNELEDMESSAGE2, 822
VCHANNELEDMESSAGE3, 822
VCOPY, 822
VFILL, 822
VWARN_MSG0, 823
VWARN_MSG1, 823
VWARN_MSG2, 823
VinterpPMG
C translation of Holst group PMG code, 321
Vipower
C translation of Holst group PMG code, 321
VLT_BCG
Vfetk class, 30
VLT_CG
Vfetk class, 30
VLT_MG
Vfetk class, 30
VLT_SLU
Vfetk class, 30
vmatrix.h
MAT2, 833
MAT3, 833
VAT3, 834
Vmatvec
C translation of Holst group PMG code, 323
VMAX_RECLEN
Vatom class, 155
vmem
sValist, 445
sVclist, 448
sVcsm, 451
sVfetk, 454
sVgreen, 461
sVpbe, 472
sVpmg, 481
Vparam, 492
Vparam_ResData, 493
Vmgdriv
C translation of Holst group PMG code, 326
Vmgdriv2
C translation of Holst group PMG code, 328
Vmgrid class, 231
Vmgrid_addGrid, 232
Vmgrid_ctor, 232
Vmgrid_ctor2, 233
Vmgrid_curvature, 233
Vmgrid_dtor, 234
Vmgrid_dtor2, 234
Vmgrid_getGridByNum, 234
Vmgrid_getGridByPoint, 235
Vmgrid_gradient, 235
Vmgrid_value, 235
Vmgrid_addGrid
Vmgrid class, 232
Vmgrid_ctor
Vmgrid class, 232
Vmgrid_ctor2
Vmgrid class, 233
Vmgrid_curvature
Vmgrid class, 233
Vmgrid_dtor
Vmgrid class, 234
Vmgrid_dtor2
Vmgrid class, 234
Vmgrid_getGridByNum
Vmgrid class, 234
Vmgrid_getGridByPoint
Vmgrid class, 235
Vmgrid_gradient
Vmgrid class, 235
Vmgrid_value
Vmgrid class, 235
Vmgsz
C translation of Holst group PMG code, 330
VML_DIRICUBE
Vfetk class, 31
VML_EXTERNAL
Vfetk class, 31
VML_NEUMCUBE
Vfetk class, 31
Vmresid
C translation of Holst group PMG code, 332

Vmvcs
C translation of Holst group PMG code, 332
Vmvfas
C translation of Holst group PMG code, 336
Vmypdefinitlpbe
C translation of Holst group PMG code, 338
Vmypdefinitnpbe
C translation of Holst group PMG code, 339
Vmypdefinitsmpbe
C translation of Holst group PMG code, 339
Vnewdriv
C translation of Holst group PMG code, 340
Vnewdriv2
C translation of Holst group PMG code, 343
Vnewton
C translation of Holst group PMG code, 345
Vnmatvec
C translation of Holst group PMG code, 346
Vnmresid
C translation of Holst group PMG code, 347
Vnsmooth
C translation of Holst group PMG code, 348
VNT_ARC
Vfetk class, 31
VNT_INC
Vfetk class, 31
VNT_NEW
Vfetk class, 31
Vopot class, 236
Vopot_ctor, 237
Vopot_ctor2, 237
Vopot_curvature, 238
Vopot_dtor, 238
Vopot_dtor2, 239
Vopot_gradient, 239
Vopot_pot, 240
Vopot_ctor
Vopot class, 237
Vopot_ctor2
Vopot class, 237
Vopot_curvature
Vopot class, 238
Vopot_dtor
Vopot class, 238
Vopot_dtor2
Vopot class, 239
Vopot_gradient
Vopot class, 239
Vopot_pot
Vopot class, 240
Vpackmg
Vpmg class, 247
Vparam, 491
nResData, 492
resData, 492
vmem, 492
Vparam class, 190
readFlatFileLine, 192
readXMLFileAtom, 193
Vparam_AtomData_copyFrom, 193
Vparam_AtomData_copyTo, 193
Vparam_AtomData_ctor, 194
Vparam_AtomData_ctor2, 194
Vparam_AtomData_dtor, 194
Vparam_AtomData_dtor2, 195
Vparam_ctor, 195
Vparam_ctor2, 195
Vparam_dtor, 196
Vparam_dtor2, 196
Vparam_getAtomData, 196
Vparam_getResData, 197
Vparam_memChk, 197
Vparam_readFlatFile, 198
Vparam_readXMLFile, 199
Vparam_ResData_copyTo, 199
Vparam_ResData_ctor, 200
Vparam_ResData_ctor2, 200
Vparam_ResData_dtor, 200
Vparam_ResData_dtor2, 201
Vparam_AtomData_copyFrom
Vparam class, 193
Vparam_AtomData_copyTo
Vparam class, 193
Vparam_AtomData_ctor
Vparam class, 194
Vparam_AtomData_ctor2
Vparam class, 194
Vparam_AtomData_dtor
Vparam class, 194
Vparam_AtomData_dtor2
Vparam class, 195
Vparam_ctor
Vparam class, 195
Vparam_ctor2
Vparam class, 195
Vparam_dtor
Vparam class, 196
Vparam_dtor2
Vparam class, 196
Vparam_getAtomData
Vparam class, 196
Vparam_getResData
Vparam class, 197
Vparam_memChk
Vparam class, 197
Vparam_readFlatFile
Vparam class, 198
Vparam_readXMLFile

Vparam class, 199
Vparam_ResData, 492
atomData, 493
name, 493
nAtomData, 493
vmem, 493
Vparam_ResData_copyTo
Vparam class, 199
Vparam_ResData_ctor
Vparam class, 200
Vparam_ResData_ctor2
Vparam class, 200
Vparam_ResData_dtor
Vparam class, 200
Vparam_ResData_dtor2
Vparam class, 201
Vpbe class, 201
Vpbe_ctor, 203
Vpbe_ctor2, 204
Vpbe_dtor, 206
Vpbe_dtor2, 206
Vpbe_getBulkIonicStrength, 206
Vpbe_getCoulombEnergy1, 207
Vpbe_getDeblen, 207
Vpbe_getGamma, 207
Vpbe_getIons, 208
Vpbe_getLmem, 208
Vpbe_getMaxIonRadius, 209
Vpbe_getmembraneDiel, 209
Vpbe_getmemv, 209
Vpbe_getSoluteCenter, 210
Vpbe_getSoluteCharge, 210
Vpbe_getSoluteDiel, 210
Vpbe_getSoluteRadius, 211
Vpbe_getSoluteXlen, 211
Vpbe_getSoluteYlen, 212
Vpbe_getSoluteZlen, 212
Vpbe_getSolventDiel, 212
Vpbe_getSolventRadius, 213
Vpbe_getTemperature, 213
Vpbe_getVacc, 213
Vpbe_getValist, 214
Vpbe_getXkappa, 214
Vpbe_getZkappa2, 215
Vpbe_getZmagic, 215
Vpbe_getzmem, 215
Vpbe_memChk, 216
Vpbe_ctor
Vpbe class, 203
Vpbe_ctor2
Vpbe class, 204
Vpbe_dtor
Vpbe class, 206
Vpbe_dtor2
Vpbe class, 206
Vpbe_getBulkIonicStrength
Vpbe class, 206
Vpbe_getCoulombEnergy1
Vpbe class, 207
Vpbe_getDeblen
Vpbe class, 207
Vpbe_getGamma
Vpbe class, 207
Vpbe_getIons
Vpbe class, 208
Vpbe_getLmem
Vpbe class, 208
Vpbe_getMaxIonRadius
Vpbe class, 209
Vpbe_getmembraneDiel
Vpbe class, 209
Vpbe_getmemv
Vpbe class, 209
Vpbe_getSoluteCenter
Vpbe class, 210
Vpbe_getSoluteCharge
Vpbe class, 210
Vpbe_getSoluteDiel
Vpbe class, 210
Vpbe_getSoluteRadius
Vpbe class, 211
Vpbe_getSoluteXlen
Vpbe class, 211
Vpbe_getSoluteYlen
Vpbe class, 212
Vpbe_getSoluteZlen
Vpbe class, 212
Vpbe_getSolventDiel
Vpbe class, 212
Vpbe_getSolventRadius
Vpbe class, 213
Vpbe_getTemperature
Vpbe class, 213
Vpbe_getVacc
Vpbe class, 213
Vpbe_getValist
Vpbe class, 214
Vpbe_getXkappa
Vpbe class, 214
Vpbe_getZkappa2
Vpbe class, 215
Vpbe_getZmagic
Vpbe class, 215
Vpbe_getzmem
Vpbe class, 215
Vpbe_memChk
Vpbe class, 216
Vpee class, 56

Vpee_ctor, 57
Vpee_ctor2, 58
Vpee_dtor, 58
Vpee_dtor2, 59
Vpee_markRefine, 59
Vpee_numSS, 60
Vpee_ctor
Vpee class, 57
Vpee_ctor2
Vpee class, 58
Vpee_dtor
Vpee class, 58
Vpee_dtor2
Vpee class, 59
Vpee_markRefine
Vpee class, 59
Vpee_numSS
Vpee class, 60
Vpmg class, 240
bcolcomp, 243
bcolcomp2, 244
bcolcomp3, 245
bcolcomp4, 245
pcolcomp, 246
Vpackmg, 247
Vpmg_ctor, 248
Vpmg_ctor2, 248
Vpmg_dbDirectPolForce, 249
Vpmg_dbForce, 249
Vpmg_dbMutualPolForce, 250
Vpmg_dbNLDirectPolForce, 251
Vpmg_dbPermanentMultipoleForce, 251
Vpmg_dielEnergy, 251
Vpmg_dielGradNorm, 252
Vpmg_dtor, 253
Vpmg_dtor2, 253
Vpmg_energy, 253
Vpmg_fieldSpline4, 254
Vpmg_fillArray, 254
Vpmg_fillco, 255
Vpmg_force, 256
Vpmg_ibDirectPolForce, 257
Vpmg_ibForce, 257
Vpmg_ibMutualPolForce, 258
Vpmg_ibNLDirectPolForce, 258
Vpmg_ibPermanentMultipoleForce, 259
Vpmg_memChk, 259
Vpmg_printColComp, 259
Vpmg_qfAtomEnergy, 260
Vpmg_qfDirectPolForce, 261
Vpmg_qfEnergy, 261
Vpmg_qfForce, 262
Vpmg_qfMutualPolForce, 263
Vpmg_qfNLDirectPolForce, 263
Vpmg_qfPermanentMultipoleEnergy, 264
Vpmg_qfPermanentMultipoleForce, 264
Vpmg_qmEnergy, 264
Vpmg_setPart, 265
Vpmg_solve, 266
Vpmg_solveLaplace, 266
Vpmg_unsetPart, 266
vpmg.c
bcCalc, 941
bcfl1, 941
bspline2, 942
bspline4, 943
d2bspline4, 943
d3bspline4, 943
dbspline2, 944
dbspline4, 944
fillcoCharge, 945
fillcoChargeMap, 945
fillcoChargeSpline1, 945
fillcoChargeSpline2, 945
fillcoCoef, 946
fillcoCoefMap, 946
fillcoCoefMol, 946
fillcoCoefMolDiel, 946
fillcoCoefMolDielNoSmooth, 946
fillcoCoefMolDielSmooth, 947
fillcoCoefMolIon, 947
fillcoCoefSpline, 947
fillcoCoefSpline3, 947
fillcoCoefSpline4, 948
fillcoPermanentMultipole, 948
markSphere, 948
multipolebc, 949
qfForceSpline1, 949
qfForceSpline2, 950
qfForceSpline4, 950
VFCHI4, 950
Vpmg_polarizEnergy, 951
Vpmg_qfEnergyPoint, 951
Vpmg_qfEnergyVolume, 952
Vpmg_qmEnergySMPBE, 952
Vpmg_splineSelect, 952
zlapSolve, 953
vpmg.h
bcCalc, 1096
bcfl1, 1096
bspline2, 1097
bspline4, 1098
d2bspline4, 1098
d3bspline4, 1098
dbspline2, 1099
dbspline4, 1099
fillcoCharge, 1099
fillcoChargeMap, 1100

fillcoChargeSpline1, 1100
fillcoChargeSpline2, 1100
fillcoCoef, 1100
fillcoCoefMap, 1101
fillcoCoefMol, 1101
fillcoCoefMolDiel, 1101
fillcoCoefMolDielNoSmooth, 1101
fillcoCoefMolDielSmooth, 1101
fillcoCoefMolIon, 1102
fillcoCoefSpline, 1102
fillcoCoefSpline3, 1102
fillcoCoefSpline4, 1102
fillcoInducedDipole, 1103
fillcoNLInducedDipole, 1103
fillcoPermanentMultipole, 1103
markSphere, 1103
multipolebc, 1104
qfForceSpline1, 1105
qfForceSpline2, 1105
qfForceSpline4, 1105
VFCHI4, 1106
Vpmg_polarizEnergy, 1106
Vpmg_qfEnergyPoint, 1106
Vpmg_qfEnergyVolume, 1107
Vpmg_qmEnergySMPBE, 1107
Vpmg_splineSelect, 1107
zlapSolve, 1108

Vpmg_ctor
Vpmg class, 248
Vpmg_ctor2
Vpmg class, 248
Vpmg_dbDirectPolForce
Vpmg class, 249
Vpmg_dbForce
Vpmg class, 249
Vpmg_dbMutualPolForce
Vpmg class, 250
Vpmg_dbNLDirectPolForce
Vpmg class, 251
Vpmg_dbPermanentMultipoleForce
Vpmg class, 251
Vpmg_dielEnergy
Vpmg class, 251
Vpmg_dielGradNorm
Vpmg class, 252
Vpmg_dtor
Vpmg class, 253
Vpmg_dtor2
Vpmg class, 253
Vpmg_energy
Vpmg class, 253
Vpmg_fieldSpline4
Vpmg class, 254
Vpmg_fillArray
Vpmg class, 254
Vpmg_fillco
Vpmg class, 255
Vpmg_force
Vpmg class, 256
Vpmg_ibDirectPolForce
Vpmg class, 257
Vpmg_ibForce
Vpmg class, 257
Vpmg_ibMutualPolForce
Vpmg class, 258
Vpmg_ibNLDirectPolForce
Vpmg class, 258
Vpmg_ibPermanentMultipoleForce
Vpmg class, 259
Vpmg_memChk
Vpmg class, 259
Vpmg_polarizEnergy
vpmg.c, 951
vpmg.h, 1106
Vpmg_printColComp
Vpmg class, 259
Vpmg_qfAtomEnergy
Vpmg class, 260
Vpmg_qfDirectPolForce
Vpmg class, 261
Vpmg_qfEnergy
Vpmg class, 261
Vpmg_qfEnergyPoint
vpmg.c, 951
vpmg.h, 1106
Vpmg_qfEnergyVolume
vpmg.c, 952
vpmg.h, 1107
Vpmg_qfForce
Vpmg class, 262
Vpmg_qfMutualPolForce
Vpmg class, 263
Vpmg_qfNLDirectPolForce
Vpmg class, 263
Vpmg_qfPermanentMultipoleEnergy
Vpmg class, 264
Vpmg_qfPermanentMultipoleForce
Vpmg class, 264
Vpmg_qmEnergy
Vpmg class, 264
Vpmg_qmEnergySMPBE
vpmg.c, 952
vpmg.h, 1107
Vpmg_setPart
Vpmg class, 265
Vpmg_solve
Vpmg class, 266
Vpmg_solveLaplace

Vpmg class, 266
Vpmg_splineSelect
vpmg.c, 952
vpmg.h, 1107
Vpmg_unsetPart
Vpmg class, 266
Vpmgp class, 267
Vpmgp_ctor, 268
Vpmgp_ctor2, 268
Vpmgp_dtor, 268
Vpmgp_dtor2, 269
Vpmgp_makeCoarse, 269
Vpmgp_size, 270
Vpmgp_ctor
Vpmgp class, 268
Vpmgp_ctor2
Vpmgp class, 268
Vpmgp_dtor
Vpmgp class, 268
Vpmgp_dtor2
Vpmgp class, 269
Vpmgp_makeCoarse
Vpmgp class, 269
Vpmgp_size
Vpmgp class, 270
Vpower
C translation of Holst group PMG code, 349
Vprtmatd
C translation of Holst group PMG code, 352
VPT_DIAG
Vfetk class, 31
VPT_IDEN
Vfetk class, 31
VPT_MG
Vfetk class, 31
VRC_FAILURE
Vhal class, 189
VRC_SUCCESS
Vhal class, 189
Vrestrc
C translation of Holst group PMG code, 352
VSM_MOL
Vhal class, 190
VSM_MOLSMOOTH
Vhal class, 190
VSM_SPLINE
Vhal class, 190
VSM_SPLINE3
Vhal class, 190
VSM_SPLINE4
Vhal class, 190
Vsmooth
C translation of Holst group PMG code, 353
Vstring class, 216
Vstring_isdigit, 217
Vstring_strcasecmp, 217
Vstring_wrappedtext, 218
Vstring_isdigit
Vstring class, 217
Vstring_strcasecmp
Vstring class, 217
Vstring_wrappedtext
Vstring class, 218
Vunit class, 218
VWARN_MSG0
vhal.h, 823
VWARN_MSG1
vhal.h, 823
VWARN_MSG2
vhal.h, 823
vx
sVfetk_LocalVar, 459
Vxaxpy
C translation of Holst group PMG code, 356
Vxcopy
C translation of Holst group PMG code, 356
Vxcopy_large
C translation of Holst group PMG code, 358
Vxcopy_small
C translation of Holst group PMG code, 359
Vxdot
C translation of Holst group PMG code, 360
Vxnrm1
C translation of Holst group PMG code, 360
Vxnrm2
C translation of Holst group PMG code, 361
Vxscal
C translation of Holst group PMG code, 361

W
sVfetk_LocalVar, 459
watepsilon
sAPOLparm, 400
watsigma
sAPOLparm, 400
wcaEnergy
sAPOLparm, 400
wcaForce
AtomForce, 394
writedataFE
High-level front-end routines, 388
writedataFlat
High-level front-end routines, 388
writedataMG
High-level front-end routines, 389
writedataXML
High-level front-end routines, 390
writefmt

sPBEparm, 438
writemat
sPBEparm, 438
writematflag
sPBEparm, 438
writematMG
High-level front-end routines, 391
writematstem
sPBEparm, 438
writestem
sPBEparm, 438
writetype
sPBEparm, 439

xcent
sVpmgp, 490
xf
sVpmg, 481
xkappa
sVpbe, 472
xlen
sVpmgp, 490
xmax
sVgrid, 463
sVpmgp, 490
xmin
sVgrid, 463
sVpmgp, 490
xp
sVgreen, 461
xpts
sVaccSurf, 443
xq
sVfetk_LocalVar, 459

ycent
sVpmgp, 490
yf
sVpmg, 481
ylen
sVpmgp, 490
ymax
sVgrid, 463
sVpmgp, 490
ymin
sVgrid, 464
sVpmgp, 491
yp
sVgreen, 461
ypts
sVaccSurf, 443

z_mem
sVpbe, 472
zcent
sVpmgp, 491
zf
sVpmg, 481
zkappa2
sVfetk_LocalVar, 459
sVpbe, 472
zks2
sVfetk_LocalVar, 460
zlapSolve
vpmg.c, 953
vpmg.h, 1108
zlen
sVpmgp, 491
zmagic
sVpbe, 472
zmax
sVgrid, 464
sVpmgp, 491
zmem
sPBEparm, 439
zmin
sVgrid, 464
sVpmgp, 491
zp
sVgreen, 461
zpts
sVaccSurf, 443