

Lamina

Lamina

Generated by Doxygen 1.10.0

1 Lemina	1
2 File Index	3
2.1 File List	3
3 File Documentation	5
3.1 README.md File Reference	5
3.2 source/AccumProps.c File Reference	5
3.2.1 Function Documentation	6
3.2.1.1 AccumProps()	6
3.3 AccumProps.c	6
3.4 source/AccumVacf.c File Reference	7
3.4.1 Function Documentation	8
3.4.1.1 AccumVacf()	8
3.4.1.2 Integrate()	9
3.4.1.3 PrintVacf()	9
3.4.1.4 ZeroVacf()	10
3.5 AccumVacf.c	10
3.6 source/AllocArrays.c File Reference	11
3.6.1 Function Documentation	12
3.6.1.1 AllocArrays()	12
3.7 AllocArrays.c	12
3.8 source/ApplyBoundaryCond.c File Reference	13
3.8.1 Function Documentation	14
3.8.1.1 ApplyBoundaryCond()	14
3.9 ApplyBoundaryCond.c	15
3.10 source/ApplyDrivingForce.c File Reference	15
3.10.1 Function Documentation	16
3.10.1.1 ApplyDrivingForce()	16
3.11 ApplyDrivingForce.c	17
3.12 source/ApplyForce.c File Reference	18
3.12.1 Function Documentation	18
3.12.1.1 ApplyForce()	18
3.13 ApplyForce.c	19
3.14 source/ApplyLeesEdwardsBoundaryCond.c File Reference	19
3.14.1 Function Documentation	20
3.14.1.1 ApplyLeesEdwardsBoundaryCond()	20
3.15 ApplyLeesEdwardsBoundaryCond.c	21
3.16 source/ApplyShear.c File Reference	21
3.16.1 Function Documentation	22
3.16.1.1 ApplyShear()	22
3.17 ApplyShear.c	22
3.18 source/ApplyViscous.c File Reference	23

3.18.1 Function Documentation	23
3.18.1.1 ApplyViscous()	23
3.19 ApplyViscous.c	24
3.20 source/BrownianStep.c File Reference	24
3.20.1 Function Documentation	25
3.20.1.1 BrownianStep()	25
3.21 BrownianStep.c	25
3.22 source/Close.c File Reference	26
3.22.1 Function Documentation	27
3.22.1.1 Close()	27
3.23 Close.c	28
3.24 source/ComputeBondForce.c File Reference	29
3.24.1 Function Documentation	30
3.24.1.1 ComputeBondForce()	30
3.25 ComputeBondForce.c	32
3.26 source/ComputeBondForce.h File Reference	34
3.26.1 Function Documentation	34
3.26.1.1 ComputeBondForce()	34
3.27 ComputeBondForce.h	36
3.28 source/ComputeForcesCells.c File Reference	36
3.28.1 Function Documentation	37
3.28.1.1 ComputeForcesCells()	37
3.29 ComputeForcesCells.c	39
3.30 source/ComputePairForce.c File Reference	41
3.30.1 Function Documentation	41
3.30.1.1 ComputePairForce()	41
3.31 ComputePairForce.c	44
3.32 source/ComputePairForce.h File Reference	46
3.32.1 Function Documentation	47
3.32.1.1 ComputePairForce()	47
3.33 ComputePairForce.h	49
3.34 source/DisplaceAtoms.c File Reference	49
3.34.1 Function Documentation	50
3.34.1.1 DisplaceAtoms()	50
3.35 DisplaceAtoms.c	51
3.36 source/DumpBonds.c File Reference	51
3.36.1 Function Documentation	52
3.36.1.1 DumpBonds()	52
3.37 DumpBonds.c	52
3.38 source/DumpPairs.c File Reference	53
3.38.1 Function Documentation	53
3.38.1.1 DumpPairs()	53

3.39 DumpPairs.c	54
3.40 source/DumpRestart.c File Reference	55
3.40.1 Function Documentation	55
3.40.1.1 DumpRestart()	55
3.41 DumpRestart.c	56
3.42 source/DumpState.c File Reference	57
3.42.1 Function Documentation	58
3.42.1.1 DumpState()	58
3.43 DumpState.c	58
3.44 source/EvalCom.c File Reference	59
3.44.1 Function Documentation	60
3.44.1.1 EvalCom()	60
3.45 EvalCom.c	60
3.46 source/EvalProps.c File Reference	61
3.46.1 Function Documentation	62
3.46.1.1 EvalProps()	62
3.47 EvalProps.c	63
3.48 source/EvalRdf.c File Reference	63
3.48.1 Function Documentation	64
3.48.1.1 EvalRdf()	64
3.49 EvalRdf.c	65
3.50 source/EvalSpacetimeCorr.c File Reference	66
3.50.1 Function Documentation	66
3.50.1.1 EvalSpacetimeCorr()	66
3.51 EvalSpacetimeCorr.c	68
3.52 source/EvalUnwrap.c File Reference	69
3.52.1 Function Documentation	70
3.52.1.1 EvalUnwrap()	70
3.53 EvalUnwrap.c	71
3.54 source/EvalVacf.c File Reference	71
3.54.1 Function Documentation	72
3.54.1.1 AccumVacf()	72
3.54.1.2 EvalVacf()	73
3.55 EvalVacf.c	73
3.56 source/EvalVrms.c File Reference	74
3.56.1 Function Documentation	75
3.56.1.1 EvalVrms()	75
3.57 EvalVrms.c	75
3.58 source/global.h File Reference	76
3.58.1 Macro Definition Documentation	80
3.58.1.1 EXTERN	80
3.58.1.2 NDIM	80

3.58.1.3 SignR	81
3.58.1.4 Sqr	81
3.58.2 Typedef Documentation	81
3.58.2.1 real	81
3.58.3 Variable Documentation	81
3.58.3.1 atom1	81
3.58.3.2 atom2	81
3.58.3.3 atomID	81
3.58.3.4 atomIDInterface	82
3.58.3.5 atomMass	82
3.58.3.6 atomRadius	82
3.58.3.7 atomType	82
3.58.3.8 ax	82
3.58.3.9 ay	82
3.58.3.10 bond	83
3.58.3.11 BondEnergy	83
3.58.3.12 BondEnergyPerAtom	83
3.58.3.13 BondID	83
3.58.3.14 BondLength	83
3.58.3.15 BondType	83
3.58.3.16 cellList	83
3.58.3.17 cells	84
3.58.3.18 cfOrg	84
3.58.3.19 cfVal	84
3.58.3.20 com	84
3.58.3.21 ComX	84
3.58.3.22 ComX0	84
3.58.3.23 ComXRatio	84
3.58.3.24 ComY	85
3.58.3.25 ComY0	85
3.58.3.26 ComYRatio	85
3.58.3.27 countAcfAv	85
3.58.3.28 countCorrAv	85
3.58.3.29 countRdf	85
3.58.3.30 DampFlag	86
3.58.3.31 deltaT	86
3.58.3.32 DeltaVXij	86
3.58.3.33 DeltaVYij	86
3.58.3.34 DeltaX	86
3.58.3.35 DeltaXij	86
3.58.3.36 DeltaXijNew	87
3.58.3.37 DeltaXijOld	87

3.58.3.38 DeltaXijOldPair	87
3.58.3.39 DeltaY	87
3.58.3.40 DeltaYij	87
3.58.3.41 DeltaYijNew	87
3.58.3.42 DeltaYijOld	87
3.58.3.43 DeltaYijOldPair	88
3.58.3.44 density	88
3.58.3.45 discDragx	88
3.58.3.46 discDragy	88
3.58.3.47 dnsty	88
3.58.3.48 dump	88
3.58.3.49 dumpPairFlag	88
3.58.3.50 fax	88
3.58.3.51 fay	89
3.58.3.52 fpbond	89
3.58.3.53 fpcom	89
3.58.3.54 fpdnsty	89
3.58.3.55 fpdump	89
3.58.3.56 fpmomentum	89
3.58.3.57 fppair	89
3.58.3.58 fprdf	89
3.58.3.59 fpresult	90
3.58.3.60 fpstress	90
3.58.3.61 fpvisc	90
3.58.3.62 fpvrms	90
3.58.3.63 fpxyz	90
3.58.3.64 freezeAtomType	90
3.58.3.65 frfAtom	90
3.58.3.66 fuSum	90
3.58.3.67 fvirSum	91
3.58.3.68 fx	91
3.58.3.69 fxByfy	91
3.58.3.70 fy	91
3.58.3.71 FyBylx	91
3.58.3.72 gamman	91
3.58.3.73 HaltCondition	91
3.58.3.74 histRdf	92
3.58.3.75 ImageX	92
3.58.3.76 ImageY	92
3.58.3.77 indexAcf	92
3.58.3.78 indexCorr	92
3.58.3.79 initUcell	92

3.58.3.80 isBonded	92
3.58.3.81 kappa	93
3.58.3.82 kb	93
3.58.3.83 kinEnergy	93
3.58.3.84 Kn	93
3.58.3.85 limitAcfAv	93
3.58.3.86 limitCorrAv	93
3.58.3.87 limitRdf	94
3.58.3.88 master	94
3.58.3.89 molID	94
3.58.3.90 momentum	94
3.58.3.91 moreCycles	94
3.58.3.92 nAtom	94
3.58.3.93 nAtomBlock	95
3.58.3.94 nAtomInterface	95
3.58.3.95 nAtomType	95
3.58.3.96 nBond	95
3.58.3.97 nBondType	95
3.58.3.98 nBuffAcf	95
3.58.3.99 nBuffCorr	95
3.58.3.100 nDiscInterface	96
3.58.3.101 nFunCorr	96
3.58.3.102 nodeDragx	96
3.58.3.103 nodeDragy	96
3.58.3.104 nPairActive	96
3.58.3.105 nPairTotal	96
3.58.3.106 nValAcf	97
3.58.3.107 nValCorr	97
3.58.3.108 pair	97
3.58.3.109 Pairatom1	97
3.58.3.110 Pairatom2	97
3.58.3.111 PairID	97
3.58.3.112 PairXij	97
3.58.3.113 PairYij	98
3.58.3.114 potEnergy	98
3.58.3.115 prefix	98
3.58.3.116 pressure	98
3.58.3.117 RadiusIJ	98
3.58.3.118 RadiusIJInv	98
3.58.3.119 rangeRdf	99
3.58.3.120 rank	99
3.58.3.121 rCut	99

3.58.3.122 rdf	99
3.58.3.123 region	99
3.58.3.124 regionH	99
3.58.3.125 result	99
3.58.3.126 rfAtom	100
3.58.3.127 ro	100
3.58.3.128 rx	100
3.58.3.129 rxUnwrap	100
3.58.3.130 ry	100
3.58.3.131 ryUnwrap	100
3.58.3.132 shearDisplacement	101
3.58.3.133 shearVelocity	101
3.58.3.134 size	101
3.58.3.135 sizeHistRdf	101
3.58.3.136 sKinEnergy	101
3.58.3.137 solver	101
3.58.3.138 spacetimeCorr	102
3.58.3.139 spacetimeCorrAv	102
3.58.3.140 speed	102
3.58.3.141 sPotEnergy	102
3.58.3.142 sPressure	102
3.58.3.143 SqrRadiusIJ	102
3.58.3.144 ssKinEnergy	103
3.58.3.145 ssPotEnergy	103
3.58.3.146 ssPressure	103
3.58.3.147 ssTotEnergy	103
3.58.3.148 stepAcf	103
3.58.3.149 stepAvg	103
3.58.3.150 stepCorr	104
3.58.3.151 stepCount	104
3.58.3.152 stepDump	104
3.58.3.153 stepEquil	104
3.58.3.154 stepLimit	104
3.58.3.155 stepRdf	104
3.58.3.156 stepTraj	105
3.58.3.157 sTotEnergy	105
3.58.3.158 strain	105
3.58.3.159 strainRate	105
3.58.3.160 strech	105
3.58.3.161 stress	105
3.58.3.162 svirSum	105
3.58.3.163 thermo	106

3.58.3.164	timeNow	106
3.58.3.165	TotalBondEnergy	106
3.58.3.166	TotalMass	106
3.58.3.167	totEnergy	106
3.58.3.168	uSum	106
3.58.3.169	uSumPair	107
3.58.3.170	uSumPairPerAtom	107
3.58.3.171	virSum	107
3.58.3.172	virSumBond	107
3.58.3.173	virSumBondxx	107
3.58.3.174	virSumBondxy	107
3.58.3.175	virSumBondyy	108
3.58.3.176	virSumPair	108
3.58.3.177	virSumPairxx	108
3.58.3.178	virSumPairxy	108
3.58.3.179	virSumPairyy	108
3.58.3.180	virSumxx	108
3.58.3.181	virSumxy	109
3.58.3.182	virSumyy	109
3.58.3.183	visc	109
3.58.3.184	viscAcf	109
3.58.3.185	viscAcfAv	109
3.58.3.186	viscAcfInt	109
3.58.3.187	viscAcfOrg	110
3.58.3.188	VMeanSqr	110
3.58.3.189	vrms	110
3.58.3.190	VRootMeanSqr	110
3.58.3.191	VSqr	110
3.58.3.192	vSum	110
3.58.3.193	vSumX	111
3.58.3.194	vSumY	111
3.58.3.195	vvSum	111
3.58.3.196	vx	111
3.58.3.197	vy	111
3.58.3.198	xBoundary	111
3.58.3.199	xyz	112
3.58.3.200	yBoundary	112
3.59	global.h	112
3.60	source/globalExtern.h File Reference	114
3.60.1	Macro Definition Documentation	118
3.60.1.1	NDIM	118
3.60.1.2	SignR	118

3.60.1.3 Sqr	118
3.60.2 Typedef Documentation	118
3.60.2.1 real	118
3.60.3 Variable Documentation	118
3.60.3.1 atom1	118
3.60.3.2 atom2	118
3.60.3.3 atomID	118
3.60.3.4 atomIDInterface	119
3.60.3.5 atomMass	119
3.60.3.6 atomRadius	119
3.60.3.7 atomType	119
3.60.3.8 ax	119
3.60.3.9 ay	119
3.60.3.10 bond	119
3.60.3.11 BondEnergy	119
3.60.3.12 BondEnergyPerAtom	119
3.60.3.13 BondID	120
3.60.3.14 BondLength	120
3.60.3.15 BondType	120
3.60.3.16 cellList	120
3.60.3.17 cells	120
3.60.3.18 cfOrg	120
3.60.3.19 cfVal	120
3.60.3.20 com	120
3.60.3.21 ComX	120
3.60.3.22 ComX0	121
3.60.3.23 ComXRatio	121
3.60.3.24 ComY	121
3.60.3.25 ComY0	121
3.60.3.26 ComYRatio	121
3.60.3.27 countAcfAv	121
3.60.3.28 countCorrAv	121
3.60.3.29 countRdf	121
3.60.3.30 DampFlag	122
3.60.3.31 deltaT	122
3.60.3.32 DeltaVXij	122
3.60.3.33 DeltaVYij	122
3.60.3.34 DeltaX	122
3.60.3.35 DeltaXij	122
3.60.3.36 DeltaXijNew	122
3.60.3.37 DeltaXijOld	122
3.60.3.38 DeltaXijOldPair	122

3.60.3.39 DeltaY	123
3.60.3.40 DeltaYij	123
3.60.3.41 DeltaYijNew	123
3.60.3.42 DeltaYijOld	123
3.60.3.43 DeltaYijOldPair	123
3.60.3.44 density	123
3.60.3.45 discDragx	123
3.60.3.46 discDragy	123
3.60.3.47 dnsty	124
3.60.3.48 dump	124
3.60.3.49 dumpPairFlag	124
3.60.3.50 fax	124
3.60.3.51 fay	124
3.60.3.52 fpbond	124
3.60.3.53 fpcom	124
3.60.3.54 fpdnsty	124
3.60.3.55 fpdump	124
3.60.3.56 fpmomentum	124
3.60.3.57 fppair	125
3.60.3.58 fprdf	125
3.60.3.59 fpresult	125
3.60.3.60 fpstress	125
3.60.3.61 fpvisc	125
3.60.3.62 fpvrms	125
3.60.3.63 fpxyz	125
3.60.3.64 freezeAtomType	125
3.60.3.65 frfAtom	125
3.60.3.66 fuSum	125
3.60.3.67 fvirSum	126
3.60.3.68 fx	126
3.60.3.69 fxByfy	126
3.60.3.70 fy	126
3.60.3.71 FyBylx	126
3.60.3.72 gamman	126
3.60.3.73 HaltCondition	126
3.60.3.74 histRdf	126
3.60.3.75 ImageX	126
3.60.3.76 ImageY	127
3.60.3.77 indexAcf	127
3.60.3.78 indexCorr	127
3.60.3.79 initUcell	127
3.60.3.80 isBonded	127

3.60.3.81 kappa	127
3.60.3.82 kb	127
3.60.3.83 kinEnergy	127
3.60.3.84 Kn	127
3.60.3.85 limitAcfAv	128
3.60.3.86 limitCorrAv	128
3.60.3.87 limitRdf	128
3.60.3.88 master	128
3.60.3.89 molID	128
3.60.3.90 momentum	128
3.60.3.91 moreCycles	128
3.60.3.92 nAtom	128
3.60.3.93 nAtomBlock	129
3.60.3.94 nAtomInterface	129
3.60.3.95 nAtomType	129
3.60.3.96 nBond	129
3.60.3.97 nBondType	129
3.60.3.98 nBuffAcf	129
3.60.3.99 nBuffCorr	129
3.60.3.100 nDiscInterface	129
3.60.3.101 nFunCorr	130
3.60.3.102 nodeDragx	130
3.60.3.103 nodeDragy	130
3.60.3.104 nPairActive	130
3.60.3.105 nPairTotal	130
3.60.3.106 nValAcf	130
3.60.3.107 nValCorr	130
3.60.3.108 pair	130
3.60.3.109 Pairatom1	131
3.60.3.110 Pairatom2	131
3.60.3.111 PairID	131
3.60.3.112 PairXij	131
3.60.3.113 PairYij	131
3.60.3.114 potEnergy	131
3.60.3.115 prefix	131
3.60.3.116 pressure	131
3.60.3.117 RadiusIJ	132
3.60.3.118 RadiusIJInv	132
3.60.3.119 rangeRdf	132
3.60.3.120 rank	132
3.60.3.121 rCut	132
3.60.3.122 rdf	132

3.60.3.123 region	132
3.60.3.124 regionH	132
3.60.3.125 result	132
3.60.3.126 rfAtom	133
3.60.3.127 ro	133
3.60.3.128 rx	133
3.60.3.129 rxUnwrap	133
3.60.3.130 ry	133
3.60.3.131 ryUnwrap	133
3.60.3.132 shearDisplacement	133
3.60.3.133 shearVelocity	133
3.60.3.134 size	133
3.60.3.135 sizeHistRdf	134
3.60.3.136 sKinEnergy	134
3.60.3.137 solver	134
3.60.3.138 spacetimeCorr	134
3.60.3.139 spacetimeCorrAv	134
3.60.3.140 speed	134
3.60.3.141 sPotEnergy	134
3.60.3.142 sPressure	134
3.60.3.143 SqrRadiusIj	135
3.60.3.144 ssKinEnergy	135
3.60.3.145 ssPotEnergy	135
3.60.3.146 ssPressure	135
3.60.3.147 ssTotEnergy	135
3.60.3.148 stepAcf	135
3.60.3.149 stepAvg	135
3.60.3.150 stepCorr	135
3.60.3.151 stepCount	136
3.60.3.152 stepDump	136
3.60.3.153 stepEquil	136
3.60.3.154 stepLimit	136
3.60.3.155 stepRdf	136
3.60.3.156 stepTraj	136
3.60.3.157 sTotEnergy	136
3.60.3.158 strain	136
3.60.3.159 strainRate	137
3.60.3.160 stretch	137
3.60.3.161 stress	137
3.60.3.162 svirSum	137
3.60.3.163 timeNow	137
3.60.3.164 TotalBondEnergy	137

3.60.3.165 TotalMass	137
3.60.3.166 totEnergy	137
3.60.3.167 uSum	138
3.60.3.168 uSumPair	138
3.60.3.169 uSumPairPerAtom	138
3.60.3.170 virSum	138
3.60.3.171 virSumBond	138
3.60.3.172 virSumBondxx	138
3.60.3.173 virSumBondxy	138
3.60.3.174 virSumBondyy	138
3.60.3.175 virSumPair	139
3.60.3.176 virSumPairxx	139
3.60.3.177 virSumPairxy	139
3.60.3.178 virSumPairyy	139
3.60.3.179 virSumxx	139
3.60.3.180 virSumxy	139
3.60.3.181 virSumyy	139
3.60.3.182 visc	139
3.60.3.183 viscAct	140
3.60.3.184 viscActAv	140
3.60.3.185 viscActInt	140
3.60.3.186 viscActOrg	140
3.60.3.187 VMeanSqr	140
3.60.3.188 vrms	140
3.60.3.189 VRootMeanSqr	140
3.60.3.190 VSqr	140
3.60.3.191 vSum	141
3.60.3.192 vSumX	141
3.60.3.193 vSumY	141
3.60.3.194 vvSum	141
3.60.3.195 vx	141
3.60.3.196 vy	141
3.60.3.197 xBoundary	141
3.60.3.198 xyz	141
3.60.3.199 yBoundary	142
3.61 globalExtern.h	142
3.62 source/Halt.c File Reference	143
3.62.1 Function Documentation	144
3.62.1.1 HaltConditionCheck()	144
3.63 Halt.c	145
3.64 source/Init.c File Reference	145
3.64.1 Function Documentation	146

3.64.1.1 Init()	146
3.65 Init.c	149
3.66 source/InitVacf.c File Reference	152
3.66.1 Function Documentation	153
3.66.1.1 InitVacf()	153
3.66.1.2 ZeroVacf()	154
3.67 InitVacf.c	154
3.68 source/Integrate.c File Reference	155
3.68.1 Function Documentation	155
3.68.1.1 Integrate()	155
3.69 Integrate.c	156
3.70 source/LeapfrogStep.c File Reference	156
3.70.1 Function Documentation	157
3.70.1.1 LeapfrogStep()	157
3.71 LeapfrogStep.c	158
3.72 source/main.c File Reference	160
3.72.1 Macro Definition Documentation	162
3.72.1.1 DEFINE_GLOBALS	162
3.72.2 Function Documentation	162
3.72.2.1 AccumProps()	162
3.72.2.2 ApplyBoundaryCond()	162
3.72.2.3 ApplyDrivingForce()	163
3.72.2.4 ApplyForce()	164
3.72.2.5 ApplyLeesEdwardsBoundaryCond()	164
3.72.2.6 ApplyShear()	165
3.72.2.7 ApplyViscous()	165
3.72.2.8 BrownianStep()	165
3.72.2.9 Close()	166
3.72.2.10 ComputeForcesCells()	167
3.72.2.11 DisplaceAtoms()	169
3.72.2.12 DumpBonds()	169
3.72.2.13 DumpPairs()	170
3.72.2.14 DumpRestart()	171
3.72.2.15 DumpState()	172
3.72.2.16 EvalCom()	172
3.72.2.17 EvalProps()	173
3.72.2.18 EvalRdf()	174
3.72.2.19 EvalSpacetimeCorr()	175
3.72.2.20 EvalUnwrap()	176
3.72.2.21 EvalVacf()	176
3.72.2.22 EvalVrms()	177
3.72.2.23 HaltConditionCheck()	178

3.72.2.24 Init()	178
3.72.2.25 LeapfrogStep()	181
3.72.2.26 main()	181
3.72.2.27 PrintCom()	184
3.72.2.28 PrintMomentum()	185
3.72.2.29 PrintStress()	185
3.72.2.30 PrintSummary()	186
3.72.2.31 PrintVrms()	186
3.72.2.32 SetupJob()	187
3.72.2.33 Trajectory()	188
3.72.2.34 VelocityVerletStep()	188
3.72.3 Variable Documentation	189
3.72.3.1 prefix	189
3.73 main.c	189
3.74 source/PrintCom.c File Reference	191
3.74.1 Function Documentation	192
3.74.1.1 PrintCom()	192
3.75 PrintCom.c	193
3.76 source/PrintMomentum.c File Reference	193
3.76.1 Function Documentation	194
3.76.1.1 PrintMomentum()	194
3.77 PrintMomentum.c	194
3.78 source/PrintStress.c File Reference	194
3.78.1 Function Documentation	195
3.78.1.1 PrintStress()	195
3.79 PrintStress.c	195
3.80 source/PrintSummary.c File Reference	196
3.80.1 Function Documentation	196
3.80.1.1 PrintSummary()	196
3.81 PrintSummary.c	197
3.82 source/PrintVacf.c File Reference	197
3.82.1 Function Documentation	198
3.82.1.1 PrintVacf()	198
3.83 PrintVacf.c	198
3.84 source/PrintVrms.c File Reference	199
3.84.1 Function Documentation	199
3.84.1.1 PrintVrms()	199
3.85 PrintVrms.c	200
3.86 source/SetupJob.c File Reference	200
3.86.1 Function Documentation	201
3.86.1.1 AccumProps()	201
3.86.1.2 AllocArrays()	202

3.86.1.3 InitVacf()	203
3.86.1.4 SetupJob()	203
3.87 SetupJob.c	204
3.88 source/Trajectory.c File Reference	205
3.88.1 Function Documentation	205
3.88.1.1 Trajectory()	205
3.89 Trajectory.c	206
3.90 source/VelocityVerletStep.c File Reference	207
3.90.1 Function Documentation	207
3.90.1.1 VelocityVerletStep()	207
3.91 VelocityVerletStep.c	208
3.92 source/ZeroVacf.c File Reference	209
3.92.1 Function Documentation	210
3.92.1.1 ZeroVacf()	210
3.93 ZeroVacf.c	210
Index	211

Chapter 1

Lemina

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

source/ AccumProps.c	5
source/ AccumVacf.c	7
source/ AllocArrays.c	11
source/ ApplyBoundaryCond.c	13
source/ ApplyDrivingForce.c	15
source/ ApplyForce.c	18
source/ ApplyLeesEdwardsBoundaryCond.c	19
source/ ApplyShear.c	21
source/ ApplyViscous.c	23
source/ BrownianStep.c	24
source/ Close.c	26
source/ ComputeBondForce.c	29
source/ ComputeBondForce.h	34
source/ ComputeForcesCells.c	36
source/ ComputePairForce.c	41
source/ ComputePairForce.h	46
source/ DisplaceAtoms.c	49
source/ DumpBonds.c	51
source/ DumpPairs.c	53
source/ DumpRestart.c	55
source/ DumpState.c	57
source/ EvalCom.c	59
source/ EvalProps.c	61
source/ EvalRdf.c	63
source/ EvalSpacetimeCorr.c	66
source/ EvalUnwrap.c	69
source/ EvalVacf.c	71
source/ EvalVrms.c	74
source/ global.h	76
source/ globalExtern.h	114
source/ Halt.c	143
source/ Init.c	145
source/ InitVacf.c	152
source/ Integrate.c	155
source/ LeapfrogStep.c	156

source/ main.c	160
source/ PrintCom.c	191
source/ PrintMomentum.c	193
source/ PrintStress.c	194
source/ PrintSummary.c	196
source/ PrintVacf.c	197
source/ PrintVrms.c	199
source/ SetupJob.c	200
source/ Trajectory.c	205
source/ VelocityVerletStep.c	207
source/ ZeroVacf.c	209

Chapter 3

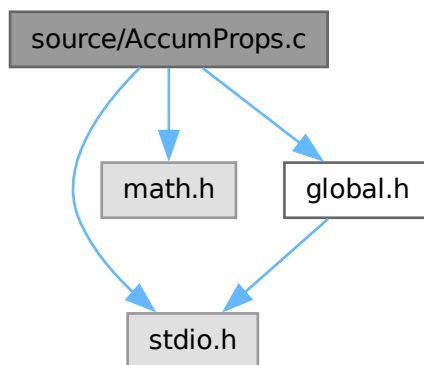
File Documentation

3.1 README.md File Reference

3.2 source/AccumProps.c File Reference

```
#include <stdio.h>
#include <math.h>
#include "global.h"
```

Include dependency graph for AccumProps.c:



Functions

- void [AccumProps](#) (int icode)

3.2.1 Function Documentation

3.2.1.1 AccumProps()

```
void AccumProps (
    int icode )
```

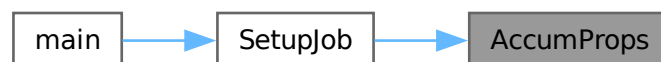
Definition at line 25 of file [AccumProps.c](#).

```
00025     {
00026     if(icode == 0){
00027         sPotEnergy = ssPotEnergy = 0.;
00028         sKinEnergy = ssKinEnergy = 0.;
00029         sPressure = ssPressure = 0.;
00030         sTotEnergy = ssTotEnergy = 0.;
00031         svirSum = 0.;
00032     }else if(icode == 1){
00033         sPotEnergy += potEnergy;
00034         ssPotEnergy += Sqr(potEnergy);
00035         sKinEnergy += kinEnergy;
00036         ssKinEnergy += Sqr(kinEnergy);
00037         sTotEnergy += totEnergy;
00038         ssTotEnergy += Sqr(totEnergy);
00039         sPressure += pressure;
00040         ssPressure += Sqr(pressure);
00041         svirSum += virSum;
00042     }else if(icode == 2){
00043         sPotEnergy /= stepAvg;
00044         ssPotEnergy /= sqrt(ssPotEnergy/stepAvg - Sqr(sPotEnergy));
00045         sTotEnergy /= stepAvg;
00046         ssTotEnergy = sqrt(ssTotEnergy/stepAvg - Sqr(sTotEnergy));
00047         sKinEnergy /= stepAvg;
00048         ssKinEnergy = sqrt(ssKinEnergy/stepAvg - Sqr(sKinEnergy));
00049         sPressure /= stepAvg;
00050         ssPressure = sqrt(ssPressure/stepAvg - Sqr(sPressure));
00051         svirSum /= stepAvg;
00052     } }
```

References [kinEnergy](#), [potEnergy](#), [pressure](#), [sKinEnergy](#), [sPotEnergy](#), [sPressure](#), [Sqr](#), [ssKinEnergy](#), [ssPotEnergy](#), [ssPressure](#), [ssTotEnergy](#), [stepAvg](#), [sTotEnergy](#), [svirSum](#), [totEnergy](#), and [virSum](#).

Referenced by [SetupJob\(\)](#).

Here is the caller graph for this function:



3.3 AccumProps.c

[Go to the documentation of this file.](#)

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
```



```

00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021  #include<stdio.h>
00022  #include<math.h>
00023  #include"global.h"
00024
00025  void AccumProps(int icode){
00026      if(icode == 0){
00027          sPotEnergy = ssPotEnergy = 0.;
00028          sKinEnergy = ssKinEnergy = 0.;
00029          sPressure = ssPressure = 0.;
00030          sTotEnergy = ssTotEnergy = 0.;
00031          svirSum = 0.;
00032      }else if(icode == 1){
00033          sPotEnergy += potEnergy;
00034          ssPotEnergy += Sqr(potEnergy);
00035          sKinEnergy += kinEnergy;
00036          ssKinEnergy += Sqr(kinEnergy);
00037          sTotEnergy += totEnergy;
00038          ssTotEnergy += Sqr(totEnergy);
00039          sPressure += pressure;
00040          ssPressure += Sqr(pressure);
00041          svirSum += virSum;
00042      }else if(icode == 2){
00043          sPotEnergy /= stepAvg;
00044          ssPotEnergy /= sqrt(ssPotEnergy/stepAvg - Sqr(sPotEnergy));
00045          sTotEnergy /= stepAvg;
00046          ssTotEnergy = sqrt(ssTotEnergy/stepAvg - Sqr(sTotEnergy));
00047          sKinEnergy /= stepAvg;
00048          ssKinEnergy = sqrt(ssKinEnergy/stepAvg - Sqr(sKinEnergy));
00049          sPressure /= stepAvg;
00050          ssPressure = sqrt(ssPressure/stepAvg - Sqr(sPressure));
00051          svirSum /= stepAvg;
00052      } }

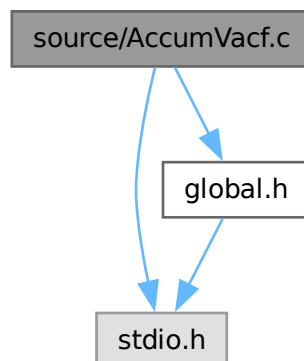
```

3.4 source/AccumVacf.c File Reference

```
#include <stdio.h>
```

```
#include "global.h"
```

Include dependency graph for AccumVacf.c:



Functions

- double [Integrate](#) (double *, int)
- void [PrintVacf](#) ()
- void [ZeroVacf](#) ()
- void [AccumVacf](#) ()

3.4.1 Function Documentation

3.4.1.1 AccumVacf()

```
void AccumVacf ( )
```

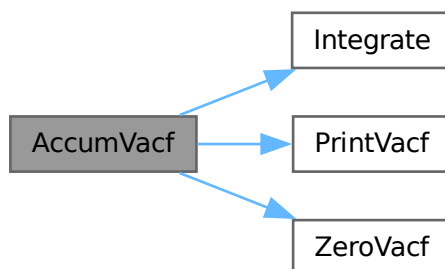
Definition at line 27 of file [AccumVacf.c](#).

```
00027     {
00028     double fac;
00029     int j, nb;
00030     for(nb = 1 ; nb <= nBuffAcf ; nb++){
00031         if(indexAcf[nb] == nValAcf){
00032             for(j = 1 ; j <= nValAcf; j++){
00033                 viscAcfAv[j] += viscAcf[nb][j];
00034             }
00035             indexAcf[nb] = 0;
00036             countAcfAv ++;
00037             if(countAcfAv == limitAcfAv){
00038                 fac = 1./(kinEnergy*region[1]*region[2]*limitAcfAv);
00039                 viscAcfInt = fac*stepAcf*deltaT*Integrate(viscAcfAv, nValAcf);
00040                 PrintVacf();
00041                 ZeroVacf();
00042             } } }
```

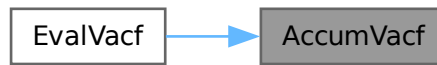
References [countAcfAv](#), [deltaT](#), [indexAcf](#), [Integrate\(\)](#), [kinEnergy](#), [limitAcfAv](#), [nBuffAcf](#), [nValAcf](#), [PrintVacf\(\)](#), [region](#), [stepAcf](#), [viscAcf](#), [viscAcfAv](#), [viscAcfInt](#), and [ZeroVacf\(\)](#).

Referenced by [EvalVacf\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.1.2 Integrate()

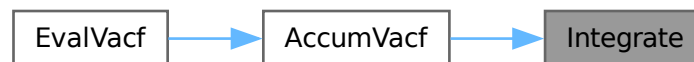
```
double Integrate (
    double * f,
    int nf )
```

Definition at line 25 of file [Integrate.c](#).

```
00025 {
00026     double s;
00027     int i;
00028     s = 0.5*(f[1] + f[nf]);
00029     for(i = 2 ; i <= nf - 1 ; i ++)
00030         s += f[i];
00031     return(s);
00032 }
```

Referenced by [AccumVacf\(\)](#).

Here is the caller graph for this function:



3.4.1.3 PrintVacf()

```
void PrintVacf ( )
```

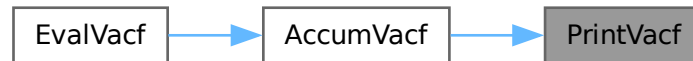
Definition at line 25 of file [PrintVacf.c](#).

```
00025 {
00026     double tVal;
00027     int j;
00028     fprintf(fpvisc,"viscosity acf\n");
00029     for(j = 1 ; j <= nValAcf ; j++){
00030         tVal = (j-1)*stepAcf*deltaT;
00031         fprintf(fpvisc, "%lf\t %lf\t %lf\n", tVal, viscAcfAv[j], viscAcfAv[j]/viscAcfAv[1]);
00032     }
00033     fprintf(fpvisc, "viscosity acf integral : %lf\n", viscAcfInt);
00034 }
```

References [deltaT](#), [fpvisc](#), [nValAcf](#), [stepAcf](#), [viscAcfAv](#), and [viscAcfInt](#).

Referenced by [AccumVacf\(\)](#).

Here is the caller graph for this function:



3.4.1.4 ZeroVacf()

```
void ZeroVacf ( )
```

Definition at line 25 of file [ZeroVacf.c](#).

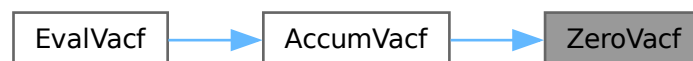
```

00025     {
00026     int j;
00027     countAcfAv= 0 ;
00028     for(j = 1 ; j <= nValAcf ; j ++)
00029         viscAcfAv[j] = 0.;
00030 }
```

References [countAcfAv](#), [nValAcf](#), and [viscAcfAv](#).

Referenced by [AccumVacf\(\)](#).

Here is the caller graph for this function:



3.5 AccumVacf.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
```

```

00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021 #include<stdio.h>
00022 #include"global.h"
00023
00024 double Integrate(double *, int);
00025 void PrintVacf();
00026 void ZeroVacf();
00027 void AccumVacf(){
00028     double fac;
00029     int j, nb;
00030     for(nb = 1 ; nb <= nBuffAcf ; nb ++){
00031         if(indexAcf[nb] == nValAcf){
00032             for(j = 1 ; j <= nValAcf; j ++){
00033                 viscAcfAv[j] += viscAcf[nb][j];
00034             }
00035             indexAcf[nb] = 0;
00036             countAcfAv ++;
00037             if(countAcfAv == limitAcfAv){
00038                 fac = 1./ (kinEnergy*region[1]*region[2]*limitAcfAv);
00039                 viscAcfInt = fac*stepAcf*deltaT*Integrate(viscAcfAv, nValAcf);
00040                 PrintVacf();
00041                 ZeroVacf();
00042             } } }
00043

```

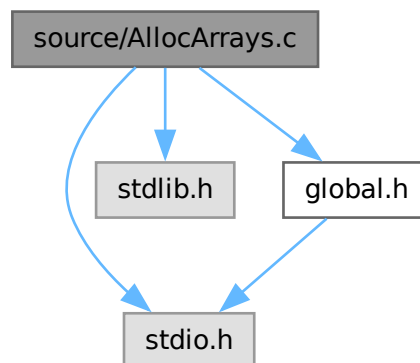
3.6 source/AllocArrays.c File Reference

```

#include <stdio.h>
#include <stdlib.h>
#include "global.h"

```

Include dependency graph for AllocArrays.c:



Functions

- void `AllocArrays()`

3.6.1 Function Documentation

3.6.1.1 AllocArrays()

void AllocArrays ()

Definition at line 25 of file [AllocArrays.c](#).

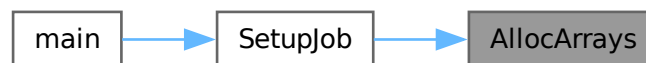
```

00025     {
00026     int n;
00027     // SPACETIME CORRELATIONS
00028     cfOrg = (double **) malloc ((nBuffCorr+1)*sizeof(double *));
00029     for (n = 0; n <= nBuffCorr; n++)
00030         cfOrg[n] = (double *) malloc ((2*nFunCorr+1)*sizeof(double));
00031
00032     cfVal = (double *) malloc ((2*nFunCorr+1)*sizeof(double));
00033     indexCorr = (int *) malloc ((nBuffCorr+1)*sizeof(int));
00034
00035     spacetimeCorr = (double **) malloc ((nBuffCorr+1)*sizeof(double));
00036     for (n = 0; n <= nBuffCorr; n++)
00037         spacetimeCorr[n] = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));
00038
00039     spacetimeCorrAv = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));
00040     // VISCOSITY
00041     indexAcf = (double *) malloc ((nBuffAcf+1)*sizeof(double));
00042     viscAcf = (double **) malloc ((nBuffAcf+1)*sizeof(double *));
00043     for (n = 0; n <= nBuffAcf; n++)
00044         viscAcf[n] = (double *) malloc ((nValAcf+1)*sizeof(double));
00045
00046     viscAcfOrg = (double *) malloc ((nBuffAcf+1)*sizeof(double));
00047     viscAcfAv = (double *) malloc ((nValAcf+1)*sizeof(double));
00048
00049     // RDF
00050     histRdf = (double *) malloc ((sizeHistRdf+1)*sizeof(double));
00051 }
```

References [cfOrg](#), [cfVal](#), [histRdf](#), [indexAcf](#), [indexCorr](#), [nBuffAcf](#), [nBuffCorr](#), [nFunCorr](#), [nValAcf](#), [nValCorr](#), [sizeHistRdf](#), [spacetimeCorr](#), [spacetimeCorrAv](#), [viscAcf](#), [viscAcfAv](#), and [viscAcfOrg](#).

Referenced by [SetupJob\(\)](#).

Here is the caller graph for this function:



3.7 AllocArrays.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  */
```

```

00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021  #include<stdio.h>
00022  #include<stdlib.h>
00023  #include"global.h"
00024
00025  void AllocArrays(){
00026      int n;
00027      // SPACETIME CORRELATIONS
00028      cfOrg = (double **) malloc ((nBuffCorr+1)*sizeof(double *));
00029      for (n = 0; n <= nBuffCorr; n++)
00030          cfOrg[n] = (double *) malloc ((2*nFunCorr+1)*sizeof(double));
00031
00032      cfVal = (double *) malloc ((2*nFunCorr+1)*sizeof(double));
00033      indexCorr = (int *) malloc ((nBuffCorr+1)*sizeof(int));
00034
00035      spatetimeCorr = (double **) malloc ((nBuffCorr+1)*sizeof(double));
00036      for (n = 0; n <= nBuffCorr; n++)
00037          spatetimeCorr[n] = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));
00038
00039      spatetimeCorrAv = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));
00040      // VISCOSITY
00041      indexAcf = (double *)malloc((nBuffAcf+1)*sizeof(double));
00042      viscAcf = (double **)malloc((nBuffAcf+1)*sizeof(double *));
00043      for (n = 0 ; n <= nBuffAcf ; n++)
00044          viscAcf[n] = (double *)malloc((nValAcf+1)*sizeof(double ));
00045
00046      viscAcfOrg = (double *)malloc((nBuffAcf+1)*sizeof(double));
00047      viscAcfAv = (double *)malloc((nValAcf+1)*sizeof(double));
00048
00049      // RDF
00050      histRdf = (double *)malloc((sizeHistRdf+1)*sizeof(double));
00051  }

```

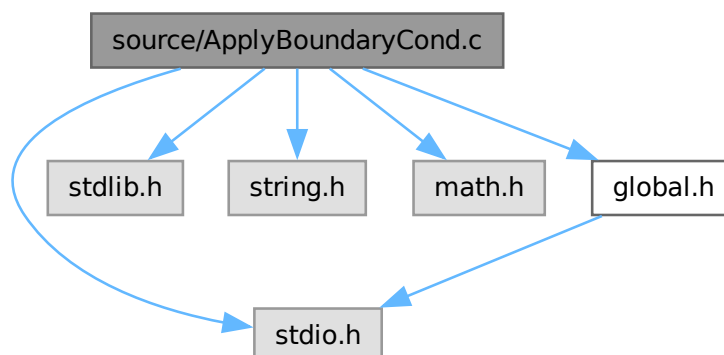
3.8 source/ApplyBoundaryCond.c File Reference

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "global.h"

```

Include dependency graph for ApplyBoundaryCond.c:



Functions

- void [ApplyBoundaryCond](#) ()

3.8.1 Function Documentation

3.8.1.1 ApplyBoundaryCond()

void [ApplyBoundaryCond](#) ()

Definition at line 27 of file [ApplyBoundaryCond.c](#).

```

00027     {
00028     int n;
00029     for(n = 1 ; n <= nAtom ; n ++){
00030     if(strcmp(xBoundary, "p") == 0 && strcmp(yBoundary, "p") == 0){           // P.B.C along x and y axis
00031         rx[n] -= region[1]*rint(rx[n]/region[1]);
00032         ry[n] -= region[2]*rint(ry[n]/region[2]);
00033     } else if (strcmp(xBoundary, "r") == 0 && strcmp(yBoundary, "r") == 0){   //R.B.C. along x and y
axis
00034         if((rx[n] + atomRadius[n]) >= regionH[1]){
00035             rx[n] = 0.999999*regionH[1] - atomRadius[n]; vx[n] = -vx[n] ;
00036         }if((rx[n]-atomRadius[n]) < -regionH[1]){
00037             rx[n] = -0.999999*regionH[1] + atomRadius[n]; vx[n] = -vx[n] ;
00038         }
00039         if((ry[n] + atomRadius[n])>= regionH[2]){
00040             ry[n] = 0.999999*regionH[2] - atomRadius[n]; vy[n] = -vy[n] ;
00041         }if((ry[n]-atomRadius[n]) < -regionH[2]){
00042             ry[n] = -0.999999*regionH[2] + atomRadius[n]; vy[n] = -vy[n] ;
00043         }
00044     } else if (strcmp(xBoundary, "p") == 0 && strcmp(yBoundary, "r") == 0){   //P.B.C. along x and R.B.C
along y axis
00045         rx[n] -= region[1]*rint(rx[n]/region[1]);
00046         if((ry[n] + atomRadius[n]) >= regionH[2]){
00047             ry[n] = 0.999999*regionH[2] - atomRadius[n]; vy[n] = -vy[n] ;
00048         }if((ry[n] - atomRadius[n]) < -regionH[2]){
00049             ry[n] = -0.999999*regionH[2] + atomRadius[n]; vy[n] = -vy[n] ;
00050         }
00051     } else if(strcmp(xBoundary, "r") == 0 && strcmp(yBoundary, "p") == 0){   //R.B.C. along x and P.B.C
along y axis
00052         if((rx[n] + atomRadius[n]) >= regionH[1]){
00053             rx[n] = 0.999999*regionH[1] - atomRadius[n]; vx[n] = -vx[n] ;
00054         }if((rx[n] - atomRadius[n]) < -regionH[1]){
00055             rx[n] = -0.999999*regionH[1] + atomRadius[n]; vx[n] = -vx[n] ;
00056         }
00057         ry[n] -= region[2]*rint(ry[n]/region[2]);
00058     } else {
00059         // Print error message and exit the program
00060         fprintf(fpresult, "Error: Invalid boundary configuration: '%s %s'\n", xBoundary, yBoundary);
00061         exit(EXIT_FAILURE); // Exit with failure status
00062     }
00063 }
00064 }

```

References [atomRadius](#), [fpresult](#), [nAtom](#), [region](#), [regionH](#), [rx](#), [ry](#), [vx](#), [vy](#), [xBoundary](#), and [yBoundary](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.9 ApplyBoundaryCond.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021 #include<stdio.h>
00022 #include<stdlib.h>
00023 #include<string.h>
00024 #include<math.h>
00025 #include"global.h"
00026
00027 void ApplyBoundaryCond(){
00028     int n;
00029     for(n = 1 ; n <= nAtom ; n ++){
00030         if(strcmp(xBoundary, "p") == 0 && strcmp(yBoundary, "p") == 0){           // P.B.C along x and y axis
00031             rx[n] -= region[1]*rint(rx[n]/region[1]);
00032             ry[n] -= region[2]*rint(ry[n]/region[2]);
00033         } else if (strcmp(xBoundary, "r") == 0 && strcmp(yBoundary, "r") == 0){ //R.B.C. along x and y
axis
00034             if((rx[n] + atomRadius[n]) >= regionH[1]){
00035                 rx[n] = 0.999999*regionH[1] - atomRadius[n]; vx[n] = -vx[n] ;
00036             }if((rx[n]-atomRadius[n]) < -regionH[1]){
00037                 rx[n] = -0.999999*regionH[1] + atomRadius[n]; vx[n] = -vx[n] ;
00038             }
00039             if((ry[n] + atomRadius[n])>= regionH[2]){
00040                 ry[n] = 0.999999*regionH[2] - atomRadius[n]; vy[n] = -vy[n] ;
00041             }if((ry[n]-atomRadius[n]) < -regionH[2]){
00042                 ry[n] = -0.999999*regionH[2] + atomRadius[n]; vy[n] = -vy[n] ;
00043             }
00044         } else if (strcmp(xBoundary, "p") == 0 && strcmp(yBoundary, "r") == 0){ //P.B.C. along x and R.B.C
along y axis
00045             rx[n] -= region[1]*rint(rx[n]/region[1]);
00046             if((ry[n] + atomRadius[n]) >= regionH[2]){
00047                 ry[n] = 0.999999*regionH[2] - atomRadius[n]; vy[n] = -vy[n] ;
00048             }if((ry[n] - atomRadius[n]) < -regionH[2]){
00049                 ry[n] = -0.999999*regionH[2] + atomRadius[n]; vy[n] = -vy[n] ;
00050             }
00051         } else if(strcmp(xBoundary, "r") == 0 && strcmp(yBoundary, "p") == 0){ //R.B.C. along x and P.B.C
along y axis
00052             if((rx[n] + atomRadius[n]) >= regionH[1]){
00053                 rx[n] = 0.999999*regionH[1] - atomRadius[n]; vx[n] = -vx[n] ;
00054             }if((rx[n] - atomRadius[n]) < -regionH[1]){
00055                 rx[n] = -0.999999*regionH[1] + atomRadius[n]; vx[n] = -vx[n] ;
00056             }
00057             ry[n] -= region[2]*rint(ry[n]/region[2]);
00058         } else {
00059             // Print error message and exit the program
00060             fprintf(fpresult, "Error: Invalid boundary configuration: '%s %s'\n", xBoundary, yBoundary);
00061             exit(EXIT_FAILURE); // Exit with failure status
00062         }
00063     }
00064 }
```

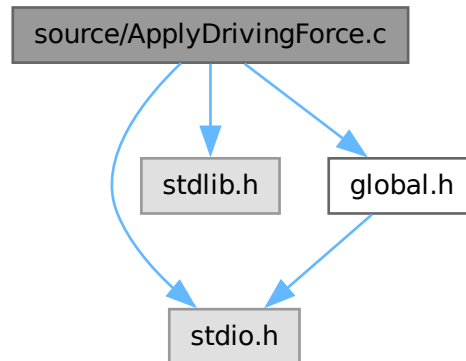
3.10 source/ApplyDrivingForce.c File Reference

```

#include <stdio.h>
#include <stdlib.h>
```

```
#include "global.h"
```

Include dependency graph for ApplyDrivingForce.c:



Functions

- void [ApplyDrivingForce](#) ()

3.10.1 Function Documentation

3.10.1.1 ApplyDrivingForce()

```
void ApplyDrivingForce ( )
```

Definition at line 25 of file [ApplyDrivingForce.c](#).

```

00025     {
00026     int n;
00027     double Vxblock, Vyblock;
00028     double Vxsubstrate, Vysubstrate;
00029     Vxblock = 0.0; Vyblock = 0.0;
00030     Vxsubstrate = 0.0; Vysubstrate = 0.0;
00031     double gammav;
00032     gammav = 0.0;
00033
00034     double count_substrate = 0;
00035     double count_block = 0;
00036
00037     for(n = 1 ; n <= nAtom; n++){
00038     if(atomType[n] == 1 || atomType[n] == 2){
00039     Vxsubstrate += vx[n]; Vysubstrate += vy[n];
00040     count_substrate++;
00041     }
00042     if(atomType[n] == 3 || atomType[n] == 4){
00043     Vxblock += vx[n]; Vyblock += vy[n];
00044     count_block++;
00045     } }
00046
00047     if(count_substrate > 0) {
00048     Vxsubstrate /= count_substrate;
00049     Vysubstrate /= count_substrate;
00050     }
00051
00052     if(count_block > 0) {
00053     Vxblock /= count_block;
00054     Vyblock /= count_block;
00055     }

```

```

00056
00057 for(n = 1 ; n <= nAtom; n++){
00058     if(atomType[n] == 1 || atomType[n] == 2){
00059         ax[n] += -gammav * (vx[n] - Vxsubstrate);
00060         ay[n] += -gammav * (vy[n] - Vysubstrate);
00061     }
00062     if(atomType[n] == 3 || atomType[n] == 4){
00063         ax[n] += -gammav * (vx[n] - Vxblock);
00064         ay[n] += -gammav * (vy[n] - Vyblock);
00065     } }

```

References [atomType](#), [ax](#), [ay](#), [nAtom](#), [vx](#), and [vy](#).

3.11 ApplyDrivingForce.c

[Go to the documentation of this file.](#)

```

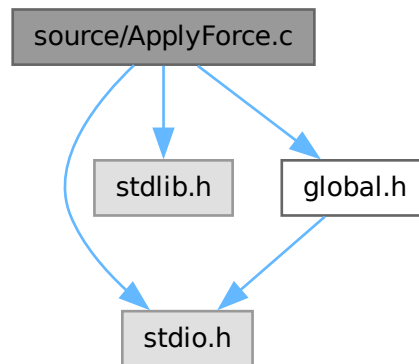
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021 #include<stdio.h>
00022 #include<stdlib.h>
00023 #include"global.h"
00024
00025 void ApplyDrivingForce(){
00026     int n;
00027     double Vxblock, Vyblock;
00028     double Vxsubstrate, Vysubstrate;
00029     Vxblock = 0.0; Vyblock = 0.0;
00030     Vxsubstrate = 0.0; Vysubstrate = 0.0;
00031     double gammav;
00032     gammav = 0.0;
00033
00034     double count_substrate = 0;
00035     double count_block = 0;
00036
00037     for(n = 1 ; n <= nAtom; n++){
00038         if(atomType[n] == 1 || atomType[n] == 2){
00039             Vxsubstrate += vx[n]; Vysubstrate += vy[n];
00040             count_substrate++;
00041         }
00042         if(atomType[n] == 3 || atomType[n] == 4){
00043             Vxblock += vx[n]; Vyblock += vy[n];
00044             count_block++;
00045         } }
00046
00047     if(count_substrate > 0) {
00048         Vxsubstrate /= count_substrate;
00049         Vysubstrate /= count_substrate;
00050     }
00051
00052     if(count_block > 0) {
00053         Vxblock /= count_block;
00054         Vyblock /= count_block;
00055     }
00056
00057     for(n = 1 ; n <= nAtom; n++){
00058         if(atomType[n] == 1 || atomType[n] == 2){
00059             ax[n] += -gammav * (vx[n] - Vxsubstrate);
00060             ay[n] += -gammav * (vy[n] - Vysubstrate);
00061         }
00062         if(atomType[n] == 3 || atomType[n] == 4){
00063             ax[n] += -gammav * (vx[n] - Vxblock);
00064             ay[n] += -gammav * (vy[n] - Vyblock);
00065         } } }
00066
00067

```

3.12 source/ApplyForce.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "global.h"
```

Include dependency graph for ApplyForce.c:



Functions

- void [ApplyForce](#) ()

3.12.1 Function Documentation

3.12.1.1 ApplyForce()

```
void ApplyForce ( )
```

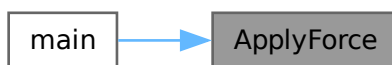
Definition at line 25 of file [ApplyForce.c](#).

```
00025     {
00026     int n;
00027     double lx;
00028     lx = regionH[1];
00029     fy = (FyBylx * lx)/nAtomBlock;
00030     fx = fxByfy * fy;
00031     for(n = 1; n <= nAtom; n++){
00032     if(molID[n] == 2){
00033         ax[n] += fx;
00034         ay[n] -= fy;
00035     } }
```

References [ax](#), [ay](#), [fx](#), [fxByfy](#), [fy](#), [FyBylx](#), [molID](#), [nAtom](#), [nAtomBlock](#), and [regionH](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.13 ApplyForce.c

[Go to the documentation of this file.](#)

```

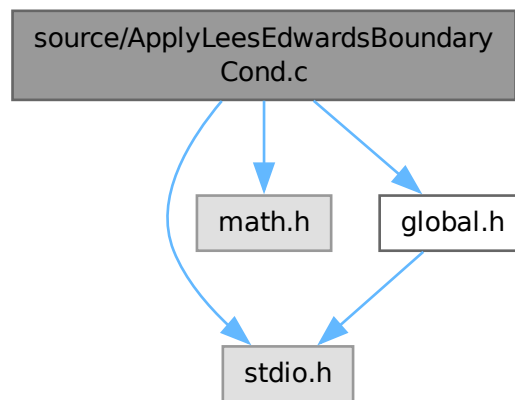
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021 #include<stdio.h>
00022 #include<stdlib.h>
00023 #include"global.h"
00024
00025 void ApplyForce() {
00026     int n;
00027     double lx;
00028     lx = regionH[1];
00029     fy = (FyBylx * lx)/nAtomBlock;
00030     fx = fxByfy * fy;
00031     for(n = 1; n <= nAtom; n++){
00032         if(molID[n] == 2){
00033             ax[n] += fx;
00034             ay[n] -= fy;
00035         } }
  
```

3.14 source/ApplyLeesEdwardsBoundaryCond.c File Reference

```

#include <stdio.h>
#include <math.h>
#include "global.h"
  
```

Include dependency graph for ApplyLeesEdwardsBoundaryCond.c:



Functions

- void [ApplyLeesEdwardsBoundaryCond](#) ()

3.14.1 Function Documentation

3.14.1.1 ApplyLeesEdwardsBoundaryCond()

```
void ApplyLeesEdwardsBoundaryCond ( )
```

Definition at line 25 of file [ApplyLeesEdwardsBoundaryCond.c](#).

```

00025                                     {
00026   int n;
00027   for (n = 1; n <= nAtom; n++) {
00028     //PBC along x-direction
00029     if(rx[n] >= regionH[1])
00030       rx[n] -= region[1];
00031     else if(rx[n] < -regionH[1])
00032       rx[n] += region[1];
00033
00034     //LEBC along y-direction
00035     if(ry[n] >= regionH[2]){
00036       rx[n] -= shearDisplacement;
00037       if(rx[n] < -regionH[1]) rx[n] += region[1];
00038       //vx[n] -= shearVelocity;
00039       ry[n] -= region[2];
00040     }else if(ry[n] < -regionH[2]){
00041       rx[n] += shearDisplacement;
00042       if(rx[n] >= regionH[1]) rx[n] -= region[1];
00043       //vx[n] += shearVelocity;
00044       ry[n] += region[2];
00045     }
00046   }
00047 }
```

References [nAtom](#), [region](#), [regionH](#), [rx](#), [ry](#), and [shearDisplacement](#).

3.15 ApplyLeesEdwardsBoundaryCond.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021 #include <stdio.h>
00022 #include <math.h>
00023 #include "global.h"
00024
00025 void ApplyLeesEdwardsBoundaryCond() {
00026     int n;
00027     for (n = 1; n <= nAtom; n++) {
00028         //PBC along x-direction
00029         if(rx[n] >= regionH[1])
00030             rx[n] -= region[1];
00031         else if(rx[n] < -regionH[1])
00032             rx[n] += region[1];
00033
00034         //LEBC along y-direction
00035         if(ry[n] >= regionH[2]){
00036             rx[n] -= shearDisplacement;
00037             if(rx[n] < -regionH[1]) rx[n] += region[1];
00038             //vx[n] -= shearVelocity;
00039             ry[n] -= region[2];
00040         }else if(ry[n] < -regionH[2]){
00041             rx[n] += shearDisplacement;
00042             if(rx[n] >= regionH[1]) rx[n] -= region[1];
00043             //vx[n] += shearVelocity;
00044             ry[n] += region[2];
00045         }
00046     }
00047 }
00048

```

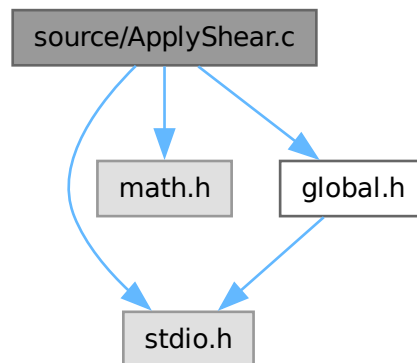
3.16 source/ApplyShear.c File Reference

```

#include <stdio.h>
#include <math.h>
#include "global.h"

```

Include dependency graph for ApplyShear.c:



Functions

- void [ApplyShear](#) ()

3.16.1 Function Documentation

3.16.1.1 ApplyShear()

```
void ApplyShear ( )
```

Definition at line 25 of file [ApplyShear.c](#).

```

00025     {
00026     int n;
00027     for(n = 1 ; n <= nAtom ; n ++){
00028         rx[n] += strain * ry[n];
00029         //vx[n] += stranRate * ry[n];
00030     } }
```

References [nAtom](#), [rx](#), [ry](#), and [strain](#).

3.17 ApplyShear.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
```



```

00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021  #include<stdio.h>
00022  #include<math.h>
00023  #include"global.h"
00024
00025  void ApplyShear() {
00026      int n;
00027      for(n = 1 ; n <= nAtom ; n ++){
00028          rx[n] += strain * ry[n];
00029          //vx[n] += stranRate * ry[n];
00030      } }

```

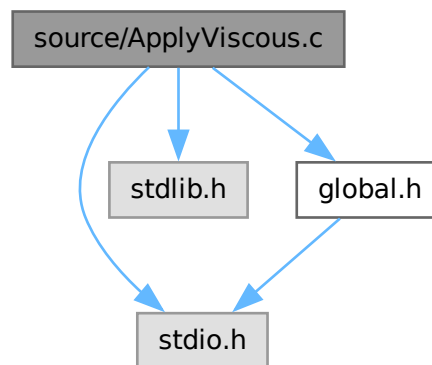
3.18 source/ApplyViscous.c File Reference

```

#include <stdio.h>
#include <stdlib.h>
#include "global.h"

```

Include dependency graph for ApplyViscous.c:



Functions

- void [ApplyViscous](#) ()

3.18.1 Function Documentation

3.18.1.1 ApplyViscous()

```
void ApplyViscous ( )
```

Definition at line 25 of file [ApplyViscous.c](#).

```

00025      {
00026      int n;
00027      double gammav;
00028      gammav = 1.0;
00029      for(n = 1 ; n <= nAtom; n ++){
00030          ax[n] += -gammav * vx[n];
00031          ay[n] += -gammav * vy[n];
00032      } }

```

References [ax](#), [ay](#), [nAtom](#), [vx](#), and [vy](#).

3.19 ApplyViscous.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021 #include<stdio.h>
00022 #include<stdlib.h>
00023 #include"global.h"
00024
00025 void ApplyViscous(){
00026     int n;
00027     double gammav;
00028     gammav = 1.0;
00029     for(n = 1 ; n <= nAtom; n++){
00030         ax[n] += -gammav * vx[n];
00031         ay[n] += -gammav * vy[n];
00032     }
00033 }
00034

```

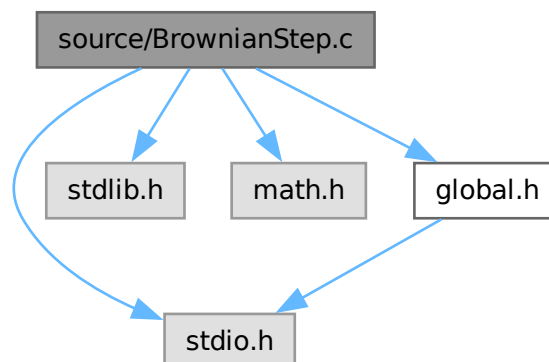
3.20 source/BrownianStep.c File Reference

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "global.h"

```

Include dependency graph for BrownianStep.c:



Functions

- void [BrownianStep](#) ()

3.20.1 Function Documentation

3.20.1.1 BrownianStep()

void BrownianStep ()

Definition at line 26 of file [BrownianStep.c](#).

```

00026     {
00027     if(stepCount <= stepEquil){
00028         double A, S1, S2, T;
00029         int n;
00030         S1 = 0.; S2 = 0;
00031         double halfdt = 0.5*deltaT;
00032         for (n = 1; n <= nAtom; n++){
00033             T = vx[n] + halfdt * ax[n];
00034             S1 += T * ax[n];
00035             S2 += Sqr(T);
00036
00037             T = vy[n] + halfdt * ay[n];
00038             S1 += T * ay[n];
00039             S2 += Sqr(T);
00040         }
00041         A = -S1 / S2;
00042         double C = 1 + A*deltaT ;
00043         double D = deltaT * (1 + 0.5 * A * deltaT);
00044         for (n = 1; n <= nAtom; n++){
00045             vx[n] = C * vx[n] + D * ax[n];
00046             rx[n] += deltaT * vx[n];
00047             vy[n] = C * vy[n] + D * ay[n];
00048             ry[n] += deltaT * vy[n];
00049         }
00050     }else{
00051         int n;
00052         //SETTING TEMP = 0.0
00053         if (stepCount == stepEquil+1){
00054             for(n = 1 ; n <= nAtom ; n ++){
00055                 vx[n] = 0.0;
00056                 vy[n] = 0.0;
00057             }
00058             double zeta = 1.0;
00059             double dx, dy;
00060             for(n = 1 ; n <= nAtom ; n ++){
00061                 dx = rx[n];
00062                 rx[n] += zeta * ax[n] * deltaT;
00063                 dx = rx[n] - dx;
00064                 vx[n] = dx/deltaT;
00065                 dy = ry[n];
00066                 ry[n] += zeta * ay[n] * deltaT;
00067                 dy = ry[n] - dy;
00068                 vy[n] = dy/deltaT;
00069             }
00070         }
00071     }

```

References [ax](#), [ay](#), [deltaT](#), [nAtom](#), [rx](#), [ry](#), [Sqr](#), [stepCount](#), [stepEquil](#), [vx](#), and [vy](#).

3.21 BrownianStep.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,

```

```

00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021 #include<stdio.h>
00022 #include<stdlib.h>
00023 #include<math.h>
00024 #include"global.h"
00025
00026 void BrownianStep(){
00027     if(stepCount <= stepEquil){
00028         double A, S1, S2, T;
00029         int n;
00030         S1 = 0.; S2 = 0;
00031         double halfdt = 0.5*deltaT;
00032         for (n = 1; n <= nAtom; n++){
00033             T = vx[n] + halfdt * ax[n];
00034             S1 += T * ax[n];
00035             S2 += Sqr(T);
00036
00037             T = vy[n] + halfdt * ay[n];
00038             S1 += T * ay[n];
00039             S2 += Sqr(T);
00040         }
00041         A = -S1 / S2;
00042         double C = 1 + A*deltaT ;
00043         double D = deltaT * (1 + 0.5 * A * deltaT);
00044         for (n = 1; n <= nAtom; n++){
00045             vx[n] = C * vx[n] + D * ax[n];
00046             rx[n] += deltaT * vx[n];
00047             vy[n] = C * vy[n] + D * ay[n];
00048             ry[n] += deltaT * vy[n];
00049         }
00050     }else{
00051         int n;
00052         //SETTING TEMP = 0.0
00053         if (stepCount == stepEquil+1){
00054             for(n = 1 ; n <= nAtom ; n ++){
00055                 vx[n] = 0.0;
00056                 vy[n] = 0.0;
00057             }
00058             double zeta = 1.0;
00059             double dx, dy;
00060             for(n = 1 ; n <= nAtom ; n ++){
00061                 dx = rx[n];
00062                 rx[n] += zeta * ax[n] * deltaT;
00063                 dx = rx[n] - dx;
00064                 vx[n] = dx/deltaT;
00065                 dy = ry[n];
00066                 ry[n] += zeta * ay[n] * deltaT;
00067                 dy = ry[n] - dy;
00068                 vy[n] = dy/deltaT;
00069             }
00070         }
00071     }
00072

```

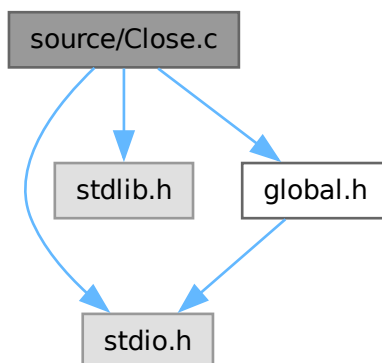
3.22 source/Close.c File Reference

```

#include <stdio.h>
#include <stdlib.h>
#include "global.h"

```

Include dependency graph for Close.c:



Functions

- void [Close](#) ()

3.22.1 Function Documentation

3.22.1.1 Close()

```
void Close ( )
```

Definition at line 24 of file [Close.c](#).

```

00024     {
00025     int n;
00026     free(rx);
00027     free(ry);
00028     free(vx);
00029     free(vy);
00030     free(ax);
00031     free(ay);
00032     free(fax);
00033     free(fay);
00034     free(cellList);
00035
00036     free(atomID); free(atomType); free(atomRadius); free(atomMass);
00037     free(speed);
00038     free(atom1); free(atom2); free(BondID);
00039     free(BondType); free(kb); free(ro);
00040     free(ImageX); free(ImageY); free(rxUnwrap); free(ryUnwrap);
00041     free(atomIDInterface);
00042     free(PairID); free(Pairatom1); free(Pairatom2);
00043     free(PairXij); free(PairYij);
00044
00045     free(DeltaXijOld);
00046     free(DeltaYijOld);
00047
00048     free(molID);
00049
00050     for (n = 0; n <= nAtom; n++) {
00051         free(isBonded[n]);
00052     }
00053     free(isBonded);
00054
00055
00056
  
```

```

00057     for(n = 0; n <= nAtom; n++) {
00058         free(DeltaXijOldPair[n]);
00059         free(DeltaYijOldPair[n]);
00060     }
00061     free(DeltaXijOldPair);
00062     free(DeltaYijOldPair);
00063
00064     for (n = 0; n <= nBuffCorr; n++){
00065         free(cfOrg[n]);
00066         free(spacetimeCorr[n]);
00067     }
00068     free(cfOrg);
00069     free(spacetimeCorr);
00070     free(cfVal);
00071     free(indexCorr);
00072     free(spacetimeCorrAv);
00073
00074     free(indexAcf);
00075     free(viscAcfOrg);
00076     free(viscAcfAv);
00077     for(n = 0 ; n <= nBuffAcf ; n ++){
00078         free(viscAcf[n]);
00079     }
00080     free(viscAcf);
00081 }

```

References [atom1](#), [atom2](#), [atomID](#), [atomIDInterface](#), [atomMass](#), [atomRadius](#), [atomType](#), [ax](#), [ay](#), [BondID](#), [BondType](#), [cellList](#), [cfOrg](#), [cfVal](#), [DeltaXijOld](#), [DeltaXijOldPair](#), [DeltaYijOld](#), [DeltaYijOldPair](#), [fax](#), [fay](#), [ImageX](#), [ImageY](#), [indexAcf](#), [indexCorr](#), [isBonded](#), [kb](#), [molID](#), [nAtom](#), [nBuffAcf](#), [nBuffCorr](#), [Pairatom1](#), [Pairatom2](#), [PairID](#), [PairXij](#), [PairYij](#), [ro](#), [rx](#), [rxUnwrap](#), [ry](#), [ryUnwrap](#), [spacetimeCorr](#), [spacetimeCorrAv](#), [speed](#), [viscAcf](#), [viscAcfAv](#), [viscAcfOrg](#), [vx](#), and [vy](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.23 Close.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021 #include<stdio.h>
00022 #include<stdlib.h>

```

```

00023 #include"global.h"
00024 void Close(){
00025     int n;
00026     free(rx);
00027     free(ry);
00028     free(vx);
00029     free(vy);
00030     free(ax);
00031     free(ay);
00032     free(fax);
00033     free(fay);
00034     free(cellList);
00035
00036     free(atomID); free(atomType); free(atomRadius); free(atomMass);
00037     free(speed);
00038     free(atom1); free(atom2); free(BondID);
00039     free(BondType); free(kb); free(ro);
00040     free(ImageX); free(ImageY); free(rxUnwrap); free(ryUnwrap);
00041     free(atomIDInterface);
00042     free(PairID); free(Pairatom1); free(Pairatom2);
00043     free(PairXij); free(PairYij);
00044
00045     free(DeltaXijOld);
00046     free(DeltaYijOld);
00047
00048     free(molID);
00049
00050     for (n = 0; n <= nAtom; n++) {
00051         free(isBonded[n]);
00052     }
00053     free(isBonded);
00054
00055
00056
00057     for(n = 0; n <= nAtom; n++) {
00058         free(DeltaXijOldPair[n]);
00059         free(DeltaYijOldPair[n]);
00060     }
00061     free(DeltaXijOldPair);
00062     free(DeltaYijOldPair);
00063
00064     for (n = 0; n <= nBuffCorr; n++){
00065         free(cfOrg[n]);
00066         free(spacetimeCorr[n]);
00067     }
00068     free(cfOrg);
00069     free(spacetimeCorr);
00070     free(cfVal);
00071     free(indexCorr);
00072     free(spacetimeCorrAv);
00073
00074     free(indexAcf);
00075     free(viscAcfOrg);
00076     free(viscAcfAv);
00077     for(n = 0 ; n <= nBuffAcf ; n ++){
00078         free(viscAcf[n]);
00079     }
00080     free(viscAcf);
00081 }

```

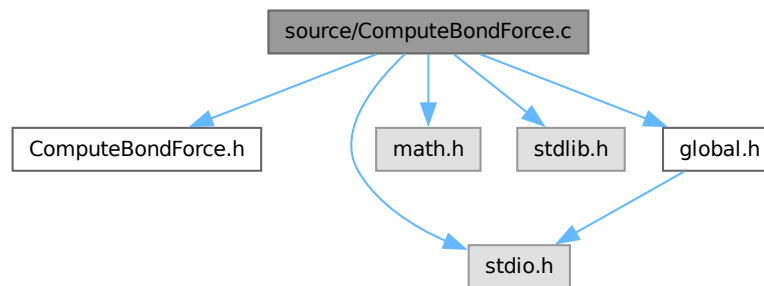
3.24 source/ComputeBondForce.c File Reference

```

#include "ComputeBondForce.h"
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "global.h"

```

Include dependency graph for ComputeBondForce.c:



Functions

- void [ComputeBondForce](#) ()

3.24.1 Function Documentation

3.24.1.1 ComputeBondForce()

```
void ComputeBondForce ( )
```

Definition at line 28 of file [ComputeBondForce.c](#).

```

00028         {
00029     int n;
00030     double dr[NDIM+1], r, rr, ri, roi;
00031     double uVal, fcVal;
00032
00033     uVal = 0.0; TotalBondEnergy = 0.0;
00034     virSumBond = 0.0; virSumBondxx = 0.0; virSumBondyy = 0.0; virSumBondxy = 0.0;
00035
00036     double vr[NDIM+1], fdVal, rri;
00037
00038     for(n = 1 ; n <= nAtom ; n ++){
00039         nodeDragx[n] = 0.0;
00040         nodeDragy[n] = 0.0;
00041     } //Important change made on 03Apr2025. Mention it in GitHub
00042
00043     int atom1ID, atom2ID;
00044
00045     for(n=1; n<=nBond; n++){
00046         rr = 0.0; rri = 0.0; fcVal = 0.0; fdVal = 0.0; stretch = 0.0;
00047         atom1ID = atom1[n];
00048         atom2ID = atom2[n];
00049
00050         dr[1] = rx[atom1ID] - rx[atom2ID];
00051         if(dr[1] >= regionH[1])
00052             dr[1] -= region[1];
00053         else if(dr[1] < -regionH[1])
00054             dr[1] += region[1];
00055
00056         dr[2] = ry[atom1ID] - ry[atom2ID];
00057         if(dr[2] >= regionH[2]){
00058             dr[1] -= shearDisplacement;
00059             if(dr[1] < -regionH[1]) dr[1] += region[1];
00060             dr[2] -= region[2];
00061         }else if(dr[2] < -regionH[2]){
00062             dr[1] += shearDisplacement;
00063             if(dr[1] >= regionH[1]) dr[1] -= region[1];
00064             dr[2] += region[2];
00065         }
00066     }

```



```

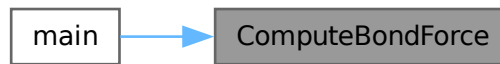
00067     rr = Sqr(dr[1]) + Sqr(dr[2]);
00068     r = sqrt(rr);
00069     rri = 1.0/rr;
00070     ri = 1.0/r;
00071     roi = 1.0/ro[n];
00072     strech = (r * roi - 1.0);
00073     uVal = 0.5 * kb[n] * ro[n] * Sqr(strech);
00074     fcVal = -kb[n] * strech * ri; //F = -Grad U
00075
00076     vr[1] = vx[atom1ID] - vx[atom2ID];
00077     vr[2] = vy[atom1ID] - vy[atom2ID];
00078     fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //node-node drag
00079
00080     //DampFlag = 1. LAMMPS version
00081     if(DampFlag == 1){
00082         nodeDragx[atom1ID] = fdVal * dr[1]; //node-node drag //Important change made on 03Apr2025.
00083         nodeDragy[atom1ID] = fdVal * dr[2]; //node-node drag //Adding the drag forces is wrong. Only add
the
00084         nodeDragx[atom2ID] = -fdVal * dr[1]; //node-node drag //total force
00085         nodeDragy[atom2ID] = -fdVal * dr[2]; //node-node drag
00086
00087         ax[atom1ID] += (fcVal + fdVal) * dr[1];
00088         ay[atom1ID] += (fcVal + fdVal) * dr[2];
00089         ax[atom2ID] += -(fcVal + fdVal) * dr[1];
00090         ay[atom2ID] += -(fcVal + fdVal) * dr[2];
00091     }
00092
00093     //DampFlag = 2. Suzanne notes version
00094     else if(DampFlag == 2){
00095         nodeDragx[atom1ID] = -gamman * vr[1]; //node-node drag
00096         nodeDragy[atom1ID] = -gamman * vr[2]; //node-node drag
00097         nodeDragx[atom2ID] = -(-gamman * vr[1]); //node-node drag
00098         nodeDragy[atom2ID] = -(-gamman * vr[2]); //node-node drag
00099
00100         ax[atom1ID] += (fcVal * dr[1] - gamman * vr[1]);
00101         ay[atom1ID] += (fcVal * dr[2] - gamman * vr[2]);
00102         ax[atom2ID] += -(fcVal * dr[1] - gamman * vr[1]);
00103         ay[atom2ID] += -(fcVal * dr[2] - gamman * vr[2]);
00104     }
00105
00106     //DampFlag = 3. Suzanne PRL, 130, 178203 (2023) version
00107     else if(DampFlag == 3){
00108         DeltaXijNew = dr[1];
00109         DeltaYijNew = dr[2];
00110
00111         if(stepCount == 0) { // First timestep
00112             DeltaXijOld[n] = DeltaXijNew;
00113             DeltaYijOld[n] = DeltaYijNew;
00114         }
00115
00116         DeltaXij = DeltaXijNew - DeltaXijOld[n];
00117         DeltaYij = DeltaYijNew - DeltaYijOld[n];
00118         DeltaVXij = DeltaXij / deltaT;
00119         DeltaVYij = DeltaYij / deltaT;
00120
00121         // Now update for the next timestep
00122         DeltaXijOld[n] = DeltaXijNew;
00123         DeltaYijOld[n] = DeltaYijNew;
00124
00125         nodeDragx[atom1ID] = -gamman * DeltaVXij; //node-node drag
00126         nodeDragy[atom1ID] = -gamman * DeltaVYij; //node-node drag
00127         nodeDragx[atom2ID] = -(-gamman * DeltaVXij); //node-node drag
00128         nodeDragy[atom2ID] = -(-gamman * DeltaVYij); //node-node drag
00129
00130         ax[atom1ID] += (fcVal * dr[1] - gamman * DeltaVXij);
00131         ay[atom1ID] += (fcVal * dr[2] - gamman * DeltaVYij);
00132         ax[atom2ID] += -(fcVal * dr[1] - gamman * DeltaVXij);
00133         ay[atom2ID] += -(fcVal * dr[2] - gamman * DeltaVYij);
00134     }
00135
00136
00137     BondLength[n] = r;
00138     BondEnergy[n] = uVal; //No 0.5 factor since it is the energy of the bond
00139     TotalBondEnergy += BondEnergy[n];
00140
00141     virSumBond += 0.5 * (fcVal + fdVal) * rr;
00142     virSumBondxx += 0.5 * (fcVal + fdVal) * dr[1] * dr[1];
00143     virSumBondyy += 0.5 * (fcVal + fdVal) * dr[2] * dr[2];
00144     virSumBondxy += 0.5 * (fcVal + fdVal) * dr[1] * dr[2];
00145 } }

```

References [atom1](#), [atom2](#), [ax](#), [ay](#), [BondEnergy](#), [BondLength](#), [DampFlag](#), [deltaT](#), [DeltaVXij](#), [DeltaVYij](#), [DeltaXij](#), [DeltaXijNew](#), [DeltaXijOld](#), [DeltaYij](#), [DeltaYijNew](#), [DeltaYijOld](#), [gamman](#), [kb](#), [nAtom](#), [nBond](#), [NDIM](#), [nodeDragx](#), [nodeDragy](#), [region](#), [regionH](#), [ro](#), [rx](#), [ry](#), [shearDisplacement](#), [Sqr](#), [stepCount](#), [strech](#), [TotalBondEnergy](#), [virSumBond](#), [virSumBondxx](#), [virSumBondxy](#), [virSumBondyy](#), [vx](#), and [vy](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.25 ComputeBondForce.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021 #include "ComputeBondForce.h"
00022
00023 #include<stdio.h>
00024 #include<math.h>
00025 #include<stdlib.h>
00026 #include"global.h"
00027
00028 void ComputeBondForce(){
00029     int n;
00030     double dr[NDIM+1], r, rr, ri, roi;
00031     double uVal, fcVal;
00032
00033     uVal = 0.0; TotalBondEnergy = 0.0;
00034     virSumBond = 0.0; virSumBondxx = 0.0; virSumBondyy = 0.0; virSumBondxy = 0.0;
00035
00036     double vr[NDIM+1], fdVal, rri;
00037
00038     for(n = 1 ; n <= nAtom ; n++){
00039         nodeDragx[n] = 0.0;
00040         nodeDragy[n] = 0.0;
00041     } //Important change made on 03Apr2025. Mention it in GitHub
00042
00043     int atom1ID, atom2ID;
00044
00045     for(n=1; n<=nBond; n++){
00046         rr = 0.0; rri = 0.0; fcVal = 0.0; fdVal = 0.0; stretch = 0.0;
00047         atom1ID = atom1[n];
00048         atom2ID = atom2[n];
00049
00050         dr[1] = rx[atom1ID] - rx[atom2ID];
00051         if(dr[1] >= regionH[1])
00052             dr[1] -= region[1];
00053         else if(dr[1] < -regionH[1])
00054             dr[1] += region[1];
00055
00056         dr[2] = ry[atom1ID] - ry[atom2ID];
00057         if(dr[2] >= regionH[2]){

```

```

00058     dr[1] -= shearDisplacement;
00059     if(dr[1] < -regionH[1]) dr[1] += region[1];
00060     dr[2] -= region[2];
00061 }else if(dr[2] < -regionH[2]){
00062     dr[1] += shearDisplacement;
00063     if(dr[1] >= regionH[1]) dr[1] -= region[1];
00064     dr[2] += region[2];
00065 }
00066
00067     rr = Sqr(dr[1]) + Sqr(dr[2]);
00068     r = sqrt(rr);
00069     rri = 1.0/rr;
00070     ri = 1.0/r;
00071     roi = 1.0/ro[n];
00072     stretch = (r * roi - 1.0);
00073     uVal = 0.5 * kb[n] * ro[n] * Sqr(strech);
00074     fcVal = -kb[n] * stretch * ri; //F = -Grad U
00075
00076     vr[1] = vx[atom1ID] - vx[atom2ID];
00077     vr[2] = vy[atom1ID] - vy[atom2ID];
00078     fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //node-node drag
00079
00080     //DampFlag = 1. LAMMPS version
00081     if(DampFlag == 1){
00082         nodeDragx[atom1ID] = fdVal * dr[1]; //node-node drag //Important change made on 03Apr2025.
00083         nodeDragy[atom1ID] = fdVal * dr[2]; //node-node drag //Adding the drag forces is wrong. Only add
the
00084         nodeDragx[atom2ID] = -fdVal * dr[1]; //node-node drag //total force
00085         nodeDragy[atom2ID] = -fdVal * dr[2]; //node-node drag
00086
00087         ax[atom1ID] += (fcVal + fdVal) * dr[1];
00088         ay[atom1ID] += (fcVal + fdVal) * dr[2];
00089         ax[atom2ID] += -(fcVal + fdVal) * dr[1];
00090         ay[atom2ID] += -(fcVal + fdVal) * dr[2];
00091     }
00092
00093     //DampFlag = 2. Suzanne notes version
00094     else if(DampFlag == 2){
00095         nodeDragx[atom1ID] = -gamman * vr[1]; //node-node drag
00096         nodeDragy[atom1ID] = -gamman * vr[2]; //node-node drag
00097         nodeDragx[atom2ID] = -(-gamman * vr[1]); //node-node drag
00098         nodeDragy[atom2ID] = -(-gamman * vr[2]); //node-node drag
00099
00100         ax[atom1ID] += (fcVal * dr[1] - gamman * vr[1]);
00101         ay[atom1ID] += (fcVal * dr[2] - gamman * vr[2]);
00102         ax[atom2ID] += -(fcVal * dr[1] - gamman * vr[1]);
00103         ay[atom2ID] += -(fcVal * dr[2] - gamman * vr[2]);
00104     }
00105
00106     //DampFlag = 3. Suzanne PRL, 130, 178203 (2023) version
00107     else if(DampFlag == 3){
00108         DeltaXijNew = dr[1];
00109         DeltaYijNew = dr[2];
00110
00111         if(stepCount == 0) { // First timestep
00112             DeltaXijOld[n] = DeltaXijNew;
00113             DeltaYijOld[n] = DeltaYijNew;
00114         }
00115
00116         DeltaXij = DeltaXijNew - DeltaXijOld[n];
00117         DeltaYij = DeltaYijNew - DeltaYijOld[n];
00118         DeltaVXij = DeltaXij / deltaT;
00119         DeltaVYij = DeltaYij / deltaT;
00120
00121         // Now update for the next timestep
00122         DeltaXijOld[n] = DeltaXijNew;
00123         DeltaYijOld[n] = DeltaYijNew;
00124
00125         nodeDragx[atom1ID] = -gamman * DeltaVXij; //node-node drag
00126         nodeDragy[atom1ID] = -gamman * DeltaVYij; //node-node drag
00127         nodeDragx[atom2ID] = -(-gamman * DeltaVXij); //node-node drag
00128         nodeDragy[atom2ID] = -(-gamman * DeltaVYij); //node-node drag
00129
00130         ax[atom1ID] += (fcVal * dr[1] - gamman * DeltaVXij);
00131         ay[atom1ID] += (fcVal * dr[2] - gamman * DeltaVYij);
00132         ax[atom2ID] += -(fcVal * dr[1] - gamman * DeltaVXij);
00133         ay[atom2ID] += -(fcVal * dr[2] - gamman * DeltaVYij);
00134     }
00135
00136
00137     BondLength[n] = r;
00138     BondEnergy[n] = uVal; //No 0.5 factor since it is the energy of the bond
00139     TotalBondEnergy += BondEnergy[n];
00140
00141     virSumBond += 0.5 * (fcVal + fdVal) * rr;
00142     virSumBondxx += 0.5 * (fcVal + fdVal) * dr[1] * dr[1];

```

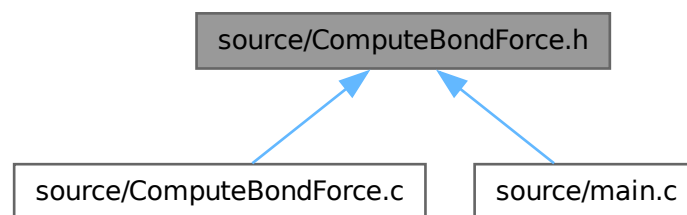
```

00143     virSumBondyy += 0.5 * (fcVal + fdVal) * dr[2] * dr[2];
00144     virSumBondxy += 0.5 * (fcVal + fdVal) * dr[1] * dr[2];
00145 } }

```

3.26 source/ComputeBondForce.h File Reference

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



Functions

- void [ComputeBondForce](#) ()

3.26.1 Function Documentation

3.26.1.1 ComputeBondForce()

```
void ComputeBondForce ( )
```

Definition at line 28 of file [ComputeBondForce.c](#).

```

00028     {
00029         int n;
00030         double dr[NDIM+1], r, rr, ri, roi;
00031         double uVal, fcVal;
00032
00033         uVal = 0.0; TotalBondEnergy = 0.0;
00034         virSumBond = 0.0; virSumBondxx = 0.0; virSumBondyy = 0.0; virSumBondxy = 0.0;
00035
00036         double vr[NDIM+1], fdVal, rri;
00037
00038         for(n = 1 ; n <= nAtom ; n ++){
00039             nodeDragx[n] = 0.0;
00040             nodeDragy[n] = 0.0;
00041             } //Important change made on 03Apr2025. Mention it in GitHub
00042
00043         int atom1ID, atom2ID;
00044
00045         for(n=1; n<=nBond; n++){
00046             rr = 0.0; rri = 0.0; fcVal = 0.0; fdVal = 0.0; stretch = 0.0;
00047             atom1ID = atom1[n];
00048             atom2ID = atom2[n];
00049
00050             dr[1] = rx[atom1ID] - rx[atom2ID];
00051             if(dr[1] >= regionH[1])
00052                 dr[1] -= region[1];
00053             else if(dr[1] < -regionH[1])
00054                 dr[1] += region[1];
00055

```

```

00056     dr[2] = ry[atom1ID] - ry[atom2ID];
00057     if(dr[2] >= regionH[2]){
00058         dr[1] -= shearDisplacement;
00059         if(dr[1] < -regionH[1]) dr[1] += region[1];
00060         dr[2] -= region[2];
00061     }else if(dr[2] < -regionH[2]){
00062         dr[1] += shearDisplacement;
00063         if(dr[1] >= regionH[1]) dr[1] -= region[1];
00064         dr[2] += region[2];
00065     }
00066
00067     rr = Sqr(dr[1]) + Sqr(dr[2]);
00068     r = sqrt(rr);
00069     rri = 1.0/rr;
00070     ri = 1.0/r;
00071     roi = 1.0/ro[n];
00072     strech = (r * roi - 1.0);
00073     uVal = 0.5 * kb[n] * ro[n] * Sqr(strech);
00074     fcVal = -kb[n] * strech * ri; //F = -Grad U
00075
00076     vr[1] = vx[atom1ID] - vx[atom2ID];
00077     vr[2] = vy[atom1ID] - vy[atom2ID];
00078     fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //node-node drag
00079
00080     //DampFlag = 1. LAMMPS version
00081     if(DampFlag == 1){
00082         nodeDragx[atom1ID] = fdVal * dr[1]; //node-node drag //Important change made on 03Apr2025.
00083         nodeDragy[atom1ID] = fdVal * dr[2]; //node-node drag //Adding the drag forces is wrong. Only add
00084         the
00085         nodeDragx[atom2ID] = -fdVal * dr[1]; //node-node drag //total force
00086         nodeDragy[atom2ID] = -fdVal * dr[2]; //node-node drag
00087
00088         ax[atom1ID] += (fcVal + fdVal) * dr[1];
00089         ay[atom1ID] += (fcVal + fdVal) * dr[2];
00090         ax[atom2ID] += -(fcVal + fdVal) * dr[1];
00091         ay[atom2ID] += -(fcVal + fdVal) * dr[2];
00092     }
00093
00094     //DampFlag = 2. Suzanne notes version
00095     else if(DampFlag == 2){
00096         nodeDragx[atom1ID] = -gamman * vr[1]; //node-node drag
00097         nodeDragy[atom1ID] = -gamman * vr[2]; //node-node drag
00098         nodeDragx[atom2ID] = -(-gamman * vr[1]); //node-node drag
00099         nodeDragy[atom2ID] = -(-gamman * vr[2]); //node-node drag
00100
00101         ax[atom1ID] += (fcVal * dr[1] - gamman * vr[1]);
00102         ay[atom1ID] += (fcVal * dr[2] - gamman * vr[2]);
00103         ax[atom2ID] += -(fcVal * dr[1] - gamman * vr[1]);
00104         ay[atom2ID] += -(fcVal * dr[2] - gamman * vr[2]);
00105     }
00106
00107     //DampFlag = 3. Suzanne PRL, 130, 178203 (2023) version
00108     else if(DampFlag == 3){
00109         DeltaXijNew = dr[1];
00110         DeltaYijNew = dr[2];
00111
00112         if(stepCount == 0) { // First timestep
00113             DeltaXijOld[n] = DeltaXijNew;
00114             DeltaYijOld[n] = DeltaYijNew;
00115         }
00116
00117         DeltaXij = DeltaXijNew - DeltaXijOld[n];
00118         DeltaYij = DeltaYijNew - DeltaYijOld[n];
00119         DeltaVXij = DeltaXij / deltaT;
00120         DeltaVYij = DeltaYij / deltaT;
00121
00122         // Now update for the next timestep
00123         DeltaXijOld[n] = DeltaXijNew;
00124         DeltaYijOld[n] = DeltaYijNew;
00125
00126         nodeDragx[atom1ID] = -gamman * DeltaVXij; //node-node drag
00127         nodeDragy[atom1ID] = -gamman * DeltaVYij; //node-node drag
00128         nodeDragx[atom2ID] = -(-gamman * DeltaVXij); //node-node drag
00129         nodeDragy[atom2ID] = -(-gamman * DeltaVYij); //node-node drag
00130
00131         ax[atom1ID] += (fcVal * dr[1] - gamman * DeltaVXij);
00132         ay[atom1ID] += (fcVal * dr[2] - gamman * DeltaVYij);
00133         ax[atom2ID] += -(fcVal * dr[1] - gamman * DeltaVXij);
00134         ay[atom2ID] += -(fcVal * dr[2] - gamman * DeltaVYij);
00135     }
00136
00137     BondLength[n] = r;
00138     BondEnergy[n] = uVal; //No 0.5 factor since it is the energy of the bond
00139     TotalBondEnergy += BondEnergy[n];
00140

```

```

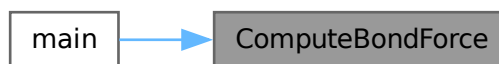
00141     virSumBond += 0.5 * (fcVal + fdVal) * rr;
00142     virSumBondxx += 0.5 * (fcVal + fdVal) * dr[1] * dr[1];
00143     virSumBondyy += 0.5 * (fcVal + fdVal) * dr[2] * dr[2];
00144     virSumBondxy += 0.5 * (fcVal + fdVal) * dr[1] * dr[2];
00145 } }

```

References [atom1](#), [atom2](#), [ax](#), [ay](#), [BondEnergy](#), [BondLength](#), [DampFlag](#), [deltaT](#), [DeltaVXij](#), [DeltaVYij](#), [DeltaXij](#), [DeltaXijNew](#), [DeltaXijOld](#), [DeltaYij](#), [DeltaYijNew](#), [DeltaYijOld](#), [gamman](#), [kb](#), [nAtom](#), [nBond](#), [NDIM](#), [nodeDragx](#), [nodeDragy](#), [region](#), [regionH](#), [ro](#), [rx](#), [ry](#), [shearDisplacement](#), [Sqr](#), [stepCount](#), [strech](#), [TotalBondEnergy](#), [virSumBond](#), [virSumBondxx](#), [virSumBondxy](#), [virSumBondyy](#), [vx](#), and [vy](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.27 ComputeBondForce.h

[Go to the documentation of this file.](#)

```

00001 #ifndef COMPUTE_BOND_FORCE_H
00002 #define COMPUTE_BOND_FORCE_H
00003
00004 void ComputeBondForce();
00005
00006 #endif
00007

```

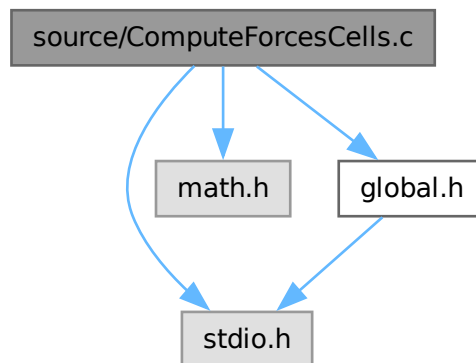
3.28 source/ComputeForcesCells.c File Reference

```

#include <stdio.h>
#include <math.h>
#include "global.h"

```

Include dependency graph for ComputeForcesCells.c:



Functions

- void [ComputeForcesCells](#) ()

3.28.1 Function Documentation

3.28.1.1 ComputeForcesCells()

```
void ComputeForcesCells ( )
```

Definition at line 25 of file [ComputeForcesCells.c](#).

```

00025     {
00026     double dr[NDIM+1], invWid[NDIM+1], shift[NDIM+1], f, fcVal, rr, ri, r, uVal;
00027     int c, I, J, m1, m1X, m1Y, m2, m2X, m2Y, n, offset;
00028     int ioFX[] = {0, 0, 1, 1, 0, -1, -1, -1, 0, 1},
00029         ioFY[] = {0, 0, 0, 1, 1, 1, 0, -1, -1, -1};
00030
00031     invWid[1] = cells[1]/region[1];
00032     invWid[2] = cells[2]/region[2];
00033
00034     for(n = nAtom+1; n <= nAtom+cells[1]*cells[2] ; n++)
00035         cellList[n] = 0;
00036
00037     for(n = 1 ; n <= nAtom ; n++){
00038         c = ((int)((ry[n] + regionH[2])*invWid[2]))*cells[1] + (int)((rx[n]+regionH[1])*invWid[1]) +
nAtom+ 1;
00039         cellList[n] = cellList[c];
00040         cellList[c] = n;
00041     }
00042
00043     for(n = 1 ; n <= nAtom ; n++){
00044         ax[n] = 0.;
00045         ay[n] = 0.;
00046     }
00047
00048     uSum = 0.0 ;
00049     virSum = 0.0;
00050     rfAtom = 0.0;
00051     RadiusIJ = 0.0;
00052
00053     gamman = 1.0;
00054     double vr[NDIM+1], fd, fdVal, rrinv;
00055     rrinv = 0.0;
00056     fd = 0.0;

```

```

00057     fdVal = 0.0;
00058
00059     int start = 1 + rank*(cells[2]/size);
00060     int end = (rank+1)*(cells[2]/size);
00061
00062     for(m1Y = start ; m1Y <= end ; m1Y ++){
00063         for(m1X = 1 ; m1X <= cells[1] ; m1X ++){
00064             m1 = (m1Y-1) * cells[1] + m1X + nAtom;
00065             for(offset = 1 ; offset <= 9 ; offset ++){
00066                 m2X = m1X + ioFX[offset]; shift[1] = 0.;
00067                 if(m2X > cells[1]){
00068                     m2X = 1; shift[1] = region[1];
00069                 }else if(m2X == 0){
00070                     m2X = cells[1]; shift[1] = -region[1];
00071                 }
00072                 m2Y = m1Y + ioFY[offset]; shift[2] = 0.;
00073                 if(m2Y > cells[2]){
00074                     m2Y = 1; shift[2] = region[2];
00075                 }else if(m2Y == 0){
00076                     m2Y = cells[2]; shift[2] = -region[2];
00077                 }
00078                 m2 = (m2Y-1)*cells[1] + m2X + nAtom;
00079                 I = cellList[m1];
00080                 while(I > 0){
00081                     J = cellList[m2];
00082                     while(J > 0){
00083                         if(m1 == m2 && J != I && (atomRadius[I] > 0. && atomRadius[J] > 0.)){
00084                             dr[1] = rx[I] - rx[J] - shift[1];
00085                             dr[2] = ry[I] - ry[J] - shift[2];
00086                             rr = Sqr(dr[1]) + Sqr(dr[2]);
00087                             RadiusIJ = atomRadius[I] + atomRadius[J];
00088                             SqrRadiusIJ = Sqr(RadiusIJ);
00089                             if(rr < SqrRadiusIJ){
00090                                 r = sqrt(rr);
00091                                 ri = 1.0/r;
00092                                 rrinv = 1.0/rr;
00093                                 vr[1] = vx[I] - vx[J];
00094                                 vr[2] = vy[I] - vy[J];
00095                                 RadiusIJInv = 1.0/RadiusIJ;
00096                                 uVal = Sqr(1.0 - r * RadiusIJInv);
00097                                 fcVal = 2.0 * RadiusIJInv * (1.0 - r * RadiusIJInv) * ri;
00098                                 fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag
00099
00100                                 f = fcVal * dr[1];
00101                                 fd = fdVal * dr[1];
00102                                 ax[I] += (f + fd);
00103                                 discDragx[I] += fd; //disc-disc drag
00104
00105                                 f = fcVal * dr[2];
00106                                 fd = fdVal * dr[2];
00107                                 ay[I] += (f + fd);
00108                                 discDragy[I] += fd; //disc-disc drag
00109
00110                                 uSum += 0.5 * uVal;
00111                                 virSum += 0.5 * fcVal * rr;
00112                                 rfAtom += 0.5 * dr[1] * fcVal * dr[2];
00113                             }
00114                         }else if(m1 != m2 && (atomRadius[I] > 0. && atomRadius[J] > 0.)){
00115                             dr[1] = rx[I] - rx[J] - shift[1];
00116                             dr[2] = ry[I] - ry[J] - shift[2];
00117                             rr = Sqr(dr[1]) + Sqr(dr[2]);
00118                             RadiusIJ = atomRadius[I] + atomRadius[J];
00119                             SqrRadiusIJ = Sqr(RadiusIJ);
00120                             if(rr < SqrRadiusIJ){
00121                                 r = sqrt(rr);
00122                                 ri = 1.0/r;
00123                                 rrinv = 1.0/r;
00124                                 vr[1] = vx[I] - vx[J];
00125                                 vr[2] = vy[I] - vy[J];
00126                                 RadiusIJInv = 1.0/RadiusIJ;
00127                                 uVal = Sqr(1.0 - r * RadiusIJInv);
00128                                 fcVal = 2.0 * RadiusIJInv * (1.0 - r * RadiusIJInv) * ri;
00129                                 fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag
00130
00131                                 f = fcVal * dr[1];
00132                                 fd = fdVal * dr[1];
00133                                 ax[I] += (f + fd);
00134                                 discDragx[I] += fd; //disc-disc drag
00135
00136                                 f = fcVal * dr[2];
00137                                 fd = fdVal * dr[2];
00138                                 ay[I] += (f + fd);
00139                                 discDragy[I] += fd; //disc-disc drag
00140
00141                                 uSum += 0.5 * uVal;
00142                                 virSum += 0.5 * fcVal * rr;
00143                                 rfAtom += 0.5 * dr[1] * fcVal * dr[2];

```



```

00144     }
00145     }
00146     J = cellList[J];
00147     }
00148     I = cellList[I];
00149     }
00150     }
00151     }
00152     }
00153 }

```

References [atomRadius](#), [ax](#), [ay](#), [cellList](#), [cells](#), [discDragx](#), [discDragy](#), [gamman](#), [nAtom](#), [NDIM](#), [RadiusIJ](#), [RadiusIJInv](#), [rank](#), [region](#), [regionH](#), [rfAtom](#), [rx](#), [ry](#), [size](#), [Sqr](#), [SqrRadiusIJ](#), [uSum](#), [virSum](#), [vx](#), and [vy](#).

3.29 ComputeForcesCells.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021 #include<stdio.h>
00022 #include<math.h>
00023 #include"global.h"
00024
00025 void ComputeForcesCells(){
00026     double dr[NDIM+1], invWid[NDIM+1], shift[NDIM+1], f, fcVal, rr, ri, r, uVal;
00027     int c, I, J, m1, m1X, m1Y, m2, m2X, m2Y, n, offset;
00028     int ioFX[] = {0, 0, 1, 1, 0, -1, -1, -1, 0, 1},
00029         ioFY[] = {0, 0, 0, 1, 1, 1, 0, -1, -1, -1};
00030
00031     invWid[1] = cells[1]/region[1];
00032     invWid[2] = cells[2]/region[2];
00033
00034     for(n = nAtom+1; n <= nAtom+cells[1]*cells[2] ; n++){
00035         cellList[n] = 0;
00036
00037         for(n = 1 ; n <= nAtom ; n++){
00038             c = ((int)((ry[n] + regionH[2])*invWid[2]))*cells[1] + (int)((rx[n]+regionH[1])*invWid[1]) +
nAtom+ 1;
00039             cellList[n] = cellList[c];
00040             cellList[c] = n;
00041         }
00042
00043         for(n = 1 ; n <= nAtom ; n++){
00044             ax[n] = 0.;
00045             ay[n] = 0.;
00046         }
00047
00048         uSum = 0.0 ;
00049         virSum = 0.0;
00050         rfAtom = 0.0;
00051         RadiusIJ = 0.0;
00052
00053         gamman = 1.0;
00054         double vr[NDIM+1], fd, fdVal, rrinv;
00055         rrinv = 0.0;
00056         fd = 0.0;
00057         fdVal = 0.0;
00058
00059         int start = 1 + rank*(cells[2]/size);
00060         int end = (rank+1)*(cells[2]/size);
00061
00062         for(m1Y = start ; m1Y <= end ; m1Y++){

```

```

00063     for(m1X = 1 ; m1X <= cells[1] ; m1X++){
00064         m1 = (m1Y-1) * cells[1] + m1X + nAtom;
00065         for(offset = 1 ; offset <= 9 ; offset++){
00066             m2X = m1X + ioFX[offset]; shift[1] = 0.;
00067             if(m2X > cells[1]){
00068                 m2X = 1; shift[1] = region[1];
00069             }else if(m2X == 0){
00070                 m2X = cells[1]; shift[1] = -region[1];
00071             }
00072             m2Y = m1Y + ioFY[offset]; shift[2] = 0.;
00073             if(m2Y > cells[2]){
00074                 m2Y = 1; shift[2] = region[2];
00075             }else if(m2Y == 0){
00076                 m2Y = cells[2]; shift[2] = -region[2];
00077             }
00078             m2 = (m2Y-1)*cells[1] + m2X + nAtom;
00079             I = cellList[m1];
00080             while(I > 0){
00081                 J = cellList[m2];
00082                 while(J > 0){
00083                     if(m1 == m2 && J != I && (atomRadius[I] > 0. && atomRadius[J] > 0.)){
00084                         dr[1] = rx[I] - rx[J] - shift[1];
00085                         dr[2] = ry[I] - ry[J] - shift[2];
00086                         rr = Sqr(dr[1]) + Sqr(dr[2]);
00087                         RadiusIJ = atomRadius[I] + atomRadius[J];
00088                         SqrRadiusIJ = Sqr(RadiusIJ);
00089                         if(rr < SqrRadiusIJ){
00090                             r = sqrt(rr);
00091                             ri = 1.0/r;
00092                             rrinv = 1.0/rr;
00093                             vr[1] = vx[I] - vx[J];
00094                             vr[2] = vy[I] - vy[J];
00095                             RadiusIJInv = 1.0/RadiusIJ;
00096                             uVal = Sqr(1.0 - r * RadiusIJInv);
00097                             fcVal = 2.0 * RadiusIJInv * (1.0 - r * RadiusIJInv) * ri;
00098                             fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag
00099
00100                             f = fcVal * dr[1];
00101                             fd = fdVal * dr[1];
00102                             ax[I] += (f + fd);
00103                             discDragx[I] += fd; //disc-disc drag
00104
00105                             f = fcVal * dr[2];
00106                             fd = fdVal * dr[2];
00107                             ay[I] += (f + fd);
00108                             discDragy[I] += fd; //disc-disc drag
00109
00110                             uSum += 0.5 * uVal;
00111                             virSum += 0.5 * fcVal * rr;
00112                             rfAtom += 0.5 * dr[1] * fcVal * dr[2];
00113                         }
00114                     }else if(m1 != m2 && (atomRadius[I] > 0. && atomRadius[J] > 0.)){
00115                         dr[1] = rx[I] - rx[J] - shift[1];
00116                         dr[2] = ry[I] - ry[J] - shift[2];
00117                         rr = Sqr(dr[1]) + Sqr(dr[2]);
00118                         RadiusIJ = atomRadius[I] + atomRadius[J];
00119                         SqrRadiusIJ = Sqr(RadiusIJ);
00120                         if(rr < SqrRadiusIJ){
00121                             r = sqrt(rr);
00122                             ri = 1.0/r;
00123                             rrinv = 1.0/r;
00124                             vr[1] = vx[I] - vx[J];
00125                             vr[2] = vy[I] - vy[J];
00126                             RadiusIJInv = 1.0/RadiusIJ;
00127                             uVal = Sqr(1.0 - r * RadiusIJInv);
00128                             fcVal = 2.0 * RadiusIJInv * (1.0 - r * RadiusIJInv) * ri;
00129                             fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag
00130
00131                             f = fcVal * dr[1];
00132                             fd = fdVal * dr[1];
00133                             ax[I] += (f + fd);
00134                             discDragx[I] += fd; //disc-disc drag
00135
00136                             f = fcVal * dr[2];
00137                             fd = fdVal * dr[2];
00138                             ay[I] += (f + fd);
00139                             discDragy[I] += fd; //disc-disc drag
00140
00141                             uSum += 0.5 * uVal;
00142                             virSum += 0.5 * fcVal * rr;
00143                             rfAtom += 0.5 * dr[1] * fcVal * dr[2];
00144                         }
00145                     }
00146                     J = cellList[J];
00147                 }
00148             I = cellList[I];
00149         }

```

```

00150     }
00151 }
00152 }
00153 }

```

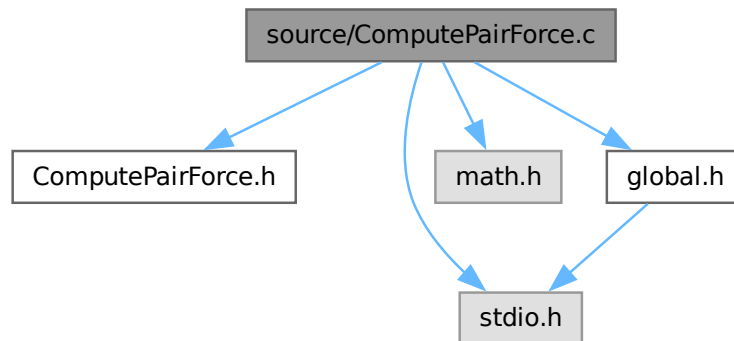
3.30 source/ComputePairForce.c File Reference

```

#include "ComputePairForce.h"
#include <stdio.h>
#include <math.h>
#include "global.h"

```

Include dependency graph for ComputePairForce.c:



Functions

- void [ComputePairForce](#) (int normFlag)

3.30.1 Function Documentation

3.30.1.1 ComputePairForce()

```

void ComputePairForce (
    int normFlag )

```

Definition at line 27 of file [ComputePairForce.c](#).

```

00027     {
00028 double dr[NDIM+1], fcVal, rr, ri, r, uVal;
00029 int n, i, j;
00030 uVal = 0.0; uSumPair = 0.0 ;
00031 virSumPair = 0.0; virSumPairxx = 0.0; virSumPairyy = 0.0; virSumPairxy = 0.0;
00032
00033 for(n = 1 ; n <= nAtom ; n ++){
00034 ax[n] = 0.0;
00035 ay[n] = 0.0;
00036 discDragx[n] = 0.0;
00037 discDragy[n] = 0.0;
00038 }
00039 for(n = 1; n <= nPairTotal; n ++){
00040 PairID[n] = 0;

```

```

00041 Pairatom1[n] = 0;
00042 Pairatom2[n] = 0;
00043 PairXij[n] = 0.0;
00044 PairYij[n] = 0.0;
00045 }
00046
00047
00048 Kn = 1.0;
00049 double vr[NDIM+1], fdVal, rri;
00050 nPairActive = 0;
00051 double meff;
00052 meff = 0.0;
00053 int atomIDi, atomIDj;
00054 //int processThisPair = 1;
00055
00056 for(i=1;i<=nAtomInterface;i++){
00057   for(j=i+1;j<=nAtomInterface;j++){
00058     atomIDi = atomIDInterface[i];
00059     atomIDj = atomIDInterface[j];
00060     if (isBonded[atomIDi][atomIDj] == 0) { //To exclude pair interaction between bonded atoms
00061       rr = 0.0; rri = 0.0; fcVal = 0.0; fdVal = 0.0; strech = 0.0;
00062       RadiusIJ = 0.0;
00063
00064       dr[1] = rx[atomIDi] - rx[atomIDj];
00065       if(dr[1] >= regionH[1]){
00066         dr[1] -= region[1];
00067       }else if(dr[1] < -regionH[1]){
00068         dr[1] += region[1];
00069       }
00070       dr[2] = ry[atomIDi] - ry[atomIDj];
00071       if(dr[2] >= regionH[2]){
00072         dr[1] -= shearDisplacement;
00073         if(dr[1] < -regionH[1]) dr[1] += region[1];
00074         dr[2] -= region[2];
00075       }else if(dr[2] < -regionH[2]){
00076         dr[1] += shearDisplacement;
00077         if(dr[1] >= regionH[1]) dr[1] -= region[1];
00078         dr[2] += region[2];
00079       }
00080
00081       rr = Sqr(dr[1]) + Sqr(dr[2]);
00082       RadiusIJ = atomRadius[atomIDi] + atomRadius[atomIDj];
00083       SqrRadiusIJ = Sqr(RadiusIJ);
00084       if(rr < SqrRadiusIJ){
00085         r = sqrt(rr);
00086         ri = 1.0/r;
00087         rri = 1.0/rr;
00088         RadiusIJInv = 1.0/RadiusIJ;
00089         strech = (RadiusIJ - r);
00090         uVal = 0.5 * Kn * Sqr(strech);
00091
00092         //NormFlag
00093         if(normFlag == 1){
00094           strech = strech * RadiusIJInv;
00095           uVal = 0.5 * Kn * RadiusIJ * Sqr(strech);
00096         }
00097
00098         fcVal = Kn * strech * ri;
00099         vr[1] = vx[atomIDi] - vx[atomIDj];
00100         vr[2] = vy[atomIDi] - vy[atomIDj];
00101
00102         nPairActive++;
00103         PairID[nPairActive] = nPairActive;
00104         Pairatom1[nPairActive] = atomIDi;
00105         Pairatom2[nPairActive] = atomIDj;
00106         PairXij[nPairActive] = dr[1];
00107         PairYij[nPairActive] = dr[2];
00108
00109         //DampFlag = 1
00110         if(DampFlag == 1){
00111           meff = (atomMass[atomIDi]*atomMass[atomIDj])/(atomMass[atomIDi] + atomMass[atomIDj]);
00112           fdVal = -gamman * meff * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //disc-disc drag
00113
00114           discDragx[atomIDi] = fdVal * dr[1]; //disc-disc drag
00115           discDragy[atomIDi] = fdVal * dr[2]; //disc-disc drag
00116           discDragx[atomIDj] = -fdVal * dr[1]; //disc-disc drag
00117           discDragy[atomIDj] = -fdVal * dr[2]; //disc-disc drag
00118
00119           discDragx[nPairActive] = discDragx[atomIDi];
00120           discDragy[nPairActive] = discDragy[atomIDi];
00121
00122
00123           ax[atomIDi] += (fcVal + fdVal) * dr[1];
00124           ay[atomIDi] += (fcVal + fdVal) * dr[2];
00125           ax[atomIDj] += -(fcVal + fdVal) * dr[1];
00126           ay[atomIDj] += -(fcVal + fdVal) * dr[2];
00127         }

```

```

00128
00129 //DampFlag = 2
00130 else if(DampFlag == 2){
00131     discDragx[atomIDi] = -gamman * vr[1]; //disc-disc drag
00132     discDragy[atomIDi] = -gamman * vr[2]; //disc-disc drag
00133     discDragx[atomIDj] = -(-gamman * vr[1]); //disc-disc drag
00134     discDragy[atomIDj] = -(-gamman * vr[2]); //disc-disc drag
00135
00136     discDragx[nPairActive] = discDragx[atomIDi];
00137     discDragy[nPairActive] = discDragy[atomIDi];
00138
00139
00140     ax[atomIDi] += (fcVal * dr[1] - gamman * vr[1]);
00141     ay[atomIDi] += (fcVal * dr[2] - gamman * vr[2]);
00142     ax[atomIDj] += -(fcVal * dr[1] - gamman * vr[1]);
00143     ay[atomIDj] += -(fcVal * dr[2] - gamman * vr[2]);
00144 }
00145
00146 //DampFlag = 3. Suzanne PRL, 130, 178203 (2023) version
00147 else if(DampFlag == 3){
00148     //Track compression velocity
00149     DeltaXijNew = dr[1];
00150     DeltaYijNew = dr[2];
00151     if(stepCount == 0) { // Initialization step
00152         DeltaXijOldPair[atomIDi][atomIDj] = DeltaXijNew;
00153         DeltaYijOldPair[atomIDi][atomIDj] = DeltaYijNew;
00154     }
00155
00156     DeltaXij = DeltaXijNew - DeltaXijOldPair[atomIDi][atomIDj];
00157     DeltaYij = DeltaYijNew - DeltaYijOldPair[atomIDi][atomIDj];
00158     DeltaVXij = DeltaXij / deltaT;
00159     DeltaVYij = DeltaYij / deltaT;
00160
00161     // Update history for next step
00162     DeltaXijOldPair[atomIDi][atomIDj] = DeltaXijNew;
00163     DeltaYijOldPair[atomIDi][atomIDj] = DeltaYijNew;
00164
00165     discDragx[atomIDi] = -gamman * DeltaVXij; //disc-disc drag
00166     discDragy[atomIDi] = -gamman * DeltaVYij; //disc-disc drag
00167     discDragx[atomIDj] = -(-gamman * DeltaVXij); //disc-disc drag
00168     discDragy[atomIDj] = -(-gamman * DeltaVYij); //disc-disc drag
00169
00170     discDragx[nPairActive] = discDragx[atomIDi];
00171     discDragy[nPairActive] = discDragy[atomIDi];
00172
00173     ax[atomIDi] += (fcVal * dr[1] - gamman * DeltaVXij);
00174     ay[atomIDi] += (fcVal * dr[2] - gamman * DeltaVYij);
00175     ax[atomIDj] += -(fcVal * dr[1] - gamman * DeltaVXij);
00176     ay[atomIDj] += -(fcVal * dr[2] - gamman * DeltaVYij);
00177 }
00178
00179 //In the following, for stress/virial term (fcVal + fdVal) is used since the total pair force =
Hookean Interaction + relative velocity drag
00180 uSumPair += 0.5 * uVal;
00181 virSumPair += 0.5 * (fcVal + fdVal) * rr;
00182 virSumPairxx += 0.5 * (fcVal + fdVal) * dr[1] * dr[1];
00183 virSumPairyy += 0.5 * (fcVal + fdVal) * dr[2] * dr[2];
00184 virSumPairxy += 0.5 * (fcVal + fdVal) * dr[1] * dr[2];
00185 }
00186 else { //Resetting the distance between two discs when they are not in contact
00187     DeltaXijOldPair[atomIDi][atomIDj] = 0.0;
00188     DeltaYijOldPair[atomIDi][atomIDj] = 0.0;
00189     DeltaXijOldPair[atomIDj][atomIDi] = 0.0;
00190     DeltaYijOldPair[atomIDj][atomIDi] = 0.0;
00191 }
00192 }
00193 }
00194 }
00195 }

```

References [atomIDInterface](#), [atomMass](#), [atomRadius](#), [ax](#), [ay](#), [DampFlag](#), [deltaT](#), [DeltaVXij](#), [DeltaVYij](#), [DeltaXij](#), [DeltaXijNew](#), [DeltaXijOldPair](#), [DeltaYij](#), [DeltaYijNew](#), [DeltaYijOldPair](#), [discDragx](#), [discDragy](#), [gamman](#), [isBonded](#), [Kn](#), [nAtom](#), [nAtomInterface](#), [NDIM](#), [nPairActive](#), [nPairTotal](#), [Pairatom1](#), [Pairatom2](#), [PairID](#), [PairXij](#), [PairYij](#), [RadiusIJ](#), [RadiusIJInv](#), [region](#), [regionH](#), [rx](#), [ry](#), [shearDisplacement](#), [Sqr](#), [SqrRadiusIJ](#), [stepCount](#), [strech](#), [uSumPair](#), [virSumPair](#), [virSumPairxx](#), [virSumPairxy](#), [virSumPairyy](#), [vx](#), and [vy](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.31 ComputePairForce.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021 #include "ComputePairForce.h"
00022
00023 #include<stdio.h>
00024 #include<math.h>
00025 #include"global.h"
00026
00027 void ComputePairForce(int normFlag){
00028 double dr[NDIM+1], fcVal, rr, ri, r, uVal;
00029 int n, i,j;
00030 uVal = 0.0; uSumPair = 0.0 ;
00031 virSumPair = 0.0; virSumPairxx = 0.0; virSumPairyy = 0.0; virSumPairxy = 0.0;
00032
00033 for(n = 1 ; n <= nAtom ; n ++){
00034 ax[n] = 0.0;
00035 ay[n] = 0.0;
00036 discDragx[n] = 0.0;
00037 discDragy[n] = 0.0;
00038 }
00039 for(n = 1; n <= nPairTotal; n ++){
00040 PairID[n] = 0;
00041 Pairatom1[n] = 0;
00042 Pairatom2[n] = 0;
00043 PairXij[n] = 0.0;
00044 PairYij[n] = 0.0;
00045 }
00046
00047 Kn = 1.0;
00048 double vr[NDIM+1], fdVal, rri;
00049 nPairActive = 0;
00050 double meff;
00051 meff = 0.0;
00052 int atomIDi, atomIDj;
00053 //int processThisPair = 1;
00054
00055 for(i=1;i<=nAtomInterface;i++){
00056 for(j=i+1;j<=nAtomInterface;j++){
00057 atomIDi = atomIDInterface[i];
00058 atomIDj = atomIDInterface[j];
00059 if (isBonded[atomIDi][atomIDj] == 0) { //To exclude pair interaction between bonded atoms

```

```

00061 rr = 0.0; rri = 0.0; fcVal = 0.0; fdVal = 0.0; strech = 0.0;
00062 RadiusIJ = 0.0;
00063
00064 dr[1] = rx[atomIDi] - rx[atomIDj];
00065 if(dr[1] >= regionH[1])
00066     dr[1] -= region[1];
00067 else if(dr[1] < -regionH[1])
00068     dr[1] += region[1];
00069
00070 dr[2] = ry[atomIDi] - ry[atomIDj];
00071 if(dr[2] >= regionH[2]){
00072     dr[1] -= shearDisplacement;
00073     if(dr[1] < -regionH[1]) dr[1] += region[1];
00074     dr[2] -= region[2];
00075 }else if(dr[2] < -regionH[2]){
00076     dr[1] += shearDisplacement;
00077     if(dr[1] >= regionH[1]) dr[1] -= region[1];
00078     dr[2] += region[2];
00079 }
00080
00081 rr = Sqr(dr[1]) + Sqr(dr[2]);
00082 RadiusIJ = atomRadius[atomIDi] + atomRadius[atomIDj];
00083 SqrRadiusIJ = Sqr(RadiusIJ);
00084 if(rr < SqrRadiusIJ){
00085     r = sqrt(rr);
00086     ri = 1.0/r;
00087     rri = 1.0/rr;
00088     RadiusIJInv = 1.0/RadiusIJ;
00089     strech = (RadiusIJ - r);
00090     uVal = 0.5 * Kn * Sqr(strech);
00091
00092     //NormFlag
00093     if(normFlag == 1){
00094         strech = strech * RadiusIJInv;
00095         uVal = 0.5 * Kn * RadiusIJ * Sqr(strech);
00096     }
00097
00098     fcVal = Kn * strech * ri;
00099     vr[1] = vx[atomIDi] - vx[atomIDj];
00100     vr[2] = vy[atomIDi] - vy[atomIDj];
00101
00102     nPairActive++;
00103     PairID[nPairActive] = nPairActive;
00104     Pairatom1[nPairActive] = atomIDi;
00105     Pairatom2[nPairActive] = atomIDj;
00106     PairXi[nPairActive] = dr[1];
00107     PairYi[nPairActive] = dr[2];
00108
00109     //DampFlag = 1
00110     if(DampFlag == 1){
00111         meff = (atomMass[atomIDi]*atomMass[atomIDj])/(atomMass[atomIDi] + atomMass[atomIDj]);
00112         fdVal = -gamman * meff * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //disc-disc drag
00113
00114         discDragx[atomIDi] = fdVal * dr[1]; //disc-disc drag
00115         discDragy[atomIDi] = fdVal * dr[2]; //disc-disc drag
00116         discDragx[atomIDj] = -fdVal * dr[1]; //disc-disc drag
00117         discDragy[atomIDj] = -fdVal * dr[2]; //disc-disc drag
00118
00119         discDragx[nPairActive] = discDragx[atomIDi];
00120         discDragy[nPairActive] = discDragy[atomIDi];
00121
00122         ax[atomIDi] += (fcVal + fdVal) * dr[1];
00123         ay[atomIDi] += (fcVal + fdVal) * dr[2];
00124         ax[atomIDj] += -(fcVal + fdVal) * dr[1];
00125         ay[atomIDj] += -(fcVal + fdVal) * dr[2];
00126     }
00127 }
00128
00129 //DampFlag = 2
00130 else if(DampFlag == 2){
00131     discDragx[atomIDi] = -gamman * vr[1]; //disc-disc drag
00132     discDragy[atomIDi] = -gamman * vr[2]; //disc-disc drag
00133     discDragx[atomIDj] = -(-gamman * vr[1]); //disc-disc drag
00134     discDragy[atomIDj] = -(-gamman * vr[2]); //disc-disc drag
00135
00136     discDragx[nPairActive] = discDragx[atomIDi];
00137     discDragy[nPairActive] = discDragy[atomIDi];
00138
00139     ax[atomIDi] += (fcVal * dr[1] - gamman * vr[1]);
00140     ay[atomIDi] += (fcVal * dr[2] - gamman * vr[2]);
00141     ax[atomIDj] += -(fcVal * dr[1] - gamman * vr[1]);
00142     ay[atomIDj] += -(fcVal * dr[2] - gamman * vr[2]);
00143 }
00144
00145 //DampFlag = 3. Suzanne PRL, 130, 178203 (2023) version
00146 else if(DampFlag == 3){

```

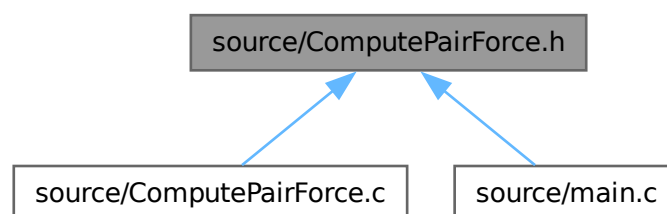
```

00148 //Track compression velocity
00149 DeltaXijNew = dr[1];
00150 DeltaYijNew = dr[2];
00151 if(stepCount == 0) { // Initialization step
00152     DeltaXijOldPair[atomIDi][atomIDj] = DeltaXijNew;
00153     DeltaYijOldPair[atomIDi][atomIDj] = DeltaYijNew;
00154 }
00155
00156 DeltaXij = DeltaXijNew - DeltaXijOldPair[atomIDi][atomIDj];
00157 DeltaYij = DeltaYijNew - DeltaYijOldPair[atomIDi][atomIDj];
00158 DeltaVXij = DeltaXij / deltaT;
00159 DeltaVYij = DeltaYij / deltaT;
00160
00161 // Update history for next step
00162 DeltaXijOldPair[atomIDi][atomIDj] = DeltaXijNew;
00163 DeltaYijOldPair[atomIDi][atomIDj] = DeltaYijNew;
00164
00165 discDragx[atomIDi] = -gamman * DeltaVXij; //disc-disc drag
00166 discDragy[atomIDi] = -gamman * DeltaVYij; //disc-disc drag
00167 discDragx[atomIDj] = -(-gamman * DeltaVXij); //disc-disc drag
00168 discDragy[atomIDj] = -(-gamman * DeltaVYij); //disc-disc drag
00169
00170 discDragx[nPairActive] = discDragx[atomIDi];
00171 discDragy[nPairActive] = discDragy[atomIDi];
00172
00173 ax[atomIDi] += (fcVal * dr[1] - gamman * DeltaVXij);
00174 ay[atomIDi] += (fcVal * dr[2] - gamman * DeltaVYij);
00175 ax[atomIDj] += -(fcVal * dr[1] - gamman * DeltaVXij);
00176 ay[atomIDj] += -(fcVal * dr[2] - gamman * DeltaVYij);
00177 }
00178
00179 //In the following, for stress/virial term (fcVal + fdVal) is used since the total pair force =
Hookean Interaction + relative velocity drag
00180 uSumPair += 0.5 * uVal;
00181 virSumPair += 0.5 * (fcVal + fdVal) * rr;
00182 virSumPairxx += 0.5 * (fcVal + fdVal) * dr[1] * dr[1];
00183 virSumPairyy += 0.5 * (fcVal + fdVal) * dr[2] * dr[2];
00184 virSumPairxy += 0.5 * (fcVal + fdVal) * dr[1] * dr[2];
00185 }
00186 else { //Resetting the distance between two discs when they are not in contact
00187     DeltaXijOldPair[atomIDi][atomIDj] = 0.0;
00188     DeltaYijOldPair[atomIDi][atomIDj] = 0.0;
00189     DeltaXijOldPair[atomIDj][atomIDi] = 0.0;
00190     DeltaYijOldPair[atomIDj][atomIDi] = 0.0;
00191 }
00192 }
00193 }
00194 }
00195 }
00196
00197
00198

```

3.32 source/ComputePairForce.h File Reference

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



Functions

- void [ComputePairForce](#) (int normFlag)

3.32.1 Function Documentation

3.32.1.1 ComputePairForce()

```
void ComputePairForce (
    int normFlag )
```

Definition at line 27 of file [ComputePairForce.c](#).

```
00027 {
00028 double dr[NDIM+1], fcVal, rr, ri, r, uVal;
00029 int n, i, j;
00030 uVal = 0.0; uSumPair = 0.0 ;
00031 virSumPair = 0.0; virSumPairxx = 0.0; virSumPairyy = 0.0; virSumPairxy = 0.0;
00032
00033 for(n = 1 ; n <= nAtom ; n ++){
00034 ax[n] = 0.0;
00035 ay[n] = 0.0;
00036 discDragx[n] = 0.0;
00037 discDragy[n] = 0.0;
00038 }
00039 for(n = 1; n <= nPairTotal; n ++){
00040 PairID[n] = 0;
00041 Pairatom1[n] = 0;
00042 Pairatom2[n] = 0;
00043 PairXij[n] = 0.0;
00044 PairYij[n] = 0.0;
00045 }
00046
00047
00048 Kn = 1.0;
00049 double vr[NDIM+1], fdVal, rri;
00050 nPairActive = 0;
00051 double meff;
00052 meff = 0.0;
00053 int atomIDi, atomIDj;
00054 //int processThisPair = 1;
00055
00056 for(i=1;i<=nAtomInterface;i++){
00057 for(j=i+1;j<=nAtomInterface;j++){
00058 atomIDi = atomIDInterface[i];
00059 atomIDj = atomIDInterface[j];
00060 if (isBonded[atomIDi][atomIDj] == 0) { //To exclude pair interaction between bonded atoms
00061 rr = 0.0; rri = 0.0; fcVal = 0.0; fdVal = 0.0; strech = 0.0;
00062 RadiusIJ = 0.0;
00063
00064 dr[1] = rx[atomIDi] - rx[atomIDj];
00065 if(dr[1] >= regionH[1])
00066 dr[1] -= region[1];
00067 else if(dr[1] < -regionH[1])
00068 dr[1] += region[1];
00069
00070 dr[2] = ry[atomIDi] - ry[atomIDj];
00071 if(dr[2] >= regionH[2]){
00072 dr[1] -= shearDisplacement;
00073 if(dr[1] < -regionH[1]) dr[1] += region[1];
00074 dr[2] -= region[2];
00075 }else if(dr[2] < -regionH[2]){
00076 dr[1] += shearDisplacement;
00077 if(dr[1] >= regionH[1]) dr[1] -= region[1];
00078 dr[2] += region[2];
00079 }
00080
00081 rr = Sqr(dr[1]) + Sqr(dr[2]);
00082 RadiusIJ = atomRadius[atomIDi] + atomRadius[atomIDj];
00083 SqrRadiusIJ = Sqr(RadiusIJ);
00084 if(rr < SqrRadiusIJ){
00085 r = sqrt(rr);
00086 ri = 1.0/r;
00087 rri = 1.0/rr;
00088 RadiusIJInv = 1.0/RadiusIJ;
00089 strech = (RadiusIJ - r);
00090 uVal = 0.5 * Kn * Sqr(strech);
00091
00092 //NormFlag
```

```

00093     if(normFlag == 1){
00094         stretch = stretch * RadiusIJInv;
00095         uVal = 0.5 * Kn * RadiusIJ * Sqr(stretch);
00096     }
00097
00098     fcVal = Kn * stretch * ri;
00099     vr[1] = vx[atomIDi] - vx[atomIDj];
00100     vr[2] = vy[atomIDi] - vy[atomIDj];
00101
00102     nPairActive++;
00103     PairID[nPairActive] = nPairActive;
00104     Pairatom1[nPairActive] = atomIDi;
00105     Pairatom2[nPairActive] = atomIDj;
00106     PairXij[nPairActive] = dr[1];
00107     PairYij[nPairActive] = dr[2];
00108
00109     //DampFlag = 1
00110     if(DampFlag == 1){
00111         meff = (atomMass[atomIDi]*atomMass[atomIDj])/(atomMass[atomIDi] + atomMass[atomIDj]);
00112         fdVal = -gamman * meff * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //disc-disc drag
00113
00114         discDragx[atomIDi] = fdVal * dr[1]; //disc-disc drag
00115         discDragy[atomIDi] = fdVal * dr[2]; //disc-disc drag
00116         discDragx[atomIDj] = -fdVal * dr[1]; //disc-disc drag
00117         discDragy[atomIDj] = -fdVal * dr[2]; //disc-disc drag
00118
00119         discDragx[nPairActive] = discDragx[atomIDi];
00120         discDragy[nPairActive] = discDragy[atomIDi];
00121
00122         ax[atomIDi] += (fcVal + fdVal) * dr[1];
00123         ay[atomIDi] += (fcVal + fdVal) * dr[2];
00124         ax[atomIDj] += -(fcVal + fdVal) * dr[1];
00125         ay[atomIDj] += -(fcVal + fdVal) * dr[2];
00126     }
00127
00128     //DampFlag = 2
00129     else if(DampFlag == 2){
00130         discDragx[atomIDi] = -gamman * vr[1]; //disc-disc drag
00131         discDragy[atomIDi] = -gamman * vr[2]; //disc-disc drag
00132         discDragx[atomIDj] = -(-gamman * vr[1]); //disc-disc drag
00133         discDragy[atomIDj] = -(-gamman * vr[2]); //disc-disc drag
00134
00135         discDragx[nPairActive] = discDragx[atomIDi];
00136         discDragy[nPairActive] = discDragy[atomIDi];
00137
00138         ax[atomIDi] += (fcVal * dr[1] - gamman * vr[1]);
00139         ay[atomIDi] += (fcVal * dr[2] - gamman * vr[2]);
00140         ax[atomIDj] += -(fcVal * dr[1] - gamman * vr[1]);
00141         ay[atomIDj] += -(fcVal * dr[2] - gamman * vr[2]);
00142     }
00143
00144     //DampFlag = 3. Suzanne PRL, 130, 178203 (2023) version
00145     else if(DampFlag == 3){
00146         //Track compression velocity
00147         DeltaXijNew = dr[1];
00148         DeltaYijNew = dr[2];
00149         if(stepCount == 0) { // Initialization step
00150             DeltaXijOldPair[atomIDi][atomIDj] = DeltaXijNew;
00151             DeltaYijOldPair[atomIDi][atomIDj] = DeltaYijNew;
00152         }
00153         DeltaXij = DeltaXijNew - DeltaXijOldPair[atomIDi][atomIDj];
00154         DeltaYij = DeltaYijNew - DeltaYijOldPair[atomIDi][atomIDj];
00155         DeltaVXij = DeltaXij / deltaT;
00156         DeltaVYij = DeltaYij / deltaT;
00157
00158         // Update history for next step
00159         DeltaXijOldPair[atomIDi][atomIDj] = DeltaXijNew;
00160         DeltaYijOldPair[atomIDi][atomIDj] = DeltaYijNew;
00161
00162         discDragx[atomIDi] = -gamman * DeltaVXij; //disc-disc drag
00163         discDragy[atomIDi] = -gamman * DeltaVYij; //disc-disc drag
00164         discDragx[atomIDj] = -(-gamman * DeltaVXij); //disc-disc drag
00165         discDragy[atomIDj] = -(-gamman * DeltaVYij); //disc-disc drag
00166
00167         discDragx[nPairActive] = discDragx[atomIDi];
00168         discDragy[nPairActive] = discDragy[atomIDi];
00169
00170         ax[atomIDi] += (fcVal * dr[1] - gamman * DeltaVXij);
00171         ay[atomIDi] += (fcVal * dr[2] - gamman * DeltaVYij);
00172         ax[atomIDj] += -(fcVal * dr[1] - gamman * DeltaVXij);
00173         ay[atomIDj] += -(fcVal * dr[2] - gamman * DeltaVYij);
00174     }
00175
00176     //In the following, for stress/virial term (fcVal + fdVal) is used since the total pair force =

```

```

    Hookean Interaction + relative velocity drag
00180     uSumPair += 0.5 * uVal;
00181     virSumPair += 0.5 * (fcVal + fdVal) * rr;
00182     virSumPairxx += 0.5 * (fcVal + fdVal) * dr[1] * dr[1];
00183     virSumPairyy += 0.5 * (fcVal + fdVal) * dr[2] * dr[2];
00184     virSumPairxy += 0.5 * (fcVal + fdVal) * dr[1] * dr[2];
00185 }
00186 else { //Resetting the distance between two discs when they are not in contact
00187     DeltaXijOldPair[atomIDi][atomIDj] = 0.0;
00188     DeltaYijOldPair[atomIDi][atomIDj] = 0.0;
00189     DeltaXijOldPair[atomIDj][atomIDi] = 0.0;
00190     DeltaYijOldPair[atomIDj][atomIDi] = 0.0;
00191 }
00192 }
00193 }
00194 }
00195 }

```

References [atomIDInterface](#), [atomMass](#), [atomRadius](#), [ax](#), [ay](#), [DampFlag](#), [deltaT](#), [DeltaVXij](#), [DeltaVYij](#), [DeltaXij](#), [DeltaXijNew](#), [DeltaXijOldPair](#), [DeltaYij](#), [DeltaYijNew](#), [DeltaYijOldPair](#), [discDragx](#), [discDragy](#), [gamman](#), [isBonded](#), [Kn](#), [nAtom](#), [nAtomInterface](#), [NDIM](#), [nPairActive](#), [nPairTotal](#), [Pairatom1](#), [Pairatom2](#), [PairID](#), [PairXij](#), [PairYij](#), [RadiusIJ](#), [RadiusIJInv](#), [region](#), [regionH](#), [rx](#), [ry](#), [shearDisplacement](#), [Sqr](#), [SqrRadiusIJ](#), [stepCount](#), [strech](#), [uSumPair](#), [virSumPair](#), [virSumPairxx](#), [virSumPairxy](#), [virSumPairyy](#), [vx](#), and [vy](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.33 ComputePairForce.h

[Go to the documentation of this file.](#)

```

00001 #ifndef COMPUTE_PAIR_FORCE_H
00002 #define COMPUTE_PAIR_FORCE_H
00003
00004 void ComputePairForce(int normFlag);
00005
00006 #endif
00007

```

3.34 source/DisplaceAtoms.c File Reference

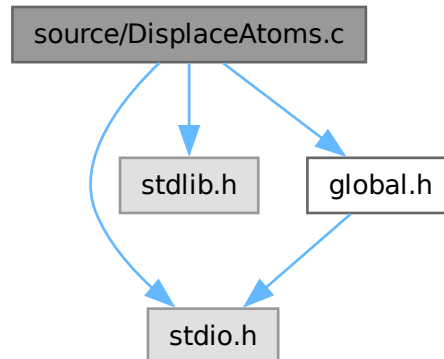
```

#include <stdio.h>
#include <stdlib.h>

```

```
#include "global.h"
```

Include dependency graph for DisplaceAtoms.c:



Functions

- void [DisplaceAtoms](#) ()

3.34.1 Function Documentation

3.34.1.1 DisplaceAtoms()

```
void DisplaceAtoms ( )
```

Definition at line 25 of file [DisplaceAtoms.c](#).

```
00025     {
00026   int n;
00027   for(n = 1; n <= nAtom; n++){
00028     if(molID[n] == 2){
00029       rx[n] += DeltaX;
00030       ry[n] += DeltaY;
00031     } } }
```

References [DeltaX](#), [DeltaY](#), [molID](#), [nAtom](#), [rx](#), and [ry](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.35 DisplaceAtoms.c

[Go to the documentation of this file.](#)

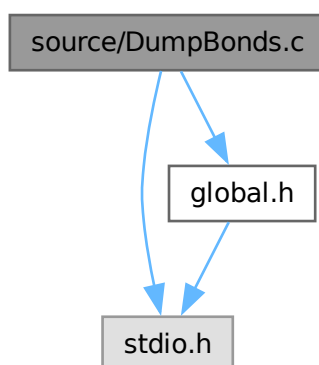
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021 #include<stdio.h>
00022 #include<stdlib.h>
00023 #include"global.h"
00024
00025 void DisplaceAtoms(){
00026     int n;
00027     for(n = 1; n <= nAtom; n ++){
00028         if(molID[n] == 2){
00029             rx[n] += DeltaX;
00030             ry[n] += DeltaY;
00031         } } }
```

3.36 source/DumpBonds.c File Reference

```
#include <stdio.h>
```

```
#include "global.h"
```

Include dependency graph for DumpBonds.c:



Functions

- void [DumpBonds](#) ()

3.36.1 Function Documentation

3.36.1.1 DumpBonds()

void DumpBonds ()

Definition at line 24 of file [DumpBonds.c](#).

```
00024     {
00025         int n;
00026         //Trajectory file in LAMMPS dump format for OVITO visualization
00027         fprintf(fpbond, "ITEM: TIMESTEP\n");
00028         fprintf(fpbond, "%lf\n", timeNow);
00029         fprintf(fpbond, "ITEM: NUMBER OF ENTRIES\n");
00030         fprintf(fpbond, "%d\n", nBond);
00031         fprintf(fpbond, "ITEM: BOX BOUNDS pp ff pp\n");
00032         fprintf(fpbond, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
00033         fprintf(fpbond, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
00034         fprintf(fpbond, "%lf %lf zlo zhi\n", -0.1, 0.1);
00035         fprintf(fpbond, "ITEM: ENTRIES BondID, BondType, atom1 atom2 BondLength BondLengthEqul nodeDragx1
nodeDragy1\n");
00036
00037         for(n=1; n<=nBond; n++)
00038             fprintf(fpbond, "%d %d %d %d %0.16lf %0.16lf %0.16lf %0.16lf\n", BondID[n], BondType[n], atom1[n],
atom2[n],
00039                 BondLength[n], ro[n], nodeDragx[atom1[n]], nodeDragy[atom1[n]]);
00040     }
```

References [atom1](#), [atom2](#), [BondID](#), [BondLength](#), [BondType](#), [fpbond](#), [nBond](#), [nodeDragx](#), [nodeDragy](#), [regionH](#), [ro](#), and [timeNow](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.37 DumpBonds.c

[Go to the documentation of this file.](#)

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021 #include<stdio.h>
```

```

00022 #include "global.h"
00023
00024 void DumpBonds() {
00025     int n;
00026     //Trajectory file in LAMMPS dump format for OVITO visualization
00027     fprintf(fpbond, "ITEM: TIMESTEP\n");
00028     fprintf(fpbond, "%lf\n", timeNow);
00029     fprintf(fpbond, "ITEM: NUMBER OF ENTRIES\n");
00030     fprintf(fpbond, "%d\n", nBond);
00031     fprintf(fpbond, "ITEM: BOX BOUNDS pp ff pp\n");
00032     fprintf(fpbond, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
00033     fprintf(fpbond, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
00034     fprintf(fpbond, "%lf %lf zlo zhi\n", -0.1, 0.1);
00035     fprintf(fpbond, "ITEM: ENTRIES BondID, BondType, atom1 atom2 BondLength BondLengthEqul nodeDragx1
nodeDragy1\n");
00036
00037     for(n=1; n<=nBond; n++)
00038         fprintf(fpbond, "%d %d %d %d %0.16lf %0.16lf %0.16lf %0.16lf\n", BondID[n], BondType[n], atom1[n],
atom2[n],
BondLength[n], ro[n], nodeDragx[atom1[n]], nodeDragy[atom1[n]]);
00039 }
00040
00041
00042
00043

```

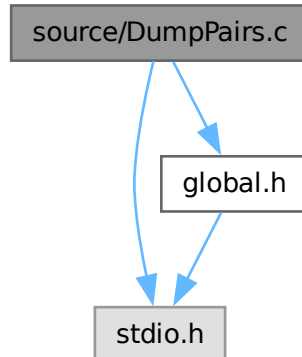
3.38 source/DumpPairs.c File Reference

```

#include <stdio.h>
#include "global.h"

```

Include dependency graph for DumpPairs.c:



Functions

- void [DumpPairs](#) ()

3.38.1 Function Documentation

3.38.1.1 DumpPairs()

```
void DumpPairs ( )
```

Definition at line 25 of file [DumpPairs.c](#).

```

00025     {
00026     int n;
00027     //Trajectory file in LAMMPS dump format for OVITO visualization
00028     fprintf(fppair, "ITEM: TIMESTEP\n");
00029     fprintf(fppair, "%lf\n", timeNow);
00030     fprintf(fppair, "ITEM: NUMBER OF ENTRIES\n");
00031     fprintf(fppair, "%d\n", nPairActive);
00032     fprintf(fppair, "ITEM: BOX BOUNDS pp ff pp\n");
00033     fprintf(fppair, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
00034     fprintf(fppair, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
00035     fprintf(fppair, "%lf %lf zlo zhi\n", -0.1, 0.1);
00036     fprintf(fppair, "ITEM: ENTRIES index, atom1 atom2 xij yij discDragx1 discDragy1\n");
00037
00038     for(n=1; n<=nPairActive; n++)
00039         fprintf(fppair, "%d %d %d %0.16lf %0.16lf %0.16lf %0.16lf\n", PairID[n], Pairatom1[n],
00040             Pairatom2[n],
00041             PairXij[n], PairYij[n], discDragx[n], discDragy[n]);
00042     }

```

References [discDragx](#), [discDragy](#), [fppair](#), [nPairActive](#), [Pairatom1](#), [Pairatom2](#), [PairID](#), [PairXij](#), [PairYij](#), [regionH](#), and [timeNow](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.39 DumpPairs.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 void DumpPairs(){
00026     int n;
00027     //Trajectory file in LAMMPS dump format for OVITO visualization
00028     fprintf(fppair, "ITEM: TIMESTEP\n");
00029     fprintf(fppair, "%lf\n", timeNow);
00030     fprintf(fppair, "ITEM: NUMBER OF ENTRIES\n");
00031     fprintf(fppair, "%d\n", nPairActive);

```



```

00032     fprintf(fppair, "ITEM: BOX BOUNDS pp ff pp\n");
00033     fprintf(fppair, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
00034     fprintf(fppair, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
00035     fprintf(fppair, "%lf %lf zlo zhi\n", -0.1, 0.1);
00036     fprintf(fppair, "ITEM: ENTRIES index, atom1 atom2 xij yij discDragx1 discDragy1\n");
00037
00038     for(n=1; n<=nPairActive; n++)
00039         fprintf(fppair, "%d %d %d %0.16lf %0.16lf %0.16lf %0.16lf\n", PairID[n], Pairatom1[n],
00040             Pairatom2[n],
00041             PairXij[n], PairYij[n], discDragx[n], discDragy[n]);
00042     }
00043
00044
00045

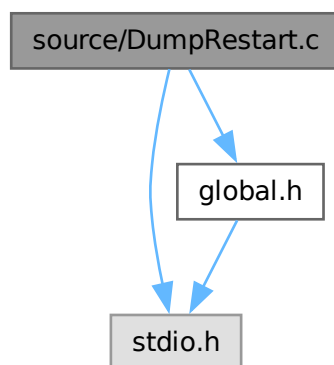
```

3.40 source/DumpRestart.c File Reference

```
#include <stdio.h>
```

```
#include "global.h"
```

Include dependency graph for DumpRestart.c:



Functions

- void [DumpRestart](#) ()

3.40.1 Function Documentation

3.40.1.1 DumpRestart()

```
void DumpRestart ( )
```

Definition at line 25 of file [DumpRestart.c](#).

```

00025     {
00026     char DUMP[256];
00027     FILE *fpDUMP;
00028     sprintf(DUMP, "%s.Restart", prefix);
00029     fpDUMP = fopen(DUMP, "w");
00030     if(fpDUMP == NULL) {

```

```

00031     fprintf(stderr, "Error opening file %s for writing\n", DUMP);
00032     return;
00033 }
00034
00035     fprintf(fpDUMP, "timeNow %lf\n", timeNow);
00036     fprintf(fpDUMP, "nAtom %d\n", nAtom);
00037     fprintf(fpDUMP, "nBond %d\n", nBond);
00038     fprintf(fpDUMP, "nAtomType %d\n", nAtomType);
00039     fprintf(fpDUMP, "nBondType %d\n", nBondType);
00040     fprintf(fpDUMP, "region[1] %0.14lf\n", region[1]);
00041     fprintf(fpDUMP, "region[2] %0.14lf\n", region[2]);
00042
00043     int n;
00044     fprintf(fpDUMP, "Atoms\n");
00045     for(n = 1; n <= nAtom; n++)
00046         fprintf(fpDUMP, "%d %d %d %0.2lf %0.16lf %0.16lf %0.16lf %0.16lf\n", atomID[n], molID[n],
atomType[n], atomRadius[n], rx[n], ry[n], vx[n], vy[n]);
00047
00048
00049     fprintf(fpDUMP, "Bonds\n");
00050     for(n=1; n<=nBond; n++)
00051         fprintf(fpDUMP, "%d %d %d %d %0.2lf %0.16lf\n", BondID[n], BondType[n], atom1[n], atom2[n], kb[n],
ro[n]);
00052
00053     fclose(fpDUMP);
00054 }

```

References [atom1](#), [atom2](#), [atomID](#), [atomRadius](#), [atomType](#), [BondID](#), [BondType](#), [kb](#), [molID](#), [nAtom](#), [nAtomType](#), [nBond](#), [nBondType](#), [prefix](#), [region](#), [ro](#), [rx](#), [ry](#), [timeNow](#), [vx](#), and [vy](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.41 DumpRestart.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022 #include <stdio.h>
00023 #include "global.h"
00024
00025 void DumpRestart() {

```

```

00026 char DUMP[256];
00027 FILE *fpDUMP;
00028 sprintf(DUMP, "%s.Restart", prefix);
00029 fpDUMP = fopen(DUMP, "w");
00030 if(fpDUMP == NULL) {
00031     fprintf(stderr, "Error opening file %s for writing\n", DUMP);
00032     return;
00033 }
00034
00035 fprintf(fpDUMP, "timeNow %lf\n", timeNow);
00036 fprintf(fpDUMP, "nAtom %d\n", nAtom);
00037 fprintf(fpDUMP, "nBond %d\n", nBond);
00038 fprintf(fpDUMP, "nAtomType %d\n", nAtomType);
00039 fprintf(fpDUMP, "nBondType %d\n", nBondType);
00040 fprintf(fpDUMP, "region[1] %0.14lf\n", region[1]);
00041 fprintf(fpDUMP, "region[2] %0.14lf\n", region[2]);
00042
00043 int n;
00044 fprintf(fpDUMP, "Atoms\n");
00045 for(n = 1; n <= nAtom; n++)
00046     fprintf(fpDUMP, "%d %d %d %0.21f %0.16lf %0.16lf %0.16lf %0.16lf\n", atomID[n], molID[n],
atomType[n], atomRadius[n], rx[n], ry[n], vx[n], vy[n]);
00047
00048
00049 fprintf(fpDUMP, "Bonds\n");
00050 for(n=1; n<=nBond; n++)
00051     fprintf(fpDUMP, "%d %d %d %d %0.21f %0.16lf\n", BondID[n], BondType[n], atom1[n], atom2[n], kb[n],
ro[n]);
00052
00053 fclose(fpDUMP);
00054 }
00055

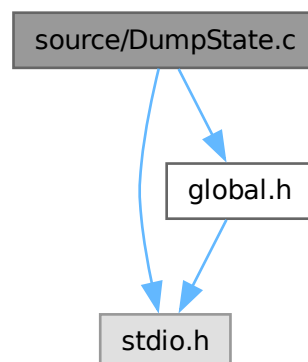
```

3.42 source/DumpState.c File Reference

```
#include <stdio.h>
```

```
#include "global.h"
```

Include dependency graph for DumpState.c:



Functions

- void [DumpState](#) ()

3.42.1 Function Documentation

3.42.1.1 DumpState()

```
void DumpState ( )
```

Definition at line 25 of file [DumpState.c](#).

```
00025 {
00026     char DUMP[256];
00027     FILE *fpDUMP;
00028     sprintf(DUMP, "%s.STATE", prefix);
00029     fpDUMP = fopen(DUMP, "w");
00030     if(fpDUMP == NULL) {
00031         fprintf(stderr, "Error opening file %s for writing\n", DUMP);
00032         return;
00033     }
00034
00035     fprintf(fpDUMP, "ITEM: TIMESTEP\n");
00036     fprintf(fpDUMP, "%lf\n", timeNow);
00037     fprintf(fpDUMP, "ITEM: NUMBER OF ATOMS\n");
00038     fprintf(fpDUMP, "%d\n", nAtom);
00039     fprintf(fpDUMP, "ITEM: BOX BOUNDS pp pp pp\n");
00040     fprintf(fpDUMP, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
00041     fprintf(fpDUMP, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
00042     fprintf(fpDUMP, "%lf %lf zlo zhi\n", -0.1, 0.1);
00043     fprintf(fpDUMP, "ITEM: ATOMS id mol type radius x y vx vy fx fy\n");
00044     int n;
00045     for (n = 1; n <= nAtom; n++) {
00046         fprintf(fpDUMP, "%d\t%d\t%d\t%0.2lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t\n",
00047             atomID[n], molID[n], atomType[n], atomRadius[n], rx[n], ry[n], vx[n], vy[n], ax[n], ay[n]);
00048     }
00049     fclose(fpDUMP);
00050 }
```

References [atomID](#), [atomRadius](#), [atomType](#), [ax](#), [ay](#), [molID](#), [nAtom](#), [prefix](#), [regionH](#), [rx](#), [ry](#), [timeNow](#), [vx](#), and [vy](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.43 DumpState.c

[Go to the documentation of this file.](#)

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
```

```

00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022  #include <stdio.h>
00023  #include "global.h"
00024
00025  void DumpState() {
00026      char DUMP[256];
00027      FILE *fpDUMP;
00028      sprintf(DUMP, "%s.STATE", prefix);
00029      fpDUMP = fopen(DUMP, "w");
00030      if(fpDUMP == NULL) {
00031          fprintf(stderr, "Error opening file %s for writing\n", DUMP);
00032          return;
00033      }
00034
00035      fprintf(fpDUMP, "ITEM: TIMESTEP\n");
00036      fprintf(fpDUMP, "%lf\n", timeNow);
00037      fprintf(fpDUMP, "ITEM: NUMBER OF ATOMS\n");
00038      fprintf(fpDUMP, "%d\n", nAtom);
00039      fprintf(fpDUMP, "ITEM: BOX BOUNDS pp pp pp\n");
00040      fprintf(fpDUMP, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
00041      fprintf(fpDUMP, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
00042      fprintf(fpDUMP, "%lf %lf zlo zhi\n", -0.1, 0.1);
00043      fprintf(fpDUMP, "ITEM: ATOMS id mol type radius x y vx vy fx fy\n");
00044      int n;
00045      for (n = 1; n <= nAtom; n++) {
00046          fprintf(fpDUMP, "%d\t%d\t%d\t%0.2lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t",
00047                  atomID[n], molID[n], atomType[n], atomRadius[n], rx[n], ry[n], vx[n], vy[n], ax[n], ay[n]);
00048      }
00049      fclose(fpDUMP);
00050  }
00051

```

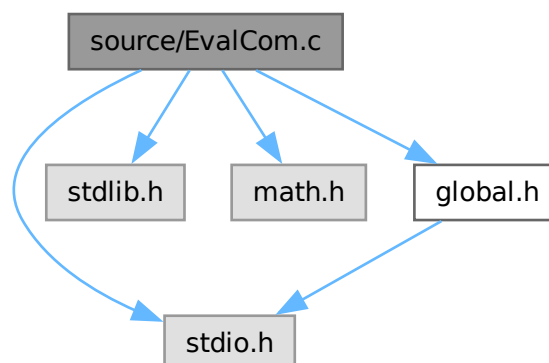
3.44 source/EvalCom.c File Reference

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "global.h"

```

Include dependency graph for EvalCom.c:



Functions

- void [EvalCom](#) ()

3.44.1 Function Documentation

3.44.1.1 EvalCom()

void EvalCom ()

Definition at line 27 of file [EvalCom.c](#).

```
00027     {
00028     int n;
00029     ComX = 0.0; ComY = 0.0; ComXRatio = 0.0; ComYRatio = 0.0;
00030     TotalMass = 0.0;
00031
00032     for(n=1; n<=nAtom; n++){
00033     if(molID[n] == 2){
00034         ComX += atomMass[n] * rxUnwrap[n];
00035         ComY += atomMass[n] * ryUnwrap[n];
00036         TotalMass += atomMass[n];
00037     } }
00038
00039     ComX = ComX/TotalMass;
00040     ComY = ComY/TotalMass;
00041
00042     if(timeNow == 0.0){
00043         ComX0 = ComX; ComY0 = ComY;
00044     }
00045     ComXRatio = ComX/ComX0;    ComYRatio = ComY/ComY0;
00046 }
```

References [atomMass](#), [ComX](#), [ComX0](#), [ComXRatio](#), [ComY](#), [ComY0](#), [ComYRatio](#), [molID](#), [nAtom](#), [rxUnwrap](#), [ryUnwrap](#), [timeNow](#), and [TotalMass](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.45 EvalCom.c

[Go to the documentation of this file.](#)

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
```

```

00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022  #include<stdio.h>
00023  #include<stdlib.h>
00024  #include<math.h>
00025  #include"global.h"
00026
00027  void EvalCom(){
00028      int n;
00029      ComX = 0.0; ComY = 0.0; ComXRatio = 0.0; ComYRatio = 0.0;
00030      TotalMass = 0.0;
00031
00032      for(n=1; n<=nAtom; n++){
00033          if(molID[n] == 2){
00034              ComX += atomMass[n] * rxUnwrap[n];
00035              ComY += atomMass[n] * ryUnwrap[n];
00036              TotalMass += atomMass[n];
00037          } }
00038
00039      ComX = ComX/TotalMass;
00040      ComY = ComY/TotalMass;
00041
00042      if(timeNow == 0.0){
00043          ComX0 = ComX; ComY0 = ComY;
00044      }
00045      ComXRatio = ComX/ComX0;   ComYRatio = ComY/ComY0;
00046  }
00047
00048
00049

```

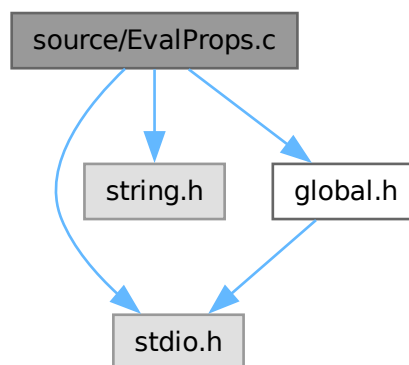
3.46 source/EvalProps.c File Reference

```

#include <stdio.h>
#include <string.h>
#include "global.h"

```

Include dependency graph for EvalProps.c:



Functions

- void [EvalProps](#) ()

3.46.1 Function Documentation

3.46.1.1 EvalProps()

void EvalProps ()

Definition at line 26 of file [EvalProps.c](#).

```

00026         {
00027     real v, vv;
00028     virSum = 0.0;
00029     vSumX = 0.0; vSumY = 0.0; vSum = 0.0;
00030     vvSum = 0.;
00031     int n;
00032
00033     for (n = 1; n <= nAtom; n++) {
00034         vv = 0.;
00035         // Initialize v with a default value to avoid "uninitialized" warning.
00036         v = 0.0;
00037         // X direction velocity
00038         if (strcmp(solver, "Verlet") == 0) {
00039             v = vx[n];
00040         } else if (strcmp(solver, "LeapFrog") == 0) {
00041             v = vx[n] - 0.5 * deltaT * ax[n];
00042         }
00043         vSum += v;
00044         vv += Sqr(v);
00045         vSumX += v;
00046         // Y direction velocity
00047         if (strcmp(solver, "Verlet") == 0) {
00048             v = vy[n];
00049         } else if (strcmp(solver, "LeapFrog") == 0) {
00050             v = vy[n] - 0.5 * deltaT * ay[n];
00051         }
00052         vSum += v;
00053         vSumY += v;
00054         vv += Sqr(v);
00055         vvSum += vv;
00056     }
00057
00058     kinEnergy = 0.5 * vvSum / nAtom ;
00059     uSumPairPerAtom = uSumPair / nAtom ;
00060     BondEnergyPerAtom = TotalBondEnergy / (0.5*nAtom); //Factor of 0.5 since each atom has one half the
    bond energy
00061     potEnergy = uSumPairPerAtom + BondEnergyPerAtom ;
00062     totEnergy = kinEnergy + potEnergy;
00063     virSumxx = virSumPairxx + virSumBondxx ;
00064     virSumyy = virSumPairyy + virSumBondyy ;
00065     virSumxy = virSumPairxy + virSumBondxy ;
00066     virSum = virSumPair + virSumBond;
00067     pressure = density * (vvSum + virSum) / (nAtom * NDIM);
00068
00069 }
```

References [ax](#), [ay](#), [BondEnergyPerAtom](#), [deltaT](#), [density](#), [kinEnergy](#), [nAtom](#), [NDIM](#), [potEnergy](#), [pressure](#), [solver](#), [Sqr](#), [TotalBondEnergy](#), [totEnergy](#), [uSumPair](#), [uSumPairPerAtom](#), [virSum](#), [virSumBond](#), [virSumBondxx](#), [virSumBondxy](#), [virSumBondyy](#), [virSumPair](#), [virSumPairxx](#), [virSumPairxy](#), [virSumPairyy](#), [virSumxx](#), [virSumxy](#), [virSumyy](#), [vSum](#), [vSumX](#), [vSumY](#), [vvSum](#), [vx](#), and [vy](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.47 EvalProps.c

[Go to the documentation of this file.](#)

```

00001  /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022 #include <stdio.h>
00023 #include <string.h>
00024 #include "global.h"
00025
00026 void EvalProps() {
00027     real v, vv;
00028     virSum = 0.0;
00029     vSumX = 0.0; vSumY = 0.0; vSum = 0.0;
00030     vvSum = 0.;
00031     int n;
00032
00033     for (n = 1; n <= nAtom; n++) {
00034         vv = 0.;
00035         // Initialize v with a default value to avoid "uninitialized" warning.
00036         v = 0.0;
00037         // X direction velocity
00038         if (strcmp(solver, "Verlet") == 0) {
00039             v = vx[n];
00040         } else if (strcmp(solver, "LeapFrog") == 0) {
00041             v = vx[n] - 0.5 * deltaT * ax[n];
00042         }
00043         vSum += v;
00044         vv += Sqr(v);
00045         vSumX += v;
00046         // Y direction velocity
00047         if (strcmp(solver, "Verlet") == 0) {
00048             v = vy[n];
00049         } else if (strcmp(solver, "LeapFrog") == 0) {
00050             v = vy[n] - 0.5 * deltaT * ay[n];
00051         }
00052         vSum += v;
00053         vSumY += v;
00054         vv += Sqr(v);
00055         vvSum += vv;
00056     }
00057
00058     kinEnergy = 0.5 * vvSum / nAtom ;
00059     uSumPairPerAtom = uSumPair / nAtom ;
00060     BondEnergyPerAtom = TotalBondEnergy / (0.5*nAtom); //Factor of 0.5 since each atom has one half the
    bond energy
00061     potEnergy = uSumPairPerAtom + BondEnergyPerAtom ;
00062     totEnergy = kinEnergy + potEnergy;
00063     virSumxx = virSumPairxx + virSumBondxx ;
00064     virSumyy = virSumPairyy + virSumBondyy ;
00065     virSumxy = virSumPairxy + virSumBondxy ;
00066     virSum = virSumPair + virSumBond;
00067     pressure = density * (vvSum + virSum) / (nAtom * NDIM);
00068
00069 }
00070

```

3.48 source/EvalRdf.c File Reference

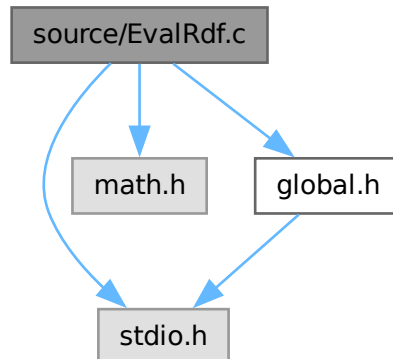
```

#include <stdio.h>
#include <math.h>

```

```
#include "global.h"
```

Include dependency graph for EvalRdf.c:



Functions

- void [EvalRdf](#) ()

3.48.1 Function Documentation

3.48.1.1 EvalRdf()

```
void EvalRdf ( )
```

Definition at line 26 of file [EvalRdf.c](#).

```

00026 {
00027     real dr[NDIM+1], deltaR, normFac, rr, rrRange;
00028     int j1, j2, n;
00029     countRdf ++;
00030     if(countRdf == 1){
00031         for(n = 1 ; n <= sizeHistRdf ; n ++){
00032             histRdf[n] = 0.;
00033         }
00034         rrRange = Sqr(rangeRdf);
00035         deltaR = rangeRdf / sizeHistRdf;
00036         for(j1 = 1 ; j1 <= nAtom - 1 ; j1 ++){
00037             for(j2 = j1 + 1 ; j2 <= nAtom ; j2 ++){
00038                 dr[1] = rx[j1] - rx[j2];
00039                 if(fabs(dr[1]) > regionH[1])
00040                     dr[1] -= SignR(region[1], dr[1]);
00041                 dr[2] = ry[j1] - ry[j2];
00042                 if(fabs(dr[2]) > regionH[2])
00043                     dr[2] -= SignR(region[2], dr[2]);
00044                 rr = Sqr(dr[1]) + Sqr(dr[2]);
00045                 if(rr < rrRange){
00046                     n = (int)(sqrt(rr)/deltaR) + 1;
00047                     histRdf[n] ++;
00048                 }
00049             }
00050         }
00051     }
00052     if(countRdf == limitRdf){

```

```

00057     normFac = region[1]*region[2] / (M_PI*Sqr(deltaR)*nAtom*nAtom*countRdf );
00058     for(n = 1 ; n <= sizeHistRdf ; n ++){
00059         histRdf[n] *= normFac/(n-0.5);
00060     // PRINT THE RADIAL DISTRIBUTION DATA ON TO DISK FILE
00061     real rBin;
00062     int n;
00063     fprintf(fprdf,"rdf @ timeNow %lf\n", timeNow);
00064     for(n = 1 ; n <= sizeHistRdf ; n ++){
00065         rBin = (n - 0.5)*rangeRdf/sizeHistRdf;
00066         fprintf(fprdf, "%lf %lf\n", rBin, histRdf[n]);
00067     }
00068 }
00069
00070 }

```

References [countRdf](#), [fprdf](#), [histRdf](#), [limitRdf](#), [nAtom](#), [NDIM](#), [rangeRdf](#), [region](#), [regionH](#), [rx](#), [ry](#), [SignR](#), [sizeHistRdf](#), [Sqr](#), and [timeNow](#).

3.49 EvalRdf.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022 #include<stdio.h>
00023 #include<math.h>
00024 #include"global.h"
00025
00026 void EvalRdf(){
00027     real dr[NDIM+1], deltaR, normFac, rr, rrRange;
00028     int j1, j2, n;
00029     countRdf ++;
00030     if(countRdf == 1){
00031         for(n = 1 ; n <= sizeHistRdf ; n ++){
00032             histRdf[n] = 0.;
00033         }
00034         rrRange = Sqr(rangeRdf);
00035         deltaR = rangeRdf / sizeHistRdf;
00036         for(j1 = 1 ; j1 <= nAtom - 1 ; j1 ++){
00037             for(j2 = j1 + 1 ; j2 <= nAtom ; j2 ++){
00038
00039                 dr[1] = rx[j1] - rx[j2];
00040                 if(fabs(dr[1]) > regionH[1])
00041                     dr[1] -= SignR(region[1], dr[1]);
00042
00043                 dr[2] = ry[j1] - ry[j2];
00044                 if(fabs(dr[2]) > regionH[2])
00045                     dr[2] -= SignR(region[2], dr[2]);
00046
00047                 rr = Sqr(dr[1]) + Sqr(dr[2]);
00048
00049                 if(rr < rrRange){
00050                     n = (int)(sqrt(rr)/deltaR) + 1;
00051                     histRdf[n] ++;
00052                 }
00053             }
00054         }
00055
00056         if(countRdf == limitRdf){
00057             normFac = region[1]*region[2] / (M_PI*Sqr(deltaR)*nAtom*nAtom*countRdf );
00058             for(n = 1 ; n <= sizeHistRdf ; n ++){
00059                 histRdf[n] *= normFac/(n-0.5);

```

```

00060    // PRINT THE RADIAL DISTRIBUTION DATA ON TO DISK FILE
00061    real rBin;
00062    int n;
00063    fprintf(fprdf,"rdf @ timeNow %lf\n", timeNow);
00064    for(n = 1 ; n <= sizeHistRdf ; n++){
00065        rBin = (n - 0.5)*rangeRdf/sizeHistRdf;
00066        fprintf(fprdf, "%lf %lf\n", rBin, histRdf[n]);
00067    }
00068 }
00069
00070 }
00071

```

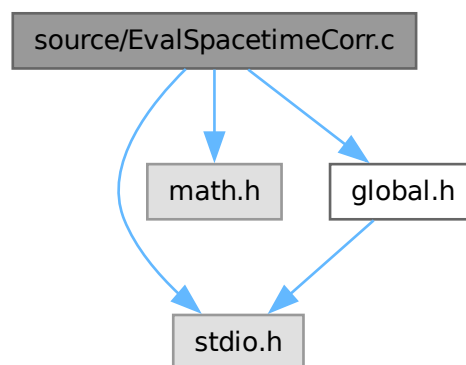
3.50 source/EvalSpacetimeCorr.c File Reference

```

#include <stdio.h>
#include <math.h>
#include "global.h"

```

Include dependency graph for EvalSpacetimeCorr.c:



Functions

- void [EvalSpacetimeCorr](#) ()

3.50.1 Function Documentation

3.50.1.1 EvalSpacetimeCorr()

```
void EvalSpacetimeCorr ( )
```

Definition at line 26 of file [EvalSpacetimeCorr.c](#).

```

00026    {
00027        real cosV=0., cosV0=0., cosV1=0., cosV2=0., sinV=0., sinV1=0., sinV2=0.;
00028        real COSA, SINA, COSV, SINV;
00029        int j, m, n, nb, ni, nv;
00030        real kMin = 2. * M_PI / region[1];
00031        real kMax = M_PI;
00032        real deltaK = (kMax - kMin) / nFunCorr;

```

```

00033
00034     for (j = 1; j <= 2*nFunCorr; j++)
00035         cfVal[j] = 0.;
00036
00037     for (n = 1; n <= nAtom; n++){
00038         j = 1;
00039         COSA = cos(kMin*rx[n]);
00040         SINA = sin(kMin*rx[n]);
00041         for (m = 1; m <= nFunCorr; m++){
00042             if(m == 1){
00043                 cosV = cos(deltaK*rx[n]);
00044                 sinV = sin(deltaK*rx[n]);
00045                 cosV0 = cosV;
00046             }else if(m == 2){
00047                 cosV1 = cosV;
00048                 sinV1 = sinV;
00049                 cosV = 2.*cosV0*cosV1-1;
00050                 sinV = 2.*cosV0*sinV1;
00051             }else{
00052                 cosV2 = cosV1;
00053                 sinV2 = sinV1;
00054                 cosV1 = cosV;
00055                 sinV1 = sinV;
00056                 cosV = 2.*cosV0*cosV1-cosV2;
00057                 sinV = 2.*cosV0*sinV1-sinV2;
00058             }
00059             COSV = COSA*cosV - SINA*sinV;
00060             SINV = SINA*cosV + COSA*sinV;
00061             cfVal[j] += COSV;
00062             cfVal[j+1] += SINV;
00063             j += 2;
00064         }
00065     }
00066
00067     for (nb = 1; nb <= nBuffCorr; nb++){
00068         indexCorr[nb] += 1;
00069         if (indexCorr[nb] <= 0) continue;
00070         ni = nFunCorr * (indexCorr[nb] - 1);
00071         if (indexCorr[nb] == 1){
00072             for (j = 1; j <= 2*nFunCorr; j++)
00073                 cfOrg[nb][j] = cfVal[j];
00074         }
00075
00076         for (j = 1; j <= nFunCorr; j++)
00077             spacetimeCorr[nb][ni + j] = 0.;
00078
00079         j = 1;
00080         for (m = 1; m <= nFunCorr; m++){
00081             nv = m + ni;
00082             spacetimeCorr[nb][nv] += cfVal[j] * cfOrg[nb][j] + cfVal[j + 1] * cfOrg[nb][j + 1];
00083             j += 2;
00084         }
00085     }
00086
00087     // ACCUMULATE SPACETIME CORRELATIONS
00088     for (nb = 1; nb <= nBuffCorr; nb++){
00089         if (indexCorr[nb] == nValCorr){
00090             for (j = 1; j <= nFunCorr*nValCorr; j++)
00091                 spacetimeCorrAv[j] += spacetimeCorr[nb][j];
00092             indexCorr[nb] = 0.;
00093             countCorrAv ++;
00094             if (countCorrAv == limitCorrAv){
00095                 for (j = 1; j <= nFunCorr*nValCorr; j++)
00096                     spacetimeCorrAv[j] /= (nAtom*limitCorrAv);
00097                 fprintf(fpdnsty, "NDIM %d\n", NDIM);
00098                 fprintf(fpdnsty, "nAtom %d\n", nAtom);
00099                 fprintf(fpdnsty, "region %lf\n", region[1]);
00100                 fprintf(fpdnsty, "nFunCorr %d\n", nFunCorr);
00101                 fprintf(fpdnsty, "limitCorrAv %d\n", limitCorrAv);
00102                 fprintf(fpdnsty, "stepCorr %d\n", stepCorr);
00103                 fprintf(fpdnsty, "nValCorr %d\n", nValCorr);
00104                 fprintf(fpdnsty, "deltaT %lf\n", deltaT);
00105                 real tVal;
00106                 for (n = 1; n <= nValCorr; n++){
00107                     tVal = (n-1)*stepCorr*deltaT;
00108                     fprintf(fpdnsty, "%e\t", tVal);
00109                     int nn = nFunCorr*(n-1);
00110                     for (j = 1; j <= nFunCorr; j++)
00111                         fprintf(fpdnsty, "%e\t", spacetimeCorrAv[nn + j]);
00112                     fprintf(fpdnsty, "\n");
00113                 }
00114             }
00115             countCorrAv = 0.;
00116             for (j = 1; j <= nFunCorr*nValCorr; j++)
00117                 spacetimeCorrAv[j] = 0.;
00118         }
00119     }

```

```

00120     }
00121   }
00122 }

```

References [cfOrg](#), [cfVal](#), [countCorrAv](#), [deltaT](#), [fpdnsty](#), [indexCorr](#), [limitCorrAv](#), [nAtom](#), [nBuffCorr](#), [NDIM](#), [nFunCorr](#), [nValCorr](#), [region](#), [rx](#), [spacetimeCorr](#), [spacetimeCorrAv](#), and [stepCorr](#).

3.51 EvalSpacetimeCorr.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022 #include<stdio.h>
00023 #include<math.h>
00024 #include"global.h"
00025
00026 void EvalSpacetimeCorr () {
00027     real cosV=0., cosV0=0., cosV1=0., cosV2=0., sinV=0., sinV1=0., sinV2=0.;
00028     real COSA, SINA, COSV, SINV;
00029     int j, m, n, nb, ni, nv;
00030     real kMin = 2. * M_PI / region[1];
00031     real kMax = M_PI;
00032     real deltaK = (kMax - kMin) / nFunCorr;
00033
00034     for (j = 1; j <= 2*nFunCorr; j++)
00035         cfVal[j] = 0.;
00036
00037     for (n = 1; n <= nAtom; n++){
00038         j = 1;
00039         COSA = cos(kMin*rx[n]);
00040         SINA = sin(kMin*rx[n]);
00041         for (m = 1; m <= nFunCorr; m++){
00042             if(m == 1){
00043                 cosV = cos(deltaK*rx[n]);
00044                 sinV = sin(deltaK*rx[n]);
00045                 cosV0 = cosV;
00046             }else if(m == 2){
00047                 cosV1 = cosV;
00048                 sinV1 = sinV;
00049                 cosV = 2.*cosV0*cosV1-1;
00050                 sinV = 2.*cosV0*sinV1;
00051             }else{
00052                 cosV2 = cosV1;
00053                 sinV2 = sinV1;
00054                 cosV1 = cosV;
00055                 sinV1 = sinV;
00056                 cosV = 2.*cosV0*cosV1-cosV2;
00057                 sinV = 2.*cosV0*sinV1-sinV2;
00058             }
00059             COSV = COSA*cosV - SINA*sinV;
00060             SINV = SINA*cosV + COSA*sinV;
00061             cfVal[j] += COSV;
00062             cfVal[j+1] += SINV;
00063             j += 2;
00064         }
00065     }
00066
00067     for (nb = 1; nb <= nBuffCorr; nb++){
00068         indexCorr[nb] += 1;
00069         if (indexCorr[nb] <= 0) continue;
00070         ni = nFunCorr * (indexCorr[nb] - 1);

```

```

00071     if (indexCorr[nb] == 1){
00072         for (j = 1; j <= 2*nFunCorr; j++)
00073             cfOrg[nb][j] = cfVal[j];
00074     }
00075
00076     for (j = 1; j <= nFunCorr; j++)
00077         spacetTimeCorr[nb][ni + j] = 0.;
00078
00079     j = 1;
00080     for (m = 1; m <= nFunCorr; m++){
00081         nv = m + ni;
00082         spacetTimeCorr[nb][nv] += cfVal[j] * cfOrg[nb][j] + cfVal[j + 1] * cfOrg[nb][j + 1];
00083         j += 2;
00084     }
00085
00086 }
00087
00088 // ACCUMULATE SPACETIME CORRELATIONS
00089 for (nb = 1; nb <= nBuffCorr; nb++){
00090     if (indexCorr[nb] == nValCorr){
00091         for (j = 1; j <= nFunCorr*nValCorr; j++)
00092             spacetTimeCorrAv[j] += spacetTimeCorr[nb][j];
00093         indexCorr[nb] = 0.;
00094         countCorrAv ++;
00095         if (countCorrAv == limitCorrAv){
00096             for (j = 1; j <= nFunCorr*nValCorr; j++)
00097                 spacetTimeCorrAv[j] /= (nAtom*limitCorrAv);
00098             fprintf(fpdnsty, "NDIM %d\n", NDIM);
00099             fprintf(fpdnsty, "nAtom %d\n", nAtom);
00100             fprintf(fpdnsty, "region %lf\n", region[1]);
00101             fprintf(fpdnsty, "nFunCorr %d\n", nFunCorr);
00102             fprintf(fpdnsty, "limitCorrAv %d\n", limitCorrAv);
00103             fprintf(fpdnsty, "stepCorr %d\n", stepCorr);
00104             fprintf(fpdnsty, "nValCorr %d\n", nValCorr);
00105             fprintf(fpdnsty, "deltaT %lf\n", deltaT);
00106             real tVal;
00107             for (n = 1; n <= nValCorr; n++){
00108                 tVal = (n-1)*stepCorr*deltaT;
00109                 fprintf (fpdnsty, "%e\t", tVal);
00110                 int nn = nFunCorr*(n-1);
00111                 for (j = 1; j <= nFunCorr; j ++){
00112                     fprintf (fpdnsty, "%e\t", spacetTimeCorrAv[nn + j]);
00113                     fprintf (fpdnsty, "\n");
00114                 }
00115
00116                 countCorrAv = 0.;
00117                 for (j = 1; j <= nFunCorr*nValCorr; j++)
00118                     spacetTimeCorrAv[j] = 0.;
00119             }
00120         }
00121     }
00122 }

```

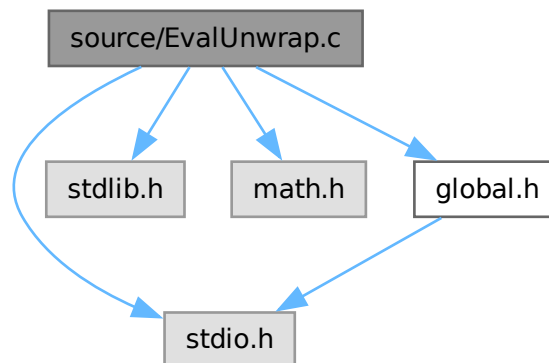
3.52 source/EvalUnwrap.c File Reference

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "global.h"

```

Include dependency graph for EvalUnwrap.c:



Functions

- void [EvalUnwrap](#) ()

3.52.1 Function Documentation

3.52.1.1 EvalUnwrap()

```
void EvalUnwrap ( )
```

Definition at line 27 of file [EvalUnwrap.c](#).

```

00027     {
00028     int n;
00029     for (n = 1; n <= nAtom; n++) {
00030         rxUnwrap[n] = rx[n] + ImageX[n] * region[1];
00031         ryUnwrap[n] = ry[n] + ImageY[n] * region[2];
00032     }
00033 }
```

References [ImageX](#), [ImageY](#), [nAtom](#), [region](#), [rx](#), [rxUnwrap](#), [ry](#), and [ryUnwrap](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.53 EvalUnwrap.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022 #include <stdio.h>
00023 #include <stdlib.h>
00024 #include <math.h>
00025 #include "global.h"
00026
00027 void EvalUnwrap() {
00028     int n;
00029     for (n = 1; n <= nAtom; n++) {
00030         rxUnwrap[n] = rx[n] + ImageX[n] * region[1];
00031         ryUnwrap[n] = ry[n] + ImageY[n] * region[2];
00032     }
00033 }
00034
```

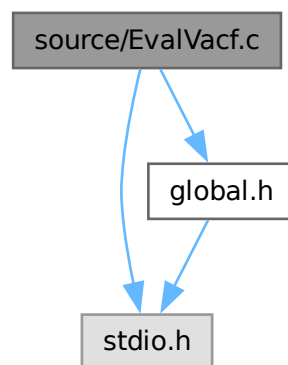
3.54 source/EvalVacf.c File Reference

```

#include <stdio.h>
#include "global.h"

```

Include dependency graph for EvalVacf.c:



Functions

- void [AccumVacf](#) ()
- void [EvalVacf](#) ()

3.54.1 Function Documentation

3.54.1.1 AccumVacf()

```
void AccumVacf ( )
```

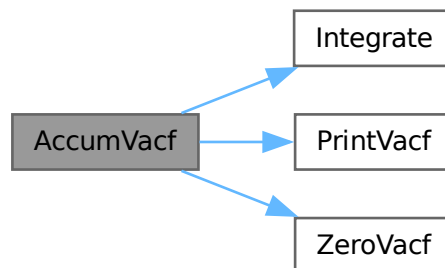
Definition at line 27 of file [AccumVacf.c](#).

```
00027     {
00028     double fac;
00029     int j, nb;
00030     for(nb = 1 ; nb <= nBuffAcf ; nb ++){
00031         if(indexAcf[nb] == nValAcf){
00032             for(j = 1 ; j <= nValAcf; j ++){
00033                 viscAcfAv[j] += viscAcf[nb][j];
00034             }
00035             indexAcf[nb] = 0;
00036             countAcfAv ++;
00037             if(countAcfAv == limitAcfAv){
00038                 fac = 1./(kinEnergy*region[1]*region[2]*limitAcfAv);
00039                 viscAcfInt = fac*stepAcf*deltaT*Integrate(viscAcfAv, nValAcf);
00040                 PrintVacf();
00041                 ZeroVacf();
00042             } } } }
```

References [countAcfAv](#), [deltaT](#), [indexAcf](#), [Integrate\(\)](#), [kinEnergy](#), [limitAcfAv](#), [nBuffAcf](#), [nValAcf](#), [PrintVacf\(\)](#), [region](#), [stepAcf](#), [viscAcf](#), [viscAcfAv](#), [viscAcfInt](#), and [ZeroVacf\(\)](#).

Referenced by [EvalVacf\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.54.1.2 EvalVacf()

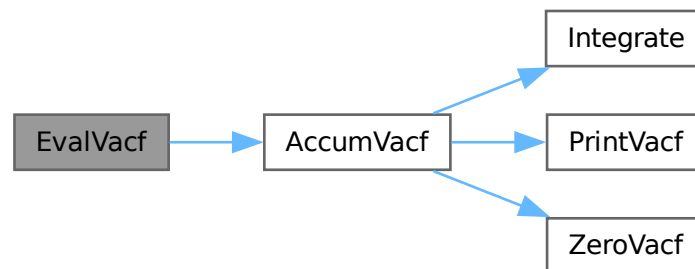
```
void EvalVacf ( )
```

Definition at line 26 of file [EvalVacf.c](#).

```
00026     {
00027     int n, nb, ni;
00028     double viscVec = 0.;
00029     double v[3];
00030     for(n = 1 ; n <= nAtom ; n ++){
00031         v[1] = vx[n] - 0.5*ax[n]*deltaT;
00032         v[2] = vy[n] - 0.5*ay[n]*deltaT;
00033         viscVec += v[1]*v[2];
00034     }
00035     viscVec += rfAtom;
00036     for(nb = 1 ; nb <= nBuffAcf ; nb ++){
00037         indexAcf[nb] ++;
00038         if(indexAcf[nb] <= 0)continue;
00039         if(indexAcf[nb] == 1){
00040             viscAcfOrg[nb] = viscVec;
00041         }
00042         ni = indexAcf[nb];
00043         viscAcf[nb][ni] = viscAcfOrg[nb]*viscVec;
00044     }
00045     AccumVacf();
00046 }
```

References [AccumVacf\(\)](#), [ax](#), [ay](#), [deltaT](#), [indexAcf](#), [nAtom](#), [nBuffAcf](#), [rfAtom](#), [viscAcf](#), [viscAcfOrg](#), [vx](#), and [vy](#).

Here is the call graph for this function:



3.55 EvalVacf.c

[Go to the documentation of this file.](#)

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
```

```

00018
00019  */
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 void AccumVacf();
00026 void EvalVacf(){
00027     int n, nb, ni;
00028     double viscVec = 0.;
00029     double v[3];
00030     for(n = 1 ; n <= nAtom ; n ++){
00031         v[1] = vx[n] - 0.5*ax[n]*deltaT;
00032         v[2] = vy[n] - 0.5*ay[n]*deltaT;
00033         viscVec += v[1]*v[2];
00034     }
00035     viscVec += rfAtom;
00036     for(nb = 1 ; nb <= nBuffAcf ; nb ++){
00037         indexAcf[nb] ++;
00038         if(indexAcf[nb] <= 0) continue;
00039         if(indexAcf[nb] == 1){
00040             viscAcfOrg[nb] = viscVec;
00041         }
00042         ni = indexAcf[nb];
00043         viscAcf[nb][ni] = viscAcfOrg[nb]*viscVec;
00044     }
00045     AccumVacf();
00046 }

```

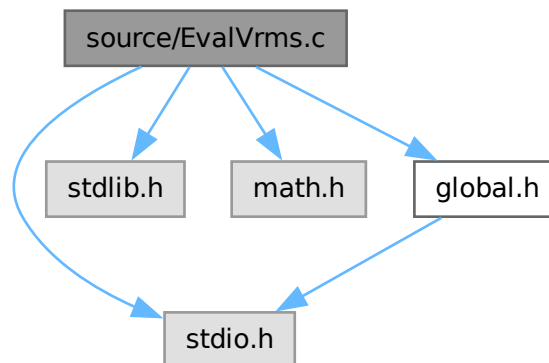
3.56 source/EvalVrms.c File Reference

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "global.h"

```

Include dependency graph for EvalVrms.c:



Functions

- void [EvalVrms](#) ()

3.56.1 Function Documentation

3.56.1.1 EvalVrms()

```
void EvalVrms ( )
```

Definition at line 27 of file [EvalVrms.c](#).

```
00027     {
00028     int n;
00029     VSqr = 0.0;
00030     VMeanSqr = 0.0;
00031     VRootMeanSqr = 0.0;
00032
00033     for(n = 1 ; n <= nAtom ; n ++){
00034     VSqr += Sqr(vx[n]) + Sqr(vy[n]);
00035     }
00036     VMeanSqr = VSqr/nAtom;
00037     VRootMeanSqr = sqrt(VMeanSqr);
00038 }
```

References [nAtom](#), [Sqr](#), [VMeanSqr](#), [VRootMeanSqr](#), [VSqr](#), [vx](#), and [vy](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.57 EvalVrms.c

[Go to the documentation of this file.](#)

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022 #include<stdio.h>
00023 #include<stdlib.h>
00024 #include<math.h>
00025 #include"global.h"
00026
00027 void EvalVrms() {
00028     int n;
00029     VSqr = 0.0;
```

```

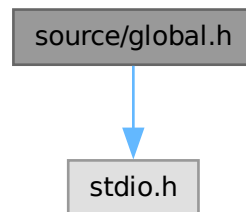
00030  VMeanSqr = 0.0;
00031  VRootMeanSqr = 0.0;
00032
00033  for(n = 1 ; n <= nAtom ; n ++){
00034    VSqr += Sqr(vx[n]) + Sqr(vy[n]);
00035  }
00036  VMeanSqr = VSqr/nAtom;
00037  VRootMeanSqr = sqrt(VMeanSqr);
00038  }
00039
00040
00041

```

3.58 source/global.h File Reference

```
#include <stdio.h>
```

Include dependency graph for global.h:



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



Macros

- #define **EXTERN** extern
- #define **NDIM** 2
- #define **Sqr**(x) ((x) * (x))
- #define **SignR**(x, y) (((y) >= 0) ? (x) : (- (x)))

Typedefs

- typedef double **real**

Variables

- double * rx
- double * ry
- double * vx
- double * vy
- double * ax
- double * ay
- double * speed
- double region [2+1]
- double regionH [2+1]
- double deltaT
- double timeNow
- double potEnergy
- double kinEnergy
- double totEnergy
- double density
- double pressure
- double rCut
- double kappa
- double uSum
- double virSum
- double svirSum
- double vSum
- double vSumX
- double vSumY
- double vvSum
- double sPotEnergy
- double sKinEnergy
- double sTotEnergy
- double sPressure
- double ssPotEnergy
- double ssKinEnergy
- double ssTotEnergy
- double ssPressure
- int initUcell [2+1]
- int moreCycles
- int nAtom
- int stepAvg
- int stepCount
- int stepEquil
- int stepLimit
- int stepTraj
- int stepDump
- double RadiusIJ
- double SqrRadiusIJ
- double RadiusIJInv
- int nAtomType
- int * atomType
- int * atomID
- double * atomRadius
- double * atomMass
- double TotalMass
- int nBond
- int nBondType

- int * [atom1](#)
- int * [atom2](#)
- int * [BondID](#)
- int * [BondType](#)
- double * [kb](#)
- double * [ro](#)
- double * [BondEnergy](#)
- double * [BondLength](#)
- double [TotalBondEnergy](#)
- double [BondEnergyPerAtom](#)
- double [gamman](#)
- double * [discDragx](#)
- double * [discDragy](#)
- double * [nodeDragx](#)
- double * [nodeDragy](#)
- double [strain](#)
- double [strainRate](#)
- double [shearDisplacement](#)
- double [shearVelocity](#)
- double [VSqr](#)
- double [VMeanSqr](#)
- double [VRootMeanSqr](#)
- double [ComX](#)
- double [ComY](#)
- double [ComX0](#)
- double [ComY0](#)
- double [ComXRatio](#)
- double [ComYRatio](#)
- double [HaltCondition](#)
- double [DeltaY](#)
- double [DeltaX](#)
- int * [ImageX](#)
- int * [ImageY](#)
- double * [rxUnwrap](#)
- double * [ryUnwrap](#)
- int [nAtomInterface](#)
- int [nDiscInterface](#)
- int [nAtomBlock](#)
- int * [atomIDInterface](#)
- double [Kn](#)
- double [fx](#)
- double [fy](#)
- double [FyBylx](#)
- double [fxByfy](#)
- int [DampFlag](#)
- double [strech](#)
- int [dumpPairFlag](#)
- int [nPairTotal](#)
- int [nPairActive](#)
- int * [PairID](#)
- int * [Pairatom1](#)
- int * [Pairatom2](#)
- double * [PairXij](#)
- double * [PairYij](#)
- char [solver](#) [128]

- char [xBoundary](#) [10]
- char [yBoundary](#) [10]
- char [thermo](#)
- double * [DeltaXijOld](#)
- double * [DeltaYijOld](#)
- double [DeltaXijNew](#)
- double [DeltaYijNew](#)
- double [DeltaXij](#)
- double [DeltaYij](#)
- double [DeltaVXij](#)
- double [DeltaVYij](#)
- double ** [DeltaXijOldPair](#)
- double ** [DeltaYijOldPair](#)
- int * [molID](#)
- int ** [isBonded](#)
- int * [cellList](#)
- int [cells](#) [2+1]
- int [rank](#)
- int [size](#)
- int [master](#)
- double * [fax](#)
- double * [fay](#)
- double [fuSum](#)
- double [fvirSum](#)
- double [frfAtom](#)
- double [uSumPair](#)
- double [uSumPairPerAtom](#)
- double [virSumPair](#)
- double [virSumPairxx](#)
- double [virSumPairyy](#)
- double [virSumPairxy](#)
- double [virSumBond](#)
- double [virSumBondxx](#)
- double [virSumBondyy](#)
- double [virSumBondxy](#)
- double [virSumxx](#)
- double [virSumyy](#)
- double [virSumxy](#)
- int [freezeAtomType](#)
- double ** [cfOrg](#)
- double ** [spacetimeCorr](#)
- double * [cfVal](#)
- double * [spacetimeCorrAv](#)
- int * [indexCorr](#)
- int [countCorrAv](#)
- int [limitCorrAv](#)
- int [nBuffCorr](#)
- int [nFunCorr](#)
- int [nValCorr](#)
- int [stepCorr](#)
- double [rfAtom](#)
- double * [indexAcf](#)
- double ** [viscAcf](#)
- double * [viscAcfOrg](#)
- double * [viscAcfAv](#)

- double [viscAcfInt](#)
- int [nValAcf](#)
- int [nBuffAcf](#)
- int [stepAcf](#)
- int [countAcfAv](#)
- int [limitAcfAv](#)
- double * [histRdf](#)
- double [rangeRdf](#)
- int [countRdf](#)
- int [limitRdf](#)
- int [sizeHistRdf](#)
- int [stepRdf](#)
- char * [prefix](#)
- char [result](#) [250]
- FILE * [fpresult](#)
- char [xyz](#) [256]
- FILE * [fpxyz](#)
- char [bond](#) [256]
- FILE * [fpbond](#)
- char [dump](#) [256]
- FILE * [fpdump](#)
- char [dnsty](#) [256]
- FILE * [fpdnsty](#)
- char [visc](#) [256]
- FILE * [fpvisc](#)
- char [rdf](#) [256]
- FILE * [fprdf](#)
- char [vrms](#) [256]
- FILE * [fpvrms](#)
- char [stress](#) [256]
- FILE * [fpstress](#)
- char [momentum](#) [256]
- FILE * [fpmomentum](#)
- char [com](#) [256]
- FILE * [fpcom](#)
- char [pair](#) [256]
- FILE * [fppair](#)

3.58.1 Macro Definition Documentation

3.58.1.1 EXTERN

```
#define EXTERN extern
```

Definition at line 8 of file [global.h](#).

3.58.1.2 NDIM

```
#define NDIM 2
```

Definition at line 13 of file [global.h](#).

Referenced by [ComputeBondForce\(\)](#), [ComputeForcesCells\(\)](#), [ComputePairForce\(\)](#), [EvalProps\(\)](#), [EvalRdf\(\)](#), [EvalSpacetimeCorr\(\)](#), and [LeapfrogStep\(\)](#).

3.58.1.3 SignR

```
#define SignR(  
    x,  
    y ) (( (y) >= 0) ? (x) : (- (x)))
```

Definition at line 15 of file [global.h](#).

Referenced by [EvalRdf\(\)](#), and [LeapfrogStep\(\)](#).

3.58.1.4 Sqr

```
#define Sqr(  
    x ) ((x) * (x))
```

Definition at line 14 of file [global.h](#).

Referenced by [AccumProps\(\)](#), [BrownianStep\(\)](#), [ComputeBondForce\(\)](#), [ComputeForcesCells\(\)](#), [ComputePairForce\(\)](#), [EvalProps\(\)](#), [EvalRdf\(\)](#), [EvalVrms\(\)](#), and [LeapfrogStep\(\)](#).

3.58.2 Typedef Documentation

3.58.2.1 real

```
typedef double real
```

Definition at line 11 of file [global.h](#).

3.58.3 Variable Documentation

3.58.3.1 atom1

```
int* atom1 [extern]
```

Referenced by [Close\(\)](#), [ComputeBondForce\(\)](#), [DumpBonds\(\)](#), [DumpRestart\(\)](#), and [Init\(\)](#).

3.58.3.2 atom2

```
int * atom2
```

Definition at line 34 of file [global.h](#).

Referenced by [Close\(\)](#), [ComputeBondForce\(\)](#), [DumpBonds\(\)](#), [DumpRestart\(\)](#), and [Init\(\)](#).

3.58.3.3 atomID

```
int* atomID [extern]
```

Referenced by [Close\(\)](#), [DumpRestart\(\)](#), [DumpState\(\)](#), [Init\(\)](#), and [Trajectory\(\)](#).

3.58.3.4 atomIDInterface

```
int* atomIDInterface [extern]
```

Referenced by [Close\(\)](#), [ComputePairForce\(\)](#), and [Init\(\)](#).

3.58.3.5 atomMass

```
double* atomMass [extern]
```

Referenced by [Close\(\)](#), [ComputePairForce\(\)](#), [EvalCom\(\)](#), and [Init\(\)](#).

3.58.3.6 atomRadius

```
double* atomRadius [extern]
```

Referenced by [ApplyBoundaryCond\(\)](#), [Close\(\)](#), [ComputeForcesCells\(\)](#), [ComputePairForce\(\)](#), [DumpRestart\(\)](#), [DumpState\(\)](#), [Init\(\)](#), and [Trajectory\(\)](#).

3.58.3.7 atomType

```
int* atomType [extern]
```

Referenced by [ApplyDrivingForce\(\)](#), [Close\(\)](#), [DumpRestart\(\)](#), [DumpState\(\)](#), [Init\(\)](#), [Trajectory\(\)](#), and [VelocityVerletStep\(\)](#).

3.58.3.8 ax

```
double * ax
```

Definition at line 17 of file [global.h](#).

Referenced by [ApplyDrivingForce\(\)](#), [ApplyForce\(\)](#), [ApplyViscous\(\)](#), [BrownianStep\(\)](#), [Close\(\)](#), [ComputeBondForce\(\)](#), [ComputeForcesCells\(\)](#), [ComputePairForce\(\)](#), [DumpState\(\)](#), [EvalProps\(\)](#), [EvalVacf\(\)](#), [Init\(\)](#), [LeapfrogStep\(\)](#), [Trajectory\(\)](#), and [VelocityVerletStep\(\)](#).

3.58.3.9 ay

```
double * ay
```

Definition at line 17 of file [global.h](#).

Referenced by [ApplyDrivingForce\(\)](#), [ApplyForce\(\)](#), [ApplyViscous\(\)](#), [BrownianStep\(\)](#), [Close\(\)](#), [ComputeBondForce\(\)](#), [ComputeForcesCells\(\)](#), [ComputePairForce\(\)](#), [DumpState\(\)](#), [EvalProps\(\)](#), [EvalVacf\(\)](#), [Init\(\)](#), [LeapfrogStep\(\)](#), [Trajectory\(\)](#), and [VelocityVerletStep\(\)](#).

3.58.3.10 bond

```
char bond[256] [extern]
```

Referenced by [main\(\)](#).

3.58.3.11 BondEnergy

```
double* BondEnergy [extern]
```

Referenced by [ComputeBondForce\(\)](#), and [Init\(\)](#).

3.58.3.12 BondEnergyPerAtom

```
double BondEnergyPerAtom
```

Definition at line 38 of file [global.h](#).

Referenced by [EvalProps\(\)](#), and [PrintSummary\(\)](#).

3.58.3.13 BondID

```
int* BondID [extern]
```

Referenced by [Close\(\)](#), [DumpBonds\(\)](#), [DumpRestart\(\)](#), and [Init\(\)](#).

3.58.3.14 BondLength

```
double * BondLength
```

Definition at line 37 of file [global.h](#).

Referenced by [ComputeBondForce\(\)](#), [DumpBonds\(\)](#), and [Init\(\)](#).

3.58.3.15 BondType

```
int * BondType
```

Definition at line 35 of file [global.h](#).

Referenced by [Close\(\)](#), [DumpBonds\(\)](#), [DumpRestart\(\)](#), and [Init\(\)](#).

3.58.3.16 cellList

```
int* cellList [extern]
```

Referenced by [Close\(\)](#), [ComputeForcesCells\(\)](#), and [Init\(\)](#).

3.58.3.17 cells

```
int cells[2+1]
```

Definition at line 78 of file [global.h](#).

Referenced by [ComputeForcesCells\(\)](#), and [Init\(\)](#).

3.58.3.18 cfOrg

```
double** cfOrg [extern]
```

Referenced by [AllocArrays\(\)](#), [Close\(\)](#), and [EvalSpacetimeCorr\(\)](#).

3.58.3.19 cfVal

```
double * cfVal
```

Definition at line 89 of file [global.h](#).

Referenced by [AllocArrays\(\)](#), [Close\(\)](#), and [EvalSpacetimeCorr\(\)](#).

3.58.3.20 com

```
char com[256] [extern]
```

Referenced by [main\(\)](#).

3.58.3.21 ComX

```
double ComX [extern]
```

Referenced by [EvalCom\(\)](#), and [PrintCom\(\)](#).

3.58.3.22 ComX0

```
double ComX0
```

Definition at line 44 of file [global.h](#).

Referenced by [EvalCom\(\)](#).

3.58.3.23 ComXRatio

```
double ComXRatio
```

Definition at line 44 of file [global.h](#).

Referenced by [EvalCom\(\)](#).

3.58.3.24 ComY

```
double ComY
```

Definition at line 44 of file [global.h](#).

Referenced by [EvalCom\(\)](#), and [PrintCom\(\)](#).

3.58.3.25 ComY0

```
double ComY0
```

Definition at line 44 of file [global.h](#).

Referenced by [EvalCom\(\)](#).

3.58.3.26 ComYRatio

```
double ComYRatio
```

Definition at line 44 of file [global.h](#).

Referenced by [EvalCom\(\)](#).

3.58.3.27 countAcfAv

```
int countAcfAv
```

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

Referenced by [AccumVacf\(\)](#), and [ZeroVacf\(\)](#).

3.58.3.28 countCorrAv

```
int countCorrAv
```

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

Referenced by [EvalSpacetimeCorr\(\)](#), and [SetupJob\(\)](#).

3.58.3.29 countRdf

```
int countRdf [extern]
```

Referenced by [EvalRdf\(\)](#), and [SetupJob\(\)](#).

3.58.3.30 DampFlag

```
int DampFlag [extern]
```

Referenced by [ComputeBondForce\(\)](#), [ComputePairForce\(\)](#), and [Init\(\)](#).

3.58.3.31 deltaT

```
double deltaT
```

Definition at line 20 of file [global.h](#).

Referenced by [AccumVacf\(\)](#), [BrownianStep\(\)](#), [ComputeBondForce\(\)](#), [ComputePairForce\(\)](#), [EvalProps\(\)](#), [EvalSpacetimeCorr\(\)](#), [EvalVacf\(\)](#), [Init\(\)](#), [LeapfrogStep\(\)](#), [main\(\)](#), [PrintVacf\(\)](#), and [VelocityVerletStep\(\)](#).

3.58.3.32 DeltaVXi*j*

```
double DeltaVXij
```

Definition at line 70 of file [global.h](#).

Referenced by [ComputeBondForce\(\)](#), and [ComputePairForce\(\)](#).

3.58.3.33 DeltaVY*ij*

```
double DeltaVYij
```

Definition at line 70 of file [global.h](#).

Referenced by [ComputeBondForce\(\)](#), and [ComputePairForce\(\)](#).

3.58.3.34 DeltaX

```
double DeltaX
```

Definition at line 46 of file [global.h](#).

Referenced by [DisplaceAtoms\(\)](#), and [Init\(\)](#).

3.58.3.35 DeltaXi*j*

```
double DeltaXij [extern]
```

Referenced by [ComputeBondForce\(\)](#), and [ComputePairForce\(\)](#).

3.58.3.36 DeltaXijNew

```
double DeltaXijNew [extern]
```

Referenced by [ComputeBondForce\(\)](#), and [ComputePairForce\(\)](#).

3.58.3.37 DeltaXijOld

```
double* DeltaXijOld [extern]
```

Referenced by [Close\(\)](#), [ComputeBondForce\(\)](#), and [Init\(\)](#).

3.58.3.38 DeltaXijOldPair

```
double** DeltaXijOldPair [extern]
```

Referenced by [Close\(\)](#), [ComputePairForce\(\)](#), and [Init\(\)](#).

3.58.3.39 DeltaY

```
double DeltaY [extern]
```

Referenced by [DisplaceAtoms\(\)](#), and [Init\(\)](#).

3.58.3.40 DeltaYij

```
double DeltaYij
```

Definition at line 70 of file [global.h](#).

Referenced by [ComputeBondForce\(\)](#), and [ComputePairForce\(\)](#).

3.58.3.41 DeltaYijNew

```
double DeltaYijNew
```

Definition at line 69 of file [global.h](#).

Referenced by [ComputeBondForce\(\)](#), and [ComputePairForce\(\)](#).

3.58.3.42 DeltaYijOld

```
double * DeltaYijOld
```

Definition at line 68 of file [global.h](#).

Referenced by [Close\(\)](#), [ComputeBondForce\(\)](#), and [Init\(\)](#).

3.58.3.43 DeltaYijOldPair

```
double ** DeltaYijOldPair
```

Definition at line 71 of file [global.h](#).

Referenced by [Close\(\)](#), [ComputePairForce\(\)](#), and [Init\(\)](#).

3.58.3.44 density

```
double density
```

Definition at line 21 of file [global.h](#).

Referenced by [EvalProps\(\)](#), and [Init\(\)](#).

3.58.3.45 discDragx

```
double* discDragx [extern]
```

Referenced by [ComputeForcesCells\(\)](#), [ComputePairForce\(\)](#), [DumpPairs\(\)](#), and [Init\(\)](#).

3.58.3.46 discDragy

```
double * discDragy
```

Definition at line 40 of file [global.h](#).

Referenced by [ComputeForcesCells\(\)](#), [ComputePairForce\(\)](#), [DumpPairs\(\)](#), and [Init\(\)](#).

3.58.3.47 dnsty

```
char dnsty[256] [extern]
```

3.58.3.48 dump

```
char dump[256] [extern]
```

3.58.3.49 dumpPairFlag

```
int dumpPairFlag [extern]
```

3.58.3.50 fax

```
double* fax [extern]
```

Referenced by [Close\(\)](#), and [Init\(\)](#).

3.58.3.51 fay

```
double * fay
```

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

Referenced by [Close\(\)](#), and [Init\(\)](#).

3.58.3.52 fpbond

```
FILE* fpbond [extern]
```

Referenced by [DumpBonds\(\)](#), and [main\(\)](#).

3.58.3.53 fpcom

```
FILE* fpcom [extern]
```

Referenced by [Init\(\)](#), [main\(\)](#), and [PrintCom\(\)](#).

3.58.3.54 fpdnsty

```
FILE* fpdnsty [extern]
```

Referenced by [EvalSpacetimeCorr\(\)](#).

3.58.3.55 fpdump

```
FILE* fpdump [extern]
```

3.58.3.56 fpmomentum

```
FILE* fpmomentum [extern]
```

Referenced by [PrintMomentum\(\)](#).

3.58.3.57 fppair

```
FILE* fppair [extern]
```

Referenced by [DumpPairs\(\)](#), and [main\(\)](#).

3.58.3.58 fprdf

```
FILE* fprdf [extern]
```

Referenced by [EvalRdf\(\)](#).

3.58.3.59 fpresult

FILE* fpresult [extern]

Referenced by [ApplyBoundaryCond\(\)](#), [HaltConditionCheck\(\)](#), [Init\(\)](#), [main\(\)](#), and [PrintSummary\(\)](#).

3.58.3.60 fpstress

FILE* fpstress [extern]

Referenced by [PrintStress\(\)](#).

3.58.3.61 fpvisc

FILE* fpvisc [extern]

Referenced by [PrintVacf\(\)](#).

3.58.3.62 fpvrms

FILE* fpvrms [extern]

Referenced by [Init\(\)](#), [main\(\)](#), and [PrintVrms\(\)](#).

3.58.3.63 fpxyz

FILE* fpxyz [extern]

Referenced by [main\(\)](#), and [Trajectory\(\)](#).

3.58.3.64 freezeAtomType

int freezeAtomType [extern]

Referenced by [Init\(\)](#), and [VelocityVerletStep\(\)](#).

3.58.3.65 frfAtom

double frfAtom

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

3.58.3.66 fuSum

double fuSum

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

3.58.3.67 fvirSum

```
double fvirSum
```

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

3.58.3.68 fx

```
double fx [extern]
```

Referenced by [ApplyForce\(\)](#).

3.58.3.69 fxByfy

```
double fxByfy
```

Definition at line 52 of file [global.h](#).

Referenced by [ApplyForce\(\)](#), and [Init\(\)](#).

3.58.3.70 fy

```
double fy
```

Definition at line 52 of file [global.h](#).

Referenced by [ApplyForce\(\)](#).

3.58.3.71 FyBylx

```
double FyBylx
```

Definition at line 52 of file [global.h](#).

Referenced by [ApplyForce\(\)](#), and [Init\(\)](#).

3.58.3.72 gamman

```
double gamman [extern]
```

Referenced by [ComputeBondForce\(\)](#), [ComputeForcesCells\(\)](#), [ComputePairForce\(\)](#), and [Init\(\)](#).

3.58.3.73 HaltCondition

```
double HaltCondition [extern]
```

Referenced by [HaltConditionCheck\(\)](#), and [Init\(\)](#).

3.58.3.74 histRdf

```
double* histRdf [extern]
```

Referenced by [AllocArrays\(\)](#), and [EvalRdf\(\)](#).

3.58.3.75 ImageX

```
int* ImageX [extern]
```

Referenced by [Close\(\)](#), [EvalUnwrap\(\)](#), [Init\(\)](#), and [VelocityVerletStep\(\)](#).

3.58.3.76 ImageY

```
int * ImageY
```

Definition at line 47 of file [global.h](#).

Referenced by [Close\(\)](#), [EvalUnwrap\(\)](#), [Init\(\)](#), and [VelocityVerletStep\(\)](#).

3.58.3.77 indexAcf

```
double* indexAcf [extern]
```

Referenced by [AccumVacf\(\)](#), [AllocArrays\(\)](#), [Close\(\)](#), [EvalVacf\(\)](#), and [InitVacf\(\)](#).

3.58.3.78 indexCorr

```
int* indexCorr [extern]
```

Referenced by [AllocArrays\(\)](#), [Close\(\)](#), [EvalSpacetimeCorr\(\)](#), and [SetupJob\(\)](#).

3.58.3.79 initUcell

```
int initUcell[2+1] [extern]
```

3.58.3.80 isBonded

```
int** isBonded [extern]
```

Referenced by [Close\(\)](#), [ComputePairForce\(\)](#), and [Init\(\)](#).

3.58.3.81 kappa

```
double kappa
```

Definition at line 21 of file [global.h](#).

Referenced by [Init\(\)](#), and [LeapfrogStep\(\)](#).

3.58.3.82 kb

```
double* kb [extern]
```

Referenced by [Close\(\)](#), [ComputeBondForce\(\)](#), [DumpRestart\(\)](#), and [Init\(\)](#).

3.58.3.83 kinEnergy

```
double kinEnergy
```

Definition at line 20 of file [global.h](#).

Referenced by [AccumProps\(\)](#), [AccumVacf\(\)](#), [EvalProps\(\)](#), and [PrintSummary\(\)](#).

3.58.3.84 Kn

```
double Kn [extern]
```

Referenced by [ComputePairForce\(\)](#).

3.58.3.85 limitAcfAv

```
int limitAcfAv
```

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

Referenced by [AccumVacf\(\)](#), and [Init\(\)](#).

3.58.3.86 limitCorrAv

```
int limitCorrAv
```

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

Referenced by [EvalSpacetimeCorr\(\)](#), and [Init\(\)](#).

3.58.3.87 limitRdf

```
int limitRdf
```

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

Referenced by [EvalRdf\(\)](#), and [Init\(\)](#).

3.58.3.88 master

```
int master
```

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

3.58.3.89 molID

```
int* molID [extern]
```

Referenced by [ApplyForce\(\)](#), [Close\(\)](#), [DisplaceAtoms\(\)](#), [DumpRestart\(\)](#), [DumpState\(\)](#), [EvalCom\(\)](#), [Init\(\)](#), and [Trajectory\(\)](#).

3.58.3.90 momentum

```
char momentum[256] [extern]
```

3.58.3.91 moreCycles

```
int moreCycles
```

Definition at line 24 of file [global.h](#).

Referenced by [main\(\)](#).

3.58.3.92 nAtom

```
int nAtom
```

Definition at line 24 of file [global.h](#).

Referenced by [ApplyBoundaryCond\(\)](#), [ApplyDrivingForce\(\)](#), [ApplyForce\(\)](#), [ApplyLeesEdwardsBoundaryCond\(\)](#), [ApplyShear\(\)](#), [ApplyViscous\(\)](#), [BrownianStep\(\)](#), [Close\(\)](#), [ComputeBondForce\(\)](#), [ComputeForcesCells\(\)](#), [ComputePairForce\(\)](#), [DisplaceAtoms\(\)](#), [DumpRestart\(\)](#), [DumpState\(\)](#), [EvalCom\(\)](#), [EvalProps\(\)](#), [EvalRdf\(\)](#), [EvalSpacetimeCorr\(\)](#), [EvalUnwrap\(\)](#), [EvalVacf\(\)](#), [EvalVrms\(\)](#), [Init\(\)](#), [LeapfrogStep\(\)](#), [Trajectory\(\)](#), and [VelocityVerletStep\(\)](#).

3.58.3.93 nAtomBlock

```
int nAtomBlock
```

Definition at line 49 of file [global.h](#).

Referenced by [ApplyForce\(\)](#), and [Init\(\)](#).

3.58.3.94 nAtomInterface

```
int nAtomInterface [extern]
```

Referenced by [ComputePairForce\(\)](#), and [Init\(\)](#).

3.58.3.95 nAtomType

```
int nAtomType [extern]
```

Referenced by [DumpRestart\(\)](#), and [Init\(\)](#).

3.58.3.96 nBond

```
int nBond [extern]
```

Referenced by [ComputeBondForce\(\)](#), [DumpBonds\(\)](#), [DumpRestart\(\)](#), and [Init\(\)](#).

3.58.3.97 nBondType

```
int nBondType
```

Definition at line 33 of file [global.h](#).

Referenced by [DumpRestart\(\)](#), and [Init\(\)](#).

3.58.3.98 nBuffAcf

```
int nBuffAcf
```

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

Referenced by [AccumVacf\(\)](#), [AllocArrays\(\)](#), [Close\(\)](#), [EvalVacf\(\)](#), [Init\(\)](#), and [InitVacf\(\)](#).

3.58.3.99 nBuffCorr

```
int nBuffCorr
```

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

Referenced by [AllocArrays\(\)](#), [Close\(\)](#), [EvalSpacetimeCorr\(\)](#), [Init\(\)](#), and [SetupJob\(\)](#).

3.58.3.100 nDiscInterface

```
int nDiscInterface
```

Definition at line 49 of file [global.h](#).

Referenced by [Init\(\)](#).

3.58.3.101 nFunCorr

```
int nFunCorr
```

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

Referenced by [AllocArrays\(\)](#), [EvalSpacetimeCorr\(\)](#), [Init\(\)](#), and [SetupJob\(\)](#).

3.58.3.102 nodeDragx

```
double * nodeDragx
```

Definition at line 40 of file [global.h](#).

Referenced by [ComputeBondForce\(\)](#), [DumpBonds\(\)](#), and [Init\(\)](#).

3.58.3.103 nodeDragy

```
double * nodeDragy
```

Definition at line 40 of file [global.h](#).

Referenced by [ComputeBondForce\(\)](#), [DumpBonds\(\)](#), and [Init\(\)](#).

3.58.3.104 nPairActive

```
int nPairActive
```

Definition at line 58 of file [global.h](#).

Referenced by [ComputePairForce\(\)](#), and [DumpPairs\(\)](#).

3.58.3.105 nPairTotal

```
int nPairTotal [extern]
```

Referenced by [ComputePairForce\(\)](#), and [Init\(\)](#).

3.58.3.106 nValAcf

```
int nValAcf [extern]
```

Referenced by [AccumVacf\(\)](#), [AllocArrays\(\)](#), [Init\(\)](#), [InitVacf\(\)](#), [PrintVacf\(\)](#), and [ZeroVacf\(\)](#).

3.58.3.107 nValCorr

```
int nValCorr
```

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

Referenced by [AllocArrays\(\)](#), [EvalSpacetimeCorr\(\)](#), [Init\(\)](#), and [SetupJob\(\)](#).

3.58.3.108 pair

```
char pair[256] [extern]
```

Referenced by [main\(\)](#).

3.58.3.109 Pairatom1

```
int * Pairatom1
```

Definition at line 59 of file [global.h](#).

Referenced by [Close\(\)](#), [ComputePairForce\(\)](#), [DumpPairs\(\)](#), and [Init\(\)](#).

3.58.3.110 Pairatom2

```
int * Pairatom2
```

Definition at line 59 of file [global.h](#).

Referenced by [Close\(\)](#), [ComputePairForce\(\)](#), [DumpPairs\(\)](#), and [Init\(\)](#).

3.58.3.111 PairID

```
int* PairID [extern]
```

Referenced by [Close\(\)](#), [ComputePairForce\(\)](#), [DumpPairs\(\)](#), and [Init\(\)](#).

3.58.3.112 PairXij

```
double* PairXij [extern]
```

Referenced by [Close\(\)](#), [ComputePairForce\(\)](#), [DumpPairs\(\)](#), and [Init\(\)](#).

3.58.3.113 PairYij

```
double * PairYij
```

Definition at line 60 of file [global.h](#).

Referenced by [Close\(\)](#), [ComputePairForce\(\)](#), [DumpPairs\(\)](#), and [Init\(\)](#).

3.58.3.114 potEnergy

```
double potEnergy
```

Definition at line 20 of file [global.h](#).

Referenced by [AccumProps\(\)](#), [EvalProps\(\)](#), and [PrintSummary\(\)](#).

3.58.3.115 prefix

```
char* prefix [extern]
```

Definition at line 13 of file [main.c](#).

Referenced by [DumpRestart\(\)](#), and [DumpState\(\)](#).

3.58.3.116 pressure

```
double pressure
```

Definition at line 21 of file [global.h](#).

Referenced by [AccumProps\(\)](#), [EvalProps\(\)](#), [PrintStress\(\)](#), and [PrintSummary\(\)](#).

3.58.3.117 RadiusIJ

```
double RadiusIJ [extern]
```

Referenced by [ComputeForcesCells\(\)](#), and [ComputePairForce\(\)](#).

3.58.3.118 RadiusIJInv

```
double RadiusIJInv
```

Definition at line 26 of file [global.h](#).

Referenced by [ComputeForcesCells\(\)](#), and [ComputePairForce\(\)](#).

3.58.3.119 rangeRdf

```
double rangeRdf
```

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

Referenced by [EvalRdf\(\)](#), and [Init\(\)](#).

3.58.3.120 rank

```
int rank [extern]
```

Referenced by [ComputeForcesCells\(\)](#).

3.58.3.121 rCut

```
double rCut
```

Definition at line 21 of file [global.h](#).

Referenced by [Init\(\)](#), and [LeapfrogStep\(\)](#).

3.58.3.122 rdf

```
char rdf[256] [extern]
```

3.58.3.123 region

```
double region[2+1] [extern]
```

Referenced by [AccumVacf\(\)](#), [ApplyBoundaryCond\(\)](#), [ApplyLeesEdwardsBoundaryCond\(\)](#), [ComputeBondForce\(\)](#), [ComputeForcesCells\(\)](#), [ComputePairForce\(\)](#), [DumpRestart\(\)](#), [EvalRdf\(\)](#), [EvalSpacetimeCorr\(\)](#), [EvalUnwrap\(\)](#), [Init\(\)](#), [LeapfrogStep\(\)](#), and [VelocityVerletStep\(\)](#).

3.58.3.124 regionH

```
double regionH[2+1]
```

Definition at line 20 of file [global.h](#).

Referenced by [ApplyBoundaryCond\(\)](#), [ApplyForce\(\)](#), [ApplyLeesEdwardsBoundaryCond\(\)](#), [ComputeBondForce\(\)](#), [ComputeForcesCells\(\)](#), [ComputePairForce\(\)](#), [DumpBonds\(\)](#), [DumpPairs\(\)](#), [DumpState\(\)](#), [EvalRdf\(\)](#), [Init\(\)](#), [LeapfrogStep\(\)](#), [Trajectory\(\)](#), and [VelocityVerletStep\(\)](#).

3.58.3.125 result

```
char result[250] [extern]
```

Referenced by [main\(\)](#).

3.58.3.126 rfAtom

```
double rfAtom [extern]
```

Referenced by [ComputeForcesCells\(\)](#), and [EvalVacf\(\)](#).

3.58.3.127 ro

```
double * ro
```

Definition at line 36 of file [global.h](#).

Referenced by [Close\(\)](#), [ComputeBondForce\(\)](#), [DumpBonds\(\)](#), [DumpRestart\(\)](#), and [Init\(\)](#).

3.58.3.128 rx

```
double* rx [extern]
```

Referenced by [ApplyBoundaryCond\(\)](#), [ApplyLeesEdwardsBoundaryCond\(\)](#), [ApplyShear\(\)](#), [BrownianStep\(\)](#), [Close\(\)](#), [ComputeBondForce\(\)](#), [ComputeForcesCells\(\)](#), [ComputePairForce\(\)](#), [DisplaceAtoms\(\)](#), [DumpRestart\(\)](#), [DumpState\(\)](#), [EvalRdf\(\)](#), [EvalSpacetimeCorr\(\)](#), [EvalUnwrap\(\)](#), [Init\(\)](#), [LeapfrogStep\(\)](#), [Trajectory\(\)](#), and [VelocityVerletStep\(\)](#).

3.58.3.129 rxUnwrap

```
double* rxUnwrap [extern]
```

Referenced by [Close\(\)](#), [EvalCom\(\)](#), [EvalUnwrap\(\)](#), and [Init\(\)](#).

3.58.3.130 ry

```
double * ry
```

Definition at line 17 of file [global.h](#).

Referenced by [ApplyBoundaryCond\(\)](#), [ApplyLeesEdwardsBoundaryCond\(\)](#), [ApplyShear\(\)](#), [BrownianStep\(\)](#), [Close\(\)](#), [ComputeBondForce\(\)](#), [ComputeForcesCells\(\)](#), [ComputePairForce\(\)](#), [DisplaceAtoms\(\)](#), [DumpRestart\(\)](#), [DumpState\(\)](#), [EvalRdf\(\)](#), [EvalUnwrap\(\)](#), [Init\(\)](#), [LeapfrogStep\(\)](#), [Trajectory\(\)](#), and [VelocityVerletStep\(\)](#).

3.58.3.131 ryUnwrap

```
double * ryUnwrap
```

Definition at line 48 of file [global.h](#).

Referenced by [Close\(\)](#), [EvalCom\(\)](#), [EvalUnwrap\(\)](#), and [Init\(\)](#).

3.58.3.132 shearDisplacement

```
double shearDisplacement [extern]
```

Referenced by [ApplyLeesEdwardsBoundaryCond\(\)](#), [ComputeBondForce\(\)](#), [ComputePairForce\(\)](#), and [Init\(\)](#).

3.58.3.133 shearVelocity

```
double shearVelocity
```

Definition at line 42 of file [global.h](#).

Referenced by [Init\(\)](#).

3.58.3.134 size

```
int size
```

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

Referenced by [ComputeForcesCells\(\)](#).

3.58.3.135 sizeHistRdf

```
int sizeHistRdf
```

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

Referenced by [AllocArrays\(\)](#), [EvalRdf\(\)](#), and [Init\(\)](#).

3.58.3.136 sKinEnergy

```
double sKinEnergy
```

Definition at line 21 of file [global.h](#).

Referenced by [AccumProps\(\)](#).

3.58.3.137 solver

```
char solver[128] [extern]
```

Referenced by [EvalProps\(\)](#), and [Init\(\)](#).

3.58.3.138 spacetimeCorr

```
double ** spacetimeCorr
```

Definition at line 89 of file [global.h](#).

Referenced by [AllocArrays\(\)](#), [Close\(\)](#), and [EvalSpacetimeCorr\(\)](#).

3.58.3.139 spacetimeCorrAv

```
double * spacetimeCorrAv
```

Definition at line 89 of file [global.h](#).

Referenced by [AllocArrays\(\)](#), [Close\(\)](#), [EvalSpacetimeCorr\(\)](#), and [SetupJob\(\)](#).

3.58.3.140 speed

```
double* speed [extern]
```

Referenced by [Close\(\)](#), and [Init\(\)](#).

3.58.3.141 sPotEnergy

```
double sPotEnergy
```

Definition at line 21 of file [global.h](#).

Referenced by [AccumProps\(\)](#).

3.58.3.142 sPressure

```
double sPressure
```

Definition at line 22 of file [global.h](#).

Referenced by [AccumProps\(\)](#).

3.58.3.143 SqrRadiusIJ

```
double SqrRadiusIJ
```

Definition at line 26 of file [global.h](#).

Referenced by [ComputeForcesCells\(\)](#), and [ComputePairForce\(\)](#).

3.58.3.144 ssKinEnergy

```
double ssKinEnergy
```

Definition at line 22 of file [global.h](#).

Referenced by [AccumProps\(\)](#).

3.58.3.145 ssPotEnergy

```
double ssPotEnergy
```

Definition at line 22 of file [global.h](#).

Referenced by [AccumProps\(\)](#).

3.58.3.146 ssPressure

```
double ssPressure
```

Definition at line 22 of file [global.h](#).

Referenced by [AccumProps\(\)](#).

3.58.3.147 ssTotEnergy

```
double ssTotEnergy
```

Definition at line 22 of file [global.h](#).

Referenced by [AccumProps\(\)](#).

3.58.3.148 stepAcf

```
int stepAcf
```

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

Referenced by [AccumVacf\(\)](#), [Init\(\)](#), and [PrintVacf\(\)](#).

3.58.3.149 stepAvg

```
int stepAvg
```

Definition at line 24 of file [global.h](#).

Referenced by [AccumProps\(\)](#), [Init\(\)](#), and [main\(\)](#).

3.58.3.150 stepCorr

```
int stepCorr
```

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

Referenced by [EvalSpacetimeCorr\(\)](#), and [Init\(\)](#).

3.58.3.151 stepCount

```
int stepCount
```

Definition at line 24 of file [global.h](#).

Referenced by [BrownianStep\(\)](#), [ComputeBondForce\(\)](#), [ComputePairForce\(\)](#), [HaltConditionCheck\(\)](#), [LeapfrogStep\(\)](#), [main\(\)](#), and [SetupJob\(\)](#).

3.58.3.152 stepDump

```
int stepDump
```

Definition at line 24 of file [global.h](#).

Referenced by [Init\(\)](#), and [main\(\)](#).

3.58.3.153 stepEquil

```
int stepEquil
```

Definition at line 24 of file [global.h](#).

Referenced by [BrownianStep\(\)](#), [Init\(\)](#), and [LeapfrogStep\(\)](#).

3.58.3.154 stepLimit

```
int stepLimit
```

Definition at line 24 of file [global.h](#).

Referenced by [Init\(\)](#), and [main\(\)](#).

3.58.3.155 stepRdf

```
int stepRdf
```

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

Referenced by [Init\(\)](#).

3.58.3.156 stepTraj

```
int stepTraj
```

Definition at line 24 of file [global.h](#).

Referenced by [Init\(\)](#), and [main\(\)](#).

3.58.3.157 sTotEnergy

```
double sTotEnergy
```

Definition at line 22 of file [global.h](#).

Referenced by [AccumProps\(\)](#).

3.58.3.158 strain

```
double strain [extern]
```

Referenced by [ApplyShear\(\)](#), and [Init\(\)](#).

3.58.3.159 strainRate

```
double strainRate
```

Definition at line 41 of file [global.h](#).

Referenced by [Init\(\)](#).

3.58.3.160 stretch

```
double stretch [extern]
```

Referenced by [ComputeBondForce\(\)](#), and [ComputePairForce\(\)](#).

3.58.3.161 stress

```
char stress[256] [extern]
```

3.58.3.162 svirSum

```
double svirSum
```

Definition at line 21 of file [global.h](#).

Referenced by [AccumProps\(\)](#).

3.58.3.163 thermo

```
char thermo [extern]
```

Referenced by [Init\(\)](#), and [LeapfrogStep\(\)](#).

3.58.3.164 timeNow

```
double timeNow
```

Definition at line 20 of file [global.h](#).

Referenced by [DumpBonds\(\)](#), [DumpPairs\(\)](#), [DumpRestart\(\)](#), [DumpState\(\)](#), [EvalCom\(\)](#), [EvalRdf\(\)](#), [Init\(\)](#), [main\(\)](#), [PrintCom\(\)](#), [PrintMomentum\(\)](#), [PrintStress\(\)](#), [PrintSummary\(\)](#), [PrintVrms\(\)](#), and [Trajectory\(\)](#).

3.58.3.165 TotalBondEnergy

```
double TotalBondEnergy [extern]
```

Referenced by [ComputeBondForce\(\)](#), and [EvalProps\(\)](#).

3.58.3.166 TotalMass

```
double TotalMass
```

Definition at line 31 of file [global.h](#).

Referenced by [EvalCom\(\)](#).

3.58.3.167 totEnergy

```
double totEnergy
```

Definition at line 20 of file [global.h](#).

Referenced by [AccumProps\(\)](#), [EvalProps\(\)](#), and [PrintSummary\(\)](#).

3.58.3.168 uSum

```
double uSum
```

Definition at line 21 of file [global.h](#).

Referenced by [ComputeForcesCells\(\)](#).

3.58.3.169 uSumPair

```
double uSumPair [extern]
```

Referenced by [ComputePairForce\(\)](#), and [EvalProps\(\)](#).

3.58.3.170 uSumPairPerAtom

```
double uSumPairPerAtom
```

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

Referenced by [EvalProps\(\)](#), and [PrintSummary\(\)](#).

3.58.3.171 virSum

```
double virSum
```

Definition at line 21 of file [global.h](#).

Referenced by [AccumProps\(\)](#), [ComputeForcesCells\(\)](#), [EvalProps\(\)](#), and [PrintSummary\(\)](#).

3.58.3.172 virSumBond

```
double virSumBond [extern]
```

Referenced by [ComputeBondForce\(\)](#), and [EvalProps\(\)](#).

3.58.3.173 virSumBondxx

```
double virSumBondxx
```

Definition at line 84 of file [global.h](#).

Referenced by [ComputeBondForce\(\)](#), and [EvalProps\(\)](#).

3.58.3.174 virSumBondxy

```
double virSumBondxy
```

Definition at line 84 of file [global.h](#).

Referenced by [ComputeBondForce\(\)](#), and [EvalProps\(\)](#).

3.58.3.175 virSumBondyy

```
double virSumBondyy
```

Definition at line 84 of file [global.h](#).

Referenced by [ComputeBondForce\(\)](#), and [EvalProps\(\)](#).

3.58.3.176 virSumPair

```
double virSumPair
```

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

Referenced by [ComputePairForce\(\)](#), and [EvalProps\(\)](#).

3.58.3.177 virSumPairxx

```
double virSumPairxx
```

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

Referenced by [ComputePairForce\(\)](#), and [EvalProps\(\)](#).

3.58.3.178 virSumPairxy

```
double virSumPairxy
```

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

Referenced by [ComputePairForce\(\)](#), and [EvalProps\(\)](#).

3.58.3.179 virSumPairyy

```
double virSumPairyy
```

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

Referenced by [ComputePairForce\(\)](#), and [EvalProps\(\)](#).

3.58.3.180 virSumxx

```
double virSumxx [extern]
```

Referenced by [EvalProps\(\)](#), and [PrintStress\(\)](#).

3.58.3.181 virSumxy

```
double virSumxy
```

Definition at line 85 of file [global.h](#).

Referenced by [EvalProps\(\)](#), and [PrintStress\(\)](#).

3.58.3.182 virSumyy

```
double virSumyy
```

Definition at line 85 of file [global.h](#).

Referenced by [EvalProps\(\)](#), and [PrintStress\(\)](#).

3.58.3.183 visc

```
char visc[256] [extern]
```

3.58.3.184 viscAcf

```
double ** viscAcf
```

Definition at line 94 of file [global.h](#).

Referenced by [AccumVacf\(\)](#), [AllocArrays\(\)](#), [Close\(\)](#), and [EvalVacf\(\)](#).

3.58.3.185 viscAcfAv

```
double * viscAcfAv
```

Definition at line 94 of file [global.h](#).

Referenced by [AccumVacf\(\)](#), [AllocArrays\(\)](#), [Close\(\)](#), [PrintVacf\(\)](#), and [ZeroVacf\(\)](#).

3.58.3.186 viscAcfInt

```
double viscAcfInt
```

Definition at line 94 of file [global.h](#).

Referenced by [AccumVacf\(\)](#), and [PrintVacf\(\)](#).

3.58.3.187 viscAcfOrg

```
double * viscAcfOrg
```

Definition at line 94 of file [global.h](#).

Referenced by [AllocArrays\(\)](#), [Close\(\)](#), and [EvalVacf\(\)](#).

3.58.3.188 VMeanSqr

```
double VMeanSqr
```

Definition at line 43 of file [global.h](#).

Referenced by [EvalVrms\(\)](#).

3.58.3.189 vrms

```
char vrms[256] [extern]
```

Referenced by [main\(\)](#).

3.58.3.190 VRootMeanSqr

```
double VRootMeanSqr
```

Definition at line 43 of file [global.h](#).

Referenced by [EvalVrms\(\)](#), [main\(\)](#), and [PrintVrms\(\)](#).

3.58.3.191 VSqr

```
double VSqr [extern]
```

Referenced by [EvalVrms\(\)](#).

3.58.3.192 vSum

```
double vSum
```

Definition at line 21 of file [global.h](#).

Referenced by [EvalProps\(\)](#), and [PrintSummary\(\)](#).

3.58.3.193 vSumX

```
double vSumX
```

Definition at line 21 of file [global.h](#).

Referenced by [EvalProps\(\)](#), and [PrintMomentum\(\)](#).

3.58.3.194 vSumY

```
double vSumY
```

Definition at line 21 of file [global.h](#).

Referenced by [EvalProps\(\)](#), and [PrintMomentum\(\)](#).

3.58.3.195 vvSum

```
double vvSum
```

Definition at line 21 of file [global.h](#).

Referenced by [EvalProps\(\)](#), and [LeapfrogStep\(\)](#).

3.58.3.196 vx

```
double * vx
```

Definition at line 17 of file [global.h](#).

Referenced by [ApplyBoundaryCond\(\)](#), [ApplyDrivingForce\(\)](#), [ApplyViscous\(\)](#), [BrownianStep\(\)](#), [Close\(\)](#), [ComputeBondForce\(\)](#), [ComputeForcesCells\(\)](#), [ComputePairForce\(\)](#), [DumpRestart\(\)](#), [DumpState\(\)](#), [EvalProps\(\)](#), [EvalVacf\(\)](#), [EvalVrms\(\)](#), [Init\(\)](#), [LeapfrogStep\(\)](#), [Trajectory\(\)](#), and [VelocityVerletStep\(\)](#).

3.58.3.197 vy

```
double * vy
```

Definition at line 17 of file [global.h](#).

Referenced by [ApplyBoundaryCond\(\)](#), [ApplyDrivingForce\(\)](#), [ApplyViscous\(\)](#), [BrownianStep\(\)](#), [Close\(\)](#), [ComputeBondForce\(\)](#), [ComputeForcesCells\(\)](#), [ComputePairForce\(\)](#), [DumpRestart\(\)](#), [DumpState\(\)](#), [EvalProps\(\)](#), [EvalVacf\(\)](#), [EvalVrms\(\)](#), [Init\(\)](#), [LeapfrogStep\(\)](#), [Trajectory\(\)](#), and [VelocityVerletStep\(\)](#).

3.58.3.198 xBoundary

```
char xBoundary[10] [extern]
```

Referenced by [ApplyBoundaryCond\(\)](#), and [Init\(\)](#).

3.58.3.199 xyz

```
char xyz[256] [extern]
```

Referenced by [main\(\)](#).

3.58.3.200 yBoundary

```
char yBoundary[10]
```

Definition at line 64 of file [global.h](#).

Referenced by [ApplyBoundaryCond\(\)](#), and [Init\(\)](#).

3.59 global.h

[Go to the documentation of this file.](#)

```
00001 #ifndef GLOBAL_H
00002 #define GLOBAL_H
00003 #include <stdio.h> // Required for FILE*
00004
00005 #ifdef DEFINE_GLOBALS
00006     #define EXTERN
00007 #else
00008     #define EXTERN extern
00009 #endif
00010
00011 typedef double real;
00012
00013 #define NDIM 2
00014 #define Sqr(x) ((x) * (x))
00015 #define SignR(x, y) ((y) >= 0) ? (x) : (- (x))
00016
00017 EXTERN double *rx, *ry, *vx, *vy, *ax, *ay;
00018 EXTERN double *speed;
00019
00020 EXTERN double region[NDIM+1], regionH[NDIM+1], deltaT, timeNow, potEnergy, kinEnergy, totEnergy,
00021 density, pressure, rCut, kappa, uSum, virSum, svirSum, vSum, vSumX, vSumY, vvSum, sPotEnergy,
00022 sKinEnergy,
00023 sTotEnergy, sPressure, ssPotEnergy, ssKinEnergy, ssTotEnergy, ssPressure;
00024 EXTERN int initUcell[NDIM+1], moreCycles, nAtom, stepAvg, stepCount, stepEquil, stepLimit,
00025 stepTraj, stepDump;
00026
00026 EXTERN double RadiusIJ, SqrRadiusIJ, RadiusIJInv;
00027 EXTERN int nAtomType;
00028 EXTERN int *atomType;
00029 EXTERN int *atomID;
00030 EXTERN double *atomRadius;
00031 EXTERN double *atomMass, TotalMass;
00032
00033 EXTERN int nBond, nBondType;
00034 EXTERN int *atom1, *atom2;
00035 EXTERN int *BondID, *BondType ;
00036 EXTERN double *kb, *ro;
00037 EXTERN double *BondEnergy, *BondLength;
00038 EXTERN double TotalBondEnergy, BondEnergyPerAtom;
00039 EXTERN double gamman;
00040 EXTERN double *discDragx, *discDragy, *nodeDragx, *nodeDragy;
00041 EXTERN double strain, strainRate;
00042 EXTERN double shearDisplacement, shearVelocity;
00043 EXTERN double VSqr, VMeanSqr, VRootMeanSqr;
00044 EXTERN double ComX, ComY, ComX0, ComY0, ComXRatio, ComYRatio;
00045 EXTERN double HaltCondition;
00046 EXTERN double DeltaY, DeltaX;
00047 EXTERN int *ImageX, *ImageY;
00048 EXTERN double *rxUnwrap, *ryUnwrap;
00049 EXTERN int nAtomInterface, nDiscInterface, nAtomBlock;
00050 EXTERN int *atomIDInterface;
00051 EXTERN double Kn;
00052 EXTERN double fx, fy, FyBylx, fxByfy;
00053 EXTERN int DampFlag;
```

```
00054 EXTERN double   strech;
00055
00056 //For dumping the pair interaction data
00057 EXTERN int        dumpPairFlag;
00058 EXTERN int        nPairTotal, nPairActive;
00059 EXTERN int        *PairID, *Pairatom1, *Pairatom2;
00060 EXTERN double     *PairXij, *PairYij;
00061
00062
00063 EXTERN char       solver[128];
00064 EXTERN char       xBoundary[10], yBoundary[10];
00065 EXTERN char       thermo;
00066
00067 //For damping as in PRL, 130, 178203 (2023)
00068 EXTERN double     *DeltaXijOld, *DeltaYijOld;
00069 EXTERN double     DeltaXijNew, DeltaYijNew;
00070 EXTERN double     DeltaXij, DeltaYij, DeltaVXij, DeltaVYij;
00071 EXTERN double     **DeltaXijOldPair, **DeltaYijOldPair;
00072
00073 //For molecule-ID as per LAMMPS, helpful!
00074 EXTERN int        *molID;
00075 EXTERN int        **isBonded;
00076
00077 //Following three for MPI only
00078 EXTERN int        *cellList, cells[NDIM+1];
00079 EXTERN int        rank, size, master;
00080 EXTERN double     *fax, *fay, fuSum, fvSum, frfAtom;
00081
00082 //For thermodynamic properties
00083 EXTERN double     uSumPair, uSumPairPerAtom, virSumPair, virSumPairxx, virSumPairyy, virSumPairxy;
00084 EXTERN double     virSumBond, virSumBondxx, virSumBondyy, virSumBondxy;
00085 EXTERN double     virSumxx, virSumyy, virSumxy;
00086 EXTERN int        freezeAtomType;
00087
00088 // Spacetime Correlations
00089 EXTERN double     **cfOrg, **spacetimeCorr, *cfVal, *spacetimeCorrAv;
00090 EXTERN int        *indexCorr, countCorrAv, limitCorrAv, nBuffCorr, nFunCorr, nValCorr, stepCorr;
00091
00092 // Viscosity
00093 EXTERN double     rfAtom, frfAtom;
00094 EXTERN double     *indexAcf, **viscAcf, *viscAcfOrg, *viscAcfAv, viscAcfInt;
00095 EXTERN int        nValAcf, nBuffAcf, stepAcf, countAcfAv, limitAcfAv;
00096
00097 // Radial distribution function
00098 EXTERN double     *histRdf, rangeRdf;
00099 EXTERN int        countRdf, limitRdf, sizeHistRdf, stepRdf;
00100
00101
00102 // Output files prefixes
00103 EXTERN char       *prefix;
00104
00105 EXTERN char       result[250];
00106 EXTERN FILE       *fpresult;
00107
00108 EXTERN char       xyz[256];
00109 EXTERN FILE       *fpxyz;
00110
00111 EXTERN char       bond[256];
00112 EXTERN FILE       *fpbond;
00113
00114
00115 EXTERN char       dump[256];
00116 EXTERN FILE       *fpdump;
00117
00118 EXTERN char       dnsty[256];
00119 EXTERN FILE       *fpdnsty;
00120
00121 EXTERN char       visc[256];
00122 EXTERN FILE       *fpvisc;
00123
00124 EXTERN char       rdf[256];
00125 EXTERN FILE       *fprdf;
00126
00127 EXTERN char       vrms[256];
00128 EXTERN FILE       *fpvrms;
00129
00130 EXTERN char       stress[256];
00131 EXTERN FILE       *fpstress;
00132
00133 EXTERN char       momentum[256];
00134 EXTERN FILE       *fpmomentum;
00135
00136 EXTERN char       com[256];
00137 EXTERN FILE       *fpcom;
00138
00139 EXTERN char       pair[256];
00140 EXTERN FILE       *fppair;
```

```
00141
00142 #endif // GLOBALEXTERN_H
```

3.60 source/globalExtern.h File Reference

Macros

- #define `NDIM` 2
- #define `Sqr(x)` ((x) * (x))
- #define `SignR(x, y)` (((y) >= 0) ? (x) : (- (x)))

Typedefs

- typedef double `real`

Variables

- double * `rx`
- double * `ry`
- double * `vx`
- double * `vy`
- double * `ax`
- double * `ay`
- double * `speed`
- double `region` [2+1]
- double `regionH` [2+1]
- double `deltaT`
- double `timeNow`
- double `potEnergy`
- double `kinEnergy`
- double `totEnergy`
- double `density`
- double `pressure`
- double `rCut`
- double `kappa`
- double `uSum`
- double `virSum`
- double `svirSum`
- double `vSum`
- double `vSumX`
- double `vSumY`
- double `vvSum`
- double `sPotEnergy`
- double `sKinEnergy`
- double `sTotEnergy`
- double `sPressure`
- double `ssPotEnergy`
- double `ssKinEnergy`
- double `ssTotEnergy`
- double `ssPressure`
- int `initUcell` [2+1]

- int [moreCycles](#)
- int [nAtom](#)
- int [stepAvg](#)
- int [stepCount](#)
- int [stepEquil](#)
- int [stepLimit](#)
- int [stepTraj](#)
- int [stepDump](#)
- double [RadiusIJ](#)
- double [SqrRadiusIJ](#)
- double [RadiusIJInv](#)
- int [nAtomType](#)
- int * [atomType](#)
- int * [atomID](#)
- double * [atomRadius](#)
- double * [atomMass](#)
- double [TotalMass](#)
- int [nBond](#)
- int [nBondType](#)
- int * [atom1](#)
- int * [atom2](#)
- int * [BondID](#)
- int * [BondType](#)
- double * [kb](#)
- double * [ro](#)
- double * [BondEnergy](#)
- double * [BondLength](#)
- double [TotalBondEnergy](#)
- double [BondEnergyPerAtom](#)
- double [gamman](#)
- double * [discDragx](#)
- double * [discDragy](#)
- double * [nodeDragx](#)
- double * [nodeDragy](#)
- double [strain](#)
- double [strainRate](#)
- double [shearDisplacement](#)
- double [shearVelocity](#)
- double [VSqr](#)
- double [VMeanSqr](#)
- double [VRootMeanSqr](#)
- double [ComX](#)
- double [ComY](#)
- double [ComX0](#)
- double [ComY0](#)
- double [ComXRatio](#)
- double [ComYRatio](#)
- double [HaltCondition](#)
- double [DeltaY](#)
- double [DeltaX](#)
- int * [ImageX](#)
- int * [ImageY](#)
- double * [rxUnwrap](#)
- double * [ryUnwrap](#)
- int [nAtomInterface](#)

- int `nDiscInterface`
- int `nAtomBlock`
- int * `atomIDInterface`
- double `Kn`
- double `fx`
- double `fy`
- double `FyBylx`
- double `fxByfy`
- int `DampFlag`
- double `strech`
- int `dumpPairFlag`
- int `nPairTotal`
- int `nPairActive`
- int * `PairID`
- int * `Pairatom1`
- int * `Pairatom2`
- double * `PairXij`
- double * `PairYij`
- char `solver` [128]
- char `xBoundary` [10]
- char `yBoundary` [10]
- double * `DeltaXijOld`
- double * `DeltaYijOld`
- double `DeltaXijNew`
- double `DeltaYijNew`
- double `DeltaXij`
- double `DeltaYij`
- double `DeltaVXij`
- double `DeltaVYij`
- double ** `DeltaXijOldPair`
- double ** `DeltaYijOldPair`
- int * `molID`
- int ** `isBonded`
- int * `cellList`
- int `cells` [2+1]
- int `rank`
- int `size`
- int `master`
- double * `fax`
- double * `fay`
- double `fuSum`
- double `fvirSum`
- double `frfAtom`
- double `uSumPair`
- double `uSumPairPerAtom`
- double `virSumPair`
- double `virSumPairxx`
- double `virSumPairyy`
- double `virSumPairxy`
- double `virSumBond`
- double `virSumBondxx`
- double `virSumBondyy`
- double `virSumBondxy`
- double `virSumxx`
- double `virSumyy`

- double [virSumxy](#)
- int [freezeAtomType](#)
- double ** [cfOrg](#)
- double ** [spacetimeCorr](#)
- double * [cfVal](#)
- double * [spacetimeCorrAv](#)
- int * [indexCorr](#)
- int [countCorrAv](#)
- int [limitCorrAv](#)
- int [nBuffCorr](#)
- int [nFunCorr](#)
- int [nValCorr](#)
- int [stepCorr](#)
- double [rfAtom](#)
- double * [indexAcf](#)
- double ** [viscAcf](#)
- double * [viscAcfOrg](#)
- double * [viscAcfAv](#)
- double [viscAcfInt](#)
- int [nValAcf](#)
- int [nBuffAcf](#)
- int [stepAcf](#)
- int [countAcfAv](#)
- int [limitAcfAv](#)
- double * [histRdf](#)
- double [rangeRdf](#)
- int [countRdf](#)
- int [limitRdf](#)
- int [sizeHistRdf](#)
- int [stepRdf](#)
- char * [prefix](#)
- char [result](#) [250]
- FILE * [fpresult](#)
- char [xyz](#) [256]
- FILE * [fpxyz](#)
- char [bond](#) [256]
- FILE * [fpbond](#)
- char [dump](#) [256]
- FILE * [fpdump](#)
- char [dnsty](#) [256]
- FILE * [fpdnsty](#)
- char [visc](#) [256]
- FILE * [fpvisc](#)
- char [rdf](#) [256]
- FILE * [fprdf](#)
- char [vrms](#) [256]
- FILE * [fpvrms](#)
- char [stress](#) [256]
- FILE * [fpstress](#)
- char [momentum](#) [256]
- FILE * [fpmomentum](#)
- char [com](#) [256]
- FILE * [fpcom](#)
- char [pair](#) [256]
- FILE * [fppair](#)

3.60.1 Macro Definition Documentation

3.60.1.1 NDIM

```
#define NDIM 2
```

Definition at line 6 of file [globalExtern.h](#).

3.60.1.2 SignR

```
#define SignR(  
    x,  
    y ) ((y) >= 0) ? (x) : (- (x))
```

Definition at line 8 of file [globalExtern.h](#).

3.60.1.3 Sqr

```
#define Sqr(  
    x ) ((x) * (x))
```

Definition at line 7 of file [globalExtern.h](#).

3.60.2 Typedef Documentation

3.60.2.1 real

```
typedef double real
```

Definition at line 4 of file [globalExtern.h](#).

3.60.3 Variable Documentation

3.60.3.1 atom1

```
int* atom1 [extern]
```

3.60.3.2 atom2

```
int * atom2
```

Definition at line 27 of file [globalExtern.h](#).

3.60.3.3 atomID

```
int* atomID [extern]
```


3.60.3.4 atomIDInterface

```
int* atomIDInterface [extern]
```

3.60.3.5 atomMass

```
double* atomMass [extern]
```

3.60.3.6 atomRadius

```
double* atomRadius [extern]
```

3.60.3.7 atomType

```
int* atomType [extern]
```

3.60.3.8 ax

```
double * ax
```

Definition at line 10 of file [globalExtern.h](#).

3.60.3.9 ay

```
double * ay
```

Definition at line 10 of file [globalExtern.h](#).

3.60.3.10 bond

```
char bond[256] [extern]
```

3.60.3.11 BondEnergy

```
double* BondEnergy [extern]
```

3.60.3.12 BondEnergyPerAtom

```
double BondEnergyPerAtom
```

Definition at line 31 of file [globalExtern.h](#).

3.60.3.13 BondID

```
int* BondID [extern]
```

3.60.3.14 BondLength

```
double * BondLength
```

Definition at line 30 of file [globalExtern.h](#).

3.60.3.15 BondType

```
int * BondType
```

Definition at line 28 of file [globalExtern.h](#).

3.60.3.16 cellList

```
int* cellList [extern]
```

3.60.3.17 cells

```
int cells[2+1]
```

Definition at line 70 of file [globalExtern.h](#).

3.60.3.18 cfOrg

```
double** cfOrg [extern]
```

3.60.3.19 cfVal

```
double * cfVal
```

Definition at line 81 of file [globalExtern.h](#).

3.60.3.20 com

```
char com[256] [extern]
```

3.60.3.21 ComX

```
double ComX [extern]
```

3.60.3.22 ComX0

```
double ComX0
```

Definition at line 37 of file [globalExtern.h](#).

3.60.3.23 ComXRatio

```
double ComXRatio
```

Definition at line 37 of file [globalExtern.h](#).

3.60.3.24 ComY

```
double ComY
```

Definition at line 37 of file [globalExtern.h](#).

3.60.3.25 ComY0

```
double ComY0
```

Definition at line 37 of file [globalExtern.h](#).

3.60.3.26 ComYRatio

```
double ComYRatio
```

Definition at line 37 of file [globalExtern.h](#).

3.60.3.27 countAcfAv

```
int countAcfAv
```

Definition at line 87 of file [globalExtern.h](#).

3.60.3.28 countCorrAv

```
int countCorrAv
```

Definition at line 82 of file [globalExtern.h](#).

3.60.3.29 countRdf

```
int countRdf [extern]
```

3.60.3.30 DampFlag

```
int DampFlag [extern]
```

3.60.3.31 deltaT

```
double deltaT
```

Definition at line 13 of file [globalExtern.h](#).

3.60.3.32 DeltaVXij

```
double DeltaVXij
```

Definition at line 62 of file [globalExtern.h](#).

3.60.3.33 DeltaVYij

```
double DeltaVYij
```

Definition at line 62 of file [globalExtern.h](#).

3.60.3.34 DeltaX

```
double DeltaX
```

Definition at line 39 of file [globalExtern.h](#).

3.60.3.35 DeltaXij

```
double DeltaXij [extern]
```

3.60.3.36 DeltaXijNew

```
double DeltaXijNew [extern]
```

3.60.3.37 DeltaXijOld

```
double* DeltaXijOld [extern]
```

3.60.3.38 DeltaXijOldPair

```
double** DeltaXijOldPair [extern]
```

3.60.3.39 DeltaY

```
double DeltaY [extern]
```

3.60.3.40 DeltaYij

```
double DeltaYij
```

Definition at line 62 of file [globalExtern.h](#).

3.60.3.41 DeltaYijNew

```
double DeltaYijNew
```

Definition at line 61 of file [globalExtern.h](#).

3.60.3.42 DeltaYijOld

```
double * DeltaYijOld
```

Definition at line 60 of file [globalExtern.h](#).

3.60.3.43 DeltaYijOldPair

```
double ** DeltaYijOldPair
```

Definition at line 63 of file [globalExtern.h](#).

3.60.3.44 density

```
double density
```

Definition at line 14 of file [globalExtern.h](#).

3.60.3.45 discDragx

```
double* discDragx [extern]
```

3.60.3.46 discDragy

```
double * discDragy
```

Definition at line 33 of file [globalExtern.h](#).

3.60.3.47 dnsty

```
char dnsty[256] [extern]
```

3.60.3.48 dump

```
char dump[256] [extern]
```

3.60.3.49 dumpPairFlag

```
int dumpPairFlag [extern]
```

3.60.3.50 fax

```
double* fax [extern]
```

3.60.3.51 fay

```
double * fay
```

Definition at line 72 of file [globalExtern.h](#).

3.60.3.52 fpbond

```
FILE* fpbond [extern]
```

3.60.3.53 fpcom

```
FILE* fpcom [extern]
```

3.60.3.54 fpdnsty

```
FILE* fpdnsty [extern]
```

3.60.3.55 fpdump

```
FILE* fpdump [extern]
```

3.60.3.56 fpmomentum

```
FILE* fpmomentum [extern]
```

3.60.3.57 fppair

FILE* fppair [extern]

3.60.3.58 fprdf

FILE* fprdf [extern]

3.60.3.59 fpresult

FILE* fpresult [extern]

3.60.3.60 fpstress

FILE* fpstress [extern]

3.60.3.61 fpvisc

FILE* fpvisc [extern]

3.60.3.62 fpvrms

FILE* fpvrms [extern]

3.60.3.63 fpxyz

FILE* fpxyz [extern]

3.60.3.64 freezeAtomType

int freezeAtomType [extern]

3.60.3.65 frfAtom

double frfAtom

Definition at line 72 of file [globalExtern.h](#).

3.60.3.66 fuSum

double fuSum

Definition at line 72 of file [globalExtern.h](#).

3.60.3.67 fvirSum

```
double fvirSum
```

Definition at line 72 of file [globalExtern.h](#).

3.60.3.68 fx

```
double fx [extern]
```

3.60.3.69 fxByfy

```
double fxByfy
```

Definition at line 45 of file [globalExtern.h](#).

3.60.3.70 fy

```
double fy
```

Definition at line 45 of file [globalExtern.h](#).

3.60.3.71 FyBylx

```
double FyBylx
```

Definition at line 45 of file [globalExtern.h](#).

3.60.3.72 gamman

```
double gamman [extern]
```

3.60.3.73 HaltCondition

```
double HaltCondition [extern]
```

3.60.3.74 histRdf

```
double* histRdf [extern]
```

3.60.3.75 ImageX

```
int* ImageX [extern]
```


3.60.3.76 ImageY

```
int * ImageY
```

Definition at line 40 of file [globalExtern.h](#).

3.60.3.77 indexAcf

```
double* indexAcf [extern]
```

3.60.3.78 indexCorr

```
int* indexCorr [extern]
```

3.60.3.79 initUcell

```
int initUcell[2+1] [extern]
```

3.60.3.80 isBonded

```
int** isBonded [extern]
```

3.60.3.81 kappa

```
double kappa
```

Definition at line 14 of file [globalExtern.h](#).

3.60.3.82 kb

```
double* kb [extern]
```

3.60.3.83 kinEnergy

```
double kinEnergy
```

Definition at line 13 of file [globalExtern.h](#).

3.60.3.84 Kn

```
double Kn [extern]
```

3.60.3.85 limitAcfAv

```
int limitAcfAv
```

Definition at line 87 of file [globalExtern.h](#).

3.60.3.86 limitCorrAv

```
int limitCorrAv
```

Definition at line 82 of file [globalExtern.h](#).

3.60.3.87 limitRdf

```
int limitRdf
```

Definition at line 91 of file [globalExtern.h](#).

3.60.3.88 master

```
int master
```

Definition at line 71 of file [globalExtern.h](#).

3.60.3.89 molID

```
int* molID [extern]
```

3.60.3.90 momentum

```
char momentum[256] [extern]
```

3.60.3.91 moreCycles

```
int moreCycles
```

Definition at line 17 of file [globalExtern.h](#).

3.60.3.92 nAtom

```
int nAtom
```

Definition at line 17 of file [globalExtern.h](#).

3.60.3.93 nAtomBlock

```
int nAtomBlock
```

Definition at line 42 of file [globalExtern.h](#).

3.60.3.94 nAtomInterface

```
int nAtomInterface [extern]
```

3.60.3.95 nAtomType

```
int nAtomType [extern]
```

3.60.3.96 nBond

```
int nBond [extern]
```

3.60.3.97 nBondType

```
int nBondType
```

Definition at line 26 of file [globalExtern.h](#).

3.60.3.98 nBuffAcf

```
int nBuffAcf
```

Definition at line 87 of file [globalExtern.h](#).

3.60.3.99 nBuffCorr

```
int nBuffCorr
```

Definition at line 82 of file [globalExtern.h](#).

3.60.3.100 nDiscInterface

```
int nDiscInterface
```

Definition at line 42 of file [globalExtern.h](#).

3.60.3.101 nFunCorr

```
int nFunCorr
```

Definition at line 82 of file [globalExtern.h](#).

3.60.3.102 nodeDragx

```
double * nodeDragx
```

Definition at line 33 of file [globalExtern.h](#).

3.60.3.103 nodeDragy

```
double * nodeDragy
```

Definition at line 33 of file [globalExtern.h](#).

3.60.3.104 nPairActive

```
int nPairActive
```

Definition at line 51 of file [globalExtern.h](#).

3.60.3.105 nPairTotal

```
int nPairTotal [extern]
```

3.60.3.106 nValAcf

```
int nValAcf [extern]
```

3.60.3.107 nValCorr

```
int nValCorr
```

Definition at line 82 of file [globalExtern.h](#).

3.60.3.108 pair

```
char pair[256] [extern]
```

3.60.3.109 Pairatom1

```
int * Pairatom1
```

Definition at line 52 of file [globalExtern.h](#).

3.60.3.110 Pairatom2

```
int * Pairatom2
```

Definition at line 52 of file [globalExtern.h](#).

3.60.3.111 PairID

```
int* PairID [extern]
```

3.60.3.112 PairXij

```
double* PairXij [extern]
```

3.60.3.113 PairYij

```
double * PairYij
```

Definition at line 53 of file [globalExtern.h](#).

3.60.3.114 potEnergy

```
double potEnergy
```

Definition at line 13 of file [globalExtern.h](#).

3.60.3.115 prefix

```
char* prefix [extern]
```

Definition at line 13 of file [main.c](#).

Referenced by [main\(\)](#).

3.60.3.116 pressure

```
double pressure
```

Definition at line 14 of file [globalExtern.h](#).

3.60.3.117 RadiusIJ

```
double RadiusIJ [extern]
```

3.60.3.118 RadiusIJInv

```
double RadiusIJInv
```

Definition at line 19 of file [globalExtern.h](#).

3.60.3.119 rangeRdf

```
double rangeRdf
```

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

3.60.3.120 rank

```
int rank [extern]
```

3.60.3.121 rCut

```
double rCut
```

Definition at line 14 of file [globalExtern.h](#).

3.60.3.122 rdf

```
char rdf[256] [extern]
```

3.60.3.123 region

```
double region[2+1] [extern]
```

3.60.3.124 regionH

```
double regionH[2+1]
```

Definition at line 13 of file [globalExtern.h](#).

3.60.3.125 result

```
char result[250] [extern]
```

3.60.3.126 rfAtom

```
double rfAtom [extern]
```

3.60.3.127 ro

```
double * ro
```

Definition at line 29 of file [globalExtern.h](#).

3.60.3.128 rx

```
double* rx [extern]
```

3.60.3.129 rxUnwrap

```
double* rxUnwrap [extern]
```

3.60.3.130 ry

```
double * ry
```

Definition at line 10 of file [globalExtern.h](#).

3.60.3.131 ryUnwrap

```
double * ryUnwrap
```

Definition at line 41 of file [globalExtern.h](#).

3.60.3.132 shearDisplacement

```
double shearDisplacement [extern]
```

3.60.3.133 shearVelocity

```
double shearVelocity
```

Definition at line 35 of file [globalExtern.h](#).

3.60.3.134 size

```
int size
```

Definition at line 71 of file [globalExtern.h](#).

3.60.3.135 sizeHistRdf

```
int sizeHistRdf
```

Definition at line 91 of file [globalExtern.h](#).

3.60.3.136 sKinEnergy

```
double sKinEnergy
```

Definition at line 14 of file [globalExtern.h](#).

3.60.3.137 solver

```
char solver[128] [extern]
```

3.60.3.138 spacetimeCorr

```
double ** spacetimeCorr
```

Definition at line 81 of file [globalExtern.h](#).

3.60.3.139 spacetimeCorrAv

```
double * spacetimeCorrAv
```

Definition at line 81 of file [globalExtern.h](#).

3.60.3.140 speed

```
double* speed [extern]
```

3.60.3.141 sPotEnergy

```
double sPotEnergy
```

Definition at line 14 of file [globalExtern.h](#).

3.60.3.142 sPressure

```
double sPressure
```

Definition at line 15 of file [globalExtern.h](#).

3.60.3.143 SqrRadiusIJ

```
double SqrRadiusIJ
```

Definition at line 19 of file [globalExtern.h](#).

3.60.3.144 ssKinEnergy

```
double ssKinEnergy
```

Definition at line 15 of file [globalExtern.h](#).

3.60.3.145 ssPotEnergy

```
double ssPotEnergy
```

Definition at line 15 of file [globalExtern.h](#).

3.60.3.146 ssPressure

```
double ssPressure
```

Definition at line 15 of file [globalExtern.h](#).

3.60.3.147 ssTotEnergy

```
double ssTotEnergy
```

Definition at line 15 of file [globalExtern.h](#).

3.60.3.148 stepAcf

```
int stepAcf
```

Definition at line 87 of file [globalExtern.h](#).

3.60.3.149 stepAvg

```
int stepAvg
```

Definition at line 17 of file [globalExtern.h](#).

3.60.3.150 stepCorr

```
int stepCorr
```

Definition at line 82 of file [globalExtern.h](#).

3.60.3.151 stepCount

```
int stepCount
```

Definition at line 17 of file [globalExtern.h](#).

3.60.3.152 stepDump

```
int stepDump
```

Definition at line 17 of file [globalExtern.h](#).

3.60.3.153 stepEquil

```
int stepEquil
```

Definition at line 17 of file [globalExtern.h](#).

3.60.3.154 stepLimit

```
int stepLimit
```

Definition at line 17 of file [globalExtern.h](#).

3.60.3.155 stepRdf

```
int stepRdf
```

Definition at line 91 of file [globalExtern.h](#).

3.60.3.156 stepTraj

```
int stepTraj
```

Definition at line 17 of file [globalExtern.h](#).

3.60.3.157 sTotEnergy

```
double sTotEnergy
```

Definition at line 15 of file [globalExtern.h](#).

3.60.3.158 strain

```
double strain [extern]
```

3.60.3.159 strainRate

```
double strainRate
```

Definition at line 34 of file [globalExtern.h](#).

3.60.3.160 strech

```
double strech [extern]
```

3.60.3.161 stress

```
char stress[256] [extern]
```

3.60.3.162 svirSum

```
double svirSum
```

Definition at line 14 of file [globalExtern.h](#).

3.60.3.163 timeNow

```
double timeNow
```

Definition at line 13 of file [globalExtern.h](#).

3.60.3.164 TotalBondEnergy

```
double TotalBondEnergy [extern]
```

3.60.3.165 TotalMass

```
double TotalMass
```

Definition at line 24 of file [globalExtern.h](#).

3.60.3.166 totEnergy

```
double totEnergy
```

Definition at line 13 of file [globalExtern.h](#).

3.60.3.167 uSum

```
double uSum
```

Definition at line 14 of file [globalExtern.h](#).

3.60.3.168 uSumPair

```
double uSumPair [extern]
```

3.60.3.169 uSumPairPerAtom

```
double uSumPairPerAtom
```

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

3.60.3.170 virSum

```
double virSum
```

Definition at line 14 of file [globalExtern.h](#).

3.60.3.171 virSumBond

```
double virSumBond [extern]
```

3.60.3.172 virSumBondxx

```
double virSumBondxx
```

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

3.60.3.173 virSumBondxy

```
double virSumBondxy
```

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

3.60.3.174 virSumBondyy

```
double virSumBondyy
```

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

3.60.3.175 virSumPair

```
double virSumPair
```

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

3.60.3.176 virSumPairxx

```
double virSumPairxx
```

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

3.60.3.177 virSumPairxy

```
double virSumPairxy
```

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

3.60.3.178 virSumPairyy

```
double virSumPairyy
```

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

3.60.3.179 virSumxx

```
double virSumxx [extern]
```

3.60.3.180 virSumxy

```
double virSumxy
```

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

3.60.3.181 virSumyy

```
double virSumyy
```

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

3.60.3.182 visc

```
char visc[256] [extern]
```

3.60.3.183 viscAcf

```
double ** viscAcf
```

Definition at line 86 of file [globalExtern.h](#).

3.60.3.184 viscAcfAv

```
double * viscAcfAv
```

Definition at line 86 of file [globalExtern.h](#).

3.60.3.185 viscAcfInt

```
double viscAcfInt
```

Definition at line 86 of file [globalExtern.h](#).

3.60.3.186 viscAcfOrg

```
double * viscAcfOrg
```

Definition at line 86 of file [globalExtern.h](#).

3.60.3.187 VMeanSqr

```
double VMeanSqr
```

Definition at line 36 of file [globalExtern.h](#).

3.60.3.188 vrms

```
char vrms[256] [extern]
```

3.60.3.189 VRootMeanSqr

```
double VRootMeanSqr
```

Definition at line 36 of file [globalExtern.h](#).

3.60.3.190 VSqr

```
double VSqr [extern]
```

3.60.3.191 vSum

```
double vSum
```

Definition at line 14 of file [globalExtern.h](#).

3.60.3.192 vSumX

```
double vSumX
```

Definition at line 14 of file [globalExtern.h](#).

3.60.3.193 vSumY

```
double vSumY
```

Definition at line 14 of file [globalExtern.h](#).

3.60.3.194 vvSum

```
double vvSum
```

Definition at line 14 of file [globalExtern.h](#).

3.60.3.195 vx

```
double * vx
```

Definition at line 10 of file [globalExtern.h](#).

3.60.3.196 vy

```
double * vy
```

Definition at line 10 of file [globalExtern.h](#).

3.60.3.197 xBoundary

```
char xBoundary[10] [extern]
```

3.60.3.198 xyz

```
char xyz[256] [extern]
```

3.60.3.199 yBoundary

```
char yBoundary[10]
```

Definition at line 57 of file [globalExtern.h](#).

3.61 globalExtern.h

[Go to the documentation of this file.](#)

```
00001 #ifndef GLOBALEXTERN_H
00002 #define GLOBALEXTERN_H
00003
00004 typedef double real;
00005
00006 #define NDIM 2
00007 #define Sqr(x) ((x) * (x))
00008 #define SignR(x, y) (((y) >= 0) ? (x) : (- (x)))
00009
00010 extern double *rx, *ry, *vx, *vy, *ax, *ay;
00011 extern double *speed;
00012
00013 extern double region[NDIM+1], regionH[NDIM+1], deltaT, timeNow, potEnergy, kinEnergy, totEnergy,
00014 density, pressure, rCut, kappa, uSum, virSum, svirSum, vSum, vSumX, vSumY, vvSum, sPotEnergy,
00015 sKinEnergy,
00016 sTotEnergy, sPressure, ssPotEnergy, ssKinEnergy, ssTotEnergy, ssPressure;
00017 extern int initUcell[NDIM+1], moreCycles, nAtom, stepAvg, stepCount, stepEquil, stepLimit,
00018 stepTraj, stepDump;
00019
00019 extern double RadiusIJ, SqrRadiusIJ, RadiusIJInv;
00020 extern int nAtomType;
00021 extern int *atomType;
00022 extern int *atomID;
00023 extern double *atomRadius;
00024 extern double *atomMass, TotalMass;
00025
00026 extern int nBond, nBondType;
00027 extern int *atom1, *atom2;
00028 extern int *BondID, *BondType ;
00029 extern double *kb, *ro;
00030 extern double *BondEnergy, *BondLength;
00031 extern double TotalBondEnergy, BondEnergyPerAtom;
00032 extern double gamman;
00033 extern double *discDragx, *discDragy, *nodeDragx, *nodeDragy;
00034 extern double strain, strainRate;
00035 extern double shearDisplacement, shearVelocity;
00036 extern double VSqr, VMeanSqr, VRootMeanSqr;
00037 extern double ComX, ComY, ComX0, ComY0, ComXRatio, ComYRatio;
00038 extern double HaltCondition;
00039 extern double DeltaY, DeltaX;
00040 extern int *ImageX, *ImageY;
00041 extern double *rxUnwrap, *ryUnwrap;
00042 extern int nAtomInterface, nDiscInterface, nAtomBlock;
00043 extern int *atomIDInterface;
00044 extern double Kn;
00045 extern double fx, fy, FyBylx, fxByfy;
00046 extern int DampFlag;
00047 extern double strech;
00048
00049 //For dumping the pair interaction data
00050 extern int dumpPairFlag;
00051 extern int nPairTotal, nPairActive;
00052 extern int *PairID, *Pairatom1, *Pairatom2;
00053 extern double *PairXij, *PairYij;
00054
00055
00056 extern char solver[128];
00057 extern char xBoundary[10], yBoundary[10];
00058
00059 //For damping as in PRL, 130, 178203 (2023)
00060 extern double *DeltaXijOld, *DeltaYijOld;
00061 extern double DeltaXijNew, DeltaYijNew;
00062 extern double DeltaXij, DeltaYij, DeltaVXij, DeltaVYij;
00063 extern double **DeltaXijOldPair, **DeltaYijOldPair;
00064
00065 //For molecule-ID as per LAMMPS, helpful!
00066 extern int *molID;
00067 extern int **isBonded;
```



```

00068
00069 //Following three for MPI only
00070 extern int      *cellList, cells[NDIM+1];
00071 extern int      rank, size, master;
00072 extern double   *fax, *fay, fuSum, fvirSum, frfAtom;
00073
00074 //For thermodynamic properties
00075 extern double   uSumPair, uSumPairPerAtom, virSumPair, virSumPairxx, virSumPairyy, virSumPairxy;
00076 extern double   virSumBond, virSumBondxx, virSumBondyy, virSumBondxy;
00077 extern double   virSumxx, virSumyy, virSumxy;
00078 extern int      freezeAtomType;
00079
00080 // Spacetime Correlations
00081 extern double   **cfOrg, **spacetimeCorr, *cfVal, *spacetimeCorrAv;
00082 extern int      *indexCorr, countCorrAv, limitCorrAv, nBuffCorr, nFunCorr, nValCorr, stepCorr;
00083
00084 // Viscosity
00085 extern double   rfAtom, frfAtom;
00086 extern double   *indexAcf, **viscAcf, *viscAcfOrg, *viscAcfAv, viscAcfInt;
00087 extern int      nValAcf, nBuffAcf, stepAcf, countAcfAv, limitAcfAv;
00088
00089 // Radial distribution function
00090 extern double   *histRdf, rangeRdf;
00091 extern int      countRdf, limitRdf, sizeHistRdf, stepRdf;
00092
00093
00094 // Output files prefixes
00095 extern char     *prefix;
00096
00097 extern char     result[250];
00098 extern FILE     *fpresult;
00099
00100 extern char     xyz[256];
00101 extern FILE     *fpxyz;
00102
00103 extern char     bond[256];
00104 extern FILE     *fpbond;
00105
00106
00107 extern char     dump[256];
00108 extern FILE     *fpdump;
00109
00110 extern char     dnsty[256];
00111 extern FILE     *fpdnsty;
00112
00113 extern char     visc[256];
00114 extern FILE     *fpvisc;
00115
00116 extern char     rdf[256];
00117 extern FILE     *fprdf;
00118
00119 extern char     vrms[256];
00120 extern FILE     *fpvrms;
00121
00122 extern char     stress[256];
00123 extern FILE     *fpstress;
00124
00125 extern char     momentum[256];
00126 extern FILE     *fpmomentum;
00127
00128 extern char     com[256];
00129 extern FILE     *fpcom;
00130
00131 extern char     pair[256];
00132 extern FILE     *fppair;
00133
00134 #endif // GLOBALEXTERN_H

```

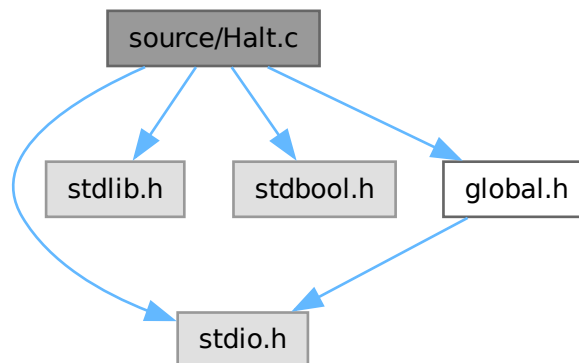
3.62 source/Halt.c File Reference

```

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "global.h"

```

Include dependency graph for Halt.c:



Functions

- bool [HaltConditionCheck](#) (double *value*, int *stepCount*)

3.62.1 Function Documentation

3.62.1.1 HaltConditionCheck()

```
bool HaltConditionCheck (
    double value,
    int stepCount )
```

Definition at line 27 of file [Halt.c](#).

```
00027                                     {
00028
00029     if(value <= HaltCondition && value != 0) {
00030         fprintf(fpresult, "Halt condition met at step = %d with Vrms = %.10f\n", stepCount, value);
00031         return true;           // Signal that the halt condition is met
00032     }
00033     return false; // Halt condition not met
00034 }
```

References [fpresult](#), [HaltCondition](#), and [stepCount](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.63 Halt.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022 #include <stdio.h>
00023 #include <stdlib.h>
00024 #include <stdbool.h>
00025 #include "global.h"
00026
00027 bool HaltConditionCheck(double value, int stepCount) {
00028
00029     if(value <= HaltCondition && value != 0) {
00030         fprintf(fpresult, "Halt condition met at step = %d with Vrms = %.10f\n", stepCount, value);
00031         return true; // Signal that the halt condition is met
00032     }
00033     return false; // Halt condition not met
00034 }
00035

```

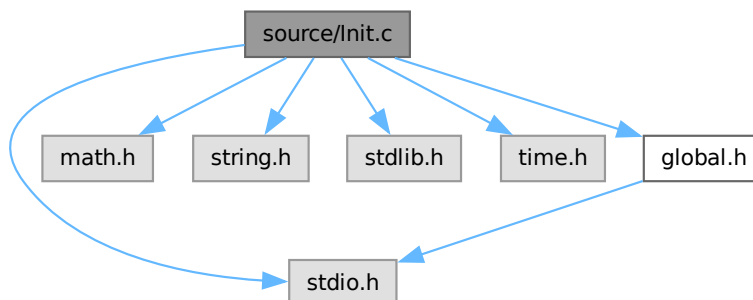
3.64 source/Init.c File Reference

```

#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include "global.h"

```

Include dependency graph for Init.c:



Functions

- void [Init](#) ()

3.64.1 Function Documentation

3.64.1.1 Init()

void Init ()

Definition at line 29 of file [Init.c](#).

```
00029     {
00030     char dummy[128];
00031     char inputConfig[128];
00032     FILE *fp;
00033     fp = fopen("input-data","r");
00034     fscanf(fp, "%s %s", dummy, inputConfig);
00035     fscanf(fp, "%s %s", dummy, solver);
00036     fscanf(fp, "%s %s %s", dummy, xBoundary, yBoundary);
00037     fscanf(fp, "%s %d", dummy, &DampFlag);
00038     fscanf(fp, "%s %d", dummy, &freezeAtomType);
00039     fscanf(fp, "%s %lf", dummy, &rCut);
00040     fscanf(fp, "%s %lf", dummy, &gamman);
00041     fscanf(fp, "%s %lf", dummy, &kappa);
00042     fscanf(fp, "%s %lf", dummy, &deltaT);
00043     fscanf(fp, "%s %lf", dummy, &strain);
00044     fscanf(fp, "%s %lf", dummy, &FyBylx);
00045     fscanf(fp, "%s %lf", dummy, &fxByfy);
00046     fscanf(fp, "%s %lf", dummy, &DeltaY);
00047     fscanf(fp, "%s %lf", dummy, &DeltaX);
00048     fscanf(fp, "%s %lf", dummy, &HaltCondition);
00049     fscanf(fp, "%s %d", dummy, &stepAvg);
00050     fscanf(fp, "%s %d", dummy, &stepEquil);
00051     fscanf(fp, "%s %d", dummy, &stepLimit);
00052     fscanf(fp, "%s %d", dummy, &stepDump);
00053     fscanf(fp, "%s %d", dummy, &stepTraj);
00054     fscanf(fp, "%s %d", dummy, &limitCorrAv);
00055     fscanf(fp, "%s %d", dummy, &nBuffCorr);
00056     fscanf(fp, "%s %d", dummy, &nFunCorr);
00057     fscanf(fp, "%s %d", dummy, &nValCorr);
00058     fscanf(fp, "%s %d", dummy, &stepCorr);
00059     fscanf(fp, "%s %d", dummy, &limitAcfAv);
00060     fscanf(fp, "%s %d", dummy, &nBuffAcf);
00061     fscanf(fp, "%s %d", dummy, &nValAcf);
00062     fscanf(fp, "%s %d", dummy, &stepAcf);
00063     fscanf(fp, "%s %lf", dummy, &rangeRdf);
00064     fscanf(fp, "%s %d", dummy, &limitRdf);
00065     fscanf(fp, "%s %d", dummy, &sizeHistRdf);
00066     fscanf(fp, "%s %d", dummy, &stepRdf);
00067
00068     fclose(fp);
00069     FILE *fpSTATE;
00070     if((fpSTATE = fopen(inputConfig,"r"))==NULL){
00071     printf("Error occurred: Could not open STATE file\n Exiting now..\n");
00072     exit(0);
00073     }
00074
00075     fscanf(fpSTATE, "%s %lf", dummy, &timeNow);
00076     fscanf(fpSTATE, "%s %d", dummy, &nAtom);
00077     fscanf(fpSTATE, "%s %d", dummy, &nBond);
00078     fscanf(fpSTATE, "%s %d", dummy, &nAtomType);
00079     fscanf(fpSTATE, "%s %d", dummy, &nBondType);
00080     fscanf(fpSTATE, "%s %lf", dummy, &region[1]);
00081     fscanf(fpSTATE, "%s %lf", dummy, &region[2]);
00082
00083     region[2] *= 1.5; //Remove this when put on GitHub
00084
00085     density = nAtom/(region[1]*region[2]);
00086     cells[1] = region[1] / rCut;
00087     cells[2] = region[2] / rCut;
00088     cellList = (int*)malloc((nAtom + cells[1] * cells[2] + 1) * sizeof(int));
00089     regionH[1] = 0.5*region[1];
00090     regionH[2] = 0.5*region[2];
00091
00092     //strain information
00093     strainRate = strain/deltaT;
00094     shearDisplacement = strain * region[2];
00095     shearVelocity = strainRate * region[2];
00096     int n;
```

```

00097
00098 rx = (double*)malloc((nAtom + 1) * sizeof(double));
00099 ry = (double*)malloc((nAtom + 1) * sizeof(double));
00100 vx = (double*)malloc((nAtom + 1) * sizeof(double));
00101 vy = (double*)malloc((nAtom + 1) * sizeof(double));
00102 ax = (double*)malloc((nAtom + 1) * sizeof(double));
00103 ay = (double*)malloc((nAtom + 1) * sizeof(double));
00104 fax = (double*)malloc((nAtom + 1) * sizeof(double));
00105 fay = (double*)malloc((nAtom + 1) * sizeof(double));
00106 atomID = (int*)malloc((nAtom+1) * sizeof(int));
00107 atomType = (int*)malloc((nAtom+1) * sizeof(int));
00108 atomRadius = (double*)malloc((nAtom + 1) * sizeof(double));
00109 atomMass = (double*)malloc((nAtom + 1) * sizeof(double));
00110 speed = (double*)malloc((nAtom + 1) * sizeof(double));
00111 atom1 = (int*)malloc((nBond+1)*sizeof(int));
00112 atom2 = (int*)malloc((nBond+1)*sizeof(int));
00113 BondID = (int*)malloc((nBond+1)*sizeof(int));
00114 BondType = (int*)malloc((nBond+1)*sizeof(int));
00115 kb = (double*)malloc((nBond+1)*sizeof(double));
00116 ro = (double*)malloc((nBond+1)*sizeof(double));
00117 BondEnergy = (double*)malloc((nBond+1)*sizeof(double));
00118 BondLength = (double*)malloc((nBond+1)*sizeof(double));
00119 discDragx = (double*)malloc((nAtom + 1) * sizeof(double));
00120 discDragy = (double*)malloc((nAtom + 1) * sizeof(double));
00121 nodeDragx = (double*)malloc((nAtom + 1) * sizeof(double));
00122 nodeDragy = (double*)malloc((nAtom + 1) * sizeof(double));
00123 ImageX = (int*)malloc((nAtom+1) * sizeof(int));
00124 ImageY = (int*)malloc((nAtom+1) * sizeof(int));
00125 rxUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
00126 ryUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
00127 DeltaXiJOld = (double*)malloc((nBond+1)*sizeof(double));
00128 DeltaYiJOld = (double*)malloc((nBond+1)*sizeof(double));
00129 DeltaXiJOldPair = (double**)malloc((nAtom+1) * sizeof(double*));
00130 DeltaYiJOldPair = (double**)malloc((nAtom+1) * sizeof(double*));
00131 for(int n = 0; n <= nAtom; n++) {
00132     DeltaXiJOldPair[n] = (double*)malloc((nAtom+1) * sizeof(double));
00133     DeltaYiJOldPair[n] = (double*)malloc((nAtom+1) * sizeof(double));
00134 }
00135 molID = (int*)malloc((nAtom+1) * sizeof(int));
00136
00137 for(n = 1; n <= nAtom; n++){
00138     atomMass[n] = 1.0;
00139 }
00140
00141 fscanf(fpSTATE, "%s\n", dummy);
00142 for(n = 1; n <= nAtom; n++)
00143     fscanf(fpSTATE, "%d %d %d %lf %lf %lf %lf %lf\n", &atomID[n], &molID[n], &atomType[n],
&atomRadius[n], &rx[n], &ry[n], &vx[n], &vy[n]);
00144
00145
00146 fscanf(fpSTATE, "%s\n", dummy);
00147 for(n=1; n<=nBond; n++)
00148     fscanf(fpSTATE, "%d %d %d %d %lf %lf\n", &BondID[n], &BondType[n], &atom1[n], &atom2[n], &kb[n],
&ro[n]);
00149
00150 fclose(fpSTATE);
00151
00152 //2D-List of bonded atoms. This is used to remove pair interaction
00153 //calculation for the bonded atoms
00154 isBonded = (int**)malloc((nAtom + 1) * sizeof(int*));
00155 for (int i = 0; i <= nAtom; i++) {
00156     isBonded[i] = (int*)malloc((nAtom + 1) * sizeof(int));
00157     for (int j = 0; j <= nAtom; j++) {
00158         isBonded[i][j] = 0;
00159     }
00160 }
00161
00162 for (n = 1; n <= nBond; n++) {
00163     int i = atom1[n];
00164     int j = atom2[n];
00165     isBonded[i][j] = 1;
00166     isBonded[j][i] = 1; // symmetric
00167 }
00168
00169 //For thermostate, update in final version
00170 thermo = 'C';
00171
00172
00173 // List the interface atoms
00174 nAtomInterface = 0;
00175 nAtomBlock = 0;
00176 nDiscInterface = 0;
00177 double InterfaceWidth, bigDiameter;
00178 bigDiameter = 2.8;
00179 InterfaceWidth = 5.0 * bigDiameter;
00180
00181 for(n = 1; n <= nAtom; n++){

```

```

00182     if(fabs(ry[n]) < InterfaceWidth){
00183         nAtomInterface++;
00184     }
00185     if(molID[n] == 2){
00186         nAtomBlock++;
00187     }
00188     if(atomRadius[n] != 0.0){
00189         nDiscInterface++;
00190     } }
00191
00192     atomIDInterface = (int*)malloc((nAtomInterface+1)*sizeof(int));
00193
00194     int m;
00195     m = 1;
00196     for(n=1; n<=nAtom; n++){
00197         if(fabs(ry[n]) < InterfaceWidth){
00198             atomIDInterface[m] = atomID[n];
00199             m++;
00200         } }
00201
00202     nPairTotal = 0.5 * nAtomInterface * (nAtomInterface-1);
00203     PairID = (int*)malloc((nPairTotal+1) * sizeof(int));
00204     Pairatom1 = (int*)malloc((nPairTotal+1) * sizeof(int));
00205     Pairatom2 = (int*)malloc((nPairTotal+1) * sizeof(int));
00206     PairXij = (double*)malloc((nPairTotal+1) * sizeof(double));
00207     PairYij = (double*)malloc((nPairTotal+1) * sizeof(double));
00208
00209     fprintf(fpresult, "-----\n");
00210     fprintf(fpresult, "-----PARAMETERS-----\n");
00211     fprintf(fpresult, "-----\n");
00212     fprintf(fpresult, "nAtom\t\t\t%d\n", nAtom);
00213     fprintf(fpresult, "nBond\t\t\t%d\n", nBond);
00214     fprintf(fpresult, "nAtomBlock\t\t\t%d\n", nAtomBlock);
00215     fprintf(fpresult, "nAtomInterface\t\t\t%d\n", nAtomInterface);
00216     fprintf(fpresult, "nDiscInterface\t\t\t%d\n", nDiscInterface);
00217     fprintf(fpresult, "gamman\t\t\t%.6g\n", gamman);
00218     fprintf(fpresult, "strain\t\t\t%.6g\n", strain);
00219     fprintf(fpresult, "strainRate\t\t\t%.6g\n", strainRate);
00220     fprintf(fpresult, "FyBylx\t\t\t%.6g\n", FyBylx);
00221     fprintf(fpresult, "fxByfy\t\t\t%.6g\n", fxByfy);
00222     fprintf(fpresult, "DeltaY\t\t\t%.6g\n", DeltaY);
00223     fprintf(fpresult, "DeltaX\t\t\t%.6g\n", DeltaX);
00224     fprintf(fpresult, "HaltCondition\t\t\t%.6g\n", HaltCondition);
00225     fprintf(fpresult, "kappa\t\t\t%.6g\n", kappa);
00226     fprintf(fpresult, "density\t\t\t%.6g\n", density);
00227     fprintf(fpresult, "rCut\t\t\t%.6g\n", rCut);
00228     fprintf(fpresult, "deltaT\t\t\t%.6g\n", deltaT);
00229     fprintf(fpresult, "stepEquil\t\t\t%d\n", stepEquil);
00230     fprintf(fpresult, "stepLimit\t\t\t%d\n", stepLimit);
00231     fprintf(fpresult, "region[1]\t\t\t%.16lf\n", region[1]);
00232     fprintf(fpresult, "region[2]\t\t\t%.16lf\n", region[2]);
00233     fprintf(fpresult, "cells[1]\t\t\t%d\n", cells[1]);
00234     fprintf(fpresult, "cells[2]\t\t\t%d\n", cells[2]);
00235     fprintf(fpresult, "solver\t\t\t%s\n", solver);
00236     fprintf(fpresult, "boundary\t\t\t%s %s\n", xBoundary, yBoundary);
00237     fprintf(fpresult, "DampFlag\t\t\t%d\n", DampFlag);
00238
00239
00240     fprintf(fpresult, "-----\n");
00241     fprintf(fpresult, "#TimeNow TotalMomentum PotEngPerAtom KinEngPerAtom TotEngPerAtom\n");
00242     fprintf(fpresult, "PairEnergyPerAtom BondEnergyPerAtom Press VirialSum\n");
00243     fprintf(fpvrms, "#timeNow\tVrms \n");
00244     fprintf(fpcom, "#timeNow\tComX\tComY\n");
00245
00246     /* //Uncomment the following as per your acquirement
00247     fprintf(fpstress, "strain\t\t\t%.1f\n", strain);
00248     fprintf(fpstress, "region[1]\t\t\t%.1f\n", region[1]);
00249     fprintf(fpstress, "region[2]\t\t\t%.1f\n", region[2]);
00250     fprintf(fpstress, "#timeNow virSumxx virSumyy virSumxy pressure\n");
00251     fprintf(fpmomentum, "#timeNow Px Py\n");
00252
00253     if((strcmp(xBoundary, "p") != 0 && strcmp(xBoundary, "r") != 0) ||
00254        (strcmp(yBoundary, "p") != 0 && strcmp(yBoundary, "r") != 0)) {
00255         fprintf(fpresult, "Error: Invalid boundary value detected: '%s %s'. Only 'p' or 'r' are\n");
00256         allowed.\n", xBoundary, yBoundary);
00257         exit(EXIT_FAILURE); // Exit with failure status
00258     }
00259 }

```

References [atom1](#), [atom2](#), [atomID](#), [atomIDInterface](#), [atomMass](#), [atomRadius](#), [atomType](#), [ax](#), [ay](#), [BondEnergy](#), [BondID](#), [BondLength](#), [BondType](#), [cellList](#), [cells](#), [DampFlag](#), [deltaT](#), [DeltaX](#), [DeltaXijOld](#), [DeltaXijOldPair](#), [DeltaY](#), [DeltaYijOld](#), [DeltaYijOldPair](#), [density](#), [discDragx](#), [discDragy](#), [fax](#), [fay](#), [fpcom](#), [fpresult](#), [fpvrms](#), [freezeAtomType](#), [fxByfy](#), [FyBylx](#), [gamman](#), [HaltCondition](#), [ImageX](#), [ImageY](#), [isBonded](#), [kappa](#), [kb](#), [limitAcfAv](#), [limitCorrAv](#), [limitRdf](#),

molID, nAtom, nAtomBlock, nAtomInterface, nAtomType, nBond, nBondType, nBuffAcf, nBuffCorr, nDiscInterface, nFunCorr, nodeDragx, nodeDragy, nPairTotal, nValAcf, nValCorr, Pairatom1, Pairatom2, PairID, PairXij, PairYij, rangeRdf, rCut, region, regionH, ro, rx, rxUnwrap, ry, ryUnwrap, shearDisplacement, shearVelocity, sizeHistRdf, solver, speed, stepAcf, stepAvg, stepCorr, stepDump, stepEquil, stepLimit, stepRdf, stepTraj, strain, strainRate, thermo, timeNow, vx, vy, xBoundary, and yBoundary.

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.65 Init.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022 #include<stdio.h>
00023 #include<math.h>
00024 #include<string.h>
00025 #include<stdlib.h>
00026 #include <time.h>
00027 #include"global.h"
00028
00029 void Init(){
00030     char dummy[128];
00031     char inputConfig[128];
00032     FILE *fp;
00033     fp = fopen("input-data","r");
00034     fscanf(fp, "%s %s", dummy, inputConfig);
00035     fscanf(fp, "%s %s", dummy, solver);
00036     fscanf(fp, "%s %s %s", dummy, xBoundary, yBoundary);
00037     fscanf(fp, "%s %d", dummy, &DampFlag);
00038     fscanf(fp, "%s %d", dummy, &freezeAtomType);
00039     fscanf(fp, "%s %lf", dummy, &rCut);
00040     fscanf(fp, "%s %lf", dummy, &gamman);
00041     fscanf(fp, "%s %lf", dummy, &kappa);
00042     fscanf(fp, "%s %lf", dummy, &deltaT);
00043     fscanf(fp, "%s %lf", dummy, &strain);
00044     fscanf(fp, "%s %lf", dummy, &FyBylx);
00045     fscanf(fp, "%s %lf", dummy, &fxByfy);
00046     fscanf(fp, "%s %lf", dummy, &DeltaY);
00047     fscanf(fp, "%s %lf", dummy, &DeltaX);
00048     fscanf(fp, "%s %lf", dummy, &HaltCondition);
00049     fscanf(fp, "%s %d", dummy, &stepAvg);
  
```

```

00050 fscanf(fp, "%s %d", dummy, &stepEquil);
00051 fscanf(fp, "%s %d", dummy, &stepLimit);
00052 fscanf(fp, "%s %d", dummy, &stepDump);
00053 fscanf(fp, "%s %d", dummy, &stepTraj);
00054 fscanf(fp, "%s %d", dummy, &limitCorrAv);
00055 fscanf(fp, "%s %d", dummy, &nBuffCorr);
00056 fscanf(fp, "%s %d", dummy, &nFunCorr);
00057 fscanf(fp, "%s %d", dummy, &nValCorr);
00058 fscanf(fp, "%s %d", dummy, &stepCorr);
00059 fscanf(fp, "%s %d", dummy, &limitAcfAv);
00060 fscanf(fp, "%s %d", dummy, &nBuffAcf);
00061 fscanf(fp, "%s %d", dummy, &nValAcf);
00062 fscanf(fp, "%s %d", dummy, &stepAcf);
00063 fscanf(fp, "%s %lf", dummy, &rangeRdf);
00064 fscanf(fp, "%s %d", dummy, &limitRdf);
00065 fscanf(fp, "%s %d", dummy, &sizeHistRdf);
00066 fscanf(fp, "%s %d", dummy, &stepRdf);
00067
00068 fclose(fp);
00069 FILE *fpSTATE;
00070 if((fpSTATE = fopen(inputConfig, "r"))==NULL){
00071 printf("Error occurred: Could not open STATE file\n Exiting now..\n");
00072 exit(0);
00073 }
00074
00075 fscanf(fpSTATE, "%s %lf", dummy, &timeNow);
00076 fscanf(fpSTATE, "%s %d", dummy, &nAtom);
00077 fscanf(fpSTATE, "%s %d", dummy, &nBond);
00078 fscanf(fpSTATE, "%s %d", dummy, &nAtomType);
00079 fscanf(fpSTATE, "%s %d", dummy, &nBondType);
00080 fscanf(fpSTATE, "%s %lf", dummy, &region[1]);
00081 fscanf(fpSTATE, "%s %lf", dummy, &region[2]);
00082
00083 region[2] *= 1.5; //Remove this when put on GitHub
00084
00085 density = nAtom/(region[1]*region[2]);
00086 cells[1] = region[1] / rCut;
00087 cells[2] = region[2] / rCut;
00088 cellList = (int*)malloc((nAtom + cells[1] * cells[2] + 1) * sizeof(int));
00089 regionH[1] = 0.5*region[1];
00090 regionH[2] = 0.5*region[2];
00091
00092 //strain information
00093 strainRate = strain/deltaT;
00094 shearDisplacement = strain * region[2];
00095 shearVelocity = strainRate * region[2];
00096 int n;
00097
00098 rx = (double*)malloc((nAtom + 1) * sizeof(double));
00099 ry = (double*)malloc((nAtom + 1) * sizeof(double));
00100 vx = (double*)malloc((nAtom + 1) * sizeof(double));
00101 vy = (double*)malloc((nAtom + 1) * sizeof(double));
00102 ax = (double*)malloc((nAtom + 1) * sizeof(double));
00103 ay = (double*)malloc((nAtom + 1) * sizeof(double));
00104 fax = (double*)malloc((nAtom + 1) * sizeof(double));
00105 fay = (double*)malloc((nAtom + 1) * sizeof(double));
00106 atomID = (int*)malloc((nAtom+1) * sizeof(int));
00107 atomType = (int*)malloc((nAtom+1) * sizeof(int));
00108 atomRadius = (double*)malloc((nAtom + 1) * sizeof(double));
00109 atomMass = (double*)malloc((nAtom + 1) * sizeof(double));
00110 speed = (double*)malloc((nAtom + 1) * sizeof(double));
00111 atom1 = (int*)malloc((nBond+1)*sizeof(int));
00112 atom2 = (int*)malloc((nBond+1)*sizeof(int));
00113 BondID = (int*)malloc((nBond+1)*sizeof(int));
00114 BondType = (int*)malloc((nBond+1)*sizeof(int));
00115 kb = (double*)malloc((nBond+1)*sizeof(double));
00116 ro = (double*)malloc((nBond+1)*sizeof(double));
00117 BondEnergy = (double*)malloc((nBond+1)*sizeof(double));
00118 BondLength = (double*)malloc((nBond+1)*sizeof(double));
00119 discDragx = (double*)malloc((nAtom + 1) * sizeof(double));
00120 discDragy = (double*)malloc((nAtom + 1) * sizeof(double));
00121 nodeDragx = (double*)malloc((nAtom + 1) * sizeof(double));
00122 nodeDragy = (double*)malloc((nAtom + 1) * sizeof(double));
00123 ImageX = (int*)malloc((nAtom+1) * sizeof(int));
00124 ImageY = (int*)malloc((nAtom+1) * sizeof(int));
00125 rxUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
00126 ryUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
00127 DeltaXijOld = (double*)malloc((nBond+1)*sizeof(double));
00128 DeltaYijOld = (double*)malloc((nBond+1)*sizeof(double));
00129 DeltaXijOldPair = (double**)malloc((nAtom+1) * sizeof(double*));
00130 DeltaYijOldPair = (double**)malloc((nAtom+1) * sizeof(double*));
00131 for(int n = 0; n <= nAtom; n++) {
00132   DeltaXijOldPair[n] = (double*)malloc((nAtom+1) * sizeof(double));
00133   DeltaYijOldPair[n] = (double*)malloc((nAtom+1) * sizeof(double));
00134 }
00135 molID = (int*)malloc((nAtom+1) * sizeof(int));
00136

```



```

00137     for(n = 1; n <= nAtom; n++){
00138         atomMass[n] = 1.0;
00139     }
00140
00141     fscanf(fpSTATE, "%s\n", dummy);
00142     for(n=1; n <= nAtom; n++){
00143         fscanf(fpSTATE, "%d %d %d %lf %lf %lf %lf %lf\n", &atomID[n], &molID[n], &atomType[n],
&atomRadius[n], &rx[n], &ry[n], &vx[n], &vy[n]);
00144
00145
00146         fscanf(fpSTATE, "%s\n", dummy);
00147         for(n=1; n<=nBond; n++){
00148             fscanf(fpSTATE, "%d %d %d %d %lf %lf\n", &BondID[n], &BondType[n], &atom1[n], &atom2[n], &kb[n],
&ro[n]);
00149
00150             fclose(fpSTATE);
00151
00152             //2D-List of bonded atoms. This is used to remove pair interaction
00153             //calculation for the bonded atoms
00154             isBonded = (int**)malloc((nAtom + 1) * sizeof(int*));
00155             for (int i = 0; i <= nAtom; i++) {
00156                 isBonded[i] = (int*)malloc((nAtom + 1) * sizeof(int));
00157                 for (int j = 0; j <= nAtom; j++) {
00158                     isBonded[i][j] = 0;
00159                 }
00160             }
00161
00162             for (n = 1; n <= nBond; n++) {
00163                 int i = atom1[n];
00164                 int j = atom2[n];
00165                 isBonded[i][j] = 1;
00166                 isBonded[j][i] = 1; // symmetric
00167             }
00168
00169             //For thermostat, update in final version
00170             thermo = 'C';
00171
00172
00173             // List the interface atoms
00174             nAtomInterface = 0;
00175             nAtomBlock = 0;
00176             nDiscInterface = 0;
00177             double InterfaceWidth, bigDiameter;
00178             bigDiameter = 2.8;
00179             InterfaceWidth = 5.0 * bigDiameter;
00180
00181             for(n = 1; n <= nAtom; n++){
00182                 if(fabs(ry[n]) < InterfaceWidth){
00183                     nAtomInterface++;
00184                 }
00185                 if(molID[n] == 2){
00186                     nAtomBlock++;
00187                 }
00188                 if(atomRadius[n] != 0.0){
00189                     nDiscInterface++;
00190                 }
00191             }
00192             atomIDInterface = (int*)malloc((nAtomInterface+1)*sizeof(int));
00193
00194             int m;
00195             m = 1;
00196             for(n=1; n<=nAtom; n++){
00197                 if(fabs(ry[n]) < InterfaceWidth){
00198                     atomIDInterface[m] = atomID[n];
00199                     m++;
00200                 }
00201             }
00202             nPairTotal = 0.5 * nAtomInterface * (nAtomInterface-1);
00203             PairID = (int*)malloc((nPairTotal+1) * sizeof(int));
00204             Pairatom1 = (int*)malloc((nPairTotal+1) * sizeof(int));
00205             Pairatom2 = (int*)malloc((nPairTotal+1) * sizeof(int));
00206             PairXij = (double*)malloc((nPairTotal+1) * sizeof(double));
00207             PairYij = (double*)malloc((nPairTotal+1) * sizeof(double));
00208
00209             fprintf(fpresult, "-----\n");
00210             fprintf(fpresult, "-----PARAMETERS-----\n");
00211             fprintf(fpresult, "-----\n");
00212             fprintf(fpresult, "nAtom\t\t\t%d\n", nAtom);
00213             fprintf(fpresult, "nBond\t\t\t%d\n", nBond);
00214             fprintf(fpresult, "nAtomBlock\t\t\t%d\n", nAtomBlock);
00215             fprintf(fpresult, "nAtomInterface\t\t\t%d\n", nAtomInterface);
00216             fprintf(fpresult, "nDiscInterface\t\t\t%d\n", nDiscInterface);
00217             fprintf(fpresult, "gamman\t\t\t%0.6g\n", gamman);
00218             fprintf(fpresult, "strain\t\t\t%0.6g\n", strain);
00219             fprintf(fpresult, "strainRate\t\t\t%0.6g\n", strainRate);
00220             fprintf(fpresult, "FyBylx\t\t\t%0.6g\n", FyBylx);
00221             fprintf(fpresult, "fxByfy\t\t\t%0.6g\n", fxByfy);

```

```

00222     fprintf(fpresult, "DeltaY\t\t\t%0.6g\n", DeltaY);
00223     fprintf(fpresult, "DeltaX\t\t\t%0.6g\n", DeltaX);
00224     fprintf(fpresult, "HaltCondition\t\t%0.6g\n", HaltCondition);
00225     fprintf(fpresult, "kappa\t\t\t%g\n", kappa);
00226     fprintf(fpresult, "density\t\t\t%g\n", density);
00227     fprintf(fpresult, "rCut\t\t\t\t%g\n", rCut);
00228     fprintf(fpresult, "deltaT\t\t\t\t%g\n", deltaT);
00229     fprintf(fpresult, "stepEquil\t\t\t%d\n", stepEquil);
00230     fprintf(fpresult, "stepLimit\t\t\t%d\n", stepLimit);
00231     fprintf(fpresult, "region[1]\t\t\t%0.16lf\n", region[1]);
00232     fprintf(fpresult, "region[2]\t\t\t%0.16lf\n", region[2]);
00233     fprintf(fpresult, "cells[1]\t\t\t%d\n", cells[1]);
00234     fprintf(fpresult, "cells[2]\t\t\t%d\n", cells[2]);
00235     fprintf(fpresult, "solver\t\t\t%s\n", solver);
00236     fprintf(fpresult, "boundary\t\t\t%s %s\n", xBoundary, yBoundary);
00237     fprintf(fpresult, "DampFlag\t\t\t%d\n", DampFlag);
00238
00239
00240     fprintf(fpresult, "-----\n");
00241     fprintf(fpresult, "#TimeNow TotalMomentum PotEngPerAtom KinEngPerAtom TotEngPerAtom
PairEnergyPerAtom BondEnergyPerAtom Press VirialSum\n");
00242     fprintf(fpvrms, "#timeNow\tVrms \n");
00243     fprintf(fpcom, "#timeNow\tComX\tComY\n");
00244
00245 /* //Uncomment the following as per your acquirement
00246     fprintf(fpstress, "strain\t\t\t%lf\n", strain);
00247     fprintf(fpstress, "region[1]\t\t\t%lf\n", region[1]);
00248     fprintf(fpstress, "region[2]\t\t\t%lf\n", region[2]);
00249     fprintf(fpstress, "#timeNow virSumxx virSumyy virSumxy pressure\n");
00250     fprintf(fpmomentum, "#timeNow Px Py\n");
00251 */
00252
00253     if((strcmp(xBoundary, "p") != 0 && strcmp(xBoundary, "r") != 0) ||
00254        (strcmp(yBoundary, "p") != 0 && strcmp(yBoundary, "r") != 0)) {
00255         fprintf(fpresult, "Error: Invalid boundary value detected: '%s %s'. Only 'p' or 'r' are
allowed.\n", xBoundary, yBoundary);
00256         exit(EXIT_FAILURE); // Exit with failure status
00257     }
00258
00259 }

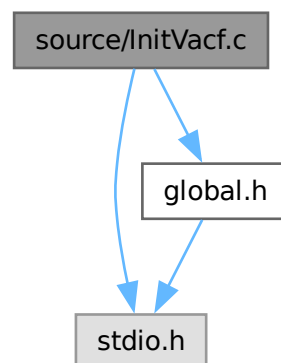
```

3.66 source/InitVacf.c File Reference

```

#include <stdio.h>
#include "global.h"
Include dependency graph for InitVacf.c:

```



Functions

- void [ZeroVacf](#) ()
- void [InitVacf](#) ()

3.66.1 Function Documentation

3.66.1.1 InitVacf()

```
void InitVacf ( )
```

Definition at line 26 of file [InitVacf.c](#).

```
00026         {  
00027     int nb;  
00028     for(nb = 1 ; nb <= nBuffAcf ; nb ++)  
00029         indexAcf[nb] = - (nb-1) * nValAcf / nBuffAcf;  
00030     ZeroVacf();  
00031 }
```

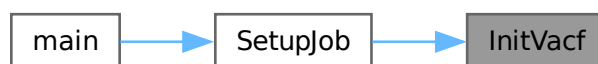
References [indexAcf](#), [nBuffAcf](#), [nValAcf](#), and [ZeroVacf](#)().

Referenced by [SetupJob](#)().

Here is the call graph for this function:



Here is the caller graph for this function:



3.66.1.2 ZeroVacf()

```
void ZeroVacf ( )
```

Definition at line 25 of file [ZeroVacf.c](#).

```
00025     {
00026     int j;
00027     countAcfAv= 0 ;
00028     for(j = 1 ; j <= nValAcf ; j ++)
00029         viscAcfAv[j] = 0.;
00030 }
```

Referenced by [InitVacf\(\)](#).

Here is the caller graph for this function:



3.67 InitVacf.c

[Go to the documentation of this file.](#)

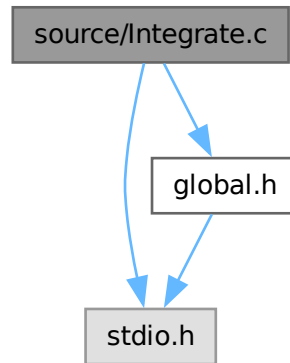
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 void ZeroVacf();
00026 void InitVacf(){
00027     int nb;
00028     for(nb = 1 ; nb <= nBuffAcf ; nb ++)
00029         indexAcf[nb] = -(nb-1)*nValAcf/nBuffAcf;
00030     ZeroVacf();
00031 }
```

3.68 source/Integrate.c File Reference

```
#include <stdio.h>
```

```
#include "global.h"
```

Include dependency graph for Integrate.c:



Functions

- double [Integrate](#) (double *f, int nf)

3.68.1 Function Documentation

3.68.1.1 Integrate()

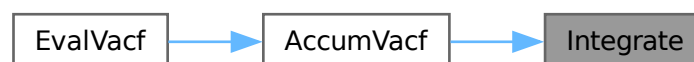
```
double Integrate (
    double * f,
    int nf )
```

Definition at line 25 of file [Integrate.c](#).

```
00025 {
00026     double s;
00027     int i;
00028     s = 0.5*(f[1] + f[nf]);
00029     for(i = 2 ; i <= nf - 1 ; i ++)
00030         s += f[i];
00031     return(s);
00032 }
```

Referenced by [AccumVacf\(\)](#).

Here is the caller graph for this function:



3.69 Integrate.c

[Go to the documentation of this file.](#)

```

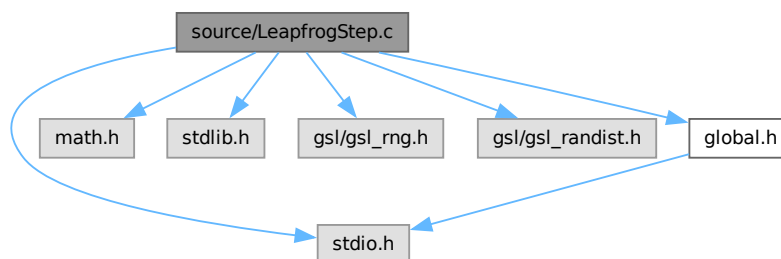
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 double Integrate(double *f, int nf){
00026     double s;
00027     int i;
00028     s = 0.5*(f[1] + f[nf]);
00029     for(i = 2 ; i <= nf - 1 ; i ++){
00030         s += f[i];
00031     }
00032     return(s);
00033 }
```

3.70 source/LeapfrogStep.c File Reference

```

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
#include "global.h"
```

Include dependency graph for LeapfrogStep.c:



Functions

- void [LeapfrogStep](#) (char [thermo](#), [gsl_rng](#) *rnd)

3.70.1 Function Documentation

3.70.1.1 LeapfrogStep()

```
void LeapfrogStep (
    char thermo,
    gsl_rng * rnd )
```

Definition at line 28 of file [LeapfrogStep.c](#).

```
00028 {
00029     double temperature, GAMMA;
00030     GAMMA = 100;
00031
00032     double *TValSum;
00033     TValSum = (double*)malloc((nAtom + 1) * sizeof(double));
00034
00035
00036     if(stepCount <= stepEquil){
00037         double gSum, varS, massS;
00038         temperature = 1./GAMMA;
00039
00040         if(stepCount == 1) varS = 0.;
00041         double A, S1, S2, T;
00042         int n;
00043         S1 = 0.; S2 = 0.; gSum = 0.; massS = 0.1;
00044
00045         vvSum = 0.;
00046         double halftdt = 0.5*deltaT;
00047         for (n = 1; n <= nAtom; n++){
00048             T = vx[n] + halftdt * ax[n];
00049             S1 += T * ax[n];
00050             S2 += Sqr(T);
00051
00052             T = vy[n] + halftdt * ay[n];
00053             S1 += T * ay[n];
00054             S2 += Sqr(T);
00055             vvSum += (Sqr(vx[n]) + Sqr(vy[n]));
00056         }
00057
00058         A = -S1 / S2;
00059         S2 = vvSum;
00060
00061         double C = 1 + A*deltaT ;
00062         double D = deltaT * (1 + 0.5 * A * deltaT);
00063
00064         int i,j;
00065         real dr[NDIM+1], r, rr, ri, rrCut;
00066         double vv;
00067
00068         double uVal, AA, AASum;
00069         double TVal;
00070
00071         double deno, VVSum;
00072         deno = 0.;
00073         VVSum = 0.;
00074         AASum = 0.;
00075
00076         for(n=1;n<=nAtom; n++)
00077             TValSum[n] = 0.;
00078
00079         rrCut = Sqr(rrCut);
00080
00081         /*****Calculating Configurational temperature*****/
00082         //Solving the equation of motion here
00083         if(thermo == 'C'){
00084             for(i = 1 ; i <= nAtom; i++){
00085                 for(j = i+1 ; j <= nAtom ; j++){
00086                     dr[1] = rx[i] - rx[j];
00087                     if(fabs(dr[1]) > regionH[1])
00088                         dr[1] -= SignR(region[1], dr[1]);
00089
00090                     dr[2] = ry[i] - ry[j];
00091                     if(fabs(dr[2]) > regionH[2])
00092                         dr[2] -= SignR(region[2], dr[2]);
00093
00094                     rr = Sqr(dr[1]) + Sqr(dr[2]);
00095                     if(rr < rrCut ){
00096                         r = sqrt(rr);
00097                         ri = 1/r;
00098                         uVal = ri*exp(-kappa*r);
00099
00100                         TVal = (1./rr + Sqr(kappa) + kappa/r)*uVal;
```

```

00101     TValSum[i] += TVal;
00102     TValSum[j] += TVal;
00103 } }
00104 AA = Sqr(ax[i]) + Sqr(ay[i]);
00105 AASum += AA;
00106 vv = Sqr(vx[i]) + Sqr(vy[i]);
00107 VVSum += vv;
00108 deno += TValSum[i];
00109 }
00110
00111 double gSumconfig, varSconfig, massSconfig;
00112 if(stepCount == 1) varSconfig = 0.;
00113 gSumconfig = 0.; massSconfig = 2.0;
00114
00115 gSumconfig = (AASum/temperature - deno)/massSconfig;
00116 varSconfig += deltaT*gSumconfig;
00117
00118 /*****Configarational Nose-Hoover thermostat*****/
00119 for (n = 1; n <= nAtom; n++){
00120     vx[n] += deltaT * ax[n];
00121     rx[n] += deltaT * (vx[n] + varSconfig * ax[n]);
00122     vy[n] += deltaT * ay[n];
00123     ry[n] += deltaT * (vy[n] + varSconfig * ay[n]);
00124 }
00125 /*****Kinetic Nose-Hoover thermostat*****/
00126 }else if(thermo == 'N'){
00127     gSum = (0.5*S2 - (nAtom + 1)*temperature)/massS;
00128     varS += deltaT*gSum;
00129     for (n = 1; n <= nAtom; n++){
00130         vx[n] += deltaT * (ax[n] - varS *vx[n]);
00131         rx[n] += deltaT * vx[n];
00132         vy[n] += deltaT * (ay[n] - varS *vy[n]);
00133         ry[n] += deltaT * vy[n];
00134     }
00135 /*****for Gaussian thermostat*****/
00136 }else if(thermo == 'G'){
00137     for (n = 1; n <= nAtom; n++){
00138         vx[n] = C * vx[n] + D * ax[n];
00139         rx[n] += deltaT * vx[n];
00140         vy[n] = C * vy[n] + D * ay[n];
00141         ry[n] += deltaT * vy[n];
00142     }
00143 }else if (thermo == 'L'){
00144     double nu = 0.03066;
00145     double var = sqrt(2*nu/(GAMMA*deltaT));
00146     double scale = 1. + nu*deltaT/2.;
00147     double scale_v = 2./scale - 1.;
00148     double scale_f = deltaT/scale;
00149     int n;
00150     for(n = 1 ; n <= nAtom ; n++){
00151         vx[n] = scale_v*vx[n] + scale_f*(ax[n] + var*gsl_ran_gaussian(rnd,1));
00152         rx[n] += deltaT * vx[n];
00153         vy[n] = scale_v*vy[n] + scale_f*(ay[n] + var*gsl_ran_gaussian(rnd,1));
00154         ry[n] += deltaT * vy[n];
00155     }
00156 }
00157 }else{
00158     int n;
00159     for(n = 1 ; n <= nAtom ; n++){
00160         vx[n] += deltaT * ax[n];
00161         rx[n] += deltaT * vx[n];
00162         vy[n] += deltaT * ay[n];
00163         ry[n] += deltaT * vy[n];
00164     }
00165 }
00166 }

```

References [ax](#), [ay](#), [deltaT](#), [kappa](#), [nAtom](#), [NDIM](#), [rCut](#), [region](#), [regionH](#), [rx](#), [ry](#), [SignR](#), [Sqr](#), [stepCount](#), [stepEquil](#), [thermo](#), [vvSum](#), [vx](#), and [vy](#).

3.71 LeapfrogStep.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.

```



```

00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020 #include<stdio.h>
00021 #include<math.h>
00022 #include<stdlib.h>
00023 #include <gsl/gsl_rng.h>
00024 #include <gsl/gsl_randist.h>
00025 #include"global.h"
00026
00027
00028 void LeapfrogStep(char thermo, gsl_rng * rnd){
00029 double temperature, GAMMA;
00030 GAMMA = 100;
00031
00032 double *TValSum;
00033 TValSum = (double*)malloc((nAtom + 1) * sizeof(double));
00034
00035
00036 if(stepCount <= stepEquil){
00037     double gSum, varS, massS;
00038     temperature = 1./GAMMA;
00039
00040     if(stepCount == 1) varS = 0.;
00041     double A, S1, S2, T;
00042     int n;
00043     S1 = 0.; S2 = 0.; gSum = 0.; massS = 0.1;
00044
00045     vvSum = 0.;
00046     double halfdt = 0.5*deltaT;
00047     for (n = 1; n <= nAtom; n++){
00048         T = vx[n] + halfdt * ax[n];
00049         S1 += T * ax[n];
00050         S2 += Sqr(T);
00051
00052         T = vy[n] + halfdt * ay[n];
00053         S1 += T * ay[n];
00054         S2 += Sqr(T);
00055         vvSum += (Sqr(vx[n]) + Sqr(vy[n]));
00056     }
00057
00058     A = -S1 / S2;
00059     S2 = vvSum;
00060
00061     double C = 1 + A*deltaT ;
00062     double D = deltaT * (1 + 0.5 * A * deltaT);
00063
00064     int i,j;
00065     real dr[NDIM+1], r, rr, ri, rrCut;
00066     double vv;
00067
00068     double uVal, AA, AASum;
00069     double TVal;
00070
00071     double deno, VVSum;
00072     deno = 0.;
00073     VVSum = 0.;
00074     AASum = 0.;
00075
00076     for(n=1;n<=nAtom; n++)
00077         TValSum[n] = 0.;
00078
00079     rrCut = Sqr(rCut);
00080
00081 /*****Calculating Configurational temperature*****/
00082 //Solving the equation of motion here
00083 if(thermo == 'C'){
00084     for(i = 1 ; i <= nAtom; i++){
00085         for(j = i+1 ; j <= nAtom ; j++){
00086             dr[1] = rx[i] - rx[j];
00087             if(fabs(dr[1]) > regionH[1])
00088                 dr[1] -= SignR(region[1], dr[1]);
00089
00090             dr[2] = ry[i] - ry[j];
00091             if(fabs(dr[2]) > regionH[2])
00092                 dr[2] -= SignR(region[2], dr[2]);
00093
00094             rr = Sqr(dr[1]) + Sqr(dr[2]);

```

```

00095     if(rr < rrCut ){
00096         r = sqrt(rr);
00097         ri = 1/r;
00098         uVal = ri*exp(-kappa*r);
00099
00100         TVal = (1./rr + Sqr(kappa) + kappa/r)*uVal;
00101         TValSum[i] += TVal;
00102         TValSum[j] += TVal;
00103     } }
00104     AA = Sqr(ax[i]) + Sqr(ay[i]);
00105     AASum += AA;
00106     vv = Sqr(vx[i]) + Sqr(vy[i]);
00107     VVSum += vv;
00108     deno += TValSum[i];
00109 }
00110
00111 double gSumconfig, varSconfig, massSconfig;
00112 if(stepCount == 1) varSconfig = 0.;
00113 gSumconfig = 0.; massSconfig = 2.0;
00114
00115 gSumconfig = (AASum/temperature - deno)/massSconfig;
00116 varSconfig += deltaT*gSumconfig;
00117
00118 /*****Configarational Nose-Hoover thermostat*****/
00119 for (n = 1; n <= nAtom; n++){
00120     vx[n] += deltaT * ax[n];
00121     rx[n] += deltaT * (vx[n] + varSconfig * ax[n]);
00122     vy[n] += deltaT * ay[n];
00123     ry[n] += deltaT * (vy[n] + varSconfig * ay[n]);
00124 }
00125 /*****Kinetic Nose-Hoover thermostat*****/
00126 }else if(thermo == 'N'){
00127     gSum = (0.5*S2 - (nAtom + 1)*temperature)/massS;
00128     varS += deltaT*gSum;
00129     for (n = 1; n <= nAtom; n++){
00130         vx[n] += deltaT * (ax[n] - varS *vx[n]);
00131         rx[n] += deltaT * vx[n];
00132         vy[n] += deltaT * (ay[n] - varS *vy[n]);
00133         ry[n] += deltaT * vy[n];
00134     }
00135     /*****for Gaussian thermostat*****/
00136 }else if(thermo == 'G'){
00137     for (n = 1; n <= nAtom; n++){
00138         vx[n] = C * vx[n] + D * ax[n];
00139         rx[n] += deltaT * vx[n];
00140         vy[n] = C * vy[n] + D * ay[n];
00141         ry[n] += deltaT * vy[n];
00142     }
00143 }else if (thermo == 'L'){
00144     double nu = 0.03066;
00145     double var = sqrt(2*nu/(GAMMA*deltaT));
00146     double scale = 1. + nu*deltaT/2.;
00147     double scale_v = 2./scale - 1.;
00148     double scale_f = deltaT/scale;
00149     int n;
00150     for(n = 1 ; n <= nAtom ; n++){
00151         vx[n] = scale_v*vx[n] + scale_f*(ax[n] + var*gsl_ran_gaussian(rnd,1));
00152         rx[n] += deltaT * vx[n];
00153         vy[n] = scale_v*vy[n] + scale_f*(ay[n] + var*gsl_ran_gaussian(rnd,1));
00154         ry[n] += deltaT * vy[n];
00155     }
00156 }
00157 }else{
00158     int n;
00159     for(n = 1 ; n <= nAtom ; n++){
00160         vx[n] += deltaT * ax[n];
00161         rx[n] += deltaT * vx[n];
00162         vy[n] += deltaT * ay[n];
00163         ry[n] += deltaT * vy[n];
00164     }
00165 }
00166 }
00167

```

3.72 source/main.c File Reference

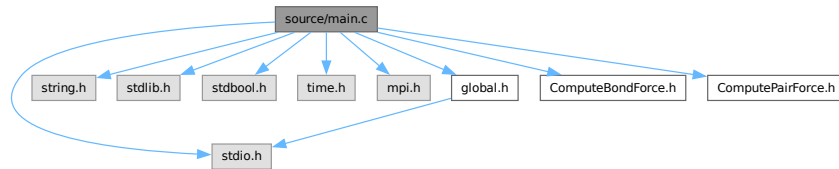
```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>

```

```
#include <time.h>
#include <mpi.h>
#include "global.h"
#include "ComputeBondForce.h"
#include "ComputePairForce.h"
```

Include dependency graph for main.c:



Macros

- `#define` [DEFINE_GLOBALS](#)

Functions

- void [Init](#) ()
- void [SetupJob](#) ()
- void [EvalSpacetimeCorr](#) ()
- void [Trajectory](#) ()
- void [DumpState](#) ()
- void [ComputeForcesCells](#) ()
- void [LeapfrogStep](#) ()
- void [BrownianStep](#) ()
- void [ApplyBoundaryCond](#) ()
- void [EvalProps](#) ()
- void [EvalVacf](#) ()
- void [EvalRdf](#) ()
- void [AccumProps](#) (int icode)
- void [PrintSummary](#) ()
- void [PrintVrms](#) ()
- void [DumpBonds](#) ()
- void [VelocityVerletStep](#) (int icode)
- void [ApplyForce](#) ()
- void [ApplyDrivingForce](#) ()
- void [ApplyShear](#) ()
- void [ApplyLeesEdwardsBoundaryCond](#) ()
- void [PrintStress](#) ()
- void [Close](#) ()
- void [PrintMomentum](#) ()
- void [DisplaceAtoms](#) ()
- void [DumpRestart](#) ()
- bool [HaltConditionCheck](#) (double value, int stepCount)
- void [EvalCom](#) ()
- void [PrintCom](#) ()
- void [EvalVrms](#) ()
- void [EvalUnwrap](#) ()
- void [DumpPairs](#) ()
- void [ApplyViscous](#) ()
- int [main](#) (int argc, char **argv)

Variables

- char * [prefix](#) = NULL

3.72.1 Macro Definition Documentation

3.72.1.1 DEFINE_GLOBALS

```
#define DEFINE_GLOBALS
```

Definition at line 7 of file [main.c](#).

3.72.2 Function Documentation

3.72.2.1 AccumProps()

```
void AccumProps (
    int icode )
```

Definition at line 25 of file [AccumProps.c](#).

```
00025 {
00026     if(icode == 0){
00027         sPotEnergy = ssPotEnergy = 0.;
00028         sKinEnergy = ssKinEnergy = 0.;
00029         sPressure = ssPressure = 0.;
00030         sTotEnergy = ssTotEnergy = 0.;
00031         svirSum = 0.;
00032     }else if(icode == 1){
00033         sPotEnergy += potEnergy;
00034         ssPotEnergy += Sqr(potEnergy);
00035         sKinEnergy += kinEnergy;
00036         ssKinEnergy += Sqr(kinEnergy);
00037         sTotEnergy += totEnergy;
00038         ssTotEnergy += Sqr(totEnergy);
00039         sPressure += pressure;
00040         ssPressure += Sqr(pressure);
00041         svirSum += virSum;
00042     }else if(icode == 2){
00043         sPotEnergy /= stepAvg;
00044         ssPotEnergy /= sqrt(ssPotEnergy/stepAvg - Sqr(sPotEnergy));
00045         sTotEnergy /= stepAvg;
00046         ssTotEnergy = sqrt(ssTotEnergy/stepAvg - Sqr(sTotEnergy));
00047         sKinEnergy /= stepAvg;
00048         ssKinEnergy = sqrt(ssKinEnergy/stepAvg - Sqr(sKinEnergy));
00049         sPressure /= stepAvg;
00050         ssPressure = sqrt(ssPressure/stepAvg - Sqr(sPressure));
00051         svirSum /= stepAvg;
00052     }
```

3.72.2.2 ApplyBoundaryCond()

```
void ApplyBoundaryCond ( )
```

Definition at line 27 of file [ApplyBoundaryCond.c](#).

```
00027 {
00028     int n;
00029     for(n = 1 ; n <= nAtom ; n ++){
00030         if(strcmp(xBoundary, "p") == 0 && strcmp(yBoundary, "p") == 0){ // P.B.C along x and y axis
00031             rx[n] -= region[1]*rint(rx[n]/region[1]);
00032             ry[n] -= region[2]*rint(ry[n]/region[2]);
00033         } else if (strcmp(xBoundary, "r") == 0 && strcmp(yBoundary, "r") == 0){ //R.B.C. along x and y
axis
00034             if((rx[n] + atomRadius[n]) >= regionH[1]){
00035                 rx[n] = 0.999999*regionH[1] - atomRadius[n]; vx[n] = -vx[n] ;
00036             }if((rx[n]-atomRadius[n]) < -regionH[1]){
```

```

00037         rx[n] = -0.999999*regionH[1] + atomRadius[n]; vx[n] = -vx[n] ;
00038     }
00039     if((ry[n] + atomRadius[n])>= regionH[2]){
00040         ry[n] = 0.999999*regionH[2] - atomRadius[n]; vy[n] = -vy[n] ;
00041     }if((ry[n]-atomRadius[n]) < -regionH[2]){
00042         ry[n] = -0.999999*regionH[2] + atomRadius[n]; vy[n] = -vy[n] ;
00043     }
00044     else if (strcmp(xBoundary, "p") == 0 && strcmp(yBoundary, "r") == 0){ //P.B.C. along x and R.B.C
along y axis
00045         rx[n] -= region[1]*rint(rx[n]/region[1]);
00046         if((ry[n] + atomRadius[n]) >= regionH[2]){
00047             ry[n] = 0.999999*regionH[2] - atomRadius[n]; vy[n] = -vy[n] ;
00048         }if((ry[n] - atomRadius[n]) < -regionH[2]){
00049             ry[n] = -0.999999*regionH[2] + atomRadius[n]; vy[n] = -vy[n] ;
00050         }
00051     else if(strcmp(xBoundary, "r") == 0 && strcmp(yBoundary, "p") == 0){ //R.B.C. along x and P.B.C
along y axis
00052         if((rx[n] + atomRadius[n]) >= regionH[1]){
00053             rx[n] = 0.999999*regionH[1] - atomRadius[n]; vx[n] = -vx[n] ;
00054         }if((rx[n] - atomRadius[n]) < -regionH[1]){
00055             rx[n] = -0.999999*regionH[1] + atomRadius[n]; vx[n] = -vx[n] ;
00056         }
00057         ry[n] -= region[2]*rint(ry[n]/region[2]);
00058     } else {
00059         // Print error message and exit the program
00060         fprintf(fpresult, "Error: Invalid boundary configuration: '%s %s'\n", xBoundary, yBoundary);
00061         exit(EXIT_FAILURE); // Exit with failure status
00062     }
00063 }
00064 }

```

References [atomRadius](#), [fpresult](#), [nAtom](#), [region](#), [regionH](#), [rx](#), [ry](#), [vx](#), [vy](#), [xBoundary](#), and [yBoundary](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.72.2.3 ApplyDrivingForce()

```
void ApplyDrivingForce ( )
```

Definition at line 25 of file [ApplyDrivingForce.c](#).

```

00025     {
00026     int n;
00027     double Vxblock, Vyblock;
00028     double Vxsubstrate, Vysubstrate;
00029     Vxblock = 0.0; Vyblock = 0.0;
00030     Vxsubstrate = 0.0; Vysubstrate = 0.0;
00031     double gammav;
00032     gammav = 0.0;
00033
00034     double count_substrate = 0;
00035     double count_block = 0;
00036
00037     for(n = 1 ; n <= nAtom; n++){
00038         if(atomType[n] == 1 || atomType[n] == 2){
00039             Vxsubstrate += vx[n]; Vysubstrate += vy[n];
00040             count_substrate++;
00041         }
00042         if(atomType[n] == 3 || atomType[n] == 4){
00043             Vxblock += vx[n]; Vyblock += vy[n];
00044             count_block++;
00045         }
00046     }
00047 }

```

```

00045     } }
00046
00047     if(count_substrate > 0) {
00048         Vxsubstrate /= count_substrate;
00049         Vysubstrate /= count_substrate;
00050     }
00051
00052     if(count_block > 0) {
00053         Vxblock /= count_block;
00054         Vyblock /= count_block;
00055     }
00056
00057     for(n = 1 ; n <= nAtom; n++){
00058         if(atomType[n] == 1 || atomType[n] == 2){
00059             ax[n] += -gammav * (vx[n] - Vxsubstrate);
00060             ay[n] += -gammav * (vy[n] - Vysubstrate);
00061         }
00062         if(atomType[n] == 3 || atomType[n] == 4){
00063             ax[n] += -gammav * (vx[n] - Vxblock);
00064             ay[n] += -gammav * (vy[n] - Vyblock);
00065         } }

```

References [atomType](#), [ax](#), [ay](#), [nAtom](#), [vx](#), and [vy](#).

3.72.2.4 ApplyForce()

```
void ApplyForce ( )
```

Definition at line 25 of file [ApplyForce.c](#).

```

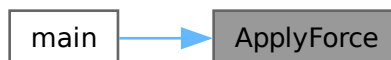
00025     {
00026     int n;
00027     double lx;
00028     lx = regionH[1];
00029     fy = (FyBylx * lx)/nAtomBlock;
00030     fx = fxByfy * fy;
00031     for(n = 1; n <= nAtom; n++){
00032         if(molID[n] == 2){
00033             ax[n] += fx;
00034             ay[n] -= fy;
00035         } }

```

References [ax](#), [ay](#), [fx](#), [fxByfy](#), [fy](#), [FyBylx](#), [molID](#), [nAtom](#), [nAtomBlock](#), and [regionH](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.72.2.5 ApplyLeesEdwardsBoundaryCond()

```
void ApplyLeesEdwardsBoundaryCond ( )
```

Definition at line 25 of file [ApplyLeesEdwardsBoundaryCond.c](#).

```

00025     {
00026     int n;
00027     for (n = 1; n <= nAtom; n++) {

```

```

00028 //PBC along x-direction
00029 if(rx[n] >= regionH[1])
00030     rx[n] -= region[1];
00031 else if(rx[n] < -regionH[1])
00032     rx[n] += region[1];
00033
00034 //LEBC along y-direction
00035 if(ry[n] >= regionH[2]){
00036     rx[n] -= shearDisplacement;
00037     if(rx[n] < -regionH[1]) rx[n] += region[1];
00038     //vx[n] -= shearVelocity;
00039     ry[n] -= region[2];
00040 }else if(ry[n] < -regionH[2]){
00041     rx[n] += shearDisplacement;
00042     if(rx[n] >= regionH[1]) rx[n] -= region[1];
00043     //vx[n] += shearVelocity;
00044     ry[n] += region[2];
00045 }
00046 }
00047 }

```

References [nAtom](#), [region](#), [regionH](#), [rx](#), [ry](#), and [shearDisplacement](#).

3.72.2.6 ApplyShear()

```
void ApplyShear ( )
```

Definition at line 25 of file [ApplyShear.c](#).

```

00025 {
00026     int n;
00027     for(n = 1 ; n <= nAtom ; n++){
00028         rx[n] += strain * ry[n];
00029         //vx[n] += strainRate * ry[n];
00030     } }

```

References [nAtom](#), [rx](#), [ry](#), and [strain](#).

3.72.2.7 ApplyViscous()

```
void ApplyViscous ( )
```

Definition at line 25 of file [ApplyViscous.c](#).

```

00025 {
00026     int n;
00027     double gammav;
00028     gammav = 1.0;
00029     for(n = 1 ; n <= nAtom; n++){
00030         ax[n] += -gammav * vx[n];
00031         ay[n] += -gammav * vy[n];
00032     } }

```

References [ax](#), [ay](#), [nAtom](#), [vx](#), and [vy](#).

3.72.2.8 BrownianStep()

```
void BrownianStep ( )
```

Definition at line 26 of file [BrownianStep.c](#).

```

00026 {
00027     if(stepCount <= stepEquil){
00028         double A, S1, S2, T;
00029         int n;
00030         S1 = 0.; S2 = 0;
00031         double halfdt = 0.5*deltaT;
00032         for (n = 1; n <= nAtom; n++){
00033             T = vx[n] + halfdt * ax[n];
00034             S1 += T * ax[n];
00035             S2 += Sqr(T);

```

```

00036
00037     T = vy[n] + halfdt * ay[n];
00038     S1 += T * ay[n];
00039     S2 += Sqr(T);
00040 }
00041 A = -S1 / S2;
00042 double C = 1 + A*deltaT ;
00043 double D = deltaT * (1 + 0.5 * A * deltaT);
00044 for (n = 1; n <= nAtom; n++){
00045     vx[n] = C * vx[n] + D * ax[n];
00046     rx[n] += deltaT * vx[n];
00047     vy[n] = C * vy[n] + D * ay[n];
00048     ry[n] += deltaT * vy[n];
00049 }
00050 }else{
00051     int n;
00052     //SETTING TEMP = 0.0
00053     if (stepCount == stepEquil+1){
00054         for(n = 1 ; n <= nAtom ; n ++){
00055             vx[n] = 0.0;
00056             vy[n] = 0.0;
00057         }}
00058     double zeta = 1.0;
00059     double dx, dy;
00060     for(n = 1 ; n <= nAtom ; n ++){
00061         dx = rx[n];
00062         rx[n] += zeta * ax[n] * deltaT;
00063         dx = rx[n] - dx;
00064         vx[n] = dx/deltaT;
00065         dy = ry[n];
00066         ry[n] += zeta * ay[n] * deltaT;
00067         dy = ry[n] - dy;
00068         vy[n] = dy/deltaT;
00069     }
00070 }
00071 }

```

References [ax](#), [ay](#), [deltaT](#), [nAtom](#), [rx](#), [ry](#), [Sqr](#), [stepCount](#), [stepEquil](#), [vx](#), and [vy](#).

3.72.2.9 Close()

```
void Close ( )
```

Definition at line 24 of file [Close.c](#).

```

00024     {
00025         int n;
00026         free(rx);
00027         free(ry);
00028         free(vx);
00029         free(vy);
00030         free(ax);
00031         free(ay);
00032         free(fax);
00033         free(fay);
00034         free(cellList);
00035
00036         free(atomID); free(atomType); free(atomRadius); free(atomMass);
00037         free(speed);
00038         free(atom1); free(atom2); free(BondID);
00039         free(BondType); free(kb); free(ro);
00040         free(ImageX); free(ImageY); free(rxUnwrap); free(ryUnwrap);
00041         free(atomIDInterface);
00042         free(PairID); free(Pairatom1); free(Pairatom2);
00043         free(PairXij); free(PairYij);
00044
00045         free(DeltaXijOld);
00046         free(DeltaYijOld);
00047
00048         free(molID);
00049
00050         for (n = 0; n <= nAtom; n++) {
00051             free(isBonded[n]);
00052         }
00053         free(isBonded);
00054
00055
00056
00057         for(n = 0; n <= nAtom; n++) {
00058             free(DeltaXijOldPair[n]);
00059             free(DeltaYijOldPair[n]);
00060         }

```



```

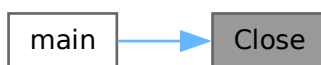
00061     free(DeltaXijOldPair);
00062     free(DeltaYijOldPair);
00063
00064     for (n = 0; n <= nBuffCorr; n++){
00065         free(cfOrg[n]);
00066         free(spacetimeCorr[n]);
00067     }
00068     free(cfOrg);
00069     free(spacetimeCorr);
00070     free(cfVal);
00071     free(indexCorr);
00072     free(spacetimeCorrAv);
00073
00074     free(indexAcf);
00075     free(viscAcfOrg);
00076     free(viscAcfAv);
00077     for(n = 0 ; n <= nBuffAcf ; n ++){
00078         free(viscAcf[n]);
00079     }
00080     free(viscAcf);
00081 }

```

References [atom1](#), [atom2](#), [atomID](#), [atomIDInterface](#), [atomMass](#), [atomRadius](#), [atomType](#), [ax](#), [ay](#), [BondID](#), [BondType](#), [cellList](#), [cfOrg](#), [cfVal](#), [DeltaXijOld](#), [DeltaXijOldPair](#), [DeltaYijOld](#), [DeltaYijOldPair](#), [fax](#), [fay](#), [ImageX](#), [ImageY](#), [indexAcf](#), [indexCorr](#), [isBonded](#), [kb](#), [molID](#), [nAtom](#), [nBuffAcf](#), [nBuffCorr](#), [Pairatom1](#), [Pairatom2](#), [PairID](#), [PairXij](#), [PairYij](#), [ro](#), [rx](#), [rxUnwrap](#), [ry](#), [ryUnwrap](#), [spacetimeCorr](#), [spacetimeCorrAv](#), [speed](#), [viscAcf](#), [viscAcfAv](#), [viscAcfOrg](#), [vx](#), and [vy](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.72.2.10 ComputeForcesCells()

```
void ComputeForcesCells ( )
```

Definition at line 25 of file [ComputeForcesCells.c](#).

```

00025     {
00026         double dr[NDIM+1], invWid[NDIM+1], shift[NDIM+1], f, fcVal, rr, ri, r, uVal;
00027         int c, I, J, m1, m1X, m1Y, m2, m2X, m2Y, n, offset;
00028         int iofX[] = {0, 0, 1, 1, 0, -1, -1, -1, 0, 1},
00029             iofY[] = {0, 0, 0, 1, 1, 1, 0, -1, -1, -1};
00030
00031         invWid[1] = cells[1]/region[1];
00032         invWid[2] = cells[2]/region[2];
00033
00034         for(n = nAtom+1; n <= nAtom+cells[1]*cells[2] ; n++){
00035             cellList[n] = 0;
00036
00037             for(n = 1 ; n <= nAtom ; n ++){
00038                 c = ((int)((ry[n] + regionH[2])*invWid[2]))*cells[1] + (int)((rx[n]+regionH[1])*invWid[1]) +
00039                     nAtom+ 1;
00039                 cellList[n] = cellList[c];
00040                 cellList[c] = n;
00041             }
00042
00043             for(n = 1 ; n <= nAtom ; n ++){
00044                 ax[n] = 0.;
00045                 ay[n] = 0.;
00046             }

```

```

00047
00048 uSum = 0.0 ;
00049 virSum = 0.0;
00050 rfAtom = 0.0;
00051 RadiusIJ = 0.0;
00052
00053 gamman = 1.0;
00054 double vr[NDIM+1], fd, fdVal, rrinv;
00055 rrinv = 0.0;
00056 fd = 0.0;
00057 fdVal = 0.0;
00058
00059 int start = 1 + rank*(cells[2]/size);
00060 int end = (rank+1)*(cells[2]/size);
00061
00062 for(m1Y = start ; m1Y <= end ; m1Y ++){
00063     for(m1X = 1 ; m1X <= cells[1] ; m1X ++){
00064         m1 = (m1Y-1) * cells[1] + m1X + nAtom;
00065         for(offset = 1 ; offset <= 9 ; offset ++){
00066             m2X = m1X + ioFX[offset]; shift[1] = 0.;
00067             if(m2X > cells[1]){
00068                 m2X = 1; shift[1] = region[1];
00069             }else if(m2X == 0){
00070                 m2X = cells[1]; shift[1] = -region[1];
00071             }
00072             m2Y = m1Y + ioFY[offset]; shift[2] = 0.;
00073             if(m2Y > cells[2]){
00074                 m2Y = 1; shift[2] = region[2];
00075             }else if(m2Y == 0){
00076                 m2Y = cells[2]; shift[2] = -region[2];
00077             }
00078             m2 = (m2Y-1)*cells[1] + m2X + nAtom;
00079             I = cellList[m1];
00080             while(I > 0){
00081                 J = cellList[m2];
00082                 while(J > 0){
00083                     if(m1 == m2 && J != I && (atomRadius[I] > 0. && atomRadius[J] > 0.)){
00084                         dr[1] = rx[I] - rx[J] - shift[1];
00085                         dr[2] = ry[I] - ry[J] - shift[2];
00086                         rr = Sqr(dr[1]) + Sqr(dr[2]);
00087                         RadiusIJ = atomRadius[I] + atomRadius[J];
00088                         SqrRadiusIJ = Sqr(RadiusIJ);
00089                         if(rr < SqrRadiusIJ){
00090                             r = sqrt(rr);
00091                             ri = 1.0/r;
00092                             rrinv = 1.0/rr;
00093                             vr[1] = vx[I] - vx[J];
00094                             vr[2] = vy[I] - vy[J];
00095                             RadiusIJInv = 1.0/RadiusIJ;
00096                             uVal = Sqr(1.0 - r * RadiusIJInv);
00097                             fcVal = 2.0 * RadiusIJInv * (1.0 - r * RadiusIJInv) * ri;
00098                             fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag
00099
00100                             f = fcVal * dr[1];
00101                             fd = fdVal * dr[1];
00102                             ax[I] += (f + fd);
00103                             discDragx[I] += fd; //disc-disc drag
00104
00105                             f = fcVal * dr[2];
00106                             fd = fdVal * dr[2];
00107                             ay[I] += (f + fd);
00108                             discDragy[I] += fd; //disc-disc drag
00109
00110                             uSum += 0.5 * uVal;
00111                             virSum += 0.5 * fcVal * rr;
00112                             rfAtom += 0.5 * dr[1] * fcVal * dr[2];
00113                         }
00114                     }else if(m1 != m2 && (atomRadius[I] > 0. && atomRadius[J] > 0.)){
00115                         dr[1] = rx[I] - rx[J] - shift[1];
00116                         dr[2] = ry[I] - ry[J] - shift[2];
00117                         rr = Sqr(dr[1]) + Sqr(dr[2]);
00118                         RadiusIJ = atomRadius[I] + atomRadius[J];
00119                         SqrRadiusIJ = Sqr(RadiusIJ);
00120                         if(rr < SqrRadiusIJ){
00121                             r = sqrt(rr);
00122                             ri = 1.0/r;
00123                             rrinv = 1.0/rr;
00124                             vr[1] = vx[I] - vx[J];
00125                             vr[2] = vy[I] - vy[J];
00126                             RadiusIJInv = 1.0/RadiusIJ;
00127                             uVal = Sqr(1.0 - r * RadiusIJInv);
00128                             fcVal = 2.0 * RadiusIJInv * (1.0 - r * RadiusIJInv) * ri;
00129                             fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag
00130
00131                             f = fcVal * dr[1];
00132                             fd = fdVal * dr[1];
00133                             ax[I] += (f + fd);

```

```

00134         discDragx[I] += fd; //disc-disc drag
00135
00136         f = fcVal * dr[2];
00137         fd = fdVal * dr[2];
00138         ay[I] += (f + fd);
00139         discDragy[I] += fd; //disc-disc drag
00140
00141         uSum += 0.5 * uVal;
00142         virSum += 0.5 * fcVal * rr;
00143         rfAtom += 0.5 * dr[1] * fcVal * dr[2];
00144     }
00145 }
00146     J = cellList[J];
00147 }
00148     I = cellList[I];
00149 }
00150 }
00151 }
00152 }
00153 }

```

References [atomRadius](#), [ax](#), [ay](#), [cellList](#), [cells](#), [discDragx](#), [discDragy](#), [gamman](#), [nAtom](#), [NDIM](#), [RadiusIJ](#), [RadiusIJInv](#), [rank](#), [region](#), [regionH](#), [rfAtom](#), [rx](#), [ry](#), [size](#), [Sqr](#), [SqrRadiusIJ](#), [uSum](#), [virSum](#), [vx](#), and [vy](#).

3.72.2.11 DisplaceAtoms()

```
void DisplaceAtoms ( )
```

Definition at line 25 of file [DisplaceAtoms.c](#).

```

00025     {
00026     int n;
00027     for(n = 1; n <= nAtom; n++){
00028         if(molID[n] == 2){
00029             rx[n] += DeltaX;
00030             ry[n] += DeltaY;
00031         } }

```

References [DeltaX](#), [DeltaY](#), [molID](#), [nAtom](#), [rx](#), and [ry](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.72.2.12 DumpBonds()

```
void DumpBonds ( )
```

Definition at line 24 of file [DumpBonds.c](#).

```

00024     {
00025     int n;
00026     //Trajectory file in LAMMPS dump format for OVITO visualization
00027     fprintf(fpbond, "ITEM: TIMESTEP\n");
00028     fprintf(fpbond, "%lf\n", timeNow);
00029     fprintf(fpbond, "ITEM: NUMBER OF ENTRIES\n");
00030     fprintf(fpbond, "%d\n", nBond);

```

```

00031     fprintf(fpbond, "ITEM: BOX BOUNDS pp ff pp\n");
00032     fprintf(fpbond, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
00033     fprintf(fpbond, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
00034     fprintf(fpbond, "%lf %lf zlo zhi\n", -0.1, 0.1);
00035     fprintf(fpbond, "ITEM: ENTRIES BondID, BondType, atom1 atom2 BondLength BondLengthEql nodeDragx1
nodeDragy1\n");
00036
00037     for(n=1; n<=nBond; n++)
00038         fprintf(fpbond, "%d %d %d %d %0.16lf %0.16lf %0.16lf %0.16lf\n", BondID[n], BondType[n], atom1[n],
atom2[n],
00039             BondLength[n], ro[n], nodeDragx[atom1[n]], nodeDragy[atom1[n]]);
00040     }

```

References [atom1](#), [atom2](#), [BondID](#), [BondLength](#), [BondType](#), [fpbond](#), [nBond](#), [nodeDragx](#), [nodeDragy](#), [regionH](#), [ro](#), and [timeNow](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.72.2.13 DumpPairs()

```
void DumpPairs ( )
```

Definition at line 25 of file [DumpPairs.c](#).

```

00025     {
00026     int n;
00027     //Trajectory file in LAMMPS dump format for OVITO visualization
00028     fprintf(fppair, "ITEM: TIMESTEP\n");
00029     fprintf(fppair, "%lf\n", timeNow);
00030     fprintf(fppair, "ITEM: NUMBER OF ENTRIES\n");
00031     fprintf(fppair, "%d\n", nPairActive);
00032     fprintf(fppair, "ITEM: BOX BOUNDS pp ff pp\n");
00033     fprintf(fppair, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
00034     fprintf(fppair, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
00035     fprintf(fppair, "%lf %lf zlo zhi\n", -0.1, 0.1);
00036     fprintf(fppair, "ITEM: ENTRIES index, atom1 atom2 xij yij discDragx1 discDragy1\n");
00037
00038     for(n=1; n<=nPairActive; n++)
00039         fprintf(fppair, "%d %d %d %0.16lf %0.16lf %0.16lf %0.16lf\n", PairID[n], Pairatom1[n],
Pairatom2[n],
00040             PairXij[n], PairYij[n], discDragx[n], discDragy[n]);
00041
00042     }

```

References [discDragx](#), [discDragy](#), [fppair](#), [nPairActive](#), [Pairatom1](#), [Pairatom2](#), [PairID](#), [PairXij](#), [PairYij](#), [regionH](#), and [timeNow](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.72.2.14 DumpRestart()

```
void DumpRestart ( )
```

Definition at line 25 of file [DumpRestart.c](#).

```

00025     {
00026     char DUMP[256];
00027     FILE *fpDUMP;
00028     sprintf(DUMP, "%s.Restart", prefix);
00029     fpDUMP = fopen(DUMP, "w");
00030     if(fpDUMP == NULL) {
00031         fprintf(stderr, "Error opening file %s for writing\n", DUMP);
00032         return;
00033     }
00034
00035     fprintf(fpDUMP, "timeNow %lf\n", timeNow);
00036     fprintf(fpDUMP, "nAtom %d\n", nAtom);
00037     fprintf(fpDUMP, "nBond %d\n", nBond);
00038     fprintf(fpDUMP, "nAtomType %d\n", nAtomType);
00039     fprintf(fpDUMP, "nBondType %d\n", nBondType);
00040     fprintf(fpDUMP, "region[1] %0.14lf\n", region[1]);
00041     fprintf(fpDUMP, "region[2] %0.14lf\n", region[2]);
00042
00043     int n;
00044     fprintf(fpDUMP, "Atoms\n");
00045     for(n = 1; n <= nAtom; n++)
00046         fprintf(fpDUMP, "%d %d %d %0.2lf %0.16lf %0.16lf %0.16lf %0.16lf\n", atomID[n], molID[n],
atomType[n], atomRadius[n], rx[n], ry[n], vx[n], vy[n]);
00047
00048
00049     fprintf(fpDUMP, "Bonds\n");
00050     for(n=1; n<=nBond; n++)
00051         fprintf(fpDUMP, "%d %d %d %d %0.2lf %0.16lf\n", BondID[n], BondType[n], atom1[n], atom2[n], kb[n],
ro[n]);
00052
00053     fclose(fpDUMP);
00054 }
  
```

References [atom1](#), [atom2](#), [atomID](#), [atomRadius](#), [atomType](#), [BondID](#), [BondType](#), [kb](#), [molID](#), [nAtom](#), [nAtomType](#), [nBond](#), [nBondType](#), [prefix](#), [region](#), [ro](#), [rx](#), [ry](#), [timeNow](#), [vx](#), and [vy](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.72.2.15 DumpState()

```
void DumpState ( )
```

Definition at line 25 of file [DumpState.c](#).

```
00025     {
00026     char DUMP[256];
00027     FILE *fpDUMP;
00028     sprintf(DUMP, "%s.STATE", prefix);
00029     fpDUMP = fopen(DUMP, "w");
00030     if(fpDUMP == NULL) {
00031         fprintf(stderr, "Error opening file %s for writing\n", DUMP);
00032         return;
00033     }
00034
00035     fprintf(fpDUMP, "ITEM: TIMESTEP\n");
00036     fprintf(fpDUMP, "%lf\n", timeNow);
00037     fprintf(fpDUMP, "ITEM: NUMBER OF ATOMS\n");
00038     fprintf(fpDUMP, "%d\n", nAtom);
00039     fprintf(fpDUMP, "ITEM: BOX BOUNDS pp pp pp\n");
00040     fprintf(fpDUMP, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
00041     fprintf(fpDUMP, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
00042     fprintf(fpDUMP, "%lf %lf zlo zhi\n", -0.1, 0.1);
00043     fprintf(fpDUMP, "ITEM: ATOMS id mol type radius x y vx vy fx fy\n");
00044     int n;
00045     for (n = 1; n <= nAtom; n++) {
00046         fprintf(fpDUMP, "%d\t%d\t%d\t%0.2lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t\n",
00047             atomID[n], molID[n], atomType[n], atomRadius[n], rx[n], ry[n], vx[n], vy[n], ax[n], ay[n]);
00048     }
00049     fclose(fpDUMP);
00050 }
```

References [atomID](#), [atomRadius](#), [atomType](#), [ax](#), [ay](#), [molID](#), [nAtom](#), [prefix](#), [regionH](#), [rx](#), [ry](#), [timeNow](#), [vx](#), and [vy](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.72.2.16 EvalCom()

```
void EvalCom ( )
```

Definition at line 27 of file [EvalCom.c](#).

```
00027     {
00028     int n;
00029     ComX = 0.0; ComY = 0.0; ComXRatio = 0.0; ComYRatio = 0.0;
00030     TotalMass = 0.0;
00031
00032     for(n=1; n<=nAtom; n++){
00033         if(molID[n] == 2){
00034             ComX += atomMass[n] * rxUnwrap[n];
00035             ComY += atomMass[n] * ryUnwrap[n];
00036             TotalMass += atomMass[n];
00037         }
00038
00039         ComX = ComX/TotalMass;
00040         ComY = ComY/TotalMass;
00041     }
```

```

00042  if(timeNow == 0.0){
00043      ComX0 = ComX; ComY0 = ComY;
00044  }
00045      ComXRatio = ComX/ComX0;   ComYRatio = ComY/ComY0;
00046  }

```

References [atomMass](#), [ComX](#), [ComX0](#), [ComXRatio](#), [ComY](#), [ComY0](#), [ComYRatio](#), [molID](#), [nAtom](#), [rxUnwrap](#), [ryUnwrap](#), [timeNow](#), and [TotalMass](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.72.2.17 EvalProps()

```
void EvalProps ( )
```

Definition at line 26 of file [EvalProps.c](#).

```

00026      {
00027  real v, vv;
00028  virSum = 0.0;
00029  vSumX = 0.0; vSumY = 0.0; vSum = 0.0;
00030  vvSum = 0.;
00031  int n;
00032
00033  for (n = 1; n <= nAtom; n++) {
00034      vv = 0.;
00035      // Initialize v with a default value to avoid "uninitialized" warning.
00036      v = 0.0;
00037      // X direction velocity
00038      if (strcmp(solver, "Verlet") == 0) {
00039          v = vx[n];
00040      } else if (strcmp(solver, "LeapFrog") == 0) {
00041          v = vx[n] - 0.5 * deltaT * ax[n];
00042      }
00043      vSum += v;
00044      vv += Sqr(v);
00045      vSumX += v;
00046      // Y direction velocity
00047      if (strcmp(solver, "Verlet") == 0) {
00048          v = vy[n];
00049      } else if (strcmp(solver, "LeapFrog") == 0) {
00050          v = vy[n] - 0.5 * deltaT * ay[n];
00051      }
00052      vSum += v;
00053      vSumY += v;
00054      vv += Sqr(v);
00055      vvSum += vv;
00056  }
00057
00058  kinEnergy = 0.5 * vvSum / nAtom ;
00059  uSumPairPerAtom = uSumPair / nAtom ;
00060  BondEnergyPerAtom = TotalBondEnergy / (0.5*nAtom); //Factor of 0.5 since each atom has one half the
bond energy
00061  potEnergy = uSumPairPerAtom + BondEnergyPerAtom ;
00062  totEnergy = kinEnergy + potEnergy;
00063  virSumxx = virSumPairxx + virSumBondxx ;
00064  virSumyy = virSumPairyy + virSumBondyy ;
00065  virSumxy = virSumPairxy + virSumBondxy ;
00066  virSum = virSumPair + virSumBond;
00067  pressure = density * (vvSum + virSum) / (nAtom * NDIM);

```

```
00068
00069 }
```

References [ax](#), [ay](#), [BondEnergyPerAtom](#), [deltaT](#), [density](#), [kinEnergy](#), [nAtom](#), [NDIM](#), [potEnergy](#), [pressure](#), [solver](#), [Sqr](#), [TotalBondEnergy](#), [totEnergy](#), [uSumPair](#), [uSumPairPerAtom](#), [virSum](#), [virSumBond](#), [virSumBondxx](#), [virSumBondxy](#), [virSumBondyy](#), [virSumPair](#), [virSumPairxx](#), [virSumPairxy](#), [virSumPairyy](#), [virSumxx](#), [virSumxy](#), [virSumyy](#), [vSum](#), [vSumX](#), [vSumY](#), [vvSum](#), [vx](#), and [vy](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.72.2.18 EvalRdf()

```
void EvalRdf ( )
```

Definition at line 26 of file [EvalRdf.c](#).

```
00026     {
00027     real dr[NDIM+1], deltaR, normFac, rr, rrRange;
00028     int j1, j2, n;
00029     countRdf ++;
00030     if(countRdf == 1){
00031         for(n = 1 ; n <= sizeHistRdf ; n ++){
00032             histRdf[n] = 0.;
00033         }
00034         rrRange = Sqr(rangeRdf);
00035         deltaR = rangeRdf / sizeHistRdf;
00036         for(j1 = 1 ; j1 <= nAtom - 1 ; j1 ++){
00037             for(j2 = j1 + 1 ; j2 <= nAtom ; j2 ++){
00038
00039                 dr[1] = rx[j1] - rx[j2];
00040                 if(fabs(dr[1]) > regionH[1])
00041                     dr[1] -= SignR(region[1], dr[1]);
00042
00043                 dr[2] = ry[j1] - ry[j2];
00044                 if(fabs(dr[2]) > regionH[2])
00045                     dr[2] -= SignR(region[2], dr[2]);
00046
00047                 rr = Sqr(dr[1]) + Sqr(dr[2]);
00048
00049                 if(rr < rrRange){
00050                     n = (int)(sqrt(rr)/deltaR) + 1;
00051                     histRdf[n] ++;
00052                 }
00053             }
00054         }
00055
00056         if(countRdf == limitRdf){
00057             normFac = region[1]*region[2] / (M_PI*Sqr(deltaR)*nAtom*nAtom*countRdf );
00058             for(n = 1 ; n <= sizeHistRdf ; n ++){
00059                 histRdf[n] *= normFac/(n-0.5);
00060             }
00061             // PRINT THE RADIAL DISTRIBUTION DATA ON TO DISK FILE
00062             real rBin;
00063             int n;
00064             fprintf(fprdf, "rdf @ timeNow %lf\n", timeNow);
00065             for(n = 1 ; n <= sizeHistRdf ; n ++){
00066                 rBin = (n - 0.5)*rangeRdf/sizeHistRdf;
00067                 fprintf(fprdf, "%lf %lf\n", rBin, histRdf[n]);
00068             }
00069         }
00070     }
```

References [countRdf](#), [fprdf](#), [histRdf](#), [limitRdf](#), [nAtom](#), [NDIM](#), [rangeRdf](#), [region](#), [regionH](#), [rx](#), [ry](#), [SignR](#), [sizeHistRdf](#), [Sqr](#), and [timeNow](#).

3.72.2.19 EvalSpacetimeCorr()

```
void EvalSpacetimeCorr ( )
```

Definition at line 26 of file [EvalSpacetimeCorr.c](#).

```
00026     {
00027     real cosV=0., cosV0=0., cosV1=0., cosV2=0., sinV=0., sinV1=0., sinV2=0.;
00028     real COSA, SINA, COSV, SINV;
00029     int j, m, n, nb, ni, nv;
00030     real kMin = 2. * M_PI / region[1];
00031     real kMax = M_PI;
00032     real deltaK = (kMax - kMin) / nFunCorr;
00033
00034     for (j = 1; j <= 2*nFunCorr; j++){
00035         cfVal[j] = 0.;
00036
00037         for (n = 1; n <= nAtom; n++){
00038             j = 1;
00039             COSA = cos(kMin*rx[n]);
00040             SINA = sin(kMin*rx[n]);
00041             for (m = 1; m <= nFunCorr; m++){
00042                 if(m == 1){
00043                     cosV = cos(deltaK*rx[n]);
00044                     sinV = sin(deltaK*rx[n]);
00045                     cosV0 = cosV;
00046                 }else if(m == 2){
00047                     cosV1 = cosV;
00048                     sinV1 = sinV;
00049                     cosV = 2.*cosV0*cosV1-1;
00050                     sinV = 2.*cosV0*sinV1;
00051                 }else{
00052                     cosV2 = cosV1;
00053                     sinV2 = sinV1;
00054                     cosV1 = cosV;
00055                     sinV1 = sinV;
00056                     cosV = 2.*cosV0*cosV1-cosV2;
00057                     sinV = 2.*cosV0*sinV1-sinV2;
00058                 }
00059                 COSV = COSA*cosV - SINA*sinV;
00060                 SINV = SINA*cosV + COSA*sinV;
00061                 cfVal[j] += COSV;
00062                 cfVal[j+1] += SINV;
00063                 j += 2;
00064             }
00065         }
00066
00067         for (nb = 1; nb <= nBuffCorr; nb++){
00068             indexCorr[nb] += 1;
00069             if (indexCorr[nb] <= 0) continue;
00070             ni = nFunCorr * (indexCorr[nb] - 1);
00071             if (indexCorr[nb] == 1){
00072                 for (j = 1; j <= 2*nFunCorr; j++){
00073                     cfOrg[nb][j] = cfVal[j];
00074                 }
00075
00076                 for (j = 1; j <= nFunCorr; j++){
00077                     spacetimeCorr[nb][ni + j] = 0.;
00078
00079                     j = 1;
00080                     for (m = 1; m <= nFunCorr; m++){
00081                         nv = m + ni;
00082                         spacetimeCorr[nb][nv] += cfVal[j] * cfOrg[nb][j] + cfVal[j + 1] * cfOrg[nb][j + 1];
00083                         j += 2;
00084                     }
00085                 }
00086             }
00087
00088             // ACCUMULATE SPACETIME CORRELATIONS
00089             for (nb = 1; nb <= nBuffCorr; nb++){
00090                 if (indexCorr[nb] == nValCorr){
00091                     for (j = 1; j <= nFunCorr*nValCorr; j++){
00092                         spacetimeCorrAv[j] += spacetimeCorr[nb][j];
00093                     }
00094                     indexCorr[nb] = 0.;
00095                     countCorrAv ++;
00096                     if (countCorrAv == limitCorrAv){
00097                         for (j = 1; j <= nFunCorr*nValCorr; j++){
00098                             spacetimeCorrAv[j] /= (nAtom*limitCorrAv);
00099                             fprintf(fpdnsty,"NDIM %d\n", NDIM);
00100                             fprintf(fpdnsty,"nAtom %d\n", nAtom);
00101                             fprintf(fpdnsty,"region %lf\n", region[1]);
00102                             fprintf(fpdnsty,"nFunCorr %d\n", nFunCorr);
00103                             fprintf(fpdnsty,"limitCorrAv %d\n", limitCorrAv);
00104                             fprintf(fpdnsty,"stepCorr %d\n", stepCorr);
00105                             fprintf(fpdnsty,"nValCorr %d\n", nValCorr);
00106                             fprintf(fpdnsty,"deltaT %lf\n", deltaT);
00107                         }
00108                     }
00109                 }
00110             }
00111         }
00112     }
00113 }
```

```

00106         real tVal;
00107         for (n = 1; n <= nValCorr; n++){
00108             tVal = (n-1)*stepCorr*deltaT;
00109             fprintf (fpdnsty, "%e\t", tVal);
00110             int nn = nFunCorr*(n-1);
00111             for (j = 1; j <= nFunCorr; j++)
00112                 fprintf (fpdnsty, "%e\t", spacetimeCorrAv[nn + j]);
00113             fprintf (fpdnsty, "\n");
00114         }
00115
00116         countCorrAv = 0.;
00117         for (j = 1; j <= nFunCorr*nValCorr; j++)
00118             spacetimeCorrAv[j] = 0.;
00119     }
00120 }
00121 }
00122 }

```

References [cfOrg](#), [cfVal](#), [countCorrAv](#), [deltaT](#), [fpdnsty](#), [indexCorr](#), [limitCorrAv](#), [nAtom](#), [nBuffCorr](#), [NDIM](#), [nFunCorr](#), [nValCorr](#), [region](#), [rx](#), [spacetimeCorr](#), [spacetimeCorrAv](#), and [stepCorr](#).

3.72.2.20 EvalUnwrap()

```
void EvalUnwrap ( )
```

Definition at line 27 of file [EvalUnwrap.c](#).

```

00027     {
00028         int n;
00029         for (n = 1; n <= nAtom; n++) {
00030             rxUnwrap[n] = rx[n] + ImageX[n] * region[1];
00031             ryUnwrap[n] = ry[n] + ImageY[n] * region[2];
00032         }
00033     }

```

References [ImageX](#), [ImageY](#), [nAtom](#), [region](#), [rx](#), [rxUnwrap](#), [ry](#), and [ryUnwrap](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.72.2.21 EvalVacf()

```
void EvalVacf ( )
```

Definition at line 26 of file [EvalVacf.c](#).

```

00026     {
00027         int n, nb, ni;
00028         double viscVec = 0.;
00029         double v[3];
00030         for (n = 1 ; n <= nAtom ; n++){
00031             v[1] = vx[n] - 0.5*ax[n]*deltaT;
00032             v[2] = vy[n] - 0.5*ay[n]*deltaT;
00033             viscVec += v[1]*v[2];
00034         }
00035         viscVec += rfAtom;

```

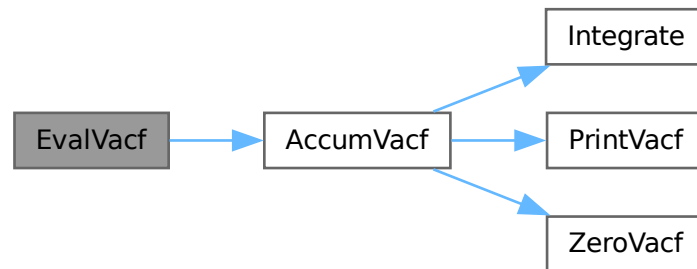
```

00036  for(nb = 1 ; nb <= nBuffAcf ; nb ++){
00037      indexAcf[nb] ++;
00038      if(indexAcf[nb] <= 0)continue;
00039      if(indexAcf[nb] == 1){
00040          viscAcfOrg[nb] = viscVec;
00041      }
00042      ni = indexAcf[nb];
00043      viscAcf[nb][ni] = viscAcfOrg[nb]*viscVec;
00044  }
00045  AccumVacf();
00046  }

```

References [AccumVacf\(\)](#), [ax](#), [ay](#), [deltaT](#), [indexAcf](#), [nAtom](#), [nBuffAcf](#), [rfAtom](#), [viscAcf](#), [viscAcfOrg](#), [vx](#), and [vy](#).

Here is the call graph for this function:



3.72.2.22 EvalVrms()

```
void EvalVrms ( )
```

Definition at line 27 of file [EvalVrms.c](#).

```

00027  {
00028      int n;
00029      VSqr = 0.0;
00030      VMeanSqr = 0.0;
00031      VRootMeanSqr = 0.0;
00032
00033      for(n = 1 ; n <= nAtom ; n ++){
00034          VSqr += Sqr(vx[n]) + Sqr(vy[n]);
00035      }
00036      VMeanSqr = VSqr/nAtom;
00037      VRootMeanSqr = sqrt(VMeanSqr);
00038  }

```

References [nAtom](#), [Sqr](#), [VMeanSqr](#), [VRootMeanSqr](#), [VSqr](#), [vx](#), and [vy](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.72.2.23 HaltConditionCheck()

```
bool HaltConditionCheck (
    double value,
    int stepCount )
```

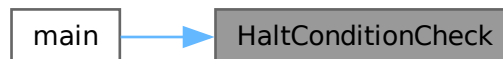
Definition at line 27 of file [Halt.c](#).

```
00027                                     {
00028
00029     if(value <= HaltCondition && value != 0) {
00030         fprintf(fpresult, "Halt condition met at step = %d with Vrms = %.10f\n", stepCount, value);
00031         return true;           // Signal that the halt condition is met
00032     }
00033     return false; // Halt condition not met
00034 }
```

References [fpresult](#), [HaltCondition](#), and [stepCount](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.72.2.24 Init()

```
void Init ( )
```

Definition at line 29 of file [Init.c](#).

```
00029     {
00030         char dummy[128];
00031         char inputConfig[128];
00032         FILE *fp;
00033         fp = fopen("input-data", "r");
00034         fscanf(fp, "%s %s", dummy, inputConfig);
00035         fscanf(fp, "%s %s", dummy, solver);
00036         fscanf(fp, "%s %s %s", dummy, xBoundary, yBoundary);
00037         fscanf(fp, "%s %d", dummy, &DampFlag);
00038         fscanf(fp, "%s %d", dummy, &freezeAtomType);
00039         fscanf(fp, "%s %lf", dummy, &rCut);
00040         fscanf(fp, "%s %lf", dummy, &gamman);
00041         fscanf(fp, "%s %lf", dummy, &kappa);
00042         fscanf(fp, "%s %lf", dummy, &deltaT);
00043         fscanf(fp, "%s %lf", dummy, &strain);
00044         fscanf(fp, "%s %lf", dummy, &FyBylx);
00045         fscanf(fp, "%s %lf", dummy, &fxByfy);
00046         fscanf(fp, "%s %lf", dummy, &DeltaY);
00047         fscanf(fp, "%s %lf", dummy, &DeltaX);
00048         fscanf(fp, "%s %lf", dummy, &HaltCondition);
00049         fscanf(fp, "%s %d", dummy, &stepAvg);
00050         fscanf(fp, "%s %d", dummy, &stepEquil);
00051         fscanf(fp, "%s %d", dummy, &stepLimit);
00052         fscanf(fp, "%s %d", dummy, &stepDump);
00053         fscanf(fp, "%s %d", dummy, &stepTraj);
00054         fscanf(fp, "%s %d", dummy, &limitCorrAv);
00055         fscanf(fp, "%s %d", dummy, &nBuffCorr);
00056         fscanf(fp, "%s %d", dummy, &nFunCorr);
00057         fscanf(fp, "%s %d", dummy, &nValCorr);
00058         fscanf(fp, "%s %d", dummy, &stepCorr);
00059         fscanf(fp, "%s %d", dummy, &limitAcfAv);
```

```

00060     fscanf(fp, "%s %d", dummy, &nBuffAcf);
00061     fscanf(fp, "%s %d", dummy, &nValAcf);
00062     fscanf(fp, "%s %d", dummy, &stepAcf);
00063     fscanf(fp, "%s %lf", dummy, &rangeRdf);
00064     fscanf(fp, "%s %d", dummy, &limitRdf);
00065     fscanf(fp, "%s %d", dummy, &sizeHistRdf);
00066     fscanf(fp, "%s %d", dummy, &stepRdf);
00067
00068     fclose(fp);
00069     FILE *fpSTATE;
00070     if((fpSTATE = fopen(inputConfig,"r"))==NULL){
00071         printf("Error occurred: Could not open STATE file\n Exiting now..\n");
00072         exit(0);
00073     }
00074
00075     fscanf(fpSTATE, "%s %lf", dummy, &timeNow);
00076     fscanf(fpSTATE, "%s %d", dummy, &nAtom);
00077     fscanf(fpSTATE, "%s %d", dummy, &nBond);
00078     fscanf(fpSTATE, "%s %d", dummy, &nAtomType);
00079     fscanf(fpSTATE, "%s %d", dummy, &nBondType);
00080     fscanf(fpSTATE, "%s %lf", dummy, &region[1]);
00081     fscanf(fpSTATE, "%s %lf", dummy, &region[2]);
00082
00083     region[2] *= 1.5; //Remove this when put on GitHub
00084
00085     density = nAtom/(region[1]*region[2]);
00086     cells[1] = region[1] / rCut;
00087     cells[2] = region[2] / rCut;
00088     cellList = (int*)malloc((nAtom + cells[1] * cells[2] + 1) * sizeof(int));
00089     regionH[1] = 0.5*region[1];
00090     regionH[2] = 0.5*region[2];
00091
00092     //strain information
00093     strainRate = strain/deltaT;
00094     shearDisplacement = strain * region[2];
00095     shearVelocity = strainRate * region[2];
00096     int n;
00097
00098     rx = (double*)malloc((nAtom + 1) * sizeof(double));
00099     ry = (double*)malloc((nAtom + 1) * sizeof(double));
00100     vx = (double*)malloc((nAtom + 1) * sizeof(double));
00101     vy = (double*)malloc((nAtom + 1) * sizeof(double));
00102     ax = (double*)malloc((nAtom + 1) * sizeof(double));
00103     ay = (double*)malloc((nAtom + 1) * sizeof(double));
00104     fax = (double*)malloc((nAtom + 1) * sizeof(double));
00105     fay = (double*)malloc((nAtom + 1) * sizeof(double));
00106     atomID = (int*)malloc((nAtom+1) * sizeof(int));
00107     atomType = (int*)malloc((nAtom+1) * sizeof(int));
00108     atomRadius = (double*)malloc((nAtom + 1) * sizeof(double));
00109     atomMass = (double*)malloc((nAtom + 1) * sizeof(double));
00110     speed = (double*)malloc((nAtom + 1) * sizeof(double));
00111     atom1 = (int*)malloc((nBond+1)*sizeof(int));
00112     atom2 = (int*)malloc((nBond+1)*sizeof(int));
00113     BondID = (int*)malloc((nBond+1)*sizeof(int));
00114     BondType = (int*)malloc((nBond+1)*sizeof(int));
00115     kb = (double*)malloc((nBond+1)*sizeof(double));
00116     ro = (double*)malloc((nBond+1)*sizeof(double));
00117     BondEnergy = (double*)malloc((nBond+1)*sizeof(double));
00118     BondLength = (double*)malloc((nBond+1)*sizeof(double));
00119     discDragx = (double*)malloc((nAtom + 1) * sizeof(double));
00120     discDragy = (double*)malloc((nAtom + 1) * sizeof(double));
00121     nodeDragx = (double*)malloc((nAtom + 1) * sizeof(double));
00122     nodeDragy = (double*)malloc((nAtom + 1) * sizeof(double));
00123     ImageX = (int*)malloc((nAtom+1) * sizeof(int));
00124     ImageY = (int*)malloc((nAtom+1) * sizeof(int));
00125     rxUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
00126     ryUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
00127     DeltaXiJOld = (double*)malloc((nBond+1)*sizeof(double));
00128     DeltaYiJOld = (double*)malloc((nBond+1)*sizeof(double));
00129     DeltaXiJOldPair = (double**)malloc((nAtom+1) * sizeof(double*));
00130     DeltaYiJOldPair = (double**)malloc((nAtom+1) * sizeof(double*));
00131     for(int n = 0; n <= nAtom; n++) {
00132         DeltaXiJOldPair[n] = (double*)malloc((nAtom+1) * sizeof(double));
00133         DeltaYiJOldPair[n] = (double*)malloc((nAtom+1) * sizeof(double));
00134     }
00135     molID = (int*)malloc((nAtom+1) * sizeof(int));
00136
00137     for(n = 1; n <= nAtom; n++){
00138         atomMass[n] = 1.0;
00139     }
00140
00141     fscanf(fpSTATE, "%s\n", dummy);
00142     for(n = 1; n <= nAtom; n++){
00143         fscanf(fpSTATE, "%d %d %d %lf %lf %lf %lf %lf\n", &atomID[n], &molID[n], &atomType[n],
00144             &atomRadius[n], &rx[n], &ry[n], &vx[n], &vy[n]);
00145

```

```

00146     fscanf(fpSTATE, "%s\n", dummy);
00147     for(n=1; n<=nBond; n++)
00148         fscanf(fpSTATE, "%d %d %d %d %lf %lf\n", &BondID[n], &BondType[n], &atom1[n], &atom2[n], &kb[n],
&ro[n]);
00149
00150     fclose(fpSTATE);
00151
00152     //2D-List of bonded atoms. This is used to remove pair interaction
00153     //calculation for the bonded atoms
00154     isBonded = (int**)malloc((nAtom + 1) * sizeof(int*));
00155     for (int i = 0; i <= nAtom; i++) {
00156         isBonded[i] = (int*)malloc((nAtom + 1) * sizeof(int));
00157         for (int j = 0; j <= nAtom; j++) {
00158             isBonded[i][j] = 0;
00159         }
00160     }
00161
00162     for (n = 1; n <= nBond; n++) {
00163         int i = atom1[n];
00164         int j = atom2[n];
00165         isBonded[i][j] = 1;
00166         isBonded[j][i] = 1; // symmetric
00167     }
00168
00169     //For thermostat, update in final version
00170     thermo = 'C';
00171
00172
00173     // List the interface atoms
00174     nAtomInterface = 0;
00175     nAtomBlock = 0;
00176     nDiscInterface = 0;
00177     double InterfaceWidth, bigDiameter;
00178     bigDiameter = 2.8;
00179     InterfaceWidth = 5.0 * bigDiameter;
00180
00181     for(n = 1; n <= nAtom; n++){
00182         if(fabs(ry[n]) < InterfaceWidth){
00183             nAtomInterface++;
00184         }
00185         if(molID[n] == 2){
00186             nAtomBlock++;
00187         }
00188         if(atomRadius[n] != 0.0){
00189             nDiscInterface++;
00190         }
00191     }
00192     atomIDInterface = (int*)malloc((nAtomInterface+1)*sizeof(int));
00193
00194     int m;
00195     m = 1;
00196     for(n=1; n<=nAtom; n++){
00197         if(fabs(ry[n]) < InterfaceWidth){
00198             atomIDInterface[m] = atomID[n];
00199             m++;
00200         }
00201     }
00202     nPairTotal = 0.5 * nAtomInterface * (nAtomInterface-1);
00203     PairID = (int*)malloc((nPairTotal+1) * sizeof(int));
00204     Pairatom1 = (int*)malloc((nPairTotal+1) * sizeof(int));
00205     Pairatom2 = (int*)malloc((nPairTotal+1) * sizeof(int));
00206     PairXij = (double*)malloc((nPairTotal+1) * sizeof(double));
00207     PairYij = (double*)malloc((nPairTotal+1) * sizeof(double));
00208
00209     fprintf(fpresult, "-----\n");
00210     fprintf(fpresult, "-----PARAMETERS-----\n");
00211     fprintf(fpresult, "-----\n");
00212     fprintf(fpresult, "nAtom\t\t\t%d\n", nAtom);
00213     fprintf(fpresult, "nBond\t\t\t%d\n", nBond);
00214     fprintf(fpresult, "nAtomBlock\t\t\t%d\n", nAtomBlock);
00215     fprintf(fpresult, "nAtomInterface\t\t\t%d\n", nAtomInterface);
00216     fprintf(fpresult, "nDiscInterface\t\t\t%d\n", nDiscInterface);
00217     fprintf(fpresult, "gamman\t\t\t0.6g\n", gamman);
00218     fprintf(fpresult, "strain\t\t\t0.6g\n", strain);
00219     fprintf(fpresult, "strainRate\t\t\t0.6g\n", strainRate);
00220     fprintf(fpresult, "FyBylx\t\t\t0.6g\n", FyBylx);
00221     fprintf(fpresult, "fxByfy\t\t\t0.6g\n", fxByfy);
00222     fprintf(fpresult, "DeltaY\t\t\t0.6g\n", DeltaY);
00223     fprintf(fpresult, "DeltaX\t\t\t0.6g\n", DeltaX);
00224     fprintf(fpresult, "HaltCondition\t\t\t0.6g\n", HaltCondition);
00225     fprintf(fpresult, "kappa\t\t\tg\n", kappa);
00226     fprintf(fpresult, "density\t\t\tg\n", density);
00227     fprintf(fpresult, "rCut\t\t\tg\n", rCut);
00228     fprintf(fpresult, "deltaT\t\t\tg\n", deltaT);
00229     fprintf(fpresult, "stepEquil\t\t\t%d\n", stepEquil);
00230     fprintf(fpresult, "stepLimit\t\t\t%d\n", stepLimit);
00231     fprintf(fpresult, "region[1]\t\t\t0.16lf\n", region[1]);

```

```

00232     fprintf(fpresult, "region[2]\t\t\t%0.16lf\n", region[2]);
00233     fprintf(fpresult, "cells[1]\t\t\t%d\n", cells[1]);
00234     fprintf(fpresult, "cells[2]\t\t\t%d\n", cells[2]);
00235     fprintf(fpresult, "solver\t\t\t%s\n", solver);
00236     fprintf(fpresult, "boundary\t\t\t%s %s\n", xBoundary, yBoundary);
00237     fprintf(fpresult, "DampFlag\t\t\t%d\n", DampFlag);
00238
00239
00240     fprintf(fpresult, "-----\n");
00241     fprintf(fpresult, "#TimeNow TotalMomentum PotEngyPerAtom KinEngyPerAtom TotEngyPerAtom
PairEnergyPerAtom BondEnergyPerAtom Press VirialSum\n");
00242     fprintf(fpvrms, "#timeNow\tVrms \n");
00243     fprintf(fpcom, "#timeNow\tComX\tComY\n");
00244
00245     /* //Uncomment the following as per your acquirement
00246     fprintf(fpstress, "strain                %lf\n", strain);
00247     fprintf(fpstress, "region[1]            %lf\n", region[1]);
00248     fprintf(fpstress, "region[2]            %lf\n", region[2]);
00249     fprintf(fpstress, "#timeNow virSumxx virSumyy virSumxy pressure\n");
00250     fprintf(fpmomentum, "#timeNow Px Py\n");
00251 */
00252
00253     if((strcmp(xBoundary, "p") != 0 && strcmp(xBoundary, "r") != 0) ||
00254        (strcmp(yBoundary, "p") != 0 && strcmp(yBoundary, "r") != 0)) {
00255         fprintf(fpresult, "Error: Invalid boundary value detected: '%s %s'. Only 'p' or 'r' are
allowed.\n", xBoundary, yBoundary);
00256         exit(EXIT_FAILURE); // Exit with failure status
00257     }
00258
00259 }

```

References [atom1](#), [atom2](#), [atomID](#), [atomIDInterface](#), [atomMass](#), [atomRadius](#), [atomType](#), [ax](#), [ay](#), [BondEnergy](#), [BondID](#), [BondLength](#), [BondType](#), [cellList](#), [cells](#), [DampFlag](#), [deltaT](#), [DeltaX](#), [DeltaXijOld](#), [DeltaXijOldPair](#), [DeltaY](#), [DeltaYijOld](#), [DeltaYijOldPair](#), [density](#), [discDragx](#), [discDragy](#), [fax](#), [fay](#), [fpcom](#), [fpresult](#), [fpvrms](#), [freezeAtomType](#), [fxByfy](#), [FyBylx](#), [gamman](#), [HaltCondition](#), [ImageX](#), [ImageY](#), [isBonded](#), [kappa](#), [kb](#), [limitAcfAv](#), [limitCorrAv](#), [limitRdf](#), [molID](#), [nAtom](#), [nAtomBlock](#), [nAtomInterface](#), [nAtomType](#), [nBond](#), [nBondType](#), [nBuffAcf](#), [nBuffCorr](#), [nDiscInterface](#), [nFunCorr](#), [nodeDragx](#), [nodeDragy](#), [nPairTotal](#), [nValAcf](#), [nValCorr](#), [Pairatom1](#), [Pairatom2](#), [PairID](#), [PairXij](#), [PairYij](#), [rangeRdf](#), [rCut](#), [region](#), [regionH](#), [ro](#), [rx](#), [rxUnwrap](#), [ry](#), [ryUnwrap](#), [shearDisplacement](#), [shearVelocity](#), [sizeHistRdf](#), [solver](#), [speed](#), [stepAcf](#), [stepAvg](#), [stepCorr](#), [stepDump](#), [stepEquil](#), [stepLimit](#), [stepRdf](#), [stepTraj](#), [strain](#), [strainRate](#), [thermo](#), [timeNow](#), [vx](#), [vy](#), [xBoundary](#), and [yBoundary](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.72.2.25 LeapfrogStep()

```
void LeapfrogStep ( )
```

3.72.2.26 main()

```

int main (
    int argc,
    char ** argv )

```

Definition at line 51 of file [main.c](#).

```

00051                                     {
00052     time_t t1 = 0, t2;
00053     if (argc < 2) {
00054         fprintf(stderr, "Usage: %s <output_prefix>\n", argv[0]);
00055         return 1;
00056     }
00057     int prefix_size = snprintf(NULL, 0, "../output/%s", argv[1]) + 1; // +1 for the null terminator
00058     prefix = malloc(prefix_size);
00059     if(prefix == NULL) {
00060         fprintf(stderr, "Memory allocation failed\n");
00061         return 1;
00062     }
00063
00064     // Write the formatted string into the allocated space
00065     snprintf(prefix, prefix_size, "../output/%s", argv[1]);
00066     sprintf(result, "%s.result", prefix);
00067     fpresult = fopen(result, "w");
00068     sprintf(xyz, "%s.xyz", prefix);
00069     fpxyz = fopen(xyz, "w");
00070     sprintf(vrms, "%s.vrms", prefix);
00071     fpvrms = fopen(vrms, "w");
00072     sprintf(bond, "%s.bond", prefix);
00073     fpbond = fopen(bond, "w");
00074     sprintf(com, "%s.com", prefix);
00075     fpcom = fopen(com, "w");
00076     sprintf(pair, "%s.pair", prefix);
00077     fppair = fopen(pair, "w");
00078
00079     /* //Uncomment the following as per your acquirement
00080     sprintf(dnsty, "%s.curr-dnsty", prefix);
00081     fpdnsty = fopen(dnsty, "w");
00082     sprintf(visc, "%s.viscosity", prefix);
00083     fpvisc = fopen(visc, "w");
00084     sprintf(rdf, "%s.rdf", prefix);
00085     fprdf = fopen(rdf, "w");
00086     sprintf(stress, "%s.stress", prefix);
00087     fpstress = fopen(stress, "w");
00088     sprintf(momentum, "%s.momentum", prefix);
00089     fpmomentum = fopen(momentum, "w");
00090     */
00091
00092     Init();
00093     SetupJob();
00094     t1 = time(NULL);
00095     moreCycles = 1;
00096     timeNow = 0.0;
00097     if(timeNow == 0.0) {
00098         DisplaceAtoms();
00099         ComputePairForce(1);
00100         ComputeBondForce();
00101         ApplyForce();
00102         DumpBonds();
00103         DumpPairs();
00104         Trajectory();
00105         EvalUnwrap();
00106         ApplyBoundaryCond();
00107         EvalProps();
00108         EvalVrms();
00109         EvalCom();
00110         PrintVrms();
00111         PrintCom();
00112         PrintSummary();
00113     }
00114
00115     //Here starts the main loop of the program
00116     while(moreCycles){
00117         if(stepLimit == 0){
00118             exit(0);
00119         }
00120
00121         stepCount++;
00122         timeNow = stepCount * deltaT; //for adaptive step size: timeNow += deltaT
00123
00124         VelocityVerletStep(1);
00125         EvalUnwrap();
00126         ApplyBoundaryCond();
00127         ComputePairForce(1);
00128         ComputeBondForce();
00129         ApplyForce();
00130         VelocityVerletStep(2);
00131         ApplyBoundaryCond();
00132         EvalProps();
00133         EvalVrms();
00134         EvalCom();
00135         if(stepCount % stepAvg == 0){
00136             PrintSummary();

```



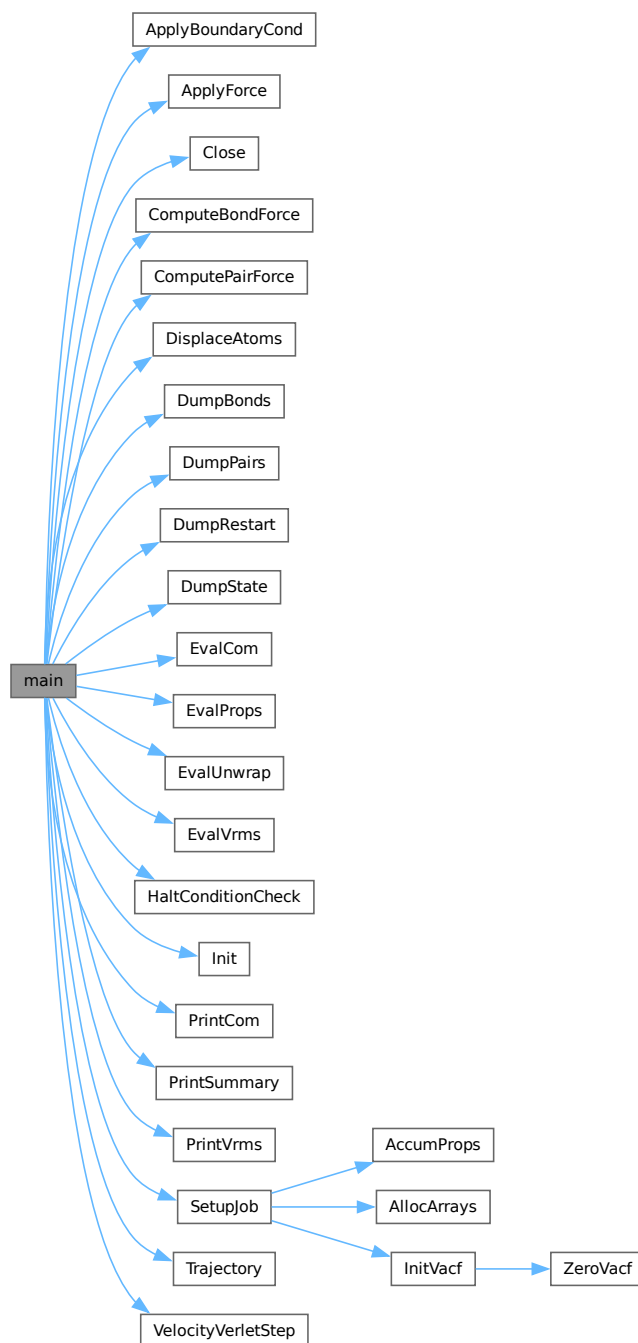
```

00137     PrintVrms();
00138     PrintCom();
00139 }
00140 if(stepCount % stepTraj == 0){
00141     Trajectory();
00142     DumpBonds();
00143     DumpPairs();
00144 }
00145 if(stepCount % stepDump == 0){
00146     DumpRestart(); // Save the current state for input
00147     DumpState();   // Save the current state for config
00148 }
00149 if(HaltConditionCheck(VRootMeanSqr, stepCount)) {
00150     DumpRestart(); // Save the current state for input
00151     DumpState();   // Save the current state for config
00152     break; // Exit the loop when the halt condition is met
00153 }
00154
00155 if(stepCount >= stepLimit)
00156     moreCycles = 0;
00157 }
00158
00159
00160 t2 = time(NULL);
00161 fprintf(fpresult, "#Execution time %lf secs\n", difftime(t2,t1));
00162 fprintf(fpresult, "#Execution speed %lf steps per secs\n", stepLimit/difftime(t2,t1));
00163
00164 fclose(fpresult);
00165 fclose(fpxyz);
00166 fclose(fpvrms);
00167 fclose(fpbond);
00168 fclose(fppair);
00169 fclose(fpcom);
00170
00171 /*//Uncomment the following as per your acquirement
00172 fclose(fpdnsty);
00173 fclose(fpvisc);
00174 fclose(fprdf);
00175 fclose(fpstress);
00176 fclose(fpmomentum);
00177 */
00178
00179 free(prefix);
00180 Close();
00181 return 0;
00182 }

```

References [ApplyBoundaryCond\(\)](#), [ApplyForce\(\)](#), [bond](#), [Close\(\)](#), [com](#), [ComputeBondForce\(\)](#), [ComputePairForce\(\)](#), [deltaT](#), [DisplaceAtoms\(\)](#), [DumpBonds\(\)](#), [DumpPairs\(\)](#), [DumpRestart\(\)](#), [DumpState\(\)](#), [EvalCom\(\)](#), [EvalProps\(\)](#), [EvalUnwrap\(\)](#), [EvalVrms\(\)](#), [fpbond](#), [fpcom](#), [fppair](#), [fpresult](#), [fpvrms](#), [fpxyz](#), [HaltConditionCheck\(\)](#), [Init\(\)](#), [moreCycles](#), [pair](#), [prefix](#), [PrintCom\(\)](#), [PrintSummary\(\)](#), [PrintVrms\(\)](#), [result](#), [SetupJob\(\)](#), [stepAvg](#), [stepCount](#), [stepDump](#), [stepLimit](#), [stepTraj](#), [timeNow](#), [Trajectory\(\)](#), [VelocityVerletStep\(\)](#), [vrms](#), [VRootMeanSqr](#), and [xyz](#).

Here is the call graph for this function:



3.72.2.27 PrintCom()

```
void PrintCom ( )
```

Definition at line 28 of file [PrintCom.c](#).

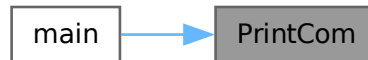
```
00028 {
```

```
00029 fprintf(fpcom, "%0.4lf\t%0.16lf\t%0.16lf\n", timeNow, ComX, ComY);
00030 fflush(fpcom);
00031 }
```

References [ComX](#), [ComY](#), [fpcom](#), and [timeNow](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.72.2.28 PrintMomentum()

```
void PrintMomentum ( )
```

Definition at line 25 of file [PrintMomentum.c](#).

```
00025 {
00026     fprintf(fpmomentum, "%0.4lf\t%0.16lf\t%0.16lf\n", timeNow, vSumX, vSumY);
00027     fflush(fpmomentum);
00028 }
```

References [fpmomentum](#), [timeNow](#), [vSumX](#), and [vSumY](#).

3.72.2.29 PrintStress()

```
void PrintStress ( )
```

Definition at line 25 of file [PrintStress.c](#).

```
00025 {
00026     fprintf(fpstress, "%0.4lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\n", timeNow, virSumxx, virSumyy,
00027         virSumxy, pressure);
00027     fflush(fpstress);
00028 }
```

References [fpstress](#), [pressure](#), [timeNow](#), [virSumxx](#), [virSumxy](#), and [virSumyy](#).

3.72.2.30 PrintSummary()

```
void PrintSummary ( )
```

Definition at line 4 of file [PrintSummary.c](#).

```
00004      {  
00005  fprintf(fpresult, "%0.4lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\n",  
00006      timeNow, vSum, potEnergy, kinEnergy, totEnergy, uSumPairPerAtom, BondEnergyPerAtom, pressure,  
      virSum);  
00007  fflush(fpresult);  
00008  }
```

References [BondEnergyPerAtom](#), [fpresult](#), [kinEnergy](#), [potEnergy](#), [pressure](#), [timeNow](#), [totEnergy](#), [uSumPairPerAtom](#), [virSum](#), and [vSum](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.72.2.31 PrintVrms()

```
void PrintVrms ( )
```

Definition at line 27 of file [PrintVrms.c](#).

```
00027      {  
00028  fprintf(fpvrms, "%0.4lf\t%0.16lf\n", timeNow, VRootMeanSqr);  
00029  fflush(fpvrms);  
00030  }
```

References [fpvrms](#), [timeNow](#), and [VRootMeanSqr](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.72.2.32 SetupJob()

```
void SetupJob ( )
```

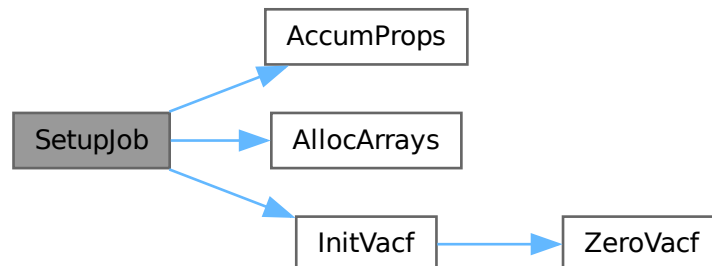
Definition at line 27 of file [SetupJob.c](#).

```
00027     {  
00028     AllocArrays ();  
00029     AccumProps (0);  
00030     InitVacf ();  
00031     stepCount = 0;  
00032     // INITIALISE SPACETIME CORRELATIONS  
00033     int n;  
00034     for (n = 1; n <= nBuffCorr; n++)  
00035         indexCorr[n] = -(n - 1)*nValCorr/nBuffCorr;  
00036  
00037     countCorrAv = 0.;  
00038  
00039     for (n = 1; n <= nFunCorr*nValCorr; n++)  
00040         spacetimeCorrAv[n] = 0.;  
00041  
00042     //RDF  
00043     countRdf = 0;  
00044 }
```

References [AccumProps\(\)](#), [AllocArrays\(\)](#), [countCorrAv](#), [countRdf](#), [indexCorr](#), [InitVacf\(\)](#), [nBuffCorr](#), [nFunCorr](#), [nValCorr](#), [spacetimeCorrAv](#), and [stepCount](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.72.2.33 Trajectory()

```
void Trajectory ( )
```

Definition at line 25 of file [Trajectory.c](#).

```
00025     {
00026     int n;
00027     //Trajectory file in LAMMPS dump format for OVITO visualization
00028     fprintf(fpxyz, "ITEM: TIMESTEP\n");
00029     fprintf(fpxyz, "%lf\n", timeNow);
00030     fprintf(fpxyz, "ITEM: NUMBER OF ATOMS\n");
00031     fprintf(fpxyz, "%d\n", nAtom);
00032     fprintf(fpxyz, "ITEM: BOX BOUNDS pp ff pp\n");
00033     fprintf(fpxyz, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
00034     fprintf(fpxyz, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
00035     fprintf(fpxyz, "%lf %lf zlo zhi\n", -0.1, 0.1);
00036     fprintf(fpxyz, "ITEM: ATOMS id mol type radius x y vx vy fx fy\n");
00037     for(n=1; n<=nAtom; n++)
00038         fprintf(fpxyz, "%d\t %d\t %d\t %0.21f\t %0.161f\t %0.161f\t %0.161f\t %0.161f\t %0.161f\t\n",
00039             atomID[n], molID[n], atomType[n], atomRadius[n], rx[n], ry[n], vx[n], vy[n], ax[n], ay[n]);
00040 }
```

References [atomID](#), [atomRadius](#), [atomType](#), [ax](#), [ay](#), [fpxyz](#), [molID](#), [nAtom](#), [regionH](#), [rx](#), [ry](#), [timeNow](#), [vx](#), and [vy](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.72.2.34 VelocityVerletStep()

```
void VelocityVerletStep (
    int icode )
```

Definition at line 26 of file [VelocityVerletStep.c](#).

```
00026     {
00027     int n;
00028     if(icode == 1){
00029     for (n= 1; n <= nAtom; n++) {
00030     if(atomType[n] != freezeAtomType){
00031     vx[n] += ax[n] * 0.5 * deltaT;
00032     vy[n] += ay[n] * 0.5 * deltaT;
00033     rx[n] += vx[n] * deltaT;
00034     ry[n] += vy[n] * deltaT;
00035     }
00036     //Calculating the image flags here
00037     if (rx[n] >= regionH[1]) {
00038     rx[n] -= region[1];
00039     ImageX[n]++;
00040     } else if (rx[n] < -regionH[1]) {
00041     rx[n] += region[1];
00042     ImageX[n]--;
00043     }
00044     if (ry[n] >= regionH[2]) {
00045     ry[n] -= region[2];
00046     ImageY[n]++;
00047     } else if (ry[n] < -regionH[2]) {
00048     ry[n] += region[2];
```

```

00049     ImageY[n]--;
00050     } } }
00051     else if(icode == 2){
00052     for(n = 1; n <= nAtom; n++) {
00053     if(atomType[n] != freezeAtomType){
00054         vx[n] += ax[n] * 0.5 * deltaT;
00055         vy[n] += ay[n] * 0.5 * deltaT;
00056     } } }

```

References [atomType](#), [ax](#), [ay](#), [deltaT](#), [freezeAtomType](#), [ImageX](#), [ImageY](#), [nAtom](#), [region](#), [regionH](#), [rx](#), [ry](#), [vx](#), and [vy](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.72.3 Variable Documentation

3.72.3.1 prefix

```
char* prefix = NULL
```

Definition at line 13 of file [main.c](#).

Referenced by [DumpRestart\(\)](#), [DumpState\(\)](#), and [main\(\)](#).

3.73 main.c

[Go to the documentation of this file.](#)

```

00001 #include<stdio.h>
00002 #include<string.h>
00003 #include<stdlib.h>
00004 #include <stdbool.h>
00005 #include <time.h>
00006 #include <mpi.h>
00007 #define DEFINE_GLOBALS
00008 #include "global.h"
00009 #include "ComputeBondForce.h"
00010 #include "ComputePairForce.h"
00011
00012
00013 char *prefix = NULL; // Definition of prefix
00014
00015 void Init();
00016 void SetupJob();
00017 void EvalSpacetimeCorr();
00018 void Trajectory();
00019 void DumpState();
00020 void ComputeForcesCells();
00021 void LeapfrogStep();
00022 void BrownianStep();
00023 void ApplyBoundaryCond();
00024 void EvalProps();
00025 void EvalVacf();

```

```

00026 void EvalRdf();
00027 void AccumProps(int icode);
00028 void PrintSummary();
00029 void PrintVrms();
00030 //void ComputeBondForce();
00031 void DumpBonds();
00032 void VelocityVerletStep(int icode);
00033 void ApplyForce();
00034 void ApplyDrivingForce();
00035 void ApplyShear();
00036 void ApplyLeesEdwardsBoundaryCond();
00037 void PrintStress();
00038 void Close();
00039 //void ComputePairForce(int normFlag);
00040 void PrintMomentum();
00041 void DisplaceAtoms();
00042 void DumpRestart();
00043 bool HaltConditionCheck(double value, int stepCount);
00044 void EvalCom();
00045 void PrintCom();
00046 void EvalVrms();
00047 void EvalUnwrap();
00048 void DumpPairs();
00049 void ApplyViscous();
00050
00051 int main(int argc, char **argv) {
00052     time_t t1 = 0, t2;
00053     if (argc < 2) {
00054         fprintf(stderr, "Usage: %s <output_prefix>\n", argv[0]);
00055         return 1;
00056     }
00057     int prefix_size = snprintf(NULL, 0, "../output/%s", argv[1]) + 1; // +1 for the null terminator
00058     prefix = malloc(prefix_size);
00059     if(prefix == NULL) {
00060         fprintf(stderr, "Memory allocation failed\n");
00061         return 1;
00062     }
00063
00064     // Write the formatted string into the allocated space
00065     snprintf(prefix, prefix_size, "../output/%s", argv[1]);
00066     sprintf(result, "%s.result", prefix);
00067     fpresult = fopen(result, "w");
00068     sprintf(xyz, "%s.xyz", prefix);
00069     fpxyz = fopen(xyz, "w");
00070     sprintf(vrms, "%s.vrms", prefix);
00071     fpvrms = fopen(vrms, "w");
00072     sprintf(bond, "%s.bond", prefix);
00073     fpbond = fopen(bond, "w");
00074     sprintf(com, "%s.com", prefix);
00075     fpcom = fopen(com, "w");
00076     sprintf(pair, "%s.pair", prefix);
00077     fppair = fopen(pair, "w");
00078
00079     /* //Uncomment the following as per your acquirement
00080     sprintf(dnsty, "%s.curr-dnsty", prefix);
00081     fpdnsty = fopen(dnsty, "w");
00082     sprintf(visc, "%s.viscosity", prefix);
00083     fpvisc = fopen(visc, "w");
00084     sprintf(rdf, "%s.rdf", prefix);
00085     fprdf = fopen(rdf, "w");
00086     sprintf(stress, "%s.stress", prefix);
00087     fpstress = fopen(stress, "w");
00088     sprintf(momentum, "%s.momentum", prefix);
00089     fpmomentum = fopen(momentum, "w");
00090     */
00091
00092     Init();
00093     SetupJob();
00094     t1 = time(NULL);
00095     moreCycles = 1;
00096     timeNow = 0.0;
00097     if(timeNow == 0.0) {
00098         DisplaceAtoms();
00099         ComputePairForce(1);
00100         ComputeBondForce();
00101         ApplyForce();
00102         DumpBonds();
00103         DumpPairs();
00104         Trajectory();
00105         EvalUnwrap();
00106         ApplyBoundaryCond();
00107         EvalProps();
00108         EvalVrms();
00109         EvalCom();
00110         PrintVrms();
00111         PrintCom();
00112         PrintSummary();

```



```

00113     }
00114
00115 //Here starts the main loop of the program
00116 while(moreCycles){
00117     if(stepLimit == 0){
00118         exit(0);
00119     }
00120
00121     stepCount ++;
00122     timeNow = stepCount * deltaT; //for adaptive step size: timeNow += deltaT
00123
00124     VelocityVerletStep(1);
00125     EvalUnwrap();
00126     ApplyBoundaryCond();
00127     ComputePairForce(1);
00128     ComputeBondForce();
00129     ApplyForce();
00130     VelocityVerletStep(2);
00131     ApplyBoundaryCond();
00132     EvalProps();
00133     EvalVrms();
00134     EvalCom();
00135     if(stepCount % stepAvg == 0){
00136         PrintSummary();
00137         PrintVrms();
00138         PrintCom();
00139     }
00140     if(stepCount % stepTraj == 0){
00141         Trajectory();
00142         DumpBonds();
00143         DumpPairs();
00144     }
00145     if(stepCount % stepDump == 0){
00146         DumpRestart(); // Save the current state for input
00147         DumpState(); // Save the current state for config
00148     }
00149     if(HaltConditionCheck(VRootMeanSqr, stepCount)) {
00150         DumpRestart(); // Save the current state for input
00151         DumpState(); // Save the current state for config
00152         break; // Exit the loop when the halt condition is met
00153     }
00154
00155     if(stepCount >= stepLimit)
00156         moreCycles = 0;
00157 }
00158
00159
00160 t2 = time(NULL);
00161 fprintf(fpresult, "#Execution time %lf secs\n", difftime(t2,t1));
00162 fprintf(fpresult, "#Execution speed %lf steps per secs\n", stepLimit/difftime(t2,t1));
00163
00164 fclose(fpresult);
00165 fclose(fpxyz);
00166 fclose(fpvrms);
00167 fclose(fpbond);
00168 fclose(fppair);
00169 fclose(fpcom);
00170
00171 /*//Uncomment the following as per your acquirement
00172     fclose(fpdnsty);
00173     fclose(fpvisc);
00174     fclose(fprdf);
00175     fclose(fpstress);
00176     fclose(fpmomentum);
00177 */
00178
00179 free(prefix);
00180 Close();
00181 return 0;
00182 }

```

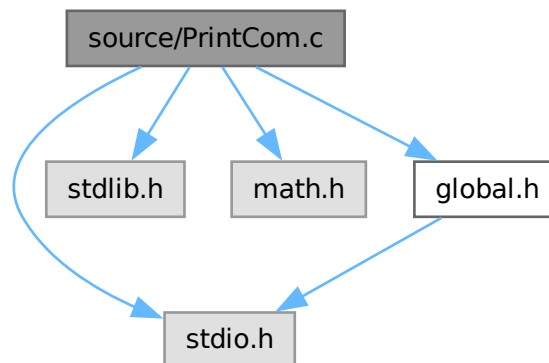
3.74 source/PrintCom.c File Reference

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "global.h"

```

Include dependency graph for PrintCom.c:



Functions

- void [PrintCom](#) ()

3.74.1 Function Documentation

3.74.1.1 PrintCom()

```
void PrintCom ( )
```

Definition at line 28 of file [PrintCom.c](#).

```
00028     {  
00029     fprintf(fpcom, "%0.4lf\t%0.16lf\t%0.16lf\n", timeNow, ComX, ComY);  
00030     fflush(fpcom);  
00031     }
```

References [ComX](#), [ComY](#), [fpcom](#), and [timeNow](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.75 PrintCom.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022
00023 #include<stdio.h>
00024 #include<stdlib.h>
00025 #include<math.h>
00026 #include"global.h"
00027
00028 void PrintCom(){
00029     fprintf(fpcom, "%0.4lf\t%0.16lf\t%0.16lf\n", timeNow, ComX, ComY);
00030     fflush(fpcom);
00031 }
00032
00033
00034

```

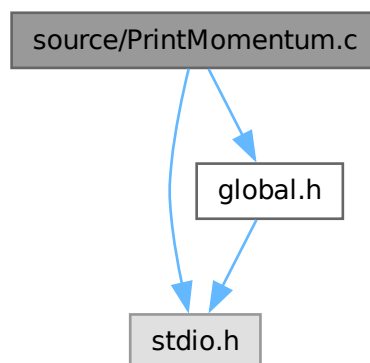
3.76 source/PrintMomentum.c File Reference

```

#include <stdio.h>
#include "global.h"

```

Include dependency graph for PrintMomentum.c:



Functions

- void [PrintMomentum](#) ()

3.76.1 Function Documentation

3.76.1.1 PrintMomentum()

```
void PrintMomentum ( )
```

Definition at line 25 of file [PrintMomentum.c](#).

```
00025     {
00026     fprintf(fpmomentum, "%0.4lf\t%0.16lf\t%0.16lf\n", timeNow, vSumX, vSumY);
00027     fflush(fpmomentum);
00028 }
```

References [fpmomentum](#), [timeNow](#), [vSumX](#), and [vSumY](#).

3.77 PrintMomentum.c

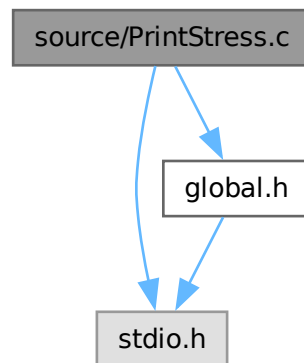
[Go to the documentation of this file.](#)

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 void PrintMomentum(){
00026     fprintf(fpmomentum, "%0.4lf\t%0.16lf\t%0.16lf\n", timeNow, vSumX, vSumY);
00027     fflush(fpmomentum);
00028 }
```

3.78 source/PrintStress.c File Reference

```
#include <stdio.h>
#include "global.h"
```

Include dependency graph for PrintStress.c:



Functions

- void [PrintStress](#) ()

3.78.1 Function Documentation

3.78.1.1 PrintStress()

```
void PrintStress ( )
```

Definition at line 25 of file [PrintStress.c](#).

```
00025     {
00026     fprintf(fpstress, "%0.4lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\n", timeNow, virSumxx, virSumyy,
virSumxy, pressure);
00027     fflush(fpstress);
00028 }
```

References [fpstress](#), [pressure](#), [timeNow](#), [virSumxx](#), [virSumxy](#), and [virSumyy](#).

3.79 PrintStress.c

[Go to the documentation of this file.](#)

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
```

```

00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 void PrintStress(){
00026     fprintf(fpstress, "%.4lf\t%.16lf\t%.16lf\t%.16lf\t%.16lf\t%.16lf\n", timeNow, virSumxx, virSumyy,
00027         virSumxy, pressure);
00027     fflush(fpstress);
00028 }

```

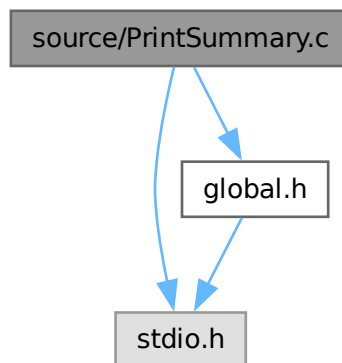
3.80 source/PrintSummary.c File Reference

```

#include <stdio.h>
#include "global.h"

```

Include dependency graph for PrintSummary.c:



Functions

- void `PrintSummary` ()

3.80.1 Function Documentation

3.80.1.1 PrintSummary()

```
void PrintSummary ( )
```

Definition at line 4 of file `PrintSummary.c`.

```

00004     {
00005     fprintf(fpresult, "%.4lf\t%.16lf\t%.16lf\t%.16lf\t%.16lf\t%.16lf\t%.16lf\t%.16lf\n",
00006     timeNow, vSum, potEnergy, kinEnergy, totEnergy, uSumPairPerAtom, BondEnergyPerAtom, pressure,
00007     virSum);
00007     fflush(fpresult);
00008 }

```

References [BondEnergyPerAtom](#), [fpresult](#), [kinEnergy](#), [potEnergy](#), [pressure](#), [timeNow](#), [totEnergy](#), [uSumPairPerAtom](#), [virSum](#), and [vSum](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.81 PrintSummary.c

[Go to the documentation of this file.](#)

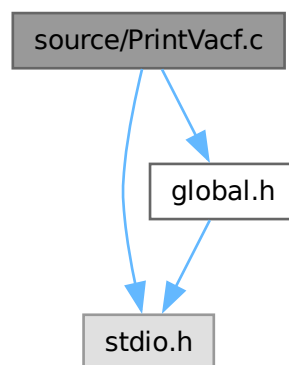
```
00001 #include<stdio.h>
00002 #include"global.h"
00003
00004 void PrintSummary(){
00005     fprintf(fpresult, "%0.4lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\n",
00006         timeNow, vSum, potEnergy, kinEnergy, totEnergy, uSumPairPerAtom, BondEnergyPerAtom, pressure,
00007         virSum);
00007     fflush(fpresult);
00008 }
```

3.82 source/PrintVacf.c File Reference

```
#include <stdio.h>
```

```
#include "global.h"
```

Include dependency graph for PrintVacf.c:



Functions

- void [PrintVacf](#) ()

3.82.1 Function Documentation

3.82.1.1 PrintVacf()

```
void PrintVacf ( )
```

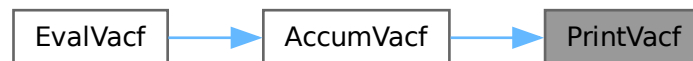
Definition at line 25 of file [PrintVacf.c](#).

```
00025     {
00026     double tVal;
00027     int j;
00028     fprintf(fpvisc,"viscosity acf\n");
00029     for(j = 1 ; j <= nValAcf ; j++){
00030         tVal = (j-1)*stepAcf*deltaT;
00031         fprintf(fpvisc, "%lf\t %lf\t %lf\n", tVal, viscAcfAv[j], viscAcfAv[j]/viscAcfAv[1]);
00032     }
00033     fprintf(fpvisc, "viscosity acf integral : %lf\n", viscAcfInt);
00034 }
```

References [deltaT](#), [fpvisc](#), [nValAcf](#), [stepAcf](#), [viscAcfAv](#), and [viscAcfInt](#).

Referenced by [AccumVacf](#)().

Here is the caller graph for this function:



3.83 PrintVacf.c

[Go to the documentation of this file.](#)

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
```



```

00024
00025 void PrintVacf(){
00026     double tVal;
00027     int j;
00028     fprintf(fpvisc,"viscosity acf\n");
00029     for(j = 1 ; j <= nValAcf ; j ++){
00030         tVal = (j-1)*stepAcf*deltaT;
00031         fprintf(fpvisc, "%lf\t %lf\t %lf\n", tVal, viscAcfAv[j], viscAcfAv[j]/viscAcfAv[1]);
00032     }
00033     fprintf(fpvisc, "viscosity acf integral : %lf\n", viscAcfInt);
00034 }
00035
00036

```

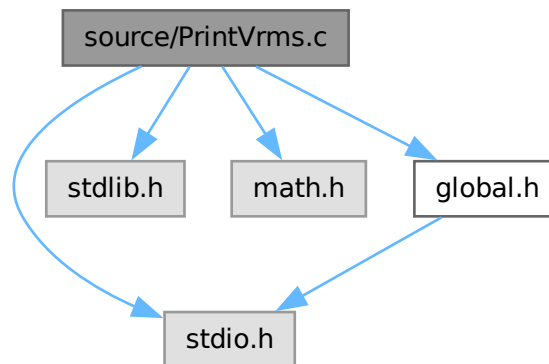
3.84 source/PrintVrms.c File Reference

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "global.h"

```

Include dependency graph for PrintVrms.c:



Functions

- void [PrintVrms](#) ()

3.84.1 Function Documentation

3.84.1.1 PrintVrms()

```
void PrintVrms ( )
```

Definition at line 27 of file [PrintVrms.c](#).

```

00027 {
00028     fprintf(fpvrms, "%0.4lf\t%0.16lf\n", timeNow, VRootMeanSqr);
00029     fflush(fpvrms);
00030 }

```

References [fpvrms](#), [timeNow](#), and [VRootMeanSqr](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.85 PrintVrms.c

[Go to the documentation of this file.](#)

```

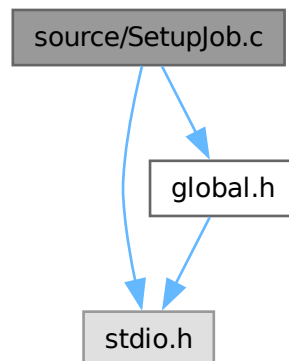
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022 #include<stdio.h>
00023 #include<stdlib.h>
00024 #include<math.h>
00025 #include"global.h"
00026
00027 void PrintVrms(){
00028     fprintf(fpvrms, "%0.4lf\t%0.16lf\n", timeNow, VRootMeanSqr);
00029     fflush(fpvrms);
00030 }
00031
00032
00033
  
```

3.86 source/SetupJob.c File Reference

```

#include <stdio.h>
#include "global.h"
  
```

Include dependency graph for SetupJob.c:



Functions

- void [AllocArrays](#) ()
- void [AccumProps](#) (int icode)
- void [InitVacf](#) ()
- void [SetupJob](#) ()

3.86.1 Function Documentation

3.86.1.1 AccumProps()

```
void AccumProps (
    int icode )
```

Definition at line 25 of file [AccumProps.c](#).

```

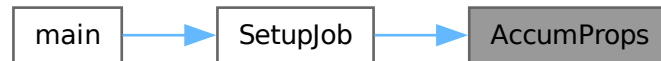
00025     {
00026     if(icode == 0){
00027     sPotEnergy = ssPotEnergy = 0.;
00028     sKinEnergy = ssKinEnergy = 0.;
00029     sPressure = ssPressure = 0.;
00030     sTotEnergy = ssTotEnergy = 0.;
00031     svirSum = 0.;
00032     }else if(icode == 1){
00033     sPotEnergy += potEnergy;
00034     ssPotEnergy += Sqr(potEnergy);
00035     sKinEnergy += kinEnergy;
00036     ssKinEnergy += Sqr(kinEnergy);
00037     sTotEnergy += totEnergy;
00038     ssTotEnergy += Sqr(totEnergy);
00039     sPressure += pressure;
00040     ssPressure += Sqr(pressure);
00041     svirSum += virSum;
00042     }else if(icode == 2){
00043     sPotEnergy /= stepAvg;
00044     ssPotEnergy /= sqrt(ssPotEnergy/stepAvg - Sqr(sPotEnergy));
00045     sTotEnergy /= stepAvg;
00046     ssTotEnergy = sqrt(ssTotEnergy/stepAvg - Sqr(sTotEnergy));
00047     sKinEnergy /= stepAvg;
00048     ssKinEnergy = sqrt(ssKinEnergy/stepAvg - Sqr(sKinEnergy));
00049     sPressure /= stepAvg;
00050     ssPressure = sqrt(ssPressure/stepAvg - Sqr(sPressure));
00051     svirSum /= stepAvg;
```

```
00052 } }
```

References [kinEnergy](#), [potEnergy](#), [pressure](#), [sKinEnergy](#), [sPotEnergy](#), [sPressure](#), [Sqr](#), [ssKinEnergy](#), [ssPotEnergy](#), [ssPressure](#), [ssTotEnergy](#), [stepAvg](#), [sTotEnergy](#), [svirSum](#), [totEnergy](#), and [virSum](#).

Referenced by [SetupJob\(\)](#).

Here is the caller graph for this function:



3.86.1.2 AllocArrays()

```
void AllocArrays ( )
```

Definition at line 25 of file [AllocArrays.c](#).

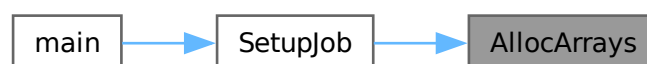
```

00025 {
00026     int n;
00027     // SPACETIME CORRELATIONS
00028     cfOrg = (double **) malloc ((nBuffCorr+1)*sizeof(double *));
00029     for (n = 0; n <= nBuffCorr; n++)
00030         cfOrg[n] = (double *) malloc ((2*nFunCorr+1)*sizeof(double));
00031
00032     cfVal = (double *) malloc ((2*nFunCorr+1)*sizeof(double));
00033     indexCorr = (int *) malloc ((nBuffCorr+1)*sizeof(int));
00034
00035     spacetimeCorr = (double **) malloc ((nBuffCorr+1)*sizeof(double));
00036     for (n = 0; n <= nBuffCorr; n++)
00037         spacetimeCorr[n] = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));
00038
00039     spacetimeCorrAv = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));
00040     // VISCOSITY
00041     indexAcf = (double *) malloc ((nBuffAcf+1)*sizeof(double));
00042     viscAcf = (double **) malloc ((nBuffAcf+1)*sizeof(double *));
00043     for (n = 0; n <= nBuffAcf; n++)
00044         viscAcf[n] = (double *) malloc ((nValAcf+1)*sizeof(double));
00045
00046     viscAcfOrg = (double *) malloc ((nBuffAcf+1)*sizeof(double));
00047     viscAcfAv = (double *) malloc ((nValAcf+1)*sizeof(double));
00048
00049     // RDF
00050     histRdf = (double *) malloc ((sizeHistRdf+1)*sizeof(double));
00051 }
```

References [cfOrg](#), [cfVal](#), [histRdf](#), [indexAcf](#), [indexCorr](#), [nBuffAcf](#), [nBuffCorr](#), [nFunCorr](#), [nValAcf](#), [nValCorr](#), [sizeHistRdf](#), [spacetimeCorr](#), [spacetimeCorrAv](#), [viscAcf](#), [viscAcfAv](#), and [viscAcfOrg](#).

Referenced by [SetupJob\(\)](#).

Here is the caller graph for this function:



3.86.1.3 InitVacf()

```
void InitVacf ( )
```

Definition at line 26 of file [InitVacf.c](#).

```
00026     {
00027     int nb;
00028     for(nb = 1 ; nb <= nBuffAcf ; nb ++)
00029         indexAcf[nb] = -(nb-1)*nValAcf/nBuffAcf;
00030     ZeroVacf();
00031 }
```

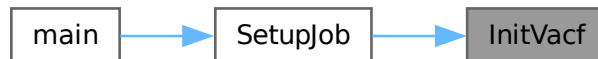
References [indexAcf](#), [nBuffAcf](#), [nValAcf](#), and [ZeroVacf\(\)](#).

Referenced by [SetupJob\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.86.1.4 SetupJob()

```
void SetupJob ( )
```

Definition at line 27 of file [SetupJob.c](#).

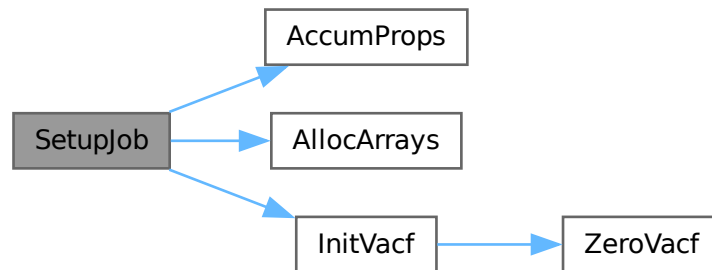
```
00027     {
00028     AllocArrays();
00029     AccumProps(0);
00030     InitVacf();
00031     stepCount = 0;
00032     // INITIALISE SPACETIME CORRELATIONS
00033     int n;
00034     for (n = 1; n <= nBuffCorr; n++)
00035         indexCorr[n] = -(n - 1)*nValCorr/nBuffCorr;
00036
00037     countCorrAv = 0.;
00038
00039     for (n = 1; n <= nFunCorr*nValCorr; n++)
00040         spacetimeCorrAv[n] = 0.;
00041
00042     //RDF
```

```
00043     countRdf = 0;
00044 }
```

References [AccumProps\(\)](#), [AllocArrays\(\)](#), [countCorrAv](#), [countRdf](#), [indexCorr](#), [InitVacf\(\)](#), [nBuffCorr](#), [nFunCorr](#), [nValCorr](#), [spacetimeCorrAv](#), and [stepCount](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.87 SetupJob.c

[Go to the documentation of this file.](#)

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
```

```

00020
00021 #include<stdio.h>
00022 #include"global.h"
00023
00024 void AllocArrays();
00025 void AccumProps(int icode);
00026 void InitVacf();
00027 void SetupJob() {
00028     AllocArrays();
00029     AccumProps(0);
00030     InitVacf();
00031     stepCount = 0;
00032     // INITIALISE SPACETIME CORRELATIONS
00033     int n;
00034     for (n = 1; n <= nBuffCorr; n++)
00035         indexCorr[n] = -(n - 1)*nValCorr/nBuffCorr;
00036
00037     countCorrAv = 0.;
00038
00039     for (n = 1; n <= nFunCorr*nValCorr; n++)
00040         spacetimeCorrAv[n] = 0.;
00041
00042     //RDF
00043     countRdf = 0;
00044 }

```

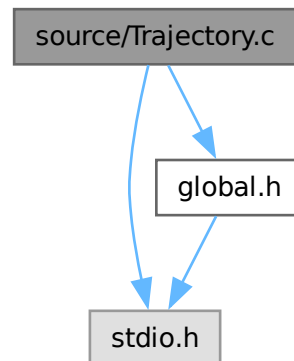
3.88 source/Trajectory.c File Reference

```

#include <stdio.h>
#include "global.h"

```

Include dependency graph for Trajectory.c:



Functions

- void [Trajectory](#) ()

3.88.1 Function Documentation

3.88.1.1 Trajectory()

```
void Trajectory ( )
```

Definition at line 25 of file [Trajectory.c](#).

```
00025     {
00026     int n;
00027     //Trajectory file in LAMMPS dump format for OVITO visualization
00028     fprintf(fpxyz, "ITEM: TIMESTEP\n");
00029     fprintf(fpxyz, "%lf\n", timeNow);
00030     fprintf(fpxyz, "ITEM: NUMBER OF ATOMS\n");
00031     fprintf(fpxyz, "%d\n", nAtom);
00032     fprintf(fpxyz, "ITEM: BOX BOUNDS pp ff pp\n");
00033     fprintf(fpxyz, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
00034     fprintf(fpxyz, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
00035     fprintf(fpxyz, "%lf %lf zlo zhi\n", -0.1, 0.1);
00036     fprintf(fpxyz, "ITEM: ATOMS id mol type radius x y vx vy fx fy\n");
00037     for(n=1; n<=nAtom; n++)
00038         fprintf(fpxyz, "%d\t %d\t %d\t %0.21f\t %0.161f\t %0.161f\t %0.161f\t %0.161f\t %0.161f\t\n",
00039             atomID[n], molID[n], atomType[n], atomRadius[n], rx[n], ry[n], vx[n], vy[n], ax[n], ay[n]);
00040 }
```

References [atomID](#), [atomRadius](#), [atomType](#), [ax](#), [ay](#), [fpxyz](#), [molID](#), [nAtom](#), [regionH](#), [rx](#), [ry](#), [timeNow](#), [vx](#), and [vy](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.89 Trajectory.c

[Go to the documentation of this file.](#)

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 void Trajectory(){
00026     int n;
00027     //Trajectory file in LAMMPS dump format for OVITO visualization
00028     fprintf(fpxyz, "ITEM: TIMESTEP\n");
00029     fprintf(fpxyz, "%lf\n", timeNow);
00030     fprintf(fpxyz, "ITEM: NUMBER OF ATOMS\n");
00031     fprintf(fpxyz, "%d\n", nAtom);
00032     fprintf(fpxyz, "ITEM: BOX BOUNDS pp ff pp\n");
00033     fprintf(fpxyz, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
00034     fprintf(fpxyz, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
00035     fprintf(fpxyz, "%lf %lf zlo zhi\n", -0.1, 0.1);
```



```

00036 fprintf(fpxyz, "ITEM: ATOMS id mol type radius x y vx vy fx fy\n");
00037 for (n=1; n<=nAtom; n++)
00038     fprintf(fpxyz, "%d\t %d\t %d\t %0.21f\t %0.161f\t %0.161f\t %0.161f\t %0.161f\t %0.161f\t\n",
00039         atomID[n], molID[n], atomType[n], atomRadius[n], rx[n], ry[n], vx[n], vy[n], ax[n], ay[n]);
00040 }
00041
00042
00043

```

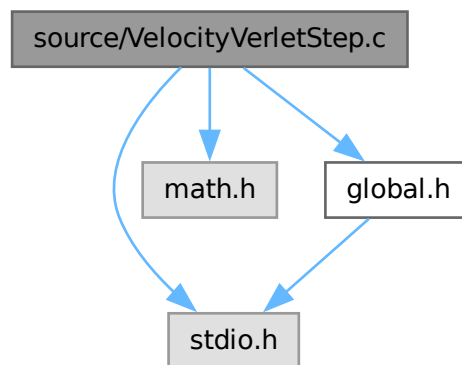
3.90 source/VelocityVerletStep.c File Reference

```

#include <stdio.h>
#include <math.h>
#include "global.h"

```

Include dependency graph for VelocityVerletStep.c:



Functions

- void [VelocityVerletStep](#) (int icode)

3.90.1 Function Documentation

3.90.1.1 VelocityVerletStep()

```

void VelocityVerletStep (
    int icode )

```

Definition at line 26 of file [VelocityVerletStep.c](#).

```

00026 {
00027     int n;
00028     if(icode == 1){
00029         for (n= 1; n <= nAtom; n++) {
00030             if(atomType[n] != freezeAtomType){
00031                 vx[n] += ax[n] * 0.5 * deltaT;
00032                 vy[n] += ay[n] * 0.5 * deltaT;
00033                 rx[n] += vx[n] * deltaT;
00034                 ry[n] += vy[n] * deltaT;

```

```

00035     }
00036     //Calculating the image flags here
00037     if (rx[n] >= regionH[1]) {
00038         rx[n] -= region[1];
00039         ImageX[n]++;
00040     } else if (rx[n] < -regionH[1]) {
00041         rx[n] += region[1];
00042         ImageX[n]--;
00043     }
00044     if (ry[n] >= regionH[2]) {
00045         ry[n] -= region[2];
00046         ImageY[n]++;
00047     } else if (ry[n] < -regionH[2]) {
00048         ry[n] += region[2];
00049         ImageY[n]--;
00050     } }
00051     else if(icode == 2){
00052         for(n = 1; n <= nAtom; n++) {
00053             if(atomType[n] != freezeAtomType){
00054                 vx[n] += ax[n] * 0.5 * deltaT;
00055                 vy[n] += ay[n] * 0.5 * deltaT;
00056             } } } }

```

References [atomType](#), [ax](#), [ay](#), [deltaT](#), [freezeAtomType](#), [ImageX](#), [ImageY](#), [nAtom](#), [region](#), [regionH](#), [rx](#), [ry](#), [vx](#), and [vy](#).

Referenced by [main\(\)](#).

Here is the caller graph for this function:



3.91 VelocityVerletStep.c

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022 #include<stdio.h>
00023 #include<math.h>
00024 #include"global.h"
00025
00026 void VelocityVerletStep(int icode){
00027     int n;
00028     if(icode == 1){
00029         for (n= 1; n <= nAtom; n++) {

```

```

00030     if(atomType[n] != freezeAtomType){
00031         vx[n] += ax[n] * 0.5 * deltaT;
00032         vy[n] += ay[n] * 0.5 * deltaT;
00033         rx[n] += vx[n] * deltaT;
00034         ry[n] += vy[n] * deltaT;
00035     }
00036     //Calculating the image flags here
00037     if (rx[n] >= regionH[1]) {
00038         rx[n] -= region[1];
00039         ImageX[n]++;
00040     } else if (rx[n] < -regionH[1]) {
00041         rx[n] += region[1];
00042         ImageX[n]--;
00043     }
00044     if (ry[n] >= regionH[2]) {
00045         ry[n] -= region[2];
00046         ImageY[n]++;
00047     } else if (ry[n] < -regionH[2]) {
00048         ry[n] += region[2];
00049         ImageY[n]--;
00050     } } }
00051     else if(icode == 2){
00052         for(n = 1; n <= nAtom; n++) {
00053             if(atomType[n] != freezeAtomType){
00054                 vx[n] += ax[n] * 0.5 * deltaT;
00055                 vy[n] += ay[n] * 0.5 * deltaT;
00056             } } }
00057

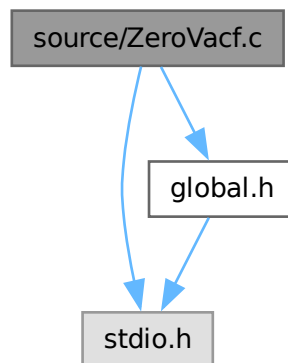
```

3.92 source/ZeroVacf.c File Reference

```
#include <stdio.h>
```

```
#include "global.h"
```

Include dependency graph for ZeroVacf.c:



Functions

- void `ZeroVacf()`

3.92.1 Function Documentation

3.92.1.1 ZeroVacf()

void ZeroVacf ()

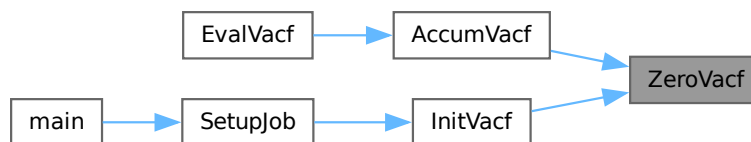
Definition at line 25 of file [ZeroVacf.c](#).

```
00025     {
00026     int j;
00027     countAcfAv= 0 ;
00028     for(j = 1 ; j <= nValAcf ; j ++)
00029         viscAcfAv[j] = 0.;
00030 }
```

References [countAcfAv](#), [nValAcf](#), and [viscAcfAv](#).

Referenced by [AccumVacf\(\)](#), and [InitVacf\(\)](#).

Here is the caller graph for this function:



3.93 ZeroVacf.c

[Go to the documentation of this file.](#)

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017 Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019 */
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 void ZeroVacf(){
00026     int j;
00027     countAcfAv= 0 ;
00028     for(j = 1 ; j <= nValAcf ; j ++)
00029         viscAcfAv[j] = 0.;
00030 }
```

Index

AccumProps
 AccumProps.c, 6
 main.c, 162
 SetupJob.c, 201
AccumProps.c
 AccumProps, 6
AccumVacf
 AccumVacf.c, 8
 EvalVacf.c, 72
AccumVacf.c
 AccumVacf, 8
 Integrate, 9
 PrintVacf, 9
 ZeroVacf, 10
AllocArrays
 AllocArrays.c, 12
 SetupJob.c, 202
AllocArrays.c
 AllocArrays, 12
ApplyBoundaryCond
 ApplyBoundaryCond.c, 14
 main.c, 162
ApplyBoundaryCond.c
 ApplyBoundaryCond, 14
ApplyDrivingForce
 ApplyDrivingForce.c, 16
 main.c, 163
ApplyDrivingForce.c
 ApplyDrivingForce, 16
ApplyForce
 ApplyForce.c, 18
 main.c, 164
ApplyForce.c
 ApplyForce, 18
ApplyLeesEdwardsBoundaryCond
 ApplyLeesEdwardsBoundaryCond.c, 20
 main.c, 164
ApplyLeesEdwardsBoundaryCond.c
 ApplyLeesEdwardsBoundaryCond, 20
ApplyShear
 ApplyShear.c, 22
 main.c, 165
ApplyShear.c
 ApplyShear, 22
ApplyViscous
 ApplyViscous.c, 23
 main.c, 165
ApplyViscous.c
 ApplyViscous, 23

atom1
 global.h, 81
 globalExtern.h, 118
atom2
 global.h, 81
 globalExtern.h, 118
atomID
 global.h, 81
 globalExtern.h, 118
atomIDInterface
 global.h, 81
 globalExtern.h, 118
atomMass
 global.h, 82
 globalExtern.h, 119
atomRadius
 global.h, 82
 globalExtern.h, 119
atomType
 global.h, 82
 globalExtern.h, 119
ax
 global.h, 82
 globalExtern.h, 119
ay
 global.h, 82
 globalExtern.h, 119

bond
 global.h, 82
 globalExtern.h, 119
BondEnergy
 global.h, 83
 globalExtern.h, 119
BondEnergyPerAtom
 global.h, 83
 globalExtern.h, 119
BondID
 global.h, 83
 globalExtern.h, 119
BondLength
 global.h, 83
 globalExtern.h, 120
BondType
 global.h, 83
 globalExtern.h, 120
BrownianStep
 BrownianStep.c, 25
 main.c, 165
BrownianStep.c

- BrownianStep, 25
- cellList
 - global.h, 83
 - globalExtern.h, 120
- cells
 - global.h, 83
 - globalExtern.h, 120
- cfOrg
 - global.h, 84
 - globalExtern.h, 120
- cfVal
 - global.h, 84
 - globalExtern.h, 120
- Close
 - Close.c, 27
 - main.c, 166
- Close.c
 - Close, 27
- com
 - global.h, 84
 - globalExtern.h, 120
- ComputeBondForce
 - ComputeBondForce.c, 30
 - ComputeBondForce.h, 34
- ComputeBondForce.c
 - ComputeBondForce, 30
- ComputeBondForce.h
 - ComputeBondForce, 34
- ComputeForcesCells
 - ComputeForcesCells.c, 37
 - main.c, 167
- ComputeForcesCells.c
 - ComputeForcesCells, 37
- ComputePairForce
 - ComputePairForce.c, 41
 - ComputePairForce.h, 47
- ComputePairForce.c
 - ComputePairForce, 41
- ComputePairForce.h
 - ComputePairForce, 47
- ComX
 - global.h, 84
 - globalExtern.h, 120
- ComX0
 - global.h, 84
 - globalExtern.h, 120
- ComXRatio
 - global.h, 84
 - globalExtern.h, 121
- ComY
 - global.h, 84
 - globalExtern.h, 121
- ComY0
 - global.h, 85
 - globalExtern.h, 121
- ComYRatio
 - global.h, 85
 - globalExtern.h, 121
- countAcfAv
 - global.h, 85
 - globalExtern.h, 121
- countCorrAv
 - global.h, 85
 - globalExtern.h, 121
- countRdf
 - global.h, 85
 - globalExtern.h, 121
- DampFlag
 - global.h, 85
 - globalExtern.h, 121
- DEFINE_GLOBALS
 - main.c, 162
- deltaT
 - global.h, 86
 - globalExtern.h, 122
- DeltaVXij
 - global.h, 86
 - globalExtern.h, 122
- DeltaVYij
 - global.h, 86
 - globalExtern.h, 122
- DeltaX
 - global.h, 86
 - globalExtern.h, 122
- DeltaXij
 - global.h, 86
 - globalExtern.h, 122
- DeltaXijNew
 - global.h, 86
 - globalExtern.h, 122
- DeltaXijOld
 - global.h, 87
 - globalExtern.h, 122
- DeltaXijOldPair
 - global.h, 87
 - globalExtern.h, 122
- DeltaY
 - global.h, 87
 - globalExtern.h, 122
- DeltaYij
 - global.h, 87
 - globalExtern.h, 123
- DeltaYijNew
 - global.h, 87
 - globalExtern.h, 123
- DeltaYijOld
 - global.h, 87
 - globalExtern.h, 123
- DeltaYijOldPair
 - global.h, 87
 - globalExtern.h, 123
- density
 - global.h, 88
 - globalExtern.h, 123
- discDragx
 - global.h, 88

- globalExtern.h, 123
- discDragy
 - global.h, 88
 - globalExtern.h, 123
- DisplaceAtoms
 - DisplaceAtoms.c, 50
 - main.c, 169
- DisplaceAtoms.c
 - DisplaceAtoms, 50
- dnsty
 - global.h, 88
 - globalExtern.h, 123
- dump
 - global.h, 88
 - globalExtern.h, 124
- DumpBonds
 - DumpBonds.c, 52
 - main.c, 169
- DumpBonds.c
 - DumpBonds, 52
- dumpPairFlag
 - global.h, 88
 - globalExtern.h, 124
- DumpPairs
 - DumpPairs.c, 53
 - main.c, 170
- DumpPairs.c
 - DumpPairs, 53
- DumpRestart
 - DumpRestart.c, 55
 - main.c, 171
- DumpRestart.c
 - DumpRestart, 55
- DumpState
 - DumpState.c, 58
 - main.c, 171
- DumpState.c
 - DumpState, 58
- EvalCom
 - EvalCom.c, 60
 - main.c, 172
- EvalCom.c
 - EvalCom, 60
- EvalProps
 - EvalProps.c, 62
 - main.c, 173
- EvalProps.c
 - EvalProps, 62
- EvalRdf
 - EvalRdf.c, 64
 - main.c, 174
- EvalRdf.c
 - EvalRdf, 64
- EvalSpacetimeCorr
 - EvalSpacetimeCorr.c, 66
 - main.c, 174
- EvalSpacetimeCorr.c
 - EvalSpacetimeCorr, 66
- EvalUnwrap
 - EvalUnwrap.c, 70
 - main.c, 176
- EvalUnwrap.c
 - EvalUnwrap, 70
- EvalVacf
 - EvalVacf.c, 72
 - main.c, 176
- EvalVacf.c
 - AccumVacf, 72
 - EvalVacf, 72
- EvalVrms
 - EvalVrms.c, 75
 - main.c, 177
- EvalVrms.c
 - EvalVrms, 75
- EXTERN
 - global.h, 80
- fax
 - global.h, 88
 - globalExtern.h, 124
- fay
 - global.h, 88
 - globalExtern.h, 124
- fpbond
 - global.h, 89
 - globalExtern.h, 124
- fpcom
 - global.h, 89
 - globalExtern.h, 124
- fpdnsty
 - global.h, 89
 - globalExtern.h, 124
- fpdump
 - global.h, 89
 - globalExtern.h, 124
- fpmomentum
 - global.h, 89
 - globalExtern.h, 124
- fppair
 - global.h, 89
 - globalExtern.h, 124
- fprdf
 - global.h, 89
 - globalExtern.h, 125
- fpresult
 - global.h, 89
 - globalExtern.h, 125
- fpstress
 - global.h, 90
 - globalExtern.h, 125
- fpvisc
 - global.h, 90
 - globalExtern.h, 125
- fpvrms
 - global.h, 90
 - globalExtern.h, 125
- fpxyz

- global.h, 90
 - globalExtern.h, 125
- freezeAtomType
 - global.h, 90
 - globalExtern.h, 125
- frfAtom
 - global.h, 90
 - globalExtern.h, 125
- fuSum
 - global.h, 90
 - globalExtern.h, 125
- fvirSum
 - global.h, 90
 - globalExtern.h, 125
- fx
 - global.h, 91
 - globalExtern.h, 126
- fxByfy
 - global.h, 91
 - globalExtern.h, 126
- fy
 - global.h, 91
 - globalExtern.h, 126
- FyBylx
 - global.h, 91
 - globalExtern.h, 126
- gamman
 - global.h, 91
 - globalExtern.h, 126
- global.h
 - atom1, 81
 - atom2, 81
 - atomID, 81
 - atomIDInterface, 81
 - atomMass, 82
 - atomRadius, 82
 - atomType, 82
 - ax, 82
 - ay, 82
 - bond, 82
 - BondEnergy, 83
 - BondEnergyPerAtom, 83
 - BondID, 83
 - BondLength, 83
 - BondType, 83
 - cellList, 83
 - cells, 83
 - cfOrg, 84
 - cfVal, 84
 - com, 84
 - ComX, 84
 - ComX0, 84
 - ComXRatio, 84
 - ComY, 84
 - ComY0, 85
 - ComYRatio, 85
 - countAcfAv, 85
 - countCorrAv, 85
 - countRdf, 85
 - DampFlag, 85
 - deltaT, 86
 - DeltaVXij, 86
 - DeltaVYij, 86
 - DeltaX, 86
 - DeltaXij, 86
 - DeltaXijNew, 86
 - DeltaXijOld, 87
 - DeltaXijOldPair, 87
 - DeltaY, 87
 - DeltaYij, 87
 - DeltaYijNew, 87
 - DeltaYijOld, 87
 - DeltaYijOldPair, 87
 - density, 88
 - discDragx, 88
 - discDragy, 88
 - dnsty, 88
 - dump, 88
 - dumpPairFlag, 88
 - EXTERN, 80
 - fax, 88
 - fay, 88
 - fpbond, 89
 - fpcom, 89
 - fpdnsty, 89
 - fpdump, 89
 - fpmomentum, 89
 - fppair, 89
 - fprdf, 89
 - fpresult, 89
 - fpstress, 90
 - fpvisc, 90
 - fpvrms, 90
 - fpxyz, 90
 - freezeAtomType, 90
 - frfAtom, 90
 - fuSum, 90
 - fvirSum, 90
 - fx, 91
 - fxByfy, 91
 - fy, 91
 - FyBylx, 91
 - gamman, 91
 - HaltCondition, 91
 - histRdf, 91
 - ImageX, 92
 - ImageY, 92
 - indexAcf, 92
 - indexCorr, 92
 - initUcell, 92
 - isBonded, 92
 - kappa, 92
 - kb, 93
 - kinEnergy, 93
 - Kn, 93
 - limitAcfAv, 93

limitCorrAv, 93
limitRdf, 93
master, 94
molID, 94
momentum, 94
moreCycles, 94
nAtom, 94
nAtomBlock, 94
nAtomInterface, 95
nAtomType, 95
nBond, 95
nBondType, 95
nBuffAcf, 95
nBuffCorr, 95
NDIM, 80
nDiscInterface, 95
nFunCorr, 96
nodeDragx, 96
nodeDragy, 96
nPairActive, 96
nPairTotal, 96
nValAcf, 96
nValCorr, 97
pair, 97
Pairatom1, 97
Pairatom2, 97
PairID, 97
PairXij, 97
PairYij, 97
potEnergy, 98
prefix, 98
pressure, 98
RadiusIJ, 98
RadiusIJInv, 98
rangeRdf, 98
rank, 99
rCut, 99
rdf, 99
real, 81
region, 99
regionH, 99
result, 99
rfAtom, 99
ro, 100
rx, 100
rxUnwrap, 100
ry, 100
ryUnwrap, 100
shearDisplacement, 100
shearVelocity, 101
SignR, 80
size, 101
sizeHistRdf, 101
sKinEnergy, 101
solver, 101
spacetimeCorr, 101
spacetimeCorrAv, 102
speed, 102
sPotEnergy, 102
sPressure, 102
Sqr, 81
SqrRadiusIJ, 102
ssKinEnergy, 102
ssPotEnergy, 103
ssPressure, 103
ssTotEnergy, 103
stepAcf, 103
stepAvg, 103
stepCorr, 103
stepCount, 104
stepDump, 104
stepEquil, 104
stepLimit, 104
stepRdf, 104
stepTraj, 104
sTotEnergy, 105
strain, 105
strainRate, 105
strech, 105
stress, 105
svirSum, 105
thermo, 105
timeNow, 106
TotalBondEnergy, 106
TotalMass, 106
totEnergy, 106
uSum, 106
uSumPair, 106
uSumPairPerAtom, 107
virSum, 107
virSumBond, 107
virSumBondxx, 107
virSumBondxy, 107
virSumBondyy, 107
virSumPair, 108
virSumPairxx, 108
virSumPairxy, 108
virSumPairyy, 108
virSumxx, 108
virSumxy, 108
virSumyy, 109
visc, 109
viscAcf, 109
viscAcfAv, 109
viscAcfInt, 109
viscAcfOrg, 109
VMeanSqr, 110
vrms, 110
VRootMeanSqr, 110
VSqr, 110
vSum, 110
vSumX, 110
vSumY, 111
vvSum, 111
vx, 111
vy, 111

- xBoundary, 111
 - xyz, 111
 - yBoundary, 112
- globalExtern.h
 - atom1, 118
 - atom2, 118
 - atomID, 118
 - atomIDInterface, 118
 - atomMass, 119
 - atomRadius, 119
 - atomType, 119
 - ax, 119
 - ay, 119
 - bond, 119
 - BondEnergy, 119
 - BondEnergyPerAtom, 119
 - BondID, 119
 - BondLength, 120
 - BondType, 120
 - cellList, 120
 - cells, 120
 - cfOrg, 120
 - cfVal, 120
 - com, 120
 - ComX, 120
 - ComX0, 120
 - ComXRatio, 121
 - ComY, 121
 - ComY0, 121
 - ComYRatio, 121
 - countAcfAv, 121
 - countCorrAv, 121
 - countRdf, 121
 - DampFlag, 121
 - deltaT, 122
 - DeltaVXij, 122
 - DeltaVYij, 122
 - DeltaX, 122
 - DeltaXij, 122
 - DeltaXijNew, 122
 - DeltaXijOld, 122
 - DeltaXijOldPair, 122
 - DeltaY, 122
 - DeltaYij, 123
 - DeltaYijNew, 123
 - DeltaYijOld, 123
 - DeltaYijOldPair, 123
 - density, 123
 - discDragx, 123
 - discDragy, 123
 - dnsty, 123
 - dump, 124
 - dumpPairFlag, 124
 - fax, 124
 - fay, 124
 - fpbond, 124
 - fpcom, 124
 - fpdnsty, 124
 - fpdump, 124
 - fpmomentum, 124
 - fppair, 124
 - fprdf, 125
 - fpresult, 125
 - fpstress, 125
 - fpvisc, 125
 - fpvrms, 125
 - fpxyz, 125
 - freezeAtomType, 125
 - frfAtom, 125
 - fuSum, 125
 - fvirSum, 125
 - fx, 126
 - fxByfy, 126
 - fy, 126
 - FyBylx, 126
 - gamman, 126
 - HaltCondition, 126
 - histRdf, 126
 - ImageX, 126
 - ImageY, 126
 - indexAcf, 127
 - indexCorr, 127
 - initUcell, 127
 - isBonded, 127
 - kappa, 127
 - kb, 127
 - kinEnergy, 127
 - Kn, 127
 - limitAcfAv, 127
 - limitCorrAv, 128
 - limitRdf, 128
 - master, 128
 - molID, 128
 - momentum, 128
 - moreCycles, 128
 - nAtom, 128
 - nAtomBlock, 128
 - nAtomInterface, 129
 - nAtomType, 129
 - nBond, 129
 - nBondType, 129
 - nBuffAcf, 129
 - nBuffCorr, 129
 - NDIM, 118
 - nDiscInterface, 129
 - nFunCorr, 129
 - nodeDragx, 130
 - nodeDragy, 130
 - nPairActive, 130
 - nPairTotal, 130
 - nValAcf, 130
 - nValCorr, 130
 - pair, 130
 - Pairatom1, 130
 - Pairatom2, 131
 - PairID, 131

- PairXij, [131](#)
- PairYij, [131](#)
- potEnergy, [131](#)
- prefix, [131](#)
- pressure, [131](#)
- RadiusIJ, [131](#)
- RadiusIInv, [132](#)
- rangeRdf, [132](#)
- rank, [132](#)
- rCut, [132](#)
- rdf, [132](#)
- real, [118](#)
- region, [132](#)
- regionH, [132](#)
- result, [132](#)
- rfAtom, [132](#)
- ro, [133](#)
- rx, [133](#)
- rxUnwrap, [133](#)
- ry, [133](#)
- ryUnwrap, [133](#)
- shearDisplacement, [133](#)
- shearVelocity, [133](#)
- SignR, [118](#)
- size, [133](#)
- sizeHistRdf, [133](#)
- sKinEnergy, [134](#)
- solver, [134](#)
- spacetimeCorr, [134](#)
- spacetimeCorrAv, [134](#)
- speed, [134](#)
- sPotEnergy, [134](#)
- sPressure, [134](#)
- Sqr, [118](#)
- SqrRadiusIJ, [134](#)
- ssKinEnergy, [135](#)
- ssPotEnergy, [135](#)
- ssPressure, [135](#)
- ssTotEnergy, [135](#)
- stepAcf, [135](#)
- stepAvg, [135](#)
- stepCorr, [135](#)
- stepCount, [135](#)
- stepDump, [136](#)
- stepEquil, [136](#)
- stepLimit, [136](#)
- stepRdf, [136](#)
- stepTraj, [136](#)
- sTotEnergy, [136](#)
- strain, [136](#)
- strainRate, [136](#)
- strech, [137](#)
- stress, [137](#)
- svirSum, [137](#)
- timeNow, [137](#)
- TotalBondEnergy, [137](#)
- TotalMass, [137](#)
- totEnergy, [137](#)
- uSum, [137](#)
- uSumPair, [138](#)
- uSumPairPerAtom, [138](#)
- virSum, [138](#)
- virSumBond, [138](#)
- virSumBondxx, [138](#)
- virSumBondxy, [138](#)
- virSumBondyy, [138](#)
- virSumPair, [138](#)
- virSumPairxx, [139](#)
- virSumPairxy, [139](#)
- virSumPairyy, [139](#)
- virSumxx, [139](#)
- virSumxy, [139](#)
- virSumyy, [139](#)
- visc, [139](#)
- viscAcf, [139](#)
- viscAcfAv, [140](#)
- viscAcfInt, [140](#)
- viscAcfOrg, [140](#)
- VMeanSqr, [140](#)
- vrms, [140](#)
- VRootMeanSqr, [140](#)
- VSqr, [140](#)
- vSum, [140](#)
- vSumX, [141](#)
- vSumY, [141](#)
- vvSum, [141](#)
- vx, [141](#)
- vy, [141](#)
- xBoundary, [141](#)
- xyz, [141](#)
- yBoundary, [141](#)
- Halt.c
 - HaltConditionCheck, [144](#)
- HaltCondition
 - global.h, [91](#)
 - globalExtern.h, [126](#)
- HaltConditionCheck
 - Halt.c, [144](#)
 - main.c, [177](#)
- histRdf
 - global.h, [91](#)
 - globalExtern.h, [126](#)
- ImageX
 - global.h, [92](#)
 - globalExtern.h, [126](#)
- ImageY
 - global.h, [92](#)
 - globalExtern.h, [126](#)
- indexAcf
 - global.h, [92](#)
 - globalExtern.h, [127](#)
- indexCorr
 - global.h, [92](#)
 - globalExtern.h, [127](#)
- Init

- Init.c, 146
- main.c, 178
- Init.c
 - Init, 146
- initUcell
 - global.h, 92
 - globalExtern.h, 127
- InitVacf
 - InitVacf.c, 153
 - SetupJob.c, 202
- InitVacf.c
 - InitVacf, 153
 - ZeroVacf, 153
- Integrate
 - AccumVacf.c, 9
 - Integrate.c, 155
- Integrate.c
 - Integrate, 155
- isBonded
 - global.h, 92
 - globalExtern.h, 127
- kappa
 - global.h, 92
 - globalExtern.h, 127
- kb
 - global.h, 93
 - globalExtern.h, 127
- kinEnergy
 - global.h, 93
 - globalExtern.h, 127
- Kn
 - global.h, 93
 - globalExtern.h, 127
- LeapfrogStep
 - LeapfrogStep.c, 157
 - main.c, 181
- LeapfrogStep.c
 - LeapfrogStep, 157
- Lemina, 1
- limitAcfAv
 - global.h, 93
 - globalExtern.h, 127
- limitCorrAv
 - global.h, 93
 - globalExtern.h, 128
- limitRdf
 - global.h, 93
 - globalExtern.h, 128
- main
 - main.c, 181
- main.c
 - AccumProps, 162
 - ApplyBoundaryCond, 162
 - ApplyDrivingForce, 163
 - ApplyForce, 164
 - ApplyLeesEdwardsBoundaryCond, 164
 - ApplyShear, 165
 - ApplyViscous, 165
 - BrownianStep, 165
 - Close, 166
 - ComputeForcesCells, 167
 - DEFINE_GLOBALS, 162
 - DisplaceAtoms, 169
 - DumpBonds, 169
 - DumpPairs, 170
 - DumpRestart, 171
 - DumpState, 171
 - EvalCom, 172
 - EvalProps, 173
 - EvalRdf, 174
 - EvalSpacetimeCorr, 174
 - EvalUnwrap, 176
 - EvalVacf, 176
 - EvalVrms, 177
 - HaltConditionCheck, 177
 - Init, 178
 - LeapfrogStep, 181
 - main, 181
 - prefix, 189
 - PrintCom, 184
 - PrintMomentum, 185
 - PrintStress, 185
 - PrintSummary, 185
 - PrintVrms, 186
 - SetupJob, 186
 - Trajectory, 187
 - VelocityVerletStep, 188
- master
 - global.h, 94
 - globalExtern.h, 128
- molID
 - global.h, 94
 - globalExtern.h, 128
- momentum
 - global.h, 94
 - globalExtern.h, 128
- moreCycles
 - global.h, 94
 - globalExtern.h, 128
- nAtom
 - global.h, 94
 - globalExtern.h, 128
- nAtomBlock
 - global.h, 94
 - globalExtern.h, 128
- nAtomInterface
 - global.h, 95
 - globalExtern.h, 129
- nAtomType
 - global.h, 95
 - globalExtern.h, 129
- nBond
 - global.h, 95
 - globalExtern.h, 129

- nBondType
 - global.h, 95
 - globalExtern.h, 129
- nBuffAcf
 - global.h, 95
 - globalExtern.h, 129
- nBuffCorr
 - global.h, 95
 - globalExtern.h, 129
- NDIM
 - global.h, 80
 - globalExtern.h, 118
- nDisclInterface
 - global.h, 95
 - globalExtern.h, 129
- nFunCorr
 - global.h, 96
 - globalExtern.h, 129
- nodeDragx
 - global.h, 96
 - globalExtern.h, 130
- nodeDragy
 - global.h, 96
 - globalExtern.h, 130
- nPairActive
 - global.h, 96
 - globalExtern.h, 130
- nPairTotal
 - global.h, 96
 - globalExtern.h, 130
- nValAcf
 - global.h, 96
 - globalExtern.h, 130
- nValCorr
 - global.h, 97
 - globalExtern.h, 130
- pair
 - global.h, 97
 - globalExtern.h, 130
- Pairatom1
 - global.h, 97
 - globalExtern.h, 130
- Pairatom2
 - global.h, 97
 - globalExtern.h, 131
- PairID
 - global.h, 97
 - globalExtern.h, 131
- PairXij
 - global.h, 97
 - globalExtern.h, 131
- PairYij
 - global.h, 97
 - globalExtern.h, 131
- potEnergy
 - global.h, 98
 - globalExtern.h, 131
- prefix
 - global.h, 98
 - globalExtern.h, 131
 - main.c, 189
- pressure
 - global.h, 98
 - globalExtern.h, 131
- PrintCom
 - main.c, 184
 - PrintCom.c, 192
- PrintCom.c
 - PrintCom, 192
- PrintMomentum
 - main.c, 185
 - PrintMomentum.c, 194
- PrintMomentum.c
 - PrintMomentum, 194
- PrintStress
 - main.c, 185
 - PrintStress.c, 195
- PrintStress.c
 - PrintStress, 195
- PrintSummary
 - main.c, 185
 - PrintSummary.c, 196
- PrintSummary.c
 - PrintSummary, 196
- PrintVacf
 - AccumVacf.c, 9
 - PrintVacf.c, 198
- PrintVacf.c
 - PrintVacf, 198
- PrintVrms
 - main.c, 186
 - PrintVrms.c, 199
- PrintVrms.c
 - PrintVrms, 199
- RadiusIJ
 - global.h, 98
 - globalExtern.h, 131
- RadiusIJInv
 - global.h, 98
 - globalExtern.h, 132
- rangeRdf
 - global.h, 98
 - globalExtern.h, 132
- rank
 - global.h, 99
 - globalExtern.h, 132
- rCut
 - global.h, 99
 - globalExtern.h, 132
- rdf
 - global.h, 99
 - globalExtern.h, 132
- README.md, 5
- real
 - global.h, 81
 - globalExtern.h, 118

- region
 - global.h, 99
 - globalExtern.h, 132
- regionH
 - global.h, 99
 - globalExtern.h, 132
- result
 - global.h, 99
 - globalExtern.h, 132
- rfAtom
 - global.h, 99
 - globalExtern.h, 132
- ro
 - global.h, 100
 - globalExtern.h, 133
- rx
 - global.h, 100
 - globalExtern.h, 133
- rxUnwrap
 - global.h, 100
 - globalExtern.h, 133
- ry
 - global.h, 100
 - globalExtern.h, 133
- ryUnwrap
 - global.h, 100
 - globalExtern.h, 133
- SetupJob
 - main.c, 186
 - SetupJob.c, 203
- SetupJob.c
 - AccumProps, 201
 - AllocArrays, 202
 - InitVacf, 202
 - SetupJob, 203
- shearDisplacement
 - global.h, 100
 - globalExtern.h, 133
- shearVelocity
 - global.h, 101
 - globalExtern.h, 133
- SignR
 - global.h, 80
 - globalExtern.h, 118
- size
 - global.h, 101
 - globalExtern.h, 133
- sizeHistRdf
 - global.h, 101
 - globalExtern.h, 133
- sKinEnergy
 - global.h, 101
 - globalExtern.h, 134
- solver
 - global.h, 101
 - globalExtern.h, 134
- source/AccumProps.c, 5, 6
- source/AccumVacf.c, 7, 10
- source/AllocArrays.c, 11, 12
- source/ApplyBoundaryCond.c, 13, 15
- source/ApplyDrivingForce.c, 15, 17
- source/ApplyForce.c, 18, 19
- source/ApplyLeesEdwardsBoundaryCond.c, 19, 21
- source/ApplyShear.c, 21, 22
- source/ApplyViscous.c, 23, 24
- source/BrownianStep.c, 24, 25
- source/Close.c, 26, 28
- source/ComputeBondForce.c, 29, 32
- source/ComputeBondForce.h, 34, 36
- source/ComputeForcesCells.c, 36, 39
- source/ComputePairForce.c, 41, 44
- source/ComputePairForce.h, 46, 49
- source/DisplaceAtoms.c, 49, 51
- source/DumpBonds.c, 51, 52
- source/DumpPairs.c, 53, 54
- source/DumpRestart.c, 55, 56
- source/DumpState.c, 57, 58
- source/EvalCom.c, 59, 60
- source/EvalProps.c, 61, 63
- source/EvalRdf.c, 63, 65
- source/EvalSpacetimeCorr.c, 66, 68
- source/EvalUnwrap.c, 69, 71
- source/EvalVacf.c, 71, 73
- source/EvalVrms.c, 74, 75
- source/global.h, 76, 112
- source/globalExtern.h, 114, 142
- source/Halt.c, 143, 145
- source/Init.c, 145, 149
- source/InitVacf.c, 152, 154
- source/Integrate.c, 155, 156
- source/LeapfrogStep.c, 156, 158
- source/main.c, 160, 189
- source/PrintCom.c, 191, 193
- source/PrintMomentum.c, 193, 194
- source/PrintStress.c, 194, 195
- source/PrintSummary.c, 196, 197
- source/PrintVacf.c, 197, 198
- source/PrintVrms.c, 199, 200
- source/SetupJob.c, 200, 204
- source/Trajectory.c, 205, 206
- source/VelocityVerletStep.c, 207, 208
- source/ZeroVacf.c, 209, 210
- spacetimeCorr
 - global.h, 101
 - globalExtern.h, 134
- spacetimeCorrAv
 - global.h, 102
 - globalExtern.h, 134
- speed
 - global.h, 102
 - globalExtern.h, 134
- sPotEnergy
 - global.h, 102
 - globalExtern.h, 134
- sPressure
 - global.h, 102

- globalExtern.h, 134
- Sqr
 - global.h, 81
 - globalExtern.h, 118
- SqrRadiusIJ
 - global.h, 102
 - globalExtern.h, 134
- ssKinEnergy
 - global.h, 102
 - globalExtern.h, 135
- ssPotEnergy
 - global.h, 103
 - globalExtern.h, 135
- ssPressure
 - global.h, 103
 - globalExtern.h, 135
- ssTotEnergy
 - global.h, 103
 - globalExtern.h, 135
- stepAcf
 - global.h, 103
 - globalExtern.h, 135
- stepAvg
 - global.h, 103
 - globalExtern.h, 135
- stepCorr
 - global.h, 103
 - globalExtern.h, 135
- stepCount
 - global.h, 104
 - globalExtern.h, 135
- stepDump
 - global.h, 104
 - globalExtern.h, 136
- stepEquil
 - global.h, 104
 - globalExtern.h, 136
- stepLimit
 - global.h, 104
 - globalExtern.h, 136
- stepRdf
 - global.h, 104
 - globalExtern.h, 136
- stepTraj
 - global.h, 104
 - globalExtern.h, 136
- sTotEnergy
 - global.h, 105
 - globalExtern.h, 136
- strain
 - global.h, 105
 - globalExtern.h, 136
- strainRate
 - global.h, 105
 - globalExtern.h, 136
- strech
 - global.h, 105
 - globalExtern.h, 137
- stress
 - global.h, 105
 - globalExtern.h, 137
- svirSum
 - global.h, 105
 - globalExtern.h, 137
- thermo
 - global.h, 105
- timeNow
 - global.h, 106
 - globalExtern.h, 137
- TotalBondEnergy
 - global.h, 106
 - globalExtern.h, 137
- TotalMass
 - global.h, 106
 - globalExtern.h, 137
- totEnergy
 - global.h, 106
 - globalExtern.h, 137
- Trajectory
 - main.c, 187
 - Trajectory.c, 205
- Trajectory.c
 - Trajectory, 205
- uSum
 - global.h, 106
 - globalExtern.h, 137
- uSumPair
 - global.h, 106
 - globalExtern.h, 138
- uSumPairPerAtom
 - global.h, 107
 - globalExtern.h, 138
- VelocityVerletStep
 - main.c, 188
 - VelocityVerletStep.c, 207
- VelocityVerletStep.c
 - VelocityVerletStep, 207
- virSum
 - global.h, 107
 - globalExtern.h, 138
- virSumBond
 - global.h, 107
 - globalExtern.h, 138
- virSumBondxx
 - global.h, 107
 - globalExtern.h, 138
- virSumBondxy
 - global.h, 107
 - globalExtern.h, 138
- virSumBondyy
 - global.h, 107
 - globalExtern.h, 138
- virSumPair
 - global.h, 108

- globalExtern.h, 138
- virSumPairxx
 - global.h, 108
 - globalExtern.h, 139
- virSumPairxy
 - global.h, 108
 - globalExtern.h, 139
- virSumPairyy
 - global.h, 108
 - globalExtern.h, 139
- virSumxx
 - global.h, 108
 - globalExtern.h, 139
- virSumxy
 - global.h, 108
 - globalExtern.h, 139
- virSumyy
 - global.h, 109
 - globalExtern.h, 139
- visc
 - global.h, 109
 - globalExtern.h, 139
- viscAcf
 - global.h, 109
 - globalExtern.h, 139
- viscAcfAv
 - global.h, 109
 - globalExtern.h, 140
- viscAcfInt
 - global.h, 109
 - globalExtern.h, 140
- viscAcfOrg
 - global.h, 109
 - globalExtern.h, 140
- VMeanSqr
 - global.h, 110
 - globalExtern.h, 140
- vrms
 - global.h, 110
 - globalExtern.h, 140
- VRootMeanSqr
 - global.h, 110
 - globalExtern.h, 140
- VSqr
 - global.h, 110
 - globalExtern.h, 140
- vSum
 - global.h, 110
 - globalExtern.h, 140
- vSumX
 - global.h, 110
 - globalExtern.h, 141
- vSumY
 - global.h, 111
 - globalExtern.h, 141
- vvSum
 - global.h, 111
 - globalExtern.h, 141
- vx
 - global.h, 111
 - globalExtern.h, 141
- vy
 - global.h, 111
 - globalExtern.h, 141
- xBoundary
 - global.h, 111
 - globalExtern.h, 141
- xyz
 - global.h, 111
 - globalExtern.h, 141
- yBoundary
 - global.h, 112
 - globalExtern.h, 141
- ZeroVacf
 - AccumVacf.c, 10
 - InitVacf.c, 153
 - ZeroVacf.c, 210
- ZeroVacf.c
 - ZeroVacf, 210