Lemina

Generated by Doxygen 1.10.0

Chapter 1

Lamina: A Molecular Dynamics Package

Welcome to the Lamina documentation!

1.1 # Lamina

Lamina is a modular 2D molecular dynamics simulation package designed for simulating hybrid soft solids, including spring networks and finite-size discs. **Lamina** is a modular and extensible molecular dynamics (MD) simulation package written in C, designed to model a wide variety of soft and condensed matter systems. It supports time evolution using robust integrators and a range of thermostats, with accurate force evaluations for bonded and non-bonded interactions. Originally built for 2D simulations of bonded systems, **Lamina** has grown to support broader research goals including active matter, granular solids, and complex fluids.

1.2 Why "Lamina"?

The word **Lamina** comes from Latin, meaning "a thin layer", "a plate", or "a sheet". In nature and science, laminae often refer to structural elements that are flat and extended in two dimensions for example, leaves, thin metal sheets, or tissue membranes.

This name reflects both the **two-dimensional** (2D) nature of the simulations and the types of materials **Lamina** is built to study; **liquids**, **soft solids**, and **networked structures** confined to thin sheets or layers. Just as natural laminae exhibit rich structural and dynamic behaviors in a seemingly simple geometry, this code is designed to explore the complexity of emergent phenomena in 2D materials and soft matter systems.

1.3 Key Features

1.4 Interaction Potentials

Yukawa potential (screened Coulomb interactions), Lennard-Jones potential (standard 12-6), Harmonic bond potential (elastic network models), Hookean granular contact potential (for soft granular matter).

1.5 Thermostats and Temperature Control

Gaussian thermostat, Nose-Hoover thermostat, Langevin thermostat, Configurational temperature evaluation and control.

1.6 Time Integration

Velocity-Verlet integrator, Brownian (overdamped) dynamics,

1.7 Physical Observables

-Radial Distribution Function (RDF) -Velocity Autocorrelation Function (VACF) -Root-Mean-Square Velocity (VRMS) -Stress tensor and momentum -Center-of-mass motion -Space-time correlation functions

1.8 Output and Utilities

The output files are saved at the ../output folder. So you have a make a directory ../ location from where you ae running ./main prefix -Structured output files (`.xyz`, `.bond`, `.pair`, `.com`, `.result`) -Restart and resume capability (`.restart` and `.state` files) -Clean separation of source code, unit tests, and output -Support for Lees–Edwards boundary conditions (sheared systems) -Configurable halting conditions (based on VRMS or custom metric) -Modular design for easy extension of potentials and features

1.9 Project Structure

1.9.1 Project Structure

```
Lamina/
  |- source/                 # All C source files; avoid placing README.md here to prevent extra re
      |- main.c              # Main driver
      |- *.c, *.h            # Modular source files
  |- unittest/               # Unit test suite (planned or implemented)
      |- test_*.c            # Individual test cases
  |- output/                 # All runtime output files will be saved here
  |- prepros/                # Preprocessing scripts or tools
      |- *.sh, *.py          # Example preprocessing scripts (shell, python, etc.)
  |- postpros/               # Postprocessing scripts or tools
      |- *.sh, *.py          # Example postprocessing scripts (shell, python, etc.)
  |- doxygen/                # Doxygen configuration and auxiliary files
      |- Doxyfile            # Doxygen config file
      |- header.tex          # Custom LaTeX header for documentation
      |- extra_stylesheet.css # Optional CSS to customize HTML output (e.g., hide Related Pages ta
  |- figures/                # All figures, logos, icons used in docs and code
      |- LogoLaminaHTML.png # Project logo used in HTML and LaTeX docs
  |- docs/                   # Generated documentation output (HTML, LaTeX, PDFs)
      |- html/               # Doxygen-generated HTML documentation
      |- latex/              # Doxygen-generated LaTeX source files
      |- refman.pdf          # Generated PDF documentation (from LaTeX)
  |- Makefile                # Build system
  |- README.md               # Main project documentation (used as main page in Doxygen)
```

##Documentation

- `Browse full HTML documentation`
- Download `Source code PDF manual`
- User manual `Physics PDF manual` This documentation was generated using `Doxygen 1.10.0` to ensure transparency and ease of review.

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

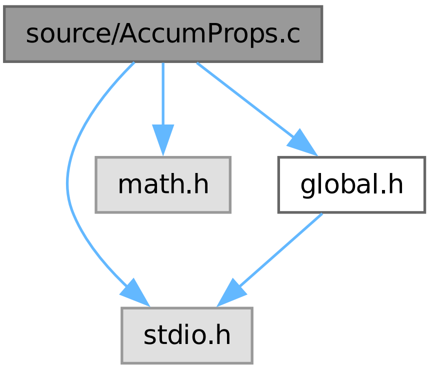
File Documentation

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 )
```

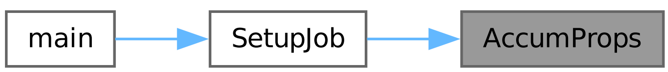
Definition at line 25 of file AccumProps.c.

```
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.

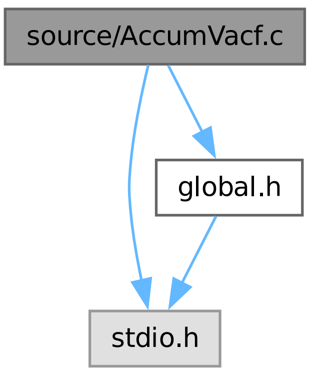
```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina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

```
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */

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

void AccumProps(int icode){
 if(icode == 0){
 sPotEnergy = ssPotEnergy = 0.;
 sKinEnergy = ssKinEnergy = 0.;
 sPressure = ssPressure = 0.;
 sTotEnergy = ssTotEnergy = 0.;
 svirSum = 0.;
 }else if(icode == 1){
 sPotEnergy += potEnergy;
 ssPotEnergy += Sqr(potEnergy);
 sKinEnergy += kinEnergy;
 ssKinEnergy += Sqr(kinEnergy);
 sTotEnergy += totEnergy;
 ssTotEnergy += Sqr(totEnergy);
 sPressure += pressure;
 ssPressure += Sqr(pressure);
 svirSum += virSum;
 }else if(icode == 2){
 sPotEnergy /= stepAvg;
 ssPotEnergy /= sqrt(ssPotEnergy/stepAvg - Sqr(sPotEnergy));
 sTotEnergy /= stepAvg;
 ssTotEnergy = sqrt(ssTotEnergy/stepAvg - Sqr(sTotEnergy));
 sKinEnergy /= stepAvg;
 ssKinEnergy = sqrt(ssKinEnergy/stepAvg - Sqr(sKinEnergy));
 sPressure /= stepAvg;
 ssPressure = sqrt(ssPressure/stepAvg - Sqr(sPressure));
 svirSum /= stepAvg;
 } }
```

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 ( )
```

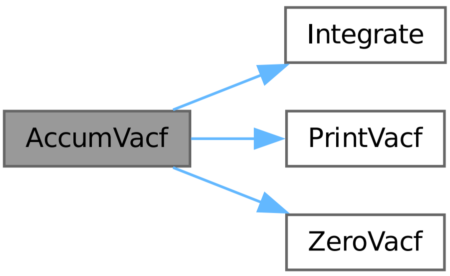
Definition at line 27 of file AccumVacf.c.

```
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 } } } }
```

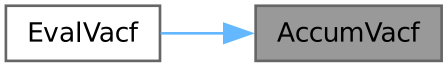
References countAcfAv, deltaT, indexAcf, Integrate(), kinEnergy, limitAcfAv, nBuffAcf, nValAcf, PrintVacf(), region, stepAcf, viscAcf, viscAcfAv, viscAcfInt, and ZeroVacf().

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.

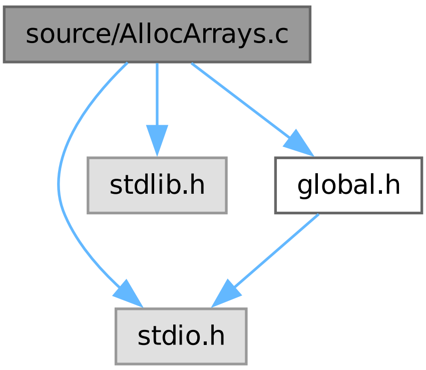
```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina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

```
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */

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

double Integrate(double *, int);
 void PrintVacf();
 void ZeroVacf();
 void AccumVacf(){
 double fac;
 int j, nb;
 for(nb = 1 ; nb <= nBuffAcf ; nb ++){
  if(indexAcf[nb] == nValAcf){
   for(j = 1 ; j <= nValAcf; j ++){
    viscAcfAv[j] +=  viscAcf[nb][j];
    }
  indexAcf[nb] = 0;
  countAcfAv ++;
  if(countAcfAv == limitAcfAv){
   fac = 1./(kinEnergy*region[1]*region[2]*limitAcfAv);
   viscAcfInt = fac*stepAcf*deltaT*Integrate(viscAcfAv, nValAcf);
   PrintVacf();
   ZeroVacf();
} } } }
```

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.

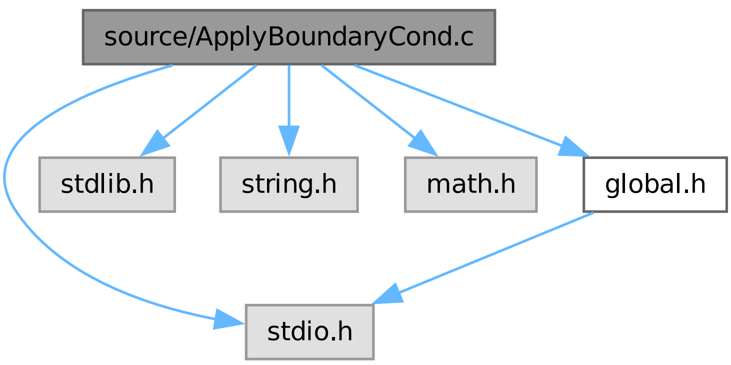
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your 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  *
```

```
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */

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

void AllocArrays(){
 int n;
 // SPACETIME CORRELATIONS
 cfOrg = (double **) malloc ((nBuffCorr+1)*sizeof(double *));
 for (n = 0; n <= nBuffCorr; n++)
  cfOrg[n] = (double *) malloc ((2*nFunCorr+1)*sizeof(double));

 cfVal = (double *) malloc ((2*nFunCorr+1)*sizeof(double));
 indexCorr = (int *) malloc ((nBuffCorr+1)*sizeof(int));

 spacetimeCorr = (double **) malloc ((nBuffCorr+1)*sizeof(double));
 for (n = 0; n <= nBuffCorr; n++)
 spacetimeCorr[n] = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));

 spacetimeCorrAv = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));
 // VISCOSITY
 indexAcf = (double *)malloc((nBuffAcf+1)*sizeof(double));
 viscAcf = (double **)malloc((nBuffAcf+1)*sizeof(double *));
 for(n = 0 ; n <= nBuffAcf ; n ++)
  viscAcf[n] = (double *)malloc((nValAcf+1)*sizeof(double ));

 viscAcfOrg = (double *)malloc((nBuffAcf+1)*sizeof(double));
 viscAcfAv = (double *)malloc((nValAcf+1)*sizeof(double));

  // RDF
  histRdf = (double *)malloc((sizeHistRdf+1)*sizeof(double));
}
```

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.

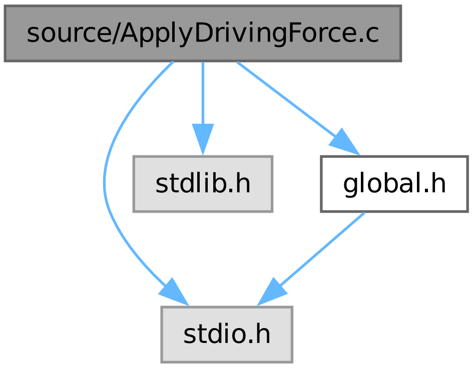
```c
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */

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

void ApplyBoundaryCond(){
  int n;
  for(n = 1 ; n <= nAtom ; n ++){
   if(strcmp(xBoundary, "p") == 0 && strcmp(yBoundary, "p") == 0){          // P.B.C along x and y axis
    rx[n] -= region[1]*rint(rx[n]/region[1]);
    ry[n] -= region[2]*rint(ry[n]/region[2]);
   } else if (strcmp(xBoundary, "r") == 0 && strcmp(yBoundary, "r") == 0){    //R.B.C. along x and y axis
     if((rx[n] + atomRadius[n]) >= regionH[1]){
        rx[n] = 0.999999*regionH[1] - atomRadius[n]; vx[n] = -vx[n] ;
     }if((rx[n]-atomRadius[n]) < -regionH[1]){
        rx[n] = -0.999999*regionH[1] + atomRadius[n]; vx[n] = -vx[n] ;
    }
     if((ry[n] + atomRadius[n])>= regionH[2]){
        ry[n] = 0.999999*regionH[2] - atomRadius[n]; vy[n] = -vy[n] ;
     }if((ry[n]-atomRadius[n]) < -regionH[2]){
        ry[n] = -0.999999*regionH[2] + atomRadius[n]; vy[n] = -vy[n] ;
    }}
    else if (strcmp(xBoundary, "p") == 0 && strcmp(yBoundary, "r") == 0){    //P.B.C. along x and R.B.C along y axis
    rx[n] -= region[1]*rint(rx[n]/region[1]);
    if((ry[n] + atomRadius[n]) >= regionH[2]){
      ry[n] = 0.999999*regionH[2] - atomRadius[n]; vy[n] = -vy[n] ;
     }if((ry[n] - atomRadius[n]) < -regionH[2]){
        ry[n] = -0.999999*regionH[2] + atomRadius[n]; vy[n] = -vy[n] ;
  }}
    else if(strcmp(xBoundary, "r") == 0 && strcmp(yBoundary, "p") == 0){    //R.B.C. along x and P.B.C along y axis
    if((rx[n] + atomRadius[n]) >= regionH[1]){
       rx[n] = 0.999999*regionH[1] - atomRadius[n]; vx[n] = -vx[n] ;
     }if((rx[n] - atomRadius[n]) < -regionH[1]){
      rx[n] = -0.999999*regionH[1] + atomRadius[n]; vx[n] = -vx[n] ;
     }
     ry[n] -= region[2]*rint(ry[n]/region[2]);
  } else {
    // Print error message and exit the program
    fprintf(fpresult, "Error: Invalid boundary configuration: '%s %s'\n", xBoundary, yBoundary);
    exit(EXIT_FAILURE); // Exit with failure status
  }
 }
}
```

3.10 source/ApplyDrivingForce.c File Reference

```c
#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   }
```

```
for(n = 1 ; n <= nAtom; n ++){
 if(atomType[n] == 1 || atomType[n] == 2){
 ax[n] += -gammav * (vx[n] - Vxsubstrate);
 ay[n] += -gammav * (vy[n] - Vysubstrate);
 }
 if(atomType[n] == 3 || atomType[n] == 4){
 ax[n] += -gammav * (vx[n] - Vxblock);
 ay[n] += -gammav * (vy[n] - Vyblock);
} } }
```

References atomType, ax, ay, nAtom, vx, and vy.

3.11 ApplyDrivingForce.c

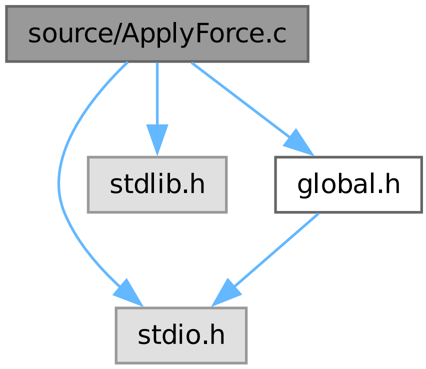
Go to the documentation of this file.

```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */

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

void ApplyDrivingForce(){
 int n;
 double Vxblock, Vyblock;
 double Vxsubstrate, Vysubstrate;
 Vxblock = 0.0;  Vyblock = 0.0;
 Vxsubstrate = 0.0; Vysubstrate = 0.0;
 double gammav;
 gammav = 0.0;

 double count_substrate = 0;
 double count_block = 0;

 for(n = 1 ; n <= nAtom; n ++){
  if(atomType[n] == 1 || atomType[n] == 2){
  Vxsubstrate += vx[n]; Vysubstrate += vy[n];
  count_substrate++;
  }
  if(atomType[n] == 3 || atomType[n] == 4){
  Vxblock += vx[n]; Vyblock += vy[n];
  count_block++;
  } }

  if(count_substrate > 0) {
    Vxsubstrate /= count_substrate;
    Vysubstrate /= count_substrate;
  }

  if(count_block > 0) {
   Vxblock /= count_block;
   Vyblock /= count_block;
  }

 for(n = 1 ; n <= nAtom; n ++){
  if(atomType[n] == 1 || atomType[n] == 2){
  ax[n] += -gammav * (vx[n] - Vxsubstrate);
  ay[n] += -gammav * (vy[n] - Vysubstrate);
  }
  if(atomType[n] == 3 || atomType[n] == 4){
  ax[n] += -gammav * (vx[n] - Vxblock);
  ay[n] += -gammav * (vy[n] - Vyblock);
 } } }
```

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 ( )
```

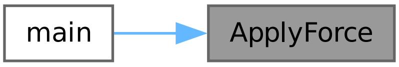
Definition at line 25 of file ApplyForce.c.

```
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

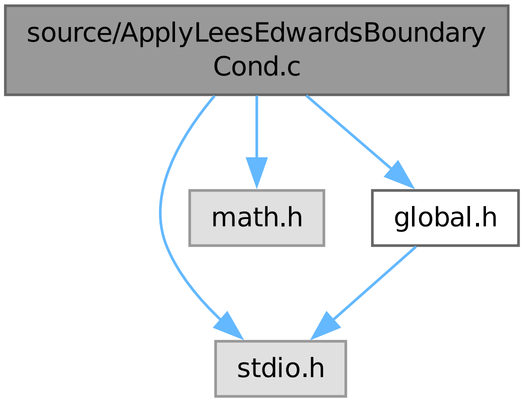
Go to the documentation of this file.

```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */

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

void ApplyForce(){
 int n;
 double lx;
 lx = regionH[1];
 fy =  (FyBylx * lx)/nAtomBlock;
 fx =  fxByfy * fy;
 for(n = 1; n <= nAtom; n ++){
 if(molID[n] == 2){
  ax[n] += fx;
  ay[n] -= fy;
} } }
```

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

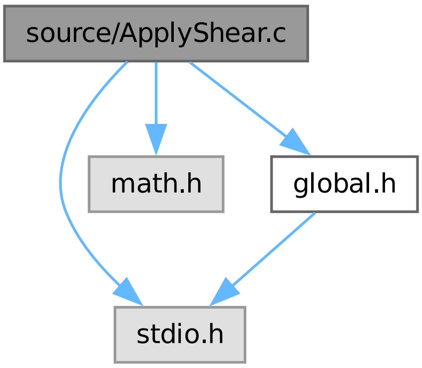
Go to the documentation of this file.

```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */

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

void ApplyLeesEdwardsBoundaryCond() {
 int n;
 for (n = 1; n <= nAtom; n++) {
//PBC along x-direction
 if(rx[n] >= regionH[1])
   rx[n] -= region[1];
 else if(rx[n] < -regionH[1])
   rx[n] += region[1];

//LEBC along y-direction
  if(ry[n] >= regionH[2]){
   rx[n] -= shearDisplacement;
   if(rx[n] < -regionH[1]) rx[n] += region[1];
   //vx[n] -= shearVelocity;
   ry[n] -= region[2];
  }else if(ry[n] < -regionH[2]){
   rx[n] += shearDisplacement;
   if(rx[n] >= regionH[1]) rx[n] -= region[1];
   //vx[n] += shearVelocity;
   ry[n] += region[2];
  }
 }
}
```

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.

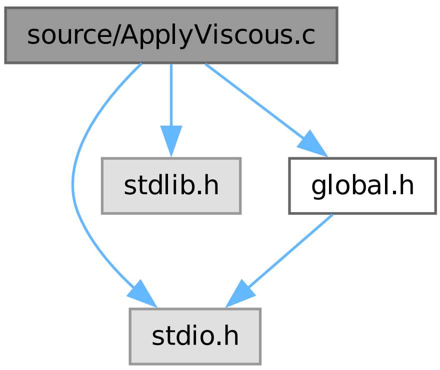
```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina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

```
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */

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

void ApplyShear(){
 int n;
 for(n = 1 ; n <= nAtom ; n ++){
  rx[n] += strain * ry[n];
  //vx[n] += stranRate * ry[n];
 } }
```

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

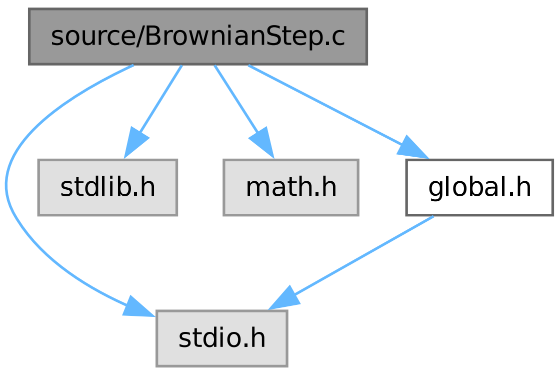
Go to the documentation of this file.

```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */

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

void ApplyViscous(){
 int n;
 double gammav;
 gammav = 1.0;
 for(n = 1 ; n <= nAtom; n ++){
  ax[n] += -gammav * vx[n];
  ay[n] += -gammav * vy[n];
 } }


```

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.

```
                        {
  if(stepCount <= stepEquil){
    double A, S1, S2, T;
    int n;
    S1 = 0.; S2 = 0;
    double halfdt = 0.5*deltaT;
    for (n = 1; n <= nAtom; n++){
      T = vx[n] + halfdt * ax[n];
      S1 += T * ax[n];
      S2 += Sqr(T);

      T = vy[n] + halfdt * ay[n];
      S1 += T * ay[n];
      S2 += Sqr(T);
    }
    A = -S1 / S2;
    double C = 1 + A*deltaT ;
    double D = deltaT * (1 + 0.5 * A * deltaT);
    for (n = 1; n <= nAtom; n++){
      vx[n] = C * vx[n] + D * ax[n];
      rx[n] += deltaT * vx[n];
      vy[n] = C * vy[n] + D * ay[n];
      ry[n] += deltaT * vy[n];
    }
  }else{
      int n;
      //SETTING TEMP = 0.0
     if (stepCount == stepEquil+1){
     for(n = 1 ; n <= nAtom ; n ++){
     vx[n] = 0.0;
     vy[n] = 0.0;
     }}
     double zeta = 1.0;
     double dx, dy;
     for(n = 1 ; n <= nAtom ; n ++){
      dx = rx[n];
      rx[n] += zeta * ax[n] * deltaT;
      dx = rx[n] - dx;
      vx[n] = dx/deltaT;
      dy = ry[n];
      ry[n] += zeta * ay[n] * deltaT;
      dy = ry[n] - dy;
      vy[n] = dy/deltaT;
    }
  }
}
```

References ax, ay, deltaT, nAtom, rx, ry, Sqr, stepCount, stepEquil, vx, and vy.

3.21 BrownianStep.c

Go to the documentation of this file.

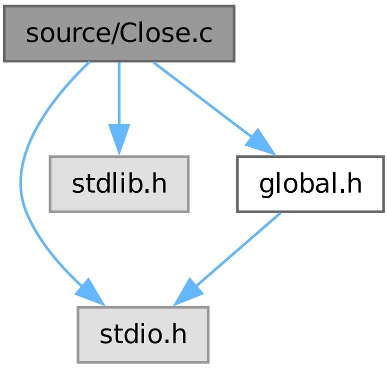
```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina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

```
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */

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

void BrownianStep(){
  if(stepCount <= stepEquil){
    double A, S1, S2, T;
    int n;
    S1 = 0.; S2 = 0;
    double halfdt = 0.5*deltaT;
    for (n = 1; n <= nAtom; n++){
      T = vx[n] + halfdt * ax[n];
      S1 += T * ax[n];
      S2 += Sqr(T);

      T = vy[n] + halfdt * ay[n];
      S1 += T * ay[n];
      S2 += Sqr(T);
    }
    A = -S1 / S2;
    double C = 1 + A*deltaT ;
    double D = deltaT * (1 + 0.5 * A * deltaT);
    for (n = 1; n <= nAtom; n++){
      vx[n] = C * vx[n] + D * ax[n];
      rx[n] += deltaT * vx[n];
      vy[n] = C * vy[n] + D * ay[n];
      ry[n] += deltaT * vy[n];
    }
  }else{
      int n;
      //SETTING TEMP = 0.0
     if (stepCount == stepEquil+1){
     for(n = 1 ; n <= nAtom ; n ++){
     vx[n] = 0.0;
     vy[n] = 0.0;
     }}
     double zeta = 1.0;
     double dx, dy;
     for(n = 1 ; n <= nAtom ; n ++){
      dx = rx[n];
      rx[n] += zeta * ax[n] * deltaT;
      dx = rx[n] - dx;
      vx[n] = dx/deltaT;
      dy = ry[n];
      ry[n] += zeta * ay[n] * deltaT;
      dy = ry[n] - dy;
      vy[n] = dy/deltaT;
    }
  }
}
```

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.

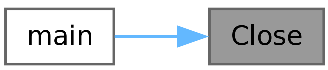
```
00024 {
00025   int n;
00026   free(rx);
00027   free(ry);
00028   free(vx);
00029   free(vy);
00030   free(ax);
00031   free(ay);
00032   free(fax);
00033   free(fay);
00034   free(cellList);
00035 
00036   free(atomID); free(atomType); free(atomRadius); free(atomMass);
00037   free(speed);
00038   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 
```

```
  for(n = 0; n <= nAtom; n++) {
   free(DeltaXijOldPair[n]);
   free(DeltaYijOldPair[n]);
   }
    free(DeltaXijOldPair);
    free(DeltaYijOldPair);

  for  (n = 0; n <= nBuffCorr; n++){
    free(cfOrg[n]);
    free(spacetimeCorr[n]);
  }
  free(cfOrg);
  free(spacetimeCorr);
  free(cfVal);
  free(indexCorr);
  free(spacetimeCorrAv);

  free(indexAcf);
  free(viscAcfOrg);
  free(viscAcfAv);
  for(n = 0 ; n <= nBuffAcf ; n ++)
    free(viscAcf[n]);
  free(viscAcf);

}
```

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.

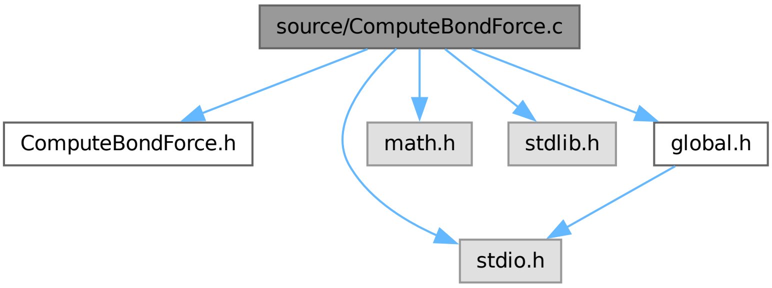
```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */

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

```c
#include"global.h"
void Close(){
  int n;
  free(rx);
  free(ry);
  free(vx);
  free(vy);
  free(ax);
  free(ay);
  free(fax);
  free(fay);
  free(cellList);

  free(atomID); free(atomType); free(atomRadius); free(atomMass);
  free(speed);
  free(atom1); free(atom2); free(BondID);
  free(BondType); free(kb); free(ro);
  free(ImageX); free(ImageY); free(rxUnwrap); free(ryUnwrap);
  free(atomIDInterface);
  free(PairID); free(Pairatom1); free(Pairatom2);
  free(PairXij); free(PairYij);

  free(DeltaXijOld);
  free(DeltaYijOld);

  free(molID);

  for (n = 0; n <= nAtom; n++) {
   free(isBonded[n]);
   }
   free(isBonded);



  for(n = 0; n <= nAtom; n++) {
   free(DeltaXijOldPair[n]);
   free(DeltaYijOldPair[n]);
   }
    free(DeltaXijOldPair);
    free(DeltaYijOldPair);

  for (n = 0; n <= nBuffCorr; n++){
    free(cfOrg[n]);
    free(spacetimeCorr[n]);
  }
  free(cfOrg);
  free(spacetimeCorr);
  free(cfVal);
  free(indexCorr);
  free(spacetimeCorrAv);

  free(indexAcf);
  free(viscAcfOrg);
  free(viscAcfAv);
  for(n = 0 ; n <= nBuffAcf ; n ++)
    free(viscAcf[n]);
  free(viscAcf);

}
```

3.24 source/ComputeBondForce.c File Reference

```c
#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; strech = 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
```

```
    rr = Sqr(dr[1]) + Sqr(dr[2]);
    r = sqrt(rr);
    rri = 1.0/rr;
    ri = 1.0/r;
    roi = 1.0/ro[n];
    strech = (r * roi - 1.0);
    uVal = 0.5 * kb[n] * ro[n] * Sqr(strech);
    fcVal = -kb[n] * strech * ri; //F = -Grad U

    vr[1] = vx[atom1ID] - vx[atom2ID];
    vr[2] = vy[atom1ID] - vy[atom2ID];
    fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //node-node drag

    //DampFlag = 1. LAMMPS version
    if(DampFlag == 1){
    nodeDragx[atom1ID] =  fdVal * dr[1];  //node-node drag  //Important change made on 03Apr2025. Mention it in GitHub
    nodeDragy[atom1ID] =  fdVal * dr[2];  //node-node drag  //Adding the drag forces is wrong. Only add the
    nodeDragx[atom2ID] =  -fdVal * dr[1];  //node-node drag //total force
    nodeDragy[atom2ID] =  -fdVal * dr[2];  //node-node drag

    ax[atom1ID] +=  (fcVal + fdVal) * dr[1];
    ay[atom1ID] +=  (fcVal + fdVal) * dr[2];
    ax[atom2ID] += -(fcVal + fdVal) * dr[1];
    ay[atom2ID] += -(fcVal + fdVal) * dr[2];
    }

    //DampFlag = 2. Suzanne notes version
    else if(DampFlag == 2){
    nodeDragx[atom1ID] =  -gamman * vr[1];  //node-node drag
    nodeDragy[atom1ID] =  -gamman * vr[2]; //node-node drag
    nodeDragx[atom2ID] =  -(-gamman * vr[1]);  //node-node drag
    nodeDragy[atom2ID] =  -(-gamman * vr[2]);  //node-node drag

    ax[atom1ID] +=  (fcVal * dr[1] - gamman * vr[1]);
    ay[atom1ID] +=  (fcVal * dr[2] - gamman * vr[2]);
    ax[atom2ID] += -(fcVal * dr[1] - gamman * vr[1]);
    ay[atom2ID] += -(fcVal * dr[2] - gamman * vr[2]);
    }

    //DampFlag = 3. Suzanne PRL, 130, 178203 (2023) version
    else if(DampFlag == 3){
     DeltaXijNew = dr[1];
     DeltaYijNew = dr[2];

     if(stepCount == 0) {  // First timestep
      DeltaXijOld[n] = DeltaXijNew;
      DeltaYijOld[n] = DeltaYijNew;
     }

     DeltaXij = DeltaXijNew - DeltaXijOld[n];
     DeltaYij = DeltaYijNew - DeltaYijOld[n];
     DeltaVXij = DeltaXij / deltaT;
     DeltaVYij = DeltaYij / deltaT;

     // Now update for the next timestep
     DeltaXijOld[n] = DeltaXijNew;
     DeltaYijOld[n] = DeltaYijNew;

     nodeDragx[atom1ID] =  -gamman * DeltaVXij;  //node-node drag
     nodeDragy[atom1ID] =  -gamman * DeltaVYij; //node-node drag
     nodeDragx[atom2ID] =  -(-gamman * DeltaVXij);  //node-node drag
     nodeDragy[atom2ID] =  -(-gamman * DeltaVYij);  //node-node drag

     ax[atom1ID] +=  (fcVal * dr[1] - gamman * DeltaVXij);
     ay[atom1ID] +=  (fcVal * dr[2] - gamman * DeltaVYij);
     ax[atom2ID] += -(fcVal * dr[1] - gamman * DeltaVXij);
     ay[atom2ID] += -(fcVal * dr[2] - gamman * DeltaVYij);
    }


    BondLength[n] = r;
    BondEnergy[n] = uVal; //No 0.5 factor since it is the energy of the bond
    TotalBondEnergy   +=  BondEnergy[n];

    virSumBond +=   0.5 * (fcVal + fdVal) * rr;
    virSumBondxx += 0.5 * (fcVal + fdVal) * dr[1] * dr[1];
    virSumBondyy += 0.5 * (fcVal + fdVal) * dr[2] * dr[2];
    virSumBondxy += 0.5 * (fcVal + fdVal) * dr[1] * dr[2];
 } }
```

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.

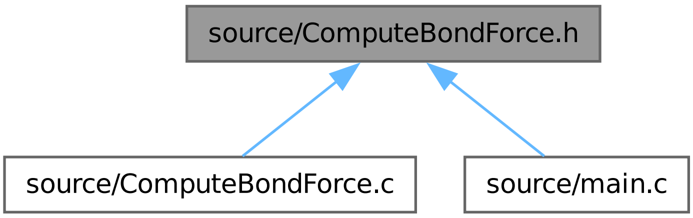
```c
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */

#include "ComputeBondForce.h"

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

void ComputeBondForce(){
  int n;
  double dr[NDIM+1], r, rr, ri, roi;
  double uVal, fcVal;

  uVal = 0.0; TotalBondEnergy = 0.0;
  virSumBond = 0.0; virSumBondxx = 0.0; virSumBondyy = 0.0; virSumBondxy = 0.0;

  double vr[NDIM+1], fdVal, rri;

  for(n = 1 ; n <= nAtom ; n ++){
   nodeDragx[n] = 0.0;
   nodeDragy[n] = 0.0;
  } //Important change made on 03Apr2025. Mention it in GitHub

  int atom1ID, atom2ID;

  for(n=1; n<=nBond; n++){
   rr = 0.0; rri = 0.0; fcVal = 0.0;  fdVal = 0.0; strech = 0.0;
   atom1ID = atom1[n];
   atom2ID = atom2[n];

   dr[1] = rx[atom1ID] - rx[atom2ID];
   if(dr[1] >= regionH[1])
    dr[1] -= region[1];
   else if(dr[1] < -regionH[1])
    dr[1] += region[1];

   dr[2] = ry[atom1ID] - ry[atom2ID];
   if(dr[2] >= regionH[2]){
```

```
    dr[1] -= shearDisplacement;
    if(dr[1] < -regionH[1]) dr[1] += region[1];
    dr[2] -= region[2];
  }else if(dr[2] < -regionH[2]){
   dr[1] += shearDisplacement;
    if(dr[1] >= regionH[1]) dr[1] -= region[1];
   dr[2] += region[2];
  }

   rr = Sqr(dr[1]) + Sqr(dr[2]);
   r = sqrt(rr);
   rri = 1.0/rr;
   ri = 1.0/r;
   roi = 1.0/ro[n];
   strech = (r * roi - 1.0);
   uVal = 0.5 * kb[n] * ro[n] * Sqr(strech);
   fcVal = -kb[n] * strech * ri; //F = -Grad U

   vr[1] = vx[atom1ID] - vx[atom2ID];
   vr[2] = vy[atom1ID] - vy[atom2ID];
   fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //node-node drag

   //DampFlag = 1. LAMMPS version
   if(DampFlag == 1){
   nodeDragx[atom1ID] =  fdVal * dr[1];  //node-node drag  //Important change made on 03Apr2025.
      Mention it in GitHub
   nodeDragy[atom1ID] =  fdVal * dr[2];  //node-node drag  //Adding the drag forces is wrong. Only add
      the
   nodeDragx[atom2ID] =  -fdVal * dr[1];  //node-node drag //total force
   nodeDragy[atom2ID] =  -fdVal * dr[2];  //node-node drag

   ax[atom1ID] +=  (fcVal + fdVal) * dr[1];
   ay[atom1ID] +=  (fcVal + fdVal) * dr[2];
   ax[atom2ID] += -(fcVal + fdVal) * dr[1];
   ay[atom2ID] += -(fcVal + fdVal) * dr[2];
   }

   //DampFlag = 2. Suzanne notes version
   else if(DampFlag == 2){
   nodeDragx[atom1ID] =  -gamman * vr[1];  //node-node drag
   nodeDragy[atom1ID] =  -gamman * vr[2]; //node-node drag
   nodeDragx[atom2ID] =  -(-gamman * vr[1]);  //node-node drag
   nodeDragy[atom2ID] =  -(-gamman * vr[2]);  //node-node drag

   ax[atom1ID] +=  (fcVal * dr[1] - gamman * vr[1]);
   ay[atom1ID] +=  (fcVal * dr[2] - gamman * vr[2]);
   ax[atom2ID] += -(fcVal * dr[1] - gamman * vr[1]);
   ay[atom2ID] += -(fcVal * dr[2] - gamman * vr[2]);
   }

   //DampFlag = 3. Suzanne PRL, 130, 178203 (2023) version
   else if(DampFlag == 3){
    DeltaXijNew = dr[1];
    DeltaYijNew = dr[2];

    if(stepCount == 0) {  // First timestep
     DeltaXijOld[n] = DeltaXijNew;
     DeltaYijOld[n] = DeltaYijNew;
    }

    DeltaXij = DeltaXijNew - DeltaXijOld[n];
    DeltaYij = DeltaYijNew - DeltaYijOld[n];
    DeltaVXij = DeltaXij / deltaT;
    DeltaVYij = DeltaYij / deltaT;

    // Now update for the next timestep
    DeltaXijOld[n] = DeltaXijNew;
    DeltaYijOld[n] = DeltaYijNew;

    nodeDragx[atom1ID] =  -gamman * DeltaVXij;  //node-node drag
    nodeDragy[atom1ID] =  -gamman * DeltaVYij; //node-node drag
    nodeDragx[atom2ID] =  -(-gamman * DeltaVXij);  //node-node drag
    nodeDragy[atom2ID] =  -(-gamman * DeltaVYij);  //node-node drag

    ax[atom1ID] +=  (fcVal * dr[1] - gamman * DeltaVXij);
    ay[atom1ID] +=  (fcVal * dr[2] - gamman * DeltaVYij);
    ax[atom2ID] += -(fcVal * dr[1] - gamman * DeltaVXij);
    ay[atom2ID] += -(fcVal * dr[2] - gamman * DeltaVYij);
   }


   BondLength[n] = r;
   BondEnergy[n] = uVal; //No 0.5 factor since it is the energy of the bond
   TotalBondEnergy   +=  BondEnergy[n];

   virSumBond +=   0.5 * (fcVal + fdVal) * rr;
   virSumBondxx += 0.5 * (fcVal + fdVal) * dr[1] * dr[1];
```

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

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; strech = 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 
```

```
    dr[2] = ry[atom1ID] - ry[atom2ID];
    if(dr[2] >= regionH[2]){
     dr[1] -= shearDisplacement;
     if(dr[1] < -regionH[1]) dr[1] += region[1];
     dr[2] -= region[2];
   }else if(dr[2] < -regionH[2]){
    dr[1] += shearDisplacement;
     if(dr[1] >= regionH[1]) dr[1] -= region[1];
    dr[2] += region[2];
   }

    rr = Sqr(dr[1]) + Sqr(dr[2]);
    r = sqrt(rr);
    rri = 1.0/rr;
    ri = 1.0/r;
    roi = 1.0/ro[n];
    strech = (r * roi - 1.0);
    uVal = 0.5 * kb[n] * ro[n] * Sqr(strech);
    fcVal = -kb[n] * strech * ri; //F = -Grad U

    vr[1] = vx[atom1ID] - vx[atom2ID];
    vr[2] = vy[atom1ID] - vy[atom2ID];
    fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //node-node drag

    //DampFlag = 1. LAMMPS version
    if(DampFlag == 1){
    nodeDragx[atom1ID] =  fdVal * dr[1];  //node-node drag  //Important change made on 03Apr2025. Mention it in GitHub
    nodeDragy[atom1ID] =  fdVal * dr[2];  //node-node drag  //Adding the drag forces is wrong. Only add the
    nodeDragx[atom2ID] =  -fdVal * dr[1];  //node-node drag //total force
    nodeDragy[atom2ID] =  -fdVal * dr[2];  //node-node drag

    ax[atom1ID] +=  (fcVal + fdVal) * dr[1];
    ay[atom1ID] +=  (fcVal + fdVal) * dr[2];
    ax[atom2ID] += -(fcVal + fdVal) * dr[1];
    ay[atom2ID] += -(fcVal + fdVal) * dr[2];
    }

    //DampFlag = 2. Suzanne notes version
    else if(DampFlag == 2){
    nodeDragx[atom1ID] =  -gamman * vr[1];  //node-node drag
    nodeDragy[atom1ID] =  -gamman * vr[2]; //node-node drag
    nodeDragx[atom2ID] =  -(-gamman * vr[1]);  //node-node drag
    nodeDragy[atom2ID] =  -(-gamman * vr[2]);  //node-node drag

    ax[atom1ID] +=  (fcVal * dr[1] - gamman * vr[1]);
    ay[atom1ID] +=  (fcVal * dr[2] - gamman * vr[2]);
    ax[atom2ID] += -(fcVal * dr[1] - gamman * vr[1]);
    ay[atom2ID] += -(fcVal * dr[2] - gamman * vr[2]);
    }

    //DampFlag = 3. Suzanne PRL, 130, 178203 (2023) version
    else if(DampFlag == 3){
     DeltaXijNew = dr[1];
     DeltaYijNew = dr[2];

     if(stepCount == 0) {  // First timestep
      DeltaXijOld[n] = DeltaXijNew;
      DeltaYijOld[n] = DeltaYijNew;
     }

     DeltaXij = DeltaXijNew - DeltaXijOld[n];
     DeltaYij = DeltaYijNew - DeltaYijOld[n];
     DeltaVXij = DeltaXij / deltaT;
     DeltaVYij = DeltaYij / deltaT;

     // Now update for the next timestep
     DeltaXijOld[n] = DeltaXijNew;
     DeltaYijOld[n] = DeltaYijNew;

     nodeDragx[atom1ID] =  -gamman * DeltaVXij;  //node-node drag
     nodeDragy[atom1ID] =  -gamman * DeltaVYij; //node-node drag
     nodeDragx[atom2ID] =  -(-gamman * DeltaVXij);  //node-node drag
     nodeDragy[atom2ID] =  -(-gamman * DeltaVYij);  //node-node drag

     ax[atom1ID] +=  (fcVal * dr[1] - gamman * DeltaVXij);
     ay[atom1ID] +=  (fcVal * dr[2] - gamman * DeltaVYij);
     ax[atom2ID] += -(fcVal * dr[1] - gamman * DeltaVXij);
     ay[atom2ID] += -(fcVal * dr[2] - gamman * DeltaVYij);
    }


    BondLength[n] = r;
    BondEnergy[n] = uVal; //No 0.5 factor since it is the energy of the bond
    TotalBondEnergy   +=  BondEnergy[n];

```

```
    virSumBond +=   0.5 * (fcVal + fdVal) * rr;
    virSumBondxx += 0.5 * (fcVal + fdVal) * dr[1] * dr[1];
    virSumBondyy += 0.5 * (fcVal + fdVal) * dr[2] * dr[2];
    virSumBondxy += 0.5 * (fcVal + fdVal) * dr[1] * dr[2];
} }
```

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

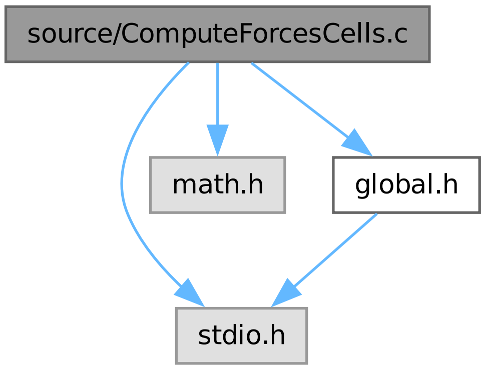
Go to the documentation of this file.

```
#ifndef COMPUTE_BOND_FORCE_H
#define COMPUTE_BOND_FORCE_H

void ComputeBondForce();

#endif

```

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;
```

```
  fdVal = 0.0;

  int start = 1 + rank*(cells[2]/size);
  int end = (rank+1)*(cells[2]/size);

  for(m1Y = start ; m1Y <= end ; m1Y ++){
    for(m1X = 1 ; m1X <= cells[1] ; m1X ++){
      m1 = (m1Y-1) * cells[1] + m1X + nAtom;
      for(offset = 1 ; offset <= 9 ; offset ++){
    m2X = m1X + iofX[offset]; shift[1] = 0.;
    if(m2X > cells[1]){
      m2X = 1; shift[1] = region[1];
    }else if(m2X == 0){
      m2X = cells[1]; shift[1] = -region[1];
    }
    m2Y = m1Y + iofY[offset]; shift[2] = 0.;
    if(m2Y > cells[2]){
      m2Y = 1; shift[2] = region[2];
    }else if(m2Y == 0){
      m2Y = cells[2]; shift[2] = -region[2];
    }
    m2 = (m2Y-1)*cells[1] + m2X + nAtom;
    I = cellList[m1];
    while(I > 0){
      J = cellList[m2];
      while(J > 0){
        if(m1 == m2 && J != I && (atomRadius[I] > 0. && atomRadius[J] > 0.)){
          dr[1] = rx[I] - rx[J] - shift[1];
          dr[2] = ry[I] - ry[J] - shift[2];
          rr = Sqr(dr[1]) + Sqr(dr[2]);
          RadiusIJ = atomRadius[I] + atomRadius[J];
          SqrRadiusIJ = Sqr(RadiusIJ);
          if(rr < SqrRadiusIJ){
        r = sqrt(rr);
        ri = 1.0/r;
                rrinv = 1.0/rr;
                vr[1] = vx[I] - vx[J];
                vr[2] = vy[I] - vy[J];
        RadiusIJInv = 1.0/RadiusIJ;
        uVal = Sqr(1.0 - r * RadiusIJInv);
        fcVal = 2.0 * RadiusIJInv * (1.0 - r * RadiusIJInv) *ri;
                fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag

        f  = fcVal * dr[1];
                fd = fdVal * dr[1];
        ax[I] += (f + fd);
                discDragx[I] += fd; //disc-disc drag

        f = fcVal * dr[2];
                fd = fdVal * dr[2];
        ay[I] += (f + fd);
                discDragy[I] += fd; //disc-disc drag

        uSum +=  0.5 * uVal;
        virSum += 0.5 * fcVal * rr;
        rfAtom += 0.5 * dr[1] * fcVal * dr[2];
          }
        }else if(m1 != m2 && (atomRadius[I] > 0. && atomRadius[J] > 0.)){
          dr[1] = rx[I] - rx[J] - shift[1];
          dr[2] = ry[I] - ry[J] - shift[2];
          rr = Sqr(dr[1]) + Sqr(dr[2]);
          RadiusIJ = atomRadius[I] + atomRadius[J];
          SqrRadiusIJ = Sqr(RadiusIJ);
          if(rr < SqrRadiusIJ){
        r = sqrt(rr);
        ri = 1.0/r;
                rrinv = 1.0/r;
                vr[1] = vx[I] - vx[J];
                vr[2] = vy[I] - vy[J];
        RadiusIJInv = 1.0/RadiusIJ;
        uVal = Sqr(1.0 - r * RadiusIJInv);
        fcVal = 2.0 * RadiusIJInv * (1.0 - r * RadiusIJInv) *ri;
                fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag

        f  = fcVal * dr[1];
                fd =  fdVal * dr[1];
        ax[I] += (f + fd);
                discDragx[I] += fd; //disc-disc drag

        f  = fcVal * dr[2];
                fd = fdVal * dr[2];
        ay[I] += (f + fd);
                discDragy[I] += fd; //disc-disc drag

        uSum +=  0.5 * uVal;
        virSum += 0.5 * fcVal * rr;
        rfAtom += 0.5 * dr[1] * fcVal * dr[2];
```

```
          }
        }
            J = cellList[J];
      }
      I = cellList[I];
    }
      }
    }
  }
}
```

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.

```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */

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

void ComputeForcesCells(){
  double dr[NDIM+1], invWid[NDIM+1], shift[NDIM+1], f, fcVal, rr, ri, r, uVal;
  int c, I, J, m1, m1X, m1Y, m2, m2X, m2Y, n, offset;
  int iofX[] = {0, 0, 1, 1, 0, -1, -1, -1, 0, 1},
      iofY[] = {0, 0, 0, 1 ,1, 1, 0, -1, -1, -1};

  invWid[1] = cells[1]/region[1];
  invWid[2] = cells[2]/region[2];

  for(n = nAtom+1; n <= nAtom+cells[1]*cells[2] ; n++)
    cellList[n] = 0;

  for(n = 1 ; n <= nAtom ; n ++){
    c = ((int)((ry[n] + regionH[2])*invWid[2]))*cells[1] + (int)((rx[n]+regionH[1])*invWid[1]) + nAtom+ 1;
    cellList[n] = cellList[c];
    cellList[c] = n;
  }

  for(n = 1 ; n <= nAtom ; n ++){
    ax[n] = 0.;
    ay[n] = 0.;
  }

  uSum = 0.0 ;
  virSum = 0.0;
  rfAtom = 0.0;
  RadiusIJ = 0.0;

  gamman = 1.0;
  double vr[NDIM+1], fd, fdVal, rrinv;
  rrinv = 0.0;
  fd = 0.0;
  fdVal = 0.0;

  int start = 1 + rank*(cells[2]/size);
  int end = (rank+1)*(cells[2]/size);

  for(m1Y = start ; m1Y <= end ; m1Y ++){
```

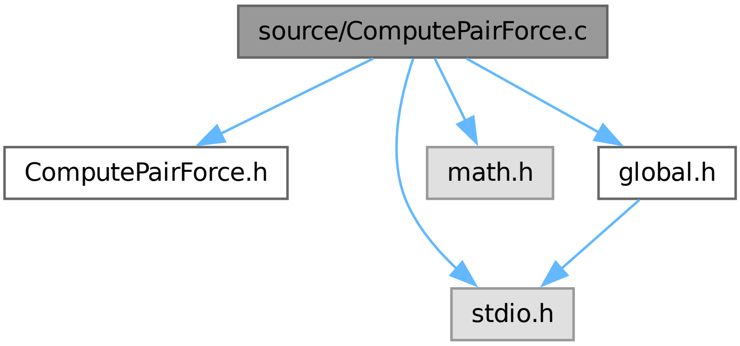
```
for(m1X = 1 ; m1X <= cells[1] ; m1X ++){
  m1 = (m1Y-1) * cells[1] + m1X + nAtom;
  for(offset = 1 ; offset <= 9 ; offset ++){
m2X = m1X + iofX[offset]; shift[1] = 0.;
if(m2X > cells[1]){
  m2X = 1; shift[1] = region[1];
}else if(m2X == 0){
  m2X = cells[1]; shift[1] = -region[1];
}
m2Y = m1Y + iofY[offset]; shift[2] = 0.;
if(m2Y > cells[2]){
  m2Y = 1; shift[2] = region[2];
}else if(m2Y == 0){
  m2Y = cells[2]; shift[2] = -region[2];
}
m2 = (m2Y-1)*cells[1] + m2X + nAtom;
I = cellList[m1];
while(I > 0){
  J = cellList[m2];
  while(J > 0){
    if(m1 == m2 && J != I && (atomRadius[I] > 0. && atomRadius[J] > 0.)){
      dr[1] = rx[I] - rx[J] - shift[1];
      dr[2] = ry[I] - ry[J] - shift[2];
      rr = Sqr(dr[1]) + Sqr(dr[2]);
      RadiusIJ = atomRadius[I] + atomRadius[J];
      SqrRadiusIJ = Sqr(RadiusIJ);
      if(rr < SqrRadiusIJ){
    r = sqrt(rr);
    ri = 1.0/r;
            rrinv = 1.0/rr;
            vr[1] = vx[I] - vx[J];
            vr[2] = vy[I] - vy[J];
    RadiusIJInv = 1.0/RadiusIJ;
    uVal = Sqr(1.0 - r * RadiusIJInv);
    fcVal = 2.0 * RadiusIJInv * (1.0 - r * RadiusIJInv) *ri;
            fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag

    f  = fcVal * dr[1];
            fd = fdVal * dr[1];
    ax[I] += (f + fd);
            discDragx[I] += fd; //disc-disc drag

    f = fcVal * dr[2];
            fd = fdVal * dr[2];
    ay[I] += (f + fd);
            discDragy[I] += fd; //disc-disc drag

    uSum +=  0.5 * uVal;
    virSum += 0.5 * fcVal * rr;
    rfAtom += 0.5 * dr[1] * fcVal * dr[2];
      }
    }else if(m1 != m2 && (atomRadius[I] > 0. && atomRadius[J] > 0.)){
      dr[1] = rx[I] - rx[J] - shift[1];
      dr[2] = ry[I] - ry[J] - shift[2];
      rr = Sqr(dr[1]) + Sqr(dr[2]);
      RadiusIJ = atomRadius[I] + atomRadius[J];
      SqrRadiusIJ = Sqr(RadiusIJ);
      if(rr < SqrRadiusIJ){
    r = sqrt(rr);
    ri = 1.0/r;
            rrinv = 1.0/r;
            vr[1] = vx[I] - vx[J];
            vr[2] = vy[I] - vy[J];
    RadiusIJInv = 1.0/RadiusIJ;
    uVal = Sqr(1.0 - r * RadiusIJInv);
    fcVal = 2.0 * RadiusIJInv * (1.0 - r * RadiusIJInv) *ri;
            fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag

    f  = fcVal * dr[1];
            fd =  fdVal * dr[1];
    ax[I] += (f + fd);
            discDragx[I] += fd; //disc-disc drag

    f  = fcVal * dr[2];
            fd = fdVal * dr[2];
    ay[I] += (f + fd);
            discDragy[I] += fd; //disc-disc drag

    uSum +=  0.5 * uVal;
    virSum += 0.5 * fcVal * rr;
    rfAtom += 0.5 * dr[1] * fcVal * dr[2];
      }
    }
        J = cellList[J];
  }
  I = cellList[I];
}
```

```
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;
```

```
 Pairatom1[n] = 0;
 Pairatom2[n] = 0;
 PairXij[n] = 0.0;
 PairYij[n] = 0.0;
}


Kn = 1.0;
double vr[NDIM+1], fdVal, rri;
nPairActive = 0;
double meff;
meff = 0.0;
int atomIDi, atomIDj;
//int processThisPair = 1;

for(i=1;i<=nAtomInterface;i++){
 for(j=i+1;j<=nAtomInterface;j++){
  atomIDi = atomIDInterface[i];
  atomIDj = atomIDInterface[j];
  if (isBonded[atomIDi][atomIDj] == 0) { //To exclude pair interaction between bonded atoms
  rr = 0.0; rri = 0.0; fcVal = 0.0;  fdVal = 0.0; strech = 0.0;
  RadiusIJ = 0.0;

  dr[1] = rx[atomIDi] - rx[atomIDj];
  if(dr[1] >= regionH[1])
   dr[1] -= region[1];
  else if(dr[1] < -regionH[1])
    dr[1] += region[1];

  dr[2] = ry[atomIDi] - ry[atomIDj];
   if(dr[2] >= regionH[2]){
    dr[1] -= shearDisplacement;
    if(dr[1] < -regionH[1]) dr[1] += region[1];
    dr[2] -= region[2];
  }else if(dr[2] < -regionH[2]){
   dr[1] += shearDisplacement;
    if(dr[1] >= regionH[1]) dr[1] -= region[1];
   dr[2] += region[2];
  }

  rr = Sqr(dr[1]) + Sqr(dr[2]);
  RadiusIJ = atomRadius[atomIDi] + atomRadius[atomIDj];
  SqrRadiusIJ = Sqr(RadiusIJ);
  if(rr < SqrRadiusIJ){
   r = sqrt(rr);
   ri = 1.0/r;
   rri = 1.0/rr;
   RadiusIJInv = 1.0/RadiusIJ;
   strech = (RadiusIJ - r);
   uVal = 0.5 * Kn * Sqr(strech);

   //NormFlag
   if(normFlag == 1){
    strech = strech * RadiusIJInv;
    uVal = 0.5 * Kn * RadiusIJ * Sqr(strech);
    }

   fcVal = Kn * strech * ri;
   vr[1] = vx[atomIDi] - vx[atomIDj];
   vr[2] = vy[atomIDi] - vy[atomIDj];

   nPairActive++;
   PairID[nPairActive] = nPairActive;
   Pairatom1[nPairActive] = atomIDi;
   Pairatom2[nPairActive] = atomIDj;
   PairXij[nPairActive] = dr[1];
   PairYij[nPairActive] = dr[2];

   //DampFlag = 1
   if(DampFlag == 1){
   meff = (atomMass[atomIDi]*atomMass[atomIDj])/(atomMass[atomIDi] + atomMass[atomIDj]);
   fdVal = -gamman * meff * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //disc-disc drag

   discDragx[atomIDi] =  fdVal * dr[1]; //disc-disc drag
   discDragy[atomIDi] =  fdVal * dr[2]; //disc-disc drag
   discDragx[atomIDj] = -fdVal * dr[1]; //disc-disc drag
   discDragy[atomIDj] = -fdVal * dr[2]; //disc-disc drag

   discDragx[nPairActive] = discDragx[atomIDi];
   discDragy[nPairActive] = discDragy[atomIDi];


   ax[atomIDi] += (fcVal + fdVal) * dr[1];
   ay[atomIDi] += (fcVal + fdVal) * dr[2];
   ax[atomIDj] += -(fcVal + fdVal) * dr[1];
   ay[atomIDj] += -(fcVal + fdVal) * dr[2];
  }
```

```
    //DampFlag = 2
    else if(DampFlag == 2){
     discDragx[atomIDi] =  -gamman * vr[1]; //disc-disc drag
     discDragy[atomIDi] =  -gamman * vr[2]; //disc-disc drag
     discDragx[atomIDj] = -(-gamman * vr[1]); //disc-disc drag
     discDragy[atomIDj] = -(-gamman * vr[2]); //disc-disc drag

     discDragx[nPairActive] = discDragx[atomIDi];
     discDragy[nPairActive] = discDragy[atomIDi];


     ax[atomIDi] +=  (fcVal * dr[1] - gamman * vr[1]);
     ay[atomIDi] +=  (fcVal * dr[2] - gamman * vr[2]);
     ax[atomIDj] += -(fcVal * dr[1] - gamman * vr[1]);
     ay[atomIDj] += -(fcVal * dr[2] - gamman * vr[2]);
    }

    //DampFlag = 3. Suzanne PRL, 130, 178203 (2023) version
    else if(DampFlag == 3){
    //Track compression velocity
    DeltaXijNew = dr[1];
    DeltaYijNew = dr[2];
    if(stepCount == 0) { // Initialization step
     DeltaXijOldPair[atomIDi][atomIDj] = DeltaXijNew;
     DeltaYijOldPair[atomIDi][atomIDj] = DeltaYijNew;
     }

     DeltaXij = DeltaXijNew - DeltaXijOldPair[atomIDi][atomIDj];
     DeltaYij = DeltaYijNew - DeltaYijOldPair[atomIDi][atomIDj];
     DeltaVXij = DeltaXij / deltaT;
     DeltaVYij = DeltaYij / deltaT;

     // Update history for next step
     DeltaXijOldPair[atomIDi][atomIDj] = DeltaXijNew;
     DeltaYijOldPair[atomIDi][atomIDj] = DeltaYijNew;

     discDragx[atomIDi] =  -gamman * DeltaVXij; //disc-disc drag
     discDragy[atomIDi] =  -gamman * DeltaVYij; //disc-disc drag
     discDragx[atomIDj] = -(-gamman * DeltaVXij); //disc-disc drag
     discDragy[atomIDj] = -(-gamman * DeltaVYij); //disc-disc drag

     discDragx[nPairActive] = discDragx[atomIDi];
     discDragy[nPairActive] = discDragy[atomIDi];

     ax[atomIDi] +=  (fcVal * dr[1] - gamman * DeltaVXij);
     ay[atomIDi] +=  (fcVal * dr[2] - gamman * DeltaVYij);
     ax[atomIDj] += -(fcVal * dr[1] - gamman * DeltaVXij);
     ay[atomIDj] += -(fcVal * dr[2] - gamman * DeltaVYij);
    }

    //In the following, for stress/virial term (fcVal + fdVal) is used since the total pair force = Hookean Interaction + relative velocity drag
    uSumPair +=  0.5 * uVal;
    virSumPair += 0.5 * (fcVal + fdVal) * rr;
    virSumPairxx += 0.5 * (fcVal + fdVal) * dr[1] * dr[1];
    virSumPairyy += 0.5 * (fcVal + fdVal) * dr[2] * dr[2];
    virSumPairxy += 0.5 * (fcVal + fdVal) * dr[1] * dr[2];
      }
    else { //Resetting the distance between two discs when they are not in contact
     DeltaXijOldPair[atomIDi][atomIDj] = 0.0;
     DeltaYijOldPair[atomIDi][atomIDj] = 0.0;
     DeltaXijOldPair[atomIDj][atomIDi] = 0.0;
     DeltaYijOldPair[atomIDj][atomIDi] = 0.0;
      }
     }
    }
   }
 }
```

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.

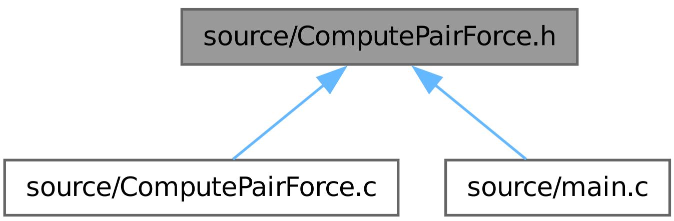
```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */

#include "ComputePairForce.h"

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

void ComputePairForce(int normFlag){
double dr[NDIM+1], fcVal, rr, ri, r, uVal;
int n, i,j;
uVal = 0.0;  uSumPair = 0.0 ;
virSumPair = 0.0; virSumPairxx = 0.0; virSumPairyy = 0.0; virSumPairxy = 0.0;

for(n = 1 ; n <= nAtom ; n ++){
 ax[n] = 0.0;
 ay[n] = 0.0;
 discDragx[n] = 0.0;
 discDragy[n] = 0.0;
}
for(n = 1; n <= nPairTotal; n ++){
 PairID[n] = 0;
 Pairatom1[n] = 0;
 Pairatom2[n] = 0;
 PairXij[n] = 0.0;
 PairYij[n] = 0.0;
}


Kn = 1.0;
double vr[NDIM+1], fdVal, rri;
nPairActive = 0;
double meff;
meff = 0.0;
int atomIDi, atomIDj;
//int processThisPair = 1;

for(i=1;i<=nAtomInterface;i++){
 for(j=i+1;j<=nAtomInterface;j++){
  atomIDi = atomIDInterface[i];
  atomIDj = atomIDInterface[j];
  if (isBonded[atomIDi][atomIDj] == 0) { //To exclude pair interaction between bonded atoms
```

```
rr = 0.0; rri = 0.0; fcVal = 0.0;  fdVal = 0.0; strech = 0.0;
RadiusIJ = 0.0;

dr[1] = rx[atomIDi] - rx[atomIDj];
if(dr[1] >= regionH[1])
 dr[1] -= region[1];
else if(dr[1] < -regionH[1])
  dr[1] += region[1];

dr[2] = ry[atomIDi] - ry[atomIDj];
 if(dr[2] >= regionH[2]){
  dr[1] -= shearDisplacement;
  if(dr[1] < -regionH[1]) dr[1] += region[1];
  dr[2] -= region[2];
}else if(dr[2] < -regionH[2]){
 dr[1] += shearDisplacement;
  if(dr[1] >= regionH[1]) dr[1] -= region[1];
 dr[2] += region[2];
}

rr = Sqr(dr[1]) + Sqr(dr[2]);
RadiusIJ = atomRadius[atomIDi] + atomRadius[atomIDj];
SqrRadiusIJ = Sqr(RadiusIJ);
if(rr < SqrRadiusIJ){
 r = sqrt(rr);
 ri = 1.0/r;
 rri = 1.0/rr;
 RadiusIJInv = 1.0/RadiusIJ;
 strech = (RadiusIJ - r);
 uVal = 0.5 * Kn * Sqr(strech);

 //NormFlag
 if(normFlag == 1){
  strech = strech * RadiusIJInv;
  uVal = 0.5 * Kn * RadiusIJ * Sqr(strech);
  }

 fcVal = Kn * strech * ri;
 vr[1] = vx[atomIDi] - vx[atomIDj];
 vr[2] = vy[atomIDi] - vy[atomIDj];

 nPairActive++;
 PairID[nPairActive] = nPairActive;
 Pairatom1[nPairActive] = atomIDi;
 Pairatom2[nPairActive] = atomIDj;
 PairXij[nPairActive] = dr[1];
 PairYij[nPairActive] = dr[2];

 //DampFlag = 1
 if(DampFlag == 1){
 meff = (atomMass[atomIDi]*atomMass[atomIDj])/(atomMass[atomIDi] + atomMass[atomIDj]);
 fdVal = -gamman * meff * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //disc-disc drag

 discDragx[atomIDi] =  fdVal * dr[1]; //disc-disc drag
 discDragy[atomIDi] =  fdVal * dr[2]; //disc-disc drag
 discDragx[atomIDj] = -fdVal * dr[1]; //disc-disc drag
 discDragy[atomIDj] = -fdVal * dr[2]; //disc-disc drag

 discDragx[nPairActive] = discDragx[atomIDi];
 discDragy[nPairActive] = discDragy[atomIDi];


 ax[atomIDi] += (fcVal + fdVal) * dr[1];
 ay[atomIDi] += (fcVal + fdVal) * dr[2];
 ax[atomIDj] += -(fcVal + fdVal) * dr[1];
 ay[atomIDj] += -(fcVal + fdVal) * dr[2];
}

//DampFlag = 2
else if(DampFlag == 2){
 discDragx[atomIDi] =  -gamman * vr[1]; //disc-disc drag
 discDragy[atomIDi] =  -gamman * vr[2]; //disc-disc drag
 discDragx[atomIDj] = -(-gamman * vr[1]); //disc-disc drag
 discDragy[atomIDj] = -(-gamman * vr[2]); //disc-disc drag

 discDragx[nPairActive] = discDragx[atomIDi];
 discDragy[nPairActive] = discDragy[atomIDi];


 ax[atomIDi] +=  (fcVal * dr[1] - gamman * vr[1]);
 ay[atomIDi] +=  (fcVal * dr[2] - gamman * vr[2]);
 ax[atomIDj] += -(fcVal * dr[1] - gamman * vr[1]);
 ay[atomIDj] += -(fcVal * dr[2] - gamman * vr[2]);
}

//DampFlag = 3. Suzanne PRL, 130, 178203 (2023) version
else if(DampFlag == 3){
```

```
  //Track compression velocity
  DeltaXijNew = dr[1];
  DeltaYijNew = dr[2];
  if(stepCount == 0) { // Initialization step
   DeltaXijOldPair[atomIDi][atomIDj] = DeltaXijNew;
   DeltaYijOldPair[atomIDi][atomIDj] = DeltaYijNew;
   }

   DeltaXij = DeltaXijNew - DeltaXijOldPair[atomIDi][atomIDj];
   DeltaYij = DeltaYijNew - DeltaYijOldPair[atomIDi][atomIDj];
   DeltaVXij = DeltaXij / deltaT;
   DeltaVYij = DeltaYij / deltaT;

   // Update history for next step
   DeltaXijOldPair[atomIDi][atomIDj] = DeltaXijNew;
   DeltaYijOldPair[atomIDi][atomIDj] = DeltaYijNew;

   discDragx[atomIDi] =  -gamman * DeltaVXij; //disc-disc drag
   discDragy[atomIDi] =  -gamman * DeltaVYij; //disc-disc drag
   discDragx[atomIDj] = -(-gamman * DeltaVXij); //disc-disc drag
   discDragy[atomIDj] = -(-gamman * DeltaVYij); //disc-disc drag

   discDragx[nPairActive] = discDragx[atomIDi];
   discDragy[nPairActive] = discDragy[atomIDi];

   ax[atomIDi] +=  (fcVal * dr[1] - gamman * DeltaVXij);
   ay[atomIDi] +=  (fcVal * dr[2] - gamman * DeltaVYij);
   ax[atomIDj] += -(fcVal * dr[1] - gamman * DeltaVXij);
   ay[atomIDj] += -(fcVal * dr[2] - gamman * DeltaVYij);
  }

  //In the following, for stress/virial term (fcVal + fdVal) is used since the total pair force = Hookean Interaction + relative velocity drag
  uSumPair +=  0.5 * uVal;
  virSumPair += 0.5 * (fcVal + fdVal) * rr;
  virSumPairxx += 0.5 * (fcVal + fdVal) * dr[1] * dr[1];
  virSumPairyy += 0.5 * (fcVal + fdVal) * dr[2] * dr[2];
  virSumPairxy += 0.5 * (fcVal + fdVal) * dr[1] * dr[2];
    }
  else { //Resetting the distance between two discs when they are not in contact
   DeltaXijOldPair[atomIDi][atomIDj] = 0.0;
   DeltaYijOldPair[atomIDi][atomIDj] = 0.0;
   DeltaXijOldPair[atomIDj][atomIDi] = 0.0;
   DeltaYijOldPair[atomIDj][atomIDi] = 0.0;
    }
   }
  }
 }
}
```

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
```

```
 if(normFlag == 1){
  strech = strech * RadiusIJInv;
  uVal = 0.5 * Kn * RadiusIJ * Sqr(strech);
  }

 fcVal = Kn * strech * ri;
 vr[1] = vx[atomIDi] - vx[atomIDj];
 vr[2] = vy[atomIDi] - vy[atomIDj];

 nPairActive++;
 PairID[nPairActive] = nPairActive;
 Pairatom1[nPairActive] = atomIDi;
 Pairatom2[nPairActive] = atomIDj;
 PairXij[nPairActive] = dr[1];
 PairYij[nPairActive] = dr[2];

 //DampFlag = 1
 if(DampFlag == 1){
 meff = (atomMass[atomIDi]*atomMass[atomIDj])/(atomMass[atomIDi] + atomMass[atomIDj]);
 fdVal = -gamman * meff * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //disc-disc drag

 discDragx[atomIDi] =  fdVal * dr[1]; //disc-disc drag
 discDragy[atomIDi] =  fdVal * dr[2]; //disc-disc drag
 discDragx[atomIDj] = -fdVal * dr[1]; //disc-disc drag
 discDragy[atomIDj] = -fdVal * dr[2]; //disc-disc drag

 discDragx[nPairActive] = discDragx[atomIDi];
 discDragy[nPairActive] = discDragy[atomIDi];


 ax[atomIDi] += (fcVal + fdVal) * dr[1];
 ay[atomIDi] += (fcVal + fdVal) * dr[2];
 ax[atomIDj] += -(fcVal + fdVal) * dr[1];
 ay[atomIDj] += -(fcVal + fdVal) * dr[2];
}

//DampFlag = 2
else if(DampFlag == 2){
 discDragx[atomIDi] =  -gamman * vr[1]; //disc-disc drag
 discDragy[atomIDi] =  -gamman * vr[2]; //disc-disc drag
 discDragx[atomIDj] = -(-gamman * vr[1]); //disc-disc drag
 discDragy[atomIDj] = -(-gamman * vr[2]); //disc-disc drag

 discDragx[nPairActive] = discDragx[atomIDi];
 discDragy[nPairActive] = discDragy[atomIDi];


 ax[atomIDi] +=  (fcVal * dr[1] - gamman * vr[1]);
 ay[atomIDi] +=  (fcVal * dr[2] - gamman * vr[2]);
 ax[atomIDj] += -(fcVal * dr[1] - gamman * vr[1]);
 ay[atomIDj] += -(fcVal * dr[2] - gamman * vr[2]);
}

//DampFlag = 3. Suzanne PRL, 130, 178203 (2023) version
else if(DampFlag == 3){
//Track compression velocity
DeltaXijNew = dr[1];
DeltaYijNew = dr[2];
if(stepCount == 0) { // Initialization step
 DeltaXijOldPair[atomIDi][atomIDj] = DeltaXijNew;
 DeltaYijOldPair[atomIDi][atomIDj] = DeltaYijNew;
 }

 DeltaXij = DeltaXijNew - DeltaXijOldPair[atomIDi][atomIDj];
 DeltaYij = DeltaYijNew - DeltaYijOldPair[atomIDi][atomIDj];
 DeltaVXij = DeltaXij / deltaT;
 DeltaVYij = DeltaYij / deltaT;

 // Update history for next step
 DeltaXijOldPair[atomIDi][atomIDj] = DeltaXijNew;
 DeltaYijOldPair[atomIDi][atomIDj] = DeltaYijNew;

 discDragx[atomIDi] =  -gamman * DeltaVXij; //disc-disc drag
 discDragy[atomIDi] =  -gamman * DeltaVYij; //disc-disc drag
 discDragx[atomIDj] = -(-gamman * DeltaVXij); //disc-disc drag
 discDragy[atomIDj] = -(-gamman * DeltaVYij); //disc-disc drag

 discDragx[nPairActive] = discDragx[atomIDi];
 discDragy[nPairActive] = discDragy[atomIDi];

 ax[atomIDi] +=  (fcVal * dr[1] - gamman * DeltaVXij);
 ay[atomIDi] +=  (fcVal * dr[2] - gamman * DeltaVYij);
 ax[atomIDj] += -(fcVal * dr[1] - gamman * DeltaVXij);
 ay[atomIDj] += -(fcVal * dr[2] - gamman * DeltaVYij);
}

//In the following, for stress/virial term (fcVal + fdVal) is used since the total pair force =
```

```
Hookean Interaction + relative velocity drag
  uSumPair +=  0.5 * uVal;
  virSumPair += 0.5 * (fcVal + fdVal) * rr;
  virSumPairxx += 0.5 * (fcVal + fdVal) * dr[1] * dr[1];
  virSumPairyy += 0.5 * (fcVal + fdVal) * dr[2] * dr[2];
  virSumPairxy += 0.5 * (fcVal + fdVal) * dr[1] * dr[2];
    }
  else { //Resetting the distance between two discs when they are not in contact
   DeltaXijOldPair[atomIDi][atomIDj] = 0.0;
   DeltaYijOldPair[atomIDi][atomIDj] = 0.0;
   DeltaXijOldPair[atomIDj][atomIDi] = 0.0;
   DeltaYijOldPair[atomIDj][atomIDi] = 0.0;
    }
   }
  }
 }
}
```

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.

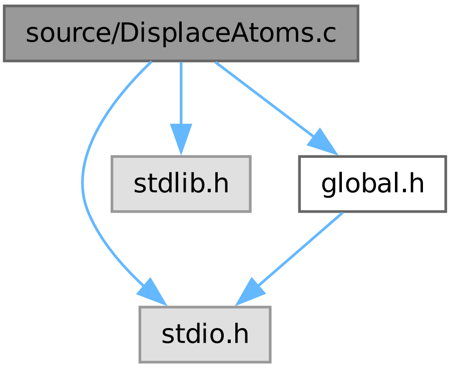
```
#ifndef COMPUTE_PAIR_FORCE_H
#define COMPUTE_PAIR_FORCE_H

void ComputePairForce(int normFlag);

#endif

```

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 ( )
```

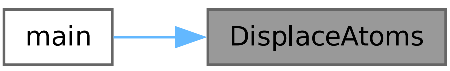
Definition at line 25 of file DisplaceAtoms.c.

```
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

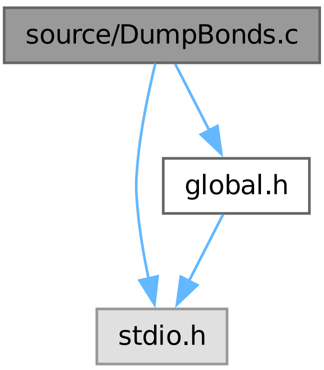
Go to the documentation of this file.

```c
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */

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

void DisplaceAtoms(){
 int n;
  for(n = 1; n <= nAtom; n ++){
   if(molID[n] == 2){
    rx[n] += DeltaX;
    ry[n] += DeltaY;
} } }
```

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 ( )
```

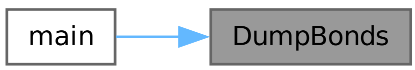
Definition at line 24 of file DumpBonds.c.

```
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.

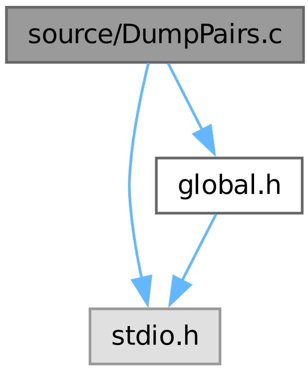
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your 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>
```

```
#include"global.h"

void DumpBonds(){
  int n;
  //Trajectory file in LAMMPS dump format for OVITO visualization
  fprintf(fpbond, "ITEM: TIMESTEP\n");
  fprintf(fpbond, "%lf\n",timeNow);
  fprintf(fpbond, "ITEM: NUMBER OF ENTRIES\n");
  fprintf(fpbond, "%d\n",nBond);
  fprintf(fpbond, "ITEM: BOX BOUNDS pp ff pp\n");
  fprintf(fpbond, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
  fprintf(fpbond, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
  fprintf(fpbond, "%lf %lf zlo zhi\n", -0.1, 0.1);
  fprintf(fpbond, "ITEM: ENTRIES BondID, BondType, atom1 atom2 BondLength BondLengthEqul nodeDragx1 nodeDragy1\n");

  for(n=1; n<=nBond; n++)
   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]]);
  }



```

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 ( )`

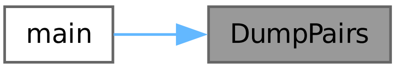
Definition at line 25 of file DumpPairs.c.

```
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],
      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:



3.39 DumpPairs.c

Go to the documentation of this file.

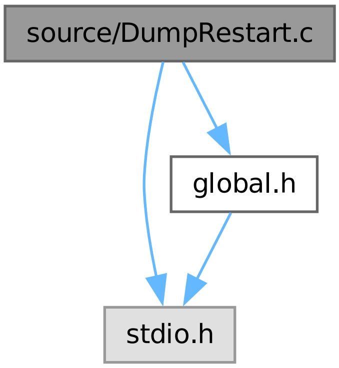
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your 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);
```

```
  fprintf(fppair, "ITEM: BOX BOUNDS pp ff pp\n");
  fprintf(fppair, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
  fprintf(fppair, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
  fprintf(fppair, "%lf %lf zlo zhi\n", -0.1, 0.1);
  fprintf(fppair, "ITEM: ENTRIES index, atom1 atom2 xij yij discDragx1 discDragy1\n");

  for(n=1; n<=nPairActive; n++)
   fprintf(fppair, "%d %d %d %0.16lf %0.16lf %0.16lf %0.16lf\n", PairID[n], Pairatom1[n], Pairatom2[n],
   PairXij[n], PairYij[n], discDragx[n], discDragy[n]);

  }
```

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.

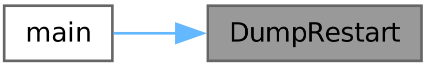
```
00025                    {
00026  char DUMP[256];
00027  FILE *fpDUMP;
00028  sprintf(DUMP, "%s.Restart", prefix);
00029  fpDUMP = fopen(DUMP, "w");
00030  if(fpDUMP == NULL) {
```

```
  fprintf(stderr, "Error opening file %s for writing\n", DUMP);
  return;
 }

  fprintf(fpDUMP, "timeNow %lf\n", timeNow);
  fprintf(fpDUMP, "nAtom %d\n",  nAtom);
  fprintf(fpDUMP, "nBond %d\n", nBond);
  fprintf(fpDUMP, "nAtomType %d\n", nAtomType);
  fprintf(fpDUMP, "nBondType %d\n", nBondType);
  fprintf(fpDUMP, "region[1] %0.14lf\n", region[1]);
  fprintf(fpDUMP, "region[2] %0.14lf\n", region[2]);

  int n;
  fprintf(fpDUMP, "Atoms\n");
  for(n = 1; n <= nAtom; n ++)
   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]);


  fprintf(fpDUMP, "Bonds\n");
  for(n=1; n<=nBond; n++)
  fprintf(fpDUMP,  "%d %d %d %d %0.2lf %0.16lf\n", BondID[n], BondType[n], atom1[n], atom2[n], kb[n], ro[n]);

  fclose(fpDUMP);
}
```

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.

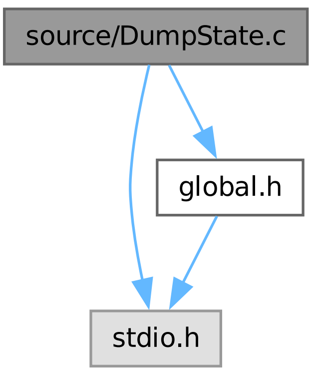
```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */


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

void DumpRestart() {
```

```
 char DUMP[256];
 FILE *fpDUMP;
 sprintf(DUMP, "%s.Restart", prefix);
 fpDUMP = fopen(DUMP, "w");
 if(fpDUMP == NULL) {
  fprintf(stderr, "Error opening file %s for writing\n", DUMP);
  return;
 }

  fprintf(fpDUMP, "timeNow %lf\n", timeNow);
  fprintf(fpDUMP, "nAtom %d\n",  nAtom);
  fprintf(fpDUMP, "nBond %d\n", nBond);
  fprintf(fpDUMP, "nAtomType %d\n", nAtomType);
  fprintf(fpDUMP, "nBondType %d\n", nBondType);
  fprintf(fpDUMP, "region[1] %0.14lf\n", region[1]);
  fprintf(fpDUMP, "region[2] %0.14lf\n", region[2]);

  int n;
  fprintf(fpDUMP, "Atoms\n");
  for(n = 1; n <= nAtom; n ++)
   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]);


  fprintf(fpDUMP, "Bonds\n");
  for(n=1; n<=nBond; n++)
  fprintf(fpDUMP,  "%d %d %d %d %0.2lf %0.16lf\n", BondID[n], BondType[n], atom1[n], atom2[n], kb[n], ro[n]);

  fclose(fpDUMP);
}

```

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 ( )
```

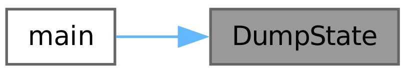
Definition at line 25 of file DumpState.c.

```
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\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.

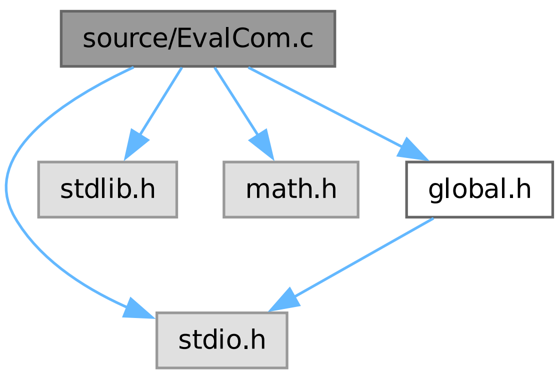
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your 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
```

```
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */


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

void DumpState() {
 char DUMP[256];
 FILE *fpDUMP;
 sprintf(DUMP, "%s.STATE", prefix);
 fpDUMP = fopen(DUMP, "w");
 if(fpDUMP == NULL) {
  fprintf(stderr, "Error opening file %s for writing\n", DUMP);
  return;
 }

  fprintf(fpDUMP, "ITEM: TIMESTEP\n");
  fprintf(fpDUMP, "%lf\n",timeNow);
  fprintf(fpDUMP, "ITEM: NUMBER OF ATOMS\n");
  fprintf(fpDUMP, "%d\n",nAtom);
  fprintf(fpDUMP, "ITEM: BOX BOUNDS pp pp pp\n");
  fprintf(fpDUMP, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
  fprintf(fpDUMP, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
  fprintf(fpDUMP, "%lf %lf zlo zhi\n",  -0.1, 0.1);
  fprintf(fpDUMP, "ITEM: ATOMS id mol type radius x y vx vy fx fy\n");
  int n;
  for (n = 1; n <= nAtom; n++) {
   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\n",
   atomID[n], molID[n], atomType[n], atomRadius[n], rx[n], ry[n], vx[n], vy[n], ax[n], ay[n]);
  }
  fclose(fpDUMP);
}

```

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 ( )
```

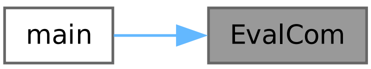
Definition at line 27 of file EvalCom.c.

```
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.

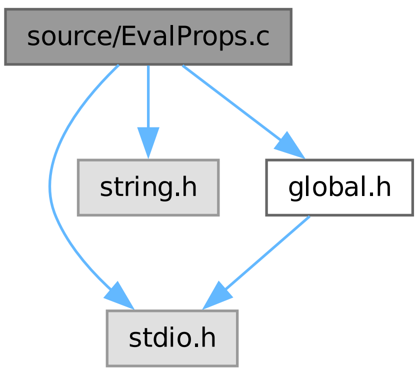
```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina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

```
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */


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

void EvalCom(){
 int n;
 ComX = 0.0; ComY = 0.0; ComXRatio = 0.0; ComYRatio = 0.0;
 TotalMass = 0.0;

 for(n=1; n<=nAtom; n++){
 if(molID[n] == 2){
  ComX += atomMass[n] * rxUnwrap[n];
  ComY += atomMass[n] * ryUnwrap[n];
  TotalMass += atomMass[n];
  } }

  ComX = ComX/TotalMass;
  ComY = ComY/TotalMass;

  if(timeNow == 0.0){
  ComX0 = ComX; ComY0 = ComY;
  }
  ComXRatio = ComX/ComX0;   ComYRatio = ComY/ComY0;
 }



```

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 ( )
```

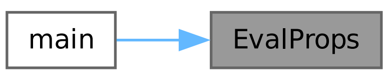
Definition at line 26 of file EvalProps.c.

```
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.

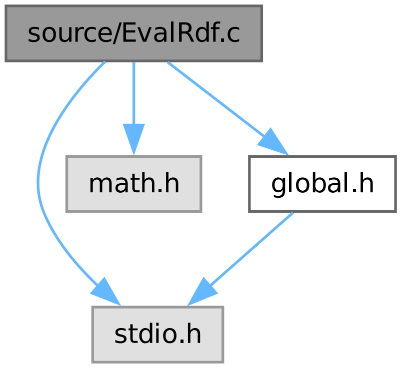
```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */


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

void EvalProps() {
 real v, vv;
 virSum = 0.0;
 vSumX = 0.0; vSumY = 0.0; vSum = 0.0;
 vvSum = 0.;
 int n;

 for (n = 1; n <= nAtom; n++) {
  vv = 0.;
  // Initialize v with a default value to avoid "uninitialized" warning.
  v = 0.0;
  // X direction velocity
  if (strcmp(solver, "Verlet") == 0) {
    v = vx[n];
  } else if (strcmp(solver, "LeapFrog") == 0) {
    v = vx[n] - 0.5 * deltaT * ax[n];
  }
   vSum += v;
   vv += Sqr(v);
   vSumX += v;
   // Y direction velocity
   if (strcmp(solver, "Verlet") == 0) {
   v = vy[n];
   } else if (strcmp(solver, "LeapFrog") == 0) {
   v = vy[n] - 0.5 * deltaT * ay[n];
   }
   vSum += v;
   vSumY += v;
   vv += Sqr(v);
   vvSum += vv;
  }

  kinEnergy = 0.5 * vvSum / nAtom ;
  uSumPairPerAtom = uSumPair / nAtom ;
  BondEnergyPerAtom = TotalBondEnergy / (0.5*nAtom); //Factor of 0.5 since each atom has one half the bond energy
  potEnergy = uSumPairPerAtom +  BondEnergyPerAtom ;
  totEnergy = kinEnergy + potEnergy;
  virSumxx = virSumPairxx  + virSumBondxx ;
  virSumyy = virSumPairyy  + virSumBondyy ;
  virSumxy = virSumPairxy  + virSumBondxy ;
  virSum = virSumPair + virSumBond;
  pressure = density * (vvSum + virSum) / (nAtom * NDIM);

}
```

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
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){
```

```
    normFac = region[1]*region[2] / (M_PI*Sqr(deltaR)*nAtom*nAtom*countRdf );
    for(n = 1 ; n <= sizeHistRdf ; n ++)
      histRdf[n] *= normFac/(n-0.5);
    // PRINT THE RADIAL DISTRIBUTION DATA ON TO DISK FILE
    real rBin;
    int n;
    fprintf(fprdf,"rdf @ timeNow %lf\n", timeNow);
    for(n = 1 ; n <= sizeHistRdf ; n ++){
      rBin = (n - 0.5)*rangeRdf/sizeHistRdf;
      fprintf(fprdf, "%lf %lf\n", rBin, histRdf[n]);
    }
  }

}
```

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.

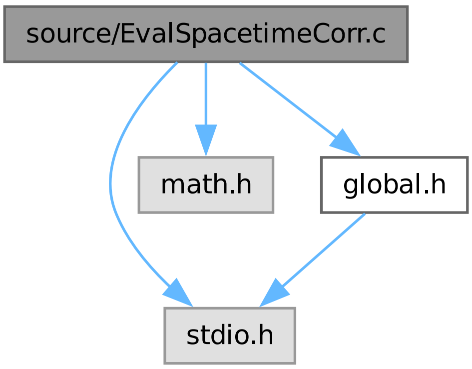
```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */


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

void EvalRdf(){
  real dr[NDIM+1], deltaR, normFac, rr, rrRange;
  int j1, j2, n;
  countRdf ++;
  if(countRdf == 1){
    for(n = 1 ; n <= sizeHistRdf ; n ++)
      histRdf[n] = 0.;
  }
  rrRange = Sqr(rangeRdf);
  deltaR = rangeRdf / sizeHistRdf;
  for(j1 = 1 ; j1 <= nAtom - 1 ; j1 ++){
    for(j2 = j1 + 1 ; j2 <= nAtom ; j2 ++){

      dr[1] = rx[j1] - rx[j2];
      if(fabs(dr[1]) > regionH[1])
    dr[1] -= SignR(region[1], dr[1]);

      dr[2] = ry[j1] - ry[j2];
      if(fabs(dr[2]) > regionH[2])
    dr[2] -= SignR(region[2], dr[2]);

      rr = Sqr(dr[1]) + Sqr(dr[2]);

      if(rr < rrRange){
    n = (int)(sqrt(rr)/deltaR) + 1;
    histRdf[n] ++;
      }
    }
  }

  if(countRdf == limitRdf){
    normFac = region[1]*region[2] / (M_PI*Sqr(deltaR)*nAtom*nAtom*countRdf );
    for(n = 1 ; n <= sizeHistRdf ; n ++)
      histRdf[n] *= normFac/(n-0.5);
```

```
    // PRINT THE RADIAL DISTRIBUTION DATA ON TO DISK FILE
    real rBin;
    int n;
    fprintf(fprdf,"rdf @ timeNow %lf\n", timeNow);
    for(n = 1 ; n <= sizeHistRdf ; n ++){
      rBin = (n - 0.5)*rangeRdf/sizeHistRdf;
      fprintf(fprdf, "%lf %lf\n", rBin, histRdf[n]);
    }
  }

}

```

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;
```

```
  for (j = 1; j <= 2*nFunCorr; j++)
    cfVal[j] = 0.;

  for (n = 1; n <= nAtom; n++){
    j = 1;
    COSA = cos(kMin*rx[n]);
    SINA = sin(kMin*rx[n]);
    for (m = 1; m <= nFunCorr; m++){
      if(m == 1){
    cosV = cos(deltaK*rx[n]);
    sinV = sin(deltaK*rx[n]);
    cosV0 = cosV;
      }else if(m == 2){
    cosV1 = cosV;
    sinV1 = sinV;
    cosV = 2.*cosV0*cosV1-1;
    sinV = 2.*cosV0*sinV1;
      }else{
    cosV2 = cosV1;
    sinV2 = sinV1;
    cosV1 = cosV;
    sinV1 = sinV;
    cosV = 2.*cosV0*cosV1-cosV2;
    sinV = 2.*cosV0*sinV1-sinV2;
      }
      COSV = COSA*cosV - SINA*sinV;
      SINV = SINA*cosV + COSA*sinV;
      cfVal[j] += COSV;
      cfVal[j+1] += SINV;
      j += 2;
    }
  }

  for (nb = 1; nb <= nBuffCorr; nb++){
    indexCorr[nb] += 1;
    if (indexCorr[nb] <= 0) continue;
    ni = nFunCorr * (indexCorr[nb] - 1);
    if (indexCorr[nb] == 1){
      for (j = 1; j <= 2*nFunCorr; j++)
    cfOrg[nb][j] = cfVal[j];
    }

    for (j = 1; j <= nFunCorr; j++)
      spacetimeCorr[nb][ni + j] = 0.;

    j = 1;
    for (m = 1; m <= nFunCorr; m++){
      nv = m + ni;
      spacetimeCorr[nb][nv] += cfVal[j] * cfOrg[nb][j] + cfVal[j + 1] * cfOrg[nb][j + 1];
      j += 2;
    }

  }

  // ACCUMULATE SPACETIME CORRELATIONS
  for (nb = 1; nb <= nBuffCorr; nb++){
   if (indexCorr[nb] == nValCorr){
     for (j = 1; j <= nFunCorr*nValCorr; j++)
       spacetimeCorrAv[j] += spacetimeCorr[nb][j];
     indexCorr[nb] = 0.;
     countCorrAv ++;
     if (countCorrAv == limitCorrAv){
       for (j = 1; j <= nFunCorr*nValCorr; j++)
     spacetimeCorrAv[j] /= (nAtom*limitCorrAv);
       fprintf(fpdnsty,"NDIM %d\n", NDIM);
       fprintf(fpdnsty,"nAtom %d\n", nAtom);
       fprintf(fpdnsty,"region %lf\n", region[1]);
       fprintf(fpdnsty,"nFunCorr %d\n", nFunCorr);
       fprintf(fpdnsty,"limitCorrAv %d\n", limitCorrAv);
       fprintf(fpdnsty,"stepCorr %d\n", stepCorr);
       fprintf(fpdnsty,"nValCorr %d\n", nValCorr);
       fprintf(fpdnsty,"deltaT %lf\n", deltaT);
       real tVal;
       for (n = 1; n <= nValCorr; n++){
     tVal = (n-1)*stepCorr*deltaT;
     fprintf (fpdnsty, "%e\t", tVal);
     int nn = nFunCorr*(n-1);
     for (j = 1; j <= nFunCorr; j ++)
           fprintf (fpdnsty, "%e\t", spacetimeCorrAv[nn + j]);
         fprintf (fpdnsty, "\n");
       }

       countCorrAv = 0.;
       for (j = 1; j <= nFunCorr*nValCorr; j++)
     spacetimeCorrAv[j] = 0.;
     }
```

```
    }
  }
}
```

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.

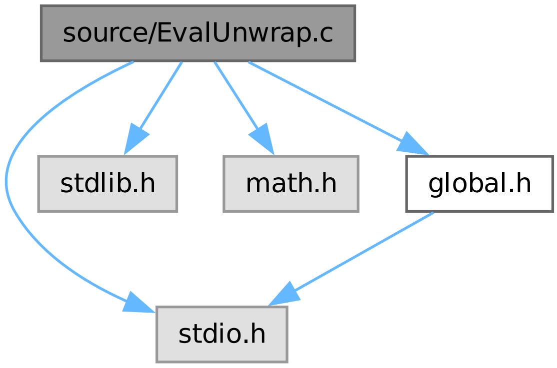
```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */


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

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

  for (j = 1; j <= 2*nFunCorr; j++)
    cfVal[j] = 0.;

  for (n = 1; n <= nAtom; n++){
    j = 1;
    COSA = cos(kMin*rx[n]);
    SINA = sin(kMin*rx[n]);
    for (m = 1; m <= nFunCorr; m++){
      if(m == 1){
    cosV = cos(deltaK*rx[n]);
    sinV = sin(deltaK*rx[n]);
    cosV0 = cosV;
      }else if(m == 2){
    cosV1 = cosV;
    sinV1 = sinV;
    cosV = 2.*cosV0*cosV1-1;
    sinV = 2.*cosV0*sinV1;
      }else{
    cosV2 = cosV1;
    sinV2 = sinV1;
    cosV1 = cosV;
    sinV1 = sinV;
    cosV = 2.*cosV0*cosV1-cosV2;
    sinV = 2.*cosV0*sinV1-sinV2;
      }
      COSV = COSA*cosV - SINA*sinV;
      SINV = SINA*cosV + COSA*sinV;
      cfVal[j] += COSV;
      cfVal[j+1] += SINV;
      j += 2;
    }
  }

  for (nb = 1; nb <= nBuffCorr; nb++){
    indexCorr[nb] += 1;
    if (indexCorr[nb] <= 0) continue;
    ni = nFunCorr * (indexCorr[nb] - 1);
```

```
    if (indexCorr[nb] == 1){
      for (j = 1; j <= 2*nFunCorr; j++)
    cfOrg[nb][j] = cfVal[j];
    }

    for (j = 1; j <= nFunCorr; j++)
      spacetimeCorr[nb][ni + j] = 0.;

    j = 1;
    for (m = 1; m <= nFunCorr; m++){
      nv = m + ni;
      spacetimeCorr[nb][nv] += cfVal[j] * cfOrg[nb][j] + cfVal[j + 1] * cfOrg[nb][j + 1];
      j += 2;
    }

  }

  // ACCUMULATE SPACETIME CORRELATIONS
  for (nb = 1; nb <= nBuffCorr; nb++){
   if (indexCorr[nb] == nValCorr){
     for (j = 1; j <= nFunCorr*nValCorr; j++)
       spacetimeCorrAv[j] += spacetimeCorr[nb][j];
     indexCorr[nb] = 0.;
     countCorrAv ++;
     if (countCorrAv == limitCorrAv){
       for (j = 1; j <= nFunCorr*nValCorr; j++)
     spacetimeCorrAv[j] /= (nAtom*limitCorrAv);
       fprintf(fpdnsty,"NDIM %d\n", NDIM);
       fprintf(fpdnsty,"nAtom %d\n", nAtom);
       fprintf(fpdnsty,"region %lf\n", region[1]);
       fprintf(fpdnsty,"nFunCorr %d\n", nFunCorr);
       fprintf(fpdnsty,"limitCorrAv %d\n", limitCorrAv);
       fprintf(fpdnsty,"stepCorr %d\n", stepCorr);
       fprintf(fpdnsty,"nValCorr %d\n", nValCorr);
       fprintf(fpdnsty,"deltaT %lf\n", deltaT);
       real tVal;
       for (n = 1; n <= nValCorr; n++){
     tVal = (n-1)*stepCorr*deltaT;
     fprintf (fpdnsty, "%e\t", tVal);
     int nn = nFunCorr*(n-1);
     for (j = 1; j <= nFunCorr; j ++)
           fprintf (fpdnsty, "%e\t", spacetimeCorrAv[nn + j]);
         fprintf (fpdnsty, "\n");
       }

       countCorrAv = 0.;
       for (j = 1; j <= nFunCorr*nValCorr; j++)
     spacetimeCorrAv[j] = 0.;
     }
   }
  }
}
```

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 ( )
```

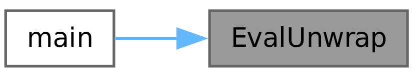
Definition at line 27 of file EvalUnwrap.c.

```
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

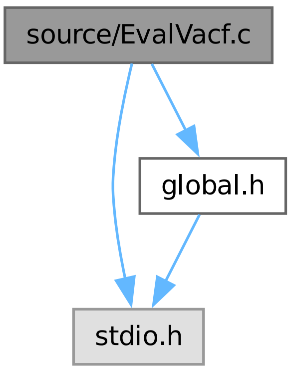
Go to the documentation of this file.

```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */


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

void EvalUnwrap() {
 int n;
 for (n = 1; n <= nAtom; n++) {
  rxUnwrap[n] = rx[n] + ImageX[n] * region[1];
  ryUnwrap[n] = ry[n] + ImageY[n] * region[2];
  }
}
```

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 ( )
```

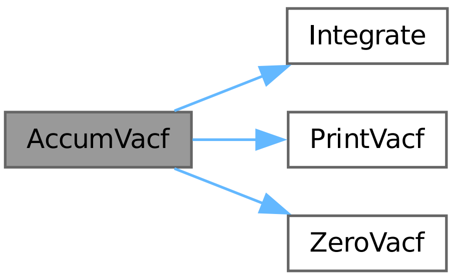
Definition at line 27 of file AccumVacf.c.

```
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 } } } }
```

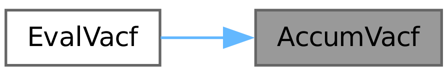
References countAcfAv, deltaT, indexAcf, Integrate(), kinEnergy, limitAcfAv, nBuffAcf, nValAcf, PrintVacf(), region, stepAcf, viscAcf, viscAcfAv, viscAcfInt, and ZeroVacf().

Referenced by EvalVacf().

Here is the call graph for this function:



Here is the caller graph for this function:



3.54.1.2 EvalVacf()

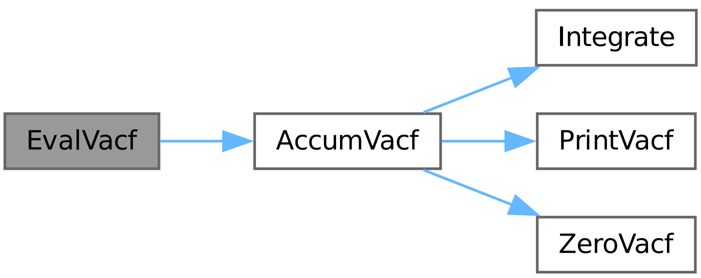
```
void EvalVacf ( )
```

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.

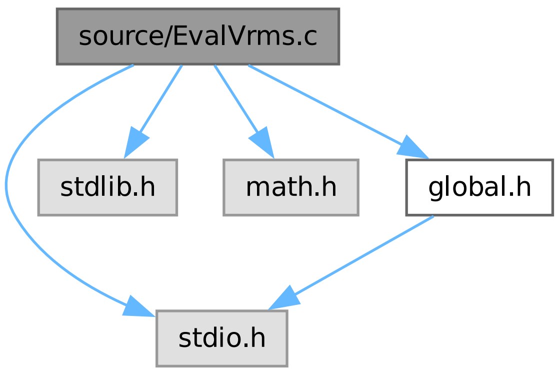
```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK
```

```
 */


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

void AccumVacf();
void EvalVacf(){
  int n, nb, ni;
  double viscVec = 0.;
  double v[3];
  for(n = 1 ; n <= nAtom ; n ++){
    v[1] = vx[n] - 0.5*ax[n]*deltaT;
    v[2] = vy[n] - 0.5*ay[n]*deltaT;
    viscVec += v[1]*v[2];
  }
  viscVec += rfAtom;
  for(nb = 1 ; nb <= nBuffAcf ; nb ++){
    indexAcf[nb] ++;
    if(indexAcf[nb] <= 0)continue;
    if(indexAcf[nb] == 1){
      viscAcfOrg[nb] = viscVec;
    }
    ni = indexAcf[nb];
    viscAcf[nb][ni] = viscAcfOrg[nb]*viscVec;
  }
  AccumVacf();
}
```

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 ( )
```

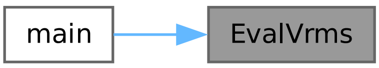
Definition at line 27 of file EvalVrms.c.

```
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.

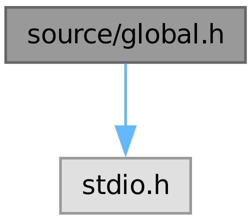
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your 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;
```

```
VMeanSqr = 0.0;
VRootMeanSqr = 0.0;

for(n = 1 ; n <= nAtom ; n ++){
VSqr += Sqr(vx[n]) + Sqr(vy[n]);
}
VMeanSqr = VSqr/nAtom;
VRootMeanSqr = sqrt(VMeanSqr);
}
```

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 DeltaVXij

```
double DeltaVXij
```

Definition at line 70 of file global.h.

Referenced by ComputeBondForce(), and ComputePairForce().

3.58.3.33 DeltaVYij

```
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 DeltaXij

```
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(), DumpState(), and main().

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 strech

```
double strech [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,
      sKinEnergy,
00022 sTotEnergy, sPressure, ssPotEnergy, ssKinEnergy, ssTotEnergy, ssPressure;
00023
00024 EXTERN int     initUcell[NDIM+1], moreCycles, nAtom, stepAvg, stepCount, stepEquil, stepLimit,
      stepTraj, stepDump;
00025
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;
```

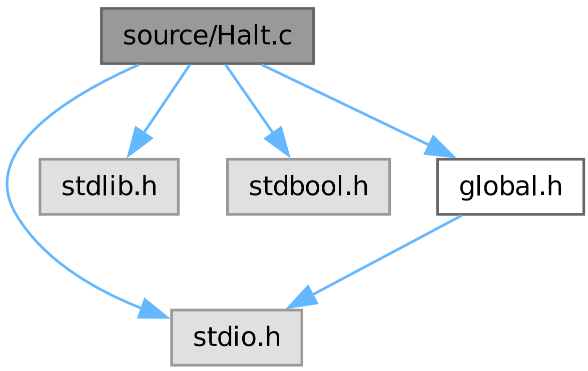
```
EXTERN double  strech;
//For dumping the pair interaction data
EXTERN int     dumpPairFlag;
EXTERN int     nPairTotal, nPairActive;
EXTERN int     *PairID, *Pairatom1, *Pairatom2;
EXTERN double  *PairXij, *PairYij;
EXTERN char   solver[128];
EXTERN char   xBoundary[10], yBoundary[10];
EXTERN char     thermo;
//For damping as in PRL, 130, 178203 (2023)
EXTERN double *DeltaXijOld, *DeltaYijOld;
EXTERN double DeltaXijNew, DeltaYijNew;
EXTERN double DeltaXij, DeltaYij, DeltaVXij, DeltaVYij;
EXTERN double **DeltaXijOldPair, **DeltaYijOldPair;
//For molecule-ID as per LAMMPS, helpful!
EXTERN int    *molID;
EXTERN int    **isBonded;
//Following three for MPI only
EXTERN int     *cellList, cells[NDIM+1];
EXTERN int     rank, size, master;
EXTERN double  *fax, *fay, fuSum, fvirSum, frfAtom;
//For thermodynamic properties
EXTERN double  uSumPair, uSumPairPerAtom, virSumPair, virSumPairxx, virSumPairyy, virSumPairxy;
EXTERN double  virSumBond, virSumBondxx, virSumBondyy, virSumBondxy;
EXTERN double  virSumxx, virSumyy, virSumxy;
EXTERN int     freezeAtomType;
// Spacetime Correlations
EXTERN double  **cfOrg, **spacetimeCorr, *cfVal, *spacetimeCorrAv;
EXTERN int     *indexCorr, countCorrAv, limitCorrAv, nBuffCorr, nFunCorr, nValCorr, stepCorr;
// Viscosity
EXTERN double  rfAtom, frfAtom;
EXTERN double  *indexAcf, **viscAcf, *viscAcfOrg, *viscAcfAv, viscAcfInt;
EXTERN int      nValAcf, nBuffAcf, stepAcf, countAcfAv, limitAcfAv;
// Radial distribution function
EXTERN double  *histRdf, rangeRdf;
EXTERN int     countRdf, limitRdf, sizeHistRdf, stepRdf;
// Output files prefixes
EXTERN char      *prefix;
EXTERN char    result[250];
EXTERN FILE    *fpresult;
EXTERN char    xyz[256];
EXTERN FILE    *fpxyz;
EXTERN char    bond[256];
EXTERN FILE    *fpbond;
EXTERN char    dump[256];
EXTERN FILE    *fpdump;
EXTERN char    dnsty[256];
EXTERN FILE    *fpdnsty;
EXTERN char    visc[256];
EXTERN FILE    *fpvisc;
EXTERN char    rdf[256];
EXTERN FILE    *fprdf;
EXTERN char    vrms[256];
EXTERN FILE    *fpvrms;
EXTERN char    stress[256];
EXTERN FILE    *fpstress;
EXTERN char    momentum[256];
EXTERN FILE    *fpmomentum;
EXTERN char    com[256];
EXTERN FILE    *fpcom;
EXTERN char    pair[256];
EXTERN FILE    *fppair;
```

```
#endif // GLOBALEXTERN_H
```

3.60 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)

3.60.1 Function Documentation

3.60.1.1 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:

3.61 Halt.c

Go to the documentation of this file.

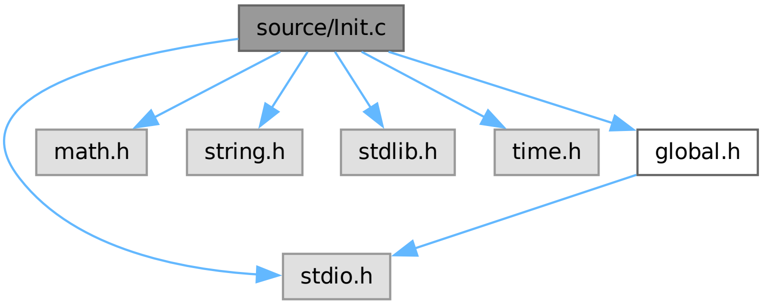
```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */


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

bool HaltConditionCheck(double value, int stepCount) {

  if(value <= HaltCondition && value != 0) {
  fprintf(fpresult, "Halt condition met at step = %d with Vrms = %.10f\n", stepCount, value);
  return true;       // Signal that the halt condition is met
  }
 return false; // Halt condition not met
}
```

3.62 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 ()

3.62.1 Function Documentation

3.62.1.1 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);
```

```
fscanf(fp, "%s %d",  dummy, &sizeHistRdf);
fscanf(fp, "%s %d",  dummy, &stepRdf);

fclose(fp);
FILE *fpSTATE;
if((fpSTATE = fopen(inputConfig,"r"))==NULL){
printf("Error occurred: Could not open STATE file\n Exiting now..\n");
exit(0);
}

fscanf(fpSTATE, "%s %lf", dummy, &timeNow);
fscanf(fpSTATE, "%s %d",  dummy, &nAtom);
fscanf(fpSTATE, "%s %d",  dummy, &nBond);
fscanf(fpSTATE, "%s %d",  dummy, &nAtomType);
fscanf(fpSTATE, "%s %d",  dummy, &nBondType);
fscanf(fpSTATE, "%s %lf", dummy, &region[1]);
fscanf(fpSTATE, "%s %lf", dummy, &region[2]);

region[2] *= 1.5; //Remove this when put on GitHub

density = nAtom/(region[1]*region[2]);
cells[1] = region[1] / rCut;
cells[2] = region[2] / rCut;
cellList = (int*)malloc((nAtom + cells[1] * cells[2] + 1) * sizeof(int));
regionH[1] = 0.5*region[1];
regionH[2] = 0.5*region[2];

//strain information
strainRate = strain/deltaT;
shearDisplacement = strain * region[2];
shearVelocity = strainRate * region[2];
int n;

rx = (double*)malloc((nAtom + 1) * sizeof(double));
ry = (double*)malloc((nAtom + 1) * sizeof(double));
vx = (double*)malloc((nAtom + 1) * sizeof(double));
vy = (double*)malloc((nAtom + 1) * sizeof(double));
ax = (double*)malloc((nAtom + 1) * sizeof(double));
ay = (double*)malloc((nAtom + 1) * sizeof(double));
fax = (double*)malloc((nAtom + 1) * sizeof(double));
fay = (double*)malloc((nAtom + 1) * sizeof(double));
atomID = (int*)malloc((nAtom+1) * sizeof(int));
atomType = (int*)malloc((nAtom+1) * sizeof(int));
atomRadius = (double*)malloc((nAtom + 1) * sizeof(double));
atomMass = (double*)malloc((nAtom + 1) * sizeof(double));
speed = (double*)malloc((nAtom + 1) * sizeof(double));
atom1 = (int*)malloc((nBond+1)*sizeof(int));
atom2 = (int*)malloc((nBond+1)*sizeof(int));
BondID = (int*)malloc((nBond+1)*sizeof(int));
BondType = (int*)malloc((nBond+1)*sizeof(int));
kb = (double*)malloc((nBond+1)*sizeof(double));
ro = (double*)malloc((nBond+1)*sizeof(double));
BondEnergy = (double*)malloc((nBond+1)*sizeof(double));
BondLength =(double*)malloc((nBond+1)*sizeof(double));
discDragx = (double*)malloc((nAtom + 1) * sizeof(double));
discDragy = (double*)malloc((nAtom + 1) * sizeof(double));
nodeDragx = (double*)malloc((nAtom + 1) * sizeof(double));
nodeDragy = (double*)malloc((nAtom + 1) * sizeof(double));
ImageX = (int*)malloc((nAtom+1) * sizeof(int));
ImageY = (int*)malloc((nAtom+1) * sizeof(int));
rxUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
ryUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
DeltaXijOld = (double*)malloc((nBond+1)*sizeof(double));
DeltaYijOld = (double*)malloc((nBond+1)*sizeof(double));
DeltaXijOldPair = (double**)malloc((nAtom+1) * sizeof(double*));
DeltaYijOldPair = (double**)malloc((nAtom+1) * sizeof(double*));
for(int n = 0; n <= nAtom; n++) {
 DeltaXijOldPair[n] = (double*)malloc((nAtom+1) * sizeof(double));
 DeltaYijOldPair[n] = (double*)malloc((nAtom+1) * sizeof(double));
}
molID = (int*)malloc((nAtom+1) * sizeof(int));

for(n = 1; n <= nAtom; n ++){
 atomMass[n] = 1.0;
}

fscanf(fpSTATE, "%s\n", dummy);
for(n = 1; n <= nAtom; n ++)
 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]);


fscanf(fpSTATE, "%s\n", dummy);
for(n=1; n<=nBond; n++)
 fscanf(fpSTATE, "%d %d %d %d %lf %lf\n", &BondID[n], &BondType[n], &atom1[n], &atom2[n], &kb[n],
&ro[n]);

```

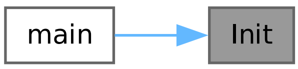
```
  fclose(fpSTATE);

 //2D-List of bonded atoms. This is used to remove pair interaction
 //calculation for the bonded atoms
   isBonded = (int**)malloc((nAtom + 1) * sizeof(int*));
  for (int i = 0; i <= nAtom; i++) {
    isBonded[i] = (int*)malloc((nAtom + 1) * sizeof(int));
    for (int j = 0; j <= nAtom; j++) {
        isBonded[i][j] = 0;
    }
  }

  for (n = 1; n <= nBond; n++) {
    int i = atom1[n];
    int j = atom2[n];
    isBonded[i][j] = 1;
    isBonded[j][i] = 1; // symmetric
}

 //For thermostate, update in final version
thermo = 'C';


 // List the interface atoms
 nAtomInterface = 0;
 nAtomBlock = 0;
 nDiscInterface = 0;
 double InterfaceWidth, bigDiameter;
 bigDiameter = 2.8;
 InterfaceWidth = 5.0 * bigDiameter;

 for(n = 1; n <= nAtom; n++){
  if(fabs(ry[n]) < InterfaceWidth){
  nAtomInterface++;
  }
  if(molID[n] == 2){
  nAtomBlock++;
  }
  if(atomRadius[n] != 0.0){
  nDiscInterface++;
  } }

   atomIDInterface =  (int*)malloc((nAtomInterface+1)*sizeof(int));

  int m;
  m = 1;
  for(n=1; n<=nAtom; n++){
   if(fabs(ry[n]) < InterfaceWidth){
   atomIDInterface[m] = atomID[n];
   m++;
   } }

  nPairTotal = 0.5 * nAtomInterface * (nAtomInterface-1);
  PairID = (int*)malloc((nPairTotal+1) * sizeof(int));
  Pairatom1 = (int*)malloc((nPairTotal+1) * sizeof(int));
  Pairatom2 = (int*)malloc((nPairTotal+1) * sizeof(int));
  PairXij = (double*)malloc((nPairTotal+1) * sizeof(double));
  PairYij = (double*)malloc((nPairTotal+1) * sizeof(double));

    fprintf(fpresult, "------------------------------------\n");
    fprintf(fpresult, "-------PARAMETERS-----------\n");
    fprintf(fpresult, "------------------------------------\n");
    fprintf(fpresult, "nAtom\t\t\t%d\n",  nAtom);
    fprintf(fpresult, "nBond\t\t\t%d\n",  nBond);
    fprintf(fpresult, "nAtomBlock\t\t%d\n",  nAtomBlock);
    fprintf(fpresult, "nAtomInterface\t%d\n",  nAtomInterface);
    fprintf(fpresult, "nDiscInterface\t\t%d\n",  nDiscInterface);
    fprintf(fpresult, "gamman\t\t\t%0.6g\n", gamman);
    fprintf(fpresult, "strain\t\t\t%0.6g\n", strain);
    fprintf(fpresult, "strainRate\t\t%0.6g\n", strainRate);
    fprintf(fpresult, "FyBylx\t\t\t%0.6g\n", FyBylx);
    fprintf(fpresult, "fxByfy\t\t\t%0.6g\n", fxByfy);
    fprintf(fpresult, "DeltaY\t\t\t%0.6g\n", DeltaY);
    fprintf(fpresult, "DeltaX\t\t\t%0.6g\n", DeltaX);
    fprintf(fpresult, "HaltCondition\t\t%0.6g\n", HaltCondition);
    fprintf(fpresult, "kappa\t\t\t%g\n", kappa);
    fprintf(fpresult, "density\t\t\t%g\n", density);
    fprintf(fpresult, "rCut\t\t\t\t%g\n", rCut);
    fprintf(fpresult, "deltaT\t\t\t%g\n", deltaT);
    fprintf(fpresult, "stepEquil\t\t\t%d\n",  stepEquil);
    fprintf(fpresult, "stepLimit\t\t\t%d\n",  stepLimit);
    fprintf(fpresult, "region[1]\t\t\t%0.16lf\n", region[1]);
    fprintf(fpresult, "region[2]\t\t\t%0.16lf\n", region[2]);
    fprintf(fpresult, "cells[1]\t\t\t%d\n",  cells[1]);
    fprintf(fpresult, "cells[2]\t\t\t%d\n",  cells[2]);
    fprintf(fpresult, "solver\t\t\t%s\n",  solver);
    fprintf(fpresult, "boundary\t\t\t%s %s\n", xBoundary, yBoundary);
```

```
    fprintf(fpresult, "DampFlag\t\t%d\n",  DampFlag);


    fprintf(fpresult, "------------------------------------\n");
    fprintf(fpresult, "#TimeNow TotalMomentum PotEngyPerAtom KinEngyPerAtom TotEngyPerAtom PairEnergyPerAtom BondEnergyPerAtom  Press VirialSum\n");
    fprintf(fpvrms, "#timeNow\tVrms \n");
    fprintf(fpcom, "#timeNow\tComX\tComY\n");

/* //Uncomment the following as per your acquirement
    fprintf(fpstress, "strain           %lf\n", strain);
    fprintf(fpstress, "region[1]        %lf\n", region[1]);
    fprintf(fpstress, "region[2]        %lf\n", region[2]);
    fprintf(fpstress, "#timeNow virSumxx virSumyy virSumxy pressure\n");
    fprintf(fpmomentum, "#timeNow Px Py\n");
*/

   if((strcmp(xBoundary, "p") != 0 && strcmp(xBoundary, "r") != 0) ||
    (strcmp(yBoundary, "p") != 0 && strcmp(yBoundary, "r") != 0)) {
    fprintf(fpresult, "Error: Invalid boundary value detected: '%s %s'. Only 'p' or 'r' are allowed.\n", xBoundary, yBoundary);
    exit(EXIT_FAILURE); // Exit with failure status
  }

}
```

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:



3.63 Init.c

Go to the documentation of this file.

```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

```

```
 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */


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

void Init(){
  char dummy[128];
  char inputConfig[128];
  FILE *fp;
  fp = fopen("input-data","r");
  fscanf(fp, "%s %s", dummy, inputConfig);
  fscanf(fp, "%s %s", dummy, solver);
  fscanf(fp, "%s %s %s", dummy, xBoundary, yBoundary);
  fscanf(fp, "%s %d",  dummy, &DampFlag);
  fscanf(fp, "%s %d",  dummy, &freezeAtomType);
  fscanf(fp, "%s %lf", dummy, &rCut);
  fscanf(fp, "%s %lf", dummy, &gamman);
  fscanf(fp, "%s %lf", dummy, &kappa);
  fscanf(fp, "%s %lf", dummy, &deltaT);
  fscanf(fp, "%s %lf", dummy, &strain);
  fscanf(fp, "%s %lf", dummy, &FyBylx);
  fscanf(fp, "%s %lf", dummy, &fxByfy);
  fscanf(fp, "%s %lf", dummy, &DeltaY);
  fscanf(fp, "%s %lf", dummy, &DeltaX);
  fscanf(fp, "%s %lf", dummy, &HaltCondition);
  fscanf(fp, "%s %d",  dummy, &stepAvg);
  fscanf(fp, "%s %d",  dummy, &stepEquil);
  fscanf(fp, "%s %d",  dummy, &stepLimit);
  fscanf(fp, "%s %d",  dummy, &stepDump);
  fscanf(fp, "%s %d",  dummy, &stepTraj);
  fscanf(fp, "%s %d",  dummy, &limitCorrAv);
  fscanf(fp, "%s %d",  dummy, &nBuffCorr);
  fscanf(fp, "%s %d",  dummy, &nFunCorr);
  fscanf(fp, "%s %d",  dummy, &nValCorr);
  fscanf(fp, "%s %d",  dummy, &stepCorr);
  fscanf(fp, "%s %d",  dummy, &limitAcfAv);
  fscanf(fp, "%s %d",  dummy, &nBuffAcf);
  fscanf(fp, "%s %d",  dummy, &nValAcf);
  fscanf(fp, "%s %d",  dummy, &stepAcf);
  fscanf(fp, "%s %lf", dummy, &rangeRdf);
  fscanf(fp, "%s %d",  dummy, &limitRdf);
  fscanf(fp, "%s %d",  dummy, &sizeHistRdf);
  fscanf(fp, "%s %d",  dummy, &stepRdf);

  fclose(fp);
  FILE *fpSTATE;
  if((fpSTATE = fopen(inputConfig,"r"))==NULL){
  printf("Error occurred: Could not open STATE file\n Exiting now..\n");
  exit(0);
  }

  fscanf(fpSTATE, "%s %lf", dummy, &timeNow);
  fscanf(fpSTATE, "%s %d",  dummy, &nAtom);
  fscanf(fpSTATE, "%s %d",  dummy, &nBond);
  fscanf(fpSTATE, "%s %d",  dummy, &nAtomType);
  fscanf(fpSTATE, "%s %d",  dummy, &nBondType);
  fscanf(fpSTATE, "%s %lf", dummy, &region[1]);
  fscanf(fpSTATE, "%s %lf", dummy, &region[2]);

  region[2] *= 1.5; //Remove this when put on GitHub

  density = nAtom/(region[1]*region[2]);
  cells[1] = region[1] / rCut;
  cells[2] = region[2] / rCut;
  cellList = (int*)malloc((nAtom + cells[1] * cells[2] + 1) * sizeof(int));
  regionH[1] = 0.5*region[1];
  regionH[2] = 0.5*region[2];

  //strain information
  strainRate = strain/deltaT;
  shearDisplacement = strain * region[2];
  shearVelocity = strainRate * region[2];
  int n;

  rx = (double*)malloc((nAtom + 1) * sizeof(double));
  ry = (double*)malloc((nAtom + 1) * sizeof(double));
  vx = (double*)malloc((nAtom + 1) * sizeof(double));
  vy = (double*)malloc((nAtom + 1) * sizeof(double));
  ax = (double*)malloc((nAtom + 1) * sizeof(double));
  ay = (double*)malloc((nAtom + 1) * sizeof(double));
```

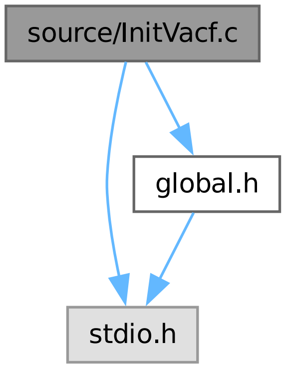
```
  fax = (double*)malloc((nAtom + 1) * sizeof(double));
  fay = (double*)malloc((nAtom + 1) * sizeof(double));
  atomID = (int*)malloc((nAtom+1) * sizeof(int));
  atomType = (int*)malloc((nAtom+1) * sizeof(int));
  atomRadius = (double*)malloc((nAtom + 1) * sizeof(double));
  atomMass = (double*)malloc((nAtom + 1) * sizeof(double));
  speed = (double*)malloc((nAtom + 1) * sizeof(double));
  atom1 = (int*)malloc((nBond+1)*sizeof(int));
  atom2 = (int*)malloc((nBond+1)*sizeof(int));
  BondID = (int*)malloc((nBond+1)*sizeof(int));
  BondType = (int*)malloc((nBond+1)*sizeof(int));
  kb = (double*)malloc((nBond+1)*sizeof(double));
  ro = (double*)malloc((nBond+1)*sizeof(double));
  BondEnergy = (double*)malloc((nBond+1)*sizeof(double));
  BondLength =(double*)malloc((nBond+1)*sizeof(double));
  discDragx = (double*)malloc((nAtom + 1) * sizeof(double));
  discDragy = (double*)malloc((nAtom + 1) * sizeof(double));
  nodeDragx = (double*)malloc((nAtom + 1) * sizeof(double));
  nodeDragy = (double*)malloc((nAtom + 1) * sizeof(double));
  ImageX = (int*)malloc((nAtom+1) * sizeof(int));
  ImageY = (int*)malloc((nAtom+1) * sizeof(int));
  rxUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
  ryUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
  DeltaXijOld = (double*)malloc((nBond+1)*sizeof(double));
  DeltaYijOld = (double*)malloc((nBond+1)*sizeof(double));
  DeltaXijOldPair = (double**)malloc((nAtom+1) * sizeof(double*));
  DeltaYijOldPair = (double**)malloc((nAtom+1) * sizeof(double*));
  for(int n = 0; n <= nAtom; n++) {
   DeltaXijOldPair[n] = (double*)malloc((nAtom+1) * sizeof(double));
   DeltaYijOldPair[n] = (double*)malloc((nAtom+1) * sizeof(double));
  }
  molID = (int*)malloc((nAtom+1) * sizeof(int));

  for(n = 1; n <= nAtom; n ++){
   atomMass[n] = 1.0;
  }

  fscanf(fpSTATE, "%s\n", dummy);
  for(n = 1; n <= nAtom; n ++)
   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]);


  fscanf(fpSTATE, "%s\n", dummy);
  for(n=1; n<=nBond; n++)
   fscanf(fpSTATE, "%d %d %d %d %lf %lf\n", &BondID[n], &BondType[n], &atom1[n], &atom2[n], &kb[n], &ro[n]);

  fclose(fpSTATE);

 //2D-List of bonded atoms. This is used to remove pair interaction
 //calculation for the bonded atoms
   isBonded = (int**)malloc((nAtom + 1) * sizeof(int*));
  for (int i = 0; i <= nAtom; i++) {
    isBonded[i] = (int*)malloc((nAtom + 1) * sizeof(int));
    for (int j = 0; j <= nAtom; j++) {
        isBonded[i][j] = 0;
    }
  }

  for (n = 1; n <= nBond; n++) {
    int i = atom1[n];
    int j = atom2[n];
    isBonded[i][j] = 1;
    isBonded[j][i] = 1; // symmetric
}

 //For thermostate, update in final version
thermo = 'C';


 // List the interface atoms
 nAtomInterface = 0;
 nAtomBlock = 0;
 nDiscInterface = 0;
 double InterfaceWidth, bigDiameter;
 bigDiameter = 2.8;
 InterfaceWidth = 5.0 * bigDiameter;

 for(n = 1; n <= nAtom; n++){
  if(fabs(ry[n]) < InterfaceWidth){
  nAtomInterface++;
  }
  if(molID[n] == 2){
  nAtomBlock++;
  }
  if(atomRadius[n] != 0.0){
```

```
  nDiscInterface++;
  } }

   atomIDInterface =  (int*)malloc((nAtomInterface+1)*sizeof(int));

  int m;
  m = 1;
  for(n=1; n<=nAtom; n++){
   if(fabs(ry[n]) < InterfaceWidth){
   atomIDInterface[m] = atomID[n];
   m++;
   } }

  nPairTotal = 0.5 * nAtomInterface * (nAtomInterface-1);
  PairID = (int*)malloc((nPairTotal+1) * sizeof(int));
  Pairatom1 = (int*)malloc((nPairTotal+1) * sizeof(int));
  Pairatom2 = (int*)malloc((nPairTotal+1) * sizeof(int));
  PairXij = (double*)malloc((nPairTotal+1) * sizeof(double));
  PairYij = (double*)malloc((nPairTotal+1) * sizeof(double));

    fprintf(fpresult, "------------------------------------\n");
    fprintf(fpresult, "-------PARAMETERS-----------\n");
    fprintf(fpresult, "------------------------------------\n");
    fprintf(fpresult, "nAtom\t\t\t%d\n",  nAtom);
    fprintf(fpresult, "nBond\t\t\t%d\n",  nBond);
    fprintf(fpresult, "nAtomBlock\t\t%d\n",  nAtomBlock);
    fprintf(fpresult, "nAtomInterface\t%d\n",  nAtomInterface);
    fprintf(fpresult, "nDiscInterface\t\t%d\n",  nDiscInterface);
    fprintf(fpresult, "gamman\t\t\t%0.6g\n", gamman);
    fprintf(fpresult, "strain\t\t\t%0.6g\n", strain);
    fprintf(fpresult, "strainRate\t\t%0.6g\n", strainRate);
    fprintf(fpresult, "FyBylx\t\t\t%0.6g\n", FyBylx);
    fprintf(fpresult, "fxByfy\t\t\t%0.6g\n", fxByfy);
    fprintf(fpresult, "DeltaY\t\t\t%0.6g\n", DeltaY);
    fprintf(fpresult, "DeltaX\t\t\t%0.6g\n", DeltaX);
    fprintf(fpresult, "HaltCondition\t\t%0.6g\n", HaltCondition);
    fprintf(fpresult, "kappa\t\t\t%g\n", kappa);
    fprintf(fpresult, "density\t\t\t%g\n", density);
    fprintf(fpresult, "rCut\t\t\t%g\n", rCut);
    fprintf(fpresult, "deltaT\t\t\t%g\n", deltaT);
    fprintf(fpresult, "stepEquil\t\t\t%d\n",  stepEquil);
    fprintf(fpresult, "stepLimit\t\t\t%d\n",  stepLimit);
    fprintf(fpresult, "region[1]\t\t\t%0.16lf\n", region[1]);
    fprintf(fpresult, "region[2]\t\t\t%0.16lf\n", region[2]);
    fprintf(fpresult, "cells[1]\t\t\t%d\n",  cells[1]);
    fprintf(fpresult, "cells[2]\t\t\t%d\n",  cells[2]);
    fprintf(fpresult, "solver\t\t\t%s\n",  solver);
    fprintf(fpresult, "boundary\t\t\t%s %s\n", xBoundary, yBoundary);
    fprintf(fpresult, "DampFlag\t\t\t%d\n",  DampFlag);


    fprintf(fpresult, "------------------------------------\n");
    fprintf(fpresult, "#TimeNow TotalMomentum PotEngyPerAtom KinEngyPerAtom TotEngyPerAtom
 PairEnergyPerAtom BondEnergyPerAtom  Press VirialSum\n");
    fprintf(fpvrms, "#timeNow\tVrms \n");
    fprintf(fpcom, "#timeNow\tComX\tComY\n");

/* //Uncomment the following as per your acquirement
    fprintf(fpstress, "strain           %lf\n", strain);
    fprintf(fpstress, "region[1]        %lf\n", region[1]);
    fprintf(fpstress, "region[2]        %lf\n", region[2]);
    fprintf(fpstress, "#timeNow virSumxx virSumyy virSumxy pressure\n");
    fprintf(fpmomentum, "#timeNow Px Py\n");
*/

   if((strcmp(xBoundary, "p") != 0 && strcmp(xBoundary, "r") != 0) ||
    (strcmp(yBoundary, "p") != 0 && strcmp(yBoundary, "r") != 0)) {
    fprintf(fpresult, "Error: Invalid boundary value detected: '%s %s'. Only 'p' or 'r' are
 allowed.\n", xBoundary, yBoundary);
    exit(EXIT_FAILURE); // Exit with failure status
  }

}
```

3.64 source/InitVacf.c File Reference

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

Include dependency graph for InitVacf.c:



Functions

- void ZeroVacf ()
- void InitVacf ()

3.64.1 Function Documentation

3.64.1.1 InitVacf()

```
void InitVacf ( )
```

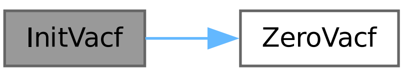
Definition at line 26 of file InitVacf.c.

```
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:

3.64.1.2 ZeroVacf()

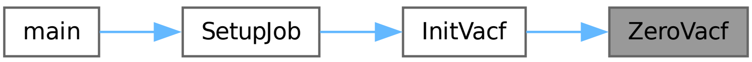
```
void ZeroVacf ( )
```

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:



3.65 InitVacf.c

Go to the documentation of this file.

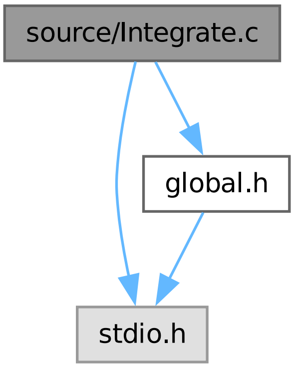
```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */


```

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

void ZeroVacf();
void InitVacf(){
  int nb;
  for(nb = 1 ; nb <= nBuffAcf ; nb ++)
    indexAcf[nb] = -(nb-1)*nValAcf/nBuffAcf;
  ZeroVacf();
}
```

3.66 source/Integrate.c File Reference

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

Include dependency graph for Integrate.c:



Functions

- double Integrate (double *f, int nf)

3.66.1 Function Documentation

3.66.1.1 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:

3.67 Integrate.c

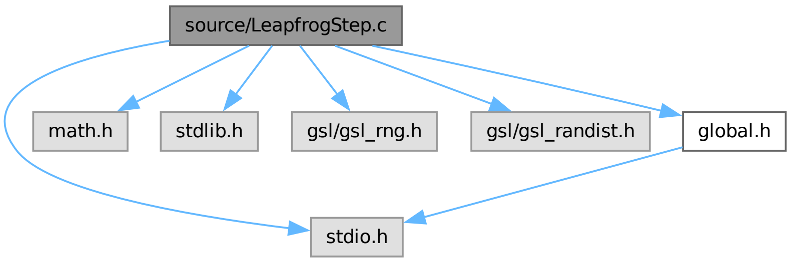
Go to the documentation of this file.

```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */


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

double Integrate(double *f, int nf){
 double s;
 int i;
 s = 0.5*(f[1] + f[nf]);
 for(i = 2 ; i <= nf - 1 ; i ++)
  s += f[i];
 return(s);
}

```

3.68 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)

3.68.1 Function Documentation

3.68.1.1 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 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
```

```
    int i,j;
    real dr[NDIM+1], r, rr, ri, rrCut;
    double vv;

    double uVal, AA, AASum;
    double TVal;

    double deno, VVSum;
    deno = 0.;
    VVSum = 0.;
    AASum = 0.;

  for(n=1;n<=nAtom; n++)
     TValSum[n] = 0.;

  rrCut = Sqr(rCut);

/*****Calculating Configarational temperature*****/
//Solving the equation of motion here
if(thermo == 'C'){
for(i = 1 ; i <= nAtom; i ++){
    for(j = i+1 ; j <= nAtom ; j ++){
      dr[1] = rx[i] - rx[j];
      if(fabs(dr[1]) > regionH[1])
    dr[1] -= SignR(region[1], dr[1]);

     dr[2] = ry[i] - ry[j];
      if(fabs(dr[2]) > regionH[2])
    dr[2] -= SignR(region[2], dr[2]);

     rr = Sqr(dr[1]) + Sqr(dr[2]);
     if(rr < rrCut ){
     r = sqrt(rr);
     ri = 1/r;
     uVal = ri*exp(-kappa*r);

     TVal = (1./rr + Sqr(kappa) + kappa/r)*uVal;
     TValSum[i] += TVal;
     TValSum[j] += TVal;
   } }
     AA = Sqr(ax[i]) + Sqr(ay[i]);
     AASum += AA;
     vv = Sqr(vx[i]) + Sqr(vy[i]);
     VVSum += vv;
     deno += TValSum[i];
}

     double gSumconfig, varSconfig, massSconfig;
     if(stepCount == 1) varSconfig = 0.;
     gSumconfig = 0.; massSconfig = 2.0;

     gSumconfig = (AASum/temperature - deno)/massSconfig;
     varSconfig += deltaT*gSumconfig;

      /*****Configarational Nose-Hoover thermostat*****/
   for (n = 1; n <= nAtom; n++){
      vx[n] += deltaT * ax[n];
      rx[n] += deltaT * (vx[n] + varSconfig * ax[n]);
      vy[n] += deltaT * ay[n];
      ry[n] += deltaT * (vy[n] + varSconfig * ay[n]);
    }
      /*****Kinetic Nose-Hoover thermostat*****/
  }else if(thermo == 'N'){
    gSum = (0.5*S2 - (nAtom + 1)*temperature)/massS;
    varS += deltaT*gSum;
   for (n = 1; n <= nAtom; n++){
      vx[n] += deltaT * (ax[n] - varS *vx[n]);
      rx[n] += deltaT * vx[n];
      vy[n] += deltaT * (ay[n] - varS *vy[n]);
      ry[n] += deltaT * vy[n];
   }
      /*****for Gaussian thermostat*****/
 }else if(thermo == 'G'){
      for (n = 1; n <= nAtom; n++){
      vx[n] = C * vx[n] + D * ax[n];
      rx[n] += deltaT * vx[n];
      vy[n] = C * vy[n] + D * ay[n];
      ry[n] += deltaT * vy[n];
     }
   }else if (thermo == 'L'){
  double nu = 0.03066;
  double var = sqrt(2*nu/(GAMMA*deltaT));
  double scale = 1. + nu*deltaT/2.;
  double scale_v = 2./scale - 1.;
  double scale_f = deltaT/scale;
  int n;
  for(n = 1 ; n <= nAtom ; n ++){
```

```
      vx[n] = scale_v*vx[n] + scale_f*(ax[n] + var*gsl_ran_gaussian(rnd,1));
      rx[n] += deltaT * vx[n];
      vy[n] = scale_v*vy[n] + scale_f*(ay[n] + var*gsl_ran_gaussian(rnd,1));
      ry[n] += deltaT * vy[n];
    }
  }
 }else{
   int n;
   for(n = 1 ; n <= nAtom ; n ++){
     vx[n] += deltaT * ax[n];
     rx[n] += deltaT * vx[n];
     vy[n] += deltaT * ay[n];
     ry[n] += deltaT * vy[n];
   }
  }
}
```

References ax, ay, deltaT, kappa, nAtom, NDIM, rCut, region, regionH, rx, ry, SignR, Sqr, stepCount, stepEquil, thermo, vvSum, vx, and vy.

3.69 LeapfrogStep.c

Go to the documentation of this file.

```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
#include"global.h"


void LeapfrogStep(char thermo, gsl_rng * rnd){
double temperature, GAMMA;
GAMMA = 100;

double *TValSum;
TValSum = (double*)malloc((nAtom + 1) * sizeof(double));


  if(stepCount <= stepEquil){
    double gSum, varS, massS;
    temperature = 1./GAMMA;

   if(stepCount == 1) varS = 0.;
    double A, S1, S2, T;
    int n;
    S1 = 0.; S2 = 0.; gSum = 0.; massS = 0.1;

    vvSum = 0.;
    double halfdt = 0.5*deltaT;
    for (n = 1; n <= nAtom; n++){
      T = vx[n] + halfdt * ax[n];
      S1 += T * ax[n];
      S2 += Sqr(T);

      T = vy[n] + halfdt * ay[n];
      S1 += T * ay[n];
      S2 += Sqr(T);
     vvSum += (Sqr(vx[n]) +  Sqr(vy[n]));
    }

```

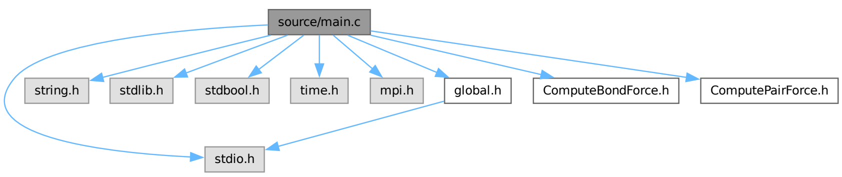
```
    A = -S1 / S2;
    S2 = vvSum;

    double C = 1 + A*deltaT ;
    double D = deltaT * (1 + 0.5 * A * deltaT);

    int i,j;
    real dr[NDIM+1], r, rr, ri, rrCut;
    double vv;

    double uVal, AA, AASum;
    double TVal;

    double deno, VVSum;
    deno = 0.;
    VVSum = 0.;
    AASum = 0.;

  for(n=1;n<=nAtom; n++)
     TValSum[n] = 0.;

  rrCut = Sqr(rCut);

/*****Calculating Configarational temperature*****/
//Solving the equation of motion here
if(thermo == 'C'){
for(i = 1 ; i <= nAtom; i ++){
    for(j = i+1 ; j <= nAtom ; j ++){
      dr[1] = rx[i] - rx[j];
      if(fabs(dr[1]) > regionH[1])
    dr[1] -= SignR(region[1], dr[1]);

     dr[2] = ry[i] - ry[j];
      if(fabs(dr[2]) > regionH[2])
    dr[2] -= SignR(region[2], dr[2]);

     rr = Sqr(dr[1]) + Sqr(dr[2]);
     if(rr < rrCut ){
     r = sqrt(rr);
     ri = 1/r;
     uVal = ri*exp(-kappa*r);

     TVal = (1./rr + Sqr(kappa) + kappa/r)*uVal;
     TValSum[i] += TVal;
     TValSum[j] += TVal;
   } }
     AA = Sqr(ax[i]) + Sqr(ay[i]);
     AASum += AA;
     vv = Sqr(vx[i]) + Sqr(vy[i]);
     VVSum += vv;
     deno += TValSum[i];
}

     double gSumconfig, varSconfig, massSconfig;
     if(stepCount == 1) varSconfig = 0.;
     gSumconfig = 0.; massSconfig = 2.0;

     gSumconfig = (AASum/temperature - deno)/massSconfig;
     varSconfig += deltaT*gSumconfig;

      /*****Configarational Nose-Hoover thermostat*****/
   for (n = 1; n <= nAtom; n++){
      vx[n] += deltaT * ax[n];
      rx[n] += deltaT * (vx[n] + varSconfig * ax[n]);
      vy[n] += deltaT * ay[n];
      ry[n] += deltaT * (vy[n] + varSconfig * ay[n]);
    }
      /*****Kinetic Nose-Hoover thermostat*****/
  }else if(thermo == 'N'){
    gSum = (0.5*S2 - (nAtom + 1)*temperature)/massS;
    varS += deltaT*gSum;
   for (n = 1; n <= nAtom; n++){
      vx[n] += deltaT * (ax[n] - varS *vx[n]);
      rx[n] += deltaT * vx[n];
      vy[n] += deltaT * (ay[n] - varS *vy[n]);
      ry[n] += deltaT * vy[n];
   }
      /*****for Gaussian thermostat*****/
 }else if(thermo == 'G'){
      for (n = 1; n <= nAtom; n++){
      vx[n] = C * vx[n] + D * ax[n];
      rx[n] += deltaT * vx[n];
      vy[n] = C * vy[n] + D * ay[n];
      ry[n] += deltaT * vy[n];
     }
   }else if (thermo == 'L'){
  double nu = 0.03066;
```

```
    double var = sqrt(2*nu/(GAMMA*deltaT));
    double scale = 1. + nu*deltaT/2.;
    double scale_v = 2./scale - 1.;
    double scale_f = deltaT/scale;
    int n;
    for(n = 1 ; n <= nAtom ; n ++){
        vx[n] = scale_v*vx[n] + scale_f*(ax[n] + var*gsl_ran_gaussian(rnd,1));
        rx[n] += deltaT * vx[n];
        vy[n] = scale_v*vy[n] + scale_f*(ay[n] + var*gsl_ran_gaussian(rnd,1));
        ry[n] += deltaT * vy[n];
      }
    }
  }else{
      int n;
      for(n = 1 ; n <= nAtom ; n ++){
        vx[n] += deltaT * ax[n];
        rx[n] += deltaT * vx[n];
        vy[n] += deltaT * ay[n];
        ry[n] += deltaT * vy[n];
      }
    }
}

```

3.70 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

3.70.1 Macro Definition Documentation

3.70.1.1 DEFINE_GLOBALS

```
#define DEFINE_GLOBALS
```

Definition at line 7 of file main.c.

3.70.2 Function Documentation

3.70.2.1 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  } }
```

3.70.2.2 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]);
```

```
  } else {
    // Print error message and exit the program
    fprintf(fpresult, "Error: Invalid boundary configuration: '%s %s'\n", xBoundary, yBoundary);
    exit(EXIT_FAILURE); // Exit with failure status
  }
 }
}
```

References atomRadius, fpresult, nAtom, region, regionH, rx, ry, vx, vy, xBoundary, and yBoundary.

Referenced by main().

Here is the caller graph for this function:

3.70.2.3 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   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.

3.70.2.4 ApplyForce()

```
void ApplyForce ( )
```

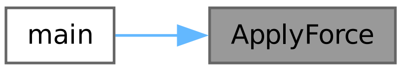
Definition at line 25 of file ApplyForce.c.

```
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:



3.70.2.5 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.

3.70.2.6 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] += stranRate * ry[n];
00030 } }
```

References nAtom, rx, ry, and strain.

3.70.2.7 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.

3.70.2.8 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.

3.70.2.9 Close()

```
void Close ( )
```

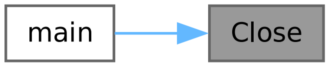
Definition at line 24 of file Close.c.

```
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     free(viscAcf);
00080 
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:



3.70.2.10 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]) +
      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.;
```

```
    if(m2Y > cells[2]){
      m2Y = 1; shift[2] = region[2];
    }else if(m2Y == 0){
      m2Y = cells[2]; shift[2] = -region[2];
    }
    m2 = (m2Y-1)*cells[1] + m2X + nAtom;
    I = cellList[m1];
    while(I > 0){
      J = cellList[m2];
      while(J > 0){
        if(m1 == m2 && J != I && (atomRadius[I] > 0. && atomRadius[J] > 0.)){
          dr[1] = rx[I] - rx[J] - shift[1];
          dr[2] = ry[I] - ry[J] - shift[2];
          rr = Sqr(dr[1]) + Sqr(dr[2]);
          RadiusIJ = atomRadius[I] + atomRadius[J];
          SqrRadiusIJ = Sqr(RadiusIJ);
          if(rr < SqrRadiusIJ){
        r = sqrt(rr);
        ri = 1.0/r;
                rrinv = 1.0/rr;
                vr[1] = vx[I] - vx[J];
                vr[2] = vy[I] - vy[J];
        RadiusIJInv = 1.0/RadiusIJ;
        uVal = Sqr(1.0 - r * RadiusIJInv);
        fcVal = 2.0 * RadiusIJInv * (1.0 - r * RadiusIJInv) *ri;
                fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag

        f  = fcVal * dr[1];
                fd = fdVal * dr[1];
        ax[I] += (f + fd);
                discDragx[I] += fd; //disc-disc drag

        f = fcVal * dr[2];
                fd = fdVal * dr[2];
        ay[I] += (f + fd);
                discDragy[I] += fd; //disc-disc drag

        uSum +=  0.5 * uVal;
        virSum += 0.5 * fcVal * rr;
        rfAtom += 0.5 * dr[1] * fcVal * dr[2];
          }
        }else if(m1 != m2 && (atomRadius[I] > 0. && atomRadius[J] > 0.)){
          dr[1] = rx[I] - rx[J] - shift[1];
          dr[2] = ry[I] - ry[J] - shift[2];
          rr = Sqr(dr[1]) + Sqr(dr[2]);
          RadiusIJ = atomRadius[I] + atomRadius[J];
          SqrRadiusIJ = Sqr(RadiusIJ);
          if(rr < SqrRadiusIJ){
        r = sqrt(rr);
        ri = 1.0/r;
                rrinv = 1.0/r;
                vr[1] = vx[I] - vx[J];
                vr[2] = vy[I] - vy[J];
        RadiusIJInv = 1.0/RadiusIJ;
        uVal = Sqr(1.0 - r * RadiusIJInv);
        fcVal = 2.0 * RadiusIJInv * (1.0 - r * RadiusIJInv) *ri;
                fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag

        f  = fcVal * dr[1];
                fd =  fdVal * dr[1];
        ax[I] += (f + fd);
                discDragx[I] += fd; //disc-disc drag

        f  = fcVal * dr[2];
                fd = fdVal * dr[2];
        ay[I] += (f + fd);
                discDragy[I] += fd; //disc-disc drag

        uSum +=  0.5 * uVal;
        virSum += 0.5 * fcVal * rr;
        rfAtom += 0.5 * dr[1] * fcVal * dr[2];
          }
        }
            J = cellList[J];
      }
      I = cellList[I];
    }
      }
    }
  }
}
```

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.

3.70.2.11 DisplaceAtoms()

```
void DisplaceAtoms ( )
```

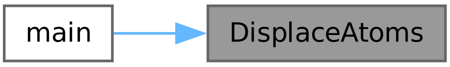
Definition at line 25 of file DisplaceAtoms.c.

```
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:



3.70.2.12 DumpBonds()

```
void DumpBonds ( )
```

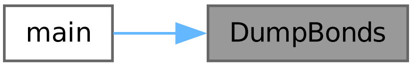
Definition at line 24 of file DumpBonds.c.

```
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 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:



3.70.2.13 DumpPairs()

```
void DumpPairs ( )
```

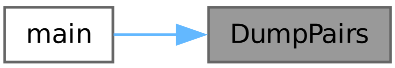
Definition at line 25 of file DumpPairs.c.

```
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:



3.70.2.14 DumpRestart()

```
void DumpRestart ( )
```

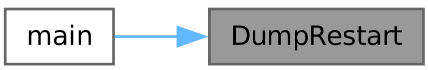
Definition at line 25 of file DumpRestart.c.

```
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:



3.70.2.15 DumpState()

```
void DumpState ( )
```

Definition at line 25 of file DumpState.c.

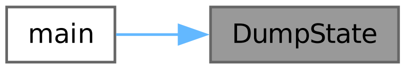
```
00025                  {
00026   char DUMP[256];
00027   FILE *fpDUMP;
00028   sprintf(DUMP, "%s.STATE", prefix);
00029   fpDUMP = fopen(DUMP, "w");
00030   if(fpDUMP == NULL) {
00031    fprintf(stderr, "Error opening file %s for writing\n", DUMP);
00032    return;
```

```
}

 fprintf(fpDUMP, "ITEM: TIMESTEP\n");
 fprintf(fpDUMP, "%lf\n",timeNow);
 fprintf(fpDUMP, "ITEM: NUMBER OF ATOMS\n");
 fprintf(fpDUMP, "%d\n",nAtom);
 fprintf(fpDUMP, "ITEM: BOX BOUNDS pp pp pp\n");
 fprintf(fpDUMP, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
 fprintf(fpDUMP, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
 fprintf(fpDUMP, "%lf %lf zlo zhi\n",  -0.1, 0.1);
 fprintf(fpDUMP, "ITEM: ATOMS id mol type radius x y vx vy fx fy\n");
 int n;
 for (n = 1; n <= nAtom; n++) {
  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\n",
  atomID[n], molID[n], atomType[n], atomRadius[n], rx[n], ry[n], vx[n], vy[n], ax[n], ay[n]);
 }
 fclose(fpDUMP);
}
```

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:



3.70.2.16 EvalCom()

```
void EvalCom ( )
```

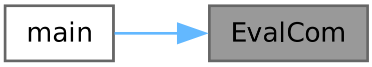
Definition at line 27 of file EvalCom.c.

```
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:



3.70.2.17 EvalProps()

```
void EvalProps ( )
```

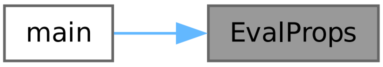
Definition at line 26 of file EvalProps.c.

```
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:



3.70.2.18 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     // 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.

3.70.2.19 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        indexCorr[nb] = 0.;
00094        countCorrAv ++;
00095        if (countCorrAv == limitCorrAv){
00096          for (j = 1; j <= nFunCorr*nValCorr; j++)
00097        spacetimeCorrAv[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);
```

```
      real tVal;
      for (n = 1; n <= nValCorr; n++){
    tVal = (n-1)*stepCorr*deltaT;
    fprintf (fpdnsty, "%e\t", tVal);
    int nn = nFunCorr*(n-1);
    for (j = 1; j <= nFunCorr; j ++)
          fprintf (fpdnsty, "%e\t", spacetimeCorrAv[nn + j]);
        fprintf (fpdnsty, "\n");
      }

      countCorrAv = 0.;
      for (j = 1; j <= nFunCorr*nValCorr; j++)
    spacetimeCorrAv[j] = 0.;
    }
  }
 }
}
```

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

3.70.2.20 EvalUnwrap()

```
void EvalUnwrap ( )
```

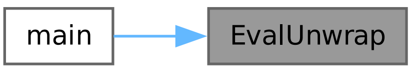
Definition at line 27 of file EvalUnwrap.c.

```
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:



3.70.2.21 EvalVacf()

```
void EvalVacf ( )
```

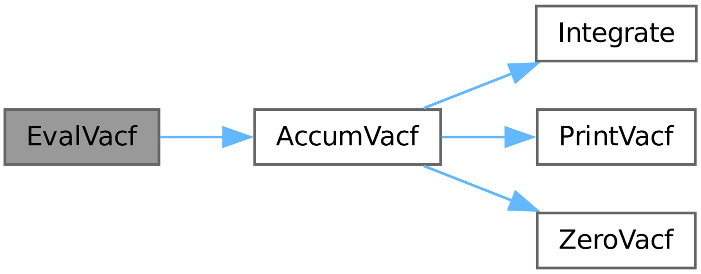
Definition at line 26 of file EvalVacf.c.

```
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;
```

```
  for(nb = 1 ; nb <= nBuffAcf ; nb ++){
    indexAcf[nb] ++;
    if(indexAcf[nb] <= 0)continue;
    if(indexAcf[nb] == 1){
      viscAcfOrg[nb] = viscVec;
    }
    ni = indexAcf[nb];
    viscAcf[nb][ni] = viscAcfOrg[nb]*viscVec;
  }
  AccumVacf();
}
```

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

Here is the call graph for this function:



3.70.2.22 EvalVrms()

```
void EvalVrms ( )
```

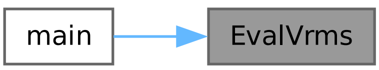
Definition at line 27 of file EvalVrms.c.

```
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:



3.70.2.23 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:

3.70.2.24 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);
```

```
fscanf(fp, "%s %d",  dummy, &nBuffAcf);
fscanf(fp, "%s %d",  dummy, &nValAcf);
fscanf(fp, "%s %d",  dummy, &stepAcf);
fscanf(fp, "%s %lf", dummy, &rangeRdf);
fscanf(fp, "%s %d",  dummy, &limitRdf);
fscanf(fp, "%s %d",  dummy, &sizeHistRdf);
fscanf(fp, "%s %d",  dummy, &stepRdf);

fclose(fp);
FILE *fpSTATE;
if((fpSTATE = fopen(inputConfig,"r"))==NULL){
printf("Error occurred: Could not open STATE file\n Exiting now..\n");
exit(0);
}

fscanf(fpSTATE, "%s %lf", dummy, &timeNow);
fscanf(fpSTATE, "%s %d",  dummy, &nAtom);
fscanf(fpSTATE, "%s %d",  dummy, &nBond);
fscanf(fpSTATE, "%s %d",  dummy, &nAtomType);
fscanf(fpSTATE, "%s %d",  dummy, &nBondType);
fscanf(fpSTATE, "%s %lf", dummy, &region[1]);
fscanf(fpSTATE, "%s %lf", dummy, &region[2]);

region[2] *= 1.5; //Remove this when put on GitHub

density = nAtom/(region[1]*region[2]);
cells[1] = region[1] / rCut;
cells[2] = region[2] / rCut;
cellList = (int*)malloc((nAtom + cells[1] * cells[2] + 1) * sizeof(int));
regionH[1] = 0.5*region[1];
regionH[2] = 0.5*region[2];

//strain information
strainRate = strain/deltaT;
shearDisplacement = strain * region[2];
shearVelocity = strainRate * region[2];
int n;

rx = (double*)malloc((nAtom + 1) * sizeof(double));
ry = (double*)malloc((nAtom + 1) * sizeof(double));
vx = (double*)malloc((nAtom + 1) * sizeof(double));
vy = (double*)malloc((nAtom + 1) * sizeof(double));
ax = (double*)malloc((nAtom + 1) * sizeof(double));
ay = (double*)malloc((nAtom + 1) * sizeof(double));
fax = (double*)malloc((nAtom + 1) * sizeof(double));
fay = (double*)malloc((nAtom + 1) * sizeof(double));
atomID = (int*)malloc((nAtom+1) * sizeof(int));
atomType = (int*)malloc((nAtom+1) * sizeof(int));
atomRadius = (double*)malloc((nAtom + 1) * sizeof(double));
atomMass = (double*)malloc((nAtom + 1) * sizeof(double));
speed = (double*)malloc((nAtom + 1) * sizeof(double));
atom1 = (int*)malloc((nBond+1)*sizeof(int));
atom2 = (int*)malloc((nBond+1)*sizeof(int));
BondID = (int*)malloc((nBond+1)*sizeof(int));
BondType = (int*)malloc((nBond+1)*sizeof(int));
kb = (double*)malloc((nBond+1)*sizeof(double));
ro = (double*)malloc((nBond+1)*sizeof(double));
BondEnergy = (double*)malloc((nBond+1)*sizeof(double));
BondLength =(double*)malloc((nBond+1)*sizeof(double));
discDragx = (double*)malloc((nAtom + 1) * sizeof(double));
discDragy = (double*)malloc((nAtom + 1) * sizeof(double));
nodeDragx = (double*)malloc((nAtom + 1) * sizeof(double));
nodeDragy = (double*)malloc((nAtom + 1) * sizeof(double));
ImageX = (int*)malloc((nAtom+1) * sizeof(int));
ImageY = (int*)malloc((nAtom+1) * sizeof(int));
rxUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
ryUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
DeltaXijOld = (double*)malloc((nBond+1)*sizeof(double));
DeltaYijOld = (double*)malloc((nBond+1)*sizeof(double));
DeltaXijOldPair = (double**)malloc((nAtom+1) * sizeof(double*));
DeltaYijOldPair = (double**)malloc((nAtom+1) * sizeof(double*));
for(int n = 0; n <= nAtom; n++) {
 DeltaXijOldPair[n] = (double*)malloc((nAtom+1) * sizeof(double));
 DeltaYijOldPair[n] = (double*)malloc((nAtom+1) * sizeof(double));
}
molID = (int*)malloc((nAtom+1) * sizeof(int));

for(n = 1; n <= nAtom; n ++){
 atomMass[n] = 1.0;
}

fscanf(fpSTATE, "%s\n", dummy);
for(n = 1; n <= nAtom; n ++)
 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]);


```

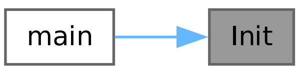
```
  fscanf(fpSTATE, "%s\n", dummy);
  for(n=1; n<=nBond; n++)
   fscanf(fpSTATE, "%d %d %d %d %lf %lf\n", &BondID[n], &BondType[n], &atom1[n], &atom2[n], &kb[n],
&ro[n]);

  fclose(fpSTATE);

 //2D-List of bonded atoms. This is used to remove pair interaction
 //calculation for the bonded atoms
   isBonded = (int**)malloc((nAtom + 1) * sizeof(int*));
  for (int i = 0; i <= nAtom; i++) {
    isBonded[i] = (int*)malloc((nAtom + 1) * sizeof(int));
    for (int j = 0; j <= nAtom; j++) {
        isBonded[i][j] = 0;
    }
  }

  for (n = 1; n <= nBond; n++) {
    int i = atom1[n];
    int j = atom2[n];
    isBonded[i][j] = 1;
    isBonded[j][i] = 1; // symmetric
}

 //For thermostate, update in final version
thermo = 'C';


 // List the interface atoms
 nAtomInterface = 0;
 nAtomBlock = 0;
 nDiscInterface = 0;
 double InterfaceWidth, bigDiameter;
 bigDiameter = 2.8;
 InterfaceWidth = 5.0 * bigDiameter;

 for(n = 1; n <= nAtom; n++){
  if(fabs(ry[n]) < InterfaceWidth){
  nAtomInterface++;
  }
  if(molID[n] == 2){
  nAtomBlock++;
  }
  if(atomRadius[n] != 0.0){
  nDiscInterface++;
  } }

   atomIDInterface =  (int*)malloc((nAtomInterface+1)*sizeof(int));

  int m;
  m = 1;
  for(n=1; n<=nAtom; n++){
   if(fabs(ry[n]) < InterfaceWidth){
   atomIDInterface[m] = atomID[n];
   m++;
   } }

  nPairTotal = 0.5 * nAtomInterface * (nAtomInterface-1);
  PairID = (int*)malloc((nPairTotal+1) * sizeof(int));
  Pairatom1 = (int*)malloc((nPairTotal+1) * sizeof(int));
  Pairatom2 = (int*)malloc((nPairTotal+1) * sizeof(int));
  PairXij = (double*)malloc((nPairTotal+1) * sizeof(double));
  PairYij = (double*)malloc((nPairTotal+1) * sizeof(double));

    fprintf(fpresult, "------------------------------------\n");
    fprintf(fpresult, "-------PARAMETERS-----------\n");
    fprintf(fpresult, "------------------------------------\n");
    fprintf(fpresult, "nAtom\t\t\t%d\n",  nAtom);
    fprintf(fpresult, "nBond\t\t\t%d\n",  nBond);
    fprintf(fpresult, "nAtomBlock\t\t%d\n",  nAtomBlock);
    fprintf(fpresult, "nAtomInterface\t%d\n",  nAtomInterface);
    fprintf(fpresult, "nDiscInterface\t\t%d\n",  nDiscInterface);
    fprintf(fpresult, "gamman\t\t\t%0.6g\n", gamman);
    fprintf(fpresult, "strain\t\t\t%0.6g\n", strain);
    fprintf(fpresult, "strainRate\t\t%0.6g\n", strainRate);
    fprintf(fpresult, "FyBylx\t\t\t%0.6g\n", FyBylx);
    fprintf(fpresult, "fxByfy\t\t\t%0.6g\n", fxByfy);
    fprintf(fpresult, "DeltaY\t\t\t%0.6g\n", DeltaY);
    fprintf(fpresult, "DeltaX\t\t\t%0.6g\n", DeltaX);
    fprintf(fpresult, "HaltCondition\t\t%0.6g\n", HaltCondition);
    fprintf(fpresult, "kappa\t\t\t%g\n", kappa);
    fprintf(fpresult, "density\t\t\t%g\n", density);
    fprintf(fpresult, "rCut\t\t\t%g\n", rCut);
    fprintf(fpresult, "deltaT\t\t\t%g\n", deltaT);
    fprintf(fpresult, "stepEquil\t\t\t%d\n",  stepEquil);
    fprintf(fpresult, "stepLimit\t\t\t%d\n",  stepLimit);
    fprintf(fpresult, "region[1]\t\t\t%0.16lf\n", region[1]);
```

```
    fprintf(fpresult, "region[2]\t\t\t%0.16lf\n", region[2]);
    fprintf(fpresult, "cells[1]\t\t\t%d\n",  cells[1]);
    fprintf(fpresult, "cells[2]\t\t\t%d\n",  cells[2]);
    fprintf(fpresult, "solver\t\t\t%s\n",  solver);
    fprintf(fpresult, "boundary\t\t\t%s %s\n", xBoundary, yBoundary);
    fprintf(fpresult, "DampFlag\t\t%d\n",  DampFlag);


    fprintf(fpresult, "------------------------------------\n");
    fprintf(fpresult, "#TimeNow TotalMomentum PotEngyPerAtom KinEngyPerAtom TotEngyPerAtom
PairEnergyPerAtom BondEnergyPerAtom  Press VirialSum\n");
    fprintf(fpvrms, "#timeNow\tVrms \n");
    fprintf(fpcom, "#timeNow\tComX\tComY\n");

/* //Uncomment the following as per your acquirement
    fprintf(fpstress, "strain          %lf\n", strain);
    fprintf(fpstress, "region[1]       %lf\n", region[1]);
    fprintf(fpstress, "region[2]       %lf\n", region[2]);
    fprintf(fpstress, "#timeNow virSumxx virSumyy virSumxy pressure\n");
    fprintf(fpmomentum, "#timeNow Px Py\n");
*/

   if((strcmp(xBoundary, "p") != 0 && strcmp(xBoundary, "r") != 0) ||
    (strcmp(yBoundary, "p") != 0 && strcmp(yBoundary, "r") != 0)) {
    fprintf(fpresult, "Error: Invalid boundary value detected: '%s %s'. Only 'p' or 'r' are
allowed.\n", xBoundary, yBoundary);
    exit(EXIT_FAILURE); // Exit with failure status
  }

}
```

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:



3.70.2.25 LeapfrogStep()

```
void LeapfrogStep ( )
```

3.70.2.26 main()

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

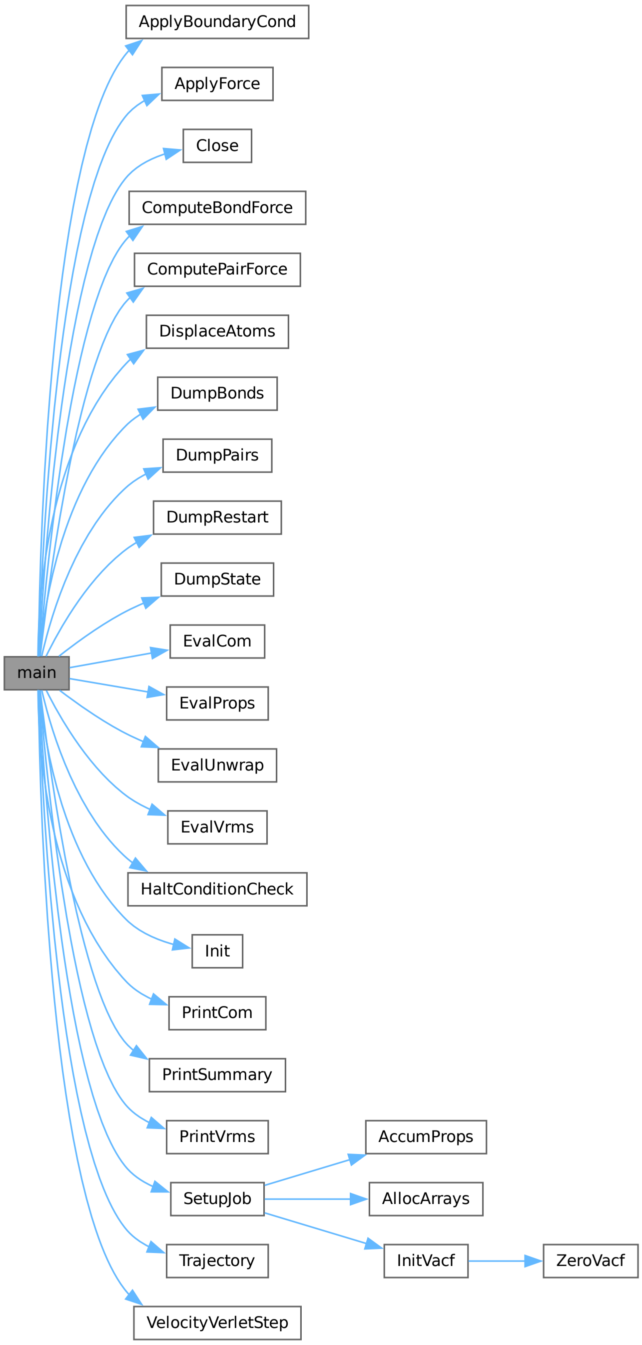
Definition at line 51 of file main.c.

```
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();
```

```
     PrintVrms();
     PrintCom();
     }
    if(stepCount % stepTraj == 0){
     Trajectory();
     DumpBonds();
     DumpPairs();
     }
    if(stepCount % stepDump == 0){
     DumpRestart();      // Save the current state for input
     DumpState();        // Save the current state for config
    }
    if(HaltConditionCheck(VRootMeanSqr, stepCount)) {
     DumpRestart();      // Save the current state for input
     DumpState();        // Save the current state for config
     break;  // Exit the loop when the halt condition is met
     }

     if(stepCount >= stepLimit)
      moreCycles = 0;
   }


   t2 = time(NULL);
   fprintf(fpresult, "#Execution time %lf secs\n", difftime(t2,t1));
   fprintf(fpresult, "#Execution speed %lf steps per secs\n", stepLimit/difftime(t2,t1));

   fclose(fpresult);
   fclose(fpxyz);
   fclose(fpvrms);
   fclose(fpbond);
   fclose(fppair);
   fclose(fpcom);

/*//Uncomment the following as per your acquirement
   fclose(fpdnsty);
   fclose(fpvisc);
   fclose(fprdf);
   fclose(fpstress);
   fclose(fpmomentum);
*/

   free(prefix);
   Close();
   return 0;
}
```

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:



3.70.2.27 PrintCom()

```
void PrintCom ( )
```

Definition at line 28 of file PrintCom.c.

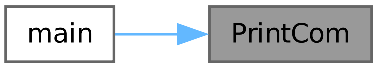
```
00028                  {
```

```
fprintf(fpcom, "%0.4lf\t%0.16lf\t%0.16lf\n", timeNow, ComX, ComY);
fflush(fpcom);
}
```

References ComX, ComY, fpcom, and timeNow.

Referenced by main().

Here is the caller graph for this function:



3.70.2.28 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.

3.70.2.29 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,
      virSumxy, pressure);
00027   fflush(fpstress);
00028 }
```

References fpstress, pressure, timeNow, virSumxx, virSumxy, and virSumyy.

3.70.2.30 PrintSummary()

```
void PrintSummary ( )
```

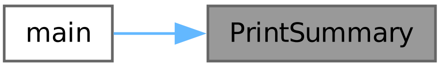
Definition at line 4 of file PrintSummary.c.

```
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\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:



3.70.2.31 PrintVrms()

```
void PrintVrms ( )
```

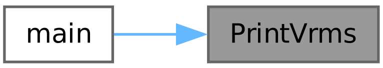
Definition at line 27 of file PrintVrms.c.

```
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:



3.70.2.32 SetupJob()

```
void SetupJob ( )
```

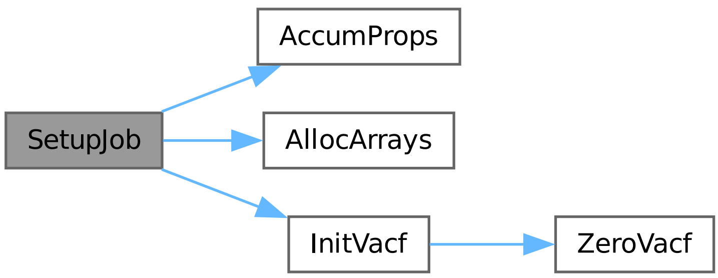
Definition at line 27 of file SetupJob.c.

```
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 }
```

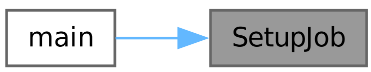
References AccumProps(), AllocArrays(), countCorrAv, countRdf, indexCorr, InitVacf(), nBuffCorr, nFunCorr, nValCorr, spacetimeCorrAv, and stepCount.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



3.70.2.33 Trajectory()

```
void Trajectory ( )
```

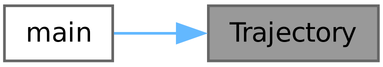
Definition at line 25 of file Trajectory.c.

```
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 %0.2lf\t %0.16lf\t %0.16lf\t %0.16lf\t %0.16lf\t %0.16lf\t
      %0.16lf\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:



3.70.2.34 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];
```

```
ImageY[n]--;
} } }
else if(icode == 2){
for(n = 1; n <= nAtom; n++) {
if(atomType[n] != freezeAtomType){
vx[n] += ax[n] * 0.5 * deltaT;
vy[n] += ay[n] * 0.5 * deltaT;
} } } }
```

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:

3.70.3 Variable Documentation

3.70.3.1 prefix

```
char* prefix = NULL
```

Definition at line 13 of file main.c.

Referenced by DumpRestart(), DumpState(), and main().

3.71 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();
```

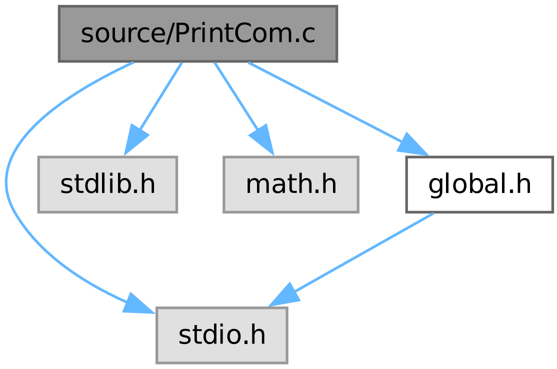
```c
void EvalRdf();
void AccumProps(int icode);
void PrintSummary();
void PrintVrms();
//void ComputeBondForce();
void DumpBonds();
void VelocityVerletStep(int icode);
void ApplyForce();
void ApplyDrivingForce();
void ApplyShear();
void ApplyLeesEdwardsBoundaryCond();
void PrintStress();
void Close();
//void ComputePairForce(int normFlag);
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) {
 time_t t1 = 0, t2;
 if (argc < 2) {
 fprintf(stderr, "Usage: %s <output_prefix>\n", argv[0]);
 return 1;
 }
  int prefix_size = snprintf(NULL, 0, "../output/%s", argv[1]) + 1; // +1 for the null terminator
  prefix = malloc(prefix_size);
  if(prefix == NULL) {
   fprintf(stderr, "Memory allocation failed\n");
   return 1;
   }

   // Write the formatted string into the allocated space
  snprintf(prefix, prefix_size, "../output/%s", argv[1]);
  sprintf(result, "%s.result", prefix);
  fpresult = fopen(result, "w");
  sprintf(xyz, "%s.xyz", prefix);
  fpxyz = fopen(xyz, "w");
  sprintf(vrms, "%s.vrms", prefix);
  fpvrms = fopen(vrms, "w");
  sprintf(bond, "%s.bond", prefix);
  fpbond = fopen(bond, "w");
  sprintf(com, "%s.com", prefix);
  fpcom = fopen(com, "w");
  sprintf(pair, "%s.pair", prefix);
  fppair = fopen(pair, "w");

  /* //Uncomment the following as per your acquirement
  sprintf(dnsty, "%s.curr-dnsty", prefix);
  fpdnsty = fopen(dnsty, "w");
  sprintf(visc, "%s.viscosity", prefix);
  fpvisc = fopen(visc, "w");
  sprintf(rdf, "%s.rdf", prefix);
  fprdf = fopen(rdf, "w");
  sprintf(stress, "%s.stress", prefix);
  fpstress = fopen(stress, "w");
  sprintf(momentum, "%s.momentum", prefix);
  fpmomentum = fopen(momentum, "w");
  */

  Init();
  SetupJob();
  t1 = time(NULL);
  moreCycles = 1;
  timeNow = 0.0;
  if(timeNow == 0.0) {
   DisplaceAtoms();
   ComputePairForce(1);
   ComputeBondForce();
   ApplyForce();
   DumpBonds();
   DumpPairs();
   Trajectory();
   EvalUnwrap();
   ApplyBoundaryCond();
   EvalProps();
   EvalVrms();
   EvalCom();
   PrintVrms();
   PrintCom();
   PrintSummary();
```

```
   }

//Here starts the main loop of the program
  while(moreCycles){
   if(stepLimit == 0){
   exit(0);
   }

   stepCount ++;
   timeNow = stepCount * deltaT; //for adaptive step size: timeNow += deltaT

   VelocityVerletStep(1);
   EvalUnwrap();
   ApplyBoundaryCond();
   ComputePairForce(1);
   ComputeBondForce();
   ApplyForce();
   VelocityVerletStep(2);
   ApplyBoundaryCond();
   EvalProps();
   EvalVrms();
   EvalCom();
   if(stepCount % stepAvg == 0){
    PrintSummary();
    PrintVrms();
    PrintCom();
    }
   if(stepCount % stepTraj == 0){
    Trajectory();
    DumpBonds();
    DumpPairs();
    }
   if(stepCount % stepDump == 0){
    DumpRestart();      // Save the current state for input
    DumpState();        // Save the current state for config
   }
   if(HaltConditionCheck(VRootMeanSqr, stepCount)) {
    DumpRestart();      // Save the current state for input
    DumpState();        // Save the current state for config
    break;  // Exit the loop when the halt condition is met
    }

    if(stepCount >= stepLimit)
     moreCycles = 0;
  }


  t2 = time(NULL);
  fprintf(fpresult, "#Execution time %lf secs\n", difftime(t2,t1));
  fprintf(fpresult, "#Execution speed %lf steps per secs\n", stepLimit/difftime(t2,t1));

  fclose(fpresult);
  fclose(fpxyz);
  fclose(fpvrms);
  fclose(fpbond);
  fclose(fppair);
  fclose(fpcom);

/*//Uncomment the following as per your acquirement
  fclose(fpdnsty);
  fclose(fpvisc);
  fclose(fprdf);
  fclose(fpstress);
  fclose(fpmomentum);
*/

  free(prefix);
  Close();
  return 0;
}
```

3.72 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 ()

3.72.1 Function Documentation

3.72.1.1 PrintCom()

```
void PrintCom ( )
```

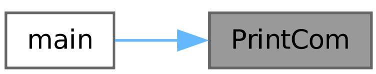
Definition at line 28 of file PrintCom.c.

```
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:



3.73 PrintCom.c

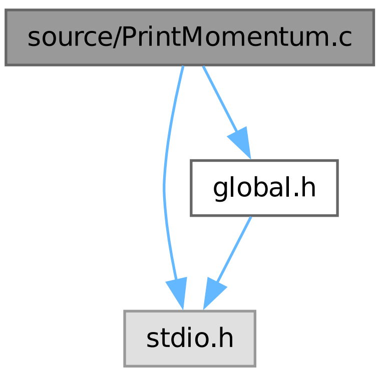
Go to the documentation of this file.

```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */



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

void PrintCom(){
 fprintf(fpcom, "%0.4lf\t%0.16lf\t%0.16lf\n", timeNow, ComX, ComY);
 fflush(fpcom);
 }



```

3.74 source/PrintMomentum.c File Reference

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

Include dependency graph for PrintMomentum.c:



Functions

- void PrintMomentum ()

3.74.1 Function Documentation

3.74.1.1 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.

3.75 PrintMomentum.c

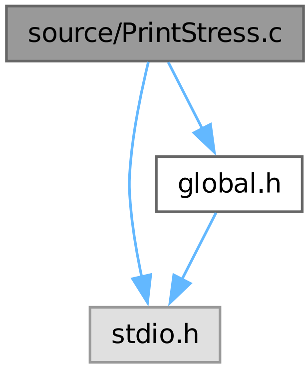
Go to the documentation of this file.

```
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 }
```

3.76 source/PrintStress.c File Reference

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

Include dependency graph for PrintStress.c:



Functions

- void PrintStress ()

3.76.1 Function Documentation

3.76.1.1 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.

3.77 PrintStress.c

Go to the documentation of this file.

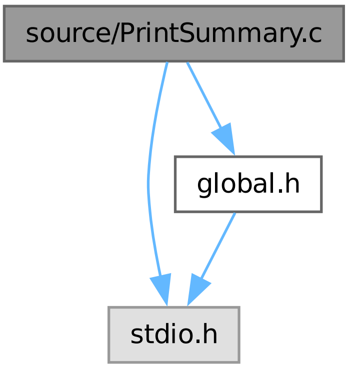
```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
```

```

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */


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

void PrintStress(){
  fprintf(fpstress, "%0.4lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\n", timeNow, virSumxx, virSumyy,
      virSumxy, pressure);
  fflush(fpstress);
}
```

3.78 source/PrintSummary.c File Reference

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

Include dependency graph for PrintSummary.c:



Functions

- void PrintSummary ()

3.78.1 Function Documentation

3.78.1.1 PrintSummary()

```
void PrintSummary ( )
```

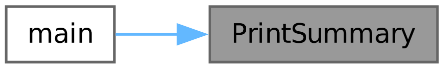
Definition at line 4 of file PrintSummary.c.

```
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\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:



3.79 PrintSummary.c

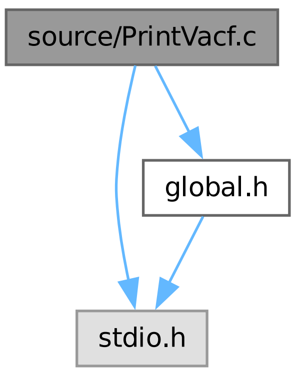
Go to the documentation of this file.

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

void PrintSummary(){
 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\t%0.16lf\n",
  timeNow, vSum, potEnergy, kinEnergy, totEnergy, uSumPairPerAtom, BondEnergyPerAtom, pressure,
 virSum);
  fflush(fpresult);
}
```

3.80 source/PrintVacf.c File Reference

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

Include dependency graph for PrintVacf.c:



Functions

- void PrintVacf ()

3.80.1 Function Documentation

3.80.1.1 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:

3.81 PrintVacf.c

Go to the documentation of this file.

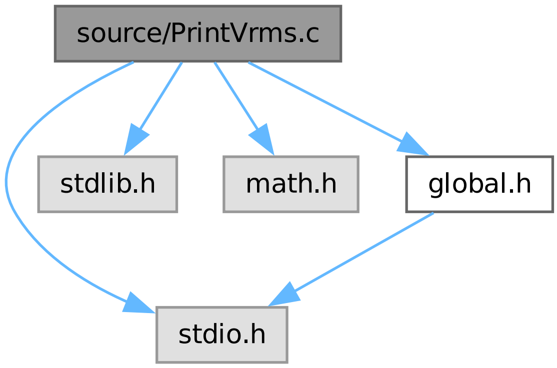
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your 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"
```

```
void PrintVacf(){
 double tVal;
 int j;
 fprintf(fpvisc,"viscosity acf\n");
 for(j = 1 ; j <= nValAcf ; j ++){
  tVal = (j-1)*stepAcf*deltaT;
  fprintf(fpvisc, "%lf\t %lf\t %lf\n", tVal, viscAcfAv[j], viscAcfAv[j]/viscAcfAv[1]);
  }
 fprintf(fpvisc, "viscosity acf integral : %lf\n", viscAcfInt);
}
```

3.82 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 ()

3.82.1 Function Documentation

3.82.1.1 PrintVrms()

```
void PrintVrms ( )
```

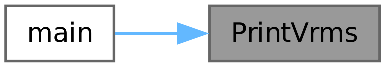
Definition at line 27 of file PrintVrms.c.

```
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:



3.83 PrintVrms.c

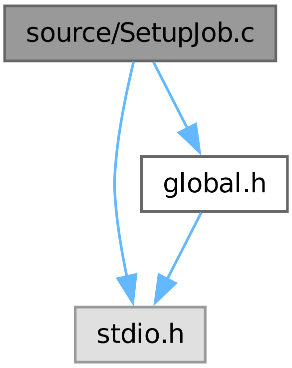
Go to the documentation of this file.

```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */


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

void PrintVrms(){
 fprintf(fpvrms, "%0.4lf\t%0.16lf\n", timeNow, VRootMeanSqr);
 fflush(fpvrms);
}



```

3.84 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 ()

3.84.1 Function Documentation

3.84.1.1 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:

3.84.1.2 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:

3.84.1.3 InitVacf()

```
void InitVacf ( )
```

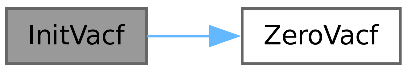
Definition at line 26 of file InitVacf.c.

```
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:

3.84.1.4 SetupJob()

```
void SetupJob ( )
```

Definition at line 27 of file SetupJob.c.

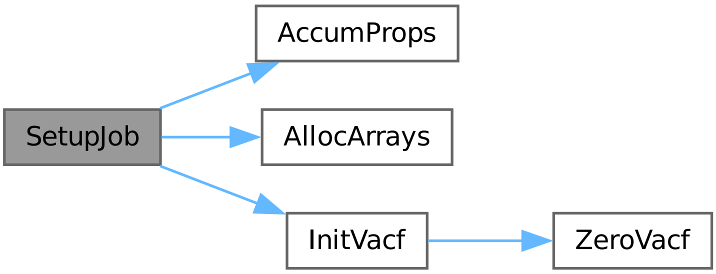
```
00027                {
00028   AllocArrays();
00029   AccumProps(0);
00030   InitVacf();
00031   stepCount = 0;
00032   // INITIALISE SPACETIME CORRELATIONS
00033   int n;
00034   for (n = 1; n <= nBuffCorr; n++)
00035     indexCorr[n] = -(n - 1)*nValCorr/nBuffCorr;
00036
00037   countCorrAv = 0.;
00038
00039   for (n = 1; n <= nFunCorr*nValCorr; n++)
00040     spacetimeCorrAv[n] = 0.;
00041
00042   //RDF
```

```
  countRdf = 0;
}
```

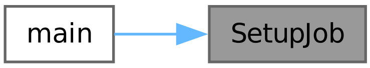
References AccumProps(), AllocArrays(), countCorrAv, countRdf, indexCorr, InitVacf(), nBuffCorr, nFunCorr, nValCorr, spacetimeCorrAv, and stepCount.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



3.85 SetupJob.c

Go to the documentation of this file.

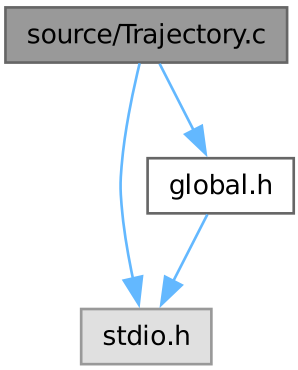
```
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */
```

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

void AllocArrays();
void AccumProps(int icode);
void InitVacf();
void SetupJob(){
  AllocArrays();
  AccumProps(0);
  InitVacf();
  stepCount = 0;
  // INITIALISE SPACETIME CORRELATIONS
  int n;
  for (n = 1; n <= nBuffCorr; n++)
    indexCorr[n] = -(n - 1)*nValCorr/nBuffCorr;

  countCorrAv = 0.;

  for (n = 1; n <= nFunCorr*nValCorr; n++)
    spacetimeCorrAv[n] = 0.;

  //RDF
  countRdf = 0;
}
```

3.86 source/Trajectory.c File Reference

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

Include dependency graph for Trajectory.c:



Functions

- void Trajectory ()

3.86.1 Function Documentation

3.86.1.1 Trajectory()

```
void Trajectory ( )
```

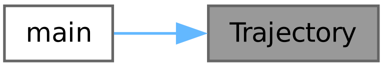
Definition at line 25 of file Trajectory.c.

```
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 %0.2lf\t %0.16lf\t %0.16lf\t %0.16lf\t %0.16lf\t %0.16lf\t
      %0.16lf\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:



3.87 Trajectory.c

Go to the documentation of this file.

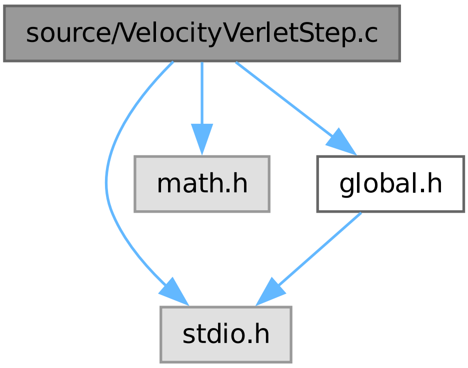
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your 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);
```

```
fprintf(fpxyz, "ITEM: ATOMS id mol type radius x y vx vy fx fy\n");
for(n=1; n<=nAtom; n++)
 fprintf(fpxyz, "%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\n",
 atomID[n], molID[n], atomType[n], atomRadius[n], rx[n], ry[n], vx[n], vy[n], ax[n], ay[n]);
}
```

3.88 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)

3.88.1 Function Documentation

3.88.1.1 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;
```

```c
 }
 //Calculating the image flags here
 if (rx[n] >= regionH[1]) {
  rx[n] -= region[1];
  ImageX[n]++;
  } else if (rx[n] < -regionH[1]) {
  rx[n] += region[1];
 ImageX[n]--;
 }
 if (ry[n] >= regionH[2]) {
  ry[n] -= region[2];
  ImageY[n]++;
  } else if (ry[n] < -regionH[2]) {
  ry[n] += region[2];
  ImageY[n]--;
  } } }
 else if(icode == 2){
 for(n = 1; n <= nAtom; n++) {
 if(atomType[n] != freezeAtomType){
  vx[n] += ax[n] * 0.5 * deltaT;
  vy[n] += ay[n] * 0.5 * deltaT;
} } } }
```

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:

3.89 VelocityVerletStep.c

Go to the documentation of this file.

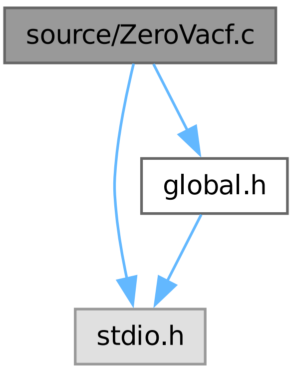
```c
/*
 * This file is part of Lamina.
 *
 * Lamina is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Lamina is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Lamina. If not, see <https://www.gnu.org/licenses/>.

 Copyright (C) 2025 Harish Charan, University of Durham, UK

 */


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

void VelocityVerletStep(int icode){
int n;
 if(icode == 1){
 for (n= 1; n <= nAtom; n++) {
```

```
  if(atomType[n] != freezeAtomType){
  vx[n] += ax[n] * 0.5 * deltaT;
  vy[n] += ay[n] * 0.5 * deltaT;
  rx[n] += vx[n] * deltaT;
  ry[n] += vy[n] * deltaT;
  }
  //Calculating the image flags here
  if (rx[n] >= regionH[1]) {
   rx[n] -= region[1];
   ImageX[n]++;
   } else if (rx[n] < -regionH[1]) {
   rx[n] += region[1];
  ImageX[n]--;
  }
  if (ry[n] >= regionH[2]) {
   ry[n] -= region[2];
   ImageY[n]++;
   } else if (ry[n] < -regionH[2]) {
   ry[n] += region[2];
   ImageY[n]--;
   } } }
  else if(icode == 2){
  for(n = 1; n <= nAtom; n++) {
  if(atomType[n] != freezeAtomType){
   vx[n] += ax[n] * 0.5 * deltaT;
   vy[n] += ay[n] * 0.5 * deltaT;
} } } }
```

3.90 source/ZeroVacf.c File Reference

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

Include dependency graph for ZeroVacf.c:



Functions

- void ZeroVacf ()

3.90.1 Function Documentation

3.90.1.1 ZeroVacf()

```
void ZeroVacf ( )
```

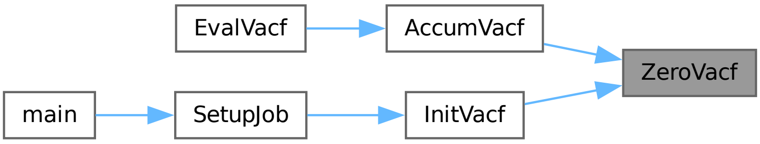
Definition at line 25 of file ZeroVacf.c.

```
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:



3.91 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