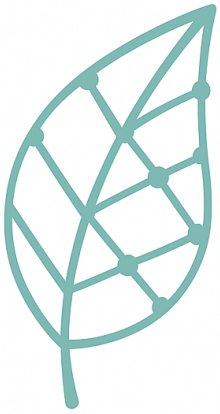


# Lamina

Lemina

Generated by Doxygen 1.13.2

# Chapter 1

# Lamina: A Molecular Dynamics Package

Welcome to the **Lamina** documentation!

## 1.1 Overview

**Lamina** is a modular 2D molecular dynamics (MD) simulation package designed for simulating hybrid soft solids, including spring networks and finite-size discs. Written in C, it models a wide variety of soft and condensed matter systems. It supports robust time evolution integrators and a range of thermostats, providing accurate force evaluations for bonded and non-bonded interactions.
Originally developed for 2D bonded systems, **Lamina** now supports 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 flat, two-dimensional structural elements such as 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 simple geometry, this code explores the complexity of emergent phenomena in 2D soft matter systems.

## 1.3 Key Features

### 1.3.1 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)

### 1.3.2 Thermostats and Temperature Control

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

### 1.3.3 Time Integration

- Leap-Frog integrator
- Velocity-Verlet integrator
- Langevin (stochastic) integrator
- Brownian (overdamped) integrator

### 1.3.4 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

### 1.3.5 Output and Utilities

- Output files saved to `../output` folder — ensure this directory exists relative to where you run the code
- Run the simulation with:

```
./main prefix
```

- Structured output files: `.xyz`, `.bond`, `.pair`, `.com`, `.result`
- Restart and resume capability: `.restart` and `.state` files
- Clear separation of source code, unit tests, and output
- Support for Lees–Edwards boundary conditions (sheared systems)
- Configurable halting conditions based on VRMS or custom metrics
- Modular design for easy extension of potentials and features

---

## 1.4 Project Structure

```
Lamina/
  |-- source/                  # C source files; avoid placing README.md here to prevent extra pages
  |     |-- main.c             # Main driver
  |     |-- *.c, *.h           # Modular source files
  |-- unittest/                # Unit test suite (planned or implemented)
  |     |-- test_*.c           # Individual test cases
  |-- output/                  # Runtime output files
  |-- prepros/                 # Preprocessing scripts/tools (.sh, .py, etc.)
  |-- postpros/                # Postprocessing scripts/tools (.sh, .py, etc.)
  |-- doxygen/                 # Doxygen configuration and auxiliary files
  |     |-- Doxyfile           # Doxygen config file
  |     |-- header.tex         # Custom LaTeX header for docs
  |     \-- extra_stylesheet.css # Optional CSS for HTML styling
  |-- figures/                 # Figures, logos, icons used in docs/code
  |     \-- LogoLaminaLatex.png # Project logo for documentation
  |-- docs/                    # Generated documentation (HTML, LaTeX, PDFs)
  |     |-- html/              # Doxygen-generated HTML docs
  |     |-- latex/             # Doxygen-generated LaTeX sources
  |     \-- refman.pdf         # Generated PDF manual
  |-- Makefile                 # Build system for manual Make builds
  |-- CMakeLists.txt           # CMake build system configuration
  |-- README.md                # This main documentation file
  \-- .github/                 # GitHub configuration directory
        \-- workflows/         # GitHub Actions workflows for CI/CD
            \-- ci.yml         # CI workflow file
  |--generate-docs.sh          # To genrate the html and latex documents
```

## 1.5 Installation Instructions

### 1.5.1 Prerequisites

1. **GCC Compiler**
   Install `gcc` to compile C code:
   - Ubuntu/Debian:
     ```
     sudo apt-get install build-essential
     ```
   - Fedora/CentOS:
     ```
     sudo dnf install gcc
     ```
   - macOS (via Homebrew):
     `bash brew install gcc` CI Build Status
2. **MPICH (MPI Library)**
   Required for parallel computations:
   - Ubuntu/Debian:
     ```
     sudo apt-get install libmpich-dev
     ```
   - Fedora/CentOS:
     ```
     sudo dnf install mpich
     ```
   - macOS:
     `bash brew install mpich`
3. **CMake (Recommended for modern builds)**
   Install `cmake` to build with the CMake system:
   - Ubuntu/Debian:
     ```
     sudo apt-get install cmake
     ```
   - Fedora/CentOS:
     ```
     sudo dnf install cmake
     ```
   - macOS:
     `bash brew install cmake`

## 1.6 Building Lamina

You can build Lamina either using the traditional Makefile or the CMake build system.

### 1.6.1 Using Makefile

```
cd Lamina/source
make clean
make
```

To build and run unit tests:

```
cd ../unittest
make clean
make all
make run
```

### 1.6.2 Using CMake (Recommended)

This builds the project in a clean isolated directory and manages dependencies automatically.

```
cd Lamina
mkdir -p build
cd build
cmake ..
make -j$(nproc)
```

To run unit tests (assuming they are built in `unittest` and `mpirun` is used):

```
cd ../unittest
make clean
make all
make run
```

## 1.7 Continuous Integration (CI) with GitHub Actions

The project includes a GitHub Actions workflow (`.github/workflows/ci.yml`) that automates building and testing on Ubuntu runners with MPI installed.
The workflow performs the following:

- Checks out the latest code on push or pull requests to the `main` branch.
- Installs build dependencies including `build-essential`, `cmake`, `mpich`, and MPI development libraries.
- Configures and builds Lamina using CMake in the `build` directory.
- Runs the unit tests in parallel using MPI.

You can view the build status and logs on the **Actions** tab of the GitHub repository.

## 1.8 Documentation

- Browse full `HTML documentation`
- Download `Source code PDF manual`
- User manual `Physics PDF manual`
- Documentation generated with `Doxygen 1.10.0`

Thank you for your interest in **Lamina**! Contributions and feedback are welcome.
Please check the repository for the latest updates and contact information.

## 1.9 Adding a Build Status Badge

# Chapter 2

# Class Index

## 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 3

# File Index

## 3.1 File List

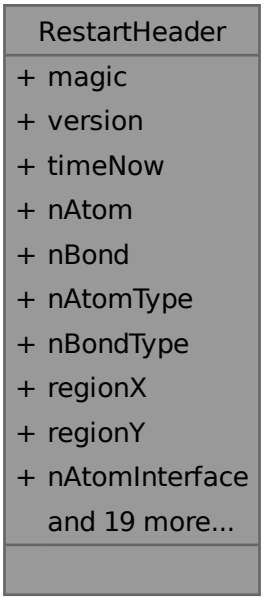
Here is a list of all files with brief descriptions:

# Chapter 4

# Class Documentation

## 4.1 RestartHeader Struct Reference

Collaboration diagram for RestartHeader:




### Public Attributes

- char magic [8]
- double version
- double timeNow
- int nAtom
- int nBond
- int nAtomType
- int nBondType
- double regionX
- double regionY
- int nAtomInterface

- int nAtomBlock
- int nDiscInterface
- double bigDiameter
- double InterfaceWidth
- int nPairActive
- int nPairTotal
- double uSumPair
- double virSumPair
- double virSumPairxx
- double virSumPairyy
- double virSumPairxy
- double TotalBondEnergy
- double virSumBond
- double virSumBondxx
- double virSumBondyy
- double virSumBondxy
- int stepCount
- double forceSumxExtern
- double forceSumyExtern

### 4.1.1 Detailed Description

Definition at line 27 of file ReadRestartBinary.c.

### 4.1.2 Member Data Documentation

#### 4.1.2.1 bigDiameter

`double RestartHeader::bigDiameter`
Definition at line 40 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.2 forceSumxExtern

`double RestartHeader::forceSumxExtern`
Definition at line 55 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.3 forceSumyExtern

`double RestartHeader::forceSumyExtern`
Definition at line 56 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.4 InterfaceWidth

`double RestartHeader::InterfaceWidth`
Definition at line 41 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.5 magic

`char RestartHeader::magic`
Definition at line 28 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.6 nAtom

`int RestartHeader::nAtom`
Definition at line 31 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.7 nAtomBlock

`int RestartHeader::nAtomBlock`
Definition at line 38 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.8 nAtomInterface

`int RestartHeader::nAtomInterface`
Definition at line 37 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.9 nAtomType

`int RestartHeader::nAtomType`
Definition at line 33 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.10 nBond

`int RestartHeader::nBond`
Definition at line 32 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.11 nBondType

`int RestartHeader::nBondType`
Definition at line 34 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.12 nDiscInterface

`int RestartHeader::nDiscInterface`
Definition at line 39 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.13 nPairActive

`int RestartHeader::nPairActive`
Definition at line 42 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.14 nPairTotal

`int RestartHeader::nPairTotal`
Definition at line 43 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.15 regionX

`double RestartHeader::regionX`
Definition at line 35 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.16 regionY

`double RestartHeader::regionY`
Definition at line 36 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.17 stepCount

`int RestartHeader::stepCount`
Definition at line 54 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.18 timeNow

`double RestartHeader::timeNow`
Definition at line 30 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.19 TotalBondEnergy

`double RestartHeader::TotalBondEnergy`
Definition at line 49 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.20 uSumPair

`double RestartHeader::uSumPair`
Definition at line 44 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.21 version

`double RestartHeader::version`
Definition at line 29 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.22 virSumBond

`double RestartHeader::virSumBond`
Definition at line 50 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.23 virSumBondxx

`double RestartHeader::virSumBondxx`
Definition at line 51 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.24 virSumBondxy

`double RestartHeader::virSumBondxy`
Definition at line 53 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.25 virSumBondyy

`double RestartHeader::virSumBondyy`
Definition at line 52 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.26 virSumPair

`double RestartHeader::virSumPair`
Definition at line 45 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.27 virSumPairxx

`double RestartHeader::virSumPairxx`
Definition at line 46 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.28 virSumPairxy

`double RestartHeader::virSumPairxy`
Definition at line 48 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().

#### 4.1.2.29 virSumPairyy

`double RestartHeader::virSumPairyy`
Definition at line 47 of file ReadRestartBinary.c.
Referenced by ReadBinaryRestart().
The documentation for this struct was generated from the following files:

- source/ReadRestartBinary.c
- source/WriteRestartBinary.c

# Chapter 5

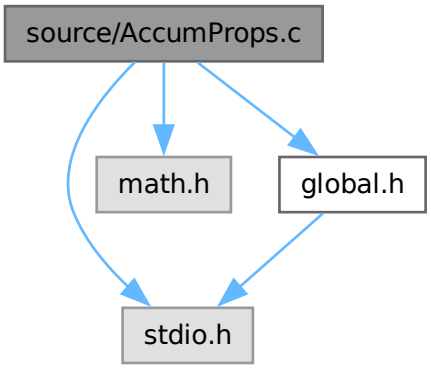
# File Documentation

## 5.1 README.md File Reference

## 5.2 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)

### 5.2.1 Function Documentation

#### 5.2.1.1 AccumProps()

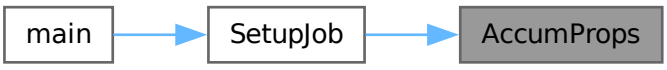
```
void AccumProps (
            int icode)
```

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:




## 5.3 AccumProps.c

Go to the documentation of this file.

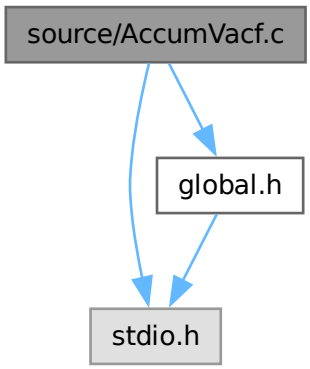
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021 #include<stdio.h>
00022 #include<math.h>
00023 #include"global.h"
00024
00025 void AccumProps(int icode){
00026  if(icode == 0){
00027  sPotEnergy = ssPotEnergy = 0.;
00028  sKinEnergy = ssKinEnergy = 0.;
00029  sPressure = ssPressure = 0.;
00030  sTotEnergy = ssTotEnergy = 0.;
00031  svirSum = 0.;
00032  }else if(icode == 1){
00033  sPotEnergy += potEnergy;
00034  ssPotEnergy += Sqr(potEnergy);
00035  sKinEnergy += kinEnergy;
```

```
00036  ssKinEnergy += Sqr(kinEnergy);
00037  sTotEnergy += totEnergy;
00038  ssTotEnergy += Sqr(totEnergy);
00039  sPressure += pressure;
00040  ssPressure += Sqr(pressure);
00041  svirSum += virSum;
00042  }else if(icode == 2){
00043  sPotEnergy /= stepAvg;
00044  ssPotEnergy /= sqrt(ssPotEnergy/stepAvg - Sqr(sPotEnergy));
00045  sTotEnergy /= stepAvg;
00046  ssTotEnergy = sqrt(ssTotEnergy/stepAvg - Sqr(sTotEnergy));
00047  sKinEnergy /= stepAvg;
00048  ssKinEnergy = sqrt(ssKinEnergy/stepAvg - Sqr(sKinEnergy));
00049  sPressure /= stepAvg;
00050  ssPressure = sqrt(ssPressure/stepAvg - Sqr(sPressure));
00051  svirSum /= stepAvg;
00052  } }
```

## 5.4 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 ()

### 5.4.1 Function Documentation

#### 5.4.1.1 AccumVacf()

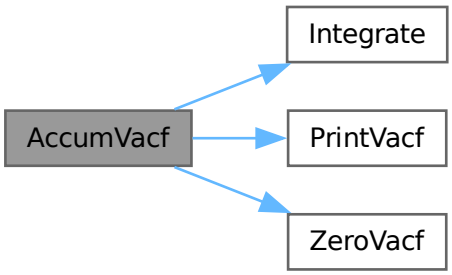
```
void AccumVacf ()
```

Definition at line 27 of file AccumVacf.c.

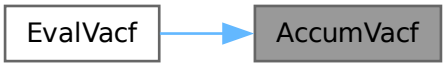
```
00027                  {
00028  double fac;
00029  int j, nb;
00030  for(nb = 1 ; nb <= nBuffAcf ; nb ++){
00031   if(indexAcf[nb] == nValAcf){
00032    for(j = 1 ; j <= nValAcf; j ++){
00033     viscAcfAv[j] +=  viscAcf[nb][j];
00034     }
00035   indexAcf[nb] = 0;
```

```
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:




#### 5.4.1.2 Integrate()

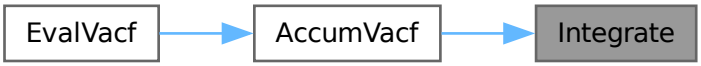
```
double Integrate (
            double * f,
            int nf)
```

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:



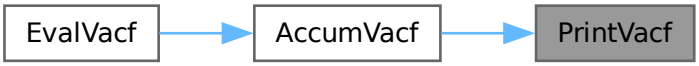

#### 5.4.1.3 PrintVacf()

```
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:




#### 5.4.1.4 ZeroVacf()

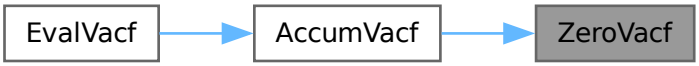
```
void ZeroVacf ()
```

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:




## 5.5 AccumVacf.c

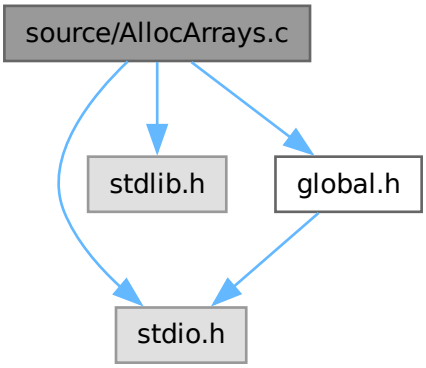
Go to the documentation of this file.

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021 #include<stdio.h>
00022 #include"global.h"
00023
00024 double Integrate(double *, int);
00025  void PrintVacf();
00026  void ZeroVacf();
00027  void AccumVacf(){
00028  double fac;
00029  int j, nb;
00030  for(nb = 1 ; nb <= nBuffAcf ; nb ++){
00031   if(indexAcf[nb] == nValAcf){
00032    for(j = 1 ; j <= nValAcf; j ++){
00033     viscAcfAv[j] +=  viscAcf[nb][j];
00034     }
00035   indexAcf[nb] = 0;
00036   countAcfAv ++;
00037   if(countAcfAv == limitAcfAv){
00038    fac = 1./(kinEnergy*region[1]*region[2]*limitAcfAv);
00039    viscAcfInt = fac*stepAcf*deltaT*Integrate(viscAcfAv, nValAcf);
00040    PrintVacf();
00041    ZeroVacf();
00042 } } } }
00043
```

## 5.6 source/AllocArrays.c File Reference

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

Include dependency graph for AllocArrays.c:




### Functions

- void AllocArrays ()

## 5.6.1 Function Documentation

### 5.6.1.1 AllocArrays()

```
void AllocArrays ()
```

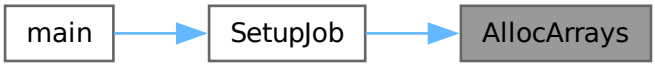
Definition at line 25 of file AllocArrays.c.

```
00025                   {
00026  int n;
00027
00028  // SPACETIME CORRELATIONS
00029  cfOrg = (double **) malloc ((nBuffCorr+1)*sizeof(double *));
00030  for (n = 0; n <= nBuffCorr; n++)
00031   cfOrg[n] = (double *) malloc ((2*nFunCorr+1)*sizeof(double));
00032
00033  cfVal = (double *) malloc ((2*nFunCorr+1)*sizeof(double));
00034  indexCorr = (int *) malloc ((nBuffCorr+1)*sizeof(int));
00035
00036  spacetimeCorr = (double **) malloc ((nBuffCorr+1)*sizeof(double));
00037  for (n = 0; n <= nBuffCorr; n++)
00038   spacetimeCorr[n] = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));
00039
00040  spacetimeCorrAv = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));
00041
00042  // VISCOSITY
00043  indexAcf = (double *)malloc((nBuffAcf+1)*sizeof(double));
00044  viscAcf = (double **)malloc((nBuffAcf+1)*sizeof(double *));
00045  for(n = 0 ; n <= nBuffAcf ; n ++)
00046   viscAcf[n] = (double *)malloc((nValAcf+1)*sizeof(double ));
00047
00048  viscAcfOrg = (double *)malloc((nBuffAcf+1)*sizeof(double));
00049  viscAcfAv = (double *)malloc((nValAcf+1)*sizeof(double));
00050
00051   // RDF
00052   histRdf = (double *)malloc((sizeHistRdf+1)*sizeof(double));
00053 }
```

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:




## 5.7 AllocArrays.c

Go to the documentation of this file.

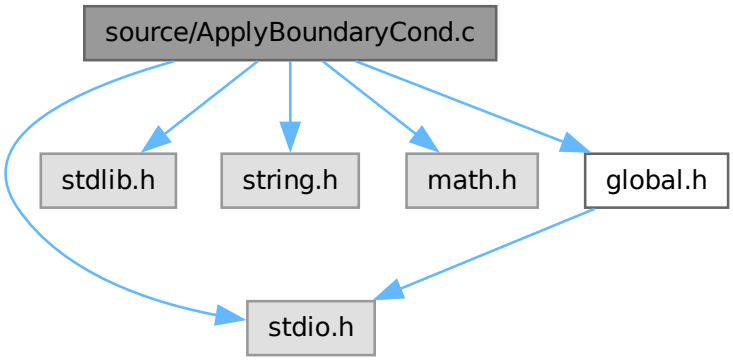
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021 #include<stdio.h>
00022 #include<stdlib.h>
00023 #include"global.h"
00024
00025 void AllocArrays(){
00026  int n;
00027
00028  // SPACETIME CORRELATIONS
00029  cfOrg = (double **) malloc ((nBuffCorr+1)*sizeof(double *));
00030  for (n = 0; n <= nBuffCorr; n++)
00031   cfOrg[n] = (double *) malloc ((2*nFunCorr+1)*sizeof(double));
00032
00033  cfVal = (double *) malloc ((2*nFunCorr+1)*sizeof(double));
00034  indexCorr = (int *) malloc ((nBuffCorr+1)*sizeof(int));
00035
00036  spacetimeCorr = (double **) malloc ((nBuffCorr+1)*sizeof(double));
00037  for (n = 0; n <= nBuffCorr; n++)
00038   spacetimeCorr[n] = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));
00039
00040  spacetimeCorrAv = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));
00041
00042  // VISCOSITY
00043  indexAcf = (double *)malloc((nBuffAcf+1)*sizeof(double));
00044  viscAcf = (double **)malloc((nBuffAcf+1)*sizeof(double *));
00045  for(n = 0 ; n <= nBuffAcf ; n ++)
00046   viscAcf[n] = (double *)malloc((nValAcf+1)*sizeof(double ));
00047
00048  viscAcfOrg = (double *)malloc((nBuffAcf+1)*sizeof(double));
00049  viscAcfAv = (double *)malloc((nValAcf+1)*sizeof(double));
00050
00051   // RDF
00052   histRdf = (double *)malloc((sizeHistRdf+1)*sizeof(double));
00053 }
```

## 5.8 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 ()

## 5.8.1 Function Documentation

### 5.8.1.1 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:


## 5.9 ApplyBoundaryCond.c

Go to the documentation of this file.

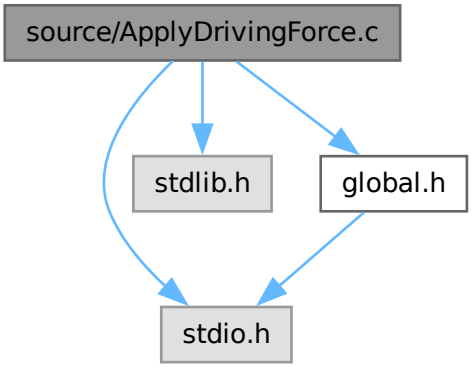
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your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

```
00054       }if((rx[n] - atomRadius[n]) < -regionH[1]){
00055        rx[n] = -0.999999*regionH[1] + atomRadius[n]; vx[n] = -vx[n] ;
00056       }
00057       ry[n] -= region[2]*rint(ry[n]/region[2]);
00058    } else {
00059      // Print error message and exit the program
00060      fprintf(fpresult, "Error: Invalid boundary configuration: '%s %s'\n", xBoundary, yBoundary);
00061      exit(EXIT_FAILURE); // Exit with failure status
00062    }
00063  }
00064 }
```

## 5.10 source/ApplyDrivingForce.c File Reference

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

Include dependency graph for ApplyDrivingForce.c:




### Functions

- void ApplyDrivingForce ()

### 5.10.1 Function Documentation

#### 5.10.1.1 ApplyDrivingForce()

```
void ApplyDrivingForce ()
```

Definition at line 25 of file ApplyDrivingForce.c.

```
00025                          {
00026  int n;
00027  double Vxblock, Vyblock;
00028  double Vxsubstrate, Vysubstrate;
00029  Vxblock = 0.0;  Vyblock = 0.0;
00030  Vxsubstrate = 0.0; Vysubstrate = 0.0;
00031  double gammav;
00032  gammav = 0.0;
00033
00034  double count_substrate = 0;
00035  double count_block = 0;
00036
00037  for(n = 1 ; n <= nAtom; n ++){
00038   if(atomType[n] == 1 || atomType[n] == 2){
00039   Vxsubstrate += vx[n]; Vysubstrate += vy[n];
00040   count_substrate++;
00041   }
00042   if(atomType[n] == 3 || atomType[n] == 4){
00043   Vxblock += vx[n]; Vyblock += vy[n];
```

```
00044   count_block++;
00045   } }
00046
00047   if(count_substrate > 0) {
00048     Vxsubstrate /= count_substrate;
00049     Vysubstrate /= count_substrate;
00050   }
00051
00052   if(count_block > 0) {
00053    Vxblock /= count_block;
00054    Vyblock /= count_block;
00055   }
00056
00057  for(n = 1 ; n <= nAtom; n ++){
00058   if(atomType[n] == 1 || atomType[n] == 2){
00059   ax[n] += -gammav * (vx[n] - Vxsubstrate);
00060   ay[n] += -gammav * (vy[n] - Vysubstrate);
00061   }
00062   if(atomType[n] == 3 || atomType[n] == 4){
00063   ax[n] += -gammav * (vx[n] - Vxblock);
00064   ay[n] += -gammav * (vy[n] - Vyblock);
00065  } } }
```

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

## 5.11 ApplyDrivingForce.c

Go to the documentation of this file.

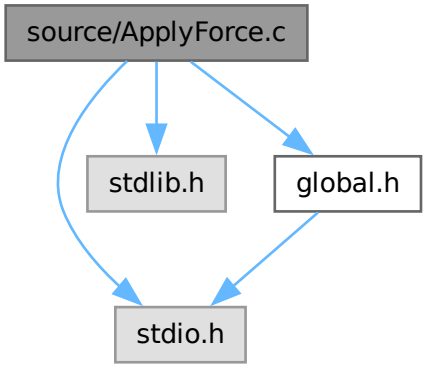
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your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

```
00057  for(n = 1 ; n <= nAtom; n ++){
00058   if(atomType[n] == 1 || atomType[n] == 2){
00059   ax[n] += -gammav * (vx[n] - Vxsubstrate);
00060   ay[n] += -gammav * (vy[n] - Vysubstrate);
00061   }
00062   if(atomType[n] == 3 || atomType[n] == 4){
00063   ax[n] += -gammav * (vx[n] - Vxblock);
00064   ay[n] += -gammav * (vy[n] - Vyblock);
00065  } } }
00066
00067
```

## 5.12 source/ApplyForce.c File Reference

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

Include dependency graph for ApplyForce.c:




### Functions

- void ApplyForce ()

### 5.12.1 Function Documentation

#### 5.12.1.1 ApplyForce()

```
void ApplyForce ()
```

Definition at line 25 of file ApplyForce.c.

```
00025                 {
00026  int n;
00027  double lx;
00028  lx = regionH[1];
00029  fyExtern =  (FyBylx * lx)/nAtomBlock;
00030  fxExtern =  fxByfy * fyExtern;
00031  forceSumxExtern = fxExtern*nAtomBlock;  forceSumyExtern = fyExtern*nAtomBlock;
00032
00033  for(n = 1; n <= nAtom; n ++){
00034  if(molID[n] == 2){
00035   fx[n] += fxExtern;
00036   fy[n] -= fyExtern;
00037 } } }
```

References forceSumxExtern, forceSumyExtern, fx, fxByfy, fxExtern, fy, FyBylx, fyExtern, molID, nAtom, nAtomBlock, and regionH.

Referenced by main().

Here is the caller graph for this function:


## 5.13 ApplyForce.c

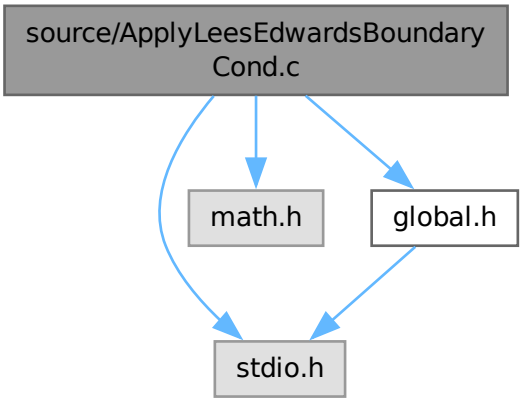
Go to the documentation of this file.

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021 #include<stdio.h>
00022 #include<stdlib.h>
00023 #include"global.h"
00024
00025 void ApplyForce(){
00026  int n;
00027  double lx;
00028  lx = regionH[1];
00029  fyExtern =  (FyBylx * lx)/nAtomBlock;
00030  fxExtern =  fxByfy * fyExtern;
00031  forceSumxExtern = fxExtern*nAtomBlock;  forceSumyExtern = fyExtern*nAtomBlock;
00032
00033  for(n = 1; n <= nAtom; n ++){
00034  if(molID[n] == 2){
00035   fx[n] += fxExtern;
00036   fy[n] -= fyExtern;
00037 } } }
```

## 5.14 source/ApplyLeesEdwardsBoundaryCond.c File Reference

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

Include dependency graph for ApplyLeesEdwardsBoundaryCond.c:




### Functions

- void ApplyLeesEdwardsBoundaryCond ()

## 5.14.1 Function Documentation

### 5.14.1.1 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.

# 5.15 ApplyLeesEdwardsBoundaryCond.c

Go to the documentation of this file.

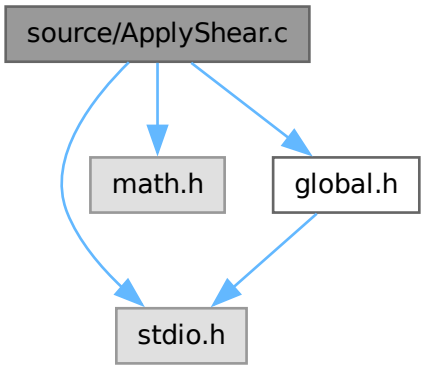
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
```

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

## 5.16 source/ApplyShear.c File Reference

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

Include dependency graph for ApplyShear.c:



**Functions**

- void ApplyShear ()

### 5.16.1 Function Documentation

#### 5.16.1.1 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.

## 5.17 ApplyShear.c

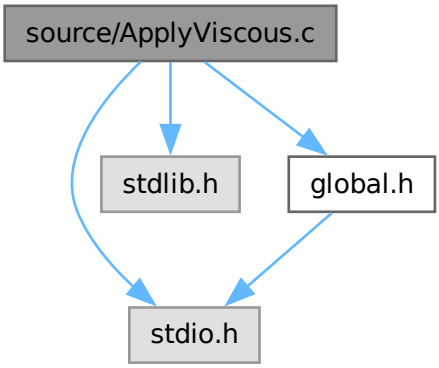
Go to the documentation of this file.

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

## 5.18 source/ApplyViscous.c File Reference

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

Include dependency graph for ApplyViscous.c:




### Functions

- void ApplyViscous ()

## 5.18.1 Function Documentation

### 5.18.1.1 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.

# 5.19 ApplyViscous.c

Go to the documentation of this file.

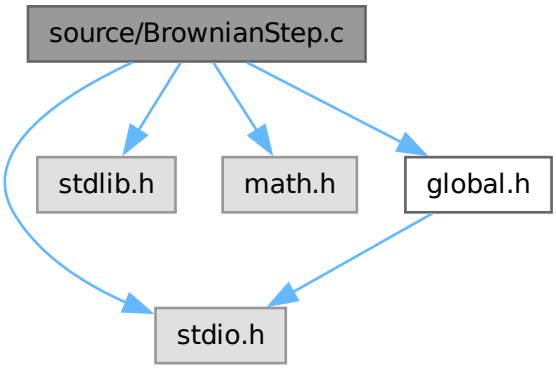
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your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

```
00024
00025 void ApplyViscous(){
00026  int n;
00027  double gammav;
00028  gammav = 1.0;
00029  for(n = 1 ; n <= nAtom; n ++){
00030   ax[n] += -gammav * vx[n];
00031   ay[n] += -gammav * vy[n];
00032  } }
00033
00034
```

## 5.20 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 ()

### 5.20.1 Function Documentation

#### 5.20.1.1 BrownianStep()

```
void BrownianStep ()
```

Definition at line 26 of file BrownianStep.c.

```
00026                   {
00027   if(stepCount <= stepEquil){
00028     double A, S1, S2, T;
00029     int n;
00030     S1 = 0.; S2 = 0;
00031     double halfdt = 0.5*deltaT;
00032     for (n = 1; n <= nAtom; n++){
00033       T = vx[n] + halfdt * ax[n];
00034       S1 += T * ax[n];
00035       S2 += Sqr(T);
00036
00037       T = vy[n] + halfdt * ay[n];
00038       S1 += T * ay[n];
00039       S2 += Sqr(T);
00040     }
00041     A = -S1 / S2;
00042     double C = 1 + A*deltaT ;
```

```
00043     double D = deltaT * (1 + 0.5 * A * deltaT);
00044     for (n = 1; n <= nAtom; n++){
00045       vx[n] = C * vx[n] + D * ax[n];
00046       rx[n] += deltaT * vx[n];
00047       vy[n] = C * vy[n] + D * ay[n];
00048       ry[n] += deltaT * vy[n];
00049     }
00050   }else{
00051       int n;
00052       //SETTING TEMP = 0.0
00053      if (stepCount == stepEquil+1){
00054      for(n = 1 ; n <= nAtom ; n ++){
00055      vx[n] = 0.0;
00056      vy[n] = 0.0;
00057      }}
00058      double zeta = 1.0;
00059      double dx, dy;
00060      for(n = 1 ; n <= nAtom ; n ++){
00061       dx = rx[n];
00062       rx[n] += zeta * ax[n] * deltaT;
00063       dx = rx[n] - dx;
00064       vx[n] = dx/deltaT;
00065       dy = ry[n];
00066       ry[n] += zeta * ay[n] * deltaT;
00067       dy = ry[n] - dy;
00068       vy[n] = dy/deltaT;
00069     }
00070   }
00071 }
```

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

## 5.21 BrownianStep.c

Go to the documentation of this file.

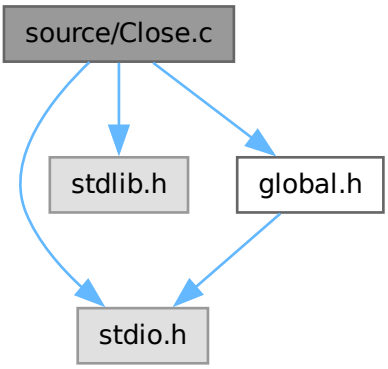
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021 #include<stdio.h>
00022 #include<stdlib.h>
00023 #include<math.h>
00024 #include"global.h"
00025
00026 void BrownianStep(){
00027   if(stepCount <= stepEquil){
00028     double A, S1, S2, T;
00029     int n;
00030     S1 = 0.; S2 = 0;
00031     double halfdt = 0.5*deltaT;
00032     for (n = 1; n <= nAtom; n++){
00033       T = vx[n] + halfdt * ax[n];
00034       S1 += T * ax[n];
00035       S2 += Sqr(T);
00036
00037       T = vy[n] + halfdt * ay[n];
00038       S1 += T * ay[n];
00039       S2 += Sqr(T);
00040     }
00041     A = -S1 / S2;
00042     double C = 1 + A*deltaT ;
00043     double D = deltaT * (1 + 0.5 * A * deltaT);
00044     for (n = 1; n <= nAtom; n++){
00045       vx[n] = C * vx[n] + D * ax[n];
00046       rx[n] += deltaT * vx[n];
00047       vy[n] = C * vy[n] + D * ay[n];
00048       ry[n] += deltaT * vy[n];
00049     }
```

```
00050   }else{
00051       int n;
00052       //SETTING TEMP = 0.0
00053      if (stepCount == stepEquil+1){
00054      for(n = 1 ; n <= nAtom ; n ++){
00055      vx[n] = 0.0;
00056      vy[n] = 0.0;
00057      }}
00058      double zeta = 1.0;
00059      double dx, dy;
00060      for(n = 1 ; n <= nAtom ; n ++){
00061       dx = rx[n];
00062       rx[n] += zeta * ax[n] * deltaT;
00063       dx = rx[n] - dx;
00064       vx[n] = dx/deltaT;
00065       dy = ry[n];
00066       ry[n] += zeta * ay[n] * deltaT;
00067       dy = ry[n] - dy;
00068       vy[n] = dy/deltaT;
00069     }
00070   }
00071 }
00072
```

## 5.22 source/Close.c File Reference

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

Include dependency graph for Close.c:




### Functions

- void Close ()

### 5.22.1 Function Documentation

#### 5.22.1.1 Close()

```
void Close ()
```

Definition at line 25 of file Close.c.

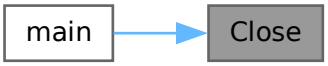
```
00025           {
00026   int n;
00027   free(rx); free(ry); free(vx); free(vy); free(ax); free(ay); free(fx); free(fy);
00028   free(fax);
00029   free(fay);
00030   free(cellList);
00031
```

```
00032   free(atomID); free(atomType); free(atomRadius); free(atomMass);
00033   free(speed);
00034   free(atom1); free(atom2); free(BondID);
00035   free(BondType); free(kb); free(ro);
00036   free(ImageX); free(ImageY); free(rxUnwrap); free(ryUnwrap);
00037   free(atomIDInterface);
00038   free(PairID); free(Pairatom1); free(Pairatom2);
00039   free(PairXij); free(PairYij);
00040   free(molID);
00041
00042   for (n = 0; n <= nAtom; n++) {
00043    free(isBonded[n]);
00044    }
00045   free(isBonded);
00046
00047   for (n = 0; n <= nBuffCorr; n++){
00048   free(cfOrg[n]);
00049   free(spacetimeCorr[n]);
00050   }
00051   free(cfOrg);
00052   free(spacetimeCorr);
00053   free(cfVal);
00054   free(indexCorr);
00055   free(spacetimeCorrAv);
00056
00057   free(indexAcf);
00058   free(viscAcfOrg);
00059   free(viscAcfAv);
00060   for(n = 0 ; n <= nBuffAcf ; n ++)
00061    free(viscAcf[n]);
00062   free(viscAcf);
00063
00064 }
```

References atom1, atom2, atomID, atomIDInterface, atomMass, atomRadius, atomType, ax, ay, BondID, BondType, cellList, cfOrg, cfVal, fax, fay, fx, fy, 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:




## 5.23 Close.c

Go to the documentation of this file.

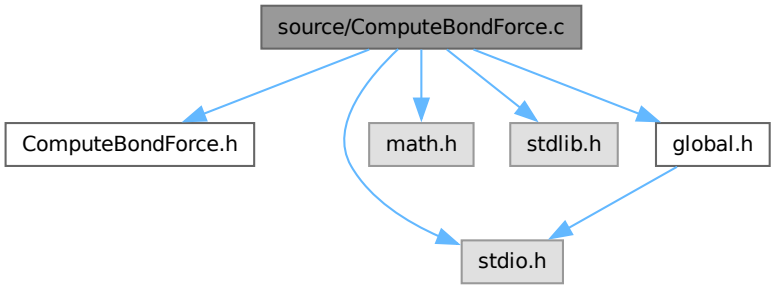
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your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

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

## 5.24 source/ComputeBondForce.c File Reference

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

Include dependency graph for ComputeBondForce.c:




### Functions

- void ComputeBondForce ()

## 5.24.1 Function Documentation

### 5.24.1.1 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
00067    rr = Sqr(dr[1]) + Sqr(dr[2]);
00068    r = sqrt(rr);
00069    rri = 1.0/rr;
00070    ri = 1.0/r;
00071    roi = 1.0/ro[n];
00072    strech = (r * roi - 1.0);
00073    uVal = 0.5 * kb[n] * ro[n] * Sqr(strech);
00074    fcVal = -kb[n] * strech * ri; //F = -Grad U
00075
00076    vr[1] = vx[atom1ID] - vx[atom2ID];
00077    vr[2] = vy[atom1ID] - vy[atom2ID];
00078    fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //node-node drag
00079
00080    //DampFlag = 1. LAMMPS version
00081    if(DampFlag == 1){
00082    nodeDragx[atom1ID] =  fdVal * dr[1];  //node-node drag  //Important change made on 03Apr2025.
      Mention it in GitHub
00083    nodeDragy[atom1ID] =  fdVal * dr[2];  //node-node drag  //Adding the drag forces is wrong. Only add
      the
00084    nodeDragx[atom2ID] = -fdVal * dr[1];  //node-node drag //total force
00085    nodeDragy[atom2ID] = -fdVal * dr[2];  //node-node drag
00086
00087    fx[atom1ID] +=  (fcVal + fdVal) * dr[1];
00088    fy[atom1ID] +=  (fcVal + fdVal) * dr[2];
00089    fx[atom2ID] += -(fcVal + fdVal) * dr[1];
00090    fy[atom2ID] += -(fcVal + fdVal) * dr[2];
00091    }
00092
00093    //DampFlag = 2. Suzanne notes version
00094    else if(DampFlag == 2){
00095    nodeDragx[atom1ID] =  -gamman * vr[1];  //node-node drag
00096    nodeDragy[atom1ID] =  -gamman * vr[2]; //node-node drag
00097    nodeDragx[atom2ID] =  -(-gamman * vr[1]);  //node-node drag
00098    nodeDragy[atom2ID] =  -(-gamman * vr[2]);  //node-node drag
00099
00100    fx[atom1ID] += (fcVal * dr[1] - gamman * vr[1]);
00101    fy[atom1ID] += (fcVal * dr[2] - gamman * vr[2]);
00102    fx[atom2ID] += -(fcVal * dr[1] - gamman * vr[1]);
00103    fy[atom2ID] += -(fcVal * dr[2] - gamman * vr[2]);
00104    }
```

```
00105
00106     BondLength[n] = r;
00107     BondEnergy[n] = uVal; //No 0.5 factor since it is the energy of the bond
00108     TotalBondEnergy   +=  BondEnergy[n];
00109
00110     //Virial and pressure are defnined as per dampFlag = 1
00111     virSumBond +=   0.5 * (fcVal + fdVal) * rr;
00112     virSumBondxx += (fcVal + fdVal) * dr[1] * dr[1]; //Virial term is just r * f
00113     virSumBondyy += (fcVal + fdVal) * dr[2] * dr[2];
00114     virSumBondxy += (fcVal + fdVal) * dr[1] * dr[2];
00115 } }
```

References atom1, atom2, BondEnergy, BondLength, DampFlag, fx, fy, gamman, kb, nAtom, nBond, NDIM, nodeDragx, nodeDragy, region, regionH, ro, rx, ry, shearDisplacement, Sqr, strech, TotalBondEnergy, virSumBond, virSumBondxx, virSumBondxy, virSumBondyy, vx, and vy.

Referenced by main().

Here is the caller graph for this function:


## 5.25 ComputeBondForce.c

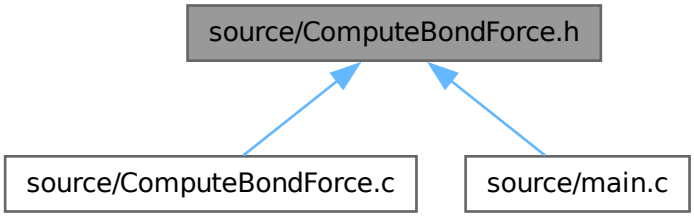
Go to the documentation of this file.

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

```
00046    rr = 0.0; rri = 0.0; fcVal = 0.0;  fdVal = 0.0; 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
00067    rr = Sqr(dr[1]) + Sqr(dr[2]);
00068    r = sqrt(rr);
00069    rri = 1.0/rr;
00070    ri = 1.0/r;
00071    roi = 1.0/ro[n];
00072    strech = (r * roi - 1.0);
00073    uVal = 0.5 * kb[n] * ro[n] * Sqr(strech);
00074    fcVal = -kb[n] * strech * ri; //F = -Grad U
00075
00076    vr[1] = vx[atom1ID] - vx[atom2ID];
00077    vr[2] = vy[atom1ID] - vy[atom2ID];
00078    fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //node-node drag
00079
00080    //DampFlag = 1. LAMMPS version
00081    if(DampFlag == 1){
00082    nodeDragx[atom1ID] =  fdVal * dr[1];  //node-node drag  //Important change made on 03Apr2025.
     Mention it in GitHub
00083    nodeDragy[atom1ID] =  fdVal * dr[2];  //node-node drag  //Adding the drag forces is wrong. Only add
     the
00084    nodeDragx[atom2ID] = -fdVal * dr[1];  //node-node drag //total force
00085    nodeDragy[atom2ID] = -fdVal * dr[2];  //node-node drag
00086
00087    fx[atom1ID] +=  (fcVal + fdVal) * dr[1];
00088    fy[atom1ID] +=  (fcVal + fdVal) * dr[2];
00089    fx[atom2ID] += -(fcVal + fdVal) * dr[1];
00090    fy[atom2ID] += -(fcVal + fdVal) * dr[2];
00091    }
00092
00093    //DampFlag = 2. Suzanne notes version
00094    else if(DampFlag == 2){
00095    nodeDragx[atom1ID] =  -gamman * vr[1];  //node-node drag
00096    nodeDragy[atom1ID] =  -gamman * vr[2]; //node-node drag
00097    nodeDragx[atom2ID] =  -(-gamman * vr[1]);  //node-node drag
00098    nodeDragy[atom2ID] =  -(-gamman * vr[2]);  //node-node drag
00099
00100    fx[atom1ID] += (fcVal * dr[1] - gamman * vr[1]);
00101    fy[atom1ID] += (fcVal * dr[2] - gamman * vr[2]);
00102    fx[atom2ID] += -(fcVal * dr[1] - gamman * vr[1]);
00103    fy[atom2ID] += -(fcVal * dr[2] - gamman * vr[2]);
00104    }
00105
00106    BondLength[n] = r;
00107    BondEnergy[n] = uVal; //No 0.5 factor since it is the energy of the bond
00108    TotalBondEnergy   +=  BondEnergy[n];
00109
00110    //Virial and pressure are defnined as per dampFlag = 1
00111    virSumBond +=   0.5 * (fcVal + fdVal) * rr;
00112    virSumBondxx += (fcVal + fdVal) * dr[1] * dr[1]; //Virial term is just r * f
00113    virSumBondyy += (fcVal + fdVal) * dr[2] * dr[2];
00114    virSumBondxy += (fcVal + fdVal) * dr[1] * dr[2];
00115 } }
```

# 5.26 source/ComputeBondForce.h File Reference

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




### Functions

- void ComputeBondForce ()

## 5.26.1 Function Documentation

### 5.26.1.1 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
00056    dr[2] = ry[atom1ID] - ry[atom2ID];
00057    if(dr[2] >= regionH[2]){
00058     dr[1] -= shearDisplacement;
00059     if(dr[1] < -regionH[1]) dr[1] += region[1];
00060     dr[2] -= region[2];
00061   }else if(dr[2] < -regionH[2]){
00062    dr[1] += shearDisplacement;
00063     if(dr[1] >= regionH[1]) dr[1] -= region[1];
00064    dr[2] += region[2];
00065   }
00066
00067    rr = Sqr(dr[1]) + Sqr(dr[2]);
00068    r = sqrt(rr);
00069    rri = 1.0/rr;
00070    ri = 1.0/r;
00071    roi = 1.0/ro[n];
```

```
00072    strech = (r * roi - 1.0);
00073    uVal = 0.5 * kb[n] * ro[n] * Sqr(strech);
00074    fcVal = -kb[n] * strech * ri; //F = -Grad U
00075
00076    vr[1] = vx[atom1ID] - vx[atom2ID];
00077    vr[2] = vy[atom1ID] - vy[atom2ID];
00078    fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //node-node drag
00079
00080    //DampFlag = 1. LAMMPS version
00081    if(DampFlag == 1){
00082    nodeDragx[atom1ID] =  fdVal * dr[1];  //node-node drag  //Important change made on 03Apr2025.
      Mention it in GitHub
00083    nodeDragy[atom1ID] =  fdVal * dr[2];  //node-node drag  //Adding the drag forces is wrong. Only add
      the
00084    nodeDragx[atom2ID] = -fdVal * dr[1];  //node-node drag //total force
00085    nodeDragy[atom2ID] = -fdVal * dr[2];  //node-node drag
00086
00087    fx[atom1ID] +=  (fcVal + fdVal) * dr[1];
00088    fy[atom1ID] +=  (fcVal + fdVal) * dr[2];
00089    fx[atom2ID] += -(fcVal + fdVal) * dr[1];
00090    fy[atom2ID] += -(fcVal + fdVal) * dr[2];
00091    }
00092
00093    //DampFlag = 2. Suzanne notes version
00094    else if(DampFlag == 2){
00095    nodeDragx[atom1ID] =  -gamman * vr[1];  //node-node drag
00096    nodeDragy[atom1ID] =  -gamman * vr[2]; //node-node drag
00097    nodeDragx[atom2ID] =  -(-gamman * vr[1]);  //node-node drag
00098    nodeDragy[atom2ID] =  -(-gamman * vr[2]);  //node-node drag
00099
00100    fx[atom1ID] += (fcVal * dr[1] - gamman * vr[1]);
00101    fy[atom1ID] += (fcVal * dr[2] - gamman * vr[2]);
00102    fx[atom2ID] += -(fcVal * dr[1] - gamman * vr[1]);
00103    fy[atom2ID] += -(fcVal * dr[2] - gamman * vr[2]);
00104    }
00105
00106    BondLength[n] = r;
00107    BondEnergy[n] = uVal; //No 0.5 factor since it is the energy of the bond
00108    TotalBondEnergy   +=  BondEnergy[n];
00109
00110    //Virial and pressure are defnined as per dampFlag = 1
00111    virSumBond +=   0.5 * (fcVal + fdVal) * rr;
00112    virSumBondxx += (fcVal + fdVal) * dr[1] * dr[1]; //Virial term is just r * f
00113    virSumBondyy += (fcVal + fdVal) * dr[2] * dr[2];
00114    virSumBondxy += (fcVal + fdVal) * dr[1] * dr[2];
00115 } }
```

References atom1, atom2, BondEnergy, BondLength, DampFlag, fx, fy, gamman, kb, nAtom, nBond, NDIM, nodeDragx, nodeDragy, region, regionH, ro, rx, ry, shearDisplacement, Sqr, strech, TotalBondEnergy, virSumBond, virSumBondxx, virSumBondxy, virSumBondyy, vx, and vy.

Referenced by main().

Here is the caller graph for this function:


## 5.27 ComputeBondForce.h

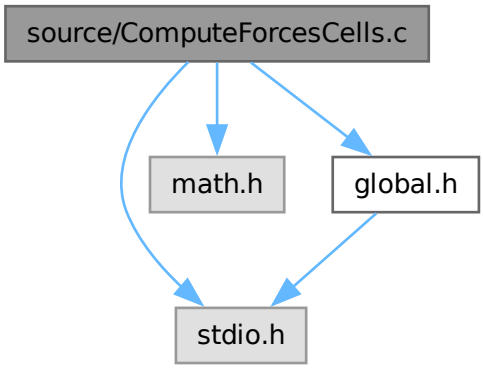
Go to the documentation of this file.

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

## 5.28 source/ComputeForcesCells.c File Reference

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

Include dependency graph for ComputeForcesCells.c:




### Functions

- void ComputeForcesCells ()

### 5.28.1 Function Documentation

#### 5.28.1.1 ComputeForcesCells()

```
void ComputeForcesCells ()
```

Definition at line 25 of file ComputeForcesCells.c.

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

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

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

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

## 5.29 ComputeForcesCells.c

Go to the documentation of this file.

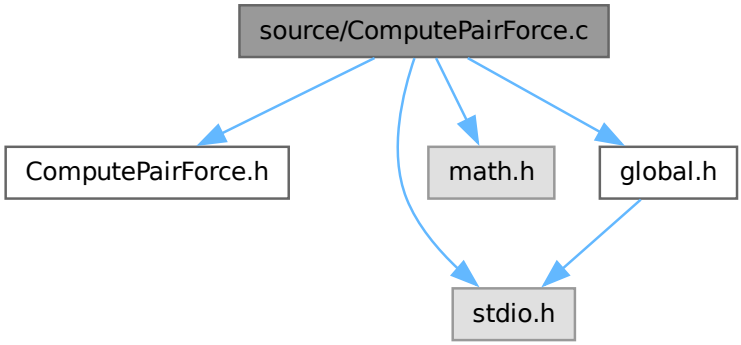
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021 #include<stdio.h>
00022 #include<math.h>
00023 #include"global.h"
00024
00025 void ComputeForcesCells(){
00026   double dr[NDIM+1], invWid[NDIM+1], shift[NDIM+1], f, fcVal, rr, ri, r, uVal;
00027   int c, I, J, m1, m1X, m1Y, m2, m2X, m2Y, n, offset;
00028   int iofX[] = {0, 0, 1, 1, 0, -1, -1, -1, 0, 1},
00029       iofY[] = {0, 0, 0, 1 ,1, 1, 0, -1, -1, -1};
00030
00031   invWid[1] = cells[1]/region[1];
00032   invWid[2] = cells[2]/region[2];
00033
00034   for(n = nAtom+1; n <= nAtom+cells[1]*cells[2] ; n++)
00035     cellList[n] = 0;
00036
00037   for(n = 1 ; n <= nAtom ; n ++){
00038     c = ((int)((ry[n] + regionH[2])*invWid[2]))*cells[1] + (int)((rx[n]+regionH[1])*invWid[1]) +
    nAtom+ 1;
00039     cellList[n] = cellList[c];
00040     cellList[c] = n;
00041   }
00042
00043   for(n = 1 ; n <= nAtom ; n ++){
00044     ax[n] = 0.;
00045     ay[n] = 0.;
00046   }
00047
00048   uSum = 0.0 ;
00049   virSum = 0.0;
00050   rfAtom = 0.0;
00051   RadiusIJ = 0.0;
00052
00053   gamman = 1.0;
00054   double vr[NDIM+1], fd, fdVal, rrinv;
00055   rrinv = 0.0;
00056   fd = 0.0;
00057   fdVal = 0.0;
00058
00059   int start = 1 + rank*(cells[2]/size);
00060   int end = (rank+1)*(cells[2]/size);
00061
00062   for(m1Y = start ; m1Y <= end ; m1Y ++){
00063     for(m1X = 1 ; m1X <= cells[1] ; m1X ++){
00064       m1 = (m1Y-1) * cells[1] + m1X + nAtom;
00065       for(offset = 1 ; offset <= 9 ; offset ++){
00066     m2X = m1X + iofX[offset]; shift[1] = 0.;
```

```
00067     if(m2X > cells[1]){
00068       m2X = 1; shift[1] = region[1];
00069     }else if(m2X == 0){
00070       m2X = cells[1]; shift[1] = -region[1];
00071     }
00072     m2Y = m1Y + iofY[offset]; shift[2] = 0.;
00073     if(m2Y > cells[2]){
00074       m2Y = 1; shift[2] = region[2];
00075     }else if(m2Y == 0){
00076       m2Y = cells[2]; shift[2] = -region[2];
00077     }
00078     m2 = (m2Y-1)*cells[1] + m2X + nAtom;
00079     I = cellList[m1];
00080     while(I > 0){
00081       J = cellList[m2];
00082       while(J > 0){
00083         if(m1 == m2 && J != I && (atomRadius[I] > 0. && atomRadius[J] > 0.)){
00084           dr[1] = rx[I] - rx[J] - shift[1];
00085           dr[2] = ry[I] - ry[J] - shift[2];
00086           rr = Sqr(dr[1]) + Sqr(dr[2]);
00087           RadiusIJ = atomRadius[I] + atomRadius[J];
00088           SqrRadiusIJ = Sqr(RadiusIJ);
00089           if(rr < SqrRadiusIJ){
00090         r = sqrt(rr);
00091         ri = 1.0/r;
00092                 rrinv = 1.0/rr;
00093                 vr[1] = vx[I] - vx[J];
00094                 vr[2] = vy[I] - vy[J];
00095         RadiusIJInv = 1.0/RadiusIJ;
00096         uVal = Sqr(1.0 - r * RadiusIJInv);
00097         fcVal = 2.0 * RadiusIJInv * (1.0 - r * RadiusIJInv) *ri;
00098                 fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag
00099
00100         f  = fcVal * dr[1];
00101                 fd = fdVal * dr[1];
00102         ax[I] += (f + fd);
00103                 discDragx[I] += fd; //disc-disc drag
00104
00105         f = fcVal * dr[2];
00106                 fd = fdVal * dr[2];
00107         ay[I] += (f + fd);
00108                 discDragy[I] += fd; //disc-disc drag
00109
00110         uSum +=  0.5 * uVal;
00111         virSum += 0.5 * fcVal * rr;
00112         rfAtom += 0.5 * dr[1] * fcVal * dr[2];
00113           }
00114         }else if(m1 != m2 && (atomRadius[I] > 0. && atomRadius[J] > 0.)){
00115           dr[1] = rx[I] - rx[J] - shift[1];
00116           dr[2] = ry[I] - ry[J] - shift[2];
00117           rr = Sqr(dr[1]) + Sqr(dr[2]);
00118           RadiusIJ = atomRadius[I] + atomRadius[J];
00119           SqrRadiusIJ = Sqr(RadiusIJ);
00120           if(rr < SqrRadiusIJ){
00121         r = sqrt(rr);
00122         ri = 1.0/r;
00123                 rrinv = 1.0/r;
00124                 vr[1] = vx[I] - vx[J];
00125                 vr[2] = vy[I] - vy[J];
00126         RadiusIJInv = 1.0/RadiusIJ;
00127         uVal = Sqr(1.0 - r * RadiusIJInv);
00128         fcVal = 2.0 * RadiusIJInv * (1.0 - r * RadiusIJInv) *ri;
00129                 fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag
00130
00131         f  = fcVal * dr[1];
00132                 fd =  fdVal * dr[1];
00133         ax[I] += (f + fd);
00134                 discDragx[I] += fd; //disc-disc drag
00135
00136         f  = fcVal * dr[2];
00137                 fd = fdVal * dr[2];
00138         ay[I] += (f + fd);
00139                 discDragy[I] += fd; //disc-disc drag
00140
00141         uSum +=  0.5 * uVal;
00142         virSum += 0.5 * fcVal * rr;
00143         rfAtom += 0.5 * dr[1] * fcVal * dr[2];
00144           }
00145         }
00146             J = cellList[J];
00147       }
00148       I = cellList[I];
00149     }
00150       }
00151     }
00152   }
00153 }
```

## 5.30 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)

### 5.30.1 Function Documentation

#### 5.30.1.1 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  fx[n] = 0.0;
00035  fy[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 double vr[NDIM+1], fdVal, rri;
00048 nPairActive = 0;
00049 double meff;
00050 meff = 0.0;
00051 int atomIDi, atomIDj;
00052 double atomiMass, atomjMass;
00053
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
00061   if (isBonded[atomIDi][atomIDj] == 0) { //To have pair interaction between nonbonded atoms only
00062   rr = 0.0; rri = 0.0; fcVal = 0.0;  fdVal = 0.0; strech = 0.0;
00063   RadiusIJ = 0.0;
00064
00065   dr[1] = rx[atomIDi] - rx[atomIDj];
00066   if(dr[1] >= regionH[1])
00067    dr[1] -= region[1];
00068   else if(dr[1] < -regionH[1])
00069     dr[1] += region[1];
00070
00071   dr[2] = ry[atomIDi] - ry[atomIDj];
00072    if(dr[2] >= regionH[2]){
00073     dr[1] -= shearDisplacement;
00074     if(dr[1] < -regionH[1]) dr[1] += region[1];
00075     dr[2] -= region[2];
00076   }else if(dr[2] < -regionH[2]){
00077    dr[1] += shearDisplacement;
00078     if(dr[1] >= regionH[1]) dr[1] -= region[1];
00079    dr[2] += region[2];
00080   }
00081
00082   rr = Sqr(dr[1]) + Sqr(dr[2]);
00083   RadiusIJ = atomRadius[atomIDi] + atomRadius[atomIDj];
00084   SqrRadiusIJ = Sqr(RadiusIJ);
00085   if(rr < SqrRadiusIJ){
00086    r = sqrt(rr);
00087    ri = 1.0/r;
00088    rri = 1.0/rr;
00089    RadiusIJInv = 1.0/RadiusIJ;
00090    strech = (RadiusIJ - r);
00091    uVal = 0.5 * Kn * Sqr(strech);
00092
00093    //NormFlag
00094    if(normFlag == 1){
00095     strech = strech * RadiusIJInv;
00096     uVal = 0.5 * Kn * RadiusIJ * Sqr(strech);
00097     }
00098
00099    fcVal = Kn * strech * ri;
00100    vr[1] = vx[atomIDi] - vx[atomIDj];
00101    vr[2] = vy[atomIDi] - vy[atomIDj];
00102
00103    nPairActive++;
00104    PairID[nPairActive] = nPairActive;
00105    Pairatom1[nPairActive] = atomIDi;
00106    Pairatom2[nPairActive] = atomIDj;
00107    PairXij[nPairActive] = dr[1];
00108    PairYij[nPairActive] = dr[2];
00109
00110    //DampFlag = 1
00111    if(DampFlag == 1){
00112    atomiMass = atomMass[atomIDi];
00113    atomjMass = atomMass[atomIDj];
00114    meff = (atomiMass * atomjMass)/(atomiMass + atomjMass);
00115    fdVal = -gamman * meff * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //disc-disc drag
00116
00117    discDragx[atomIDi] =  fdVal * dr[1]; //disc-disc drag
00118    discDragy[atomIDi] =  fdVal * dr[2]; //disc-disc drag
00119    discDragx[atomIDj] = -fdVal * dr[1]; //disc-disc drag
00120    discDragy[atomIDj] = -fdVal * dr[2]; //disc-disc drag
00121
00122    discDragx[nPairActive] = discDragx[atomIDi];
00123    discDragy[nPairActive] = discDragy[atomIDi];
00124
00125
00126    fx[atomIDi] +=  (fcVal + fdVal) * dr[1];
00127    fy[atomIDi] +=  (fcVal + fdVal) * dr[2];
00128    fx[atomIDj] += -(fcVal + fdVal) * dr[1];
00129    fy[atomIDj] += -(fcVal + fdVal) * dr[2];
00130   }
00131
00132   //DampFlag = 2
00133   else if(DampFlag == 2){
00134    discDragx[atomIDi] =  -gamman * vr[1]; //disc-disc drag
00135    discDragy[atomIDi] =  -gamman * vr[2]; //disc-disc drag
00136    discDragx[atomIDj] = -(-gamman * vr[1]); //disc-disc drag
00137    discDragy[atomIDj] = -(-gamman * vr[2]); //disc-disc drag
00138
00139    discDragx[nPairActive] = discDragx[atomIDi];
00140    discDragy[nPairActive] = discDragy[atomIDi];
00141
00142    fx[atomIDi] +=  (fcVal * dr[1] - gamman * vr[1]);
00143    fy[atomIDi] +=  (fcVal * dr[2] - gamman * vr[2]);
00144    fx[atomIDj] += -(fcVal * dr[1] - gamman * vr[1]);
00145    fy[atomIDj] += -(fcVal * dr[2] - gamman * vr[2]);
```

```
00146   }
00147
00148   //In the following, for stress/virial term (fcVal + fdVal) is used since the total pair force =
      Hookean Interaction + relative velocity drag
00149   uSumPair +=  0.5 * uVal;
00150   virSumPair += (fcVal + fdVal) * rr;
00151   virSumPairxx += (fcVal + fdVal) * dr[1] * dr[1];
00152   virSumPairyy += (fcVal + fdVal) * dr[2] * dr[2];
00153   virSumPairxy += (fcVal + fdVal) * dr[1] * dr[2];
00154     }
00155    }
00156   }
00157  }
00158 }
```

References atomIDInterface, atomMass, atomRadius, DampFlag, discDragx, discDragy, fx, fy, gamman, isBonded, Kn, nAtom, nAtomInterface, NDIM, nPairActive, nPairTotal, Pairatom1, Pairatom2, PairID, PairXij, PairYij, RadiusIJ, RadiusIJInv, region, regionH, rx, ry, shearDisplacement, Sqr, SqrRadiusIJ, strech, uSumPair, virSumPair, virSumPairxx, virSumPairxy, virSumPairyy, vx, and vy.

Referenced by main().

Here is the caller graph for this function:


## 5.31 ComputePairForce.c

Go to the documentation of this file.

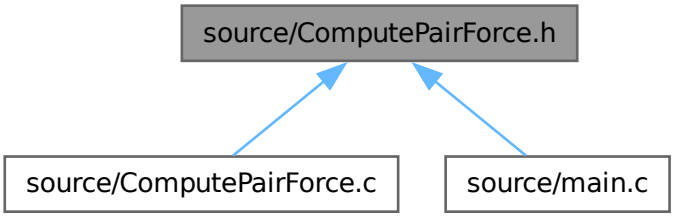
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021 #include "ComputePairForce.h"
00022
00023 #include<stdio.h>
00024 #include<math.h>
00025 #include"global.h"
00026
00027 void ComputePairForce(int normFlag){
00028 double dr[NDIM+1], fcVal, rr, ri, r, uVal;
00029 int n, i,j;
00030 uVal = 0.0;  uSumPair = 0.0 ;
00031 virSumPair = 0.0; virSumPairxx = 0.0; virSumPairyy = 0.0; virSumPairxy = 0.0;
00032
00033 for(n = 1 ; n <= nAtom ; n ++){
00034  fx[n] = 0.0;
00035  fy[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 double vr[NDIM+1], fdVal, rri;
00048 nPairActive = 0;
00049 double meff;
00050 meff = 0.0;
00051 int atomIDi, atomIDj;
00052 double atomiMass, atomjMass;
00053
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
00061   if (isBonded[atomIDi][atomIDj] == 0) { //To have pair interaction between nonbonded atoms only
00062   rr = 0.0; rri = 0.0; fcVal = 0.0;  fdVal = 0.0; strech = 0.0;
00063   RadiusIJ = 0.0;
00064
00065   dr[1] = rx[atomIDi] - rx[atomIDj];
00066   if(dr[1] >= regionH[1])
00067    dr[1] -= region[1];
00068   else if(dr[1] < -regionH[1])
00069     dr[1] += region[1];
00070
00071   dr[2] = ry[atomIDi] - ry[atomIDj];
00072    if(dr[2] >= regionH[2]){
00073     dr[1] -= shearDisplacement;
00074     if(dr[1] < -regionH[1]) dr[1] += region[1];
00075     dr[2] -= region[2];
00076   }else if(dr[2] < -regionH[2]){
00077    dr[1] += shearDisplacement;
00078     if(dr[1] >= regionH[1]) dr[1] -= region[1];
00079    dr[2] += region[2];
00080   }
00081
00082   rr = Sqr(dr[1]) + Sqr(dr[2]);
00083   RadiusIJ = atomRadius[atomIDi] + atomRadius[atomIDj];
00084   SqrRadiusIJ = Sqr(RadiusIJ);
00085   if(rr < SqrRadiusIJ){
00086    r = sqrt(rr);
00087    ri = 1.0/r;
00088    rri = 1.0/rr;
00089    RadiusIJInv = 1.0/RadiusIJ;
00090    strech = (RadiusIJ - r);
00091    uVal = 0.5 * Kn * Sqr(strech);
00092
00093    //NormFlag
00094    if(normFlag == 1){
00095     strech = strech * RadiusIJInv;
00096     uVal = 0.5 * Kn * RadiusIJ * Sqr(strech);
00097     }
00098
00099    fcVal = Kn * strech * ri;
00100    vr[1] = vx[atomIDi] - vx[atomIDj];
00101    vr[2] = vy[atomIDi] - vy[atomIDj];
00102
00103    nPairActive++;
00104    PairID[nPairActive] = nPairActive;
00105    Pairatom1[nPairActive] = atomIDi;
00106    Pairatom2[nPairActive] = atomIDj;
00107    PairXij[nPairActive] = dr[1];
00108    PairYij[nPairActive] = dr[2];
00109
00110    //DampFlag = 1
00111    if(DampFlag == 1){
00112    atomiMass = atomMass[atomIDi];
00113    atomjMass = atomMass[atomIDj];
00114    meff = (atomiMass * atomjMass)/(atomiMass + atomjMass);
00115    fdVal = -gamman * meff * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //disc-disc drag
00116
00117    discDragx[atomIDi] =  fdVal * dr[1]; //disc-disc drag
00118    discDragy[atomIDi] =  fdVal * dr[2]; //disc-disc drag
00119    discDragx[atomIDj] = -fdVal * dr[1]; //disc-disc drag
00120    discDragy[atomIDj] = -fdVal * dr[2]; //disc-disc drag
00121
00122    discDragx[nPairActive] = discDragx[atomIDi];
00123    discDragy[nPairActive] = discDragy[atomIDi];
00124
00125
00126    fx[atomIDi] +=  (fcVal + fdVal) * dr[1];
00127    fy[atomIDi] +=  (fcVal + fdVal) * dr[2];
00128    fx[atomIDj] += -(fcVal + fdVal) * dr[1];
```

```
00129     fy[atomIDj] += -(fcVal + fdVal) * dr[2];
00130    }
00131
00132    //DampFlag = 2
00133    else if(DampFlag == 2){
00134     discDragx[atomIDi] =  -gamman * vr[1]; //disc-disc drag
00135     discDragy[atomIDi] =  -gamman * vr[2]; //disc-disc drag
00136     discDragx[atomIDj] = -(-gamman * vr[1]); //disc-disc drag
00137     discDragy[atomIDj] = -(-gamman * vr[2]); //disc-disc drag
00138
00139     discDragx[nPairActive] = discDragx[atomIDi];
00140     discDragy[nPairActive] = discDragy[atomIDi];
00141
00142     fx[atomIDi] +=  (fcVal * dr[1] - gamman * vr[1]);
00143     fy[atomIDi] +=  (fcVal * dr[2] - gamman * vr[2]);
00144     fx[atomIDj] += -(fcVal * dr[1] - gamman * vr[1]);
00145     fy[atomIDj] += -(fcVal * dr[2] - gamman * vr[2]);
00146    }
00147
00148    //In the following, for stress/virial term (fcVal + fdVal) is used since the total pair force =
      Hookean Interaction + relative velocity drag
00149    uSumPair +=  0.5 * uVal;
00150    virSumPair += (fcVal + fdVal) * rr;
00151    virSumPairxx += (fcVal + fdVal) * dr[1] * dr[1];
00152    virSumPairyy += (fcVal + fdVal) * dr[2] * dr[2];
00153    virSumPairxy += (fcVal + fdVal) * dr[1] * dr[2];
00154      }
00155     }
00156    }
00157   }
00158 }
00159
00160
00161
```

# 5.32 source/ComputePairForce.h File Reference

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




### Functions

- void ComputePairForce (int normFlag)

## 5.32.1 Function Documentation

### 5.32.1.1 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  fx[n] = 0.0;
00035  fy[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 double vr[NDIM+1], fdVal, rri;
00048 nPairActive = 0;
00049 double meff;
00050 meff = 0.0;
00051 int atomIDi, atomIDj;
00052 double atomiMass, atomjMass;
00053
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
00061   if (isBonded[atomIDi][atomIDj] == 0) { //To have pair interaction between nonbonded atoms only
00062   rr = 0.0; rri = 0.0; fcVal = 0.0;  fdVal = 0.0; strech = 0.0;
00063   RadiusIJ = 0.0;
00064
00065   dr[1] = rx[atomIDi] - rx[atomIDj];
00066   if(dr[1] >= regionH[1])
00067    dr[1] -= region[1];
00068   else if(dr[1] < -regionH[1])
00069     dr[1] += region[1];
00070
00071   dr[2] = ry[atomIDi] - ry[atomIDj];
00072    if(dr[2] >= regionH[2]){
00073     dr[1] -= shearDisplacement;
00074     if(dr[1] < -regionH[1]) dr[1] += region[1];
00075     dr[2] -= region[2];
00076   }else if(dr[2] < -regionH[2]){
00077    dr[1] += shearDisplacement;
00078     if(dr[1] >= regionH[1]) dr[1] -= region[1];
00079    dr[2] += region[2];
00080   }
00081
00082   rr = Sqr(dr[1]) + Sqr(dr[2]);
00083   RadiusIJ = atomRadius[atomIDi] + atomRadius[atomIDj];
00084   SqrRadiusIJ = Sqr(RadiusIJ);
00085   if(rr < SqrRadiusIJ){
00086    r = sqrt(rr);
00087    ri = 1.0/r;
00088    rri = 1.0/rr;
00089    RadiusIJInv = 1.0/RadiusIJ;
00090    strech = (RadiusIJ - r);
00091    uVal = 0.5 * Kn * Sqr(strech);
00092
00093    //NormFlag
00094    if(normFlag == 1){
00095     strech = strech * RadiusIJInv;
00096     uVal = 0.5 * Kn * RadiusIJ * Sqr(strech);
00097     }
00098
00099    fcVal = Kn * strech * ri;
00100    vr[1] = vx[atomIDi] - vx[atomIDj];
00101    vr[2] = vy[atomIDi] - vy[atomIDj];
00102
00103    nPairActive++;
00104    PairID[nPairActive] = nPairActive;
00105    Pairatom1[nPairActive] = atomIDi;
00106    Pairatom2[nPairActive] = atomIDj;
00107    PairXij[nPairActive] = dr[1];
00108    PairYij[nPairActive] = dr[2];
00109
00110    //DampFlag = 1
00111    if(DampFlag == 1){
00112    atomiMass = atomMass[atomIDi];
00113    atomjMass = atomMass[atomIDj];
00114    meff = (atomiMass * atomjMass)/(atomiMass + atomjMass);
00115    fdVal = -gamman * meff * (vr[1]*dr[1] + vr[2]*dr[2]) * rri; //disc-disc drag
00116
00117    discDragx[atomIDi] =  fdVal * dr[1]; //disc-disc drag
00118    discDragy[atomIDi] =  fdVal * dr[2]; //disc-disc drag
00119    discDragx[atomIDj] = -fdVal * dr[1]; //disc-disc drag
```

```
00120    discDragy[atomIDj] = -fdVal * dr[2]; //disc-disc drag
00121
00122    discDragx[nPairActive] = discDragx[atomIDi];
00123    discDragy[nPairActive] = discDragy[atomIDi];
00124
00125
00126    fx[atomIDi] +=  (fcVal + fdVal) * dr[1];
00127    fy[atomIDi] +=  (fcVal + fdVal) * dr[2];
00128    fx[atomIDj] += -(fcVal + fdVal) * dr[1];
00129    fy[atomIDj] += -(fcVal + fdVal) * dr[2];
00130   }
00131
00132   //DampFlag = 2
00133   else if(DampFlag == 2){
00134    discDragx[atomIDi] =  -gamman * vr[1]; //disc-disc drag
00135    discDragy[atomIDi] =  -gamman * vr[2]; //disc-disc drag
00136    discDragx[atomIDj] = -(-gamman * vr[1]); //disc-disc drag
00137    discDragy[atomIDj] = -(-gamman * vr[2]); //disc-disc drag
00138
00139    discDragx[nPairActive] = discDragx[atomIDi];
00140    discDragy[nPairActive] = discDragy[atomIDi];
00141
00142    fx[atomIDi] +=  (fcVal * dr[1] - gamman * vr[1]);
00143    fy[atomIDi] +=  (fcVal * dr[2] - gamman * vr[2]);
00144    fx[atomIDj] += -(fcVal * dr[1] - gamman * vr[1]);
00145    fy[atomIDj] += -(fcVal * dr[2] - gamman * vr[2]);
00146   }
00147
00148   //In the following, for stress/virial term (fcVal + fdVal) is used since the total pair force =
      Hookean Interaction + relative velocity drag
00149   uSumPair +=  0.5 * uVal;
00150   virSumPair += (fcVal + fdVal) * rr;
00151   virSumPairxx += (fcVal + fdVal) * dr[1] * dr[1];
00152   virSumPairyy += (fcVal + fdVal) * dr[2] * dr[2];
00153   virSumPairxy += (fcVal + fdVal) * dr[1] * dr[2];
00154     }
00155    }
00156   }
00157  }
00158 }
```

References atomIDInterface, atomMass, atomRadius, DampFlag, discDragx, discDragy, fx, fy, gamman, isBonded, Kn, nAtom, nAtomInterface, NDIM, nPairActive, nPairTotal, Pairatom1, Pairatom2, PairID, PairXij, PairYij, RadiusIJ, RadiusIJInv, region, regionH, rx, ry, shearDisplacement, Sqr, SqrRadiusIJ, strech, uSumPair, virSumPair, virSumPairxx, virSumPairxy, virSumPairyy, vx, and vy.

Referenced by main().

Here is the caller graph for this function:


## 5.33 ComputePairForce.h

Go to the documentation of this file.

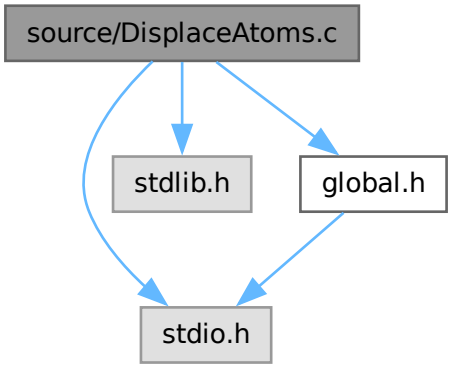
```
00001 #ifndef COMPUTE_PAIR_FORCE_H
00002 #define COMPUTE_PAIR_FORCE_H
00003
00004 void ComputePairForce(int normFlag);
00005
00006 #endif
00007
```

## 5.34 source/DisplaceAtoms.c File Reference

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

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

Include dependency graph for DisplaceAtoms.c:




**Functions**

- void DisplaceAtoms ()

### 5.34.1 Function Documentation

#### 5.34.1.1 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:


## 5.35 DisplaceAtoms.c

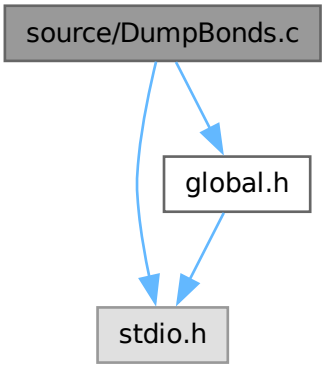
Go to the documentation of this file.

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
```

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

## 5.36 source/DumpBonds.c File Reference

`#include <stdio.h>`
`#include "global.h"`
Include dependency graph for DumpBonds.c:




### Functions

- void DumpBonds ()

### 5.36.1 Function Documentation

#### 5.36.1.1 DumpBonds()

`void DumpBonds ()`
Definition at line 24 of file DumpBonds.c.

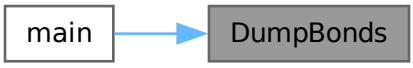
```
00024                {
00025   int n;
00026   //Trajectory file in LAMMPS dump format for OVITO visualization
00027   fprintf(fpbond, "ITEM: TIMESTEP\n");
```

```
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:




## 5.37 DumpBonds.c

Go to the documentation of this file.

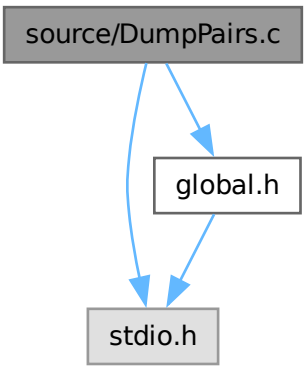
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021 #include<stdio.h>
00022 #include"global.h"
00023
00024 void DumpBonds(){
00025   int n;
00026   //Trajectory file in LAMMPS dump format for OVITO visualization
00027   fprintf(fpbond, "ITEM: TIMESTEP\n");
00028   fprintf(fpbond, "%lf\n",timeNow);
00029   fprintf(fpbond, "ITEM: NUMBER OF ENTRIES\n");
00030   fprintf(fpbond, "%d\n",nBond);
00031   fprintf(fpbond, "ITEM: BOX BOUNDS pp ff pp\n");
00032   fprintf(fpbond, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
00033   fprintf(fpbond, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
00034   fprintf(fpbond, "%lf %lf zlo zhi\n", -0.1, 0.1);
00035   fprintf(fpbond, "ITEM: ENTRIES BondID, BondType, atom1 atom2 BondLength BondLengthEqul nodeDragx1
      nodeDragy1\n");
00036
00037   for(n=1; n<=nBond; n++)
00038    fprintf(fpbond, "%d %d %d %d %0.16lf %0.16lf %0.16lf %0.16lf\n", BondID[n], BondType[n], atom1[n],
      atom2[n],
00039    BondLength[n], ro[n], nodeDragx[atom1[n]], nodeDragy[atom1[n]]);
00040   }
00041
```

```
00042
00043
```

## 5.38 source/DumpPairs.c File Reference

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

Include dependency graph for DumpPairs.c:




### Functions

- void DumpPairs ()

### 5.38.1 Function Documentation

#### 5.38.1.1 DumpPairs()

```
void DumpPairs ()
```

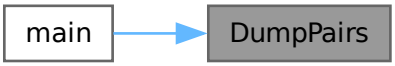
Definition at line 25 of file DumpPairs.c.

```
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:




## 5.39 DumpPairs.c

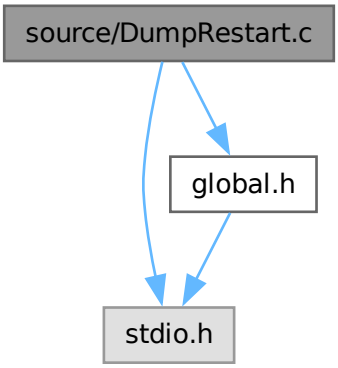
Go to the documentation of this file.

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 void DumpPairs(){
00026   int n;
00027   //Trajectory file in LAMMPS dump format for OVITO visualization
00028   fprintf(fppair, "ITEM: TIMESTEP\n");
00029   fprintf(fppair, "%lf\n",timeNow);
00030   fprintf(fppair, "ITEM: NUMBER OF ENTRIES\n");
00031   fprintf(fppair, "%d\n",nPairActive);
00032   fprintf(fppair, "ITEM: BOX BOUNDS pp ff pp\n");
00033   fprintf(fppair, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
00034   fprintf(fppair, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
00035   fprintf(fppair, "%lf %lf zlo zhi\n", -0.1, 0.1);
00036   fprintf(fppair, "ITEM: ENTRIES index, atom1 atom2 xij yij discDragx1 discDragy1\n");
00037
00038   for(n=1; n<=nPairActive; n++)
00039    fprintf(fppair, "%d %d %d %0.16lf %0.16lf %0.16lf %0.16lf\n", PairID[n], Pairatom1[n],
      Pairatom2[n],
00040    PairXij[n], PairYij[n], discDragx[n], discDragy[n]);
00041
00042   }
00043
00044
00045
```

## 5.40 source/DumpRestart.c File Reference

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

Include dependency graph for DumpRestart.c:




### Functions

- void DumpRestart ()

## 5.40.1 Function Documentation

### 5.40.1.1 DumpRestart()

```
void DumpRestart ()
```

Definition at line 25 of file DumpRestart.c.

```
00025                    {
00026  char DUMP[256];
00027  FILE *fpDUMP;
00028  sprintf(DUMP, "%s.Restart", prefix);
00029  fpDUMP = fopen(DUMP, "w");
00030  if(fpDUMP == NULL) {
00031   fprintf(stderr, "Error opening file %s for writing\n", DUMP);
00032   return;
00033  }
00034
00035   fprintf(fpDUMP, "timeNow %lf\n", timeNow);
00036   fprintf(fpDUMP, "nAtom %d\n",  nAtom);
00037   fprintf(fpDUMP, "nBond %d\n", nBond);
00038   fprintf(fpDUMP, "nAtomType %d\n", nAtomType);
00039   fprintf(fpDUMP, "nBondType %d\n", nBondType);
00040   fprintf(fpDUMP, "region[1] %0.16lf\n", region[1]);
00041   fprintf(fpDUMP, "region[2] %0.16lf\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:


## 5.41 DumpRestart.c

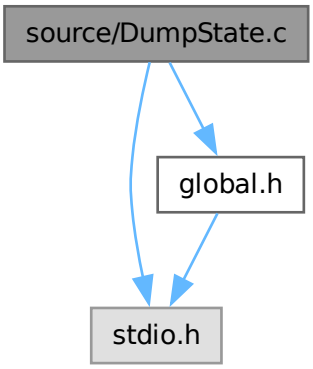
Go to the documentation of this file.

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022 #include <stdio.h>
00023 #include "global.h"
00024
00025 void DumpRestart() {
00026  char DUMP[256];
00027  FILE *fpDUMP;
00028  sprintf(DUMP, "%s.Restart", prefix);
00029  fpDUMP = fopen(DUMP, "w");
00030  if(fpDUMP == NULL) {
00031   fprintf(stderr, "Error opening file %s for writing\n", DUMP);
00032   return;
00033  }
00034
00035   fprintf(fpDUMP, "timeNow %lf\n", timeNow);
00036   fprintf(fpDUMP, "nAtom %d\n",  nAtom);
00037   fprintf(fpDUMP, "nBond %d\n", nBond);
00038   fprintf(fpDUMP, "nAtomType %d\n", nAtomType);
00039   fprintf(fpDUMP, "nBondType %d\n", nBondType);
00040   fprintf(fpDUMP, "region[1] %0.16lf\n", region[1]);
00041   fprintf(fpDUMP, "region[2] %0.16lf\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 }
00055
```

## 5.42 source/DumpState.c File Reference

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

Include dependency graph for DumpState.c:




### Functions

- void DumpState ()

### 5.42.1 Function Documentation

#### 5.42.1.1 DumpState()

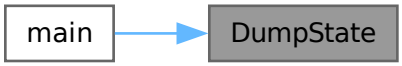
```
void DumpState ()
```

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:




## 5.43 DumpState.c

Go to the documentation of this file.

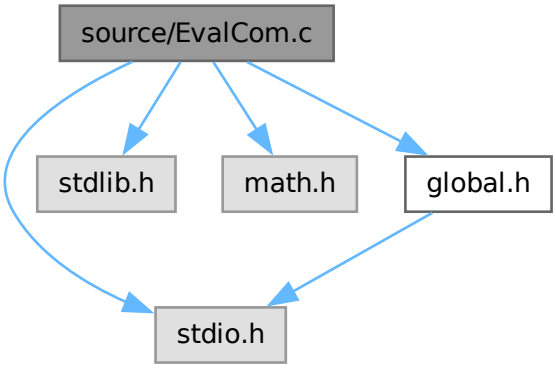
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022 #include <stdio.h>
00023 #include "global.h"
00024
00025 void DumpState() {
00026  char DUMP[256];
00027  FILE *fpDUMP;
00028  sprintf(DUMP, "%s.STATE", prefix);
00029  fpDUMP = fopen(DUMP, "w");
00030  if(fpDUMP == NULL) {
00031   fprintf(stderr, "Error opening file %s for writing\n", DUMP);
00032   return;
00033  }
00034
00035   fprintf(fpDUMP, "ITEM: TIMESTEP\n");
00036   fprintf(fpDUMP, "%lf\n",timeNow);
00037   fprintf(fpDUMP, "ITEM: NUMBER OF ATOMS\n");
00038   fprintf(fpDUMP, "%d\n",nAtom);
00039   fprintf(fpDUMP, "ITEM: BOX BOUNDS pp pp pp\n");
00040   fprintf(fpDUMP, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
00041   fprintf(fpDUMP, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
00042   fprintf(fpDUMP, "%lf %lf zlo zhi\n",  -0.1, 0.1);
00043   fprintf(fpDUMP, "ITEM: ATOMS id mol type radius x y vx vy fx fy\n");
00044   int n;
00045   for (n = 1; n <= nAtom; n++) {
00046    fprintf(fpDUMP, "%d\t %d\t %d\t %0.2lf\t %0.16lf\t %0.16lf\t %0.16lf\t %0.16lf\t %0.16lf\t
     %0.16lf\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 }
00051
```

## 5.44 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 ()

## 5.44.1 Function Documentation

### 5.44.1.1 EvalCom()

```
void EvalCom ()
```

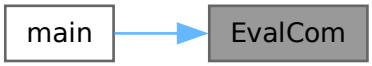
Definition at line 27 of file EvalCom.c.

```
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:




## 5.45 EvalCom.c

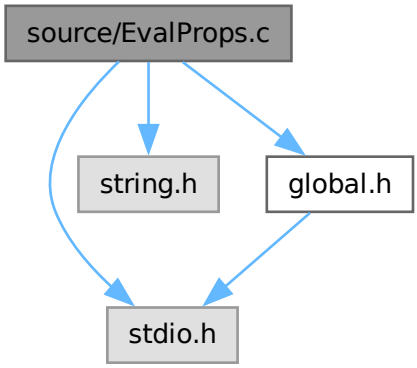
Go to the documentation of this file.

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022 #include<stdio.h>
00023 #include<stdlib.h>
00024 #include<math.h>
00025 #include"global.h"
00026
00027 void EvalCom(){
00028  int n;
00029  ComX = 0.0; ComY = 0.0; ComXRatio = 0.0; ComYRatio = 0.0;
00030  TotalMass = 0.0;
00031
00032  for(n=1; n<=nAtom; n++){
00033  if(molID[n] == 2){
00034   ComX += atomMass[n] * rxUnwrap[n];
00035   ComY += atomMass[n] * ryUnwrap[n];
00036   TotalMass += atomMass[n];
00037   } }
00038
00039   ComX = ComX/TotalMass;
00040   ComY = ComY/TotalMass;
00041
00042   if(timeNow == 0.0){
00043   ComX0 = ComX; ComY0 = ComY;
00044   }
00045   ComXRatio = ComX/ComX0;    ComYRatio = ComY/ComY0;
00046  }
00047
00048
00049
```

## 5.46 source/EvalProps.c File Reference

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

Include dependency graph for EvalProps.c:




### Functions

- void EvalProps ()

## 5.46.1 Function Documentation

### 5.46.1.1 EvalProps()

```
void EvalProps ()
```

Definition at line 26 of file EvalProps.c.

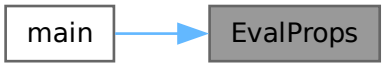
```
00026                 {
00027  double v;
00028  int n;
00029  double atomMassn;
00030  double KineEnrXSum, KineEnrYSum;
00031  virSum = 0.0;
00032  vSumX = 0.0; vSumY = 0.0; vSum = 0.0; vvSum = 0.0;
00033  KineEnrXSum = 0.0; KineEnrYSum = 0.0;
00034
00035  for (n = 1; n <= nAtom; n++) {
00036   //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

```
00065   totEnergy = kinEnergy + potEnergy;
00066   virSumxx = virSumPairxx  + virSumBondxx ;
00067   virSumyy = virSumPairyy  + virSumBondyy ;
00068   virSumxy = virSumPairxy  + virSumBondxy ;
00069   virSum = virSumPair + virSumBond;
00070   pressure = density * (vvSum + virSum) / (nAtom * NDIM);
00071
00072 }
```

References atomMass, 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:




## 5.47 EvalProps.c

Go to the documentation of this file.

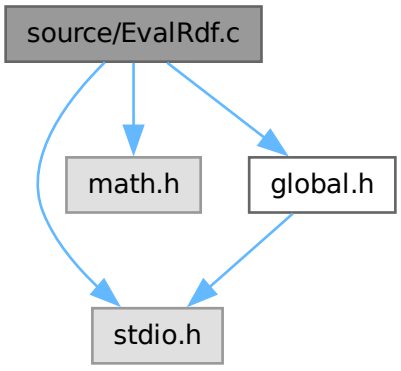
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022 #include <stdio.h>
00023 #include <string.h>
00024 #include "global.h"
00025
00026 void EvalProps() {
00027  double v;
00028  int n;
00029  double atomMassn;
00030  double KineEnrXSum, KineEnrYSum;
00031  virSum = 0.0;
00032  vSumX = 0.0; vSumY = 0.0; vSum = 0.0; vvSum = 0.0;
00033  KineEnrXSum = 0.0; KineEnrYSum = 0.0;
00034
00035  for (n = 1; n <= nAtom; n++) {
00036   // Initialize v with a default value to avoid "uninitialized" warning.
00037   v = 0.0;
00038   atomMassn = atomMass[n];
00039   // X direction velocity
00040   if (strcmp(solver, "Verlet") == 0) {
00041     v = vx[n];
00042   } else if (strcmp(solver, "LeapFrog") == 0) {
00043     v = vx[n] - 0.5 * deltaT * ax[n];
00044   }
00045    vSum += v;
00046    vvSum += Sqr(v);
00047    KineEnrXSum += 0.5 * atomMassn * Sqr(v);
```

```
00048     vSumX += v;
00049     // Y direction velocity
00050     if (strcmp(solver, "Verlet") == 0) {
00051     v = vy[n];
00052     } else if (strcmp(solver, "LeapFrog") == 0) {
00053     v = vy[n] - 0.5 * deltaT * ay[n];
00054     }
00055     vSum += v;
00056     vSumY += v;
00057     vvSum += Sqr(v);
00058     KineEnrYSum += 0.5 * atomMassn * Sqr(v);
00059    }
00060
00061    kinEnergy = (KineEnrXSum + KineEnrYSum) / nAtom ;
00062    uSumPairPerAtom = uSumPair / nAtom ;
00063    BondEnergyPerAtom = TotalBondEnergy / (0.5*nAtom); //Factor of 0.5 since each atom has one half the
      bond energy
00064    potEnergy = uSumPairPerAtom +  BondEnergyPerAtom ;
00065    totEnergy = kinEnergy + potEnergy;
00066    virSumxx = virSumPairxx  + virSumBondxx ;
00067    virSumyy = virSumPairyy  + virSumBondyy ;
00068    virSumxy = virSumPairxy  + virSumBondxy ;
00069    virSum = virSumPair + virSumBond;
00070    pressure = density * (vvSum + virSum) / (nAtom * NDIM);
00071
00072 }
00073
```

## 5.48 source/EvalRdf.c File Reference

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

Include dependency graph for EvalRdf.c:




### Functions

- void EvalRdf ()

### 5.48.1 Function Documentation

#### 5.48.1.1 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){
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     fprintf(fprdf,"rdf @ timeNow %lf\n", timeNow);
00063     for(n = 1 ; n <= sizeHistRdf ; n ++){
00064       rBin = (n - 0.5)*rangeRdf/sizeHistRdf;
00065       fprintf(fprdf, "%lf %lf\n", rBin, histRdf[n]);
00066     }
00067   }
00068
00069 }
```

References countRdf, fprdf, histRdf, limitRdf, nAtom, NDIM, rangeRdf, region, regionH, rx, ry, SignR, sizeHistRdf, Sqr, and timeNow.

## 5.49 EvalRdf.c

Go to the documentation of this file.

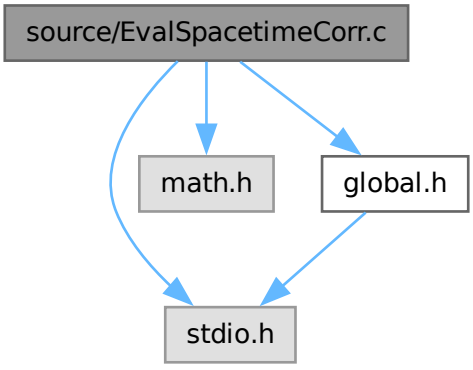
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your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

```
00037     for(j2 = j1 + 1 ; j2 <= nAtom ; j2 ++){
00038
00039       dr[1] = rx[j1] - rx[j2];
00040       if(fabs(dr[1]) > regionH[1])
00041     dr[1] -= SignR(region[1], dr[1]);
00042
00043       dr[2] = ry[j1] - ry[j2];
00044       if(fabs(dr[2]) > regionH[2])
00045     dr[2] -= SignR(region[2], dr[2]);
00046
00047       rr = Sqr(dr[1]) + Sqr(dr[2]);
00048
00049       if(rr < rrRange){
00050     n = (int)(sqrt(rr)/deltaR) + 1;
00051     histRdf[n] ++;
00052       }
00053     }
00054   }
00055
00056   if(countRdf == limitRdf){
00057     normFac = region[1]*region[2] / (M_PI*Sqr(deltaR)*nAtom*nAtom*countRdf );
00058     for(n = 1 ; n <= sizeHistRdf ; n ++)
00059       histRdf[n] *= normFac/(n-0.5);
00060     // PRINT THE RADIAL DISTRIBUTION DATA ON TO DISK FILE
00061     real rBin;
00062     fprintf(fprdf,"rdf @ timeNow %lf\n", timeNow);
00063     for(n = 1 ; n <= sizeHistRdf ; n ++){
00064       rBin = (n - 0.5)*rangeRdf/sizeHistRdf;
00065       fprintf(fprdf, "%lf %lf\n", rBin, histRdf[n]);
00066     }
00067   }
00068
00069 }
00070
```

## 5.50 source/EvalSpacetimeCorr.c File Reference

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

Include dependency graph for EvalSpacetimeCorr.c:




### Functions

- void EvalSpacetimeCorr ()

## 5.50.1 Function Documentation

### 5.50.1.1 EvalSpacetimeCorr()

```
void EvalSpacetimeCorr ()
```

Definition at line 26 of file EvalSpacetimeCorr.c.

```
00026                           {
00027   real cosV=0., cosV0=0., cosV1=0., cosV2=0., sinV=0., sinV1=0., sinV2=0.;
00028   real COSA, SINA, COSV, SINV;
00029   int j, m, n, nb, ni, nv;
00030   real kMin = 2. * M_PI / region[1];
00031   real kMax = M_PI;
00032   real deltaK = (kMax - kMin) / nFunCorr;
00033
00034   for (j = 1; j <= 2*nFunCorr; j++)
00035     cfVal[j] = 0.;
00036
00037   for (n = 1; n <= nAtom; n++){
00038     j = 1;
00039     COSA = cos(kMin*rx[n]);
00040     SINA = sin(kMin*rx[n]);
00041     for (m = 1; m <= nFunCorr; m++){
00042       if(m == 1){
00043     cosV = cos(deltaK*rx[n]);
00044     sinV = sin(deltaK*rx[n]);
00045     cosV0 = cosV;
00046       }else if(m == 2){
00047     cosV1 = cosV;
00048     sinV1 = sinV;
00049     cosV = 2.*cosV0*cosV1-1;
00050     sinV = 2.*cosV0*sinV1;
00051       }else{
00052     cosV2 = cosV1;
00053     sinV2 = sinV1;
00054     cosV1 = cosV;
00055     sinV1 = sinV;
00056     cosV = 2.*cosV0*cosV1-cosV2;
00057     sinV = 2.*cosV0*sinV1-sinV2;
00058       }
00059       COSV = COSA*cosV - SINA*sinV;
00060       SINV = SINA*cosV + COSA*sinV;
00061       cfVal[j] += COSV;
00062       cfVal[j+1] += SINV;
00063       j += 2;
00064     }
00065   }
00066
00067   for (nb = 1; nb <= nBuffCorr; nb++){
00068     indexCorr[nb] += 1;
00069     if (indexCorr[nb] <= 0) continue;
00070     ni = nFunCorr * (indexCorr[nb] - 1);
00071     if (indexCorr[nb] == 1){
00072       for (j = 1; j <= 2*nFunCorr; j++)
00073     cfOrg[nb][j] = cfVal[j];
00074     }
00075
00076     for (j = 1; j <= nFunCorr; j++)
00077       spacetimeCorr[nb][ni + j] = 0.;
00078
00079     j = 1;
00080     for (m = 1; m <= nFunCorr; m++){
00081       nv = m + ni;
00082       spacetimeCorr[nb][nv] += cfVal[j] * cfOrg[nb][j] + cfVal[j + 1] * cfOrg[nb][j + 1];
00083       j += 2;
00084     }
00085
00086   }
00087
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);
00106         real tVal;
00107         for (n = 1; n <= nValCorr; n++){
00108       tVal = (n-1)*stepCorr*deltaT;
00109       fprintf (fpdnsty, "%e\t", tVal);
00110       int nn = nFunCorr*(n-1);
00111       for (j = 1; j <= nFunCorr; j ++)
00112             fprintf (fpdnsty, "%e\t", spacetimeCorrAv[nn + j]);
00113           fprintf (fpdnsty, "\n");
00114         }
00115
00116         countCorrAv = 0.;
00117         for (j = 1; j <= nFunCorr*nValCorr; j++)
00118       spacetimeCorrAv[j] = 0.;
00119       }
00120     }
00121   }
00122 }
```

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

## 5.51 EvalSpacetimeCorr.c

Go to the documentation of this file.

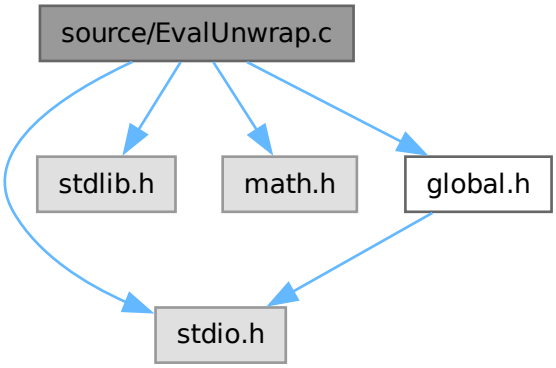
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your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

```
00060       SINV = SINA*cosV + COSA*sinV;
00061       cfVal[j] += COSV;
00062       cfVal[j+1] += SINV;
00063       j += 2;
00064     }
00065   }
00066
00067   for (nb = 1; nb <= nBuffCorr; nb++){
00068     indexCorr[nb] += 1;
00069     if (indexCorr[nb] <= 0) continue;
00070     ni = nFunCorr * (indexCorr[nb] - 1);
00071     if (indexCorr[nb] == 1){
00072       for (j = 1; j <= 2*nFunCorr; j++)
00073     cfOrg[nb][j] = cfVal[j];
00074     }
00075
00076     for (j = 1; j <= nFunCorr; j++)
00077       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);
00106       real tVal;
00107       for (n = 1; n <= nValCorr; n++){
00108     tVal = (n-1)*stepCorr*deltaT;
00109     fprintf (fpdnsty, "%e\t", tVal);
00110     int nn = nFunCorr*(n-1);
00111     for (j = 1; j <= nFunCorr; j ++)
00112           fprintf (fpdnsty, "%e\t", spacetimeCorrAv[nn + j]);
00113         fprintf (fpdnsty, "\n");
00114       }
00115
00116       countCorrAv = 0.;
00117       for (j = 1; j <= nFunCorr*nValCorr; j++)
00118     spacetimeCorrAv[j] = 0.;
00119     }
00120    }
00121   }
00122 }
```

## 5.52 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 ()

### 5.52.1 Function Documentation

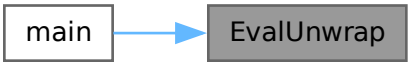
#### 5.52.1.1 EvalUnwrap()

```
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:




## 5.53 EvalUnwrap.c

Go to the documentation of this file.

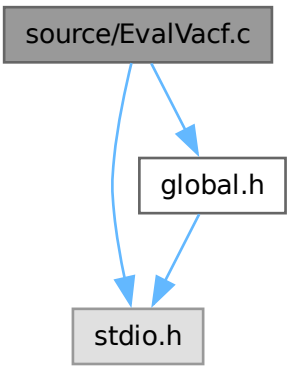
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
```

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

## 5.54 source/EvalVacf.c File Reference

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

Include dependency graph for EvalVacf.c:




### Functions

- void AccumVacf ()
- void EvalVacf ()

### 5.54.1 Function Documentation

#### 5.54.1.1 AccumVacf()

```
void AccumVacf ()
```

Definition at line 27 of file AccumVacf.c.

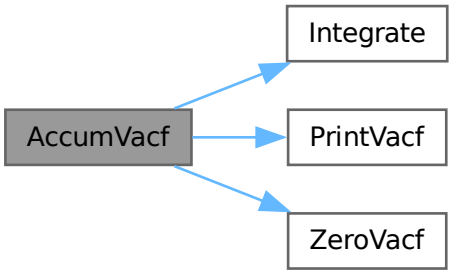
```
00027                {
```

```
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 } } } }
```

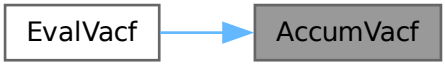
References countAcfAv, deltaT, indexAcf, Integrate(), kinEnergy, limitAcfAv, nBuffAcf, nValAcf, PrintVacf(), region, stepAcf, viscAcf, viscAcfAv, viscAcfInt, and ZeroVacf().

Referenced by EvalVacf().

Here is the call graph for this function:




Here is the caller graph for this function:




#### 5.54.1.2 EvalVacf()

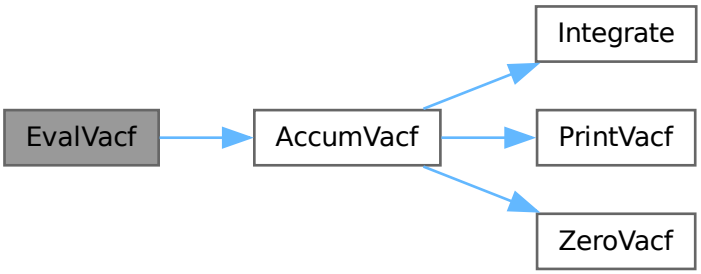
```
void EvalVacf ()
```

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:




## 5.55 EvalVacf.c

Go to the documentation of this file.

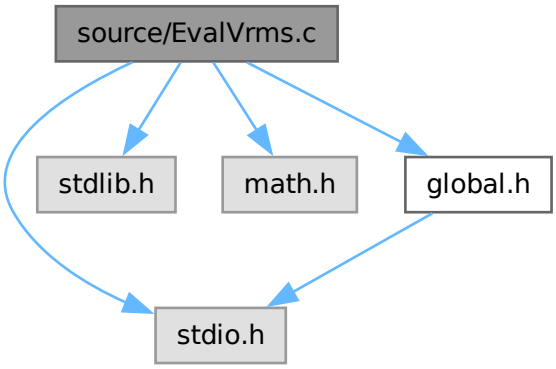
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 void AccumVacf();
00026 void EvalVacf(){
00027   int n, nb, ni;
00028   double viscVec = 0.;
00029   double v[3];
00030   for(n = 1 ; n <= nAtom ; n ++){
00031     v[1] = vx[n] - 0.5*ax[n]*deltaT;
00032     v[2] = vy[n] - 0.5*ay[n]*deltaT;
00033     viscVec += v[1]*v[2];
00034   }
00035   viscVec += rfAtom;
00036   for(nb = 1 ; nb <= nBuffAcf ; nb ++){
00037     indexAcf[nb] ++;
00038     if(indexAcf[nb] <= 0)continue;
00039     if(indexAcf[nb] == 1){
00040       viscAcfOrg[nb] = viscVec;
00041     }
00042     ni = indexAcf[nb];
00043     viscAcf[nb][ni] = viscAcfOrg[nb]*viscVec;
```

```
00044   }
00045   AccumVacf();
00046 }
```

## 5.56 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 ()

### 5.56.1 Function Documentation

#### 5.56.1.1 EvalVrms()

```
void EvalVrms ()
```

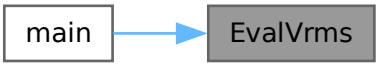
Definition at line 27 of file EvalVrms.c.

```
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:



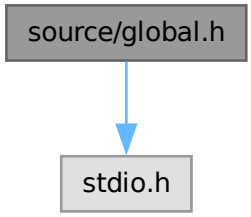

## 5.57 EvalVrms.c

Go to the documentation of this file.

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022 #include<stdio.h>
00023 #include<stdlib.h>
00024 #include<math.h>
00025 #include"global.h"
00026
00027 void EvalVrms(){
00028   int n;
00029   VSqr = 0.0;
00030   VMeanSqr = 0.0;
00031   VRootMeanSqr = 0.0;
00032
00033   for(n = 1 ; n <= nAtom ; n ++){
00034   VSqr += Sqr(vx[n]) + Sqr(vy[n]);
00035   }
00036   VMeanSqr = VSqr/nAtom;
00037   VRootMeanSqr = sqrt(VMeanSqr);
00038   }
00039
00040
00041
```

## 5.58 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)
- #define SignR(x, y)

### Typedefs

- typedef double real

### Variables

- double * rx
- double * ry
- double * vx
- double * vy
- double * ax
- double * ay
- double * fx
- double * fy
- 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
- double mass
- 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
- double Kn
- double fxExtern
- double fyExtern
- double FyByIx
- double fxByfy
- double forceSumxExtern
- double forceSumyExtern
- int DampFlag
- double strech
- char mode [64]
- char inputConfig [128]
- 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]
- int * molID
- int ** isBonded
- double InterfaceWidth
- double bigDiameter
- int nAtomInterface
- int nDiscInterface
- int nAtomBlock
- int * atomIDInterface
- int BondPairFlag
- int nAtomInterfaceTotal
- 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 force [256]
- FILE * fpforce
- char com [256]
- FILE * fpcom
- char pair [256]
- FILE * fppair

## 5.58.1 Macro Definition Documentation

### 5.58.1.1 EXTERN

```
#define EXTERN extern
```

Definition at line 8 of file global.h.

### 5.58.1.2 NDIM

```
#define NDIM 2
```

Definition at line 13 of file global.h.
Referenced by ComputeBondForce(), ComputeForcesCells(), ComputePairForce(), EvalProps(), EvalRdf(), EvalSpacetimeCorr(), and LeapfrogStep().

### 5.58.1.3 SignR

```
#define SignR(
              x,
              y)
```

**Value:**

```
(((y) >= 0) ? (x) : (- (x)))
```

Definition at line 15 of file global.h.
Referenced by EvalRdf(), and LeapfrogStep().

### 5.58.1.4 Sqr

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

**Value:**

```
((x) * (x))
```

Definition at line 14 of file global.h.
Referenced by AccumProps(), BrownianStep(), ComputeBondForce(), ComputeForcesCells(), ComputePairForce(), EvalProps(), EvalRdf(), EvalVrms(), and LeapfrogStep().

## 5.58.2 Typedef Documentation

### 5.58.2.1 real

```
typedef double real
```

Definition at line 11 of file global.h.

### 5.58.3 Variable Documentation

#### 5.58.3.1 atom1

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

Referenced by Close(), ComputeBondForce(), DumpBonds(), DumpRestart(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.2 atom2

```
int * atom2
```

Definition at line 36 of file global.h.
Referenced by Close(), ComputeBondForce(), DumpBonds(), DumpRestart(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.3 atomID

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

Referenced by Close(), DumpRestart(), DumpState(), Init(), ReadBinaryRestart(), Trajectory(), and WriteBinaryRestart().

#### 5.58.3.4 atomIDInterface

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

Referenced by Close(), ComputePairForce(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.5 atomMass

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

Referenced by Close(), ComputePairForce(), EvalCom(), EvalProps(), Init(), ReadBinaryRestart(), VelocityVerletStep(), and WriteBinaryRestart().

#### 5.58.3.6 atomRadius

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

Referenced by ApplyBoundaryCond(), Close(), ComputeForcesCells(), ComputePairForce(), DumpRestart(), DumpState(), Init(), ReadBinaryRestart(), Trajectory(), and WriteBinaryRestart().

#### 5.58.3.7 atomType

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

Referenced by ApplyDrivingForce(), Close(), DumpRestart(), DumpState(), Init(), PrintForceSum(), ReadBinaryRestart(), Trajectory(), VelocityVerletStep(), and WriteBinaryRestart().

#### 5.58.3.8 ax

```
double * ax
```

Definition at line 17 of file global.h.
Referenced by ApplyDrivingForce(), ApplyViscous(), BrownianStep(), Close(), ComputeForcesCells(), DumpState(), EvalProps(), EvalVacf(), Init(), LeapfrogStep(), ReadBinaryRestart(), VelocityVerletStep(), and WriteBinaryRestart().

#### 5.58.3.9 ay

```
double * ay
```

Definition at line 17 of file global.h.
Referenced by ApplyDrivingForce(), ApplyViscous(), BrownianStep(), Close(), ComputeForcesCells(), DumpState(), EvalProps(), EvalVacf(), Init(), LeapfrogStep(), ReadBinaryRestart(), VelocityVerletStep(), and WriteBinaryRestart().

#### 5.58.3.10 bigDiameter

```
double bigDiameter
```
Definition at line 75 of file global.h.
Referenced by Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.11 bond

```
char bond[256]  [extern]
```
Referenced by main().

#### 5.58.3.12 BondEnergy

```
double* BondEnergy  [extern]
```
Referenced by ComputeBondForce(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.13 BondEnergyPerAtom

```
double BondEnergyPerAtom
```
Definition at line 40 of file global.h.
Referenced by EvalProps(), HaltConditionCheck(), and PrintSummary().

#### 5.58.3.14 BondID

```
int* BondID  [extern]
```
Referenced by Close(), DumpBonds(), DumpRestart(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.15 BondLength

```
double * BondLength
```
Definition at line 39 of file global.h.
Referenced by ComputeBondForce(), DumpBonds(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.16 BondPairFlag

```
int BondPairFlag  [extern]
```

#### 5.58.3.17 BondType

```
int * BondType
```
Definition at line 37 of file global.h.
Referenced by Close(), DumpBonds(), DumpRestart(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.18 cellList

```
int* cellList  [extern]
```
Referenced by Close(), ComputeForcesCells(), Init(), and ReadBinaryRestart().

#### 5.58.3.19 cells

```
int cells[2+1]
```
Definition at line 83 of file global.h.
Referenced by ComputeForcesCells(), Init(), and ReadBinaryRestart().

#### 5.58.3.20 cfOrg

```
double** cfOrg  [extern]
```
Referenced by AllocArrays(), Close(), and EvalSpacetimeCorr().

#### 5.58.3.21 cfVal

```
double * cfVal
```
Definition at line 94 of file global.h.
Referenced by AllocArrays(), Close(), and EvalSpacetimeCorr().

#### 5.58.3.22 com

```
char com[256]  [extern]
```
Referenced by main().

#### 5.58.3.23 ComX

```
double ComX  [extern]
```
Referenced by EvalCom(), and PrintCom().

#### 5.58.3.24 ComX0

```
double ComX0
```
Definition at line 47 of file global.h.
Referenced by EvalCom().

#### 5.58.3.25 ComXRatio

```
double ComXRatio
```
Definition at line 47 of file global.h.
Referenced by EvalCom().

#### 5.58.3.26 ComY

```
double ComY
```
Definition at line 47 of file global.h.
Referenced by EvalCom(), and PrintCom().

#### 5.58.3.27 ComY0

```
double ComY0
```
Definition at line 47 of file global.h.
Referenced by EvalCom().

#### 5.58.3.28 ComYRatio

```
double ComYRatio
```
Definition at line 47 of file global.h.
Referenced by EvalCom().

#### 5.58.3.29 countAcfAv

```
int countAcfAv
```
Definition at line 100 of file global.h.
Referenced by AccumVacf(), and ZeroVacf().

#### 5.58.3.30 countCorrAv

```
int countCorrAv
```
Definition at line 95 of file global.h.
Referenced by EvalSpacetimeCorr(), and SetupJob().

#### 5.58.3.31 countRdf

```
int countRdf  [extern]
```
Referenced by EvalRdf(), and SetupJob().

#### 5.58.3.32 DampFlag

```
int DampFlag  [extern]
```
Referenced by ComputeBondForce(), ComputePairForce(), Init(), and ReadBinaryRestart().

#### 5.58.3.33 deltaT

```
double deltaT
```
Definition at line 20 of file global.h.
Referenced by AccumVacf(), BrownianStep(), EvalProps(), EvalSpacetimeCorr(), EvalVacf(), Init(), LeapfrogStep(), main(), PrintVacf(), ReadBinaryRestart(), and VelocityVerletStep().

#### 5.58.3.34 DeltaX

```
double DeltaX
```
Definition at line 49 of file global.h.
Referenced by DisplaceAtoms(), Init(), and ReadBinaryRestart().

#### 5.58.3.35 DeltaY

```
double DeltaY  [extern]
```
Referenced by DisplaceAtoms(), Init(), and ReadBinaryRestart().

#### 5.58.3.36 density

```
double density
```
Definition at line 20 of file global.h.
Referenced by EvalProps(), Init(), and ReadBinaryRestart().

#### 5.58.3.37 discDragx

```
double* discDragx  [extern]
```
Referenced by ComputeForcesCells(), ComputePairForce(), DumpPairs(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.38 discDragy

```
double * discDragy
```
Definition at line 42 of file global.h.
Referenced by ComputeForcesCells(), ComputePairForce(), DumpPairs(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.39 dnsty

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

#### 5.58.3.40 dump

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

#### 5.58.3.41 dumpPairFlag

```
int dumpPairFlag  [extern]
```

#### 5.58.3.42 fax

```
double* fax  [extern]
```
Referenced by Close(), and Init().

#### 5.58.3.43 fay

```
double * fay
```
Definition at line 85 of file global.h.
Referenced by Close(), and Init().

#### 5.58.3.44 force

```
char force[256]  [extern]
```
Referenced by main().

#### 5.58.3.45 forceSumxExtern

```
double forceSumxExtern
```
Definition at line 53 of file global.h.
Referenced by ApplyForce(), PrintForceSum(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.46 forceSumyExtern

```
double forceSumyExtern
```
Definition at line 53 of file global.h.
Referenced by ApplyForce(), PrintForceSum(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.47 fpbond

```
FILE* fpbond  [extern]
```
Referenced by DumpBonds(), and main().

#### 5.58.3.48 fpcom

```
FILE* fpcom  [extern]
```
Referenced by Init(), main(), PrintCom(), and ReadBinaryRestart().

#### 5.58.3.49 fpdnsty

```
FILE* fpdnsty  [extern]
```
Referenced by EvalSpacetimeCorr().

#### 5.58.3.50 fpdump

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

#### 5.58.3.51 fpforce

```
FILE* fpforce  [extern]
```
Referenced by Init(), main(), PrintForceSum(), and ReadBinaryRestart().

#### 5.58.3.52 fpmomentum

```
FILE* fpmomentum  [extern]
```
Referenced by PrintMomentum().

#### 5.58.3.53 fppair

```
FILE* fppair  [extern]
```
Referenced by DumpPairs(), and main().

#### 5.58.3.54 fprdf

```
FILE* fprdf  [extern]
```
Referenced by EvalRdf().

#### 5.58.3.55 fpresult

```
FILE* fpresult  [extern]
```
Referenced by ApplyBoundaryCond(), HaltConditionCheck(), Init(), main(), PrintSummary(), and ReadBinaryRestart().

#### 5.58.3.56 fpstress

```
FILE* fpstress  [extern]
```
Referenced by PrintStress().

#### 5.58.3.57 fpvisc

```
FILE* fpvisc  [extern]
```
Referenced by PrintVacf().

#### 5.58.3.58 fpvrms

```
FILE* fpvrms  [extern]
```
Referenced by Init(), main(), PrintVrms(), and ReadBinaryRestart().

#### 5.58.3.59 fpxyz

```
FILE* fpxyz  [extern]
```
Referenced by main(), and Trajectory().

#### 5.58.3.60 freezeAtomType

```
int freezeAtomType  [extern]
```
Referenced by Init(), and VelocityVerletStep().

#### 5.58.3.61 frfAtom

```
double frfAtom
```
Definition at line 85 of file global.h.

#### 5.58.3.62 fuSum

```
double fuSum
```
Definition at line 85 of file global.h.

#### 5.58.3.63 fvirSum

```
double fvirSum
```
Definition at line 85 of file global.h.

#### 5.58.3.64 fx

```
double * fx
```
Definition at line 17 of file global.h.
Referenced by ApplyForce(), Close(), ComputeBondForce(), ComputePairForce(), Init(), PrintForceSum(), ReadBinaryRestart(), Trajectory(), VelocityVerletStep(), and WriteBinaryRestart().

#### 5.58.3.65 fxByfy

```
double fxByfy
```
Definition at line 53 of file global.h.
Referenced by ApplyForce(), Init(), and ReadBinaryRestart().

#### 5.58.3.66 fxExtern

```
double fxExtern  [extern]
```
Referenced by ApplyForce().

#### 5.58.3.67 fy

```
double * fy
```
Definition at line 17 of file global.h.
Referenced by ApplyForce(), Close(), ComputeBondForce(), ComputePairForce(), Init(), PrintForceSum(), ReadBinaryRestart(), Trajectory(), VelocityVerletStep(), and WriteBinaryRestart().

#### 5.58.3.68 FyBylx

```
double FyBylx
```
Definition at line 53 of file global.h.
Referenced by ApplyForce(), Init(), and ReadBinaryRestart().

#### 5.58.3.69 fyExtern

```
double fyExtern
```
Definition at line 53 of file global.h.
Referenced by ApplyForce().

#### 5.58.3.70 gamman

```
double gamman  [extern]
```
Referenced by ComputeBondForce(), ComputeForcesCells(), ComputePairForce(), Init(), and ReadBinaryRestart().

#### 5.58.3.71 HaltCondition

```
double HaltCondition  [extern]
```
Referenced by HaltConditionCheck(), Init(), and ReadBinaryRestart().

#### 5.58.3.72 histRdf

```
double* histRdf  [extern]
```
Referenced by AllocArrays(), and EvalRdf().

#### 5.58.3.73 ImageX

```
int* ImageX  [extern]
```
Referenced by Close(), EvalUnwrap(), Init(), ReadBinaryRestart(), VelocityVerletStep(), and WriteBinaryRestart().

#### 5.58.3.74 ImageY

```
int * ImageY
```
Definition at line 50 of file global.h.
Referenced by Close(), EvalUnwrap(), Init(), ReadBinaryRestart(), VelocityVerletStep(), and WriteBinaryRestart().

#### 5.58.3.75 indexAcf

```
double* indexAcf  [extern]
```
Referenced by AccumVacf(), AllocArrays(), Close(), EvalVacf(), and InitVacf().

#### 5.58.3.76 indexCorr

```
int* indexCorr  [extern]
```
Referenced by AllocArrays(), Close(), EvalSpacetimeCorr(), and SetupJob().

#### 5.58.3.77 initUcell

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

#### 5.58.3.78 inputConfig

```
char inputConfig[128]
```
Definition at line 58 of file global.h.
Referenced by Init().

#### 5.58.3.79 InterfaceWidth

```
double InterfaceWidth  [extern]
```
Referenced by Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.80 isBonded

```
int** isBonded  [extern]
```
Referenced by Close(), ComputePairForce(), Init(), and ReadBinaryRestart().

#### 5.58.3.81 kappa

```
double kappa
```
Definition at line 21 of file global.h.
Referenced by Init(), LeapfrogStep(), and ReadBinaryRestart().

#### 5.58.3.82 kb

```
double* kb  [extern]
```
Referenced by Close(), ComputeBondForce(), DumpRestart(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.83 kinEnergy

```
double kinEnergy
```
Definition at line 20 of file global.h.
Referenced by AccumProps(), AccumVacf(), EvalProps(), HaltConditionCheck(), and PrintSummary().

#### 5.58.3.84 Kn

```
double Kn  [extern]
```
Referenced by ComputePairForce(), and Init().

#### 5.58.3.85 limitAcfAv

```
int limitAcfAv
```
Definition at line 100 of file global.h.
Referenced by AccumVacf(), and Init().

#### 5.58.3.86 limitCorrAv

```
int limitCorrAv
```
Definition at line 95 of file global.h.
Referenced by EvalSpacetimeCorr(), and Init().

#### 5.58.3.87 limitRdf

```
int limitRdf
```
Definition at line 104 of file global.h.
Referenced by EvalRdf(), and Init().

#### 5.58.3.88 mass

```
double mass  [extern]
```
Referenced by Init(), and ReadBinaryRestart().

#### 5.58.3.89 master

```
int master
```
Definition at line 84 of file global.h.

#### 5.58.3.90 mode

```
char mode[64]  [extern]
```
Referenced by Init().

#### 5.58.3.91 molID

```
int* molID  [extern]
```
Referenced by ApplyForce(), Close(), DisplaceAtoms(), DumpRestart(), DumpState(), EvalCom(), Init(), ReadBinaryRestart(), Trajectory(), and WriteBinaryRestart().

#### 5.58.3.92 momentum

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

#### 5.58.3.93 moreCycles

```
int moreCycles
```
Definition at line 24 of file global.h.
Referenced by main().

#### 5.58.3.94 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(), PrintForceSum(), ReadBinaryRestart(), Trajectory(), VelocityVerletStep(), and WriteBinaryRestart().

#### 5.58.3.95 nAtomBlock

```
int nAtomBlock
```
Definition at line 76 of file global.h.
Referenced by ApplyForce(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.96 nAtomInterface

```
int nAtomInterface  [extern]
```
Referenced by ComputePairForce(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.97 nAtomInterfaceTotal

```
int nAtomInterfaceTotal  [extern]
```

#### 5.58.3.98 nAtomType

```
int nAtomType  [extern]
```
Referenced by DumpRestart(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.99 nBond

```
int nBond  [extern]
```
Referenced by ComputeBondForce(), DumpBonds(), DumpRestart(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.100 nBondType

```
int nBondType
```
Definition at line 35 of file global.h.
Referenced by DumpRestart(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.101 nBuffAcf

```
int nBuffAcf
```
Definition at line 100 of file global.h.
Referenced by AccumVacf(), AllocArrays(), Close(), EvalVacf(), Init(), and InitVacf().

#### 5.58.3.102 nBuffCorr

```
int nBuffCorr
```
Definition at line 95 of file global.h.
Referenced by AllocArrays(), Close(), EvalSpacetimeCorr(), Init(), and SetupJob().

#### 5.58.3.103 nDiscInterface

```
int nDiscInterface
```
Definition at line 76 of file global.h.
Referenced by Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.104 nFunCorr

```
int nFunCorr
```
Definition at line 95 of file global.h.
Referenced by AllocArrays(), EvalSpacetimeCorr(), Init(), and SetupJob().

#### 5.58.3.105 nodeDragx

```
double * nodeDragx
```
Definition at line 42 of file global.h.
Referenced by ComputeBondForce(), DumpBonds(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.106 nodeDragy

```
double * nodeDragy
```
Definition at line 42 of file global.h.
Referenced by ComputeBondForce(), DumpBonds(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.107 nPairActive

```
int nPairActive
```
Definition at line 62 of file global.h.
Referenced by ComputePairForce(), DumpPairs(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.108 nPairTotal

```
int nPairTotal  [extern]
```
Referenced by ComputePairForce(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.109 nValAcf

```
int nValAcf  [extern]
```
Referenced by AccumVacf(), AllocArrays(), Init(), InitVacf(), PrintVacf(), and ZeroVacf().

#### 5.58.3.110 nValCorr

```
int nValCorr
```
Definition at line 95 of file global.h.
Referenced by AllocArrays(), EvalSpacetimeCorr(), Init(), and SetupJob().

#### 5.58.3.111 pair

```
char pair[256]  [extern]
```
Referenced by main().

#### 5.58.3.112 Pairatom1

```
int * Pairatom1
```
Definition at line 63 of file global.h.
Referenced by Close(), ComputePairForce(), DumpPairs(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.113 Pairatom2

```
int * Pairatom2
```
Definition at line 63 of file global.h.
Referenced by Close(), ComputePairForce(), DumpPairs(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.114 PairID

```
int* PairID  [extern]
```
Referenced by Close(), ComputePairForce(), DumpPairs(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.115 PairXij

```
double* PairXij  [extern]
```
Referenced by Close(), ComputePairForce(), DumpPairs(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.116 PairYij

```
double * PairYij
```
Definition at line 64 of file global.h.
Referenced by Close(), ComputePairForce(), DumpPairs(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.117 potEnergy

```
double potEnergy
```
Definition at line 20 of file global.h.
Referenced by AccumProps(), EvalProps(), HaltConditionCheck(), and PrintSummary().

#### 5.58.3.118 prefix

```
char* prefix  [extern]
```
Definition at line 13 of file main.c.
Referenced by DumpRestart(), DumpState(), main(), and WriteBinaryRestart().

#### 5.58.3.119 pressure

```
double pressure
```
Definition at line 21 of file global.h.
Referenced by AccumProps(), EvalProps(), HaltConditionCheck(), PrintStress(), and PrintSummary().

#### 5.58.3.120 RadiusIJ

```
double RadiusIJ  [extern]
```
Referenced by ComputeForcesCells(), and ComputePairForce().

#### 5.58.3.121 RadiusIJInv

```
double RadiusIJInv
```
Definition at line 27 of file global.h.
Referenced by ComputeForcesCells(), and ComputePairForce().

#### 5.58.3.122 rangeRdf

```
double rangeRdf
```
Definition at line 103 of file global.h.
Referenced by EvalRdf(), and Init().

#### 5.58.3.123 rank

```
int rank  [extern]
```
Referenced by ComputeForcesCells().

#### 5.58.3.124 rCut

```
double rCut
```
Definition at line 21 of file global.h.
Referenced by Init(), LeapfrogStep(), and ReadBinaryRestart().

#### 5.58.3.125 rdf

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

#### 5.58.3.126 region

```
double region[2+1]  [extern]
```
Referenced by AccumVacf(), ApplyBoundaryCond(), ApplyLeesEdwardsBoundaryCond(), ComputeBondForce(), ComputeForcesCells(), ComputePairForce(), DumpRestart(), EvalRdf(), EvalSpacetimeCorr(), EvalUnwrap(), Init(), LeapfrogStep(), ReadBinaryRestart(), VelocityVerletStep(), and WriteBinaryRestart().

#### 5.58.3.127 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(), ReadBinaryRestart(), Trajectory(), and VelocityVerletStep().

#### 5.58.3.128 result

```
char result[250]  [extern]
```
Referenced by main().

#### 5.58.3.129 rfAtom

```
double rfAtom  [extern]
```
Referenced by ComputeForcesCells(), and EvalVacf().

#### 5.58.3.130 ro

```
double * ro
```
Definition at line 38 of file global.h.
Referenced by Close(), ComputeBondForce(), DumpBonds(), DumpRestart(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.131 rx

```
double* rx  [extern]
```
Referenced by ApplyBoundaryCond(), ApplyLeesEdwardsBoundaryCond(), ApplyShear(), BrownianStep(), Close(), ComputeBondForce(), ComputeForcesCells(), ComputePairForce(), DisplaceAtoms(), DumpRestart(), DumpState(), EvalRdf(), EvalSpacetimeCorr(), EvalUnwrap(), Init(), LeapfrogStep(), ReadBinaryRestart(), Trajectory(), VelocityVerletStep(), and WriteBinaryRestart().

#### 5.58.3.132 rxUnwrap

```
double* rxUnwrap  [extern]
```
Referenced by Close(), EvalCom(), EvalUnwrap(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.133 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(), ReadBinaryRestart(), Trajectory(), VelocityVerletStep(), and WriteBinaryRestart().

#### 5.58.3.134 ryUnwrap

```
double * ryUnwrap
```
Definition at line 51 of file global.h.
Referenced by Close(), EvalCom(), EvalUnwrap(), Init(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.135 shearDisplacement

```
double shearDisplacement  [extern]
```
Referenced by ApplyLeesEdwardsBoundaryCond(), ComputeBondForce(), ComputePairForce(), and Init().

#### 5.58.3.136 shearVelocity

```
double shearVelocity
```
Definition at line 45 of file global.h.
Referenced by Init().

#### 5.58.3.137 size

```
int size
```
Definition at line 84 of file global.h.
Referenced by ComputeForcesCells().

#### 5.58.3.138 sizeHistRdf

```
int sizeHistRdf
```
Definition at line 104 of file global.h.
Referenced by AllocArrays(), EvalRdf(), and Init().

#### 5.58.3.139 sKinEnergy

```
double sKinEnergy
```
Definition at line 21 of file global.h.
Referenced by AccumProps().

#### 5.58.3.140 solver

```
char solver[128]  [extern]
```
Referenced by EvalProps(), Init(), and ReadBinaryRestart().

#### 5.58.3.141 spacetimeCorr

```
double ** spacetimeCorr
```
Definition at line 94 of file global.h.
Referenced by AllocArrays(), Close(), and EvalSpacetimeCorr().

#### 5.58.3.142 spacetimeCorrAv

```
double * spacetimeCorrAv
```
Definition at line 94 of file global.h.
Referenced by AllocArrays(), Close(), EvalSpacetimeCorr(), and SetupJob().

#### 5.58.3.143 speed

```
double* speed  [extern]
```
Referenced by Close(), and Init().

#### 5.58.3.144 sPotEnergy

```
double sPotEnergy
```
Definition at line 21 of file global.h.
Referenced by AccumProps().

#### 5.58.3.145 sPressure

```
double sPressure
```
Definition at line 22 of file global.h.
Referenced by AccumProps().

#### 5.58.3.146 SqrRadiusIJ

```
double SqrRadiusIJ
```
Definition at line 27 of file global.h.
Referenced by ComputeForcesCells(), and ComputePairForce().

#### 5.58.3.147 ssKinEnergy

```
double ssKinEnergy
```
Definition at line 22 of file global.h.
Referenced by AccumProps().

#### 5.58.3.148 ssPotEnergy

```
double ssPotEnergy
```
Definition at line 22 of file global.h.
Referenced by AccumProps().

#### 5.58.3.149 ssPressure

```
double ssPressure
```
Definition at line 22 of file global.h.
Referenced by AccumProps().

#### 5.58.3.150 ssTotEnergy

```
double ssTotEnergy
```
Definition at line 22 of file global.h.
Referenced by AccumProps().

#### 5.58.3.151 stepAcf

```
int stepAcf
```
Definition at line 100 of file global.h.
Referenced by AccumVacf(), Init(), and PrintVacf().

#### 5.58.3.152 stepAvg

```
int stepAvg
```
Definition at line 24 of file global.h.
Referenced by AccumProps(), Init(), and main().

#### 5.58.3.153 stepCorr

```
int stepCorr
```
Definition at line 95 of file global.h.
Referenced by EvalSpacetimeCorr(), and Init().

#### 5.58.3.154 stepCount

```
int stepCount
```
Definition at line 24 of file global.h.
Referenced by BrownianStep(), HaltConditionCheck(), Init(), LeapfrogStep(), main(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.155 stepDump

```
int stepDump
```
Definition at line 24 of file global.h.
Referenced by Init(), and main().

#### 5.58.3.156 stepEquil

```
int stepEquil
```
Definition at line 24 of file global.h.
Referenced by BrownianStep(), Init(), LeapfrogStep(), and ReadBinaryRestart().

#### 5.58.3.157 stepLimit

```
int stepLimit
```
Definition at line 24 of file global.h.
Referenced by Init(), main(), and ReadBinaryRestart().

#### 5.58.3.158 stepRdf

```
int stepRdf
```
Definition at line 104 of file global.h.
Referenced by Init().

#### 5.58.3.159 stepTraj

```
int stepTraj
```
Definition at line 24 of file global.h.
Referenced by Init(), and main().

#### 5.58.3.160 sTotEnergy

```
double sTotEnergy
```
Definition at line 22 of file global.h.
Referenced by AccumProps().

#### 5.58.3.161 strain

```
double strain  [extern]
```
Referenced by ApplyShear(), Init(), and ReadBinaryRestart().

#### 5.58.3.162 strainRate

```
double strainRate
```
Definition at line 44 of file global.h.
Referenced by Init(), and ReadBinaryRestart().

#### 5.58.3.163 strech

```
double strech  [extern]
```
Referenced by ComputeBondForce(), and ComputePairForce().

#### 5.58.3.164 stress

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

#### 5.58.3.165 svirSum

```
double svirSum
```
Definition at line 21 of file global.h.
Referenced by AccumProps().

#### 5.58.3.166 timeNow

```
double timeNow
```
Definition at line 20 of file global.h.
Referenced by DumpBonds(), DumpPairs(), DumpRestart(), DumpState(), EvalCom(), EvalRdf(), HaltConditionCheck(), Init(), main(), PrintCom(), PrintForceSum(), PrintMomentum(), PrintStress(), PrintSummary(), PrintVrms(), ReadBinaryRestart(), Trajectory(), and WriteBinaryRestart().

#### 5.58.3.167 TotalBondEnergy

```
double TotalBondEnergy  [extern]
```
Referenced by ComputeBondForce(), EvalProps(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.168 TotalMass

```
double TotalMass
```
Definition at line 32 of file global.h.
Referenced by EvalCom().

#### 5.58.3.169 totEnergy

```
double totEnergy
```
Definition at line 20 of file global.h.
Referenced by AccumProps(), EvalProps(), HaltConditionCheck(), and PrintSummary().

#### 5.58.3.170 uSum

```
double uSum
```
Definition at line 21 of file global.h.
Referenced by ComputeForcesCells().

#### 5.58.3.171 uSumPair

```
double uSumPair  [extern]
```
Referenced by ComputePairForce(), EvalProps(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.172 uSumPairPerAtom

```
double uSumPairPerAtom
```
Definition at line 88 of file global.h.
Referenced by EvalProps(), HaltConditionCheck(), and PrintSummary().

#### 5.58.3.173 virSum

```
double virSum
```
Definition at line 21 of file global.h.
Referenced by AccumProps(), ComputeForcesCells(), EvalProps(), HaltConditionCheck(), and PrintSummary().

#### 5.58.3.174 virSumBond

```
double virSumBond  [extern]
```
Referenced by ComputeBondForce(), EvalProps(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.175 virSumBondxx

```
double virSumBondxx
```
Definition at line 89 of file global.h.
Referenced by ComputeBondForce(), EvalProps(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.176 virSumBondxy

```
double virSumBondxy
```
Definition at line 89 of file global.h.
Referenced by ComputeBondForce(), EvalProps(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.177 virSumBondyy

```
double virSumBondyy
```
Definition at line 89 of file global.h.
Referenced by ComputeBondForce(), EvalProps(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.178 virSumPair

```
double virSumPair
```
Definition at line 88 of file global.h.
Referenced by ComputePairForce(), EvalProps(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.179 virSumPairxx

```
double virSumPairxx
```
Definition at line 88 of file global.h.
Referenced by ComputePairForce(), EvalProps(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.180 virSumPairxy

```
double virSumPairxy
```
Definition at line 88 of file global.h.
Referenced by ComputePairForce(), EvalProps(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.181 virSumPairyy

```
double virSumPairyy
```
Definition at line 88 of file global.h.
Referenced by ComputePairForce(), EvalProps(), ReadBinaryRestart(), and WriteBinaryRestart().

#### 5.58.3.182 virSumxx

```
double virSumxx  [extern]
```
Referenced by EvalProps(), and PrintStress().

#### 5.58.3.183 virSumxy

```
double virSumxy
```
Definition at line 90 of file global.h.
Referenced by EvalProps(), and PrintStress().

#### 5.58.3.184 virSumyy

```
double virSumyy
```
Definition at line 90 of file global.h.
Referenced by EvalProps(), and PrintStress().

#### 5.58.3.185 visc

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

#### 5.58.3.186 viscAcf

```
double ** viscAcf
```
Definition at line 99 of file global.h.
Referenced by AccumVacf(), AllocArrays(), Close(), and EvalVacf().

#### 5.58.3.187 viscAcfAv

```
double * viscAcfAv
```
Definition at line 99 of file global.h.
Referenced by AccumVacf(), AllocArrays(), Close(), PrintVacf(), and ZeroVacf().

#### 5.58.3.188 viscAcfInt

```
double viscAcfInt
```
Definition at line 99 of file global.h.
Referenced by AccumVacf(), and PrintVacf().

#### 5.58.3.189 viscAcfOrg

```
double * viscAcfOrg
```
Definition at line 99 of file global.h.
Referenced by AllocArrays(), Close(), and EvalVacf().

#### 5.58.3.190 VMeanSqr

```
double VMeanSqr
```
Definition at line 46 of file global.h.
Referenced by EvalVrms().

#### 5.58.3.191 vrms

```
char vrms[256]  [extern]
```
Referenced by main().

#### 5.58.3.192 VRootMeanSqr

```
double VRootMeanSqr
```
Definition at line 46 of file global.h.
Referenced by EvalVrms(), main(), and PrintVrms().

#### 5.58.3.193 VSqr

```
double VSqr  [extern]
```
Referenced by EvalVrms().

#### 5.58.3.194 vSum

```
double vSum
```
Definition at line 21 of file global.h.
Referenced by EvalProps(), HaltConditionCheck(), and PrintSummary().

#### 5.58.3.195 vSumX

```
double vSumX
```
Definition at line 21 of file global.h.
Referenced by EvalProps(), and PrintMomentum().

#### 5.58.3.196 vSumY

```
double vSumY
```
Definition at line 21 of file global.h.
Referenced by EvalProps(), and PrintMomentum().

#### 5.58.3.197 vvSum

```
double vvSum
```
Definition at line 21 of file global.h.
Referenced by EvalProps(), and LeapfrogStep().

#### 5.58.3.198 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(), ReadBinaryRestart(), Trajectory(), VelocityVerletStep(), and WriteBinaryRestart().

#### 5.58.3.199 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(), ReadBinaryRestart(), Trajectory(), VelocityVerletStep(), and WriteBinaryRestart().

#### 5.58.3.200 xBoundary

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

Referenced by ApplyBoundaryCond(), Init(), and ReadBinaryRestart().

#### 5.58.3.201 xyz

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

Referenced by main().

#### 5.58.3.202 yBoundary

```
char yBoundary[10]
```

Definition at line 68 of file global.h.
Referenced by ApplyBoundaryCond(), Init(), and ReadBinaryRestart().

## 5.59 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, *fx, *fy;
00018 EXTERN double *speed;
00019
00020 EXTERN double region[NDIM+1], regionH[NDIM+1], deltaT, timeNow, potEnergy, kinEnergy, totEnergy,
      density,
00021 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
00027 EXTERN double RadiusIJ, SqrRadiusIJ, RadiusIJInv;
00028 EXTERN int    nAtomType;
00029 EXTERN int    *atomType;
00030 EXTERN int    *atomID;
00031 EXTERN double *atomRadius;
00032 EXTERN double *atomMass, TotalMass;
00033 EXTERN double mass; //mass from input file
```

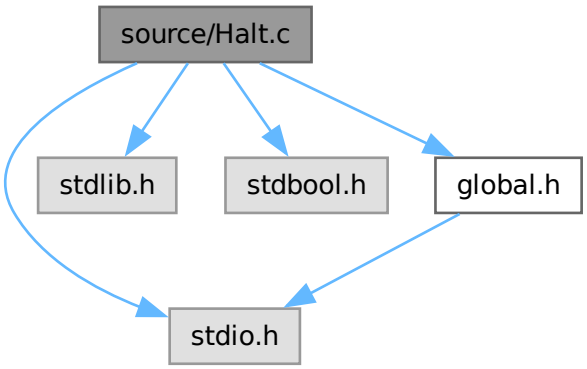
```
00034
00035 EXTERN int    nBond, nBondType;
00036 EXTERN int     *atom1, *atom2;
00037 EXTERN int     *BondID, *BondType ;
00038 EXTERN double *kb, *ro;
00039 EXTERN double *BondEnergy, *BondLength;
00040 EXTERN double TotalBondEnergy, BondEnergyPerAtom;
00041 EXTERN double gamman;
00042 EXTERN double *discDragx, *discDragy, *nodeDragx, *nodeDragy;
00043
00044 EXTERN double strain, strainRate;
00045 EXTERN double shearDisplacement, shearVelocity;
00046 EXTERN double VSqr, VMeanSqr, VRootMeanSqr;
00047 EXTERN double ComX, ComY, ComX0, ComY0, ComXRatio, ComYRatio;
00048 EXTERN double HaltCondition;
00049 EXTERN double DeltaY, DeltaX;
00050 EXTERN int    *ImageX, *ImageY;
00051 EXTERN double *rxUnwrap, *ryUnwrap;
00052 EXTERN double Kn;
00053 EXTERN double fxExtern, fyExtern, FyBylx, fxByfy, forceSumxExtern, forceSumyExtern;
00054 EXTERN int    DampFlag;
00055 EXTERN double strech;
00056
00057 //For reading the inputfile name
00058 EXTERN char mode[64], inputConfig[128];
00059
00060 //For dumping the pair interaction data
00061 EXTERN int    dumpPairFlag;
00062 EXTERN int    nPairTotal, nPairActive;
00063 EXTERN int    *PairID, *Pairatom1, *Pairatom2;
00064 EXTERN double *PairXij, *PairYij;
00065
00066
00067 EXTERN char   solver[128];
00068 EXTERN char   xBoundary[10], yBoundary[10];
00069
00070 //For molecule-ID as per LAMMPS, helpful!
00071 EXTERN int *molID;
00072 EXTERN int **isBonded;
00073
00074 //Interface properties
00075 EXTERN double InterfaceWidth, bigDiameter;
00076 EXTERN int  nAtomInterface,  nDiscInterface, nAtomBlock;
00077 EXTERN int  *atomIDInterface;
00078 EXTERN int BondPairFlag;
00079 EXTERN int nAtomInterfaceTotal; //The endLoop index in the ComputePairForce
00080
00081
00082 //Following three for MPI only
00083 EXTERN int     *cellList, cells[NDIM+1];
00084 EXTERN int     rank, size, master;
00085 EXTERN double *fax, *fay, fuSum, fvirSum, frfAtom;
00086
00087
00088 EXTERN double uSumPair, uSumPairPerAtom, virSumPair, virSumPairxx, virSumPairyy, virSumPairxy;
00089 EXTERN double virSumBond, virSumBondxx, virSumBondyy, virSumBondxy;
00090 EXTERN double virSumxx, virSumyy, virSumxy;
00091 EXTERN int    freezeAtomType;
00092
00093 // Spacetime Correlations
00094 EXTERN double **cfOrg, **spacetimeCorr, *cfVal, *spacetimeCorrAv;
00095 EXTERN int    *indexCorr, countCorrAv, limitCorrAv, nBuffCorr, nFunCorr, nValCorr, stepCorr;
00096
00097 // Viscosity
00098 EXTERN double rfAtom, frfAtom;
00099 EXTERN double *indexAcf, **viscAcf, *viscAcfOrg, *viscAcfAv, viscAcfInt;
00100 EXTERN int    nValAcf, nBuffAcf, stepAcf, countAcfAv, limitAcfAv;
00101
00102 // Radial distribution function
00103 EXTERN double *histRdf, rangeRdf;
00104 EXTERN int     countRdf, limitRdf, sizeHistRdf, stepRdf;
00105
00106
00107 // Output files prefixes
00108 EXTERN char   *prefix;
00109
00110 EXTERN char   result[250];
00111 EXTERN FILE   *fpresult;
00112
00113 EXTERN char   xyz[256];
00114 EXTERN FILE   *fpxyz;
00115
00116 EXTERN char   bond[256];
00117 EXTERN FILE   *fpbond;
00118
00119 EXTERN char   dump[256];
00120 EXTERN FILE   *fpdump;
```

```
00121
00122 EXTERN char   dnsty[256];
00123 EXTERN FILE   *fpdnsty;
00124
00125 EXTERN char  visc[256];
00126 EXTERN FILE   *fpvisc;
00127
00128 EXTERN char  rdf[256];
00129 EXTERN FILE  *fprdf;
00130
00131 EXTERN char  vrms[256];
00132 EXTERN FILE  *fpvrms;
00133
00134 EXTERN char  stress[256];
00135 EXTERN FILE  *fpstress;
00136
00137 EXTERN char  momentum[256];
00138 EXTERN FILE  *fpmomentum;
00139
00140 EXTERN char  force[256];
00141 EXTERN FILE  *fpforce;
00142
00143 EXTERN char  com[256];
00144 EXTERN FILE  *fpcom;
00145
00146 EXTERN char  pair[256];
00147 EXTERN FILE  *fppair;
00148
00149 #endif // GLOBALEXTERN_H
```

## 5.60 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)

### 5.60.1 Function Documentation

#### 5.60.1.1 HaltConditionCheck()

```
bool HaltConditionCheck (
            double value)
```

Definition at line 27 of file Halt.c.

```
00027                                     {
00028
00029   if(value <= HaltCondition && value != 0) {
00030   printf("Halt condition met at step = %d with Vrms = %.16f\n", stepCount, value);
00031   fprintf(fpresult, "Halt condition met at step = %d with Vrms = %.16f\n", stepCount, value);
00032   fprintf(fpresult, "Final thermodynamic values:\n");
00033   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",
00034   timeNow, vSum, potEnergy, kinEnergy, totEnergy, uSumPairPerAtom, BondEnergyPerAtom, pressure,
      virSum);
00035   return true;       // Signal that the halt condition is met
00036   }
00037  return false; // Halt condition not met
00038 }
```

References BondEnergyPerAtom, fpresult, HaltCondition, kinEnergy, potEnergy, pressure, stepCount, timeNow, totEnergy, uSumPairPerAtom, virSum, and vSum.

Referenced by main().

Here is the caller graph for this function:


## 5.61 Halt.c

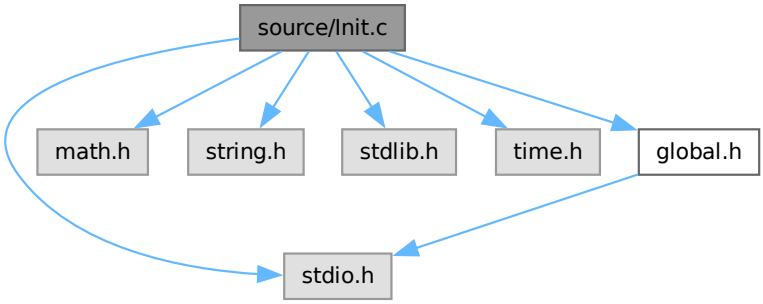
Go to the documentation of this file.

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022 #include <stdio.h>
00023 #include <stdlib.h>
00024 #include <stdbool.h>
00025 #include "global.h"
00026
00027 bool HaltConditionCheck(double value) {
00028
00029   if(value <= HaltCondition && value != 0) {
00030   printf("Halt condition met at step = %d with Vrms = %.16f\n", stepCount, value);
00031   fprintf(fpresult, "Halt condition met at step = %d with Vrms = %.16f\n", stepCount, value);
00032   fprintf(fpresult, "Final thermodynamic values:\n");
00033   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",
00034   timeNow, vSum, potEnergy, kinEnergy, totEnergy, uSumPairPerAtom, BondEnergyPerAtom, pressure,
      virSum);
00035   return true;       // Signal that the halt condition is met
00036   }
00037  return false; // Halt condition not met
00038 }
00039
```

## 5.62 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 ReadBinaryRestart (const char *filename)
- void Init ()

### 5.62.1 Function Documentation

#### 5.62.1.1 Init()

```
void Init ()
```

Definition at line 31 of file Init.c.

```
00031            {
00032  char dummy[128];
00033
00034  // Always read input parameters
00035  FILE *fp = fopen("input-data", "r");
00036  if(!fp) {
00037   perror("input-data");
00038   exit(EXIT_FAILURE);
00039  }
00040
00041   fscanf(fp, "%s %s", mode, inputConfig);  // config type + filename
00042   fscanf(fp, "%s %s", dummy, solver);
00043   fscanf(fp, "%s %s %s", dummy, xBoundary, yBoundary);
00044   fscanf(fp, "%s %d",  dummy, &DampFlag);
00045   fscanf(fp, "%s %d",  dummy, &freezeAtomType);
00046   fscanf(fp, "%s %lf", dummy, &rCut);
00047   fscanf(fp, "%s %lf", dummy, &Kn);
00048   fscanf(fp, "%s %lf", dummy, &mass);
00049   fscanf(fp, "%s %lf", dummy, &gamman);
00050   fscanf(fp, "%s %lf", dummy, &kappa);
00051   fscanf(fp, "%s %lf", dummy, &deltaT);
00052   fscanf(fp, "%s %lf", dummy, &strain);
00053   fscanf(fp, "%s %lf", dummy, &FyBylx);
00054   fscanf(fp, "%s %lf", dummy, &fxByfy);
00055   fscanf(fp, "%s %lf", dummy, &DeltaY);
00056   fscanf(fp, "%s %lf", dummy, &DeltaX);
00057   fscanf(fp, "%s %lf", dummy, &HaltCondition);
00058   fscanf(fp, "%s %d",  dummy, &stepAvg);
00059   fscanf(fp, "%s %d",  dummy, &stepEquil);
00060   fscanf(fp, "%s %d",  dummy, &stepLimit);
00061   fscanf(fp, "%s %d",  dummy, &stepDump);
```

```
00062   fscanf(fp, "%s %d",  dummy, &stepTraj);
00063   fscanf(fp, "%s %d",  dummy, &limitCorrAv);
00064   fscanf(fp, "%s %d",  dummy, &nBuffCorr);
00065   fscanf(fp, "%s %d",  dummy, &nFunCorr);
00066   fscanf(fp, "%s %d",  dummy, &nValCorr);
00067   fscanf(fp, "%s %d",  dummy, &stepCorr);
00068   fscanf(fp, "%s %d",  dummy, &limitAcfAv);
00069   fscanf(fp, "%s %d",  dummy, &nBuffAcf);
00070   fscanf(fp, "%s %d",  dummy, &nValAcf);
00071   fscanf(fp, "%s %d",  dummy, &stepAcf);
00072   fscanf(fp, "%s %lf", dummy, &rangeRdf);
00073   fscanf(fp, "%s %d",  dummy, &limitRdf);
00074   fscanf(fp, "%s %d",  dummy, &sizeHistRdf);
00075   fscanf(fp, "%s %d",  dummy, &stepRdf);
00076   fclose(fp);
00077
00078   int useBinaryRestart = 0;
00079   if(strcmp(mode, "read_restart") == 0) {
00080    useBinaryRestart = 1;
00081   } else if (strcmp(mode, "read_data") != 0) {
00082    fprintf(stderr, "ERROR: First line of input-data must be 'read_data' or 'read_restart'\n");
00083    exit(0);
00084   }
00085
00086   //Conditionally read binary config
00087   if(useBinaryRestart) {
00088    ReadBinaryRestart(inputConfig);  // uses global prefix + config file
00089    printf("»> Binary restart loaded from %s «<\n", inputConfig);
00090    printf("»> Restarting simulation from time = %.8lf «<\n", timeNow);
00091    return; //Exiting from Init() from here
00092   }
00093
00094   FILE *fpSTATE;
00095   if((fpSTATE = fopen(inputConfig,"r"))==NULL){
00096   printf("Error occurred: Could not open STATE file\n Exiting now..\n");
00097   exit(0);
00098   }
00099
00100   if(fscanf(fpSTATE, "%s %lf", dummy, &timeNow) != 2 || strcmp(dummy, "timeNow") != 0) {
00101    fprintf(stderr, "ERROR [%s:%d:%s]: Expected 'timeNow <value>' as the first line in the config
      file.\n",
00102    __FILE__, __LINE__, __func__);
00103    exit(EXIT_FAILURE);
00104   }
00105
00106   if(timeNow == 0.0) {
00107    printf("»> Running from time = 0.0: Beginning of the simulation\n");
00108    stepCount = 0;
00109   }
00110
00111   fscanf(fpSTATE, "%s %d",  dummy, &nAtom);
00112   fscanf(fpSTATE, "%s %d",  dummy, &nBond);
00113   fscanf(fpSTATE, "%s %d",  dummy, &nAtomType);
00114   fscanf(fpSTATE, "%s %d",  dummy, &nBondType);
00115   fscanf(fpSTATE, "%s %lf", dummy, &region[1]);
00116   fscanf(fpSTATE, "%s %lf", dummy, &region[2]);
00117
00118   if(timeNow == 0.0)  region[2] *= 1.5; //Remove this when put on GitHub
00119
00120   density = nAtom/(region[1]*region[2]);
00121   cells[1] = region[1] / rCut;
00122   cells[2] = region[2] / rCut;
00123   cellList = (int*)malloc((nAtom + cells[1] * cells[2] + 1) * sizeof(int));
00124   regionH[1] = 0.5*region[1];
00125   regionH[2] = 0.5*region[2];
00126
00127   //strain information
00128   strainRate = strain/deltaT;
00129   shearDisplacement = strain * region[2];
00130   shearVelocity = strainRate * region[2];
00131   int n;
00132
00133   rx = (double*)malloc((nAtom + 1) * sizeof(double));
00134   ry = (double*)malloc((nAtom + 1) * sizeof(double));
00135   vx = (double*)malloc((nAtom + 1) * sizeof(double));
00136   vy = (double*)malloc((nAtom + 1) * sizeof(double));
00137   ax = (double*)malloc((nAtom + 1) * sizeof(double));
00138   ay = (double*)malloc((nAtom + 1) * sizeof(double));
00139   fx = (double*)malloc((nAtom + 1) * sizeof(double));
00140   fy = (double*)malloc((nAtom + 1) * sizeof(double));
00141   fax = (double*)malloc((nAtom + 1) * sizeof(double));
00142   fay = (double*)malloc((nAtom + 1) * sizeof(double));
00143   atomID = (int*)malloc((nAtom+1) * sizeof(int));
00144   atomType = (int*)malloc((nAtom+1) * sizeof(int));
00145   atomRadius = (double*)malloc((nAtom + 1) * sizeof(double));
00146   atomMass = (double*)malloc((nAtom + 1) * sizeof(double));
00147   speed = (double*)malloc((nAtom + 1) * sizeof(double));
```

```
00148   discDragx = (double*)malloc((nAtom + 1) * sizeof(double));
00149   discDragy = (double*)malloc((nAtom + 1) * sizeof(double));
00150   molID = (int*)malloc((nAtom+1) * sizeof(int));
00151
00152   BondID = (int*)malloc((nBond+1)*sizeof(int));
00153   BondType = (int*)malloc((nBond+1)*sizeof(int));
00154   atom1 = (int*)malloc((nBond+1)*sizeof(int));
00155   atom2 = (int*)malloc((nBond+1)*sizeof(int));
00156   kb = (double*)malloc((nBond+1)*sizeof(double));
00157   ro = (double*)malloc((nBond+1)*sizeof(double));
00158   BondEnergy = (double*)malloc((nBond+1)*sizeof(double));
00159   BondLength =(double*)malloc((nBond+1)*sizeof(double));
00160   nodeDragx = (double*)malloc((nAtom + 1) * sizeof(double));
00161   nodeDragy = (double*)malloc((nAtom + 1) * sizeof(double));
00162   rxUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
00163   ryUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
00164   ImageX = (int*)malloc((nAtom+1) * sizeof(int));
00165   ImageY = (int*)malloc((nAtom+1) * sizeof(int));
00166
00167
00168   for(n = 1; n <= nAtom; n ++){
00169    atomMass[n] = mass;
00170   }
00171
00172   fscanf(fpSTATE, "%s\n", dummy);
00173   for(n = 1; n <= nAtom; n ++)
00174    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]);
00175
00176
00177   fscanf(fpSTATE, "%s\n", dummy);
00178   for(n=1; n<=nBond; n++)
00179    fscanf(fpSTATE, "%d %d %d %d %lf %lf\n", &BondID[n], &BondType[n], &atom1[n], &atom2[n], &kb[n],
      &ro[n]);
00180
00181   fclose(fpSTATE);
00182
00183  //2D-List of bonded atoms. This is used to remove pair interaction
00184  //calculation for the bonded atoms
00185    isBonded = (int**)malloc((nAtom + 1) * sizeof(int*));
00186   for (int i = 0; i <= nAtom; i++) {
00187     isBonded[i] = (int*)malloc((nAtom + 1) * sizeof(int));
00188     for (int j = 0; j <= nAtom; j++) {
00189         isBonded[i][j] = 0;
00190     }
00191   }
00192
00193   for (n = 1; n <= nBond; n++) {
00194     int i = atom1[n];
00195     int j = atom2[n];
00196     isBonded[i][j] = 1;
00197     isBonded[j][i] = 1; // symmetric
00198 }
00199
00200  // List the interface atoms
00201  nAtomInterface = 0;
00202  nAtomBlock = 0;
00203  nDiscInterface = 0;
00204  bigDiameter = 2.8;
00205  InterfaceWidth = 5.0 * bigDiameter;
00206
00207  for(n = 1; n <= nAtom; n++){
00208   if(fabs(ry[n]) < InterfaceWidth){
00209   nAtomInterface++;
00210   }
00211   if(molID[n] == 2){
00212   nAtomBlock++;
00213   }
00214   if(atomRadius[n] != 0.0){
00215   nDiscInterface++;
00216   } }
00217
00218
00219   int BondPairInteract;
00220   BondPairInteract = 0;
00221   int m;
00222   if(BondPairInteract == 1){
00223    atomIDInterface =  (int *)malloc((nAtomInterface+1)*sizeof(int));
00224    m = 1;
00225    for(n=1; n<=nAtom; n++){
00226     if(fabs(ry[n]) < InterfaceWidth){
00227     atomIDInterface[m] = atomID[n];
00228     m++;
00229   } } }
00230   else if(BondPairInteract == 0){
00231    nAtomInterface = nDiscInterface;
00232    atomIDInterface =  (int *)malloc((nAtomInterface+1)*sizeof(int));
```

```
00233     m = 1;
00234     for(n=1; n<=nAtom; n++){
00235      if(atomRadius[n] != 0.0){
00236      atomIDInterface[m] = atomID[n];
00237      m++;
00238    } } }
00239
00240    nPairTotal = 0.5 * nAtomInterface * (nAtomInterface-1);
00241    PairID = (int*)malloc((nPairTotal+1) * sizeof(int));
00242    Pairatom1 = (int*)malloc((nPairTotal+1) * sizeof(int));
00243    Pairatom2 = (int*)malloc((nPairTotal+1) * sizeof(int));
00244    PairXij = (double*)malloc((nPairTotal+1) * sizeof(double));
00245    PairYij = (double*)malloc((nPairTotal+1) * sizeof(double));
00246
00247    fprintf(fpresult, "-------------------------------------\n");
00248    fprintf(fpresult, "-------PARAMETERS-----------\n");
00249    fprintf(fpresult, "-------------------------------------\n");
00250    fprintf(fpresult, "nAtom\t\t\t%d\n",  nAtom);
00251    fprintf(fpresult, "nBond\t\t\t%d\n",  nBond);
00252    fprintf(fpresult, "nAtomType\t\t%d\n", nAtomType);
00253    fprintf(fpresult, "nBondType\t\t%d\n",  nBondType);
00254    fprintf(fpresult, "nAtomBlock\t\t%d\n",  nAtomBlock);
00255    fprintf(fpresult, "nAtomInterface\t\t%d\n",  nAtomInterface);
00256    fprintf(fpresult, "nDiscInterface\t\t%d\n",  nDiscInterface);
00257    fprintf(fpresult, "mass\t\t\t%0.6g\n", mass);
00258    fprintf(fpresult, "gamman\t\t\t%0.6g\n", gamman);
00259    fprintf(fpresult, "strain\t\t\t%0.6g\n", strain);
00260    fprintf(fpresult, "strainRate\t\t%0.6g\n", strainRate);
00261    fprintf(fpresult, "FyBylx\t\t\t%0.6g\n", FyBylx);
00262    fprintf(fpresult, "fxByfy\t\t\t%0.6g\n", fxByfy);
00263    fprintf(fpresult, "DeltaY\t\t\t%0.6g\n", DeltaY);
00264    fprintf(fpresult, "DeltaX\t\t\t%0.6g\n", DeltaX);
00265    fprintf(fpresult, "HaltCondition\t\t%0.6g\n", HaltCondition);
00266    fprintf(fpresult, "kappa\t\t\t%g\n", kappa);
00267    fprintf(fpresult, "density\t\t\t%g\n", density);
00268    fprintf(fpresult, "rCut\t\t\t%g\n", rCut);
00269    fprintf(fpresult, "deltaT\t\t\t%g\n", deltaT);
00270    fprintf(fpresult, "stepEquil\t\t%d\n",  stepEquil);
00271    fprintf(fpresult, "stepLimit\t\t%d\n",  stepLimit);
00272    fprintf(fpresult, "region[1]\t\t%0.16lf\n", region[1]);
00273    fprintf(fpresult, "region[2]\t\t%0.16lf\n", region[2]);
00274    fprintf(fpresult, "cells[1]\t\t%d\n",  cells[1]);
00275    fprintf(fpresult, "cells[2]\t\t%d\n",  cells[2]);
00276    fprintf(fpresult, "solver\t\t\t%s\n",  solver);
00277    fprintf(fpresult, "boundary\t\t%s %s\n", xBoundary, yBoundary);
00278    fprintf(fpresult, "DampFlag\t\t%d\n",  DampFlag);
00279    fprintf(fpresult, "-------------------------------------\n");
00280    fprintf(fpresult, "#TimeNow TotalMomentum PotEngyPerAtom KinEngyPerAtom TotEngyPerAtom
       PairEnergyPerAtom BondEnergyPerAtom  Press VirialSum\n");
00281    fprintf(fpvrms, "#timeNow\tVrms \n");
00282    fprintf(fpcom, "#timeNow\tComX\tComY\n");
00283    fprintf(fpforce,
       "#timeNow\tforceSumxAtomType1\tforceSumyAtomType1\tforceSumxAtomType2\tforceSumyAtomType2\tforceSumxAtomType3\tforceSumy
00284
00285 /* //Uncomment the following as per your acquirement
00286     fprintf(fpstress, "strain           %lf\n", strain);
00287     fprintf(fpstress, "region[1]        %lf\n", region[1]);
00288     fprintf(fpstress, "region[2]        %lf\n", region[2]);
00289     fprintf(fpstress, "#timeNow virSumxx virSumyy virSumxy pressure\n");
00290     fprintf(fpmomentum, "#timeNow Px Py\n");
00291 */
00292
00293    if((strcmp(xBoundary, "p") != 0 && strcmp(xBoundary, "r") != 0) ||
00294     (strcmp(yBoundary, "p") != 0 && strcmp(yBoundary, "r") != 0)) {
00295     fprintf(fpresult, "Error: Invalid boundary value detected: '%s %s'. Only 'p' or 'r' are
       allowed.\n", xBoundary, yBoundary);
00296     exit(EXIT_FAILURE); // Exit with failure status
00297    }
00298
00299 }
```

References atom1, atom2, atomID, atomIDInterface, atomMass, atomRadius, atomType, ax, ay, bigDiameter, BondEnergy, BondID, BondLength, BondType, cellList, cells, DampFlag, deltaT, DeltaX, DeltaY, density, discDragx, discDragy, fax, fay, fpcom, fpforce, fpresult, fpvrms, freezeAtomType, fx, fxByfy, fy, FyBylx, gamman, HaltCondition, ImageX, ImageY, inputConfig, InterfaceWidth, isBonded, kappa, kb, Kn, limitAcfAv, limitCorrAv, limitRdf, mass, mode, molID, nAtom, nAtomBlock, nAtomInterface, nAtomType, nBond, nBondType, nBuffAcf, nBuffCorr, nDiscInterface, nFunCorr, nodeDragx, nodeDragy, nPairTotal, nValAcf, nValCorr, Pairatom1, Pairatom2, PairID, PairXij, PairYij, rangeRdf, rCut, ReadBinaryRestart(), region, regionH, ro, rx, rxUnwrap, ry, ryUnwrap, shearDisplacement, shearVelocity, sizeHistRdf, solver, speed, stepAcf, stepAvg, stepCorr, stepCount, stepDump, stepEquil, stepLimit, stepRdf, stepTraj, strain, strainRate, timeNow, vx, vy, xBoundary, and yBoundary.
Referenced by main().

Here is the call graph for this function:


Here is the caller graph for this function:


#### 5.62.1.2 ReadBinaryRestart()

```
void ReadBinaryRestart (
            const char * filename)
```

Definition at line 59 of file ReadRestartBinary.c.

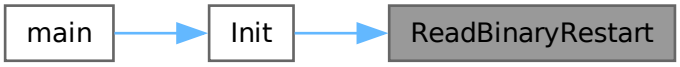
```
00059                                           {
00060
00061   FILE *fp = fopen(filename, "rb");
00062   if (!fp) {
00063     fprintf(stderr, "Error opening binary restart file %s for reading\n", filename);
00064     exit(EXIT_FAILURE);
00065   }
00066   RestartHeader hdr;  //Declare here
00067   fread(&hdr, sizeof(RestartHeader), 1, fp);   //Use it
00068
00069   if(strncmp(hdr.magic, "LAMINA", 6) != 0) {
00070    fprintf(stderr, "Invalid file format: magic = %.8s [from %s()]\n", hdr.magic, __func__);
00071    fclose(fp);
00072    exit(EXIT_FAILURE);  //Must return void, not return 1
00073   }
00074
00075   //Now assigned the values that were read from binary file to global parameters
00076   timeNow = hdr.timeNow;
00077   nAtom = hdr.nAtom;
00078   nBond = hdr.nBond;
00079   nAtomType = hdr.nAtomType;
00080   nBondType = hdr.nBondType;
00081   region[1] = hdr.regionX;
00082   region[2] = hdr.regionY;
00083   nAtomInterface = hdr.nAtomInterface;
00084   nAtomBlock = hdr.nAtomBlock;
00085   nDiscInterface = hdr.nDiscInterface;
00086   bigDiameter = hdr.bigDiameter;
00087   InterfaceWidth = hdr.InterfaceWidth;
00088   nPairActive = hdr.nPairActive;
00089   nPairTotal = hdr.nPairTotal;
00090   uSumPair = hdr.uSumPair;
00091   virSumPair = hdr.virSumPair;
00092   virSumPairxx = hdr.virSumPairxx;
00093   virSumPairyy = hdr.virSumPairyy;
00094   virSumPairxy = hdr.virSumPairxy;
00095   TotalBondEnergy = hdr.TotalBondEnergy;
00096   virSumBond = hdr.virSumBond;
00097   virSumBondxx = hdr.virSumBondxx;
00098   virSumBondyy = hdr.virSumBondyy;
00099   virSumBondxy = hdr.virSumBondxy;
```

```
00100   stepCount = hdr.stepCount;
00101   forceSumxExtern = hdr.forceSumxExtern;
00102   forceSumyExtern = hdr.forceSumyExtern;
00103
00104   density = nAtom / (region[1] * region[2]);
00105   cells[1] = region[1] / rCut;
00106   cells[2] = region[2] / rCut;
00107   regionH[1] = 0.5 * region[1];
00108   regionH[2] = 0.5 * region[2];
00109
00110   cellList = (int *)malloc((nAtom + cells[1] * cells[2] + 1) * sizeof(int));
00111
00112   printf("Running from restart file:\n");
00113   printf("Running: %s, version: %0.3lf\n", hdr.magic, hdr.version);
00114   printf("timeNow: %lf\n", timeNow);
00115   printf("stepCount: %d\n", stepCount);
00116
00117   //Allocating the memory to arrays
00118   atomID = (int *)malloc((nAtom + 1) * sizeof(int));
00119   molID = (int *)malloc((nAtom + 1) * sizeof(int));
00120   atomType = (int *)malloc((nAtom + 1) * sizeof(int));
00121   atomRadius = (double *)malloc((nAtom + 1) * sizeof(double));
00122   rx = (double *)malloc((nAtom + 1) * sizeof(double));
00123   ry = (double *)malloc((nAtom + 1) * sizeof(double));
00124   vx = (double *)malloc((nAtom + 1) * sizeof(double));
00125   vy = (double *)malloc((nAtom + 1) * sizeof(double));
00126   ax = (double *)malloc((nAtom + 1) * sizeof(double));
00127   ay = (double *)malloc((nAtom + 1) * sizeof(double));
00128   fx = (double *)malloc((nAtom + 1) * sizeof(double));
00129   fy = (double *)malloc((nAtom + 1) * sizeof(double));
00130   atomMass = (double *)malloc((nAtom + 1) * sizeof(double));
00131   discDragx = (double *)malloc((nAtom + 1) * sizeof(double));
00132   discDragy = (double *)malloc((nAtom + 1) * sizeof(double));
00133   atomIDInterface = (int *)malloc((nAtom + 1) * sizeof(int));
00134
00135   BondID = (int *)malloc((nBond + 1) * sizeof(int));
00136   BondType = (int *)malloc((nBond + 1) * sizeof(int));
00137   atom1 = (int *)malloc((nBond + 1) * sizeof(int));
00138   atom2 = (int *)malloc((nBond + 1) * sizeof(int));
00139   kb = (double *)malloc((nBond + 1) * sizeof(double));
00140   ro = (double *)malloc((nBond + 1) * sizeof(double));
00141   BondEnergy = (double *)malloc((nBond + 1) * sizeof(double));
00142   BondLength = (double *)malloc((nBond + 1) * sizeof(double));
00143   nodeDragx = (double *)malloc((nAtom + 1) * sizeof(double));
00144   nodeDragy = (double *)malloc((nAtom + 1) * sizeof(double));
00145   rxUnwrap = (double *)malloc((nAtom + 1) * sizeof(double));
00146   ryUnwrap = (double *)malloc((nAtom + 1) * sizeof(double));
00147   ImageX = (int *)malloc((nAtom + 1) * sizeof(int));
00148   ImageY = (int *)malloc((nAtom + 1) * sizeof(int));
00149
00150   PairID = (int *)malloc((nPairTotal + 1) * sizeof(int));
00151   Pairatom1 = (int *)malloc((nPairTotal + 1) * sizeof(int));
00152   Pairatom2 = (int *)malloc((nPairTotal + 1) * sizeof(int));
00153   PairXij = (double *)malloc((nPairTotal + 1) * sizeof(double));
00154   PairYij = (double *)malloc((nPairTotal + 1) * sizeof(double));
00155
00156  //Here we are reading the data to binary file
00157  fread(&atomID[1], sizeof(int), nAtom, fp);
00158  fread(&molID[1], sizeof(int), nAtom, fp);
00159  fread(&atomType[1], sizeof(int), nAtom, fp);
00160  fread(&atomRadius[1], sizeof(double), nAtom, fp);
00161  fread(&rx[1], sizeof(double), nAtom, fp);
00162  fread(&ry[1], sizeof(double), nAtom, fp);
00163  fread(&vx[1], sizeof(double), nAtom, fp);
00164  fread(&vy[1], sizeof(double), nAtom, fp);
00165  fread(&ax[1], sizeof(double), nAtom, fp);
00166  fread(&ay[1], sizeof(double), nAtom, fp);
00167  fread(&fx[1], sizeof(double), nAtom, fp);
00168  fread(&fy[1], sizeof(double), nAtom, fp);
00169  fread(&atomMass[1], sizeof(double), nAtom, fp);
00170  fread(&discDragx[1], sizeof(double), nAtom, fp);
00171  fread(&discDragy[1], sizeof(double), nAtom, fp);
00172  fread(&atomIDInterface[1], sizeof(int), nAtomInterface, fp);
00173
00174  fread(&BondID[1], sizeof(int), nBond, fp);
00175  fread(&BondType[1], sizeof(int), nBond, fp);
00176  fread(&atom1[1], sizeof(int), nBond, fp);
00177  fread(&atom2[1], sizeof(int), nBond, fp);
00178  fread(&kb[1], sizeof(double), nBond, fp);
00179  fread(&ro[1], sizeof(double), nBond, fp);
00180  fread(&BondEnergy[1], sizeof(double), nBond, fp);
00181  fread(&BondLength[1], sizeof(double), nBond, fp);
00182  fread(&nodeDragx[1], sizeof(double), nAtom, fp);
00183  fread(&nodeDragy[1], sizeof(double), nAtom, fp);
00184  fread(&rxUnwrap[1], sizeof(double), nAtom, fp);
00185  fread(&ryUnwrap[1], sizeof(double), nAtom, fp);
00186  fread(&ImageX[1], sizeof(int), nAtom, fp);
```

```
00187   fread(&ImageY[1], sizeof(int), nAtom, fp);
00188
00189   fread(&PairID[1], sizeof(int), nPairActive, fp);
00190   fread(&Pairatom1[1], sizeof(int), nPairActive, fp);
00191   fread(&Pairatom2[1], sizeof(int), nPairActive, fp);
00192   fread(&PairXij[1], sizeof(double), nPairActive, fp);
00193   fread(&PairYij[1], sizeof(double), nPairActive, fp);
00194
00195   //2D-List of bonded atoms. This is used to remove pair interaction
00196   //calculation for the bonded atoms
00197   isBonded = (int **)malloc((nAtom + 1) * sizeof(int*));
00198   for (int i = 0; i <= nAtom; i++) {
00199    isBonded[i] = (int *)malloc((nAtom + 1) * sizeof(int));
00200    for (int j = 0; j <= nAtom; j++) {
00201    isBonded[i][j] = 0;
00202    } }
00203    for (int n = 1; n <= nBond; n++) {
00204     int i = atom1[n];
00205     int j = atom2[n];
00206     isBonded[i][j] = 1;
00207     isBonded[j][i] = 1; // symmetric
00208  }
00209
00210    fprintf(fpresult, "-----------------------------------\n");
00211    fprintf(fpresult, "-------PARAMETERS-----------\n");
00212    fprintf(fpresult, "-----------------------------------\n");
00213    fprintf(fpresult, "nAtom\t\t\t%d\n",   nAtom);
00214    fprintf(fpresult, "nBond\t\t\t%d\n",   nBond);
00215    fprintf(fpresult, "nAtomType\t\t%d\n", nAtomType);
00216    fprintf(fpresult, "nBondType\t\t%d\n",   nBondType);
00217    fprintf(fpresult, "nAtomBlock\t\t%d\n",   nAtomBlock);
00218    fprintf(fpresult, "nAtomInterface\t\t%d\n",   nAtomInterface);
00219    fprintf(fpresult, "nDiscInterface\t\t%d\n",   nDiscInterface);
00220    fprintf(fpresult, "mass\t\t\t%0.6g\n", mass);
00221    fprintf(fpresult, "gamman\t\t\t%0.6g\n", gamman);
00222    fprintf(fpresult, "strain\t\t\t%0.6g\n", strain);
00223    fprintf(fpresult, "strainRate\t\t%0.6g\n", strainRate);
00224    fprintf(fpresult, "FyBylx\t\t\t%0.6g\n", FyBylx);
00225    fprintf(fpresult, "fxByfy\t\t\t%0.6g\n", fxByfy);
00226    fprintf(fpresult, "DeltaY\t\t\t%0.6g\n", DeltaY);
00227    fprintf(fpresult, "DeltaX\t\t\t%0.6g\n", DeltaX);
00228    fprintf(fpresult, "HaltCondition\t\t%0.6g\n", HaltCondition);
00229    fprintf(fpresult, "kappa\t\t\t%g\n", kappa);
00230    fprintf(fpresult, "density\t\t\t%g\n", density);
00231    fprintf(fpresult, "rCut\t\t\t%g\n", rCut);
00232    fprintf(fpresult, "deltaT\t\t\t%g\n", deltaT);
00233    fprintf(fpresult, "stepEquil\t\t%d\n",   stepEquil);
00234    fprintf(fpresult, "stepLimit\t\t%d\n",   stepLimit);
00235    fprintf(fpresult, "region[1]\t\t%0.16lf\n", region[1]);
00236    fprintf(fpresult, "region[2]\t\t%0.16lf\n", region[2]);
00237    fprintf(fpresult, "cells[1]\t\t%d\n",   cells[1]);
00238    fprintf(fpresult, "cells[2]\t\t%d\n",   cells[2]);
00239    fprintf(fpresult, "solver\t\t\t%s\n",   solver);
00240    fprintf(fpresult, "boundary\t\t%s %s\n", xBoundary, yBoundary);
00241    fprintf(fpresult, "DampFlag\t\t%d\n",   DampFlag);
00242    fprintf(fpresult, "-----------------------------------\n");
00243    fprintf(fpresult, "#TimeNow TotalMomentum PotEngyPerAtom KinEngyPerAtom TotEngyPerAtom
       PairEnergyPerAtom BondEnergyPerAtom  Press VirialSum\n");
00244    fprintf(fpvrms, "#timeNow\tVrms \n");
00245    fprintf(fpcom, "#timeNow\tComX\tComY\n");
00246    fprintf(fpforce,
       "#timeNow\tforceSumxAtomType1\tforceSumyAtomType1\tforceSumxAtomType2\tforceSumyAtomType2\tforceSumxAtomType3\tforceSumy
00247    fclose(fp);
00248   }
```

References atom1, atom2, atomID, atomIDInterface, atomMass, atomRadius, atomType, ax, ay, bigDiameter, RestartHeader::bigDiameter, BondEnergy, BondID, BondLength, BondType, cellList, cells, DampFlag, deltaT, DeltaX, DeltaY, density, discDragx, discDragy, forceSumxExtern, RestartHeader::forceSumxExtern, forceSumyExtern, RestartHeader::forceSumyExtern, fpcom, fpforce, fpresult, fpvrms, fx, fxByfy, fy, FyBylx, gamman, HaltCondition, ImageX, ImageY, InterfaceWidth, RestartHeader::InterfaceWidth, isBonded, kappa, kb, RestartHeader::magic, mass, molID, nAtom, RestartHeader::nAtom, nAtomBlock, RestartHeader::nAtomBlock, nAtomInterface, RestartHeader::nAtomInterface, nAtomType, RestartHeader::nAtomType, nBond, RestartHeader::nBond, nBondType, RestartHeader::nBondType, nDiscInterface, RestartHeader::nDiscInterface, nodeDragx, nodeDragy, nPairActive, RestartHeader::nPairActive, nPairTotal, RestartHeader::nPairTotal, Pairatom1, Pairatom2, PairID, PairXij, PairYij, rCut, region, regionH, RestartHeader::regionX, RestartHeader::regionY, ro, rx, rxUnwrap, ry, ryUnwrap, solver, RestartHeader::stepCount, stepCount, stepEquil, stepLimit, strain, strainRate, RestartHeader::timeNow, timeNow, RestartHeader::TotalBondEnergy, TotalBondEnergy, RestartHeader::uSumPair, uSumPair, RestartHeader::version, RestartHeader::virSumBond, virSumBond, RestartHeader::virSumBondxx, virSumBondxx, RestartHeader::virSumBondxy, virSumBondxy, RestartHeader::virSumBondyy, virSumBondyy, RestartHeader::virSumPair, virSumPair, RestartHeader::virSumPairxx, virSumPairxx, RestartHeader::virSumPairxy, virSumPairxy, RestartHeader::virSumPairyy,

virSumPairyy, vx, vy, xBoundary, and yBoundary.
Referenced by Init().
Here is the caller graph for this function:




## 5.63 Init.c

Go to the documentation of this file.

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022 #include<stdio.h>
00023 #include<math.h>
00024 #include<string.h>
00025 #include<stdlib.h>
00026 #include <time.h>
00027 #include"global.h"
00028
00029 void ReadBinaryRestart(const char *filename);
00030
00031 void Init() {
00032  char dummy[128];
00033
00034  // Always read input parameters
00035  FILE *fp = fopen("input-data", "r");
00036  if(!fp) {
00037   perror("input-data");
00038   exit(EXIT_FAILURE);
00039  }
00040
00041   fscanf(fp, "%s %s", mode, inputConfig);  // config type + filename
00042   fscanf(fp, "%s %s", dummy, solver);
00043   fscanf(fp, "%s %s %s", dummy, xBoundary, yBoundary);
00044   fscanf(fp, "%s %d",  dummy, &DampFlag);
00045   fscanf(fp, "%s %d",  dummy, &freezeAtomType);
00046   fscanf(fp, "%s %lf", dummy, &rCut);
00047   fscanf(fp, "%s %lf", dummy, &Kn);
00048   fscanf(fp, "%s %lf", dummy, &mass);
00049   fscanf(fp, "%s %lf", dummy, &gamman);
00050   fscanf(fp, "%s %lf", dummy, &kappa);
00051   fscanf(fp, "%s %lf", dummy, &deltaT);
00052   fscanf(fp, "%s %lf", dummy, &strain);
00053   fscanf(fp, "%s %lf", dummy, &FyBylx);
00054   fscanf(fp, "%s %lf", dummy, &fxByfy);
00055   fscanf(fp, "%s %lf", dummy, &DeltaY);
00056   fscanf(fp, "%s %lf", dummy, &DeltaX);
00057   fscanf(fp, "%s %lf", dummy, &HaltCondition);
00058   fscanf(fp, "%s %d",  dummy, &stepAvg);
00059   fscanf(fp, "%s %d",  dummy, &stepEquil);
00060   fscanf(fp, "%s %d",  dummy, &stepLimit);
```

```
00061   fscanf(fp, "%s %d",  dummy, &stepDump);
00062   fscanf(fp, "%s %d",  dummy, &stepTraj);
00063   fscanf(fp, "%s %d",  dummy, &limitCorrAv);
00064   fscanf(fp, "%s %d",  dummy, &nBuffCorr);
00065   fscanf(fp, "%s %d",  dummy, &nFunCorr);
00066   fscanf(fp, "%s %d",  dummy, &nValCorr);
00067   fscanf(fp, "%s %d",  dummy, &stepCorr);
00068   fscanf(fp, "%s %d",  dummy, &limitAcfAv);
00069   fscanf(fp, "%s %d",  dummy, &nBuffAcf);
00070   fscanf(fp, "%s %d",  dummy, &nValAcf);
00071   fscanf(fp, "%s %d",  dummy, &stepAcf);
00072   fscanf(fp, "%s %lf", dummy, &rangeRdf);
00073   fscanf(fp, "%s %d",  dummy, &limitRdf);
00074   fscanf(fp, "%s %d",  dummy, &sizeHistRdf);
00075   fscanf(fp, "%s %d",  dummy, &stepRdf);
00076   fclose(fp);
00077
00078   int useBinaryRestart = 0;
00079   if(strcmp(mode, "read_restart") == 0) {
00080    useBinaryRestart = 1;
00081   } else if (strcmp(mode, "read_data") != 0) {
00082    fprintf(stderr, "ERROR: First line of input-data must be 'read_data' or 'read_restart'\n");
00083    exit(0);
00084   }
00085
00086   //Conditionally read binary config
00087   if(useBinaryRestart) {
00088    ReadBinaryRestart(inputConfig);  // uses global prefix + config file
00089    printf("»> Binary restart loaded from %s ««\n", inputConfig);
00090    printf("»> Restarting simulation from time = %.8lf ««\n", timeNow);
00091    return; //Exiting from Init() from here
00092   }
00093
00094   FILE *fpSTATE;
00095   if((fpSTATE = fopen(inputConfig,"r"))==NULL){
00096   printf("Error occurred: Could not open STATE file\n Exiting now..\n");
00097   exit(0);
00098   }
00099
00100   if(fscanf(fpSTATE, "%s %lf", dummy, &timeNow) != 2 || strcmp(dummy, "timeNow") != 0) {
00101    fprintf(stderr, "ERROR [%s:%d:%s]: Expected 'timeNow <value>' as the first line in the config
      file.\n",
00102    __FILE__, __LINE__, __func__);
00103    exit(EXIT_FAILURE);
00104   }
00105
00106   if(timeNow == 0.0) {
00107    printf("»> Running from time = 0.0: Beginning of the simulation\n");
00108    stepCount = 0;
00109   }
00110
00111   fscanf(fpSTATE, "%s %d",  dummy, &nAtom);
00112   fscanf(fpSTATE, "%s %d",  dummy, &nBond);
00113   fscanf(fpSTATE, "%s %d",  dummy, &nAtomType);
00114   fscanf(fpSTATE, "%s %d",  dummy, &nBondType);
00115   fscanf(fpSTATE, "%s %lf", dummy, &region[1]);
00116   fscanf(fpSTATE, "%s %lf", dummy, &region[2]);
00117
00118   if(timeNow == 0.0)  region[2] *= 1.5; //Remove this when put on GitHub
00119
00120   density = nAtom/(region[1]*region[2]);
00121   cells[1] = region[1] / rCut;
00122   cells[2] = region[2] / rCut;
00123   cellList = (int*)malloc((nAtom + cells[1] * cells[2] + 1) * sizeof(int));
00124   regionH[1] = 0.5*region[1];
00125   regionH[2] = 0.5*region[2];
00126
00127   //strain information
00128   strainRate = strain/deltaT;
00129   shearDisplacement = strain * region[2];
00130   shearVelocity = strainRate * region[2];
00131   int n;
00132
00133   rx = (double*)malloc((nAtom + 1) * sizeof(double));
00134   ry = (double*)malloc((nAtom + 1) * sizeof(double));
00135   vx = (double*)malloc((nAtom + 1) * sizeof(double));
00136   vy = (double*)malloc((nAtom + 1) * sizeof(double));
00137   ax = (double*)malloc((nAtom + 1) * sizeof(double));
00138   ay = (double*)malloc((nAtom + 1) * sizeof(double));
00139   fx = (double*)malloc((nAtom + 1) * sizeof(double));
00140   fy = (double*)malloc((nAtom + 1) * sizeof(double));
00141   fax = (double*)malloc((nAtom + 1) * sizeof(double));
00142   fay = (double*)malloc((nAtom + 1) * sizeof(double));
00143   atomID = (int*)malloc((nAtom+1) * sizeof(int));
00144   atomType = (int*)malloc((nAtom+1) * sizeof(int));
00145   atomRadius = (double*)malloc((nAtom + 1) * sizeof(double));
00146   atomMass = (double*)malloc((nAtom + 1) * sizeof(double));
```

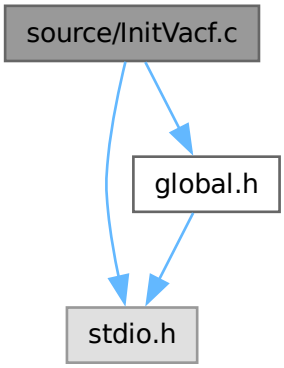
```
00147   speed = (double*)malloc((nAtom + 1) * sizeof(double));
00148   discDragx = (double*)malloc((nAtom + 1) * sizeof(double));
00149   discDragy = (double*)malloc((nAtom + 1) * sizeof(double));
00150   molID = (int*)malloc((nAtom+1) * sizeof(int));
00151
00152   BondID = (int*)malloc((nBond+1)*sizeof(int));
00153   BondType = (int*)malloc((nBond+1)*sizeof(int));
00154   atom1 = (int*)malloc((nBond+1)*sizeof(int));
00155   atom2 = (int*)malloc((nBond+1)*sizeof(int));
00156   kb = (double*)malloc((nBond+1)*sizeof(double));
00157   ro = (double*)malloc((nBond+1)*sizeof(double));
00158   BondEnergy = (double*)malloc((nBond+1)*sizeof(double));
00159   BondLength =(double*)malloc((nBond+1)*sizeof(double));
00160   nodeDragx = (double*)malloc((nAtom + 1) * sizeof(double));
00161   nodeDragy = (double*)malloc((nAtom + 1) * sizeof(double));
00162   rxUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
00163   ryUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
00164   ImageX = (int*)malloc((nAtom+1) * sizeof(int));
00165   ImageY = (int*)malloc((nAtom+1) * sizeof(int));
00166
00167
00168   for(n = 1; n <= nAtom; n ++){
00169    atomMass[n] = mass;
00170   }
00171
00172   fscanf(fpSTATE, "%s\n", dummy);
00173   for(n = 1; n <= nAtom; n ++)
00174    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]);
00175
00176
00177   fscanf(fpSTATE, "%s\n", dummy);
00178   for(n=1; n<=nBond; n++)
00179    fscanf(fpSTATE, "%d %d %d %d %lf %lf\n", &BondID[n], &BondType[n], &atom1[n], &atom2[n], &kb[n],
      &ro[n]);
00180
00181   fclose(fpSTATE);
00182
00183  //2D-List of bonded atoms. This is used to remove pair interaction
00184  //calculation for the bonded atoms
00185    isBonded = (int**)malloc((nAtom + 1) * sizeof(int*));
00186   for (int i = 0; i <= nAtom; i++) {
00187     isBonded[i] = (int*)malloc((nAtom + 1) * sizeof(int));
00188     for (int j = 0; j <= nAtom; j++) {
00189         isBonded[i][j] = 0;
00190     }
00191   }
00192
00193   for (n = 1; n <= nBond; n++) {
00194     int i = atom1[n];
00195     int j = atom2[n];
00196     isBonded[i][j] = 1;
00197     isBonded[j][i] = 1; // symmetric
00198 }
00199
00200  // List the interface atoms
00201  nAtomInterface = 0;
00202  nAtomBlock = 0;
00203  nDiscInterface = 0;
00204  bigDiameter = 2.8;
00205  InterfaceWidth = 5.0 * bigDiameter;
00206
00207  for(n = 1; n <= nAtom; n++){
00208   if(fabs(ry[n]) < InterfaceWidth){
00209   nAtomInterface++;
00210   }
00211   if(molID[n] == 2){
00212   nAtomBlock++;
00213   }
00214   if(atomRadius[n] != 0.0){
00215   nDiscInterface++;
00216   } }
00217
00218
00219   int BondPairInteract;
00220   BondPairInteract = 0;
00221   int m;
00222   if(BondPairInteract == 1){
00223    atomIDInterface =  (int *)malloc((nAtomInterface+1)*sizeof(int));
00224    m = 1;
00225    for(n=1; n<=nAtom; n++){
00226     if(fabs(ry[n]) < InterfaceWidth){
00227     atomIDInterface[m] = atomID[n];
00228     m++;
00229   } } }
00230   else if(BondPairInteract == 0){
00231    nAtomInterface = nDiscInterface;
```

```
00232    atomIDInterface =  (int *)malloc((nAtomInterface+1)*sizeof(int));
00233    m = 1;
00234    for(n=1; n<=nAtom; n++){
00235     if(atomRadius[n] != 0.0){
00236     atomIDInterface[m] = atomID[n];
00237     m++;
00238   } } }
00239
00240   nPairTotal = 0.5 * nAtomInterface * (nAtomInterface-1);
00241   PairID = (int*)malloc((nPairTotal+1) * sizeof(int));
00242   Pairatom1 = (int*)malloc((nPairTotal+1) * sizeof(int));
00243   Pairatom2 = (int*)malloc((nPairTotal+1) * sizeof(int));
00244   PairXij = (double*)malloc((nPairTotal+1) * sizeof(double));
00245   PairYij = (double*)malloc((nPairTotal+1) * sizeof(double));
00246
00247   fprintf(fpresult, "-----------------------------------\n");
00248   fprintf(fpresult, "-------PARAMETERS-----------\n");
00249   fprintf(fpresult, "-----------------------------------\n");
00250   fprintf(fpresult, "nAtom\t\t\t%d\n",  nAtom);
00251   fprintf(fpresult, "nBond\t\t\t%d\n",  nBond);
00252   fprintf(fpresult, "nAtomType\t\t%d\n", nAtomType);
00253   fprintf(fpresult, "nBondType\t\t%d\n",  nBondType);
00254   fprintf(fpresult, "nAtomBlock\t\t%d\n",  nAtomBlock);
00255   fprintf(fpresult, "nAtomInterface\t\t%d\n",  nAtomInterface);
00256   fprintf(fpresult, "nDiscInterface\t\t%d\n",  nDiscInterface);
00257   fprintf(fpresult, "mass\t\t\t%0.6g\n", mass);
00258   fprintf(fpresult, "gamman\t\t\t%0.6g\n", gamman);
00259   fprintf(fpresult, "strain\t\t\t%0.6g\n", strain);
00260   fprintf(fpresult, "strainRate\t\t%0.6g\n", strainRate);
00261   fprintf(fpresult, "FyBylx\t\t\t%0.6g\n", FyBylx);
00262   fprintf(fpresult, "fxByfy\t\t\t%0.6g\n", fxByfy);
00263   fprintf(fpresult, "DeltaY\t\t\t%0.6g\n", DeltaY);
00264   fprintf(fpresult, "DeltaX\t\t\t%0.6g\n", DeltaX);
00265   fprintf(fpresult, "HaltCondition\t\t%0.6g\n", HaltCondition);
00266   fprintf(fpresult, "kappa\t\t\t%g\n", kappa);
00267   fprintf(fpresult, "density\t\t\t%g\n", density);
00268   fprintf(fpresult, "rCut\t\t\t%g\n", rCut);
00269   fprintf(fpresult, "deltaT\t\t\t%g\n", deltaT);
00270   fprintf(fpresult, "stepEquil\t\t%d\n",  stepEquil);
00271   fprintf(fpresult, "stepLimit\t\t%d\n",  stepLimit);
00272   fprintf(fpresult, "region[1]\t\t%0.16lf\n", region[1]);
00273   fprintf(fpresult, "region[2]\t\t%0.16lf\n", region[2]);
00274   fprintf(fpresult, "cells[1]\t\t%d\n",  cells[1]);
00275   fprintf(fpresult, "cells[2]\t\t%d\n",  cells[2]);
00276   fprintf(fpresult, "solver\t\t\t%s\n",  solver);
00277   fprintf(fpresult, "boundary\t\t%s %s\n", xBoundary, yBoundary);
00278   fprintf(fpresult, "DampFlag\t\t%d\n",  DampFlag);
00279   fprintf(fpresult, "-----------------------------------\n");
00280   fprintf(fpresult, "#TimeNow TotalMomentum PotEngyPerAtom KinEngyPerAtom TotEngyPerAtom
      PairEnergyPerAtom BondEnergyPerAtom  Press VirialSum\n");
00281   fprintf(fpvrms, "#timeNow\tVrms \n");
00282   fprintf(fpcom, "#timeNow\tComX\tComY\n");
00283   fprintf(fpforce,
      "#timeNow\tforceSumxAtomType1\tforceSumyAtomType1\tforceSumxAtomType2\tforceSumyAtomType2\tforceSumxAtomType3\tforceSumy
00284
00285 /* //Uncomment the following as per your acquirement
00286     fprintf(fpstress, "strain           %lf\n", strain);
00287     fprintf(fpstress, "region[1]        %lf\n", region[1]);
00288     fprintf(fpstress, "region[2]        %lf\n", region[2]);
00289     fprintf(fpstress, "#timeNow virSumxx virSumyy virSumxy pressure\n");
00290     fprintf(fpmomentum, "#timeNow Px Py\n");
00291 */
00292
00293    if((strcmp(xBoundary, "p") != 0 && strcmp(xBoundary, "r") != 0) ||
00294     (strcmp(yBoundary, "p") != 0 && strcmp(yBoundary, "r") != 0)) {
00295     fprintf(fpresult, "Error: Invalid boundary value detected: '%s %s'. Only 'p' or 'r' are
      allowed.\n", xBoundary, yBoundary);
00296     exit(EXIT_FAILURE); // Exit with failure status
00297   }
00298
00299 }
```

## 5.64 source/InitVacf.c File Reference

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

Include dependency graph for InitVacf.c:




**Functions**

- void ZeroVacf ()
- void InitVacf ()

## 5.64.1 Function Documentation

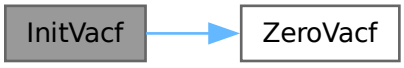
### 5.64.1.1 InitVacf()

```
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:


#### 5.64.1.2 ZeroVacf()

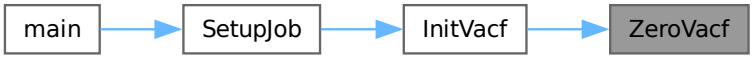
```
void ZeroVacf ()
```

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:




## 5.65 InitVacf.c

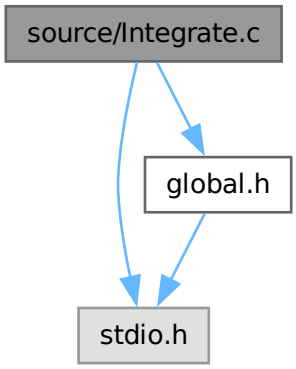
Go to the documentation of this file.

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

## 5.66 source/Integrate.c File Reference

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

Include dependency graph for Integrate.c:




### Functions

- double Integrate (double *f, int nf)

### 5.66.1 Function Documentation

#### 5.66.1.1 Integrate()

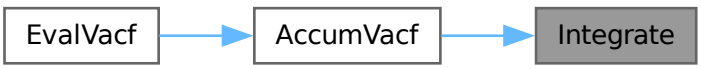
```
double Integrate (
            double * f,
            int nf)
```

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:



## 5.67 Integrate.c

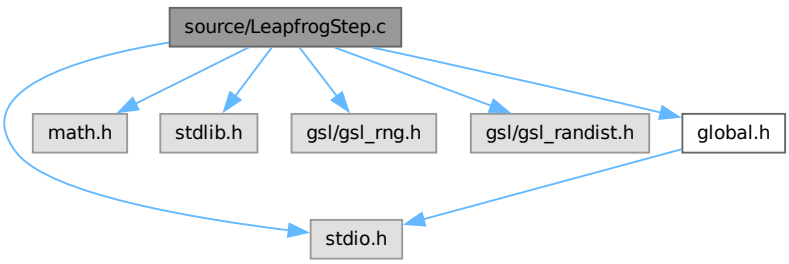
Go to the documentation of this file.

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 double Integrate(double *f, int nf){
00026  double s;
00027  int i;
00028  s = 0.5*(f[1] + f[nf]);
00029  for(i = 2 ; i <= nf - 1 ; i ++)
00030   s += f[i];
00031  return(s);
00032 }
00033
```

## 5.68 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)

## 5.68.1 Function Documentation

### 5.68.1.1 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
00064     int i,j;
00065     real dr[NDIM+1], r, rr, ri, rrCut;
00066     double vv;
00067
00068     double uVal, AA, AASum;
00069     double TVal;
00070
00071     double deno, VVSum;
00072     deno = 0.;
00073     VVSum = 0.;
00074     AASum = 0.;
00075
00076   for(n=1;n<=nAtom; n++)
00077      TValSum[n] = 0.;
00078
00079   rrCut = Sqr(rCut);
00080
00081 /*****Calculating Configarational temperature*****/
00082 //Solving the equation of motion here
00083 if(thermo == 'C'){
00084 for(i = 1 ; i <= nAtom; i ++){
00085     for(j = i+1 ; j <= nAtom ; j ++){
00086       dr[1] = rx[i] - rx[j];
00087       if(fabs(dr[1]) > regionH[1])
00088     dr[1] -= SignR(region[1], dr[1]);
00089
00090      dr[2] = ry[i] - ry[j];
00091       if(fabs(dr[2]) > regionH[2])
00092     dr[2] -= SignR(region[2], dr[2]);
00093
00094      rr = Sqr(dr[1]) + Sqr(dr[2]);
00095      if(rr < rrCut ){
00096      r = sqrt(rr);
00097      ri = 1/r;
00098      uVal = ri*exp(-kappa*r);
00099
00100      TVal = (1./rr + Sqr(kappa) + kappa/r)*uVal;
00101      TValSum[i] += TVal;
00102      TValSum[j] += TVal;
00103    } }
```

```
00104       AA = Sqr(ax[i]) + Sqr(ay[i]);
00105       AASum += AA;
00106       vv = Sqr(vx[i]) + Sqr(vy[i]);
00107       VVSum += vv;
00108       deno += TValSum[i];
00109 }
00110
00111       double gSumconfig, varSconfig, massSconfig;
00112       if(stepCount == 1) varSconfig = 0.;
00113       gSumconfig = 0.; massSconfig = 2.0;
00114
00115       gSumconfig = (AASum/temperature - deno)/massSconfig;
00116       varSconfig += deltaT*gSumconfig;
00117
00118        /*****Configarational Nose-Hoover thermostat*****/
00119     for (n = 1; n <= nAtom; n++){
00120        vx[n] += deltaT * ax[n];
00121        rx[n] += deltaT * (vx[n] + varSconfig * ax[n]);
00122        vy[n] += deltaT * ay[n];
00123        ry[n] += deltaT * (vy[n] + varSconfig * ay[n]);
00124      }
00125        /*****Kinetic Nose-Hoover thermostat*****/
00126    }else if(thermo == 'N'){
00127      gSum = (0.5*S2 - (nAtom + 1)*temperature)/massS;
00128      varS += deltaT*gSum;
00129     for (n = 1; n <= nAtom; n++){
00130        vx[n] += deltaT * (ax[n] - varS *vx[n]);
00131        rx[n] += deltaT * vx[n];
00132        vy[n] += deltaT * (ay[n] - varS *vy[n]);
00133        ry[n] += deltaT * vy[n];
00134     }
00135        /*****for Gaussian thermostat*****/
00136   }else if(thermo == 'G'){
00137        for (n = 1; n <= nAtom; n++){
00138        vx[n] = C * vx[n] + D * ax[n];
00139        rx[n] += deltaT * vx[n];
00140        vy[n] = C * vy[n] + D * ay[n];
00141        ry[n] += deltaT * vy[n];
00142       }
00143     }else if (thermo == 'L'){
00144    double nu = 0.03066;
00145    double var = sqrt(2*nu/(GAMMA*deltaT));
00146    double scale = 1. + nu*deltaT/2.;
00147    double scale_v = 2./scale - 1.;
00148    double scale_f = deltaT/scale;
00149    for(n = 1 ; n <= nAtom ; n ++){
00150        vx[n] = scale_v*vx[n] + scale_f*(ax[n] + var*gsl_ran_gaussian(rnd,1));
00151        rx[n] += deltaT * vx[n];
00152        vy[n] = scale_v*vy[n] + scale_f*(ay[n] + var*gsl_ran_gaussian(rnd,1));
00153        ry[n] += deltaT * vy[n];
00154      }
00155    }
00156   }else{
00157      int n;
00158      for(n = 1 ; n <= nAtom ; n ++){
00159        vx[n] += deltaT * ax[n];
00160        rx[n] += deltaT * vx[n];
00161        vy[n] += deltaT * ay[n];
00162        ry[n] += deltaT * vy[n];
00163      }
00164    }
00165 }
```

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

## 5.69 LeapfrogStep.c

Go to the documentation of this file.

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

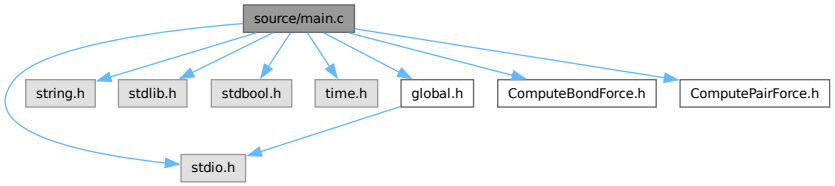
```
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020 #include<stdio.h>
00021 #include<math.h>
00022 #include<stdlib.h>
00023 #include <gsl/gsl_rng.h>
00024 #include <gsl/gsl_randist.h>
00025 #include"global.h"
00026
00027
00028 void LeapfrogStep(char thermo, gsl_rng * rnd){
00029 double temperature, GAMMA;
00030 GAMMA = 100;
00031
00032 double *TValSum;
00033 TValSum = (double*)malloc((nAtom + 1) * sizeof(double));
00034
00035
00036   if(stepCount <= stepEquil){
00037     double gSum, varS, massS;
00038     temperature = 1./GAMMA;
00039
00040    if(stepCount == 1) varS = 0.;
00041     double A, S1, S2, T;
00042     int n;
00043     S1 = 0.; S2 = 0.; gSum = 0.; massS = 0.1;
00044
00045     vvSum = 0.;
00046     double halfdt = 0.5*deltaT;
00047     for (n = 1; n <= nAtom; n++){
00048       T = vx[n] + halfdt * ax[n];
00049       S1 += T * ax[n];
00050       S2 += Sqr(T);
00051
00052       T = vy[n] + halfdt * ay[n];
00053       S1 += T * ay[n];
00054       S2 += Sqr(T);
00055      vvSum += (Sqr(vx[n]) +  Sqr(vy[n]));
00056     }
00057
00058     A = -S1 / S2;
00059     S2 = vvSum;
00060
00061     double C = 1 + A*deltaT ;
00062     double D = deltaT * (1 + 0.5 * A * deltaT);
00063
00064     int i,j;
00065     real dr[NDIM+1], r, rr, ri, rrCut;
00066     double vv;
00067
00068     double uVal, AA, AASum;
00069     double TVal;
00070
00071     double deno, VVSum;
00072     deno = 0.;
00073     VVSum = 0.;
00074     AASum = 0.;
00075
00076   for(n=1;n<=nAtom; n++)
00077      TValSum[n] = 0.;
00078
00079   rrCut = Sqr(rCut);
00080
00081 /*****Calculating Configarational temperature*****/
00082 //Solving the equation of motion here
00083 if(thermo == 'C'){
00084 for(i = 1 ; i <= nAtom; i ++){
00085     for(j = i+1 ; j <= nAtom ; j ++){
00086       dr[1] = rx[i] - rx[j];
00087       if(fabs(dr[1]) > regionH[1])
00088     dr[1] -= SignR(region[1], dr[1]);
00089
00090      dr[2] = ry[i] - ry[j];
00091       if(fabs(dr[2]) > regionH[2])
00092     dr[2] -= SignR(region[2], dr[2]);
00093
00094      rr = Sqr(dr[1]) + Sqr(dr[2]);
00095      if(rr < rrCut ){
00096      r = sqrt(rr);
00097      ri = 1/r;
00098      uVal = ri*exp(-kappa*r);
00099
00100      TVal = (1./rr + Sqr(kappa) + kappa/r)*uVal;
00101      TValSum[i] += TVal;
00102      TValSum[j] += TVal;
```

```
00103    } }
00104      AA = Sqr(ax[i]) + Sqr(ay[i]);
00105      AASum += AA;
00106      vv = Sqr(vx[i]) + Sqr(vy[i]);
00107      VVSum += vv;
00108      deno += TValSum[i];
00109 }
00110
00111      double gSumconfig, varSconfig, massSconfig;
00112      if(stepCount == 1) varSconfig = 0.;
00113      gSumconfig = 0.; massSconfig = 2.0;
00114
00115      gSumconfig = (AASum/temperature - deno)/massSconfig;
00116      varSconfig += deltaT*gSumconfig;
00117
00118       /*****Configarational Nose-Hoover thermostat*****/
00119    for (n = 1; n <= nAtom; n++){
00120       vx[n] += deltaT * ax[n];
00121       rx[n] += deltaT * (vx[n] + varSconfig * ax[n]);
00122       vy[n] += deltaT * ay[n];
00123       ry[n] += deltaT * (vy[n] + varSconfig * ay[n]);
00124     }
00125       /*****Kinetic Nose-Hoover thermostat*****/
00126   }else if(thermo == 'N'){
00127     gSum = (0.5*S2 - (nAtom + 1)*temperature)/massS;
00128     varS += deltaT*gSum;
00129    for (n = 1; n <= nAtom; n++){
00130       vx[n] += deltaT * (ax[n] - varS *vx[n]);
00131       rx[n] += deltaT * vx[n];
00132       vy[n] += deltaT * (ay[n] - varS *vy[n]);
00133       ry[n] += deltaT * vy[n];
00134    }
00135       /*****for Gaussian thermostat*****/
00136  }else if(thermo == 'G'){
00137       for (n = 1; n <= nAtom; n++){
00138       vx[n] = C * vx[n] + D * ax[n];
00139       rx[n] += deltaT * vx[n];
00140       vy[n] = C * vy[n] + D * ay[n];
00141       ry[n] += deltaT * vy[n];
00142      }
00143    }else if (thermo == 'L'){
00144   double nu = 0.03066;
00145   double var = sqrt(2*nu/(GAMMA*deltaT));
00146   double scale = 1. + nu*deltaT/2.;
00147   double scale_v = 2./scale - 1.;
00148   double scale_f = deltaT/scale;
00149   for(n = 1 ; n <= nAtom ; n ++){
00150       vx[n] = scale_v*vx[n] + scale_f*(ax[n] + var*gsl_ran_gaussian(rnd,1));
00151       rx[n] += deltaT * vx[n];
00152       vy[n] = scale_v*vy[n] + scale_f*(ay[n] + var*gsl_ran_gaussian(rnd,1));
00153       ry[n] += deltaT * vy[n];
00154     }
00155   }
00156  }else{
00157     int n;
00158     for(n = 1 ; n <= nAtom ; n ++){
00159       vx[n] += deltaT * ax[n];
00160       rx[n] += deltaT * vx[n];
00161       vy[n] += deltaT * ay[n];
00162       ry[n] += deltaT * vy[n];
00163     }
00164   }
00165 }
00166
```

## 5.70 source/main.c File Reference

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.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 ApplyBoundaryCond ()
- void EvalProps ()
- void AccumProps (int icode)
- void PrintSummary ()
- void PrintVrms ()
- void VelocityVerletStep (int icode)
- void ApplyForce ()
- void ApplyLeesEdwardsBoundaryCond ()
- void PrintStress ()
- void Close ()
- void PrintMomentum ()
- void DisplaceAtoms ()
- void DumpRestart ()
- bool HaltConditionCheck (double value)
- void EvalCom ()
- void PrintCom ()
- void EvalVrms ()
- void EvalUnwrap ()
- void DumpBonds ()
- void DumpPairs ()
- void WriteBinaryRestart ()
- void PrintForceSum ()
- int main (int argc, char **argv)

### Variables

- char * prefix = NULL

## 5.70.1 Macro Definition Documentation

### 5.70.1.1 DEFINE_GLOBALS

```
#define DEFINE_GLOBALS
```

Definition at line 7 of file main.c.

## 5.70.2 Function Documentation

### 5.70.2.1 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  } }
```

### 5.70.2.2 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]);
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:


#### 5.70.2.3 ApplyForce()

```
void ApplyForce ()
```

Definition at line 25 of file ApplyForce.c.

```
00025                 {
00026  int n;
00027  double lx;
00028  lx = regionH[1];
00029  fyExtern =  (FyBylx * lx)/nAtomBlock;
00030  fxExtern =  fxByfy * fyExtern;
00031  forceSumxExtern = fxExtern*nAtomBlock;  forceSumyExtern = fyExtern*nAtomBlock;
00032
00033  for(n = 1; n <= nAtom; n ++){
00034  if(molID[n] == 2){
00035   fx[n] += fxExtern;
00036   fy[n] -= fyExtern;
00037 } } }
```

References forceSumxExtern, forceSumyExtern, fx, fxByfy, fxExtern, fy, FyBylx, fyExtern, molID, nAtom, nAtomBlock, and regionH.
Referenced by main().
Here is the caller graph for this function:


#### 5.70.2.4 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.

#### 5.70.2.5 Close()

```
void Close ()
```

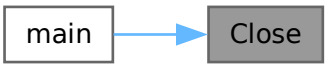
Definition at line 25 of file Close.c.

```
00025            {
00026   int n;
00027   free(rx); free(ry); free(vx); free(vy); free(ax); free(ay); free(fx); free(fy);
00028   free(fax);
00029   free(fay);
00030   free(cellList);
00031
00032   free(atomID); free(atomType); free(atomRadius); free(atomMass);
00033   free(speed);
00034   free(atom1); free(atom2); free(BondID);
00035   free(BondType); free(kb); free(ro);
00036   free(ImageX); free(ImageY); free(rxUnwrap); free(ryUnwrap);
00037   free(atomIDInterface);
00038   free(PairID); free(Pairatom1); free(Pairatom2);
00039   free(PairXij); free(PairYij);
00040   free(molID);
00041
00042   for (n = 0; n <= nAtom; n++) {
00043    free(isBonded[n]);
00044    }
00045   free(isBonded);
00046
00047   for (n = 0; n <= nBuffCorr; n++){
00048   free(cfOrg[n]);
00049   free(spacetimeCorr[n]);
00050   }
00051   free(cfOrg);
00052   free(spacetimeCorr);
00053   free(cfVal);
00054   free(indexCorr);
00055   free(spacetimeCorrAv);
00056
00057   free(indexAcf);
00058   free(viscAcfOrg);
00059   free(viscAcfAv);
00060   for(n = 0 ; n <= nBuffAcf ; n ++)
00061    free(viscAcf[n]);
00062   free(viscAcf);
00063
00064  }
```

References atom1, atom2, atomID, atomIDInterface, atomMass, atomRadius, atomType, ax, ay, BondID, BondType, cellList, cfOrg, cfVal, fax, fay, fx, fy, 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:



#### 5.70.2.6 ComputeForcesCells()

```
void ComputeForcesCells ()
```

Definition at line 25 of file ComputeForcesCells.c.

```
00025                           {
00026   double dr[NDIM+1], invWid[NDIM+1], shift[NDIM+1], f, fcVal, rr, ri, r, uVal;
00027   int c, I, J, m1, m1X, m1Y, m2, m2X, m2Y, n, offset;
00028   int iofX[] = {0, 0, 1, 1, 0, -1, -1, -1, 0, 1},
00029       iofY[] = {0, 0, 0, 1 ,1, 1, 0, -1, -1, -1};
00030
00031   invWid[1] = cells[1]/region[1];
00032   invWid[2] = cells[2]/region[2];
00033
00034   for(n = nAtom+1; n <= nAtom+cells[1]*cells[2] ; n++)
00035     cellList[n] = 0;
00036
00037   for(n = 1 ; n <= nAtom ; n ++){
00038     c = ((int)((ry[n] + regionH[2])*invWid[2]))*cells[1] + (int)((rx[n]+regionH[1])*invWid[1]) +
      nAtom+ 1;
00039     cellList[n] = cellList[c];
00040     cellList[c] = n;
00041   }
00042
00043   for(n = 1 ; n <= nAtom ; n ++){
00044     ax[n] = 0.;
00045     ay[n] = 0.;
00046   }
00047
00048   uSum = 0.0 ;
00049   virSum = 0.0;
00050   rfAtom = 0.0;
00051   RadiusIJ = 0.0;
00052
00053   gamman = 1.0;
00054   double vr[NDIM+1], fd, fdVal, rrinv;
00055   rrinv = 0.0;
00056   fd = 0.0;
00057   fdVal = 0.0;
00058
00059   int start = 1 + rank*(cells[2]/size);
00060   int end = (rank+1)*(cells[2]/size);
00061
00062   for(m1Y = start ; m1Y <= end ; m1Y ++){
00063     for(m1X = 1 ; m1X <= cells[1] ; m1X ++){
00064       m1 = (m1Y-1) * cells[1] + m1X + nAtom;
00065       for(offset = 1 ; offset <= 9 ; offset ++){
00066     m2X = m1X + iofX[offset]; shift[1] = 0.;
00067     if(m2X > cells[1]){
00068       m2X = 1; shift[1] = region[1];
00069     }else if(m2X == 0){
00070       m2X = cells[1]; shift[1] = -region[1];
00071     }
00072     m2Y = m1Y + iofY[offset]; shift[2] = 0.;
00073     if(m2Y > cells[2]){
00074       m2Y = 1; shift[2] = region[2];
00075     }else if(m2Y == 0){
00076       m2Y = cells[2]; shift[2] = -region[2];
00077     }
00078     m2 = (m2Y-1)*cells[1] + m2X + nAtom;
00079     I = cellList[m1];
00080     while(I > 0){
00081       J = cellList[m2];
00082       while(J > 0){
00083         if(m1 == m2 && J != I && (atomRadius[I] > 0. && atomRadius[J] > 0.)){
00084           dr[1] = rx[I] - rx[J] - shift[1];
00085           dr[2] = ry[I] - ry[J] - shift[2];
00086           rr = Sqr(dr[1]) + Sqr(dr[2]);
00087           RadiusIJ = atomRadius[I] + atomRadius[J];
00088           SqrRadiusIJ = Sqr(RadiusIJ);
00089           if(rr < SqrRadiusIJ){
00090         r = sqrt(rr);
00091         ri = 1.0/r;
00092                 rrinv = 1.0/rr;
00093                 vr[1] = vx[I] - vx[J];
00094                 vr[2] = vy[I] - vy[J];
00095         RadiusIJInv = 1.0/RadiusIJ;
00096         uVal = Sqr(1.0 - r * RadiusIJInv);
00097         fcVal = 2.0 * RadiusIJInv * (1.0 - r * RadiusIJInv) *ri;
00098                 fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag
00099
00100         f  = fcVal * dr[1];
00101                 fd = fdVal * dr[1];
00102         ax[I] += (f + fd);
00103                 discDragx[I] += fd; //disc-disc drag
00104
00105         f = fcVal * dr[2];
```

```
00106                 fd = fdVal * dr[2];
00107         ay[I] += (f + fd);
00108                 discDragy[I] += fd; //disc-disc drag
00109
00110         uSum +=  0.5 * uVal;
00111         virSum += 0.5 * fcVal * rr;
00112         rfAtom += 0.5 * dr[1] * fcVal * dr[2];
00113           }
00114         }else if(m1 != m2 && (atomRadius[I] > 0. && atomRadius[J] > 0.)){
00115           dr[1] = rx[I] - rx[J] - shift[1];
00116           dr[2] = ry[I] - ry[J] - shift[2];
00117           rr = Sqr(dr[1]) + Sqr(dr[2]);
00118           RadiusIJ = atomRadius[I] + atomRadius[J];
00119           SqrRadiusIJ = Sqr(RadiusIJ);
00120           if(rr < SqrRadiusIJ){
00121         r = sqrt(rr);
00122         ri = 1.0/r;
00123                 rrinv = 1.0/r;
00124                 vr[1] = vx[I] - vx[J];
00125                 vr[2] = vy[I] - vy[J];
00126         RadiusIJInv = 1.0/RadiusIJ;
00127         uVal = Sqr(1.0 - r * RadiusIJInv);
00128         fcVal = 2.0 * RadiusIJInv * (1.0 - r * RadiusIJInv) *ri;
00129                 fdVal = -gamman * (vr[1]*dr[1] + vr[2]*dr[2]) * rrinv; //disc-disc drag
00130
00131         f  = fcVal * dr[1];
00132                 fd =  fdVal * dr[1];
00133         ax[I] += (f + fd);
00134                 discDragx[I] += fd; //disc-disc drag
00135
00136         f  = fcVal * dr[2];
00137                 fd = fdVal * dr[2];
00138         ay[I] += (f + fd);
00139                 discDragy[I] += fd; //disc-disc drag
00140
00141         uSum +=  0.5 * uVal;
00142         virSum += 0.5 * fcVal * rr;
00143         rfAtom += 0.5 * dr[1] * fcVal * dr[2];
00144           }
00145         }
00146             J = cellList[J];
00147       }
00148       I = cellList[I];
00149     }
00150       }
00151     }
00152   }
00153 }
```

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

#### 5.70.2.7 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:

#### 5.70.2.8 DumpBonds()

```
void DumpBonds ()
```

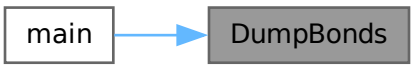
Definition at line 24 of file DumpBonds.c.

```
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:




#### 5.70.2.9 DumpPairs()

```
void DumpPairs ()
```

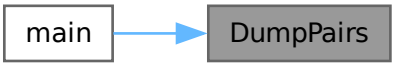
Definition at line 25 of file DumpPairs.c.

```
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:




#### 5.70.2.10 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] %0.16lf\n", region[1]);
00041   fprintf(fpDUMP, "region[2] %0.16lf\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:

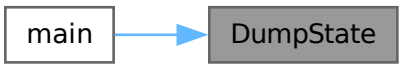

#### 5.70.2.11 DumpState()

```
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:




#### 5.70.2.12 EvalCom()

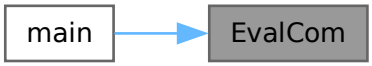
```
void EvalCom ()
```

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:




#### 5.70.2.13 EvalProps()

```
void EvalProps ()
```

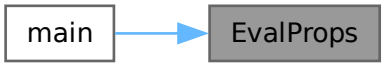
Definition at line 26 of file EvalProps.c.

```
00026                 {
00027  double v;
00028  int n;
00029  double atomMassn;
00030  double KineEnrXSum, KineEnrYSum;
00031  virSum = 0.0;
00032  vSumX = 0.0; vSumY = 0.0; vSum = 0.0; vvSum = 0.0;
00033  KineEnrXSum = 0.0; KineEnrYSum = 0.0;
00034
00035  for (n = 1; n <= nAtom; n++) {
00036   // Initialize v with a default value to avoid "uninitialized" warning.
00037   v = 0.0;
00038   atomMassn = atomMass[n];
00039   // X direction velocity
00040   if (strcmp(solver, "Verlet") == 0) {
00041     v = vx[n];
00042   } else if (strcmp(solver, "LeapFrog") == 0) {
00043     v = vx[n] - 0.5 * deltaT * ax[n];
00044   }
00045    vSum += v;
00046    vvSum += Sqr(v);
00047    KineEnrXSum += 0.5 * atomMassn * Sqr(v);
00048    vSumX += v;
00049    // Y direction velocity
00050    if (strcmp(solver, "Verlet") == 0) {
00051    v = vy[n];
00052    } else if (strcmp(solver, "LeapFrog") == 0) {
00053    v = vy[n] - 0.5 * deltaT * ay[n];
00054    }
00055    vSum += v;
00056    vSumY += v;
00057    vvSum += Sqr(v);
00058    KineEnrYSum += 0.5 * atomMassn * Sqr(v);
00059   }
00060
00061   kinEnergy = (KineEnrXSum + KineEnrYSum) / nAtom ;
00062   uSumPairPerAtom = uSumPair / nAtom ;
00063   BondEnergyPerAtom = TotalBondEnergy / (0.5*nAtom); //Factor of 0.5 since each atom has one half the
      bond energy
00064   potEnergy = uSumPairPerAtom +  BondEnergyPerAtom ;
00065   totEnergy = kinEnergy + potEnergy;
00066   virSumxx = virSumPairxx  + virSumBondxx ;
00067   virSumyy = virSumPairyy  + virSumBondyy ;
00068   virSumxy = virSumPairxy  + virSumBondxy ;
00069   virSum = virSumPair + virSumBond;
00070   pressure = density * (vvSum + virSum) / (nAtom * NDIM);
00071
00072 }
```

References atomMass, 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:




#### 5.70.2.14 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);
00106       real tVal;
00107       for (n = 1; n <= nValCorr; n++){
00108     tVal = (n-1)*stepCorr*deltaT;
00109     fprintf (fpdnsty, "%e\t", tVal);
00110     int nn = nFunCorr*(n-1);
00111     for (j = 1; j <= nFunCorr; j ++)
00112           fprintf (fpdnsty, "%e\t", spacetimeCorrAv[nn + j]);
00113         fprintf (fpdnsty, "\n");
00114       }
00115
00116       countCorrAv = 0.;
00117       for (j = 1; j <= nFunCorr*nValCorr; j++)
00118     spacetimeCorrAv[j] = 0.;
00119     }
00120    }
00121   }
00122 }
```

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

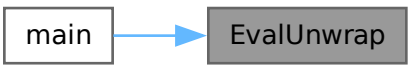
#### 5.70.2.15 EvalUnwrap()

```
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:




#### 5.70.2.16 EvalVrms()

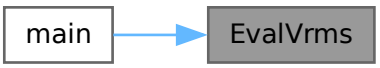
```
void EvalVrms ()
```

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:




#### 5.70.2.17 HaltConditionCheck()

```
bool HaltConditionCheck (
            double value)
```

Definition at line 27 of file Halt.c.

```
00027                                   {
00028
00029   if(value <= HaltCondition && value != 0) {
00030   printf("Halt condition met at step = %d with Vrms = %.16f\n", stepCount, value);
00031   fprintf(fpresult, "Halt condition met at step = %d with Vrms = %.16f\n", stepCount, value);
00032   fprintf(fpresult, "Final thermodynamic values:\n");
00033   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",
00034   timeNow, vSum, potEnergy, kinEnergy, totEnergy, uSumPairPerAtom, BondEnergyPerAtom, pressure,
      virSum);
00035   return true;       // Signal that the halt condition is met
00036   }
00037  return false; // Halt condition not met
00038 }
```

References BondEnergyPerAtom, fpresult, HaltCondition, kinEnergy, potEnergy, pressure, stepCount, timeNow, totEnergy, uSumPairPerAtom, virSum, and vSum.
Referenced by main().
Here is the caller graph for this function:


#### 5.70.2.18 Init()

```
void Init ()
```

Definition at line 31 of file Init.c.

```
00031           {
00032  char dummy[128];
00033
```

```
00034  // Always read input parameters
00035  FILE *fp = fopen("input-data", "r");
00036  if(!fp) {
00037   perror("input-data");
00038   exit(EXIT_FAILURE);
00039  }
00040
00041   fscanf(fp, "%s %s", mode, inputConfig);  // config type + filename
00042   fscanf(fp, "%s %s", dummy, solver);
00043   fscanf(fp, "%s %s %s", dummy, xBoundary, yBoundary);
00044   fscanf(fp, "%s %d",  dummy, &DampFlag);
00045   fscanf(fp, "%s %d",  dummy, &freezeAtomType);
00046   fscanf(fp, "%s %lf", dummy, &rCut);
00047   fscanf(fp, "%s %lf", dummy, &Kn);
00048   fscanf(fp, "%s %lf", dummy, &mass);
00049   fscanf(fp, "%s %lf", dummy, &gamman);
00050   fscanf(fp, "%s %lf", dummy, &kappa);
00051   fscanf(fp, "%s %lf", dummy, &deltaT);
00052   fscanf(fp, "%s %lf", dummy, &strain);
00053   fscanf(fp, "%s %lf", dummy, &FyBylx);
00054   fscanf(fp, "%s %lf", dummy, &fxByfy);
00055   fscanf(fp, "%s %lf", dummy, &DeltaY);
00056   fscanf(fp, "%s %lf", dummy, &DeltaX);
00057   fscanf(fp, "%s %lf", dummy, &HaltCondition);
00058   fscanf(fp, "%s %d",  dummy, &stepAvg);
00059   fscanf(fp, "%s %d",  dummy, &stepEquil);
00060   fscanf(fp, "%s %d",  dummy, &stepLimit);
00061   fscanf(fp, "%s %d",  dummy, &stepDump);
00062   fscanf(fp, "%s %d",  dummy, &stepTraj);
00063   fscanf(fp, "%s %d",  dummy, &limitCorrAv);
00064   fscanf(fp, "%s %d",  dummy, &nBuffCorr);
00065   fscanf(fp, "%s %d",  dummy, &nFunCorr);
00066   fscanf(fp, "%s %d",  dummy, &nValCorr);
00067   fscanf(fp, "%s %d",  dummy, &stepCorr);
00068   fscanf(fp, "%s %d",  dummy, &limitAcfAv);
00069   fscanf(fp, "%s %d",  dummy, &nBuffAcf);
00070   fscanf(fp, "%s %d",  dummy, &nValAcf);
00071   fscanf(fp, "%s %d",  dummy, &stepAcf);
00072   fscanf(fp, "%s %lf", dummy, &rangeRdf);
00073   fscanf(fp, "%s %d",  dummy, &limitRdf);
00074   fscanf(fp, "%s %d",  dummy, &sizeHistRdf);
00075   fscanf(fp, "%s %d",  dummy, &stepRdf);
00076   fclose(fp);
00077
00078   int useBinaryRestart = 0;
00079   if(strcmp(mode, "read_restart") == 0) {
00080    useBinaryRestart = 1;
00081   } else if (strcmp(mode, "read_data") != 0) {
00082    fprintf(stderr, "ERROR: First line of input-data must be 'read_data' or 'read_restart'\n");
00083    exit(0);
00084   }
00085
00086   //Conditionally read binary config
00087   if(useBinaryRestart) {
00088    ReadBinaryRestart(inputConfig);  // uses global prefix + config file
00089    printf(">> Binary restart loaded from %s <<\n", inputConfig);
00090    printf(">> Restarting simulation from time = %.8lf <<\n", timeNow);
00091    return; //Exiting from Init() from here
00092   }
00093
00094   FILE *fpSTATE;
00095   if((fpSTATE = fopen(inputConfig,"r"))==NULL){
00096   printf("Error occurred: Could not open STATE file\n Exiting now..\n");
00097   exit(0);
00098   }
00099
00100   if(fscanf(fpSTATE, "%s %lf", dummy, &timeNow) != 2 || strcmp(dummy, "timeNow") != 0) {
00101    fprintf(stderr, "ERROR [%s:%d:%s]: Expected 'timeNow <value>' as the first line in the config
     file.\n",
00102    __FILE__, __LINE__, __func__);
00103    exit(EXIT_FAILURE);
00104   }
00105
00106   if(timeNow == 0.0) {
00107    printf(">> Running from time = 0.0: Beginning of the simulation\n");
00108    stepCount = 0;
00109   }
00110
00111   fscanf(fpSTATE, "%s %d",  dummy, &nAtom);
00112   fscanf(fpSTATE, "%s %d",  dummy, &nBond);
00113   fscanf(fpSTATE, "%s %d",  dummy, &nAtomType);
00114   fscanf(fpSTATE, "%s %d",  dummy, &nBondType);
00115   fscanf(fpSTATE, "%s %lf", dummy, &region[1]);
00116   fscanf(fpSTATE, "%s %lf", dummy, &region[2]);
00117
00118   if(timeNow == 0.0)  region[2] *= 1.5; //Remove this when put on GitHub
00119
```

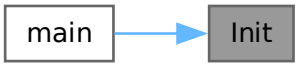
```
00120   density = nAtom/(region[1]*region[2]);
00121   cells[1] = region[1] / rCut;
00122   cells[2] = region[2] / rCut;
00123   cellList = (int*)malloc((nAtom + cells[1] * cells[2] + 1) * sizeof(int));
00124   regionH[1] = 0.5*region[1];
00125   regionH[2] = 0.5*region[2];
00126
00127   //strain information
00128   strainRate = strain/deltaT;
00129   shearDisplacement = strain * region[2];
00130   shearVelocity = strainRate * region[2];
00131   int n;
00132
00133   rx = (double*)malloc((nAtom + 1) * sizeof(double));
00134   ry = (double*)malloc((nAtom + 1) * sizeof(double));
00135   vx = (double*)malloc((nAtom + 1) * sizeof(double));
00136   vy = (double*)malloc((nAtom + 1) * sizeof(double));
00137   ax = (double*)malloc((nAtom + 1) * sizeof(double));
00138   ay = (double*)malloc((nAtom + 1) * sizeof(double));
00139   fx = (double*)malloc((nAtom + 1) * sizeof(double));
00140   fy = (double*)malloc((nAtom + 1) * sizeof(double));
00141   fax = (double*)malloc((nAtom + 1) * sizeof(double));
00142   fay = (double*)malloc((nAtom + 1) * sizeof(double));
00143   atomID = (int*)malloc((nAtom+1) * sizeof(int));
00144   atomType = (int*)malloc((nAtom+1) * sizeof(int));
00145   atomRadius = (double*)malloc((nAtom + 1) * sizeof(double));
00146   atomMass = (double*)malloc((nAtom + 1) * sizeof(double));
00147   speed = (double*)malloc((nAtom + 1) * sizeof(double));
00148   discDragx = (double*)malloc((nAtom + 1) * sizeof(double));
00149   discDragy = (double*)malloc((nAtom + 1) * sizeof(double));
00150   molID = (int*)malloc((nAtom+1) * sizeof(int));
00151
00152   BondID = (int*)malloc((nBond+1)*sizeof(int));
00153   BondType = (int*)malloc((nBond+1)*sizeof(int));
00154   atom1 = (int*)malloc((nBond+1)*sizeof(int));
00155   atom2 = (int*)malloc((nBond+1)*sizeof(int));
00156   kb = (double*)malloc((nBond+1)*sizeof(double));
00157   ro = (double*)malloc((nBond+1)*sizeof(double));
00158   BondEnergy = (double*)malloc((nBond+1)*sizeof(double));
00159   BondLength =(double*)malloc((nBond+1)*sizeof(double));
00160   nodeDragx = (double*)malloc((nAtom + 1) * sizeof(double));
00161   nodeDragy = (double*)malloc((nAtom + 1) * sizeof(double));
00162   rxUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
00163   ryUnwrap = (double*)malloc((nAtom + 1) * sizeof(double));
00164   ImageX = (int*)malloc((nAtom+1) * sizeof(int));
00165   ImageY = (int*)malloc((nAtom+1) * sizeof(int));
00166
00167
00168   for(n = 1; n <= nAtom; n ++){
00169    atomMass[n] = mass;
00170   }
00171
00172   fscanf(fpSTATE, "%s\n", dummy);
00173   for(n = 1; n <= nAtom; n ++)
00174    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]);
00175
00176
00177   fscanf(fpSTATE, "%s\n", dummy);
00178   for(n=1; n<=nBond; n++)
00179    fscanf(fpSTATE, "%d %d %d %d %lf %lf\n", &BondID[n], &BondType[n], &atom1[n], &atom2[n], &kb[n],
      &ro[n]);
00180
00181   fclose(fpSTATE);
00182
00183  //2D-List of bonded atoms. This is used to remove pair interaction
00184  //calculation for the bonded atoms
00185    isBonded = (int**)malloc((nAtom + 1) * sizeof(int*));
00186   for (int i = 0; i <= nAtom; i++) {
00187     isBonded[i] = (int*)malloc((nAtom + 1) * sizeof(int));
00188     for (int j = 0; j <= nAtom; j++) {
00189         isBonded[i][j] = 0;
00190     }
00191   }
00192
00193   for (n = 1; n <= nBond; n++) {
00194     int i = atom1[n];
00195     int j = atom2[n];
00196     isBonded[i][j] = 1;
00197     isBonded[j][i] = 1; // symmetric
00198 }
00199
00200  // List the interface atoms
00201  nAtomInterface = 0;
00202  nAtomBlock = 0;
00203  nDiscInterface = 0;
00204  bigDiameter = 2.8;
```

```
00205  InterfaceWidth = 5.0 * bigDiameter;
00206
00207  for(n = 1; n <= nAtom; n++){
00208   if(fabs(ry[n]) < InterfaceWidth){
00209   nAtomInterface++;
00210   }
00211   if(molID[n] == 2){
00212   nAtomBlock++;
00213   }
00214   if(atomRadius[n] != 0.0){
00215   nDiscInterface++;
00216   } }
00217
00218
00219   int BondPairInteract;
00220   BondPairInteract = 0;
00221   int m;
00222   if(BondPairInteract == 1){
00223    atomIDInterface =  (int *)malloc((nAtomInterface+1)*sizeof(int));
00224    m = 1;
00225    for(n=1; n<=nAtom; n++){
00226     if(fabs(ry[n]) < InterfaceWidth){
00227     atomIDInterface[m] = atomID[n];
00228     m++;
00229   } } }
00230   else if(BondPairInteract == 0){
00231    nAtomInterface = nDiscInterface;
00232    atomIDInterface =  (int *)malloc((nAtomInterface+1)*sizeof(int));
00233    m = 1;
00234    for(n=1; n<=nAtom; n++){
00235     if(atomRadius[n] != 0.0){
00236     atomIDInterface[m] = atomID[n];
00237     m++;
00238   } } }
00239
00240   nPairTotal = 0.5 * nAtomInterface * (nAtomInterface-1);
00241   PairID = (int*)malloc((nPairTotal+1) * sizeof(int));
00242   Pairatom1 = (int*)malloc((nPairTotal+1) * sizeof(int));
00243   Pairatom2 = (int*)malloc((nPairTotal+1) * sizeof(int));
00244   PairXij = (double*)malloc((nPairTotal+1) * sizeof(double));
00245   PairYij = (double*)malloc((nPairTotal+1) * sizeof(double));
00246
00247   fprintf(fpresult, "-----------------------------------\n");
00248   fprintf(fpresult, "-------PARAMETERS-----------\n");
00249   fprintf(fpresult, "-----------------------------------\n");
00250   fprintf(fpresult, "nAtom\t\t\t%d\n",  nAtom);
00251   fprintf(fpresult, "nBond\t\t\t%d\n",  nBond);
00252   fprintf(fpresult, "nAtomType\t\t%d\n", nAtomType);
00253   fprintf(fpresult, "nBondType\t\t%d\n",  nBondType);
00254   fprintf(fpresult, "nAtomBlock\t\t%d\n",  nAtomBlock);
00255   fprintf(fpresult, "nAtomInterface\t\t%d\n",  nAtomInterface);
00256   fprintf(fpresult, "nDiscInterface\t\t%d\n",  nDiscInterface);
00257   fprintf(fpresult, "mass\t\t\t%0.6g\n", mass);
00258   fprintf(fpresult, "gamman\t\t\t%0.6g\n", gamman);
00259   fprintf(fpresult, "strain\t\t\t%0.6g\n", strain);
00260   fprintf(fpresult, "strainRate\t\t%0.6g\n", strainRate);
00261   fprintf(fpresult, "FyBylx\t\t\t%0.6g\n", FyBylx);
00262   fprintf(fpresult, "fxByfy\t\t\t%0.6g\n", fxByfy);
00263   fprintf(fpresult, "DeltaY\t\t\t%0.6g\n", DeltaY);
00264   fprintf(fpresult, "DeltaX\t\t\t%0.6g\n", DeltaX);
00265   fprintf(fpresult, "HaltCondition\t\t%0.6g\n", HaltCondition);
00266   fprintf(fpresult, "kappa\t\t\t%g\n", kappa);
00267   fprintf(fpresult, "density\t\t\t%g\n", density);
00268   fprintf(fpresult, "rCut\t\t\t%g\n", rCut);
00269   fprintf(fpresult, "deltaT\t\t\t%g\n", deltaT);
00270   fprintf(fpresult, "stepEquil\t\t%d\n",  stepEquil);
00271   fprintf(fpresult, "stepLimit\t\t%d\n",  stepLimit);
00272   fprintf(fpresult, "region[1]\t\t%0.16lf\n", region[1]);
00273   fprintf(fpresult, "region[2]\t\t%0.16lf\n", region[2]);
00274   fprintf(fpresult, "cells[1]\t\t%d\n",  cells[1]);
00275   fprintf(fpresult, "cells[2]\t\t%d\n",  cells[2]);
00276   fprintf(fpresult, "solver\t\t\t%s\n",  solver);
00277   fprintf(fpresult, "boundary\t\t%s %s\n", xBoundary, yBoundary);
00278   fprintf(fpresult, "DampFlag\t\t%d\n",  DampFlag);
00279   fprintf(fpresult, "-----------------------------------\n");
00280   fprintf(fpresult, "#TimeNow TotalMomentum PotEngyPerAtom KinEngyPerAtom TotEngyPerAtom
      PairEnergyPerAtom BondEnergyPerAtom  Press VirialSum\n");
00281   fprintf(fpvrms, "#timeNow\tVrms \n");
00282   fprintf(fpcom, "#timeNow\tComX\tComY\n");
00283   fprintf(fpforce,
      "#timeNow\tforceSumxAtomType1\tforceSumyAtomType1\tforceSumxAtomType2\tforceSumyAtomType2\tforceSumxAtomType3\tforceSumy
00284
00285 /* //Uncomment the following as per your acquirement
00286     fprintf(fpstress, "strain          %lf\n", strain);
00287     fprintf(fpstress, "region[1]       %lf\n", region[1]);
00288     fprintf(fpstress, "region[2]       %lf\n", region[2]);
00289     fprintf(fpstress, "#timeNow virSumxx virSumyy virSumxy pressure\n");
```

```
00290     fprintf(fpmomentum, "#timeNow Px Py\n");
00291 */
00292
00293    if((strcmp(xBoundary, "p") != 0 && strcmp(xBoundary, "r") != 0) ||
00294     (strcmp(yBoundary, "p") != 0 && strcmp(yBoundary, "r") != 0)) {
00295     fprintf(fpresult, "Error: Invalid boundary value detected: '%s %s'. Only 'p' or 'r' are
      allowed.\n", xBoundary, yBoundary);
00296     exit(EXIT_FAILURE); // Exit with failure status
00297   }
00298
00299 }
```

References atom1, atom2, atomID, atomIDInterface, atomMass, atomRadius, atomType, ax, ay, bigDiameter, BondEnergy, BondID, BondLength, BondType, cellList, cells, DampFlag, deltaT, DeltaX, DeltaY, density, discDragx, discDragy, fax, fay, fpcom, fpforce, fpresult, fpvrms, freezeAtomType, fx, fxByfy, fy, FyBylx, gamman, HaltCondition, ImageX, ImageY, inputConfig, InterfaceWidth, isBonded, kappa, kb, Kn, limitAcfAv, limitCorrAv, limitRdf, mass, mode, molID, nAtom, nAtomBlock, nAtomInterface, nAtomType, nBond, nBondType, nBuffAcf, nBuffCorr, nDiscInterface, nFunCorr, nodeDragx, nodeDragy, nPairTotal, nValAcf, nValCorr, Pairatom1, Pairatom2, PairID, PairXij, PairYij, rangeRdf, rCut, ReadBinaryRestart(), region, regionH, ro, rx, rxUnwrap, ry, ryUnwrap, shearDisplacement, shearVelocity, sizeHistRdf, solver, speed, stepAcf, stepAvg, stepCorr, stepCount, stepDump, stepEquil, stepLimit, stepRdf, stepTraj, strain, strainRate, timeNow, vx, vy, xBoundary, and yBoundary.
Referenced by main().
Here is the call graph for this function:


Here is the caller graph for this function:




#### 5.70.2.19 main()

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

Definition at line 43 of file main.c.

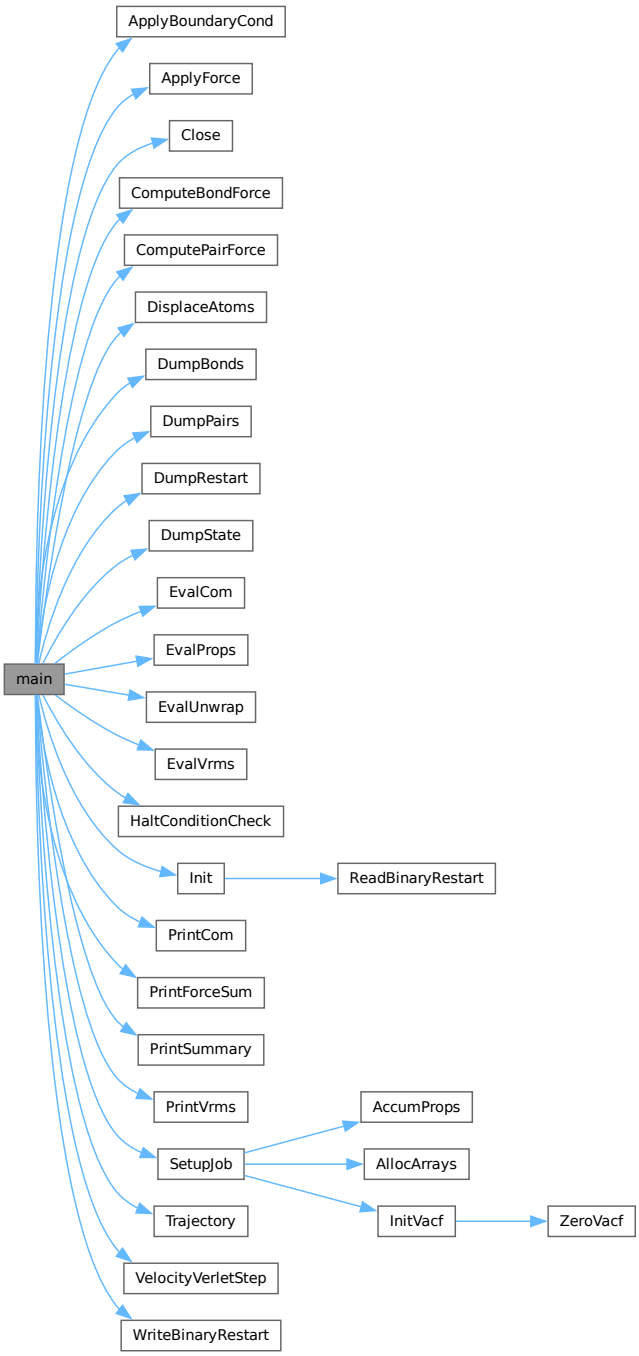
```
00043                                {
00044  time_t t1 = 0, t2;
00045  if (argc < 2) {
00046  fprintf(stderr, "Usage: %s <output_prefix>\n", argv[0]);
00047  return 1;
00048  }
00049
00050  int prefix_size = snprintf(NULL, 0, "../output/%s", argv[1]) + 1; // +1 for the null terminator
00051  prefix = malloc(prefix_size);
00052  if(prefix == NULL) {
00053   fprintf(stderr, "Memory allocation failed\n");
00054  return 1;
00055  }
```

```
00056
00057    // Write the formatted string into the allocated space
00058   snprintf(prefix, prefix_size, "../output/%s", argv[1]);
00059   sprintf(result, "%s.result", prefix);
00060   fpresult = fopen(result, "w");
00061   sprintf(xyz, "%s.xyz", prefix);
00062   fpxyz = fopen(xyz, "w");
00063   sprintf(vrms, "%s.vrms", prefix);
00064   fpvrms = fopen(vrms, "w");
00065   sprintf(bond, "%s.bond", prefix);
00066   fpbond = fopen(bond, "w");
00067   sprintf(com, "%s.com", prefix);
00068   fpcom = fopen(com, "w");
00069   sprintf(pair, "%s.pair", prefix);
00070   fppair = fopen(pair, "w");
00071   sprintf(force, "%s.force", prefix);
00072   fpforce = fopen(force, "w");
00073
00074   /* //Uncomment the following as per your acquirement
00075   sprintf(dnsty, "%s.curr-dnsty", prefix);
00076   fpdnsty = fopen(dnsty, "w");
00077   sprintf(visc, "%s.viscosity", prefix);
00078   fpvisc = fopen(visc, "w");
00079   sprintf(rdf, "%s.rdf", prefix);
00080   fprdf = fopen(rdf, "w");
00081   sprintf(stress, "%s.stress", prefix);
00082   fpstress = fopen(stress, "w");
00083   sprintf(momentum, "%s.momentum", prefix);
00084   fpmomentum = fopen(momentum, "w");
00085   */
00086
00087   Init();
00088   SetupJob();
00089   t1 = time(NULL);
00090   moreCycles = 1;
00091   if(stepCount >= 0) {
00092    if (timeNow == 0.0) {
00093     printf("»> Run type: Fresh simulation «<\n");
00094     DisplaceAtoms();
00095     ComputePairForce(1);
00096     ComputeBondForce();
00097     ApplyForce();
00098     } else {
00099     printf("»> Run type: Restart simulation «<\n");
00100    }
00101    DumpBonds();
00102    DumpPairs();
00103    Trajectory();
00104    EvalUnwrap();
00105    ApplyBoundaryCond();
00106    EvalProps();
00107    EvalVrms();
00108    EvalCom();
00109    PrintVrms();
00110    PrintCom();
00111    PrintSummary();
00112    PrintForceSum();
00113    }
00114
00115 //Here starts the main loop of the program
00116   while(moreCycles){
00117    if(stepLimit == 0){
00118    printf("Error occured: stepLimit must be > 0\n");
00119    printf("Exiting now ...\n");
00120    exit(0);
00121    }
00122
00123    stepCount ++;
00124    timeNow += deltaT ; //stepCount * deltaT; //for adaptive step size: timeNow += deltaT
00125
00126    VelocityVerletStep(1);
00127    EvalUnwrap();
00128    ApplyBoundaryCond();
00129    ComputePairForce(1);
00130    ComputeBondForce();
00131    ApplyForce();
00132    VelocityVerletStep(2);
00133    ApplyBoundaryCond();
00134    EvalProps();
00135    EvalVrms();
00136    EvalCom();
00137    if(stepCount % stepAvg == 0){
00138     PrintSummary();
00139     PrintVrms();
00140     PrintCom();
00141     PrintForceSum();
00142     }
```

```
00143    if(stepCount % stepTraj == 0){
00144     Trajectory();
00145     DumpBonds();
00146     DumpPairs();
00147     }
00148    if(stepCount % stepDump == 0){
00149     DumpRestart();      // Save the current state for input
00150     DumpState();        // Save the current state for config
00151     WriteBinaryRestart();
00152    }
00153    if(HaltConditionCheck(VRootMeanSqr)) {
00154     DumpRestart();      // Save the current state for input
00155     DumpState();        // Save the current state for config
00156     WriteBinaryRestart();
00157     break;  // Exit the loop when the halt condition is met
00158     }
00159
00160     moreCycles ++;
00161     if(moreCycles >= stepLimit)
00162     moreCycles = 0;
00163   }
00164
00165
00166   t2 = time(NULL);
00167   fprintf(fpresult, "#Execution time %lf secs\n", difftime(t2,t1));
00168   fprintf(fpresult, "#Execution speed %lf steps per secs\n", stepLimit/difftime(t2,t1));
00169   printf(">> Simulation run completed <<\n");
00170   printf(">> Execution time %lf secs <<\n", difftime(t2,t1));
00171   printf(">> Execution speed %lf steps per secs << \n", stepLimit/difftime(t2,t1));
00172
00173   fclose(fpresult);
00174   fclose(fpxyz);
00175   fclose(fpvrms);
00176   fclose(fpbond);
00177   fclose(fppair);
00178   fclose(fpcom);
00179   fclose(fpforce);
00180
00181 /*//Uncomment the following as per your acquirement
00182   fclose(fpdnsty);
00183   fclose(fpvisc);
00184   fclose(fprdf);
00185   fclose(fpstress);
00186   fclose(fpmomentum);
00187 */
00188
00189   free(prefix);
00190   Close();
00191   return 0;
00192 }
```

References ApplyBoundaryCond(), ApplyForce(), bond, Close(), com, ComputeBondForce(), ComputePairForce(), deltaT, DisplaceAtoms(), DumpBonds(), DumpPairs(), DumpRestart(), DumpState(), EvalCom(), EvalProps(), EvalUnwrap(), EvalVrms(), force, fpbond, fpcom, fpforce, fppair, fpresult, fpvrms, fpxyz, HaltConditionCheck(), Init(), moreCycles, pair, prefix, PrintCom(), PrintForceSum(), PrintSummary(), PrintVrms(), result, SetupJob(), stepAvg, stepCount, stepDump, stepLimit, stepTraj, timeNow, Trajectory(), VelocityVerletStep(), vrms, VRootMeanSqr, WriteBinaryRestart(), and xyz.

Here is the call graph for this function:




### 5.70.2.20 PrintCom()

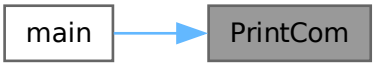
```
void PrintCom ()
```

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:




#### 5.70.2.21 PrintForceSum()

```
void PrintForceSum ()
```

Definition at line 28 of file PrintForceSum.c.

```
00028                    {
00029  int n;
00030  double forceSumxAtomType1, forceSumxAtomType2, forceSumxAtomType3, forceSumxAtomType4,
      forceSumxAtomType5;
00031  double forceSumyAtomType1, forceSumyAtomType2, forceSumyAtomType3, forceSumyAtomType4,
      forceSumyAtomType5;
00032
00033  forceSumxAtomType1 = 0.0;  forceSumyAtomType1 = 0.0;
00034  forceSumxAtomType2 = 0.0;  forceSumyAtomType2 = 0.0;
00035  forceSumxAtomType3 = 0.0;  forceSumyAtomType3 = 0.0;
00036  forceSumxAtomType4 = 0.0;  forceSumyAtomType4 = 0.0;
00037  forceSumxAtomType5 = 0.0;  forceSumyAtomType5 = 0.0;
00038
00039
00040  for(n = 1; n <= nAtom; n++){
00041  if(atomType[n] == 1){
00042   forceSumxAtomType1 += fx[n];
00043   forceSumyAtomType1  += fy[n];
00044   } else if(atomType[n] == 2){
00045   forceSumxAtomType2 += fx[n];
00046   forceSumyAtomType2  += fy[n];
00047   } else if(atomType[n] == 3){
00048   forceSumxAtomType3 += fx[n];
00049   forceSumyAtomType3  += fy[n];
00050   } else if(atomType[n] == 4){
00051   forceSumxAtomType4 += fx[n];
00052   forceSumyAtomType4 += fy[n];
00053   } else if(atomType[n] == 5){
00054   forceSumxAtomType5 += fx[n];
00055   forceSumyAtomType5 += fy[n];
00056   }
00057  }
00058
00059  fprintf(fpforce,
      "%0.4lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16f\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16f\t%0.16f\n",
      timeNow,
00060  forceSumxAtomType1, forceSumyAtomType1,
00061  forceSumxAtomType2, forceSumyAtomType2,
00062  forceSumxAtomType3, forceSumyAtomType3,
00063  forceSumxAtomType4, forceSumyAtomType4,
00064  forceSumxAtomType5, forceSumyAtomType5,
00065  forceSumxExtern, forceSumyExtern);
00066  fflush(fpforce);
00067  }
```

References atomType, forceSumxExtern, forceSumyExtern, fpforce, fx, fy, nAtom, and timeNow.
Referenced by main().

Here is the caller graph for this function:


#### 5.70.2.22 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.

#### 5.70.2.23 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.

#### 5.70.2.24 PrintSummary()

```
void PrintSummary ()
```

Definition at line 25 of file PrintSummary.c.

```
00025                     {
00026   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",
00027     timeNow, vSum, potEnergy, kinEnergy, totEnergy, uSumPairPerAtom, BondEnergyPerAtom, pressure,
      virSum);
00028     fflush(fpresult);
00029 }
```

References BondEnergyPerAtom, fpresult, kinEnergy, potEnergy, pressure, timeNow, totEnergy, uSumPairPerAtom, virSum, and vSum.

Referenced by main().

Here is the caller graph for this function:

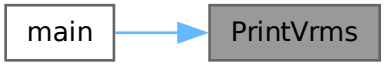

#### 5.70.2.25 PrintVrms()

```
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:



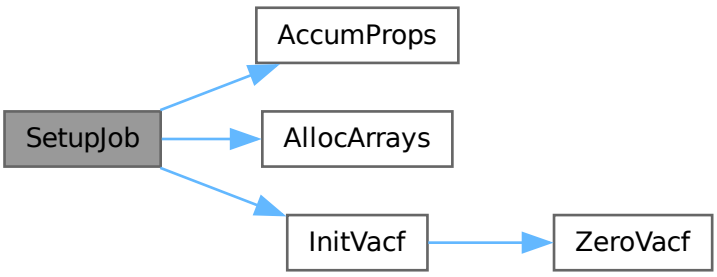

#### 5.70.2.26 SetupJob()

```
void SetupJob ()
```

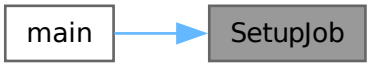
Definition at line 27 of file SetupJob.c.

```
00027                 {
00028   AllocArrays();
00029   AccumProps(0);
00030   InitVacf();
00031   // INITIALISE SPACETIME CORRELATIONS
00032   int n;
00033   for (n = 1; n <= nBuffCorr; n++)
00034     indexCorr[n] = -(n - 1)*nValCorr/nBuffCorr;
00035
00036   countCorrAv = 0.;
00037
00038   for (n = 1; n <= nFunCorr*nValCorr; n++)
00039     spacetimeCorrAv[n] = 0.;
00040
00041   //RDF
00042   countRdf = 0;
00043 }
```

References AccumProps(), AllocArrays(), countCorrAv, countRdf, indexCorr, InitVacf(), nBuffCorr, nFunCorr, nValCorr, and spacetimeCorrAv.
Referenced by main().
Here is the call graph for this function:



Here is the caller graph for this function:



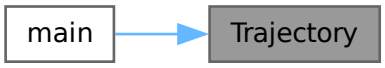

#### 5.70.2.27 Trajectory()

```
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], fx[n], fy[n]);
00040 }
```

References atomID, atomRadius, atomType, fpxyz, fx, fy, molID, nAtom, regionH, rx, ry, timeNow, vx, and vy.
Referenced by main().
Here is the caller graph for this function:




#### 5.70.2.28 VelocityVerletStep()

```
void VelocityVerletStep (
             int icode)
```

Definition at line 26 of file VelocityVerletStep.c.

```
00026                                  {
00027 int n;
00028 double atomMassi;
00029
00030  if(icode == 1){
00031  for (n= 1; n <= nAtom; n++) {
00032   if(atomType[n] != freezeAtomType){
00033   atomMassi = 1./atomMass[n];
00034   ax[n] = fx[n] * atomMassi;    ay[n] = fy[n] * atomMassi;
00035   vx[n] += ax[n] * 0.5 * deltaT;
00036   vy[n] += ay[n] * 0.5 * deltaT;
00037   rx[n] += vx[n] * deltaT;
00038   ry[n] += vy[n] * deltaT;
```

```
00039   }
00040   //Calculating the image flags here
00041   if (rx[n] >= regionH[1]) {
00042    rx[n] -= region[1];
00043    ImageX[n]++;
00044    } else if (rx[n] < -regionH[1]) {
00045    rx[n] += region[1];
00046   ImageX[n]--;
00047   }
00048   if (ry[n] >= regionH[2]) {
00049    ry[n] -= region[2];
00050    ImageY[n]++;
00051    } else if (ry[n] < -regionH[2]) {
00052    ry[n] += region[2];
00053    ImageY[n]--;
00054    } } }
00055   else if(icode == 2){
00056   for(n = 1; n <= nAtom; n++) {
00057   if(atomType[n] != freezeAtomType){
00058    atomMassi = 1./atomMass[n];
00059    ax[n] = fx[n] * atomMassi;    ay[n] = fy[n] * atomMassi;
00060    vx[n] += ax[n] * 0.5 * deltaT;
00061    vy[n] += ay[n] * 0.5 * deltaT;
00062 } } } }
```

References atomMass, atomType, ax, ay, deltaT, freezeAtomType, fx, fy, ImageX, ImageY, nAtom, region, regionH, rx, ry, vx, and vy.

Referenced by main().

Here is the caller graph for this function:


#### 5.70.2.29 WriteBinaryRestart()

```
void WriteBinaryRestart ()
```

Definition at line 60 of file WriteRestartBinary.c.

```
00060                                {
00061   RestartHeader hdr = {
00062   .magic = "LAMINA",
00063   .version = 1.0,
00064   .timeNow = timeNow,
00065   .nAtom = nAtom,
00066   .nBond = nBond,
00067   .nAtomType = nAtomType,
00068   .nBondType = nBondType,
00069   .regionX = region[1],
00070   .regionY = region[2],
00071   .nAtomInterface = nAtomInterface,
00072   .nAtomBlock = nAtomBlock,
00073   .nDiscInterface = nDiscInterface,
00074   .bigDiameter = bigDiameter,
00075   .InterfaceWidth = InterfaceWidth,
00076   .nPairActive = nPairActive,
00077   .nPairTotal = nPairTotal,
00078   .uSumPair = uSumPair,
00079   .virSumPair = virSumPair,
00080   .virSumPairxx = virSumPairxx,
00081   .virSumPairyy = virSumPairyy,
00082   .virSumPairxy = virSumPairxy,
00083   .TotalBondEnergy = TotalBondEnergy,
00084   .virSumBond = virSumBond,
00085   .virSumBondxx = virSumBondxx,
00086   .virSumBondyy = virSumBondyy,
00087   .virSumBondxy = virSumBondxy,
00088   .stepCount = stepCount,
00089   .forceSumxExtern = forceSumxExtern,
00090   .forceSumyExtern = forceSumyExtern
00091   };
```

```
00092
00093  char DUMP[256];
00094  FILE *fp;
00095  sprintf(DUMP, "%s.bin", prefix);  // Produces e.g. "../output/test.bin"
00096  fp = fopen(DUMP, "wb");
00097  if (!fp) {
00098  fprintf(stderr, "Error opening binary restart file %s for writing\n", DUMP);
00099  exit(EXIT_FAILURE);
00100  }
00101
00102  //Here we are writing the data to binary file
00103  fwrite(&hdr, sizeof(RestartHeader), 1, fp);
00104  fwrite(&atomID[1], sizeof(int), nAtom, fp);
00105  fwrite(&molID[1], sizeof(int), nAtom, fp);
00106  fwrite(&atomType[1], sizeof(int), nAtom, fp);
00107  fwrite(&atomRadius[1], sizeof(double), nAtom, fp);
00108  fwrite(&rx[1], sizeof(double), nAtom, fp);
00109  fwrite(&ry[1], sizeof(double), nAtom, fp);
00110  fwrite(&vx[1], sizeof(double), nAtom, fp);
00111  fwrite(&vy[1], sizeof(double), nAtom, fp);
00112  fwrite(&ax[1], sizeof(double), nAtom, fp);
00113  fwrite(&ay[1], sizeof(double), nAtom, fp);
00114  fwrite(&fx[1], sizeof(double), nAtom, fp);
00115  fwrite(&fy[1], sizeof(double), nAtom, fp);
00116  fwrite(&atomMass[1], sizeof(double), nAtom, fp);
00117  fwrite(&discDragx[1], sizeof(double), nAtom, fp);
00118  fwrite(&discDragy[1], sizeof(double), nAtom, fp);
00119  fwrite(&atomIDInterface[1], sizeof(int), nAtomInterface, fp);
00120
00121  fwrite(&BondID[1], sizeof(int), nBond, fp);
00122  fwrite(&BondType[1], sizeof(int), nBond, fp);
00123  fwrite(&atom1[1], sizeof(int), nBond, fp);
00124  fwrite(&atom2[1], sizeof(int), nBond, fp);
00125  fwrite(&kb[1], sizeof(double), nBond, fp);
00126  fwrite(&ro[1], sizeof(double), nBond, fp);
00127  fwrite(&BondEnergy[1], sizeof(double), nBond, fp);
00128  fwrite(&BondLength[1], sizeof(double), nBond, fp);
00129  fwrite(&nodeDragx[1], sizeof(double), nAtom, fp);
00130  fwrite(&nodeDragy[1], sizeof(double), nAtom, fp);
00131  fwrite(&rxUnwrap[1], sizeof(double), nAtom, fp);
00132  fwrite(&ryUnwrap[1], sizeof(double), nAtom, fp);
00133  fwrite(&ImageX[1], sizeof(int), nAtom, fp);
00134  fwrite(&ImageY[1], sizeof(int), nAtom, fp);
00135
00136  fwrite(&PairID[1], sizeof(int), nPairActive, fp);
00137  fwrite(&Pairatom1[1], sizeof(int), nPairActive, fp);
00138  fwrite(&Pairatom2[1], sizeof(int), nPairActive, fp);
00139  fwrite(&PairXij[1], sizeof(double), nPairActive, fp);
00140  fwrite(&PairYij[1], sizeof(double), nPairActive, fp);
00141
00142  fclose(fp);
00143 }
```

References atom1, atom2, atomID, atomIDInterface, atomMass, atomRadius, atomType, ax, ay, bigDiameter, BondEnergy, BondID, BondLength, BondType, discDragx, discDragy, forceSumxExtern, forceSumyExtern, fx, fy, ImageX, ImageY, InterfaceWidth, kb, molID, nAtom, nAtomBlock, nAtomInterface, nAtomType, nBond, nBondType, nDiscInterface, nodeDragx, nodeDragy, nPairActive, nPairTotal, Pairatom1, Pairatom2, PairID, PairXij, PairYij, prefix, region, ro, rx, rxUnwrap, ry, ryUnwrap, stepCount, timeNow, TotalBondEnergy, uSumPair, virSumBond, virSumBondxx, virSumBondxy, virSumBondyy, virSumPair, virSumPairxx, virSumPairxy, virSumPairyy, vx, and vy.
Referenced by main().
Here is the caller graph for this function:

### 5.70.3 Variable Documentation

#### 5.70.3.1 prefix

```
char* prefix = NULL
```

Definition at line 13 of file main.c.
Referenced by DumpRestart(), DumpState(), main(), and WriteBinaryRestart().

## 5.71 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> //For future parallel version
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 void Init();
00015 void SetupJob();
00016 void EvalSpacetimeCorr();
00017 void Trajectory();
00018 void DumpState();
00019 void ComputeForcesCells();
00020 void ApplyBoundaryCond();
00021 void EvalProps();
00022 void AccumProps(int icode);
00023 void PrintSummary();
00024 void PrintVrms();
00025 void VelocityVerletStep(int icode);
00026 void ApplyForce();
00027 void ApplyLeesEdwardsBoundaryCond();
00028 void PrintStress();
00029 void Close();
00030 void PrintMomentum();
00031 void DisplaceAtoms();
00032 void DumpRestart();
00033 bool HaltConditionCheck(double value);
00034 void EvalCom();
00035 void PrintCom();
00036 void EvalVrms();
00037 void EvalUnwrap();
00038 void DumpBonds();
00039 void DumpPairs();
00040 void WriteBinaryRestart();
00041 void PrintForceSum();
00042
00043 int main(int argc, char **argv) {
00044  time_t t1 = 0, t2;
00045  if (argc < 2) {
00046  fprintf(stderr, "Usage: %s <output_prefix>\n", argv[0]);
00047  return 1;
00048  }
00049
00050  int prefix_size = snprintf(NULL, 0, "../output/%s", argv[1]) + 1; // +1 for the null terminator
00051  prefix = malloc(prefix_size);
00052  if(prefix == NULL) {
00053   fprintf(stderr, "Memory allocation failed\n");
00054  return 1;
00055  }
00056
00057    // Write the formatted string into the allocated space
00058   snprintf(prefix, prefix_size, "../output/%s", argv[1]);
00059   sprintf(result, "%s.result", prefix);
00060   fpresult = fopen(result, "w");
00061   sprintf(xyz, "%s.xyz", prefix);
00062   fpxyz = fopen(xyz, "w");
00063   sprintf(vrms, "%s.vrms", prefix);
00064   fpvrms = fopen(vrms, "w");
00065   sprintf(bond, "%s.bond", prefix);
00066   fpbond = fopen(bond, "w");
00067   sprintf(com, "%s.com", prefix);
00068   fpcom = fopen(com, "w");
00069   sprintf(pair, "%s.pair", prefix);
00070   fppair = fopen(pair, "w");
00071   sprintf(force, "%s.force", prefix);
```

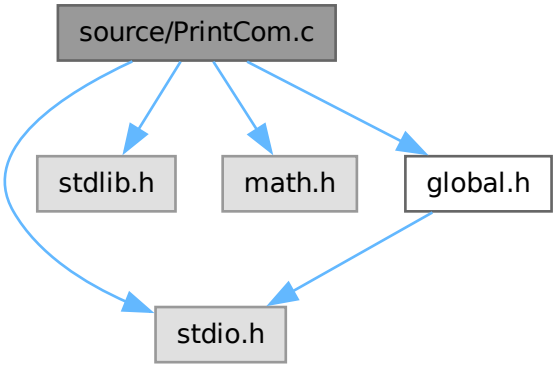
```
00072   fpforce = fopen(force, "w");
00073
00074   /* //Uncomment the following as per your acquirement
00075   sprintf(dnsty, "%s.curr-dnsty", prefix);
00076   fpdnsty = fopen(dnsty, "w");
00077   sprintf(visc, "%s.viscosity", prefix);
00078   fpvisc = fopen(visc, "w");
00079   sprintf(rdf, "%s.rdf", prefix);
00080   fprdf = fopen(rdf, "w");
00081   sprintf(stress, "%s.stress", prefix);
00082   fpstress = fopen(stress, "w");
00083   sprintf(momentum, "%s.momentum", prefix);
00084   fpmomentum = fopen(momentum, "w");
00085   */
00086
00087   Init();
00088   SetupJob();
00089   t1 = time(NULL);
00090   moreCycles = 1;
00091   if(stepCount >= 0) {
00092    if (timeNow == 0.0) {
00093     printf("»> Run type: Fresh simulation «<\n");
00094     DisplaceAtoms();
00095     ComputePairForce(1);
00096     ComputeBondForce();
00097     ApplyForce();
00098     } else {
00099     printf("»> Run type: Restart simulation «<\n");
00100    }
00101    DumpBonds();
00102    DumpPairs();
00103    Trajectory();
00104    EvalUnwrap();
00105    ApplyBoundaryCond();
00106    EvalProps();
00107    EvalVrms();
00108    EvalCom();
00109    PrintVrms();
00110    PrintCom();
00111    PrintSummary();
00112    PrintForceSum();
00113    }
00114
00115 //Here starts the main loop of the program
00116   while(moreCycles){
00117    if(stepLimit == 0){
00118    printf("Error occured: stepLimit must be > 0\n");
00119    printf("Exiting now ...\n");
00120    exit(0);
00121    }
00122
00123    stepCount ++;
00124    timeNow += deltaT ; //stepCount * deltaT; //for adaptive step size: timeNow += deltaT
00125
00126    VelocityVerletStep(1);
00127    EvalUnwrap();
00128    ApplyBoundaryCond();
00129    ComputePairForce(1);
00130    ComputeBondForce();
00131    ApplyForce();
00132    VelocityVerletStep(2);
00133    ApplyBoundaryCond();
00134    EvalProps();
00135    EvalVrms();
00136    EvalCom();
00137    if(stepCount % stepAvg == 0){
00138     PrintSummary();
00139     PrintVrms();
00140     PrintCom();
00141     PrintForceSum();
00142     }
00143    if(stepCount % stepTraj == 0){
00144     Trajectory();
00145     DumpBonds();
00146     DumpPairs();
00147     }
00148    if(stepCount % stepDump == 0){
00149     DumpRestart();      // Save the current state for input
00150     DumpState();        // Save the current state for config
00151     WriteBinaryRestart();
00152    }
00153    if(HaltConditionCheck(VRootMeanSqr)) {
00154     DumpRestart();      // Save the current state for input
00155     DumpState();        // Save the current state for config
00156     WriteBinaryRestart();
00157     break;  // Exit the loop when the halt condition is met
00158     }
```

```
00159
00160     moreCycles ++;
00161     if(moreCycles >= stepLimit)
00162     moreCycles = 0;
00163   }
00164
00165
00166   t2 = time(NULL);
00167   fprintf(fpresult, "#Execution time %lf secs\n", difftime(t2,t1));
00168   fprintf(fpresult, "#Execution speed %lf steps per secs\n", stepLimit/difftime(t2,t1));
00169   printf(">> Simulation run completed <<\n");
00170   printf(">> Execution time %lf secs <<\n", difftime(t2,t1));
00171   printf(">> Execution speed %lf steps per secs << \n", stepLimit/difftime(t2,t1));
00172
00173   fclose(fpresult);
00174   fclose(fpxyz);
00175   fclose(fpvrms);
00176   fclose(fpbond);
00177   fclose(fppair);
00178   fclose(fpcom);
00179   fclose(fpforce);
00180
00181 /*//Uncomment the following as per your acquirement
00182   fclose(fpdnsty);
00183   fclose(fpvisc);
00184   fclose(fprdf);
00185   fclose(fpstress);
00186   fclose(fpmomentum);
00187 */
00188
00189   free(prefix);
00190   Close();
00191   return 0;
00192 }
```

## 5.72 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 ()

### 5.72.1 Function Documentation

#### 5.72.1.1 PrintCom()

```
void PrintCom ()
```

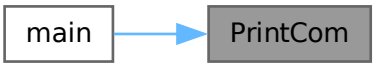
Definition at line 28 of file PrintCom.c.

```
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:




## 5.73 PrintCom.c

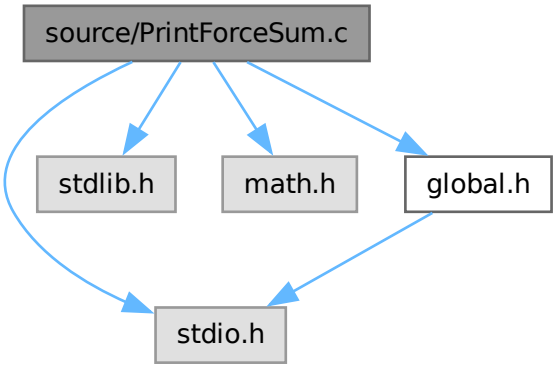
Go to the documentation of this file.

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022
00023 #include<stdio.h>
00024 #include<stdlib.h>
00025 #include<math.h>
00026 #include"global.h"
00027
00028 void PrintCom(){
00029  fprintf(fpcom, "%0.4lf\t%0.16lf\t%0.16lf\n", timeNow, ComX, ComY);
00030  fflush(fpcom);
00031  }
00032
00033
00034
```

## 5.74 source/PrintForceSum.c File Reference

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

Include dependency graph for PrintForceSum.c:




### Functions

- void PrintForceSum ()

## 5.74.1 Function Documentation

### 5.74.1.1 PrintForceSum()

```
void PrintForceSum ()
```

Definition at line 28 of file PrintForceSum.c.

```
00028                       {
00029  int n;
00030  double forceSumxAtomType1, forceSumxAtomType2, forceSumxAtomType3, forceSumxAtomType4,
      forceSumxAtomType5;
00031  double forceSumyAtomType1, forceSumyAtomType2, forceSumyAtomType3, forceSumyAtomType4,
      forceSumyAtomType5;
00032
00033  forceSumxAtomType1 = 0.0;  forceSumyAtomType1 = 0.0;
00034  forceSumxAtomType2 = 0.0;  forceSumyAtomType2 = 0.0;
00035  forceSumxAtomType3 = 0.0;  forceSumyAtomType3 = 0.0;
00036  forceSumxAtomType4 = 0.0;  forceSumyAtomType4 = 0.0;
00037  forceSumxAtomType5 = 0.0;  forceSumyAtomType5 = 0.0;
00038
00039
00040  for(n = 1; n <= nAtom; n++){
00041  if(atomType[n] == 1){
00042   forceSumxAtomType1 += fx[n];
00043   forceSumyAtomType1  += fy[n];
00044   } else if(atomType[n] == 2){
00045   forceSumxAtomType2 += fx[n];
00046   forceSumyAtomType2  += fy[n];
00047   } else if(atomType[n] == 3){
00048   forceSumxAtomType3 += fx[n];
00049   forceSumyAtomType3  += fy[n];
00050   } else if(atomType[n] == 4){
00051   forceSumxAtomType4 += fx[n];
00052   forceSumyAtomType4 += fy[n];
00053   } else if(atomType[n] == 5){
00054   forceSumxAtomType5 += fx[n];
00055   forceSumyAtomType5 += fy[n];
00056   }
00057  }
00058
00059  fprintf(fpforce,
      "%0.4lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16f\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16f\t%0.16f\n",
      timeNow,
00060  forceSumxAtomType1, forceSumyAtomType1,
00061  forceSumxAtomType2, forceSumyAtomType2,
00062  forceSumxAtomType3, forceSumyAtomType3,
00063  forceSumxAtomType4, forceSumyAtomType4,
```

```
00064  forceSumxAtomType5, forceSumyAtomType5,
00065  forceSumxExtern, forceSumyExtern);
00066  fflush(fpforce);
00067  }
```

References atomType, forceSumxExtern, forceSumyExtern, fpforce, fx, fy, nAtom, and timeNow.
Referenced by main().
Here is the caller graph for this function:


## 5.75 PrintForceSum.c

Go to the documentation of this file.

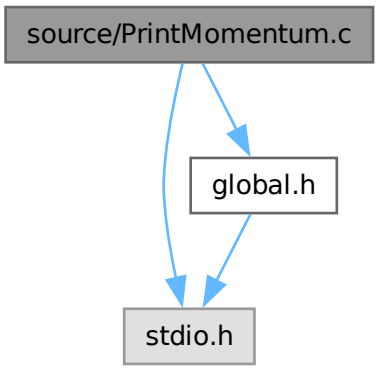
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022
00023 #include<stdio.h>
00024 #include<stdlib.h>
00025 #include<math.h>
00026 #include"global.h"
00027
00028 void PrintForceSum(){
00029  int n;
00030  double forceSumxAtomType1, forceSumxAtomType2, forceSumxAtomType3, forceSumxAtomType4,
     forceSumxAtomType5;
00031  double forceSumyAtomType1, forceSumyAtomType2, forceSumyAtomType3, forceSumyAtomType4,
     forceSumyAtomType5;
00032
00033  forceSumxAtomType1 = 0.0;  forceSumyAtomType1 = 0.0;
00034  forceSumxAtomType2 = 0.0;  forceSumyAtomType2 = 0.0;
00035  forceSumxAtomType3 = 0.0;  forceSumyAtomType3 = 0.0;
00036  forceSumxAtomType4 = 0.0;  forceSumyAtomType4 = 0.0;
00037  forceSumxAtomType5 = 0.0;  forceSumyAtomType5 = 0.0;
00038
00039
00040  for(n = 1; n <= nAtom; n++){
00041  if(atomType[n] == 1){
00042   forceSumxAtomType1 += fx[n];
00043   forceSumyAtomType1  += fy[n];
00044   } else if(atomType[n] == 2){
00045   forceSumxAtomType2 += fx[n];
00046   forceSumyAtomType2  += fy[n];
00047   } else if(atomType[n] == 3){
00048   forceSumxAtomType3 += fx[n];
00049   forceSumyAtomType3  += fy[n];
00050   } else if(atomType[n] == 4){
00051   forceSumxAtomType4 += fx[n];
00052   forceSumyAtomType4 += fy[n];
00053   } else if(atomType[n] == 5){
```

```
00054   forceSumxAtomType5 += fx[n];
00055   forceSumyAtomType5 += fy[n];
00056   }
00057  }
00058
00059  fprintf(fpforce,
      "%0.4lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16f\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16f\t%0.16f\n",
      timeNow,
00060  forceSumxAtomType1, forceSumyAtomType1,
00061  forceSumxAtomType2, forceSumyAtomType2,
00062  forceSumxAtomType3, forceSumyAtomType3,
00063  forceSumxAtomType4, forceSumyAtomType4,
00064  forceSumxAtomType5, forceSumyAtomType5,
00065  forceSumxExtern, forceSumyExtern);
00066  fflush(fpforce);
00067  }
00068
00069
00070
```

## 5.76 source/PrintMomentum.c File Reference

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

Include dependency graph for PrintMomentum.c:




### Functions

- void PrintMomentum ()

### 5.76.1 Function Documentation

#### 5.76.1.1 PrintMomentum()

```
void PrintMomentum ()
```

Definition at line 25 of file PrintMomentum.c.

```
00025                   {
00026  fprintf(fpmomentum, "%0.4lf\t%0.16lf\t%0.16lf\n", timeNow, vSumX, vSumY);
00027  fflush(fpmomentum);
00028 }
```

References fpmomentum, timeNow, vSumX, and vSumY.

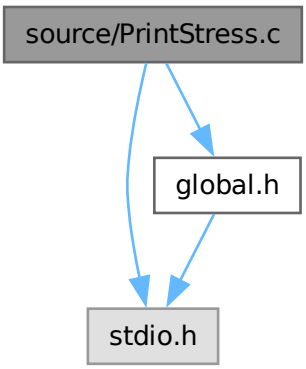
## 5.77 PrintMomentum.c

Go to the documentation of this file.

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 void PrintMomentum(){
00026  fprintf(fpmomentum, "%0.4lf\t%0.16lf\t%0.16lf\n", timeNow, vSumX, vSumY);
00027  fflush(fpmomentum);
00028 }
```

## 5.78 source/PrintStress.c File Reference

`#include <stdio.h>`
`#include "global.h"`
Include dependency graph for PrintStress.c:




### Functions

- void PrintStress ()

### 5.78.1 Function Documentation

#### 5.78.1.1 PrintStress()

`void PrintStress ()`
Definition at line 25 of file PrintStress.c.

```
00025                 {
00026   fprintf(fpstress, "%0.4lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\n", timeNow, virSumxx, virSumyy,
      virSumxy, pressure);
```

```
00027   fflush(fpstress);
00028 }
```

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

## 5.79 PrintStress.c

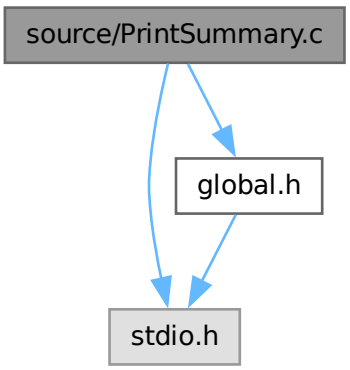
Go to the documentation of this file.

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 void PrintStress(){
00026   fprintf(fpstress, "%0.4lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\n", timeNow, virSumxx, virSumyy,
      virSumxy, pressure);
00027   fflush(fpstress);
00028 }
```

## 5.80 source/PrintSummary.c File Reference

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

Include dependency graph for PrintSummary.c:




### Functions

- void PrintSummary ()

### 5.80.1 Function Documentation

#### 5.80.1.1 PrintSummary()

```
void PrintSummary ()
```

Definition at line 25 of file PrintSummary.c.

```
00025                   {
00026  fprintf(fpresult, "%0.4lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\n",
00027   timeNow, vSum, potEnergy, kinEnergy, totEnergy, uSumPairPerAtom, BondEnergyPerAtom, pressure,
      virSum);
00028   fflush(fpresult);
00029 }
```

References BondEnergyPerAtom, fpresult, kinEnergy, potEnergy, pressure, timeNow, totEnergy, uSumPairPerAtom, virSum, and vSum.

Referenced by main().

Here is the caller graph for this function:


## 5.81 PrintSummary.c

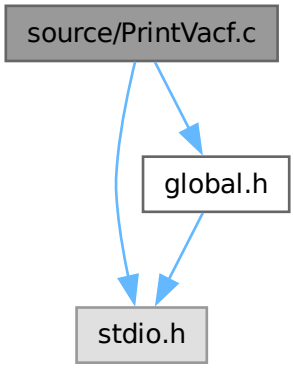
Go to the documentation of this file.

```
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 PrintSummary(){
00026  fprintf(fpresult, "%0.4lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\t%0.16lf\n",
00027   timeNow, vSum, potEnergy, kinEnergy, totEnergy, uSumPairPerAtom, BondEnergyPerAtom, pressure,
      virSum);
00028   fflush(fpresult);
00029 }
```

## 5.82 source/PrintVacf.c File Reference

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

Include dependency graph for PrintVacf.c:




### Functions

- void PrintVacf ()

## 5.82.1 Function Documentation

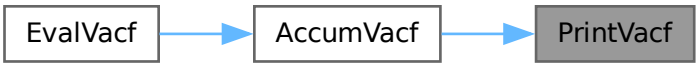
### 5.82.1.1 PrintVacf()

```
void PrintVacf ()
```

Definition at line 25 of file PrintVacf.c.

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

References deltaT, fpvisc, nValAcf, stepAcf, viscAcfAv, and viscAcfInt.
Referenced by AccumVacf().
Here is the caller graph for this function:




# 5.83 PrintVacf.c

Go to the documentation of this file.

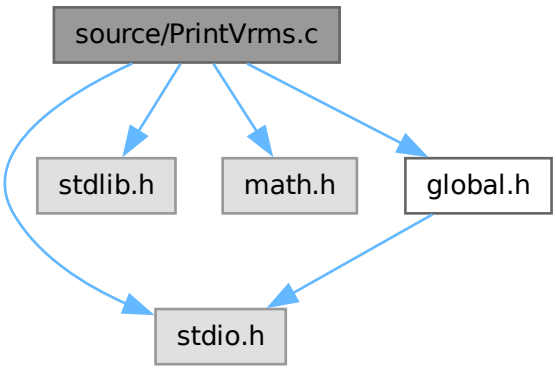
```
00001 /*
00002  * This file is part of Lamina.
```

```
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 void PrintVacf(){
00026  double tVal;
00027  int j;
00028  fprintf(fpvisc,"viscosity acf\n");
00029  for(j = 1 ; j <= nValAcf ; j ++){
00030   tVal = (j-1)*stepAcf*deltaT;
00031   fprintf(fpvisc, "%lf\t %lf\t %lf\n", tVal, viscAcfAv[j], viscAcfAv[j]/viscAcfAv[1]);
00032   }
00033  fprintf(fpvisc, "viscosity acf integral : %lf\n", viscAcfInt);
00034 }
00035
00036
```

## 5.84 source/PrintVrms.c File Reference

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

Include dependency graph for PrintVrms.c:




### Functions

- void PrintVrms ()

### 5.84.1 Function Documentation

#### 5.84.1.1 PrintVrms()

```
void PrintVrms ()
```

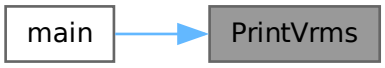
Definition at line 27 of file PrintVrms.c.

```
00027                {
00028  fprintf(fpvrms, "%0.4lf\t%0.16lf\n", timeNow, VRootMeanSqr);
00029  fflush(fpvrms);
00030 }
```

References fpvrms, timeNow, and VRootMeanSqr.

Referenced by main().

Here is the caller graph for this function:




## 5.85 PrintVrms.c

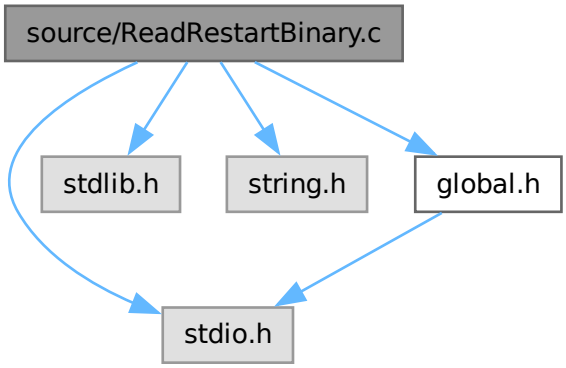
Go to the documentation of this file.

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022 #include<stdio.h>
00023 #include<stdlib.h>
00024 #include<math.h>
00025 #include"global.h"
00026
00027 void PrintVrms(){
00028  fprintf(fpvrms, "%0.4lf\t%0.16lf\n", timeNow, VRootMeanSqr);
00029  fflush(fpvrms);
00030 }
00031
00032
00033
```

## 5.86 source/ReadRestartBinary.c File Reference

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

Include dependency graph for ReadRestartBinary.c:




### Classes

- struct RestartHeader

### Functions

- void ReadBinaryRestart (const char *filename)

## 5.86.1 Function Documentation

### 5.86.1.1 ReadBinaryRestart()

```
void ReadBinaryRestart (
            const char * filename)
```

Definition at line 59 of file ReadRestartBinary.c.

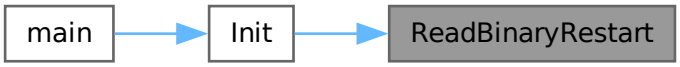
```
00059                                           {
00060
00061   FILE *fp = fopen(filename, "rb");
00062   if (!fp) {
00063     fprintf(stderr, "Error opening binary restart file %s for reading\n", filename);
00064     exit(EXIT_FAILURE);
00065   }
00066   RestartHeader hdr;  //Declare here
00067   fread(&hdr, sizeof(RestartHeader), 1, fp);   //Use it
00068
00069   if(strncmp(hdr.magic, "LAMINA", 6) != 0) {
00070    fprintf(stderr, "Invalid file format: magic = %.8s [from %s()]\n", hdr.magic, __func__);
00071    fclose(fp);
00072    exit(EXIT_FAILURE);  //Must return void, not return 1
00073   }
00074
00075   //Now assigned the values that were read from binary file to global parameters
00076   timeNow = hdr.timeNow;
00077   nAtom = hdr.nAtom;
00078   nBond = hdr.nBond;
00079   nAtomType = hdr.nAtomType;
00080   nBondType = hdr.nBondType;
00081   region[1] = hdr.regionX;
00082   region[2] = hdr.regionY;
00083   nAtomInterface = hdr.nAtomInterface;
00084   nAtomBlock = hdr.nAtomBlock;
00085   nDiscInterface = hdr.nDiscInterface;
00086   bigDiameter = hdr.bigDiameter;
00087   InterfaceWidth = hdr.InterfaceWidth;
00088   nPairActive = hdr.nPairActive;
00089   nPairTotal = hdr.nPairTotal;
00090   uSumPair = hdr.uSumPair;
00091   virSumPair = hdr.virSumPair;
```

```
00092   virSumPairxx = hdr.virSumPairxx;
00093   virSumPairyy = hdr.virSumPairyy;
00094   virSumPairxy = hdr.virSumPairxy;
00095   TotalBondEnergy = hdr.TotalBondEnergy;
00096   virSumBond = hdr.virSumBond;
00097   virSumBondxx = hdr.virSumBondxx;
00098   virSumBondyy = hdr.virSumBondyy;
00099   virSumBondxy = hdr.virSumBondxy;
00100   stepCount = hdr.stepCount;
00101   forceSumxExtern = hdr.forceSumxExtern;
00102   forceSumyExtern = hdr.forceSumyExtern;
00103
00104   density = nAtom / (region[1] * region[2]);
00105   cells[1] = region[1] / rCut;
00106   cells[2] = region[2] / rCut;
00107   regionH[1] = 0.5 * region[1];
00108   regionH[2] = 0.5 * region[2];
00109
00110   cellList = (int *)malloc((nAtom + cells[1] * cells[2] + 1) * sizeof(int));
00111
00112   printf("Running from restart file:\n");
00113   printf("Running: %s, version: %0.3lf\n", hdr.magic, hdr.version);
00114   printf("timeNow: %lf\n", timeNow);
00115   printf("stepCount: %d\n", stepCount);
00116
00117   //Allocating the memory to arrays
00118   atomID = (int *)malloc((nAtom + 1) * sizeof(int));
00119   molID = (int *)malloc((nAtom + 1) * sizeof(int));
00120   atomType = (int *)malloc((nAtom + 1) * sizeof(int));
00121   atomRadius = (double *)malloc((nAtom + 1) * sizeof(double));
00122   rx = (double *)malloc((nAtom + 1) * sizeof(double));
00123   ry = (double *)malloc((nAtom + 1) * sizeof(double));
00124   vx = (double *)malloc((nAtom + 1) * sizeof(double));
00125   vy = (double *)malloc((nAtom + 1) * sizeof(double));
00126   ax = (double *)malloc((nAtom + 1) * sizeof(double));
00127   ay = (double *)malloc((nAtom + 1) * sizeof(double));
00128   fx = (double *)malloc((nAtom + 1) * sizeof(double));
00129   fy = (double *)malloc((nAtom + 1) * sizeof(double));
00130   atomMass = (double *)malloc((nAtom + 1) * sizeof(double));
00131   discDragx = (double *)malloc((nAtom + 1) * sizeof(double));
00132   discDragy = (double *)malloc((nAtom + 1) * sizeof(double));
00133   atomIDInterface = (int *)malloc((nAtom + 1) * sizeof(int));
00134
00135   BondID = (int *)malloc((nBond + 1) * sizeof(int));
00136   BondType = (int *)malloc((nBond + 1) * sizeof(int));
00137   atom1 = (int *)malloc((nBond + 1) * sizeof(int));
00138   atom2 = (int *)malloc((nBond + 1) * sizeof(int));
00139   kb = (double *)malloc((nBond + 1) * sizeof(double));
00140   ro = (double *)malloc((nBond + 1) * sizeof(double));
00141   BondEnergy = (double *)malloc((nBond + 1) * sizeof(double));
00142   BondLength = (double *)malloc((nBond + 1) * sizeof(double));
00143   nodeDragx = (double *)malloc((nAtom + 1) * sizeof(double));
00144   nodeDragy = (double *)malloc((nAtom + 1) * sizeof(double));
00145   rxUnwrap = (double *)malloc((nAtom + 1) * sizeof(double));
00146   ryUnwrap = (double *)malloc((nAtom + 1) * sizeof(double));
00147   ImageX = (int *)malloc((nAtom + 1) * sizeof(int));
00148   ImageY = (int *)malloc((nAtom + 1) * sizeof(int));
00149
00150   PairID = (int *)malloc((nPairTotal + 1) * sizeof(int));
00151   Pairatom1 = (int *)malloc((nPairTotal + 1) * sizeof(int));
00152   Pairatom2 = (int *)malloc((nPairTotal + 1) * sizeof(int));
00153   PairXij = (double *)malloc((nPairTotal + 1) * sizeof(double));
00154   PairYij = (double *)malloc((nPairTotal + 1) * sizeof(double));
00155
00156  //Here we are reading the data to binary file
00157  fread(&atomID[1], sizeof(int), nAtom, fp);
00158  fread(&molID[1], sizeof(int), nAtom, fp);
00159  fread(&atomType[1], sizeof(int), nAtom, fp);
00160  fread(&atomRadius[1], sizeof(double), nAtom, fp);
00161  fread(&rx[1], sizeof(double), nAtom, fp);
00162  fread(&ry[1], sizeof(double), nAtom, fp);
00163  fread(&vx[1], sizeof(double), nAtom, fp);
00164  fread(&vy[1], sizeof(double), nAtom, fp);
00165  fread(&ax[1], sizeof(double), nAtom, fp);
00166  fread(&ay[1], sizeof(double), nAtom, fp);
00167  fread(&fx[1], sizeof(double), nAtom, fp);
00168  fread(&fy[1], sizeof(double), nAtom, fp);
00169  fread(&atomMass[1], sizeof(double), nAtom, fp);
00170  fread(&discDragx[1], sizeof(double), nAtom, fp);
00171  fread(&discDragy[1], sizeof(double), nAtom, fp);
00172  fread(&atomIDInterface[1], sizeof(int), nAtomInterface, fp);
00173
00174  fread(&BondID[1], sizeof(int), nBond, fp);
00175  fread(&BondType[1], sizeof(int), nBond, fp);
00176  fread(&atom1[1], sizeof(int), nBond, fp);
00177  fread(&atom2[1], sizeof(int), nBond, fp);
00178  fread(&kb[1], sizeof(double), nBond, fp);
```

```
00179  fread(&ro[1], sizeof(double), nBond, fp);
00180  fread(&BondEnergy[1], sizeof(double), nBond, fp);
00181  fread(&BondLength[1], sizeof(double), nBond, fp);
00182  fread(&nodeDragx[1], sizeof(double), nAtom, fp);
00183  fread(&nodeDragy[1], sizeof(double), nAtom, fp);
00184  fread(&rxUnwrap[1], sizeof(double), nAtom, fp);
00185  fread(&ryUnwrap[1], sizeof(double), nAtom, fp);
00186  fread(&ImageX[1], sizeof(int), nAtom, fp);
00187  fread(&ImageY[1], sizeof(int), nAtom, fp);
00188
00189  fread(&PairID[1], sizeof(int), nPairActive, fp);
00190  fread(&Pairatom1[1], sizeof(int), nPairActive, fp);
00191  fread(&Pairatom2[1], sizeof(int), nPairActive, fp);
00192  fread(&PairXij[1], sizeof(double), nPairActive, fp);
00193  fread(&PairYij[1], sizeof(double), nPairActive, fp);
00194
00195  //2D-List of bonded atoms. This is used to remove pair interaction
00196  //calculation for the bonded atoms
00197  isBonded = (int **)malloc((nAtom + 1) * sizeof(int*));
00198  for (int i = 0; i <= nAtom; i++) {
00199   isBonded[i] = (int *)malloc((nAtom + 1) * sizeof(int));
00200   for (int j = 0; j <= nAtom; j++) {
00201   isBonded[i][j] = 0;
00202   } }
00203   for (int n = 1; n <= nBond; n++) {
00204    int i = atom1[n];
00205    int j = atom2[n];
00206    isBonded[i][j] = 1;
00207    isBonded[j][i] = 1; // symmetric
00208 }
00209
00210   fprintf(fpresult, "-----------------------------------\n");
00211   fprintf(fpresult, "-------PARAMETERS-----------\n");
00212   fprintf(fpresult, "-----------------------------------\n");
00213   fprintf(fpresult, "nAtom\t\t\t%d\n",  nAtom);
00214   fprintf(fpresult, "nBond\t\t\t%d\n",  nBond);
00215   fprintf(fpresult, "nAtomType\t\t%d\n", nAtomType);
00216   fprintf(fpresult, "nBondType\t\t%d\n",  nBondType);
00217   fprintf(fpresult, "nAtomBlock\t\t%d\n",  nAtomBlock);
00218   fprintf(fpresult, "nAtomInterface\t\t%d\n",  nAtomInterface);
00219   fprintf(fpresult, "nDiscInterface\t\t%d\n",  nDiscInterface);
00220   fprintf(fpresult, "mass\t\t\t%0.6g\n", mass);
00221   fprintf(fpresult, "gamman\t\t\t%0.6g\n", gamman);
00222   fprintf(fpresult, "strain\t\t\t%0.6g\n", strain);
00223   fprintf(fpresult, "strainRate\t\t%0.6g\n", strainRate);
00224   fprintf(fpresult, "FyBylx\t\t\t%0.6g\n", FyBylx);
00225   fprintf(fpresult, "fxByfy\t\t\t%0.6g\n", fxByfy);
00226   fprintf(fpresult, "DeltaY\t\t\t%0.6g\n", DeltaY);
00227   fprintf(fpresult, "DeltaX\t\t\t%0.6g\n", DeltaX);
00228   fprintf(fpresult, "HaltCondition\t\t%0.6g\n", HaltCondition);
00229   fprintf(fpresult, "kappa\t\t\t%g\n", kappa);
00230   fprintf(fpresult, "density\t\t\t%g\n", density);
00231   fprintf(fpresult, "rCut\t\t\t%g\n", rCut);
00232   fprintf(fpresult, "deltaT\t\t\t%g\n", deltaT);
00233   fprintf(fpresult, "stepEquil\t\t%d\n",  stepEquil);
00234   fprintf(fpresult, "stepLimit\t\t%d\n",  stepLimit);
00235   fprintf(fpresult, "region[1]\t\t%0.16lf\n", region[1]);
00236   fprintf(fpresult, "region[2]\t\t%0.16lf\n", region[2]);
00237   fprintf(fpresult, "cells[1]\t\t%d\n",  cells[1]);
00238   fprintf(fpresult, "cells[2]\t\t%d\n",  cells[2]);
00239   fprintf(fpresult, "solver\t\t\t%s\n",  solver);
00240   fprintf(fpresult, "boundary\t\t%s %s\n", xBoundary, yBoundary);
00241   fprintf(fpresult, "DampFlag\t\t%d\n",  DampFlag);
00242   fprintf(fpresult, "-----------------------------------\n");
00243   fprintf(fpresult, "#TimeNow TotalMomentum PotEngyPerAtom KinEngyPerAtom TotEngyPerAtom
      PairEnergyPerAtom BondEnergyPerAtom  Press VirialSum\n");
00244   fprintf(fpvrms, "#timeNow\tVrms \n");
00245   fprintf(fpcom, "#timeNow\tComX\tComY\n");
00246   fprintf(fpforce,
      "#timeNow\tforceSumxAtomType1\tforceSumyAtomType1\tforceSumxAtomType2\tforceSumyAtomType2\tforceSumxAtomType3\tforceSumy
00247   fclose(fp);
00248  }
```

References atom1, atom2, atomID, atomIDInterface, atomMass, atomRadius, atomType, ax, ay, bigDiameter, RestartHeader::bigDiameter, BondEnergy, BondID, BondLength, BondType, cellList, cells, DampFlag, deltaT, DeltaX, DeltaY, density, discDragx, discDragy, forceSumxExtern, RestartHeader::forceSumxExtern, forceSumyExtern, RestartHeader::forceSumyExtern, fpcom, fpforce, fpresult, fpvrms, fx, fxByfy, fy, FyBylx, gamman, HaltCondition, ImageX, ImageY, InterfaceWidth, RestartHeader::InterfaceWidth, isBonded, kappa, kb, RestartHeader::magic, mass, molID, nAtom, RestartHeader::nAtom, nAtomBlock, RestartHeader::nAtomBlock, nAtomInterface, RestartHeader::nAtomInterface, nAtomType, RestartHeader::nAtomType, nBond, RestartHeader::nBond, nBondType, RestartHeader::nBondType, nDiscInterface, RestartHeader::nDiscInterface, nodeDragx, nodeDragy, nPairActive, RestartHeader::nPairActive, nPairTotal, RestartHeader::nPairTotal, Pairatom1, Pairatom2, PairID, PairXij, PairYij, rCut, region, regionH, RestartHeader::regionX, RestartHeader::regionY, ro, rx, rxUnwrap, ry,

ryUnwrap, solver, RestartHeader::stepCount, stepCount, stepEquil, stepLimit, strain, strainRate, RestartHeader::timeNow, timeNow, RestartHeader::TotalBondEnergy, TotalBondEnergy, RestartHeader::uSumPair, uSumPair, RestartHeader::version, RestartHeader::virSumBond, virSumBond, RestartHeader::virSumBondxx, virSumBondxx, RestartHeader::virSumBondxy, virSumBondxy, RestartHeader::virSumBondyy, virSumBondyy, RestartHeader::virSumPair, virSumPair, RestartHeader::virSumPairxx, virSumPairxx, RestartHeader::virSumPairxy, virSumPairxy, RestartHeader::virSumPairyy, virSumPairyy, vx, vy, xBoundary, and yBoundary.
Referenced by Init().
Here is the caller graph for this function:




## 5.87 ReadRestartBinary.c

Go to the documentation of this file.

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021 #include <stdio.h>
00022 #include <stdlib.h>
00023 #include <string.h>
00024 #include"global.h"
00025
00026 // Must match RestartHeader definition in WriteBinaryRestart
00027 typedef struct {
00028  char magic[8];         // "LAMINA\0"
00029  double version;
00030  double timeNow;
00031  int nAtom;
00032  int nBond;
00033  int nAtomType;
00034  int nBondType;
00035  double regionX;   // = region[1]
00036  double regionY;   // = region[2]
00037  int nAtomInterface;
00038  int nAtomBlock;
00039  int nDiscInterface;
00040  double bigDiameter;
00041  double InterfaceWidth;
00042  int nPairActive;
00043  int nPairTotal;
00044  double uSumPair;
00045  double virSumPair;
00046  double virSumPairxx;
00047  double virSumPairyy;
00048  double virSumPairxy;
00049  double TotalBondEnergy;
00050  double virSumBond;
00051  double virSumBondxx;
00052  double virSumBondyy;
```

```
00053  double virSumBondxy;
00054  int stepCount;
00055  double forceSumxExtern;
00056  double forceSumyExtern;
00057 } RestartHeader;
00058
00059 void ReadBinaryRestart(const char *filename) {
00060
00061   FILE *fp = fopen(filename, "rb");
00062   if (!fp) {
00063     fprintf(stderr, "Error opening binary restart file %s for reading\n", filename);
00064     exit(EXIT_FAILURE);
00065   }
00066   RestartHeader hdr;  //Declare here
00067   fread(&hdr, sizeof(RestartHeader), 1, fp);   //Use it
00068
00069   if(strncmp(hdr.magic, "LAMINA", 6) != 0) {
00070    fprintf(stderr, "Invalid file format: magic = %.8s [from %s()]\n", hdr.magic, __func__);
00071    fclose(fp);
00072    exit(EXIT_FAILURE);  //Must return void, not return 1
00073   }
00074
00075   //Now assigned the values that were read from binary file to global parameters
00076   timeNow = hdr.timeNow;
00077   nAtom = hdr.nAtom;
00078   nBond = hdr.nBond;
00079   nAtomType = hdr.nAtomType;
00080   nBondType = hdr.nBondType;
00081   region[1] = hdr.regionX;
00082   region[2] = hdr.regionY;
00083   nAtomInterface = hdr.nAtomInterface;
00084   nAtomBlock = hdr.nAtomBlock;
00085   nDiscInterface = hdr.nDiscInterface;
00086   bigDiameter = hdr.bigDiameter;
00087   InterfaceWidth = hdr.InterfaceWidth;
00088   nPairActive = hdr.nPairActive;
00089   nPairTotal = hdr.nPairTotal;
00090   uSumPair = hdr.uSumPair;
00091   virSumPair = hdr.virSumPair;
00092   virSumPairxx = hdr.virSumPairxx;
00093   virSumPairyy = hdr.virSumPairyy;
00094   virSumPairxy = hdr.virSumPairxy;
00095   TotalBondEnergy = hdr.TotalBondEnergy;
00096   virSumBond = hdr.virSumBond;
00097   virSumBondxx = hdr.virSumBondxx;
00098   virSumBondyy = hdr.virSumBondyy;
00099   virSumBondxy = hdr.virSumBondxy;
00100   stepCount = hdr.stepCount;
00101   forceSumxExtern = hdr.forceSumxExtern;
00102   forceSumyExtern = hdr.forceSumyExtern;
00103
00104   density = nAtom / (region[1] * region[2]);
00105   cells[1] = region[1] / rCut;
00106   cells[2] = region[2] / rCut;
00107   regionH[1] = 0.5 * region[1];
00108   regionH[2] = 0.5 * region[2];
00109
00110   cellList = (int *)malloc((nAtom + cells[1] * cells[2] + 1) * sizeof(int));
00111
00112   printf("Running from restart file:\n");
00113   printf("Running: %s, version: %0.3lf\n", hdr.magic, hdr.version);
00114   printf("timeNow: %lf\n", timeNow);
00115   printf("stepCount: %d\n", stepCount);
00116
00117   //Allocating the memory to arrays
00118   atomID = (int *)malloc((nAtom + 1) * sizeof(int));
00119   molID = (int *)malloc((nAtom + 1) * sizeof(int));
00120   atomType = (int *)malloc((nAtom + 1) * sizeof(int));
00121   atomRadius = (double *)malloc((nAtom + 1) * sizeof(double));
00122   rx = (double *)malloc((nAtom + 1) * sizeof(double));
00123   ry = (double *)malloc((nAtom + 1) * sizeof(double));
00124   vx = (double *)malloc((nAtom + 1) * sizeof(double));
00125   vy = (double *)malloc((nAtom + 1) * sizeof(double));
00126   ax = (double *)malloc((nAtom + 1) * sizeof(double));
00127   ay = (double *)malloc((nAtom + 1) * sizeof(double));
00128   fx = (double *)malloc((nAtom + 1) * sizeof(double));
00129   fy = (double *)malloc((nAtom + 1) * sizeof(double));
00130   atomMass = (double *)malloc((nAtom + 1) * sizeof(double));
00131   discDragx = (double *)malloc((nAtom + 1) * sizeof(double));
00132   discDragy = (double *)malloc((nAtom + 1) * sizeof(double));
00133   atomIDInterface = (int *)malloc((nAtom + 1) * sizeof(int));
00134
00135   BondID = (int *)malloc((nBond + 1) * sizeof(int));
00136   BondType = (int *)malloc((nBond + 1) * sizeof(int));
00137   atom1 = (int *)malloc((nBond + 1) * sizeof(int));
00138   atom2 = (int *)malloc((nBond + 1) * sizeof(int));
00139   kb = (double *)malloc((nBond + 1) * sizeof(double));
```

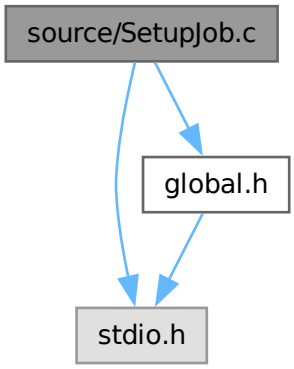
```
00140   ro = (double *)malloc((nBond + 1) * sizeof(double));
00141   BondEnergy = (double *)malloc((nBond + 1) * sizeof(double));
00142   BondLength = (double *)malloc((nBond + 1) * sizeof(double));
00143   nodeDragx = (double *)malloc((nAtom + 1) * sizeof(double));
00144   nodeDragy = (double *)malloc((nAtom + 1) * sizeof(double));
00145   rxUnwrap = (double *)malloc((nAtom + 1) * sizeof(double));
00146   ryUnwrap = (double *)malloc((nAtom + 1) * sizeof(double));
00147   ImageX = (int *)malloc((nAtom + 1) * sizeof(int));
00148   ImageY = (int *)malloc((nAtom + 1) * sizeof(int));
00149
00150   PairID = (int *)malloc((nPairTotal + 1) * sizeof(int));
00151   Pairatom1 = (int *)malloc((nPairTotal + 1) * sizeof(int));
00152   Pairatom2 = (int *)malloc((nPairTotal + 1) * sizeof(int));
00153   PairXij = (double *)malloc((nPairTotal + 1) * sizeof(double));
00154   PairYij = (double *)malloc((nPairTotal + 1) * sizeof(double));
00155
00156  //Here we are reading the data to binary file
00157  fread(&atomID[1], sizeof(int), nAtom, fp);
00158  fread(&molID[1], sizeof(int), nAtom, fp);
00159  fread(&atomType[1], sizeof(int), nAtom, fp);
00160  fread(&atomRadius[1], sizeof(double), nAtom, fp);
00161  fread(&rx[1], sizeof(double), nAtom, fp);
00162  fread(&ry[1], sizeof(double), nAtom, fp);
00163  fread(&vx[1], sizeof(double), nAtom, fp);
00164  fread(&vy[1], sizeof(double), nAtom, fp);
00165  fread(&ax[1], sizeof(double), nAtom, fp);
00166  fread(&ay[1], sizeof(double), nAtom, fp);
00167  fread(&fx[1], sizeof(double), nAtom, fp);
00168  fread(&fy[1], sizeof(double), nAtom, fp);
00169  fread(&atomMass[1], sizeof(double), nAtom, fp);
00170  fread(&discDragx[1], sizeof(double), nAtom, fp);
00171  fread(&discDragy[1], sizeof(double), nAtom, fp);
00172  fread(&atomIDInterface[1], sizeof(int), nAtomInterface, fp);
00173
00174  fread(&BondID[1], sizeof(int), nBond, fp);
00175  fread(&BondType[1], sizeof(int), nBond, fp);
00176  fread(&atom1[1], sizeof(int), nBond, fp);
00177  fread(&atom2[1], sizeof(int), nBond, fp);
00178  fread(&kb[1], sizeof(double), nBond, fp);
00179  fread(&ro[1], sizeof(double), nBond, fp);
00180  fread(&BondEnergy[1], sizeof(double), nBond, fp);
00181  fread(&BondLength[1], sizeof(double), nBond, fp);
00182  fread(&nodeDragx[1], sizeof(double), nAtom, fp);
00183  fread(&nodeDragy[1], sizeof(double), nAtom, fp);
00184  fread(&rxUnwrap[1], sizeof(double), nAtom, fp);
00185  fread(&ryUnwrap[1], sizeof(double), nAtom, fp);
00186  fread(&ImageX[1], sizeof(int), nAtom, fp);
00187  fread(&ImageY[1], sizeof(int), nAtom, fp);
00188
00189  fread(&PairID[1], sizeof(int), nPairActive, fp);
00190  fread(&Pairatom1[1], sizeof(int), nPairActive, fp);
00191  fread(&Pairatom2[1], sizeof(int), nPairActive, fp);
00192  fread(&PairXij[1], sizeof(double), nPairActive, fp);
00193  fread(&PairYij[1], sizeof(double), nPairActive, fp);
00194
00195  //2D-List of bonded atoms. This is used to remove pair interaction
00196  //calculation for the bonded atoms
00197  isBonded = (int **)malloc((nAtom + 1) * sizeof(int*));
00198  for (int i = 0; i <= nAtom; i++) {
00199   isBonded[i] = (int *)malloc((nAtom + 1) * sizeof(int));
00200   for (int j = 0; j <= nAtom; j++) {
00201   isBonded[i][j] = 0;
00202   } }
00203   for (int n = 1; n <= nBond; n++) {
00204    int i = atom1[n];
00205    int j = atom2[n];
00206    isBonded[i][j] = 1;
00207    isBonded[j][i] = 1; // symmetric
00208 }
00209
00210   fprintf(fpresult, "-----------------------------------\n");
00211   fprintf(fpresult, "-------PARAMETERS-----------\n");
00212   fprintf(fpresult, "-----------------------------------\n");
00213   fprintf(fpresult, "nAtom\t\t\t%d\n",  nAtom);
00214   fprintf(fpresult, "nBond\t\t\t%d\n",  nBond);
00215   fprintf(fpresult, "nAtomType\t\t%d\n", nAtomType);
00216   fprintf(fpresult, "nBondType\t\t%d\n",  nBondType);
00217   fprintf(fpresult, "nAtomBlock\t\t%d\n",  nAtomBlock);
00218   fprintf(fpresult, "nAtomInterface\t\t%d\n",  nAtomInterface);
00219   fprintf(fpresult, "nDiscInterface\t\t%d\n",  nDiscInterface);
00220   fprintf(fpresult, "mass\t\t\t%0.6g\n", mass);
00221   fprintf(fpresult, "gamman\t\t\t%0.6g\n", gamman);
00222   fprintf(fpresult, "strain\t\t\t%0.6g\n", strain);
00223   fprintf(fpresult, "strainRate\t\t%0.6g\n", strainRate);
00224   fprintf(fpresult, "FyBylx\t\t\t%0.6g\n", FyBylx);
00225   fprintf(fpresult, "fxByfy\t\t\t%0.6g\n", fxByfy);
00226   fprintf(fpresult, "DeltaY\t\t\t%0.6g\n", DeltaY);
```

```
00227   fprintf(fpresult, "DeltaX\t\t\t%0.6g\n", DeltaX);
00228   fprintf(fpresult, "HaltCondition\t\t%0.6g\n", HaltCondition);
00229   fprintf(fpresult, "kappa\t\t\t%g\n", kappa);
00230   fprintf(fpresult, "density\t\t\t%g\n", density);
00231   fprintf(fpresult, "rCut\t\t\t%g\n", rCut);
00232   fprintf(fpresult, "deltaT\t\t\t%g\n", deltaT);
00233   fprintf(fpresult, "stepEquil\t\t%d\n",  stepEquil);
00234   fprintf(fpresult, "stepLimit\t\t%d\n",  stepLimit);
00235   fprintf(fpresult, "region[1]\t\t%0.16lf\n", region[1]);
00236   fprintf(fpresult, "region[2]\t\t%0.16lf\n", region[2]);
00237   fprintf(fpresult, "cells[1]\t\t%d\n",  cells[1]);
00238   fprintf(fpresult, "cells[2]\t\t%d\n",  cells[2]);
00239   fprintf(fpresult, "solver\t\t\t%s\n",  solver);
00240   fprintf(fpresult, "boundary\t\t%s %s\n", xBoundary, yBoundary);
00241   fprintf(fpresult, "DampFlag\t\t%d\n",  DampFlag);
00242   fprintf(fpresult, "-----------------------------------\n");
00243   fprintf(fpresult, "#TimeNow TotalMomentum PotEngyPerAtom KinEngyPerAtom TotEngyPerAtom
      PairEnergyPerAtom BondEnergyPerAtom  Press VirialSum\n");
00244   fprintf(fpvrms, "#timeNow\tVrms \n");
00245   fprintf(fpcom, "#timeNow\tComX\tComY\n");
00246   fprintf(fpforce,
      "#timeNow\tforceSumxAtomType1\tforceSumyAtomType1\tforceSumxAtomType2\tforceSumyAtomType2\tforceSumxAtomType3\tforceSumy
00247   fclose(fp);
00248  }
00249
```

## 5.88 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 ()

### 5.88.1 Function Documentation

#### 5.88.1.1 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:


#### 5.88.1.2 AllocArrays()

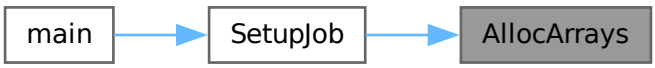
```
void AllocArrays ()
```

Definition at line 25 of file AllocArrays.c.

```
00025                   {
00026   int n;
00027
00028   // SPACETIME CORRELATIONS
00029   cfOrg = (double **) malloc ((nBuffCorr+1)*sizeof(double *));
00030   for (n = 0; n <= nBuffCorr; n++)
00031    cfOrg[n] = (double *) malloc ((2*nFunCorr+1)*sizeof(double));
00032
00033   cfVal = (double *) malloc ((2*nFunCorr+1)*sizeof(double));
00034   indexCorr = (int *) malloc ((nBuffCorr+1)*sizeof(int));
00035
00036   spacetimeCorr = (double **) malloc ((nBuffCorr+1)*sizeof(double));
00037   for (n = 0; n <= nBuffCorr; n++)
00038    spacetimeCorr[n] = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));
00039
00040   spacetimeCorrAv = (double *) malloc ((nFunCorr*nValCorr+1)*sizeof(double));
00041
00042   // VISCOSITY
00043   indexAcf = (double *)malloc((nBuffAcf+1)*sizeof(double));
00044   viscAcf = (double **)malloc((nBuffAcf+1)*sizeof(double *));
00045   for(n = 0 ; n <= nBuffAcf ; n ++)
00046    viscAcf[n] = (double *)malloc((nValAcf+1)*sizeof(double ));
00047
00048   viscAcfOrg = (double *)malloc((nBuffAcf+1)*sizeof(double));
00049   viscAcfAv = (double *)malloc((nValAcf+1)*sizeof(double));
00050
00051    // RDF
00052    histRdf = (double *)malloc((sizeHistRdf+1)*sizeof(double));
```

```
00053 }
```

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:



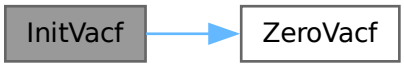

#### 5.88.1.3 InitVacf()

```
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:


#### 5.88.1.4 SetupJob()

```
void SetupJob ()
```

Definition at line 27 of file SetupJob.c.

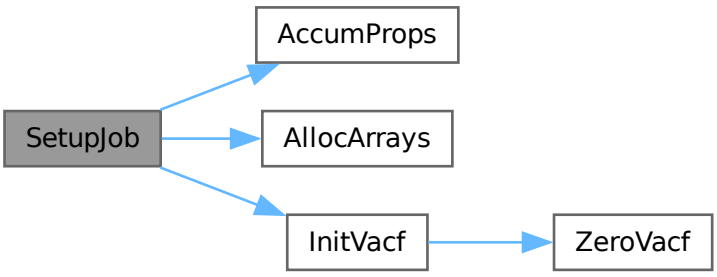
```
00027                {
```

```
00028   AllocArrays();
00029   AccumProps(0);
00030   InitVacf();
00031   // INITIALISE SPACETIME CORRELATIONS
00032   int n;
00033   for (n = 1; n <= nBuffCorr; n++)
00034     indexCorr[n] = -(n - 1)*nValCorr/nBuffCorr;
00035
00036   countCorrAv = 0.;
00037
00038   for (n = 1; n <= nFunCorr*nValCorr; n++)
00039     spacetimeCorrAv[n] = 0.;
00040
00041   //RDF
00042   countRdf = 0;
00043 }
```

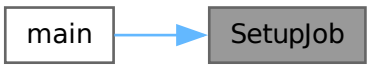
References AccumProps(), AllocArrays(), countCorrAv, countRdf, indexCorr, InitVacf(), nBuffCorr, nFunCorr, nValCorr, and spacetimeCorrAv.

Referenced by main().

Here is the call graph for this function:




Here is the caller graph for this function:




## 5.89 SetupJob.c

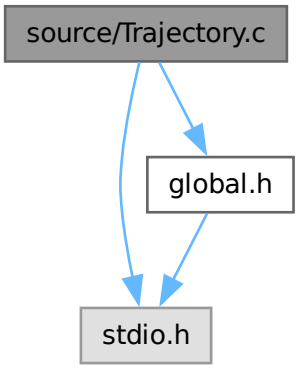
Go to the documentation of this file.

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

```
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021 #include<stdio.h>
00022 #include"global.h"
00023
00024 void AllocArrays();
00025 void AccumProps(int icode);
00026 void InitVacf();
00027 void SetupJob(){
00028   AllocArrays();
00029   AccumProps(0);
00030   InitVacf();
00031   // INITIALISE SPACETIME CORRELATIONS
00032   int n;
00033   for (n = 1; n <= nBuffCorr; n++)
00034     indexCorr[n] = -(n - 1)*nValCorr/nBuffCorr;
00035
00036   countCorrAv = 0.;
00037
00038   for (n = 1; n <= nFunCorr*nValCorr; n++)
00039     spacetimeCorrAv[n] = 0.;
00040
00041   //RDF
00042   countRdf = 0;
00043 }
```

## 5.90 source/Trajectory.c File Reference

`#include <stdio.h>`
`#include "global.h"`
Include dependency graph for Trajectory.c:




### Functions

- void Trajectory ()

### 5.90.1 Function Documentation

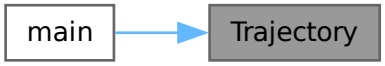
#### 5.90.1.1 Trajectory()

`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], fx[n], fy[n]);
00040 }
```

References atomID, atomRadius, atomType, fpxyz, fx, fy, molID, nAtom, regionH, rx, ry, timeNow, vx, and vy.
Referenced by main().
Here is the caller graph for this function:




## 5.91 Trajectory.c

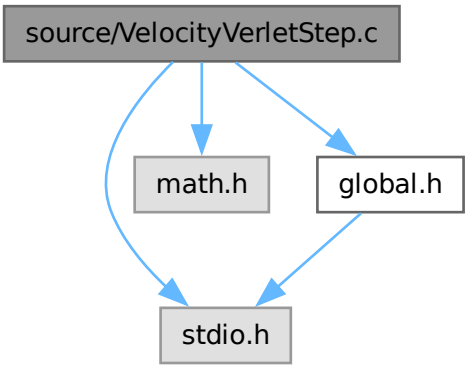
Go to the documentation of this file.

```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022 #include<stdio.h>
00023 #include"global.h"
00024
00025 void Trajectory(){
00026  int n;
00027  //Trajectory file in LAMMPS dump format for OVITO visualization
00028  fprintf(fpxyz, "ITEM: TIMESTEP\n");
00029  fprintf(fpxyz, "%lf\n",timeNow);
00030  fprintf(fpxyz, "ITEM: NUMBER OF ATOMS\n");
00031  fprintf(fpxyz, "%d\n",nAtom);
00032  fprintf(fpxyz, "ITEM: BOX BOUNDS pp ff pp\n");
00033  fprintf(fpxyz, "%lf %lf xlo xhi\n", -regionH[1], regionH[1]);
00034  fprintf(fpxyz, "%lf %lf ylo yhi\n", -regionH[2], regionH[2]);
00035  fprintf(fpxyz, "%lf %lf zlo zhi\n", -0.1, 0.1);
00036  fprintf(fpxyz, "ITEM: ATOMS id mol type radius x y vx vy fx fy\n");
00037  for(n=1; n<=nAtom; n++)
00038   fprintf(fpxyz, "%d\t %d\t %d\t %0.2lf\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], fx[n], fy[n]);
00040 }
00041
00042
00043
```

## 5.92 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)

### 5.92.1 Function Documentation

#### 5.92.1.1 VelocityVerletStep()

```
void VelocityVerletStep (
            int icode)
```

Definition at line 26 of file VelocityVerletStep.c.

```
00026                                {
00027 int n;
00028 double atomMassi;
00029
00030  if(icode == 1){
00031  for (n= 1; n <= nAtom; n++) {
00032   if(atomType[n] != freezeAtomType){
00033   atomMassi = 1./atomMass[n];
00034   ax[n] = fx[n] * atomMassi;    ay[n] = fy[n] * atomMassi;
00035   vx[n] += ax[n] * 0.5 * deltaT;
00036   vy[n] += ay[n] * 0.5 * deltaT;
00037   rx[n] += vx[n] * deltaT;
00038   ry[n] += vy[n] * deltaT;
00039   }
00040   //Calculating the image flags here
00041   if (rx[n] >= regionH[1]) {
00042    rx[n] -= region[1];
00043    ImageX[n]++;
00044    } else if (rx[n] < -regionH[1]) {
00045    rx[n] += region[1];
00046   ImageX[n]--;
00047   }
00048   if (ry[n] >= regionH[2]) {
00049    ry[n] -= region[2];
00050    ImageY[n]++;
00051    } else if (ry[n] < -regionH[2]) {
00052    ry[n] += region[2];
00053    ImageY[n]--;
00054    } } }
00055   else if(icode == 2){
00056   for(n = 1; n <= nAtom; n++) {
```

```
00057   if(atomType[n] != freezeAtomType){
00058    atomMassi = 1./atomMass[n];
00059    ax[n] = fx[n] * atomMassi;    ay[n] = fy[n] * atomMassi;
00060    vx[n] += ax[n] * 0.5 * deltaT;
00061    vy[n] += ay[n] * 0.5 * deltaT;
00062 } } } }
```

References atomMass, atomType, ax, ay, deltaT, freezeAtomType, fx, fy, ImageX, ImageY, nAtom, region, regionH, rx, ry, vx, and vy.

Referenced by main().

Here is the caller graph for this function:


## 5.93 VelocityVerletStep.c

Go to the documentation of this file.

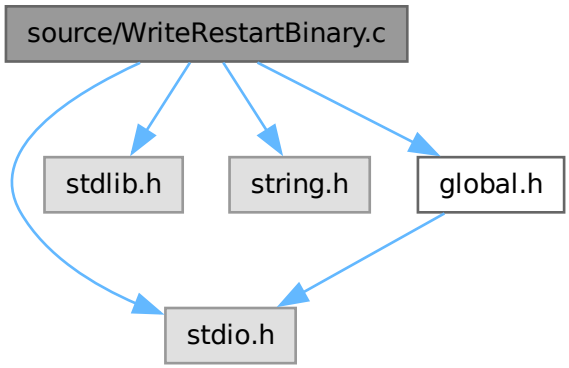
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your option) any later version.
00008  *
00009  * Lamina is distributed in the hope that it will be useful,
00010  * but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  * GNU General Public License for more details.
00013  *
00014  * You should have received a copy of the GNU General Public License
00015  * along with Lamina. If not, see <https://www.gnu.org/licenses/>.
00016
00017  Copyright (C) 2025 Harish Charan, University of Durham, UK
00018
00019  */
00020
00021
00022 #include<stdio.h>
00023 #include<math.h>
00024 #include"global.h"
00025
00026 void VelocityVerletStep(int icode){
00027 int n;
00028 double atomMassi;
00029
00030  if(icode == 1){
00031  for (n= 1; n <= nAtom; n++) {
00032   if(atomType[n] != freezeAtomType){
00033   atomMassi = 1./atomMass[n];
00034   ax[n] = fx[n] * atomMassi;    ay[n] = fy[n] * atomMassi;
00035   vx[n] += ax[n] * 0.5 * deltaT;
00036   vy[n] += ay[n] * 0.5 * deltaT;
00037   rx[n] += vx[n] * deltaT;
00038   ry[n] += vy[n] * deltaT;
00039   }
00040   //Calculating the image flags here
00041   if (rx[n] >= regionH[1]) {
00042    rx[n] -= region[1];
00043    ImageX[n]++;
00044    } else if (rx[n] < -regionH[1]) {
00045    rx[n] += region[1];
00046   ImageX[n]--;
00047   }
00048   if (ry[n] >= regionH[2]) {
00049    ry[n] -= region[2];
00050    ImageY[n]++;
00051    } else if (ry[n] < -regionH[2]) {
00052    ry[n] += region[2];
```

```
00053     ImageY[n]--;
00054     } } }
00055    else if(icode == 2){
00056    for(n = 1; n <= nAtom; n++) {
00057    if(atomType[n] != freezeAtomType){
00058     atomMassi = 1./atomMass[n];
00059     ax[n] = fx[n] * atomMassi;     ay[n] = fy[n] * atomMassi;
00060     vx[n] += ax[n] * 0.5 * deltaT;
00061     vy[n] += ay[n] * 0.5 * deltaT;
00062 } } } }
00063
```

## 5.94 source/WriteRestartBinary.c File Reference

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

Include dependency graph for WriteRestartBinary.c:




### Classes

- struct RestartHeader

### Functions

- void WriteBinaryRestart ()

### 5.94.1 Function Documentation

#### 5.94.1.1 WriteBinaryRestart()

```
void WriteBinaryRestart ()
```

Definition at line 60 of file WriteRestartBinary.c.

```
00060                           {
00061  RestartHeader hdr = {
00062  .magic = "LAMINA",
00063  .version = 1.0,
00064  .timeNow = timeNow,
00065  .nAtom = nAtom,
00066  .nBond = nBond,
00067  .nAtomType = nAtomType,
00068  .nBondType = nBondType,
00069  .regionX = region[1],
00070  .regionY = region[2],
```

```
00071   .nAtomInterface = nAtomInterface,
00072   .nAtomBlock = nAtomBlock,
00073   .nDiscInterface = nDiscInterface,
00074   .bigDiameter = bigDiameter,
00075   .InterfaceWidth = InterfaceWidth,
00076   .nPairActive = nPairActive,
00077   .nPairTotal = nPairTotal,
00078   .uSumPair = uSumPair,
00079   .virSumPair = virSumPair,
00080   .virSumPairxx = virSumPairxx,
00081   .virSumPairyy = virSumPairyy,
00082   .virSumPairxy = virSumPairxy,
00083   .TotalBondEnergy = TotalBondEnergy,
00084   .virSumBond = virSumBond,
00085   .virSumBondxx = virSumBondxx,
00086   .virSumBondyy = virSumBondyy,
00087   .virSumBondxy = virSumBondxy,
00088   .stepCount = stepCount,
00089   .forceSumxExtern = forceSumxExtern,
00090   .forceSumyExtern = forceSumyExtern
00091   };
00092
00093   char DUMP[256];
00094   FILE *fp;
00095   sprintf(DUMP, "%s.bin", prefix);  // Produces e.g. "../output/test.bin"
00096   fp = fopen(DUMP, "wb");
00097   if (!fp) {
00098   fprintf(stderr, "Error opening binary restart file %s for writing\n", DUMP);
00099   exit(EXIT_FAILURE);
00100   }
00101
00102   //Here we are writing the data to binary file
00103   fwrite(&hdr, sizeof(RestartHeader), 1, fp);
00104   fwrite(&atomID[1], sizeof(int), nAtom, fp);
00105   fwrite(&molID[1], sizeof(int), nAtom, fp);
00106   fwrite(&atomType[1], sizeof(int), nAtom, fp);
00107   fwrite(&atomRadius[1], sizeof(double), nAtom, fp);
00108   fwrite(&rx[1], sizeof(double), nAtom, fp);
00109   fwrite(&ry[1], sizeof(double), nAtom, fp);
00110   fwrite(&vx[1], sizeof(double), nAtom, fp);
00111   fwrite(&vy[1], sizeof(double), nAtom, fp);
00112   fwrite(&ax[1], sizeof(double), nAtom, fp);
00113   fwrite(&ay[1], sizeof(double), nAtom, fp);
00114   fwrite(&fx[1], sizeof(double), nAtom, fp);
00115   fwrite(&fy[1], sizeof(double), nAtom, fp);
00116   fwrite(&atomMass[1], sizeof(double), nAtom, fp);
00117   fwrite(&discDragx[1], sizeof(double), nAtom, fp);
00118   fwrite(&discDragy[1], sizeof(double), nAtom, fp);
00119   fwrite(&atomIDInterface[1], sizeof(int), nAtomInterface, fp);
00120
00121   fwrite(&BondID[1], sizeof(int), nBond, fp);
00122   fwrite(&BondType[1], sizeof(int), nBond, fp);
00123   fwrite(&atom1[1], sizeof(int), nBond, fp);
00124   fwrite(&atom2[1], sizeof(int), nBond, fp);
00125   fwrite(&kb[1], sizeof(double), nBond, fp);
00126   fwrite(&ro[1], sizeof(double), nBond, fp);
00127   fwrite(&BondEnergy[1], sizeof(double), nBond, fp);
00128   fwrite(&BondLength[1], sizeof(double), nBond, fp);
00129   fwrite(&nodeDragx[1], sizeof(double), nAtom, fp);
00130   fwrite(&nodeDragy[1], sizeof(double), nAtom, fp);
00131   fwrite(&rxUnwrap[1], sizeof(double), nAtom, fp);
00132   fwrite(&ryUnwrap[1], sizeof(double), nAtom, fp);
00133   fwrite(&ImageX[1], sizeof(int), nAtom, fp);
00134   fwrite(&ImageY[1], sizeof(int), nAtom, fp);
00135
00136   fwrite(&PairID[1], sizeof(int), nPairActive, fp);
00137   fwrite(&Pairatom1[1], sizeof(int), nPairActive, fp);
00138   fwrite(&Pairatom2[1], sizeof(int), nPairActive, fp);
00139   fwrite(&PairXij[1], sizeof(double), nPairActive, fp);
00140   fwrite(&PairYij[1], sizeof(double), nPairActive, fp);
00141
00142   fclose(fp);
00143 }
```

References atom1, atom2, atomID, atomIDInterface, atomMass, atomRadius, atomType, ax, ay, bigDiameter, BondEnergy, BondID, BondLength, BondType, discDragx, discDragy, forceSumxExtern, forceSumyExtern, fx, fy, ImageX, ImageY, InterfaceWidth, kb, molID, nAtom, nAtomBlock, nAtomInterface, nAtomType, nBond, nBondType, nDiscInterface, nodeDragx, nodeDragy, nPairActive, nPairTotal, Pairatom1, Pairatom2, PairID, PairXij, PairYij, prefix, region, ro, rx, rxUnwrap, ry, ryUnwrap, stepCount, timeNow, TotalBondEnergy, uSumPair, virSumBond, virSumBondxx, virSumBondxy, virSumBondyy, virSumPair, virSumPairxx, virSumPairxy, virSumPairyy, vx, and vy.
Referenced by main().

Here is the caller graph for this function:


## 5.95 WriteRestartBinary.c

Go to the documentation of this file.

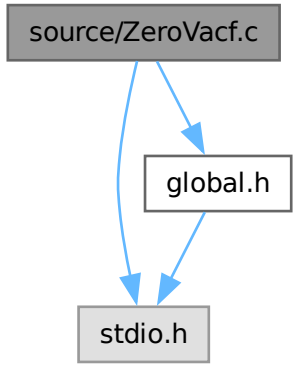
```
00001 /*
00002  * This file is part of Lamina.
00003  *
00004  * Lamina is free software: you can redistribute it and/or modify
00005  * it under the terms of the GNU General Public License as published by
00006  * the Free Software Foundation, either version 3 of the License, or
00007  * (at your 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 <string.h>
00025 #include "global.h"
00026
00027 // Header struct must match that of WriteRestartBinary
00028 typedef struct {
00029  char magic[8];         // "LAMINA\0"
00030  double version;
00031  double timeNow;
00032  int nAtom;
00033  int nBond;
00034  int nAtomType;
00035  int nBondType;
00036  double regionX;   // = region[1]
00037  double regionY;   // = region[2]
00038  int nAtomInterface;
00039  int nAtomBlock;
00040  int nDiscInterface;
00041  double bigDiameter;
00042  double InterfaceWidth;
00043  int nPairActive;
00044  int nPairTotal;
00045  double uSumPair;
00046  double virSumPair;
00047  double virSumPairxx;
00048  double virSumPairyy;
00049  double virSumPairxy;
00050  double TotalBondEnergy;
00051  double virSumBond;
00052  double virSumBondxx;
00053  double virSumBondyy;
00054  double virSumBondxy;
00055  int stepCount;
00056  double forceSumxExtern;
00057  double forceSumyExtern;
00058 } RestartHeader;
00059
00060 void WriteBinaryRestart() {
00061  RestartHeader hdr = {
00062  .magic = "LAMINA",
00063  .version = 1.0,
```

```
00064   .timeNow = timeNow,
00065   .nAtom = nAtom,
00066   .nBond = nBond,
00067   .nAtomType = nAtomType,
00068   .nBondType = nBondType,
00069   .regionX = region[1],
00070   .regionY = region[2],
00071   .nAtomInterface = nAtomInterface,
00072   .nAtomBlock = nAtomBlock,
00073   .nDiscInterface = nDiscInterface,
00074   .bigDiameter = bigDiameter,
00075   .InterfaceWidth = InterfaceWidth,
00076   .nPairActive = nPairActive,
00077   .nPairTotal = nPairTotal,
00078   .uSumPair = uSumPair,
00079   .virSumPair = virSumPair,
00080   .virSumPairxx = virSumPairxx,
00081   .virSumPairyy = virSumPairyy,
00082   .virSumPairxy = virSumPairxy,
00083   .TotalBondEnergy = TotalBondEnergy,
00084   .virSumBond = virSumBond,
00085   .virSumBondxx = virSumBondxx,
00086   .virSumBondyy = virSumBondyy,
00087   .virSumBondxy = virSumBondxy,
00088   .stepCount = stepCount,
00089   .forceSumxExtern = forceSumxExtern,
00090   .forceSumyExtern = forceSumyExtern
00091   };
00092
00093   char DUMP[256];
00094   FILE *fp;
00095   sprintf(DUMP, "%s.bin", prefix);  // Produces e.g. "../output/test.bin"
00096   fp = fopen(DUMP, "wb");
00097   if (!fp) {
00098   fprintf(stderr, "Error opening binary restart file %s for writing\n", DUMP);
00099   exit(EXIT_FAILURE);
00100   }
00101
00102   //Here we are writing the data to binary file
00103   fwrite(&hdr, sizeof(RestartHeader), 1, fp);
00104   fwrite(&atomID[1], sizeof(int), nAtom, fp);
00105   fwrite(&molID[1], sizeof(int), nAtom, fp);
00106   fwrite(&atomType[1], sizeof(int), nAtom, fp);
00107   fwrite(&atomRadius[1], sizeof(double), nAtom, fp);
00108   fwrite(&rx[1], sizeof(double), nAtom, fp);
00109   fwrite(&ry[1], sizeof(double), nAtom, fp);
00110   fwrite(&vx[1], sizeof(double), nAtom, fp);
00111   fwrite(&vy[1], sizeof(double), nAtom, fp);
00112   fwrite(&ax[1], sizeof(double), nAtom, fp);
00113   fwrite(&ay[1], sizeof(double), nAtom, fp);
00114   fwrite(&fx[1], sizeof(double), nAtom, fp);
00115   fwrite(&fy[1], sizeof(double), nAtom, fp);
00116   fwrite(&atomMass[1], sizeof(double), nAtom, fp);
00117   fwrite(&discDragx[1], sizeof(double), nAtom, fp);
00118   fwrite(&discDragy[1], sizeof(double), nAtom, fp);
00119   fwrite(&atomIDInterface[1], sizeof(int), nAtomInterface, fp);
00120
00121   fwrite(&BondID[1], sizeof(int), nBond, fp);
00122   fwrite(&BondType[1], sizeof(int), nBond, fp);
00123   fwrite(&atom1[1], sizeof(int), nBond, fp);
00124   fwrite(&atom2[1], sizeof(int), nBond, fp);
00125   fwrite(&kb[1], sizeof(double), nBond, fp);
00126   fwrite(&ro[1], sizeof(double), nBond, fp);
00127   fwrite(&BondEnergy[1], sizeof(double), nBond, fp);
00128   fwrite(&BondLength[1], sizeof(double), nBond, fp);
00129   fwrite(&nodeDragx[1], sizeof(double), nAtom, fp);
00130   fwrite(&nodeDragy[1], sizeof(double), nAtom, fp);
00131   fwrite(&rxUnwrap[1], sizeof(double), nAtom, fp);
00132   fwrite(&ryUnwrap[1], sizeof(double), nAtom, fp);
00133   fwrite(&ImageX[1], sizeof(int), nAtom, fp);
00134   fwrite(&ImageY[1], sizeof(int), nAtom, fp);
00135
00136   fwrite(&PairID[1], sizeof(int), nPairActive, fp);
00137   fwrite(&Pairatom1[1], sizeof(int), nPairActive, fp);
00138   fwrite(&Pairatom2[1], sizeof(int), nPairActive, fp);
00139   fwrite(&PairXij[1], sizeof(double), nPairActive, fp);
00140   fwrite(&PairYij[1], sizeof(double), nPairActive, fp);
00141
00142   fclose(fp);
00143 }
00144
```

## 5.96 source/ZeroVacf.c File Reference

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

Include dependency graph for ZeroVacf.c:




### Functions

- void ZeroVacf ()

### 5.96.1 Function Documentation

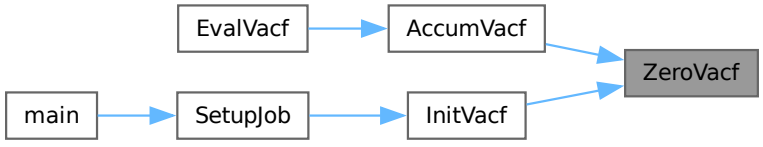
#### 5.96.1.1 ZeroVacf()

```
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:




## 5.97 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