Phase-field Solver with Dynamic Interface Refinement

Generated by Doxygen 1.9.1

Chapter 1

Code-guide for solver phaseFieldDynamic

The solver requires *dynamicInterfaceRefineFvMesh* library, and has been successfully tested using OpenFOAM v6. Before compiling the solver, the library must be obtained in the following procedure:

- You can compile the lib where ever you want. This is just an example:

  ```
  mkdir -p $FOAM_RUN/../OpenFOAM_extensions
  ```

- Switch to this directory

  ```
  cd $FOAM_RUN/../OpenFOAM_extensions
  ```

- Clone the repository

  ```
  git clone https://bitbucket.org/shor-ty/dynamicinterfacerefinefvmesh.git
  ```

- Move to the new library folder

  ```
  cd dynamicinterfacerefinefvmesh
  ```

- Checkout the openfoam version you need (e.g. using 5.x)

  ```
  git checkout OpenFOAM-5.x
  ```

- Go to the library source

  ```
  cd src/dynamicFvMesh
  ```

- Compile the lib

  ```
  wmake libso
  ```

- Finally you have to include the libary into your solver set-up. Therefore add the following line to your dynamicMeshDict

  ```
  dynamicFvMeshLibs ( "libdynamicInterfaceRefineFvMesh.so" );
  ```

- The best way is to copy the dynamicMeshDict to your case and modify it.

1.1 Compiling the solver

- Following commands should create the executable of the solver

  ```
  cd $FOAM_RUN/phaseFieldSolverDynamic/phaseFieldSolverDynamic
  wclean
  wmake
  ```

- The solver can be run by following the instructions in *userGuide*.

1.2 Further details

The implementation, client and header files of the solver have been written following OpenFOAM conventions. These are explained next with flow charts generated from the source code using Doxygen. It must be noted that the solver is based on `laplacianFoam` solver within OpenFOAM. Hence, it may be helpful for the user to become familiar with `OpenFOAM Programmer's Guide` and `laplacianFoam` beforehand.

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

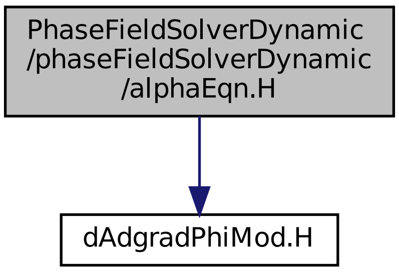
Chapter 3

File Documentation

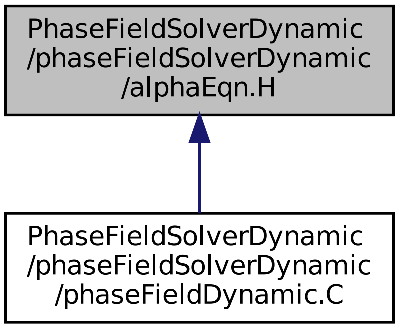
3.1 PhaseFieldSolverDynamic/phaseFieldSolverDynamic/alphaEqn.H File Reference

```
#include "dAdgradPhiMod.H"
```

Include dependency graph for alphaEqn.H:


PhaseFieldSolverDynamic /phaseFieldSolverDynamic /alphaEqn.H
dAdgradPhiMod.H

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



Functions

- Random obj (1)

  Random number generation for adding noise to induce side branching.
- const scalar randNumber (obj.scalar01())
- fvScalarMatrix alphaEqn (omega ∗epsilon ∗dimt ∗fvm::ddt(phi)==2.0 ∗epsilon ∗gamma ∗dimx ∗dimx ∗fvm↩ ::laplacian(ac_01 ∗ac_01, phi)+2 ∗gamma ∗epsilon ∗dimx ∗fvc::div(dadgradPhi) - 18.0 ∗(gamma/epsilon) ∗(phi) ∗(1-phi) ∗(1-2.0 ∗phi)+B ∗(c_Sol-c_Liq) ∗(mu -(2 ∗A ∗c_eq - 2 ∗A ∗(T0-T)/m_1)) ∗30.0 ∗phi ∗phi ∗(1.0-phi) ∗(1.0-phi)+6 ∗noise_mag ∗phi ∗(1.0-phi) ∗phi ∗(1-phi) ∗randNumber)
- alphaEqn solve ()
- fvScalarMatrix muEqn ((0.5) ∗(1/A) ∗dimt ∗fvm::ddt(mu)==diff_Liq ∗0.5 ∗(1/A) ∗dimx ∗dimx ∗fvm↩ ::laplacian((1-phi), mu) -(c_Sol-c_Liq) ∗dimt ∗fvc::ddt(phi) ∗30.0 ∗phi ∗phi ∗(1.0-phi) ∗(1.0-phi) - anti_trap ∗epsilon ∗(c_Sol-c_Liq) ∗dimx ∗fvc::div((n ∗dimt ∗fvc::ddt(phi))))

  Chemical potential equation with approximate slope of c-mu curve according to the parabolic approximation for free energy.

Variables

- volVectorField n =dimx∗fvc::grad(phi)/(1E-20+mag(dimx∗fvc::grad(phi)))

  The unit normal vector to the interface with a small number in denominator to prevent solution from diverging.

3.1.1 Function Documentation

3.1.1.1 obj()

```
Random obj (
            1 )
```

Random number generation for adding noise to induce side branching.

3.1.1.2 randNumber()

```
const scalar randNumber (
             obj.  scalar01() )
```

3.1.1.3 alphaEqn()

```
fvScalarMatrix alphaEqn (
             omega *epsilon *dimt * fvm::ddtphi = =2.0 *epsilon *gamma *dimx *dimx *fvm:↩
:laplacian(ac_01 *ac_01, phi)+2 *gamma *epsilon *dimx *fvc::div(dadgradPhi) - 18.0 *(gamma/epsilon) *(phi) *(1-
)
```

Implicit discretization using fvm class for time derivative and laplacian. Explicit discretization using fvc class for divergence. Phase-field equation with approximate relation between c, mu and T according to the parabolic approximation for free energy

3.1.1.4 solve()

```
muEqn solve ( )
```

3.1.1.5 muEqn()

```
fvScalarMatrix muEqn (
             (0.5) *(1/A) *dimt *fvm::ddt(mu)   = =diff_Liq *0.5 *(1/A) *dimx *dimx *fvm:↩
:laplacian((1-phi), mu) -(c_Sol-c_Liq) *dimt *fvc::ddt(phi) *30.0 *phi *phi *(1.0-phi) *(1.0-phi) - anti_trap *
::div((n *dimt *fvc::ddt(phi))) )
```

Chemical potential equation with approximate slope of c-mu curve according to the parabolic approximation for free energy.

3.1.2 Variable Documentation

3.1.2.1 n

```
volVectorField n =dimx*fvc::grad(phi)/(1E-20+mag(dimx*fvc::grad(phi)))
```

The unit normal vector to the interface with a small number in denominator to prevent solution from diverging.

Definition at line 26 of file alphaEqn.H.

3.2 PhaseFieldSolverDynamic/phaseFieldSolverDynamic/correctPhi.H File Reference

Functions

- surfaceScalarField ("alpha", fvc::interpolate(alpha))

  The interpolation for smoothening of the fields.
- surfaceScalarField ("mu", fvc::interpolate(mu))
- geometricZeroField ()

3.2.1 Function Documentation

3.2.1.1 surfaceScalarField() [1/2]

```
surfaceScalarField (
             "alpha" ,
             fvc::interpolate(alpha)  )
```

The interpolation for smoothening of the fields.

3.2.1.2 surfaceScalarField() [2/2]

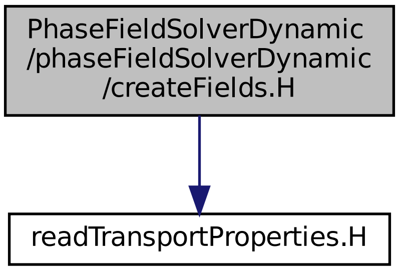
```
surfaceScalarField (
             "mu" ,
             fvc::interpolate(mu)  )
```

3.2.1.3 geometricZeroField()

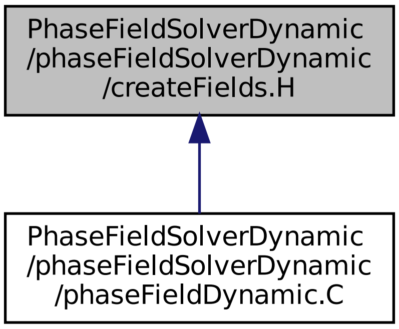
```
geometricZeroField ( )
```

3.3 PhaseFieldSolverDynamic/phaseFieldSolverDynamic/createFields.H File Reference

```
#include "readTransportProperties.H"
```
Include dependency graph for createFields.H:



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



Functions

- volScalarField phi (IOobject("phi", runTime.timeName(), mesh, IOobject::MUST_READ, IOobject::AUTO_↩ WRITE), mesh)

  Creating phase-field with the option to write.
- volScalarField mu (IOobject("mu", runTime.timeName(), mesh, IOobject::MUST_READ, IOobject::AUTO_↩ WRITE), mesh)

  Creating chemical potential field with the option to write.

3.3.1 Function Documentation

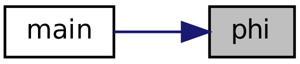
3.3.1.1 phi()

```
volScalarField phi (
            IOobject("phi", runTime.timeName(), mesh, IOobject::MUST_READ, IOobject::AUTO_↩
WRITE) ,
            mesh  )
```

Creating phase-field with the option to write.

Referenced by main().

Here is the caller graph for this function:



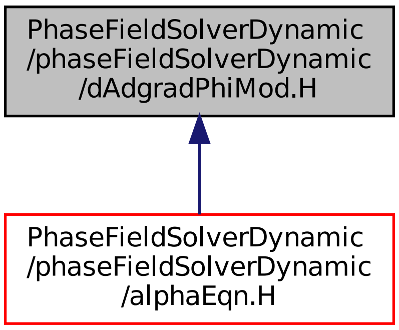
3.3.1.2 mu()

```
volScalarField mu (
            IOobject("mu", runTime.timeName(), mesh, IOobject::MUST_READ, IOobject::AUTO_↩
WRITE) ,
            mesh  )
```

Creating chemical potential field with the option to write.

3.4 PhaseFieldSolverDynamic/phaseFieldSolverDynamic/dAdgradPhi↩ Mod.H File Reference

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



Variables

- volVectorField q =dimx∗fvc::grad(phi)

  Normal vector to the interface, q.

3.4.1 Variable Documentation

3.4.1.1 q

```
volVectorField q =dimx*fvc::grad(phi)
```

Normal vector to the interface, q.

Definition at line 3 of file dAdgradPhiMod.H.

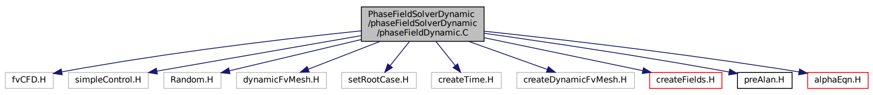
Referenced by main().

3.5 PhaseFieldSolverDynamic/phaseFieldSolverDynamic/info.md File Reference

3.6 PhaseFieldSolverDynamic/phaseFieldSolverDynamic/phaseField↩ Dynamic.C File Reference

```
#include "fvCFD.H"
#include "simpleControl.H"
#include "Random.H"
#include "dynamicFvMesh.H"
#include "setRootCase.H"
#include "createTime.H"
#include "createDynamicFvMesh.H"
#include "createFields.H"
#include "preAlan.H"
#include "alphaEqn.H"
```

Include dependency graph for phaseFieldDynamic.C:



Functions

- int main (int argc, char ∗argv[ ])

  Using dynamicFvMesh for dynamic interface refinement.

3.6.1 Function Documentation

3.6.1.1 main()

```
int main (
            int argc,
            char ∗ argv[ ] )
```

Using dynamicFvMesh for dynamic interface refinement.

The imposed temperature field as a function of thermal gradient in the x direction, G and pulling velocity, v

The interpolation equation is used for smoothening of the phase-field variable

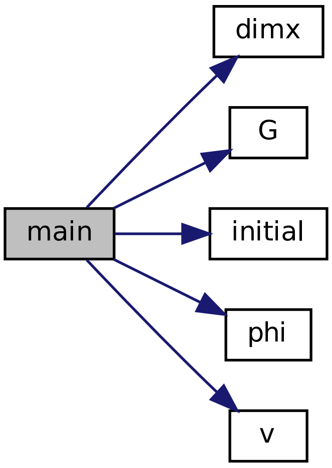
Solving the phase-field and chemical potential equations after updating the mesh

Writing the results according to keywords in controlDict

Definition at line 40 of file phaseFieldDynamic.C.

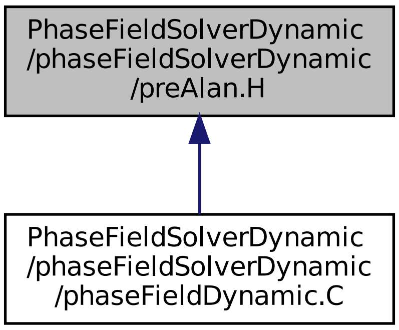
References dimx(), G(), initial(), phi(), q, and v().

Here is the call graph for this function:



3.7 PhaseFieldSolverDynamic/phaseFieldSolverDynamic/preAlan.H File Reference

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



Functions

- fvScalarMatrix alpha1Eqn (omega ∗epsilon ∗dimt ∗fvm::ddt(phi) -epsilon ∗gamma ∗dimx ∗dimx ∗fvm↩ ::laplacian(phi)+2.0 ∗phi ∗(phi-1.0) ∗(2.0 ∗phi-1.0)/(epsilon))
- alpha1Eqn solve ()

3.7.1 Function Documentation

3.7.1.1 alpha1Eqn()

```
fvScalarMatrix alpha1Eqn (
            omega *epsilon *dimt * fvm::ddtphi) -epsilon *gamma *dimx *dimx *fvm::laplacian(phi)+2.↩
0 *phi *(phi-1.0) *(2.0 *phi-1.0)/(epsilon )
```

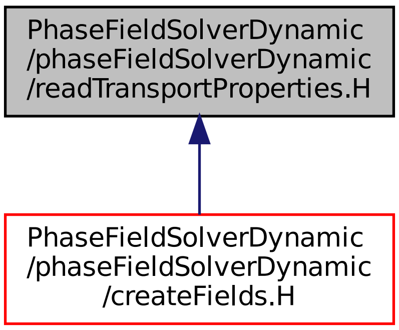
The interpolation equation for smoothening of the phase-field variable. Implicit discretization using fvm class for time derivative and laplacian.

3.7.1.2 solve()

```
alpha1Eqn solve ( )
```

3.8 PhaseFieldSolverDynamic/phaseFieldSolverDynamic/read↩ TransportProperties.H File Reference

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


PhaseFieldSolverDynamic /phaseFieldSolverDynamic /readTransportProperties.H
PhaseFieldSolverDynamic /phaseFieldSolverDynamic /createFields.H

Functions

- IOdictionary transportProperties (IOobject("transportProperties", runTime.constant(), mesh, IOobject::↩ MUST_READ, IOobject::NO_WRITE))

  The input properties to be exported to createFields.
- dimensionedScalar dimt (transportProperties.lookup("dimt"))

  Dimension of time.
- dimensionedScalar dimx (transportProperties.lookup("dimx"))

Dimension of position.

- dimensionedScalar m_1 (transportProperties.lookup("m_1"))

  Slope liquidus.
- dimensionedScalar m_0 (transportProperties.lookup("m_0"))

  Slope solidus.
- dimensionedScalar omega (transportProperties.lookup("omega"))

  Relaxation coefficient for phi (order parameter)
- dimensionedScalar gamma (transportProperties.lookup("gamma"))

  Surface Energy.
- dimensionedScalar epsilon (transportProperties.lookup("epsilon"))

  Interface Width.
- dimensionedScalar c_Sol (transportProperties.lookup("c_Sol"))

  Composition of solid in equilibrium with liquid.
- dimensionedScalar c_Liq (transportProperties.lookup("c_Liq"))

  Composition of liquid in equilibrium with solid.
- dimensionedScalar c_eq (transportProperties.lookup("c_eq"))

  Equilibrium composition or average composition of alloy.
- dimensionedScalar anti_trap (transportProperties.lookup("anti_trap"))

  Anti-trapping coefficient.
- dimensionedScalar diff_Sol (transportProperties.lookup("diff_Sol"))

  Diffusivity in solid.
- dimensionedScalar diff_Liq (transportProperties.lookup("diff_Liq"))

  Diffusivity in liquid.
- dimensionedScalar G (transportProperties.lookup("G"))

  Thermal gradient.
- dimensionedScalar v (transportProperties.lookup("v"))

  Velocity.
- dimensionedScalar delta_01 (transportProperties.lookup("delta_01"))

  Strength of anisotropy.
- dimensionedScalar A (transportProperties.lookup("A"))
- dimensionedScalar B (transportProperties.lookup("B"))
- dimensionedScalar T0 (transportProperties.lookup("T0"))

  Melting temperature.
- dimensionedScalar noise_mag (transportProperties.lookup("noise_mag"))

  Noise magnitude.
- dimensionedScalar initial (transportProperties.lookup("initial"))

  Constant value from temperature profile.

Variables

- dimensionedScalar pi = constant::mathematical::pi

  The input properties are read from constant/transportProperties dictionary.

3.8.1 Function Documentation

3.8.1.1 transportProperties()

```
IOdictionary transportProperties (
            IOobject("transportProperties", runTime.constant(), mesh, IOobject::MUST_READ,
IOobject::NO_WRITE)  )
```

The input properties to be exported to createFields.

3.8.1.2 dimt()

```
dimensionedScalar dimt (
            transportProperties.  lookup"dimt" )
```

Dimension of time.

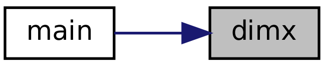
3.8.1.3 dimx()

```
dimensionedScalar dimx (
            transportProperties.  lookup"dimx" )
```

Dimension of position.

Referenced by main().

Here is the caller graph for this function:



3.8.1.4 m_1()

```
dimensionedScalar m_1 (
            transportProperties.  lookup"m_1" )
```

Slope liquidus.

3.8.1.5 m_0()

```
dimensionedScalar m_0 (
            transportProperties.  lookup"m_0" )
```

Slope solidus.

3.8.1.6 omega()

```
dimensionedScalar omega (
            transportProperties.  lookup"omega" )
```

Relaxation coefficient for phi (order parameter)

3.8.1.7 gamma()

```
dimensionedScalar gamma (
            transportProperties.  lookup"gamma" )
```

Surface Energy.

3.8.1.8 epsilon()

```
dimensionedScalar epsilon (
            transportProperties.  lookup"epsilon" )
```

Interface Width.

3.8.1.9 c_Sol()

```
dimensionedScalar c_Sol (
            transportProperties.  lookup"c_Sol" )
```

Composition of solid in equilibrium with liquid.

3.8.1.10 c_Liq()

```
dimensionedScalar c_Liq (
            transportProperties.  lookup"c_Liq" )
```

Composition of liquid in equilibrium with solid.

3.8.1.11 c_eq()

```
dimensionedScalar c_eq (
            transportProperties.  lookup"c_eq" )
```

Equilibrium composition or average composition of alloy.

3.8.1.12 anti_trap()

```
dimensionedScalar anti_trap (
            transportProperties.  lookup"anti_trap" )
```

Anti-trapping coefficient.

3.8.1.13 diff_Sol()

```
dimensionedScalar diff_Sol (
            transportProperties.  lookup"diff_Sol" )
```

Diffusivity in solid.

3.8.1.14 diff_Liq()

```
dimensionedScalar diff_Liq (
            transportProperties.  lookup"diff_Liq" )
```

Diffusivity in liquid.

3.8.1.15 G()

```
dimensionedScalar G (
            transportProperties.  lookup"G" )
```

Thermal gradient.

Referenced by main().

Here is the caller graph for this function:

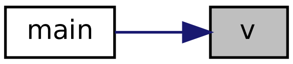
3.8.1.16 v()

```
dimensionedScalar v (
            transportProperties.  lookup"v" )
```

Velocity.

Referenced by main().

Here is the caller graph for this function:



3.8.1.17 delta_01()

```
dimensionedScalar delta_01 (
            transportProperties.  lookup"delta_01" )
```

Strength of anisotropy.

3.8.1.18 A()

```
dimensionedScalar A (
            transportProperties.  lookup"A" )
```

3.8.1.19 B()

```
dimensionedScalar B (
            transportProperties.  lookup"B" )
```

3.8.1.20 T0()

```
dimensionedScalar T0 (
            transportProperties.  lookup"T0" )
```

Melting temperature.

3.8.1.21 noise_mag()

```
dimensionedScalar noise_mag (
            transportProperties.  lookup"noise_mag" )
```

Noise magnitude.

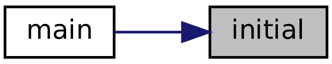
3.8.1.22 initial()

```
dimensionedScalar initial (
            transportProperties.  lookup"initial" )
```

Constant value from temperature profile.

Referenced by main().

Here is the caller graph for this function:



3.8.2 Variable Documentation

3.8.2.1 pi

```
dimensionedScalar pi = constant::mathematical::pi
```

The input properties are read from constant/transportProperties dictionary.

Definition at line 15 of file readTransportProperties.H.

3.9 PhaseFieldSolverDynamic/phaseFieldSolverDynamic/write.H File Reference

Functions

- if (runTime.writeTime())

  For writing the results from main.

3.9.1 Function Documentation

3.9.1.1 if()

```
if (
            runTime. writeTime() )
```

For writing the results from main.

Definition at line 2 of file write.H.