ASTEROPE

0.1

Generated by Doxygen 1.8.13

Contents

Chapter 1

Main Page

1.1 Getting Started

Welcome to the documentation of the Phase-Field Solver based on MFEM

1.1.1 Installation guide

1. **Prerequisites**
2. **Installation of the code**

1.1.2 Basics features for building phase-field applications

1. **Code overview**
2. **How to build of a phase-field application?**

1.1.3 User examples

1. **Diffusion problems**
2. **Allen-Cahn problems**
3. **Cahn-Hilliard problems**

1.1.4 Doxygen documentation

1. List of namespaces
2. Data structures
3. Files

1.1.5 Licence

(to be defined)

1.1.6 Contact

Please use the `GitLab issue tracker` to report bugs or post questions or comments.

1.1.7 Contributors

- `Clément Introïni`

Chapter 2

installation

A straightforward way to install MFEM is to use spack

- First, clone spack and install it (see spack)
- Second, run the following command to install mfem with right additional packages

```
./bin/spack install mfem +mpi  +debug +openmp +petsc +strumpack +suite-sparse +sundials +superlu-dist
```

- Third, apply the following changes in the config.mk file
  - remove the C++14 standard to C++17 in order to avoid compilation errors (MFEM_CXXFLAGS)
  - check if external packages are well set to YES before compiling
- compile mfem application (main.cpp) (pb avec petsc sundials)

<= back to main page

Chapter 3

Code overview

<= back to main page

Chapter 4

How to build of a phase-field application?

<= back to main page

Chapter 5

Diffusion problems

5.1 1D Problem

The distance between (x_1, y_1) and (x_2, y_2) is $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

<= back to main page

Chapter 6

Allen-Cahn problems

6.1 1D Problem

<= back to main page

Chapter 7

Cahn-Hilliard problems

7.1 1D Problem

<= back to main page

Chapter 8

SpatialDiscretization

8.1 Description

Chapter 9

Variables

Chapter 10

Namespace Index

10.1 Namespace List

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

Chapter 11

Hierarchical Index

11.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 12

Data Structure Index

12.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 13

Namespace Documentation

13.1 MassDefaultConstant Namespace Reference

Default constant used by Mass Solver.

Variables

- const auto **iter_max** = 30
- const auto **abs_tol** = 1.e-15
- const auto **rel_tol** = 1.e-15
- const bool **iterative_mode** = false
- const auto **print_level** = -1

13.1.1 Detailed Description

Default constant used by Mass Solver.

13.2 NewtonDefaultConstant Namespace Reference

Defaukt constant used by Newton Solver.

Variables

- const auto **iter_max** = 100
- const auto **abs_tol** = 1.e-15
- const auto **rel_tol** = 1.e-15
- const bool **iterative_mode** = false
- const auto **print_level** = 1

13.2.1 Detailed Description

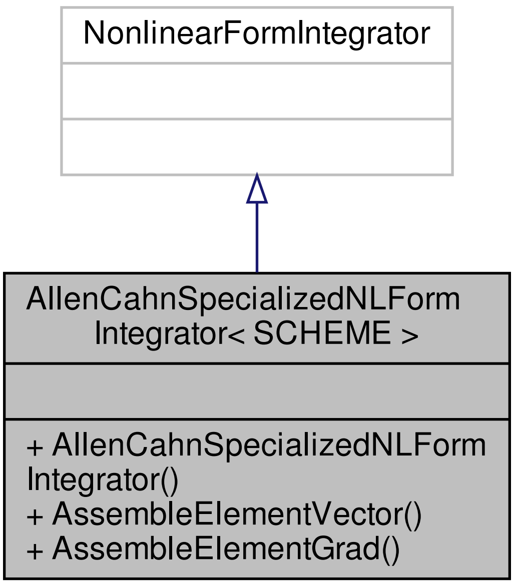
Defaukt constant used by Newton Solver.

Chapter 14

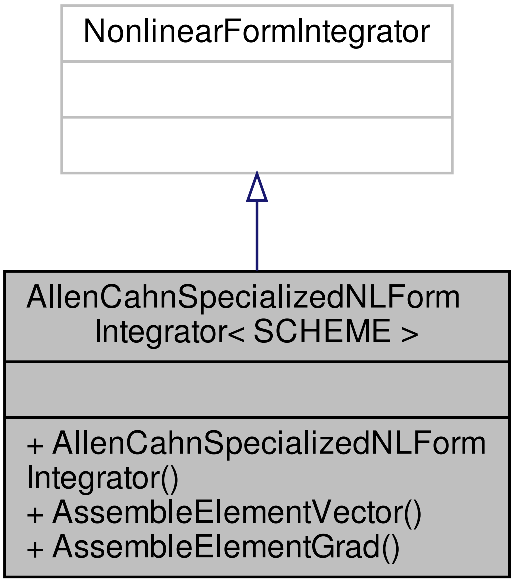
Data Structure Documentation

14.1 AllenCahnSpecializedNLFormIntegrator< SCHEME > Class Template Reference

Inheritance diagram for AllenCahnSpecializedNLFormIntegrator< SCHEME >:



Collaboration diagram for AllenCahnSpecializedNLFormIntegrator< SCHEME >:



Public Member Functions

- AllenCahnSpecializedNLFormIntegrator (const mfem::GridFunction &_u_old, const double &_omega, const double &_lambda, const double &_alpha, MobilityCoefficient _mob)

  Construct a new Allen Cahn Specialized N L Form Integrator:: Allen Cahn Specialized N L Form Integrator object.
- virtual void AssembleElementVector (const mfem::FiniteElement &el, mfem::ElementTransformation &Tr, const mfem::Vector &elfun, mfem::Vector &elvect)

  Residual part of the non linear problem.
- virtual void AssembleElementGrad (const mfem::FiniteElement &el, mfem::ElementTransformation &Tr, const mfem::Vector &elfun, mfem::DenseMatrix &elmat)

  Jacobian part of the non linear problem.

14.1.1 Detailed Description

template<ThermodynamicsPotentialDiscretization SCHEME>
class AllenCahnSpecializedNLFormIntegrator< SCHEME >

Definition at line 20 of file AllenCahnSpecializedNLFormIntegrator.hpp.

14.1.2 Constructor & Destructor Documentation

14.1.2.1 AllenCahnSpecializedNLFormIntegrator()

```
template<ThermodynamicsPotentialDiscretization SCHEME>
AllenCahnSpecializedNLFormIntegrator< SCHEME >::AllenCahnSpecializedNLFormIntegrator (
            const mfem::GridFunction & _u_old,
            const double & _omega,
            const double & _lambda,
            const double & _alpha,
            MobilityCoefficient _mob )
```

Construct a new Allen Cahn Specialized N L Form Integrator:: Allen Cahn Specialized N L Form Integrator object.

Template Parameters

SCHEME	

Parameters

_u_old	
_omega	
_lambda	
_alpha	
_mob	
_w_scheme	
_h_scheme	

Definition at line 67 of file AllenCahnSpecializedNLFormIntegrator.hpp.

```
70     : u_old(_u_old), omega(_omega), lambda(_lambda), alpha(_alpha), mob(_mob) {}
```

14.1.3 Member Function Documentation

14.1.3.1 AssembleElementGrad()

```
template<ThermodynamicsPotentialDiscretization SCHEME>
void AllenCahnSpecializedNLFormIntegrator< SCHEME >::AssembleElementGrad (
            const mfem::FiniteElement & el,
            mfem::ElementTransformation & Tr,
            const mfem::Vector & elfun,
            mfem::DenseMatrix & elmat ) [virtual]
```

Jacobian part of the non linear problem.

Template Parameters

SCHEME	

Parameters

el	
Tr	
elfun	
elmat	

Definition at line 144 of file AllenCahnSpecializedNLFormIntegrator.hpp.

References PotentialFunctions< ORDER, SCHEME >::getPotentialFunction().

```
                            {
  int nd = el.GetDof();
  int dim = el.GetDim();
  int spaceDim = Tr.GetSpaceDim();
  bool square = (dim == spaceDim);
  double w;

  shape.SetSize(nd);
  dshape.SetSize(nd, dim);
  dshapedxt.SetSize(nd, spaceDim);
  elmat.SetSize(nd);

  const mfem::IntegrationRule* ir =
      &mfem::IntRules.Get(el.GetGeomType(), 2 * el.GetOrder() + Tr.OrderW());

  elmat = 0.0;
  for (int i = 0; i < ir->GetNPoints(); i++) {
    const mfem::IntegrationPoint& ip = ir->IntPoint(i);
    el.CalcDShape(ip, dshape);  // dphi
    const auto u = elfun * shape;
    const auto un = u_old.GetValue(Tr, ip);

    const auto W = this->second_derivative_potential_.getPotentialFunction("W", un);
    const auto H = this->second_derivative_potential_.getPotentialFunction("H", un);
    const auto Wsecond = W(u);
    const auto Hsecond = H(u);
    const auto Mphi = mob.Eval(Tr, ip);

    Tr.SetIntPoint(&ip);
    w = Tr.Weight();  // det(J)
    // std::cout << " SQUARE  ? " << square << std::endl;
    w = ip.weight / (square ? w : w * w * w);
    // AdjugateJacobian = / adj(J),         if J is square
    //                    \ adj(J^t.J).J^t, otherwise

    // Tr.AdjugateJacobian() det(J)J-1

    // w = w* Mphi * lambda
    w *= Mphi * this->lambda;

    // dshapedxt =  det(J)J-1 dshape
    Mult(dshape, Tr.AdjugateJacobian(), dshapedxt);
    // elmat += w * dshapedxt * dshapedxt^T
    AddMult_a_AAt(w, dshapedxt, elmat);

    //  (this->omega * secondDerivativedoubleWellPotential(elfun * shape) +
    //   this->alpha * secondDerivativeInterpolationPotential(elfun * shape)) *
    // Compute w'(u)*(du,v), v is shape function
    double fun_val =
        Mphi * (this->omega * Wsecond + this->alpha * Hsecond) * ip.weight * Tr.Weight();  // w'(u)
    // elmat += fun_val * shape * shape^T
    AddMult_a_VVt(fun_val, shape, elmat);  // w'(u)*(du, v)
  }
}
```

14.1.3.2 AssembleElementVector()

```
template<ThermodynamicsPotentialDiscretization SCHEME>
void AllenCahnSpecializedNLFormIntegrator< SCHEME >::AssembleElementVector (
```

```
            const mfem::FiniteElement & el,
            mfem::ElementTransformation & Tr,
            const mfem::Vector & elfun,
            mfem::Vector & elvect )  [virtual]
```

Residual part of the non linear problem.

Template Parameters

SCHEME	

Parameters

el	
Tr	
elfun	
elvect	

Definition at line 82 of file AllenCahnSpecializedNLFormIntegrator.hpp.

References PotentialFunctions< ORDER, SCHEME >::getPotentialFunction().

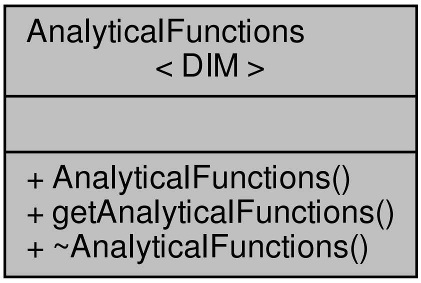
```
                            {
    int nd = el.GetDof();
    int dim = el.GetDim();
    int spaceDim = Tr.GetSpaceDim();
    dshape.SetSize(nd, dim);
    shape.SetSize(nd);
    invdfdx.SetSize(dim, spaceDim);
    vec.SetSize(dim);
    pointflux.SetSize(spaceDim);

    elvect.SetSize(nd);
    const mfem::IntegrationRule* ir =
        &mfem::IntRules.Get(el.GetGeomType(), 2 * el.GetOrder() + Tr.OrderW());

    elvect = 0.0;
    for (int i = 0; i < ir->GetNPoints(); i++) {
       const mfem::IntegrationPoint& ip = ir->IntPoint(i);
       el.CalcDShape(ip, dshape);  // dphi
       el.CalcShape(ip, shape);    // phi
       Tr.SetIntPoint(&ip);

       const auto u = elfun * shape;
       const auto un = u_old.GetValue(Tr, ip);

       const auto W = this->first_derivative_potential_.getPotentialFunction("W", un);
       const auto H = this->first_derivative_potential_.getPotentialFunction("H", un);
       const auto Wprime = W(u);
       const auto Hprime = H(u);
       const auto Mphi = mob.Eval(Tr, ip);

       CalcAdjugate(Tr.Jacobian(), invdfdx);  // invdfdx = adj(J)

       dshape.MultTranspose(elfun, vec);
       invdfdx.MultTranspose(vec, pointflux);

       const auto fun_val = Mphi * (this->omega * Wprime + this->alpha * Hprime);

       // Given phi, compute (w'(phi)-f, v), v is shape function
       const double ww = ip.weight * Tr.Weight() * fun_val;
       add(elvect, ww, shape, elvect);

       // Laplacian : given u, compute (grad(u), grad(v)), v is shape function.
       double w;
       w = Mphi * ip.weight * this->lambda / Tr.Weight();
       pointflux *= w;
       invdfdx.Mult(pointflux, vec);
       dshape.AddMult(vec, elvect);
     }
  }
```

The documentation for this class was generated from the following file:

- AllenCahnSpecializedNLFormIntegrator.hpp

14.2 AnalyticalFunctions< DIM > Class Template Reference

Collaboration diagram for AnalyticalFunctions< DIM >:



Public Member Functions

- AnalyticalFunctions ()

    Construct a new analytical function:: analytical function object.
- template<class... Args>
  std::function< double(const mfem::Vector &)> getAnalyticalFunctions (const std::string &analytical_↩
  function_name, Args... args)

    return the function associated with the analytical_function_name
- ∼AnalyticalFunctions ()

    Destroy the analytical function :: analytical function object.

14.2.1 Detailed Description

template<int DIM>
class AnalyticalFunctions< DIM >

Definition at line 21 of file Utils/AnalyticalFunctions.hpp.

14.2.2 Member Function Documentation

14.2.2.1 getAnalyticalFunctions()

```
template<int DIM>
template<class... Args>
std::function< double(const mfem::Vector &)> AnalyticalFunctions< DIM >::getAnalytical←
Functions (
            const std::string & analytical_function_name,
            Args... args )
```

return the function associated with the analytical_function_name

Parameters

analytical_function_name	

Returns

const double

Definition at line 232 of file Utils/AnalyticalFunctions.hpp.

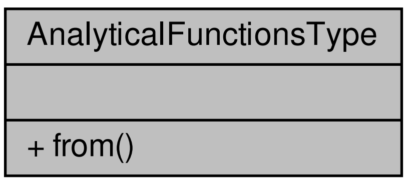
```
                                                              {
  switch (AnalyticalFunctionsType::from(analytical_function_name)) {
    case AnalyticalFunctionsType::Heaviside:
      return this->getHeaviside(args...);
    case AnalyticalFunctionsType::HyperbolicTangent:
      return this->getHyperbolicTangent(args...);
    case AnalyticalFunctionsType::Uniform:
      return this->getUniform(args...);
    default:
      throw std::runtime_error(
          "AnalyticalFunctions::getAnalyticalFunctions: Heaviside, HyperbolicTangent and Uniform "
          "analytical function  are available");
      break;
  }
}
```

The documentation for this class was generated from the following file:

- Utils/AnalyticalFunctions.hpp

14.3 AnalyticalFunctionsType Struct Reference

Collaboration diagram for AnalyticalFunctionsType:



Public Types

- enum **value** { **Heaviside**, **HyperbolicTangent**, **Uniform** }

Static Public Member Functions

- static value **from** (const std::string &)

14.3.1 Detailed Description

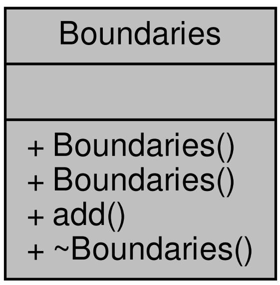
Definition at line 67 of file Utils/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

- Utils/PhaseFieldOptions.hpp

14.4 Boundaries Class Reference

Collaboration diagram for Boundaries:



Public Member Functions

- template<class... Args>
  Boundaries (const Args &...args)
- void **add** (const Boundary &boundary)

14.4.1 Detailed Description

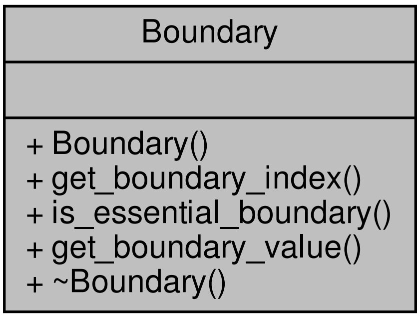
Definition at line 206 of file BoundaryConditions.hpp.

The documentation for this class was generated from the following file:

- BoundaryConditions.hpp

14.5 Boundary Class Reference

Collaboration diagram for Boundary:



Public Member Functions

- Boundary (const std::string &boundary_name, const int &boundary_index, const std::string &boundary_type, const double &boundary_value)

  Construct a new Boundary:: Boundary object.
- int get_boundary_index () const

  return the index associated to the boundary
- bool is_essential_boundary () const

  flag to identify essential boundary
- double get_boundary_value () const

  return the double value prescribed on boundary
- ∼Boundary ()

  Destroy the Boundary:: Boundary object.

14.5.1 Detailed Description

Definition at line 23 of file BoundaryConditions.hpp.

14.5.2 Constructor & Destructor Documentation

14.5.2.1 Boundary()

```
Boundary::Boundary (
            const std::string & boundary_name,
            const int & boundary_index,
            const std::string & boundary_type,
            const double & boundary_value )
```

Construct a new Boundary:: Boundary object.

Parameters

boundary_name	
boundary_index	
boundary_type	
boundary_value	

Definition at line 49 of file BoundaryConditions.hpp.

```
    : boundary_name_(boundary_name),
      boundary_index_(boundary_index),
      boundary_value_(boundary_value) {
  switch (BoundaryConditionType::from(boundary_type)) {
    case BoundaryConditionType::Dirichlet:
      this->is_essential_boundary_ = 1;
      break;
    case BoundaryConditionType::Neumann:
    case BoundaryConditionType::Periodic:
    case BoundaryConditionType::Robin:
      this->is_essential_boundary_ = 0;
      break;
    default:
      throw std::runtime_error(
          "Boundary::Boundary(): only Dirichlet, Neumann, Periodic and Robin BoundaryConditionType "
          "are available");
      break;
  }
}
```

14.5.3 Member Function Documentation

14.5.3.1 get_boundary_index()

```
int Boundary::get_boundary_index ( ) const
```

return the index associated to the boundary

Returns

int

Definition at line 76 of file BoundaryConditions.hpp.

```
{ return this->boundary_index_; }
```

14.5.3.2 get_boundary_value()

```
double Boundary::get_boundary_value ( ) const
```

return the double value prescribed on boundary

Returns

double

Definition at line 91 of file BoundaryConditions.hpp.

```
91 { return this->boundary_value_; }
```

14.5.3.3 is_essential_boundary()

```
bool Boundary::is_essential_boundary ( ) const
```

flag to identify essential boundary

Returns

true
false

Definition at line 84 of file BoundaryConditions.hpp.

```
84 { return this->is_essential_boundary_; }
```

The documentation for this class was generated from the following file:

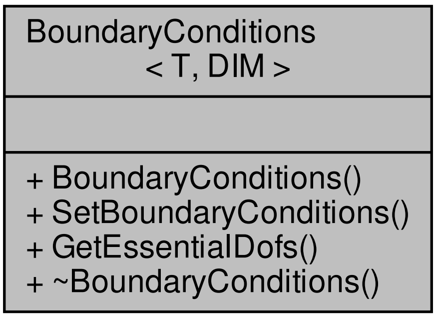
- BoundaryConditions.hpp

14.6 BoundaryConditions< T, DIM > Class Template Reference

Class used to manage boundary conditions.

```
#include </home/ci230846/home-local/MyGitProjects/COMPONENT/PF-MFEM/BCs/↩
BoundaryConditions.hpp>
```

Collaboration diagram for BoundaryConditions< T, DIM >:



Public Member Functions

- template<class... Args>
  BoundaryConditions (SpatialDiscretization< T, DIM > *spatial, const Args &...boundaries)

  Construct a new Boundary Conditions:: Boundary Conditions object.
- void SetBoundaryConditions (mfem::Vector &u)

  Set boundary conditions.
- mfem::Array< int > GetEssentialDofs ()

  return the list of essential dofs
- ∼BoundaryConditions ()

  Destroy the Boundary Conditions:: Boundary Conditions object.

14.6.1 Detailed Description

template<class T, int DIM>
class BoundaryConditions< T, DIM >

Class used to manage boundary conditions.

Definition at line 104 of file BoundaryConditions.hpp.

14.6.2 Constructor & Destructor Documentation

14.6.2.1 BoundaryConditions()

```
template<class T , int DIM>
template<class... Args>
BoundaryConditions< T, DIM >::BoundaryConditions (
            SpatialDiscretization< T, DIM > * spatial,
            const Args &... boundaries )
```

Construct a new Boundary Conditions:: Boundary Conditions object.

Template Parameters

Args	

Parameters

fespace	
mesh_max_bdr_attributes	
boundaries	

Definition at line 138 of file BoundaryConditions.hpp.

References SpatialDiscretization< T, DIM >::get_finite_element_space(), and SpatialDiscretization< T, DIM >::get_max_bdr_attributes().

```
                                                                      {
  this->fespace_ = spatial->get_finite_element_space();
  const auto &mesh_max_bdr_attributes = spatial->get_max_bdr_attributes();

  auto bdrs = std::vector<Boundary>{boundaries...};

  Dirichlet_bdr_.SetSize(mesh_max_bdr_attributes);
  Dirichlet_value_.SetSize(mesh_max_bdr_attributes);
  if (mesh_max_bdr_attributes == bdrs.size()) {
    for (const auto &bdr : bdrs) {
      const auto &id = bdr.get_boundary_index();
      if (bdr.is_essential_boundary()) {
        Dirichlet_bdr_[id] = 1;
      } else {
        Dirichlet_bdr_[id] = 0;
      }
      Dirichlet_value_[id] = bdr.get_boundary_value();
    }
    this->fespace_->GetEssentialTrueDofs(this->Dirichlet_bdr_, this->ess_tdof_list_);

  } else {
    throw std::runtime_error(
        "BoundaryConditions::BoundaryConditions(): the number of user-defined boundaries is "
        "different from the total number of boundaries associated to the mesh ");
  }
}
```

14.6.3 Member Function Documentation

14.6.3.1 GetEssentialDofs()

```
template<class T , int DIM>
mfem::Array< int > BoundaryConditions< T, DIM >::GetEssentialDofs ( )
```

return the list of essential dofs

Returns

mfem::Array<int> array of essential dofs

Definition at line 172 of file BoundaryConditions.hpp.

```
172                                                         {
173   return this->ess_tdof_list_;
174 }
```

14.6.3.2 SetBoundaryConditions()

```
template<class T , int DIM>
void BoundaryConditions< T, DIM >::SetBoundaryConditions (
             mfem::Vector & u )
```

Set boundary conditions.

Parameters

u	unknown vector

Definition at line 182 of file BoundaryConditions.hpp.

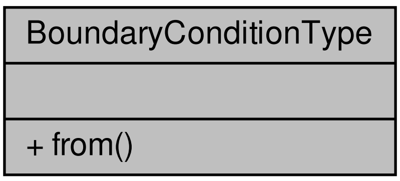
```
182                                                                     {
183   mfem::Array<int> tmp_array_bdr(this->Dirichlet_bdr_.Size());
184   for (auto i = 0; i < this->Dirichlet_bdr_.Size(); i++) {
185     tmp_array_bdr = 0;
186     mfem::Array<int> dof;
187     if (this->Dirichlet_bdr_[i] > 0) {
188       tmp_array_bdr[i] = 1;
189       this->fespace_->GetEssentialTrueDofs(tmp_array_bdr, dof);
190       u.SetSubVector(dof, this->Dirichlet_value_[i]);
191     }
192   }
193 }  // end of SetBoundaryConditions
```

The documentation for this class was generated from the following file:

- BoundaryConditions.hpp

14.7 BoundaryConditionType Struct Reference

Collaboration diagram for BoundaryConditionType:



Public Types

- enum **value** { **Dirichlet**, **Neumann**, **Periodic**, **Robin** }

Static Public Member Functions

- static value **from** (const std::string &)

14.7.1 Detailed Description

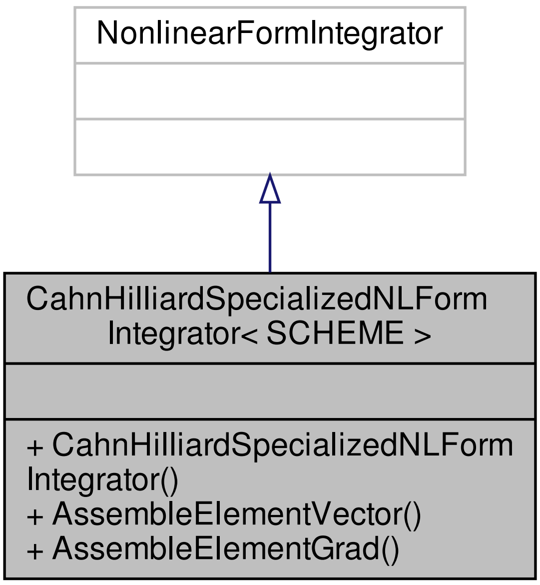
Definition at line 102 of file Utils/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

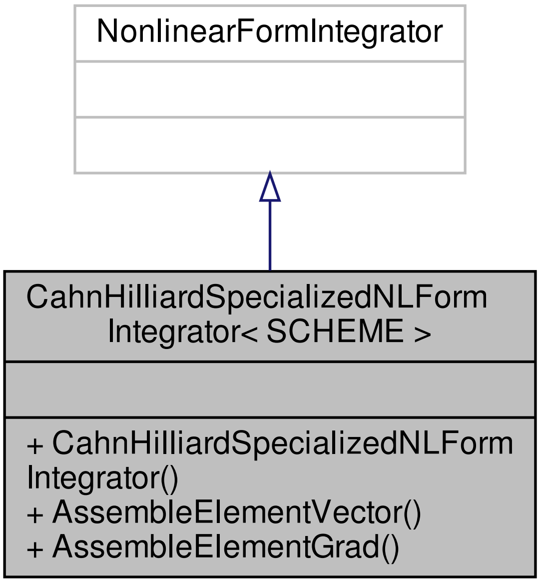
- Utils/PhaseFieldOptions.hpp

14.8 CahnHilliardSpecializedNLFormIntegrator< SCHEME > Class Template Reference

Inheritance diagram for CahnHilliardSpecializedNLFormIntegrator< SCHEME >:



Collaboration diagram for CahnHilliardSpecializedNLFormIntegrator< SCHEME >:



Public Member Functions

- CahnHilliardSpecializedNLFormIntegrator (const mfem::GridFunction &_u_old, const double &_omega, const double &_lambda, const double &_alpha, MobilityCoefficient _mob)

  Construct a new Cahn Hilliard Specialized N L Form Integrator< S C H E M E>:: Cahn Hilliard Specialized N L Form Integrator object.
- virtual void AssembleElementVector (const mfem::FiniteElement &el, mfem::ElementTransformation &Tr, const mfem::Vector &elfun, mfem::Vector &elvect)

  Residual part of the non linear problem.
- virtual void AssembleElementGrad (const mfem::FiniteElement &el, mfem::ElementTransformation &Tr, const mfem::Vector &elfun, mfem::DenseMatrix &elmat)

  Jacobian part of the non linear problem.

14.8.1 Detailed Description

template<ThermodynamicsPotentialDiscretization SCHEME>
class CahnHilliardSpecializedNLFormIntegrator< SCHEME >

Definition at line 18 of file CahnHilliardSpecializedNLFormIntegrator.hpp.

14.8.2 Constructor & Destructor Documentation

14.8.2.1 CahnHilliardSpecializedNLFormIntegrator()

```
template<ThermodynamicsPotentialDiscretization SCHEME>
CahnHilliardSpecializedNLFormIntegrator< SCHEME >::CahnHilliardSpecializedNLFormIntegrator (
            const mfem::GridFunction & _u_old,
            const double & _omega,
            const double & _lambda,
            const double & _alpha,
            MobilityCoefficient _mob )
```

Construct a new Cahn Hilliard Specialized N L Form Integrator< S C H E M E>:: Cahn Hilliard Specialized N L Form Integrator object.

Template Parameters

SCHEME	

Parameters

_u_old	
_omega	
_lambda	
_alpha	
_mob	

Definition at line 63 of file CahnHilliardSpecializedNLFormIntegrator.hpp.

```
66     : u_old(_u_old), omega(_omega), lambda(_lambda), alpha(_alpha), mob(_mob) {}
```

14.8.3 Member Function Documentation

14.8.3.1 AssembleElementGrad()

```
template<ThermodynamicsPotentialDiscretization SCHEME>
void CahnHilliardSpecializedNLFormIntegrator< SCHEME >::AssembleElementGrad (
            const mfem::FiniteElement & el,
            mfem::ElementTransformation & Tr,
            const mfem::Vector & elfun,
            mfem::DenseMatrix & elmat ) [virtual]
```

Jacobian part of the non linear problem.

Template Parameters

SCHEME	

Parameters

el	
Tr	
elfun	
elmat	

Definition at line 140 of file CahnHilliardSpecializedNLFormIntegrator.hpp.

References PotentialFunctions< ORDER, SCHEME >::getPotentialFunction().

```
                              {
  int nd = el.GetDof();
  int dim = el.GetDim();
  int spaceDim = Tr.GetSpaceDim();
  bool square = (dim == spaceDim);
  double w;

  shape.SetSize(nd);
  dshape.SetSize(nd, dim);
  dshapedxt.SetSize(nd, spaceDim);
  elmat.SetSize(nd);

  const mfem::IntegrationRule* ir =
      &mfem::IntRules.Get(el.GetGeomType(), 2 * el.GetOrder() + Tr.OrderW());

  elmat = 0.0;
  for (int i = 0; i < ir->GetNPoints(); i++) {
    const mfem::IntegrationPoint& ip = ir->IntPoint(i);
    el.CalcDShape(ip, dshape);  // dphi
    const auto u = elfun * shape;
    const auto un = u_old.GetValue(Tr, ip);
    const auto W = this->second_derivative_potential_.getPotentialFunction("W", un);
    const auto H = this->second_derivative_potential_.getPotentialFunction("H", un);
    const auto Wsecond = W(u);
    const auto Hsecond = H(u);
    const auto Mphi = mob.Eval(Tr, ip);

    Tr.SetIntPoint(&ip);
    w = Tr.Weight();  // det(J)
    // std::cout << " SQUARE  ? " << square << std::endl;
    w = ip.weight / (square ? w : w * w * w);
    // AdjugateJacobian = / adj(J),         if J is square
    //                    \ adj(J^t.J).J^t, otherwise

    // Tr.AdjugateJacobian() det(J)J-1

    // w = w* Mphi * lambda
    w *= Mphi * this->lambda;

    // dshapedxt =  det(J)J-1 dshape
    Mult(dshape, Tr.AdjugateJacobian(), dshapedxt);
    // elmat += w * dshapedxt * dshapedxt^T
    AddMult_a_AAt(w, dshapedxt, elmat);

    //  (this->omega * secondDerivativedoubleWellPotential(elfun * shape) +
    //   this->alpha * secondDerivativeInterpolationPotential(elfun * shape)) *
    // Compute w'(u)*(du,v), v is shape function
    double fun_val =
        Mphi * (this->omega * Wsecond + this->alpha * Hsecond) * ip.weight * Tr.Weight();  // w'(u)
    // elmat += fun_val * shape * shape^T
    AddMult_a_VVt(fun_val, shape, elmat);  // w'(u)*(du, v)
  }
}
```

14.8.3.2 AssembleElementVector()

template<ThermodynamicsPotentialDiscretization SCHEME>
void CahnHilliardSpecializedNLFormIntegrator< SCHEME >::AssembleElementVector (

```
const mfem::FiniteElement & el,
mfem::ElementTransformation & Tr,
const mfem::Vector & elfun,
mfem::Vector & elvect )  [virtual]
```

Residual part of the non linear problem.

Template Parameters

SCHEME	

Parameters

el	
Tr	
elfun	
elvect	

Definition at line 78 of file CahnHilliardSpecializedNLFormIntegrator.hpp.

References PotentialFunctions< ORDER, SCHEME >::getPotentialFunction().

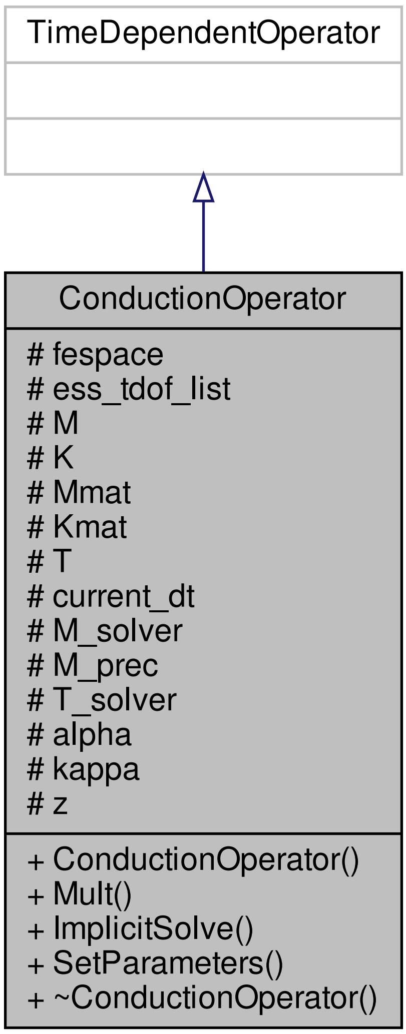
```
                            {
  int nd = el.GetDof();
  int dim = el.GetDim();
  int spaceDim = Tr.GetSpaceDim();
  dshape.SetSize(nd, dim);
  shape.SetSize(nd);
  invdfdx.SetSize(dim, spaceDim);
  vec.SetSize(dim);
  pointflux.SetSize(spaceDim);

  elvect.SetSize(nd);
  const mfem::IntegrationRule* ir =
      &mfem::IntRules.Get(el.GetGeomType(), 2 * el.GetOrder() + Tr.OrderW());

  elvect = 0.0;
  for (int i = 0; i < ir->GetNPoints(); i++) {
    const mfem::IntegrationPoint& ip = ir->IntPoint(i);
    el.CalcDShape(ip, dshape);  // dphi
    el.CalcShape(ip, shape);    // phi
    Tr.SetIntPoint(&ip);

     const auto u = elfun * shape;
     const auto un = u_old.GetValue(Tr, ip);

     const auto W = this->first_derivative_potential_.getPotentialFunction("W", un);
     const auto H = this->first_derivative_potential_.getPotentialFunction("H", un);
     const auto Wprime = W(u);
     const auto Hprime = H(u);
     const auto Mphi = mob.Eval(Tr, ip);

     CalcAdjugate(Tr.Jacobian(), invdfdx);  // invdfdx = adj(J)

     dshape.MultTranspose(elfun, vec);
     invdfdx.MultTranspose(vec, pointflux);

     const auto fun_val = Mphi * (this->omega * Wprime + this->alpha * Hprime);

     // Given phi, compute (w'(phi)-f, v), v is shape function
     const double ww = ip.weight * Tr.Weight() * fun_val;
     add(elvect, ww, shape, elvect);

     // Laplacian : given u, compute (grad(u), grad(v)), v is shape function.
     double w;
     w = Mphi * ip.weight * this->lambda / Tr.Weight();
     pointflux *= w;
     invdfdx.Mult(pointflux, vec);
     dshape.AddMult(vec, elvect);
   }
 }
```

The documentation for this class was generated from the following file:

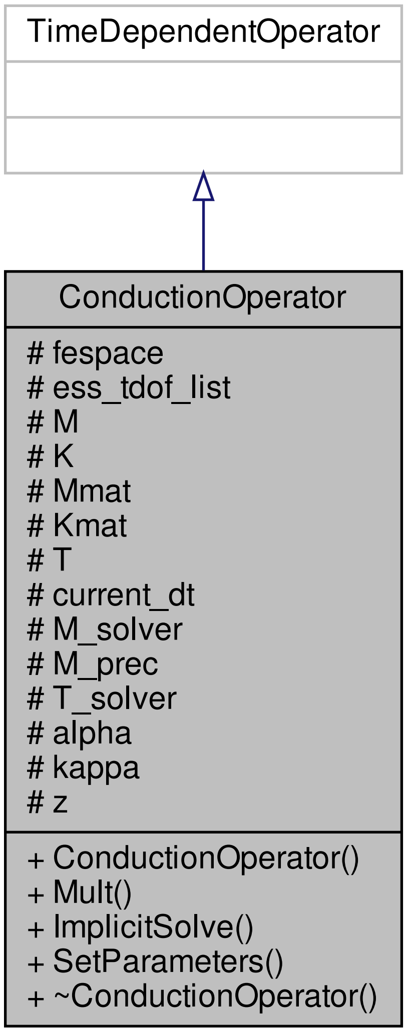
- CahnHilliardSpecializedNLFormIntegrator.hpp

14.9 ConductionOperator Class Reference

Inheritance diagram for ConductionOperator:



Collaboration diagram for ConductionOperator:



Public Member Functions

- **ConductionOperator** (mfem::FiniteElementSpace &f, double alpha, double kappa, mfem::Vector &u)
- virtual void **Mult** (const mfem::Vector &u, mfem::Vector &du_dt) const
- virtual void ImplicitSolve (const double dt, const mfem::Vector &u, mfem::Vector &k)
- void SetParameters (const mfem::Vector &u)

  Update the diffusion BilinearForm K using the given true-dof vector u.

Protected Attributes

- mfem::FiniteElementSpace & **fespace**
- mfem::Array< int > **ess_tdof_list**

- mfem::BilinearForm ∗ **M**
- mfem::BilinearForm ∗ **K**
- mfem::SparseMatrix **Mmat**
- mfem::SparseMatrix **Kmat**
- mfem::SparseMatrix ∗ **T**
- double **current_dt**
- mfem::CGSolver **M_solver**
- mfem::DSmoother **M_prec**
- mfem::UMFPackSolver **T_solver**
- double **alpha**
- double **kappa**
- mfem::Vector **z**

14.9.1 Detailed Description

Definition at line 19 of file ConductionOperator.hpp.

14.9.2 Member Function Documentation

14.9.2.1 ImplicitSolve()

```
void ConductionOperator::ImplicitSolve (
            const double dt,
            const mfem::Vector & u,
            mfem::Vector & k )  [virtual]
```

Solve the Backward-Euler equation: k = f(u + dt∗k, t), for the unknown k. This is the only requirement for high-order SDIRK implicit integration.

Definition at line 137 of file ConductionOperator.hpp.

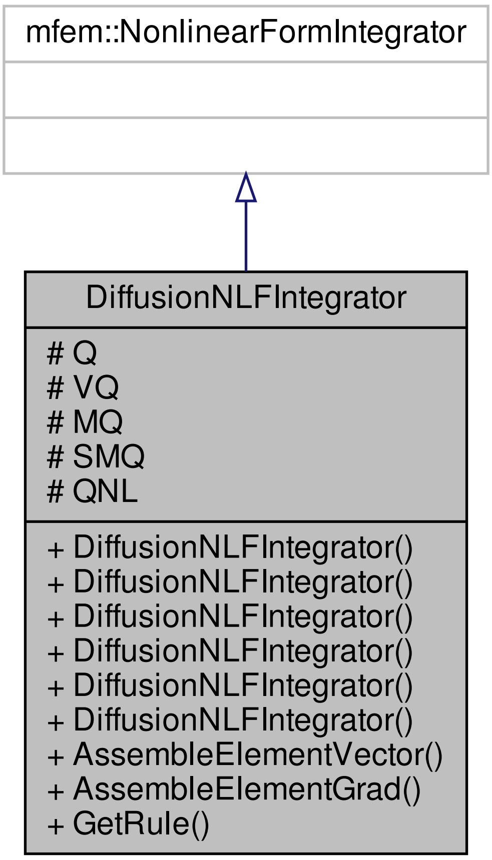
```
                                                          {
  // Solve the equation:
  //    du_dt = M^{-1}*[-K(u + dt*du_dt)]
  // for du_dt, where K is linearized by using u from the previous timestep
  if (!T) {
    T = Add(1.0, Mmat, dt, Kmat);
    current_dt = dt;
    T_solver.SetOperator(*T);
  }
  MFEM_VERIFY(dt == current_dt, "");  // SDIRK methods use the same dt
  Kmat.Mult(u, z);
  z.Neg();

  T_solver.Mult(z, du_dt);
  du_dt.SetSubVector(ess_tdof_list, 0.0);
}
```

The documentation for this class was generated from the following file:

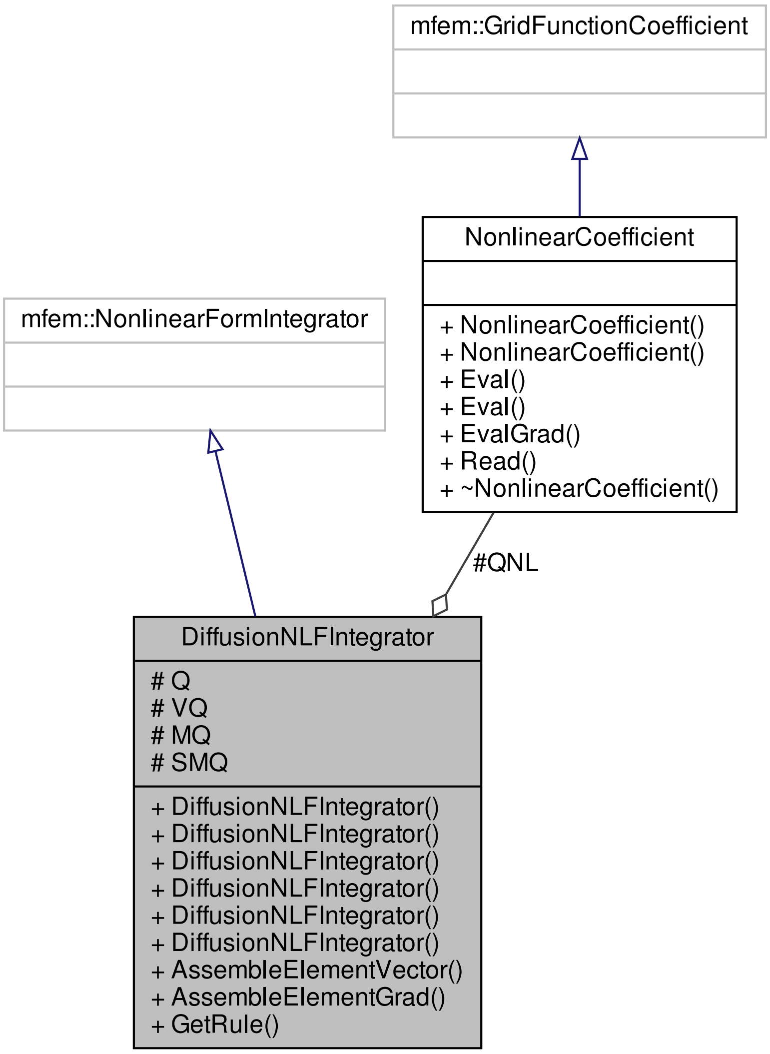
- ConductionOperator.hpp

14.10 DiffusionNLFIntegrator Class Reference

Inheritance diagram for DiffusionNLFIntegrator:



Collaboration diagram for DiffusionNLFIntegrator:



Public Member Functions

- DiffusionNLFIntegrator ()

  Construct a diffusion nonliner integrator with coefficient Q = 1.
- DiffusionNLFIntegrator (mfem::ConstantCoefficient &q)
- DiffusionNLFIntegrator (mfem::ConstantCoefficient &q, NonlinearCoefficient &qq)
- DiffusionNLFIntegrator (mfem::VectorCoefficient &q, NonlinearCoefficient &qq)
- DiffusionNLFIntegrator (mfem::MatrixCoefficient &q, NonlinearCoefficient &qq)
- DiffusionNLFIntegrator (mfem::SymmetricMatrixCoefficient &q, NonlinearCoefficient &qq)
- virtual void AssembleElementVector (const mfem::FiniteElement &el, mfem::ElementTransformation &Tr, const mfem::Vector &elfun, mfem::Vector &elvect)

  Given a elfun values perform the local action of the NonlinearFormIntegrator.
- virtual void AssembleElementGrad (const mfem::FiniteElement &el, mfem::ElementTransformation &Tr, const mfem::Vector &elfun, mfem::DenseMatrix &elmat)

  Assemble the local gradient matrix.

Static Public Member Functions

- static const mfem::IntegrationRule & **GetRule** (const mfem::FiniteElement &fe, mfem::Element↩ Transformation &T)

Protected Attributes

- mfem::ConstantCoefficient * **Q**
- mfem::VectorCoefficient * **VQ**
- mfem::MatrixCoefficient * **MQ**
- mfem::SymmetricMatrixCoefficient * **SMQ**
- NonlinearCoefficient * **QNL**

14.10.1 Detailed Description

Definition at line 48 of file DiffusionNLFIntegrator.hpp.

14.10.2 Constructor & Destructor Documentation

14.10.2.1 DiffusionNLFIntegrator() [1/5]

```
DiffusionNLFIntegrator::DiffusionNLFIntegrator (
            mfem::ConstantCoefficient & q )  [inline]
```

Construct a diffusion integrator with a scalar coefficient q and nonlinear coefficient qq

Definition at line 69 of file DiffusionNLFIntegrator.hpp.

```
70      : Q(&q), VQ(NULL), MQ(NULL), SMQ(NULL), QNL(NULL) {}
```

14.10.2.2 DiffusionNLFIntegrator() [2/5]

```
DiffusionNLFIntegrator::DiffusionNLFIntegrator (
            mfem::ConstantCoefficient & q,
            NonlinearCoefficient & qq )  [inline]
```

Construct a diffusion integrator with a scalar coefficient q and nonlinear coefficient qq

Definition at line 74 of file DiffusionNLFIntegrator.hpp.

```
75      : Q(&q), VQ(NULL), MQ(NULL), SMQ(NULL), QNL(&qq) {}
```

14.10.2.3 DiffusionNLFIntegrator() [3/5]

```
DiffusionNLFIntegrator::DiffusionNLFIntegrator (
            mfem::VectorCoefficient & q,
            NonlinearCoefficient & qq )  [inline]
```

Construct a diffusion integrator with a vector coefficient q and nonlinear coefficient qq

Definition at line 79 of file DiffusionNLFIntegrator.hpp.

```
80       : Q(NULL), VQ(&q), MQ(NULL), SMQ(NULL), QNL(&qq) {}
```

14.10.2.4 DiffusionNLFIntegrator() [4/5]

```
DiffusionNLFIntegrator::DiffusionNLFIntegrator (
            mfem::MatrixCoefficient & q,
            NonlinearCoefficient & qq )  [inline]
```

Construct a diffusion integrator with a matrix coefficient q and nonlinear coefficient qq

Definition at line 84 of file DiffusionNLFIntegrator.hpp.

```
85       : Q(NULL), VQ(NULL), MQ(&q), SMQ(NULL), QNL(&qq) {}
```

14.10.2.5 DiffusionNLFIntegrator() [5/5]

```
DiffusionNLFIntegrator::DiffusionNLFIntegrator (
            mfem::SymmetricMatrixCoefficient & q,
            NonlinearCoefficient & qq )  [inline]
```

Construct a diffusion integrator with a symmetric matrix coefficient q and nonlinear coefficient qq

Definition at line 89 of file DiffusionNLFIntegrator.hpp.

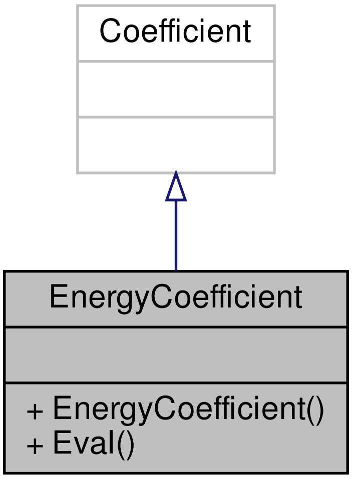
```
90       : Q(NULL), VQ(NULL), MQ(NULL), SMQ(&q), QNL(&qq) {}
```

The documentation for this class was generated from the following file:

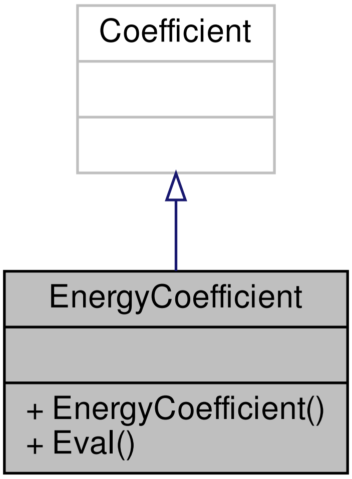
- DiffusionNLFIntegrator.hpp

14.11 EnergyCoefficient Class Reference

Inheritance diagram for EnergyCoefficient:



Collaboration diagram for EnergyCoefficient:



Public Member Functions

- **EnergyCoefficient** (mfem::GridFunction ∗gfu_, const double &lambda_, const double &omega_)
- double **Eval** (mfem::ElementTransformation &T, const mfem::IntegrationPoint &ip)

14.11.1 Detailed Description

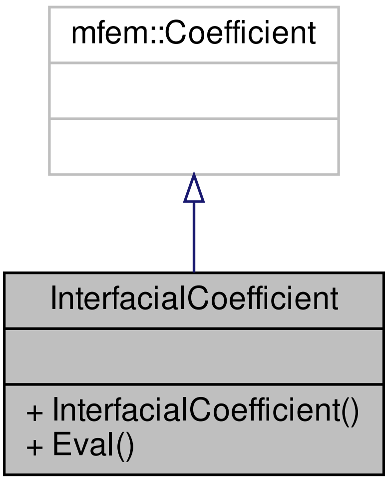
Definition at line 37 of file EnergyCoefficient.hpp.

The documentation for this class was generated from the following file:

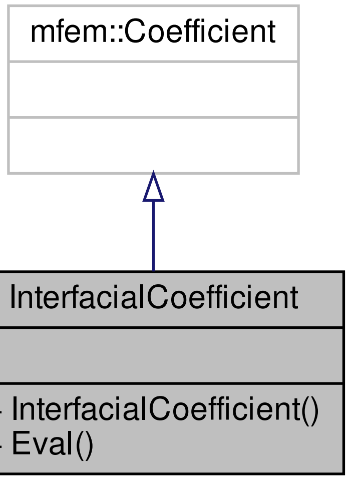
- EnergyCoefficient.hpp

14.12 InterfacialCoefficient Class Reference

Inheritance diagram for InterfacialCoefficient:



Collaboration diagram for InterfacialCoefficient:



Public Member Functions

- **InterfacialCoefficient** (mfem::GridFunction ∗gfu_, const double &lambda_)
- double **Eval** (mfem::ElementTransformation &T, const mfem::IntegrationPoint &ip)

14.12.1 Detailed Description

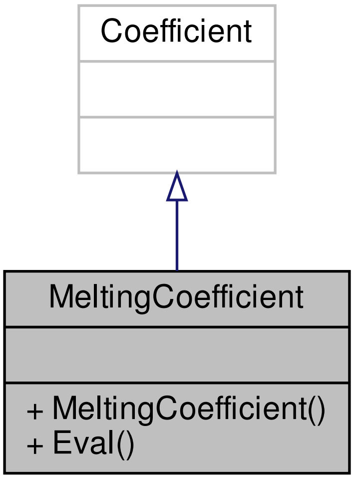
Definition at line 15 of file EnergyCoefficient.hpp.

The documentation for this class was generated from the following file:

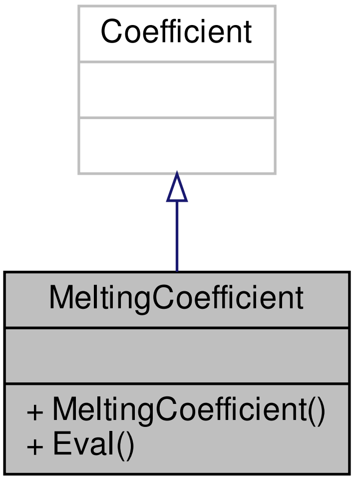
- EnergyCoefficient.hpp

14.13 MeltingCoefficient Class Reference

Inheritance diagram for MeltingCoefficient:



Collaboration diagram for MeltingCoefficient:



Public Member Functions

- **MeltingCoefficient** (mfem::GridFunction ∗gfu_, const double &dh_)
- double **Eval** (mfem::ElementTransformation &T, const mfem::IntegrationPoint &ip)

14.13.1 Detailed Description

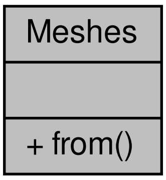
Definition at line 62 of file EnergyCoefficient.hpp.

The documentation for this class was generated from the following file:

- EnergyCoefficient.hpp

14.14 Meshes Struct Reference

Collaboration diagram for Meshes:



Public Types

- enum **value** {
  InlineLineWithSegments, **InlineSquareWithTriangles**, **InlineSquareWithQuadrangles**, **InlineSquare←WithTetraedres**,
  InlineSquareWithHexaedres, **GMSH** }

Static Public Member Functions

- static value **from** (const std::string &)

14.14.1 Detailed Description

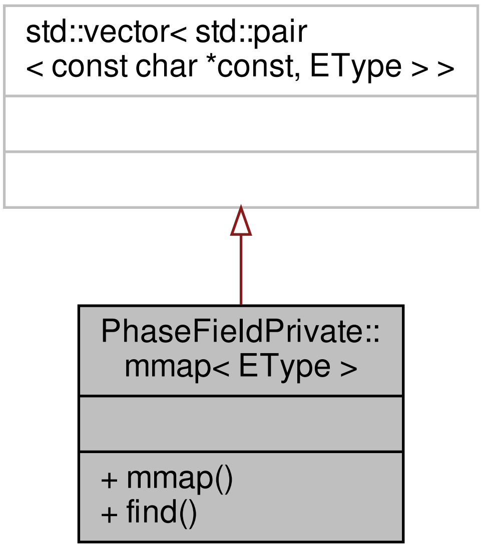
Definition at line 87 of file Utils/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

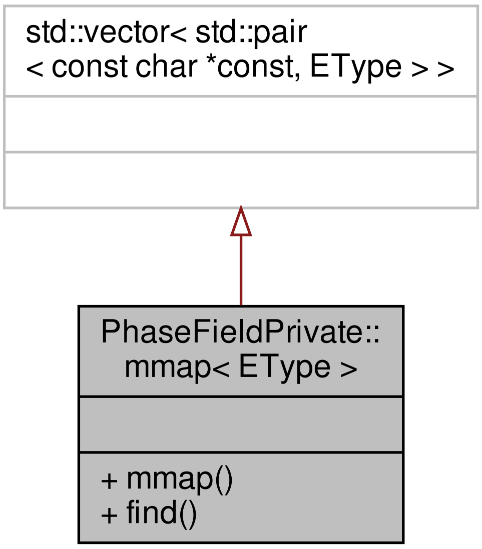
- Utils/PhaseFieldOptions.hpp

14.15 PhaseFieldPrivate::mmap< EType > Struct Template Reference

Inheritance diagram for PhaseFieldPrivate::mmap< EType >:



Collaboration diagram for PhaseFieldPrivate::mmap< EType >:



Public Types

- using **mpair** = std::pair< const char ∗const, EType >

Public Member Functions

- **mmap** (const std::initializer_list< mpair > &)
- EType **find** (const char ∗const, const std::string &)

14.15.1 Detailed Description

template<typename EType>
struct PhaseFieldPrivate::mmap< EType >

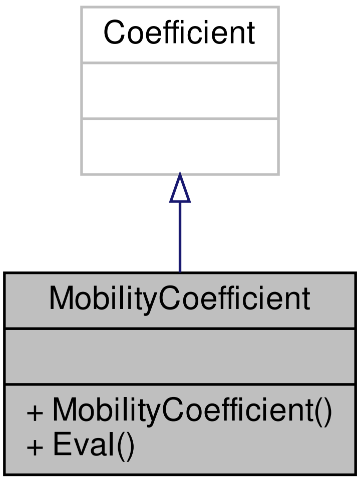
Definition at line 21 of file Utils/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

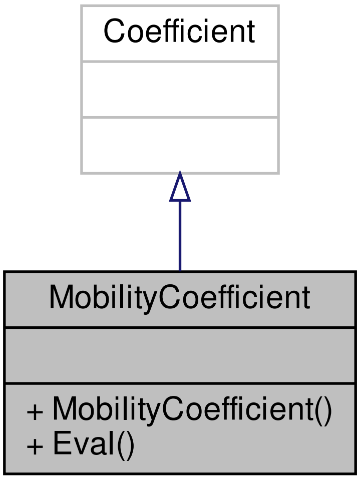
- Utils/PhaseFieldOptions.hpp

14.16 MobilityCoefficient Class Reference

Inheritance diagram for MobilityCoefficient:



Collaboration diagram for MobilityCoefficient:



Public Member Functions

- **MobilityCoefficient** (mfem::GridFunction mob_gf, const double &mob_c, const int &order)
- double **Eval** (mfem::ElementTransformation &T, const mfem::IntegrationPoint &ip)

14.16.1 Detailed Description

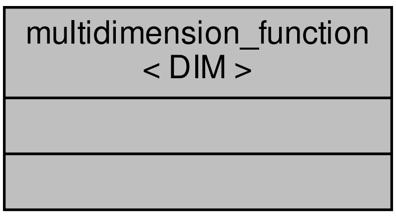
Definition at line 12 of file MobilityCoefficient.hpp.

The documentation for this class was generated from the following file:

- MobilityCoefficient.hpp

14.17 multidimension_function< DIM > Struct Template Reference

Collaboration diagram for multidimension_function< DIM >:



14.17.1 Detailed Description

template<int DIM>
struct multidimension_function< DIM >

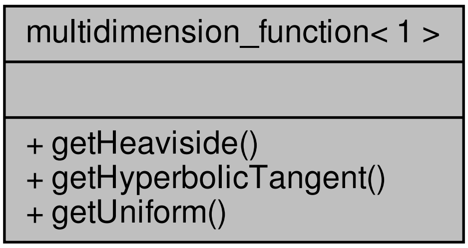
Definition at line 18 of file Utils/AnalyticalFunctions.hpp.

The documentation for this struct was generated from the following file:

- Utils/AnalyticalFunctions.hpp

14.18 multidimension_function< 1 > Struct Template Reference

Collaboration diagram for multidimension_function< 1 >:



Public Member Functions

- template<typename... Args>
  std::function< double(const mfem::Vector &)> **getHeaviside** (Args... args)
- template<typename... Args>
  std::function< double(const mfem::Vector &)> **getHyperbolicTangent** (Args... args)
- template<typename... Args>
  std::function< double(const mfem::Vector &)> **getUniform** (Args... args)

14.18.1 Detailed Description

template<>
struct multidimension_function< 1 >

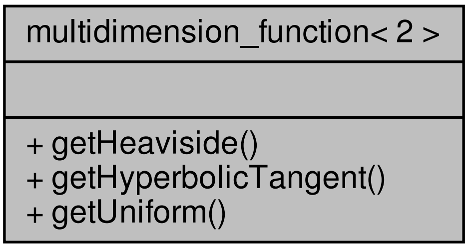
Definition at line 55 of file Utils/AnalyticalFunctions.hpp.

The documentation for this struct was generated from the following file:

- Utils/AnalyticalFunctions.hpp

14.19 multidimension_function< 2 > Struct Template Reference

Collaboration diagram for multidimension_function< 2 >:



Public Member Functions

- template<typename... Args>
  std::function< double(const mfem::Vector &)> **getHeaviside** (Args... args)
- template<typename... Args>
  std::function< double(const mfem::Vector &)> **getHyperbolicTangent** (Args... args)
- template<typename... Args>
  std::function< double(const mfem::Vector &)> **getUniform** (Args... args)

14.19.1 Detailed Description

template<>
struct multidimension_function< 2 >

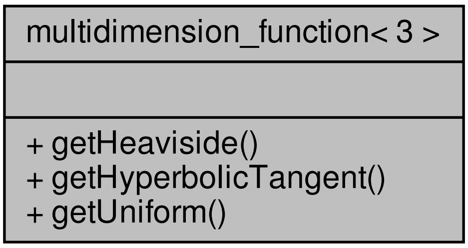
Definition at line 103 of file Utils/AnalyticalFunctions.hpp.

The documentation for this struct was generated from the following file:

- Utils/AnalyticalFunctions.hpp

14.20 multidimension_function< 3 > Struct Template Reference

Collaboration diagram for multidimension_function< 3 >:



Public Member Functions

- template<typename... Args>
  std::function< double(const mfem::Vector &)> **getHeaviside** (Args... args)
- template<typename... Args>
  std::function< double(const mfem::Vector &)> **getHyperbolicTangent** (Args... args)
- template<typename... Args>
  std::function< double(const mfem::Vector &)> **getUniform** (Args... args)

14.20.1 Detailed Description

template<>
struct multidimension_function< 3 >

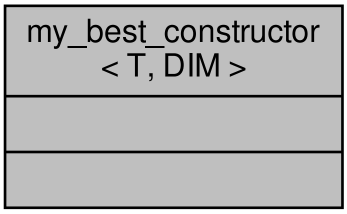
Definition at line 154 of file Utils/AnalyticalFunctions.hpp.

The documentation for this struct was generated from the following file:

- Utils/AnalyticalFunctions.hpp

14.21 my_best_constructor< T, DIM > Struct Template Reference

Collaboration diagram for my_best_constructor< T, DIM >:



14.21.1 Detailed Description

template<class T, int DIM>
struct my_best_constructor< T, DIM >

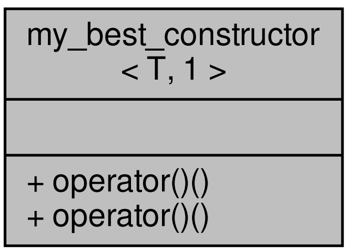
Definition at line 27 of file Spatial.hpp.

The documentation for this struct was generated from the following file:

- Spatial.hpp

14.22 my_best_constructor< T, 1 > Struct Template Reference

Collaboration diagram for my_best_constructor< T, 1 >:



Public Member Functions

- template<typename... Args>
  void **operator()** (SpatialDiscretization< T, 1 > &a_my_class, const std::string &mesh_type, const int &fe_order, const std::string &file)
- template<typename... Args>
  void **operator()** (SpatialDiscretization< T, 1 > &a_my_class, const std::string &mesh_type, const int &fe_order, std::tuple< Args... > tup_args)

14.22.1 Detailed Description

template<typename T>
struct my_best_constructor< T, 1 >

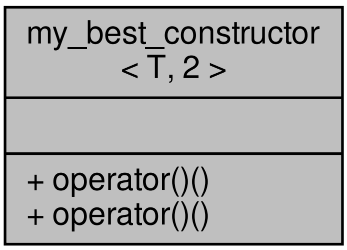
Definition at line 76 of file Spatial.hpp.

The documentation for this struct was generated from the following file:

- Spatial.hpp

14.23 my_best_constructor< T, 2 > Struct Template Reference

Collaboration diagram for my_best_constructor< T, 2 >:



Public Member Functions

- template<typename... Args>
  void **operator()** (SpatialDiscretization< T, 2 > &a_my_class, const std::string &mesh_type, const int &fe_order, const std::string &file)
- template<typename... Args>
  void **operator()** (SpatialDiscretization< T, 2 > &a_my_class, const std::string &mesh_type, const int &fe_order, std::tuple< Args... > tup_args)

14.23.1 Detailed Description

template<typename T>
struct my_best_constructor< T, 2 >

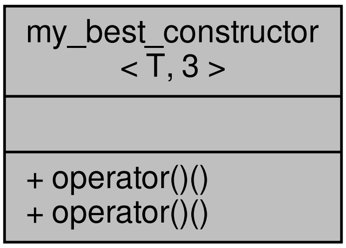
Definition at line 141 of file Spatial.hpp.

The documentation for this struct was generated from the following file:

- Spatial.hpp

14.24 my_best_constructor< T, 3 > Struct Template Reference

Collaboration diagram for my_best_constructor< T, 3 >:



Public Member Functions

- template<typename... Args>
  void **operator()** (SpatialDiscretization< T, 3 > &a_my_class, const std::string &mesh_type, const int &fe_order, const std::string &file)
- template<typename... Args>
  void **operator()** (SpatialDiscretization< T, 3 > &a_my_class, const std::string &mesh_type, const int &fe_order, std::tuple< Args... > tup_args)

14.24.1 Detailed Description

template<typename T>
struct my_best_constructor< T, 3 >

Definition at line 206 of file Spatial.hpp.

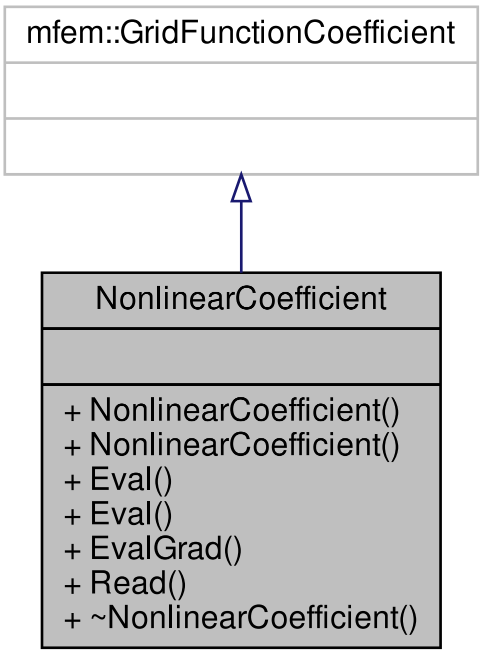
The documentation for this struct was generated from the following file:

- Spatial.hpp

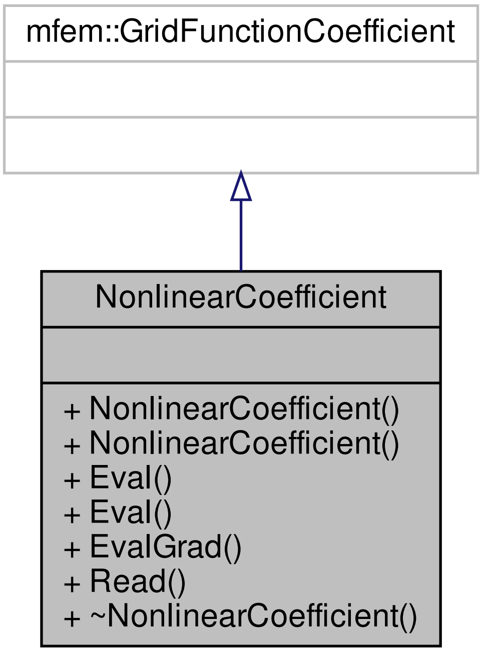
14.25 NonlinearCoefficient Class Reference

```
#include </home/ci230846/home-local/MyGitProjects/COMPONENT/PF-MFEM/Integrators/↩
DiffusionNLFIntegrator.hpp>
```

Inheritance diagram for NonlinearCoefficient:



Collaboration diagram for NonlinearCoefficient:



Public Member Functions

- **NonlinearCoefficient** (double rho_, double bt_, double p0)
- **NonlinearCoefficient** (mfem::GridFunction ∗u_, double rho_, double bt_, double p0)
- virtual double **Eval** (mfem::ElementTransformation &T, const mfem::IntegrationPoint &ip)
- virtual double **Eval** (mfem::ElementTransformation &T, const mfem::IntegrationPoint &ip, const double &u)
- virtual double **EvalGrad** (mfem::ElementTransformation &T, const mfem::IntegrationPoint &ip, const double &u)
- virtual void **Read** (std::istream &in)

14.25.1 Detailed Description

Function representing a nonlinear coefficient (density) for the given state (pressure). Used in DiffusionNLFIntegrator::AssembleElementGrad.

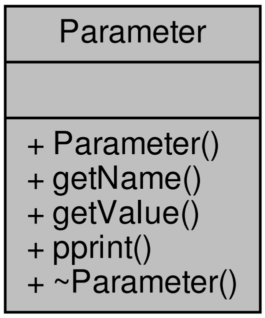
Definition at line 13 of file DiffusionNLFIntegrator.hpp.

The documentation for this class was generated from the following file:

- DiffusionNLFIntegrator.hpp

14.26 Parameter Class Reference

Collaboration diagram for Parameter:



Public Member Functions

- **Parameter** (std::string name, var value)
- std::string getName () const
- var getValue () const
- void **pprint** ()

14.26.1 Detailed Description

Definition at line 18 of file Parameter.hpp.

14.26.2 Member Function Documentation

14.26.2.1 getName()

```
std::string Parameter::getName ( ) const  [inline]
```

Method used to get the name of the parameter return name of the parameter of type string

Definition at line 31 of file Parameter.hpp.

```
31 { return name; }  // end of getName
```

14.26.2.2 getValue()

```
var Parameter::getValue ( ) const  [inline]
```

Method used to get the value of the parameter return value of the parameter of any type (see variant)

Definition at line 36 of file Parameter.hpp.

```
36 { return value; }  // end of getValue
```

The documentation for this class was generated from the following file:

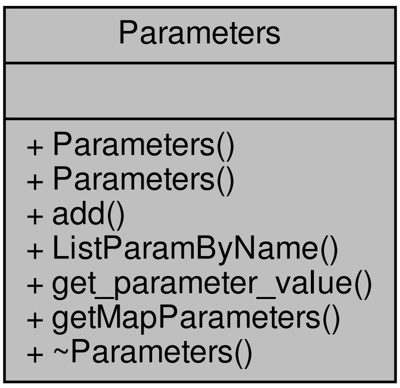
- Parameter.hpp

14.27 Parameters Class Reference

Class used to manage a list of Parameter.

```
#include </home/ci230846/home-local/MyGitProjects/COMPONENT/PF-MFEM/Parameters/↩
Parameters.hpp>
```

Collaboration diagram for Parameters:



Public Member Functions

- Parameters ()

  Construct a new Parameters:: Parameters object.
- template<class... Args>
  Parameters (const Args &... args)

  Construct a new Parameters:: Parameters object.
- void add (const Parameter &param)

  add a new parameters
- void **ListParamByName** ()
- double get_parameter_value (const std::string &name) const

  get double value of a parameter by name
- std::map< std::string, double > getMapParameters () const

  transform list of parameters into a map<string,double>
- ∼Parameters ()

  Destroy the Parameters:: Parameters object.

14.27.1 Detailed Description

Class used to manage a list of Parameter.

Definition at line 23 of file Parameters.hpp.

14.27.2 Constructor & Destructor Documentation

14.27.2.1 Parameters()

```
template<class... Args>
Parameters::Parameters (
            const Args &... args ) [explicit]
```

Construct a new Parameters:: Parameters object.

Template Parameters

Args	

Parameters

args	

Definition at line 52 of file Parameters.hpp.

```
52                                              {
53   this->vect_params_ = std::vector<Parameter>{args...};
54 }
```

14.27.3 Member Function Documentation

14.27.3.1 add()

```
void Parameters::add (
            const Parameter & param )
```

add a new parameters

Parameters

param	parameter to add

Definition at line 61 of file Parameters.hpp.

```
61 { this->vect_params_.emplace_back(param); }
```

14.27.3.2 get_parameter_value()

```
double Parameters::get_parameter_value (
              const std::string & name ) const
```

get double value of a parameter by name

Parameters

name	name of the parameter

Returns

double double value of the parameter

Definition at line 69 of file Parameters.hpp.

Referenced by PhaseFieldOperator< T, DIM >::PhaseFieldOperator(), and TimeDiscretization< T, DC, DIM >::↩ TimeDiscretization().

```
69                                                                     {
70   const auto lowest_float = std::numeric_limits<float>::lowest();
71   auto value = std::numeric_limits<double>::lowest();
72 
73   for (const auto& p : this->vect_params_) {
74     auto pn = p.getName();
75     if (pn == name) {
76       value = std::get<double>(p.getValue());
77     }
78   }
79   if (value > lowest_float) {
80     return value;
81   } else {
82     throw std::runtime_error("Parameter " + name + " not found");
83   }
84 }  // end of getValueByName
```

14.27.3.3 getMapParameters()

```
std::map< std::string, double > Parameters::getMapParameters ( ) const
```

transform list of parameters into a map<string,double>

Returns

std::map<std::string, double>

Definition at line 91 of file Parameters.hpp.

```
91                                                               {
92   std::map<std::string, double> map_par;
93   for (auto p : this->vect_params_) {
94     auto name = p.getName();
95     auto value = std::get<double>(p.getValue());
96     map_par.try_emplace(name, value);
97   }
98   return map_par;
99 }
```

The documentation for this class was generated from the following file:

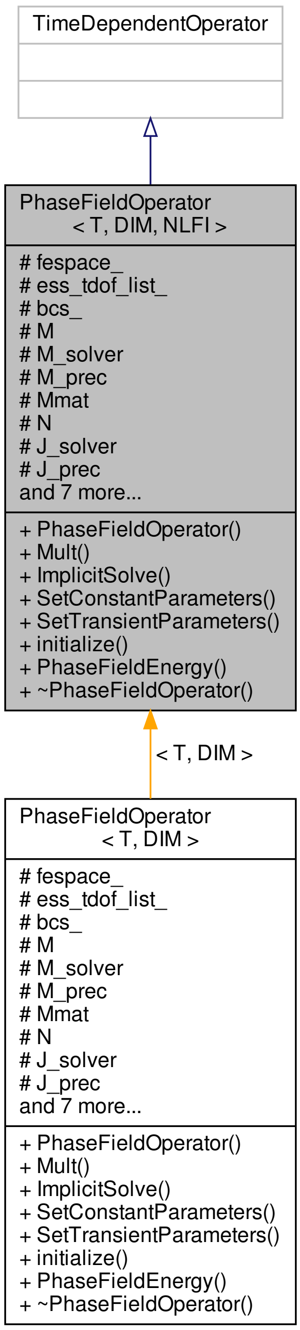
- Parameters.hpp

14.28 PhaseFieldOperator< T, DIM, NLFI > Class Template Reference

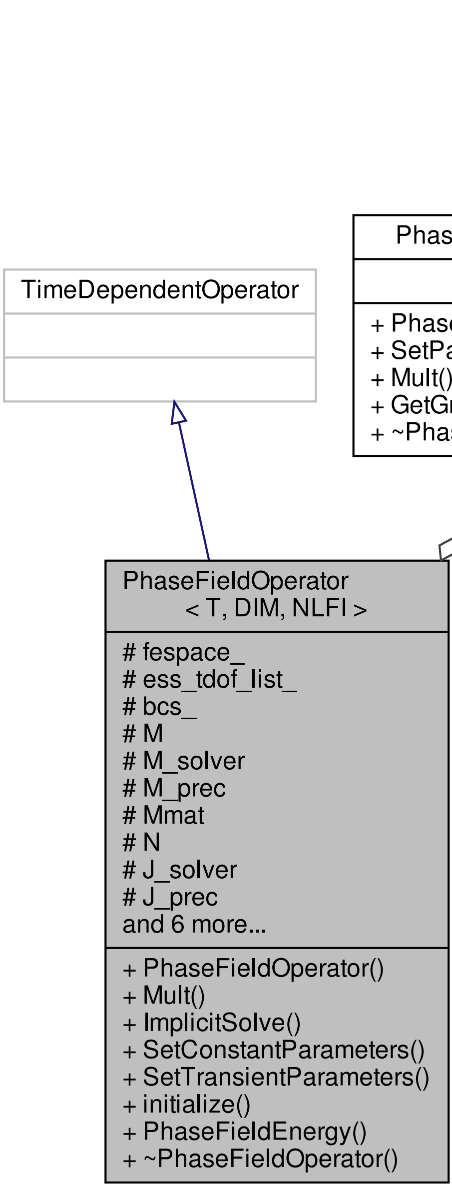
PhaseFieldOperator class.

```
#include </home/ci230846/home-local/MyGitProjects/COMPONENT/PF-MFEM/Operators/↩
PhaseFieldOperator.hpp>
```

Inheritance diagram for PhaseFieldOperator< T, DIM, NLFI >:



Collaboration diagram for PhaseFieldOperator< T, DIM, NLFI >:



Public Member Functions

- PhaseFieldOperator (SpatialDiscretization< T, DIM > ∗spatial, const Parameters &params, Variables< T, DIM > &vars)

    Construct a new Phase Field Operator:: Phase Field Operator object.
- virtual void Mult (const mfem::Vector &u, mfem::Vector &du_dt) const

    Compute the right-hand side of the ODE system.
- virtual void ImplicitSolve (const double dt, const mfem::Vector &u, mfem::Vector &k)

    Solve the Backward-Euler equation: k = f(phi + dt∗k, t), for the unknown k.
- void SetConstantParameters (const double dt, mfem::Vector &u)

Set current dt, unk values - needed to compute action and Jacobian.

- void SetTransientParameters (const double dt, const mfem::Vector &u)

  Set current dt, unk values - needed to compute action and Jacobian.
- void initialize (Variable< T, DIM > &vv)

  Initialization stage.
- const double PhaseFieldEnergy (const mfem::Vector &x) const

  Compute Phase-field Energy.
- virtual ∼PhaseFieldOperator ()

  Destroy the Phase Field Operator:: Phase Field Operator object.

Protected Attributes

- mfem::FiniteElementSpace ∗ **fespace_**
- mfem::Array< int > **ess_tdof_list_**
- BoundaryConditions< T, DIM > ∗ **bcs_**
- mfem::BilinearForm ∗ **M**
- mfem::CGSolver **M_solver**
- mfem::DSmoother **M_prec**
- mfem::SparseMatrix **Mmat**
- mfem::NonlinearForm ∗ **N**
- mfem::Solver ∗ **J_solver**
- mfem::Solver ∗ **J_prec**
- mfem::NewtonSolver **newton_solver_**
- PhaseFieldReducedOperator ∗ reduced_oper
- double **mobility_coeff_**
- double **omega_**
- double **lambda_**
- double **current_dt_**
- mfem::Vector **z**

14.28.1 Detailed Description

template<class T, int DIM, class NLFI>
class PhaseFieldOperator< T, DIM, NLFI >

PhaseFieldOperator class.

Definition at line 47 of file PhaseFieldOperator.hpp.

14.28.2 Constructor & Destructor Documentation

14.28.2.1 PhaseFieldOperator()

```
template<class T, int DIM, class NLFI >
PhaseFieldOperator< T, DIM, NLFI >::PhaseFieldOperator (
            SpatialDiscretization< T, DIM > * spatial,
            const Parameters & params,
            Variables< T, DIM > & vars )
```

Construct a new Phase Field Operator:: Phase Field Operator object.

Parameters

fespace	Finite Element space
params	list of Parameters
u	unknown vector

Definition at line 141 of file PhaseFieldOperator.hpp.

```
    : mfem::TimeDependentOperator(spatial->getSize(), 0.0),
      M(NULL),
      N(NULL),
      current_dt_(0.0),
      z(height) {
  this->fespace_ = spatial->get_finite_element_space();
  this->omega_ = params.get_parameter_value("omega");
  this->lambda_ = params.get_parameter_value("lambda");
  this->mobility_coeff_ = params.get_parameter_value("mobility");

  auto &vv = vars.get_variable("phi");
  this->initialize(vv);
  // auto u = vv.get_unknown();
  // this->ess_tdof_list_ = this->bcs_.GetEssentialDofs();
  // this->bcs_.SetBoundaryConditions(u);
  // this->SetConstantParameters(this->current_dt_, u);
  // this->SetTransientParameters(this->current_dt_, u);
  // vv.update(u);
}
```

14.28.3 Member Function Documentation

14.28.3.1 ImplicitSolve()

```
template<class T , int DIM, class NLFI >
void PhaseFieldOperator< T, DIM, NLFI >::ImplicitSolve (
            const double dt,
            const mfem::Vector & u,
            mfem::Vector & du_dt )  [virtual]
```

Solve the Backward-Euler equation: k = f(phi + dt∗k, t), for the unknown k.

Solve the Backward-Euler equation: k = f(u + dt∗k, t), for the unknown k. This is the only requirement for high-order SDIRK implicit integration.

Parameters

dt	current time step
u	unknown vector
du_dt	unkwon time derivative vector

Definition at line 280 of file PhaseFieldOperator.hpp.

```
                                                                   {
  const auto sc = height;
```

```
    mfem::Vector v(u.GetData(), sc);
    mfem::Vector dv_dt(du_dt.GetData(), sc);
    // // Solve the equation:
    // //    du_dt = M^{-1}*[-K(u + dt*du_dt)]
    // // for du_dt

    this->bcs_->SetBoundaryConditions(v);
    this->SetTransientParameters(dt, v);

    reduced_oper->SetParameters(dt, &v);

    mfem::Vector zero;  // empty vector is interpreted as zero r.h.s. by NewtonSolver
    dv_dt = v;
    dv_dt *= (1. / dt);
    this->newton_solver_.Mult(zero, dv_dt);
    dv_dt.SetSubVector(this->ess_tdof_list_, 0.0);  // pour  Dirichlet ... uniquement?
    // std::cout << " PhaseFieldOperator this->newton_solver_->Mult " << std::endl;

    MFEM_VERIFY(this->newton_solver_.GetConverged(), "Nonlinear solver did not converge.");
}
```

14.28.3.2 initialize()

```
template<class T, int DIM, class NLFI >
void PhaseFieldOperator< T, DIM, NLFI >::initialize (
            Variable< T, DIM > & vv )
```

Initialization stage.

Parameters

vv	

Definition at line 170 of file PhaseFieldOperator.hpp.

Referenced by PhaseFieldOperator< T, DIM >::PhaseFieldOperator().

```
170                                                             {
171   auto u = vv.get_unknown();
172   this->bcs_ = vv.get_boundary_conditions();
173   this->ess_tdof_list_ = this->bcs_->GetEssentialDofs();
174   this->bcs_->SetBoundaryConditions(u);
175   this->SetConstantParameters(this->current_dt_, u);
176   this->SetTransientParameters(this->current_dt_, u);
177   vv.update(u);
178 }
```

14.28.3.3 Mult()

```
template<class T , int DIM, class NLFI >
void PhaseFieldOperator< T, DIM, NLFI >::Mult (
            const mfem::Vector & u,
            mfem::Vector & du_dt ) const [virtual]
```

Compute the right-hand side of the ODE system.

Parameters

u	unknown vector
du_dt	unkwon time derivative vector

Definition at line 263 of file PhaseFieldOperator.hpp.

```
263 {
264   const auto sc = height;
265   mfem::Vector v(u.GetData(), sc);
266   mfem::Vector dv_dt(du_dt.GetData(), sc);
267   N->Mult(v, z);
268   z.Neg();  // z = -z
269   M_solver.Mult(z, dv_dt);
270 }
```

14.28.3.4 PhaseFieldEnergy()

```
template<class T , int DIM, class NLFI >
const double PhaseFieldOperator< T, DIM, NLFI >::PhaseFieldEnergy (
              const mfem::Vector & u ) const
```

Compute Phase-field Energy.

Parameters

u	unknown vector

Returns

const double

Definition at line 311 of file PhaseFieldOperator.hpp.

```
311 {
312   mfem::GridFunction un_gf(this->fespace_);
313   un_gf.SetFromTrueDofs(u);
314   mfem::GridFunction un(this->fespace_);
315   un.SetFromTrueDofs(u);
316
317   MobilityCoefficient mob(un_gf, this->mobility_coeff_, 0);
318
319   auto energy = 0.;
320   mfem::FunctionCoefficient coeff([](const mfem::Vector &x) { return 1.; });
321   mfem::FunctionCoefficient zero([](const mfem::Vector &x) { return 0.; });
322
323   std::cout << "L'intégrale de coeff sur le domaine est : " << energy << std::endl;
324   // Création d'un objet GridFunction
325   mfem::GridFunction gf(this->fespace_);
326   gf.ProjectCoefficient(coeff);
327
328   // Calcul de l'intégrale de l'objet FunctionCoefficient sur le domaine
329   energy = gf.ComputeL2Error(zero);
330
331   std::cout << "L'intégrale de coeff sur le domaine est : " << energy << std::endl;
332
333   Energy hf(this->fespace_, coeff);
334   auto nrj_test = hf.compute();
335   std::cout << "L'intégrale de nrj_test sur le domaine est : " << nrj_test << std::endl;
336
337   return energy;
338 }
```

14.28.3.5 SetConstantParameters()

```
template<class T , int DIM, class NLFI >
void PhaseFieldOperator< T, DIM, NLFI >::SetConstantParameters (
            const double dt,
            mfem::Vector & u )
```

Set current dt, unk values - needed to compute action and Jacobian.

Parameters

dt	time-step
u	unknown vector
ess_tdof_list	array of dofs

Definition at line 233 of file PhaseFieldOperator.hpp.

Referenced by PhaseFieldOperator< T, DIM >::initialize().

```
233                                                                                             {
234   delete M;
235   delete N;
236   delete reduced_oper;
237
239   // Mass matrix
241   M = new mfem::BilinearForm(this->fespace_);
242   M->AddDomainIntegrator(new mfem::MassIntegrator());
243   M->Assemble(0);
244   mfem::SparseMatrix tmp;
245   M->FormSystemMatrix(this->ess_tdof_list_, Mmat);
246   this->ut_solver_.SetSolverParameters(
247       M_solver, MassDefaultConstant::print_level, MassDefaultConstant::iterative_mode,
248       MassDefaultConstant::iter_max, MassDefaultConstant::rel_tol, MassDefaultConstant::abs_tol);
249   this->ut_solver_.BuildSolver(M_solver, M_prec, Mmat);
250 }
```

14.28.3.6 SetTransientParameters()

```
template<class T , int DIM, class NLFI >
void PhaseFieldOperator< T, DIM, NLFI >::SetTransientParameters (
            const double dt,
            const mfem::Vector & u )
```

Set current dt, unk values - needed to compute action and Jacobian.

Parameters

dt	time-step
u	unknown vector

Definition at line 191 of file PhaseFieldOperator.hpp.

Referenced by PhaseFieldOperator< T, DIM >::ImplicitSolve(), and PhaseFieldOperator< T, DIM >::initialize().

```
{
delete N;
delete reduced_oper;

// PhaseField reduced operator N
N = new mfem::NonlinearForm(this->fespace_);
mfem::GridFunction un_gf(this->fespace_);
un_gf.SetFromTrueDofs(u);
mfem::GridFunction un(this->fespace_);
un.SetFromTrueDofs(u);

MobilityCoefficient mob(un_gf, this->mobility_coeff_, 0);
// SourceCoefficient
auto dh = 0.;  // 7.e4;
N->AddDomainIntegrator(new NLFI(un, this->omega_, this->lambda_, dh, mob));
N->SetEssentialTrueDofs(this->ess_tdof_list_);

reduced_oper = new PhaseFieldReducedOperator(M, N);

// Newton Solver
this->ut_solver_.SetSolverParameters(
    this->newton_solver_, NewtonDefaultConstant::print_level,
    NewtonDefaultConstant::iterative_mode, NewtonDefaultConstant::iter_max,
    NewtonDefaultConstant::rel_tol, NewtonDefaultConstant::abs_tol);
// TODO(ci230846) : cette partie devra etre generalisee pour un solveur iteratif
J_solver = new mfem::UMFPackSolver;
this->ut_solver_.BuildSolver(this->newton_solver_, *J_solver, *
reduced_oper);
}
```

14.28.4 Field Documentation

14.28.4.1 reduced_oper

```
template<class T, int DIM, class NLFI>
PhaseFieldReducedOperator* PhaseFieldOperator< T, DIM, NLFI >::reduced_oper  [protected]
```

Nonlinear operator defining the reduced backward Euler equation for the velocity. Used in the implementation of method ImplicitSolve.

Definition at line 71 of file PhaseFieldOperator.hpp.

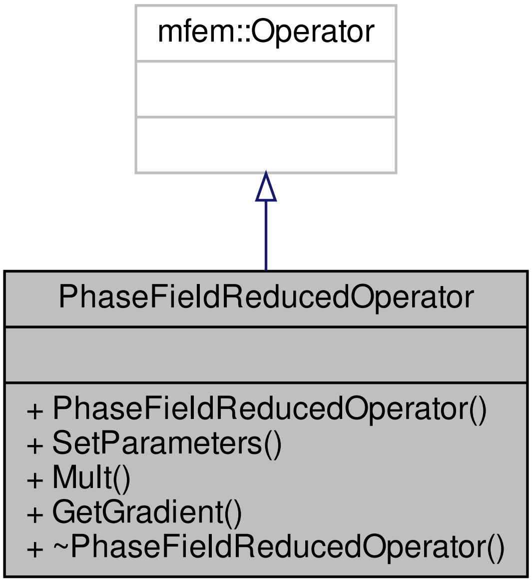
Referenced by PhaseFieldOperator< T, DIM >::ImplicitSolve(), PhaseFieldOperator< T, DIM >::SetConstantParameters(), PhaseFieldOperator< T, DIM >::SetTransientParameters(), and PhaseFieldOperator< T, DIM >::∼PhaseFieldOperator().

The documentation for this class was generated from the following file:

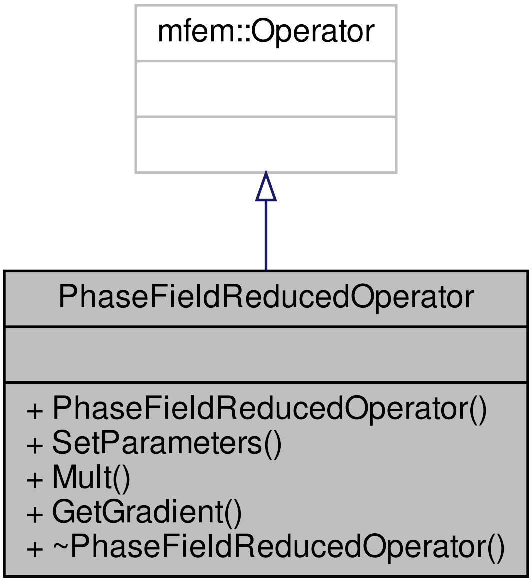
- PhaseFieldOperator.hpp

14.29 PhaseFieldReducedOperator Class Reference

Inheritance diagram for PhaseFieldReducedOperator:



Collaboration diagram for PhaseFieldReducedOperator:



Public Member Functions

- **PhaseFieldReducedOperator** (mfem::BilinearForm ∗M_, mfem::NonlinearForm ∗N_)
- void SetParameters (double dt_, const mfem::Vector ∗unk_)

  Set current dt, unk values - needed to compute action and Jacobian.
- void Mult (const mfem::Vector &k, mfem::Vector &y) const

  Compute y = N(unk + dt∗k) + M k.
- mfem::Operator & GetGradient (const mfem::Vector &k) const

  Compute y = dt∗grad_N(unk + dt∗k) + M.

14.29.1 Detailed Description

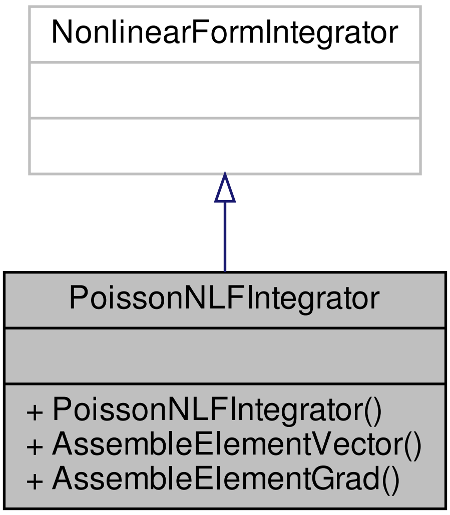
Definition at line 22 of file ReducedOperator.hpp.

The documentation for this class was generated from the following file:

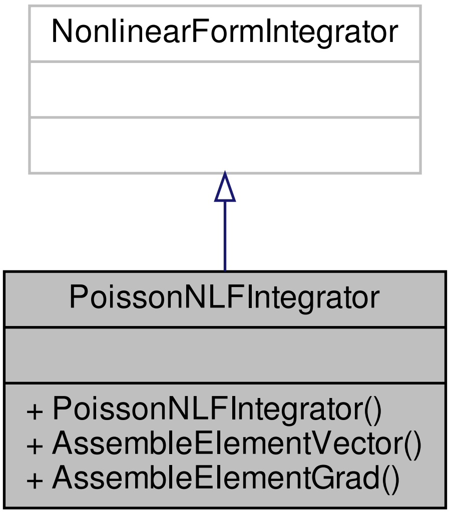
- ReducedOperator.hpp

14.30 PoissonNLFIntegrator Class Reference

Inheritance diagram for PoissonNLFIntegrator:



Collaboration diagram for PoissonNLFIntegrator:



Public Member Functions

- **PoissonNLFIntegrator** (mfem::Coefficient ∗f_)
- virtual void **AssembleElementVector** (const mfem::FiniteElement &el, mfem::ElementTransformation &Tr, const mfem::Vector &elfun, mfem::Vector &elvect)
- virtual void **AssembleElementGrad** (const mfem::FiniteElement &el, mfem::ElementTransformation &Tr, const mfem::Vector &elfun, mfem::DenseMatrix &elmat)

14.30.1 Detailed Description

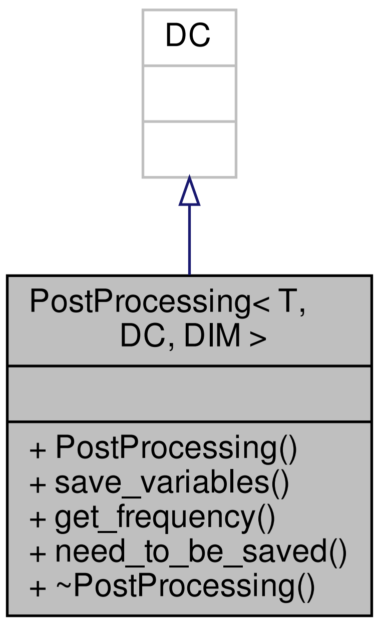
Definition at line 12 of file PoissonNLFIntegrator.hpp.

The documentation for this class was generated from the following file:

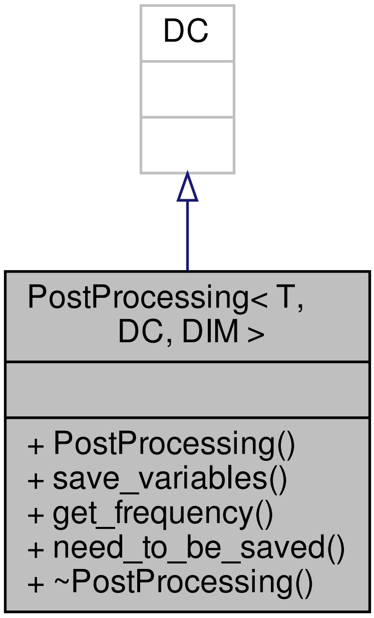
- PoissonNLFIntegrator.hpp

14.31 PostProcessing< T, DC, DIM > Class Template Reference

Inheritance diagram for PostProcessing< T, DC, DIM >:



Collaboration diagram for PostProcessing< T, DC, DIM >:



Public Member Functions

- PostProcessing (const std::string &main_folder_path, const std::string &calculation_path, SpatialDiscretization< T, DIM > ∗space, const int &frequency, const int &level_of_detail)

  Construct a new Post Processing:: Post Processing object.
- void save_variables (const Variables< T, DIM > &vars, const int &iter, const double &time)

  save variables objet at given iter/time
- int get_frequency ()

  Get the frequency of post-processing in terms of number of iterations (1 means each iteration)
- bool need_to_be_saved (const int &iteration)

  check if results have to be saved at iteration
- ∼PostProcessing ()

  Destroy the Post Processing:: Post Processing object.

14.31.1 Detailed Description

template< class T, class DC, int DIM >
class PostProcessing< T, DC, DIM >

Definition at line 24 of file postprocessing.hpp.

14.31.2 Constructor & Destructor Documentation

14.31.2.1 PostProcessing()

```
template<class T , class DC , int DIM>
PostProcessing< T, DC, DIM >::PostProcessing (
            const std::string & main_folder_path,
            const std::string & calculation_path,
            SpatialDiscretization< T, DIM > * space,
            const int & frequency,
            const int & level_of_detail )
```

Construct a new Post Processing:: Post Processing object.

Parameters

main_folder_path	
calculation_path	
mesh	
level_of_detail	

Definition at line 56 of file postprocessing.hpp.

```
60     : DC(calculation_path, &space->get_mesh()), frequency_(frequency) {
61   this->SetPrefixPath(main_folder_path);
62   this->SetLevelsOfDetail(level_of_detail);
63   this->SetDataFormat(mfem::VTKFormat::BINARY);
64   this->SetHighOrderOutput(true);
65 }
```

14.31.3 Member Function Documentation

14.31.3.1 get_frequency()

```
template<class T , class DC , int DIM>
int PostProcessing< T, DC, DIM >::get_frequency ( )
```

Get the frequency of post-processing in terms of number of iterations (1 means each iteration)

Returns

int

Definition at line 93 of file postprocessing.hpp.

```
93                                           {
94   return this->frequency_;
95 }
```

14.31.3.2 need_to_be_saved()

```
template<class T , class DC , int DIM>
bool PostProcessing< T, DC, DIM >::need_to_be_saved (
            const int & iteration )
```

check if results have to be saved at iteration

Parameters

iteration	

Returns

true
false

Definition at line 105 of file postprocessing.hpp.

```
105                                                                   {
106   bool check = (iteration % this->frequency_ == 0);
107   return check;
108 }
```

14.31.3.3 save_variables()

```
template<class T , class DC , int DIM>
void PostProcessing< T, DC, DIM >::save_variables (
            const Variables< T, DIM > & vars,
            const int & iter,
            const double & time )
```

save variables objet at given iter/time

Parameters

vars	
iter	
time	

Definition at line 75 of file postprocessing.hpp.

References Variables< T, DIM >::get_map_gridfunction().

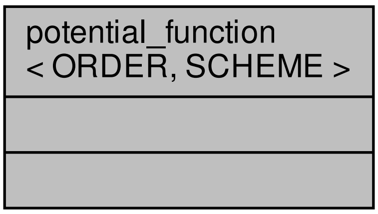
```
76                                                                 {
77   this->SetCycle(iter);
78   this->SetTime(time);
79   auto map_var = vars.get_map_gridfunction();
80   for (auto [name, gf] : map_var) {
81     this->RegisterField(name, &gf);
82     this->Save();
83   }
84 }
```

The documentation for this class was generated from the following file:

- postprocessing.hpp

14.32 potential_function< ORDER, SCHEME > Struct Template Reference

Collaboration diagram for potential_function< ORDER, SCHEME >:



14.32.1 Detailed Description

template<int ORDER, ThermodynamicsPotentialDiscretization SCHEME>
struct potential_function< ORDER, SCHEME >

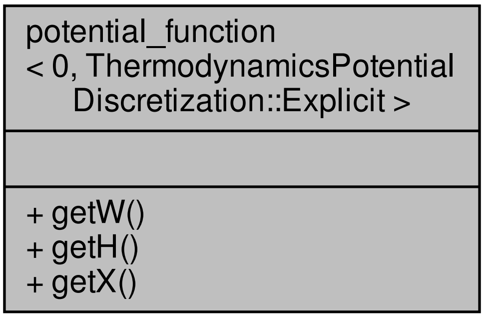
Definition at line 20 of file PhaseFieldPotentials.hpp.

The documentation for this struct was generated from the following file:

- PhaseFieldPotentials.hpp

14.33 potential_function< 0, ThermodynamicsPotentialDiscretization::Explicit > Struct Template Reference

Collaboration diagram for potential_function< 0, ThermodynamicsPotentialDiscretization::Explicit >:



Public Member Functions

- template<typename... Args>
  std::function< double(const double &)> getW (Args... args)

  Double Well potential $W(x)=x^2*(1-x)^2$
- template<typename... Args>
  std::function< double(const double &)> getH (Args... args)

  Interpolation potential $H(x)=x^3*(6x^2-15x+10)$
- template<typename... Args>
  std::function< double(const double &)> getX (Args... args)

  Identity potential $X(x)=x$.

14.33.1 Detailed Description

template<>
struct potential_function< 0, ThermodynamicsPotentialDiscretization::Explicit >

Definition at line 207 of file PhaseFieldPotentials.hpp.

14.33.2 Member Function Documentation

14.33.2.1 getH()

```
template<typename...  Args>
std::function<double(const double&)> potential_function< 0, ThermodynamicsPotentialDiscretization←
::Explicit >::getH (
            Args...  args )  [inline]
```

Interpolation potential $H(x)=x^3*(6x^2-15x+10)$

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 230 of file PhaseFieldPotentials.hpp.

```
230                                                           {
231     return std::function<double(const double&)>([](double x) {
232       const auto pot = x * x * x * (6.0 * x * x - 15.0 * x + 10.0);
233       return pot;
234     });
235   }
```

14.33.2.2 getW()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 0, ThermodynamicsPotentialDiscretization↩
::Explicit >::getW (
               Args... args ) [inline]
```

Double Well potential $W(x)=x^2 * (1-x)^2$

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 216 of file PhaseFieldPotentials.hpp.

```
216                                                           {
217     return std::function<double(const double&)>([](double x) {
218       const auto pot = x * x * (1.0 - x) * (1.0 - x);
219       return pot;
220     });
221   }
```

14.33.2.3 getX()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 0, ThermodynamicsPotentialDiscretization↩
::Explicit >::getX (
            Args... args ) [inline]
```

Identity potential X(x)=x.

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 244 of file PhaseFieldPotentials.hpp.

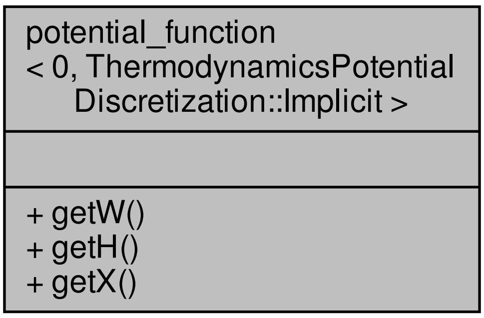
```
244                                                        {
245     return std::function<double(const double&)>([](double x) {
246       const auto pot = x;
247       return pot;
248     });
249   }
```

The documentation for this struct was generated from the following file:

- PhaseFieldPotentials.hpp

14.34 potential_function< 0, ThermodynamicsPotentialDiscretization::Implicit > Struct Template Reference

Collaboration diagram for potential_function< 0, ThermodynamicsPotentialDiscretization::Implicit >:



Public Member Functions

- template<typename... Args>
  std::function< double(const double &)> getW (Args... args)

  Double Well potential $W(x)=x^2 * (1-x)^2$
- template<typename... Args>
  std::function< double(const double &)> getH (Args... args)

  Interpolation potential $H(x)=x^3 * (6x^2-15x+10)$
- template<typename... Args>
  std::function< double(const double &)> getX (Args... args)

  Identity potential X(x)=x.

14.34.1 Detailed Description

template<>
struct potential_function< 0, ThermodynamicsPotentialDiscretization::Implicit >

Definition at line 60 of file PhaseFieldPotentials.hpp.

14.34.2 Member Function Documentation

14.34.2.1 getH()

```
template<typename...  Args>
std::function<double(const double&)> potential_function< 0, ThermodynamicsPotentialDiscretization↩
::Implicit >::getH (
            Args...  args )  [inline]
```

Interpolation potential $H(x)=x^3 * (6x^2-15x+10)$

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 83 of file PhaseFieldPotentials.hpp.

```
83                                                  {
```

```
    return std::function<double(const double&)>([](double x) {
      const auto pot = x * x * x * (6.0 * x * x - 15.0 * x + 10.0);
      return pot;
    });
  }
```

14.34.2.2 getW()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 0, ThermodynamicsPotentialDiscretization↩
::Implicit >::getW (
            Args... args ) [inline]
```

Double Well potential W(x)=x² ∗ (1-x)²

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 69 of file PhaseFieldPotentials.hpp.

```
69                                                    {
70    return std::function<double(const double&)>([](double x) {
71      const auto pot = x * x * (1.0 - x) * (1.0 - x);
72      return pot;
73    });
74  }
```

14.34.2.3 getX()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 0, ThermodynamicsPotentialDiscretization↩
::Implicit >::getX (
            Args... args ) [inline]
```

Identity potential X(x)=x.

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 97 of file PhaseFieldPotentials.hpp.

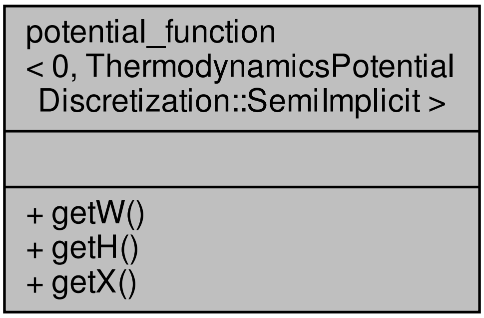
```
97                                                         {
98      return std::function<double(const double&)>([](double x) {
99        const auto pot = x;
100        return pot;
101      });
102    }
```

The documentation for this struct was generated from the following file:

- PhaseFieldPotentials.hpp

14.35 potential_function< 0, ThermodynamicsPotentialDiscretization::SemiImplicit > Struct Template Reference

Collaboration diagram for potential_function< 0, ThermodynamicsPotentialDiscretization::SemiImplicit >:



Public Member Functions

- template<typename... Args>
  std::function< double(const double &)> getW (Args... args)

  Double Well potential W(x)=x² ∗ (1-x)² with semi-implicit scheme (as implicit/explicit schemes)
- template<typename... Args>
  std::function< double(const double &)> getH (Args... args)

  Interpolation potential H(x)=x³ ∗ (6x²-15x+10) with semi-implicit scheme (as implicit/explicit schemes)
- template<typename... Args>
  std::function< double(const double &)> getX (Args... args)

  Identity potential X(x)=x with semi-implicit scheme (as implicit/explicit schemes)

14.35.1 Detailed Description

template<>
struct potential_function< 0, ThermodynamicsPotentialDiscretization::SemiImplicit >

Definition at line 360 of file PhaseFieldPotentials.hpp.

14.35.2 Member Function Documentation

14.35.2.1 getH()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 0, ThermodynamicsPotentialDiscretization↩
::SemiImplicit >::getH (
            Args... args ) [inline]
```

Interpolation potential $H(x)=x^3 * (6x^2-15x+10)$ with semi-implicit scheme (as implicit/explicit schemes)

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 385 of file PhaseFieldPotentials.hpp.

```
385                                                         {
386     return std::function<double(const double&)>([](double x) {
387       const auto pot = x * x * x * (6.0 * x * x - 15.0 * x + 10.0);
388       return pot;
389     });
390   }
```

14.35.2.2 getW()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 0, ThermodynamicsPotentialDiscretization↩
::SemiImplicit >::getW (
            Args... args ) [inline]
```

Double Well potential $W(x)=x^2 * (1-x)^2$ with semi-implicit scheme (as implicit/explicit schemes)

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 370 of file PhaseFieldPotentials.hpp.

```
                                                        {
    return std::function<double(const double&)>([](double x) {
      const auto pot = x * x * (1.0 - x) * (1.0 - x);
      return pot;
    });
  }
```

14.35.2.3 getX()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 0, ThermodynamicsPotentialDiscretization↩
::SemiImplicit >::getX (
              Args... args ) [inline]
```

Identity potential X(x)=x with semi-implicit scheme (as implicit/explicit schemes)

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 400 of file PhaseFieldPotentials.hpp.

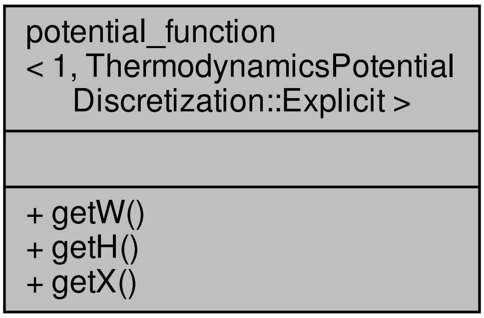
```
                                                        {
    return std::function<double(const double&)>([](double x) {
      const auto pot = x;
      return pot;
    });
  }
```

The documentation for this struct was generated from the following file:

- PhaseFieldPotentials.hpp

14.36 potential_function< 1, ThermodynamicsPotentialDiscretization::Explicit > Struct Template Reference

Collaboration diagram for potential_function< 1, ThermodynamicsPotentialDiscretization::Explicit >:



Public Member Functions

- template<typename... Args>
  std::function< double(const double &)> getW (Args... args)

  First derivative of the double Well potential $W(x)=x^2 * (1-x)^2$ with explicit scheme (as implicit scheme)
- template<typename... Args>
  std::function< double(const double &)> getH (Args... args)

  First derivative of the interpolation potential $H(x)=x^3 * (6x^2-15x+10)$ with explicit scheme (as implicit scheme)
- template<typename... Args>
  std::function< double(const double &)> getX (Args... args)

  First derivative of the identity potential $X(x)=x$ with explicit scheme (as implicit scheme)

14.36.1 Detailed Description

template<>
struct potential_function< 1, ThermodynamicsPotentialDiscretization::Explicit >

Definition at line 255 of file PhaseFieldPotentials.hpp.

14.36.2 Member Function Documentation

14.36.2.1 getH()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 1, ThermodynamicsPotentialDiscretization↩
::Explicit >::getH (
            Args... args ) [inline]
```

First derivative of the interpolation potential $H(x)=x^3 * (6x^2-15x+10)$ with explicit scheme (as implicit scheme)

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 280 of file PhaseFieldPotentials.hpp.

```
280                                                    {
281     return std::function<double(const double&)>([](double x) {
282       const auto pot = 30. * x * x * (1.0 - x) * (1.0 - x);
283       return pot;
284     });
285   }
```

14.36.2.2 getW()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 1, ThermodynamicsPotentialDiscretization↩
::Explicit >::getW (
            Args... args ) [inline]
```

First derivative of the double Well potential $W(x)=x^2 * (1-x)^2$ with explicit scheme (as implicit scheme)

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 265 of file PhaseFieldPotentials.hpp.

```
265                                                            {
266     return std::function<double(const double&)>([](double x) {
267       const auto pot = 2. * x * (1.0 - x) * (1.0 - 2. * x);
268       return pot;
269     });
270   }
```

14.36.2.3 getX()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 1, ThermodynamicsPotentialDiscretization↵
::Explicit >::getX (
              Args...  args )  [inline]
```

First derivative of the identity potential X(x)=x with explicit scheme (as implicit scheme)

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 295 of file PhaseFieldPotentials.hpp.

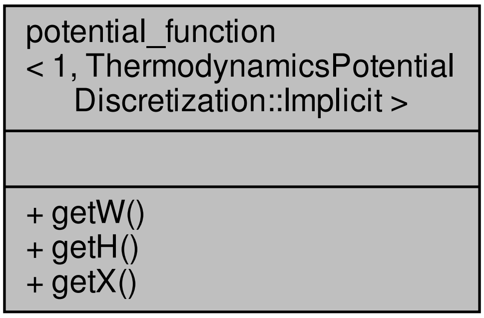
```
295                                                            {
296     return std::function<double(const double&)>([](double x) {
297       const auto pot = 1.;
298       return pot;
299     });
300   }
```

The documentation for this struct was generated from the following file:

- PhaseFieldPotentials.hpp

14.37 potential_function< 1, ThermodynamicsPotentialDiscretization::Implicit > Struct Template Reference

Collaboration diagram for potential_function< 1, ThermodynamicsPotentialDiscretization::Implicit >:



Public Member Functions

- template<typename... Args>
  std::function< double(const double &)> getW (Args... args)
  First derivative of the double Well potential W(x)=x² ∗ (1-x)²
- template<typename... Args>
  std::function< double(const double &)> getH (Args... args)
  First derivative of the interpolation potential H(x)=x³ ∗ (6x²-15x+10)
- template<typename... Args>
  std::function< double(const double &)> getX (Args... args)
  First derivative of the identity potential X(x)=x.

14.37.1 Detailed Description

template<>
struct potential_function< 1, ThermodynamicsPotentialDiscretization::Implicit >

Definition at line 108 of file PhaseFieldPotentials.hpp.

14.37.2 Member Function Documentation

14.37.2.1 getH()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 1, ThermodynamicsPotentialDiscretization↩
::Implicit >::getH (
            Args... args ) [inline]
```

First derivative of the interpolation potential H(x)=x³ ∗ (6x²-15x+10)

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 131 of file PhaseFieldPotentials.hpp.

```
131                                                         {
132     return std::function<double(const double&)>([](double x) {
133       const auto pot = 30. * x * x * (1.0 - x) * (1.0 - x);
134       return pot;
135     });
136   }
```

14.37.2.2 getW()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 1, ThermodynamicsPotentialDiscretization↩
::Implicit >::getW (
              Args... args ) [inline]
```

First derivative of the double Well potential W(x)=x² ∗ (1-x)²

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 117 of file PhaseFieldPotentials.hpp.

```
117                                                         {
118     return std::function<double(const double&)>([](double x) {
119       const auto pot = 2. * x * (1.0 - x) * (1.0 - 2. * x);
120       return pot;
121     });
122   }
```

14.37.2.3 getX()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 1, ThermodynamicsPotentialDiscretization↩
::Implicit >::getX (
            Args... args ) [inline]
```

First derivative of the identity potential X(x)=x.

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 145 of file PhaseFieldPotentials.hpp.

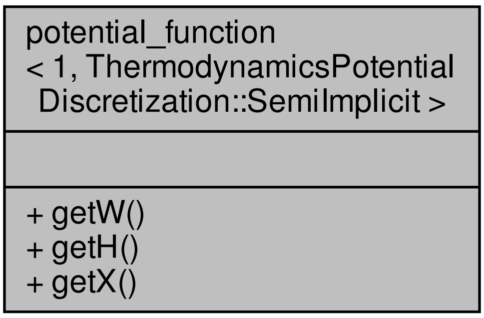
```
145                                                    {
146     return std::function<double(const double&)>([](double x) {
147       const auto pot = 1.;
148       return pot;
149     });
150   }
```

The documentation for this struct was generated from the following file:

- PhaseFieldPotentials.hpp

14.38 potential_function< 1, ThermodynamicsPotentialDiscretization::SemiImplicit > Struct Template Reference

Collaboration diagram for potential_function< 1, ThermodynamicsPotentialDiscretization::SemiImplicit >:



Public Member Functions

- template<typename... Args>
  std::function< double(const double &)> getW (Args... args)

  First derivative of the double Well potential W(x)=x² ∗ (1-x)² with semi-implicit scheme.
- template<typename... Args>
  std::function< double(const double &)> getH (Args... args)

  First derivative of the interpolation potential H(x)=x³ ∗ (6x²-15x+10) with semi-implicit scheme (as implicit/explicit schemes)
- template<typename... Args>
  std::function< double(const double &)> getX (Args... args)

  First derivative of the identity potential X(x)=x with semi-implicit scheme (as implicit/explicit schemes)

14.38.1 Detailed Description

template<>
struct potential_function< 1, ThermodynamicsPotentialDiscretization::SemiImplicit >

Definition at line 411 of file PhaseFieldPotentials.hpp.

14.38.2 Member Function Documentation

14.38.2.1 getH()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 1, ThermodynamicsPotentialDiscretization↩
::SemiImplicit >::getH (
            Args... args ) [inline]
```

First derivative of the interpolation potential H(x)=x³ ∗ (6x²-15x+10) with semi-implicit scheme (as implicit/explicit schemes)

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 444 of file PhaseFieldPotentials.hpp.

```
                                                        {
    return std::function<double(const double&)>([](double x) {
      const auto pot = 30. * x * x * (1.0 - x) * (1.0 - x);
      return pot;
    });
  }
```

14.38.2.2 getW()

template<typename... Args>
std::function<double(const double&)> potential_function< 1, ThermodynamicsPotentialDiscretization::SemiImplicit >::getW (
Args... *args*) [inline]

First derivative of the double Well potential $W(x)=x^2 * (1-x)^2$ with semi-implicit scheme.

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 421 of file PhaseFieldPotentials.hpp.

```
421                                                         {
422     auto v = std::vector<double>{args...};
423 
424     if (v.size() == 1) {
425       const auto xn = v[0];
426       return std::function<double(const double&)>([xn](double x) {
427         const auto pot = (1.0 - x - xn) * (x + xn - x * x - xn * xn);
428         return pot;
429       });
430     } else {
431       throw std::runtime_error(
432           "potential_function::getW: only one argument is expected for smei-implicit scheme");
433     }
434   }
```

14.38.2.3 getX()

template<typename... Args>
std::function<double(const double&)> potential_function< 1, ThermodynamicsPotentialDiscretization::SemiImplicit >::getX (
Args... *args*) [inline]

First derivative of the identity potential X(x)=x with semi-implicit scheme (as implicit/explicit schemes)

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 459 of file PhaseFieldPotentials.hpp.

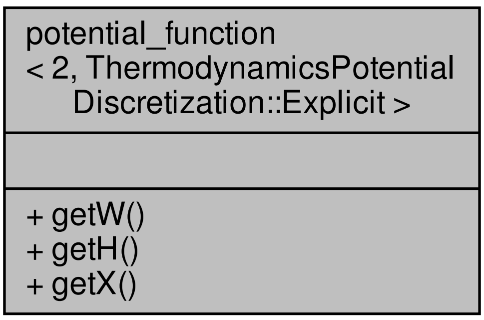
```
459                                                           {
460     return std::function<double(const double&)>([](double x) {
461       const auto pot = 1.;
462       return pot;
463     });
464   }
```

The documentation for this struct was generated from the following file:

- PhaseFieldPotentials.hpp

14.39 potential_function< 2, ThermodynamicsPotentialDiscretization::Explicit > Struct Template Reference

Collaboration diagram for potential_function< 2, ThermodynamicsPotentialDiscretization::Explicit >:



Public Member Functions

- template<typename... Args>
  std::function< double(const double &)> getW (Args... args)

  Second derivative of the double Well potential W(x)=$x^2 * (1-x)^2$ with explicit scheme (as implicit scheme)
- template<typename... Args>
  std::function< double(const double &)> getH (Args... args)

  Second derivative of the interpolation potential H(x)=$x^3 * (6x^2-15x+10)$ with explicit scheme (as implicit scheme)
- template<typename... Args>
  std::function< double(const double &)> getX (Args... args)

  Second derivative of the identity potential X(x)=x with explicit scheme (as implicit scheme)

14.39.1 Detailed Description

template<>
struct potential_function< 2, ThermodynamicsPotentialDiscretization::Explicit >

Definition at line 306 of file PhaseFieldPotentials.hpp.

14.39.2 Member Function Documentation

14.39.2.1 getH()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 2, ThermodynamicsPotentialDiscretization↩
::Explicit >::getH (
            Args... args ) [inline]
```

Second derivative of the interpolation potential H(x)=$x^3 * (6x^2-15x+10)$ with explicit scheme (as implicit scheme)

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 331 of file PhaseFieldPotentials.hpp.

```
331 {
```

```
    return std::function<double(const double&)>([](double x) {
      const auto pot = 60. * x * (1.0 - x) * (1.0 - 2. * x);
      return pot;
    });
  }
```

14.39.2.2 getW()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 2, ThermodynamicsPotentialDiscretization↩
::Explicit >::getW (
            Args... args ) [inline]
```

Second derivative of the double Well potential W(x)=x² ∗ (1-x)² with explicit scheme (as implicit scheme)

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 316 of file PhaseFieldPotentials.hpp.

```
                                                    {
    return std::function<double(const double&)>([](double x) {
      const auto pot = 2. * (1. - 6. * x + 6. * x * x);
      return pot;
    });
  }
```

14.39.2.3 getX()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 2, ThermodynamicsPotentialDiscretization↩
::Explicit >::getX (
            Args... args ) [inline]
```

Second derivative of the identity potential X(x)=x with explicit scheme (as implicit scheme)

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 346 of file PhaseFieldPotentials.hpp.

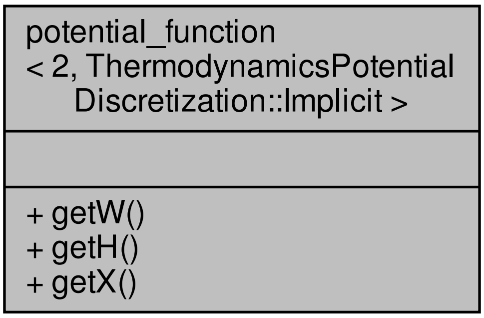
```
346                                                         {
347     return std::function<double(const double&)>([](double x) {
348       const auto pot = 0.;
349       return pot;
350     });
351   }
```

The documentation for this struct was generated from the following file:

- PhaseFieldPotentials.hpp

14.40 potential_function< 2, ThermodynamicsPotentialDiscretization::Implicit > Struct Template Reference

Collaboration diagram for potential_function< 2, ThermodynamicsPotentialDiscretization::Implicit >:



Public Member Functions

- template<typename... Args>
  std::function< double(const double &)> getW (Args... args)

  Second derivative of the double Well potential W(x)=x²∗ (1-x)²
- template<typename... Args>
  std::function< double(const double &)> getH (Args... args)

  Second derivative of the interpolation potential H(x)=x³∗ (6x²-15x+10)
- template<typename... Args>
  std::function< double(const double &)> getX (Args... args)

  Second derivative of the identity potential X(x)=x.

14.40.1 Detailed Description

template<>
struct potential_function< 2, ThermodynamicsPotentialDiscretization::Implicit >

Definition at line 156 of file PhaseFieldPotentials.hpp.

14.40.2 Member Function Documentation

14.40.2.1 getH()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 2, ThermodynamicsPotentialDiscretization↩
::Implicit >::getH (
            Args... args ) [inline]
```

Second derivative of the interpolation potential $H(x)=x^3 * (6x^2-15x+10)$

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 179 of file PhaseFieldPotentials.hpp.

```
179                                                          {
180     return std::function<double(const double&)>([](double x) {
181       const auto pot = 60. * x * (1.0 - x) * (1.0 - 2. * x);
182       return pot;
183     });
184   }
```

14.40.2.2 getW()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 2, ThermodynamicsPotentialDiscretization↩
::Implicit >::getW (
            Args... args ) [inline]
```

Second derivative of the double Well potential $W(x)=x^2 * (1-x)^2$

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 165 of file PhaseFieldPotentials.hpp.

```
165                                                       {
166     return std::function<double(const double&)>([](double x) {
167       const auto pot = 2. * (1. - 6. * x + 6. * x * x);
168       return pot;
169     });
170   }
```

14.40.2.3 getX()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 2, ThermodynamicsPotentialDiscretization↩
::Implicit >::getX (
              Args... args ) [inline]
```

Second derivative of the identity potential X(x)=x.

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 193 of file PhaseFieldPotentials.hpp.

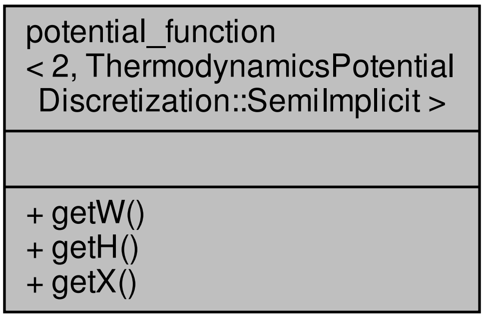
```
193                                                       {
194     return std::function<double(const double&)>([](double x) {
195       const auto pot = 0.;
196       return pot;
197     });
198   }
```

The documentation for this struct was generated from the following file:

- PhaseFieldPotentials.hpp

14.41 potential_function< 2, ThermodynamicsPotentialDiscretization::SemiImplicit > Struct Template Reference

Collaboration diagram for potential_function< 2, ThermodynamicsPotentialDiscretization::SemiImplicit >:



Public Member Functions

- template<typename... Args>
  std::function< double(const double &)> getW (Args... args)

  Second derivative of the double Well potential $W(x)=x^2 * (1-x)^2$ with semi-implicit scheme.
- template<typename... Args>
  std::function< double(const double &)> getH (Args... args)

  Second derivative of the interpolation potential $H(x)=x^3 * (6x^2-15x+10)$ with semi-implicit scheme.
- template<typename... Args>
  std::function< double(const double &)> getX (Args... args)

  Second derivative of the identity potential X(x)=x with semi-implicit scheme (as implicit/explicit schemes)

14.41.1 Detailed Description

template<>
struct potential_function< 2, ThermodynamicsPotentialDiscretization::SemiImplicit >

Definition at line 470 of file PhaseFieldPotentials.hpp.

14.41.2 Member Function Documentation

14.41.2.1 getH()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 2, ThermodynamicsPotentialDiscretization↩
::SemiImplicit >::getH (
            Args... args ) [inline]
```

Second derivative of the interpolation potential $H(x)=x^3 * (6x^2-15x+10)$ with semi-implicit scheme.

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 503 of file PhaseFieldPotentials.hpp.

```
                                                        {
    auto v = std::vector<double>{args...};

    if (v.size() == 1) {
      const auto xn = v[0];
      return std::function<double(const double&)>([xn](double x) {
        const auto pot = 30. * (1.0 - x - xn) * (x + xn - x * x - xn * xn);
        return pot;
      });
    } else {
      throw std::runtime_error(
          "potential_function::getH: only one argument is expected for smei-implicit scheme");
    }
  }
```

14.41.2.2 getW()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 2, ThermodynamicsPotentialDiscretization↩
::SemiImplicit >::getW (
            Args... args ) [inline]
```

Second derivative of the double Well potential $W(x)=x^2 * (1-x)^2$ with semi-implicit scheme.

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 480 of file PhaseFieldPotentials.hpp.

```
                                                        {
    auto v = std::vector<double>{args...};

    if (v.size() == 1) {
      const auto xn = v[0];
      return std::function<double(const double&)>([xn](double x) {
        const auto pot = ((1.0 - x - xn) * (1.0 - 2.0 * x) - (x + xn - x * x - xn * xn));
        return pot;
      });
    } else {
      throw std::runtime_error(
          "potential_function::getW: only one argument is expected for smei-implicit scheme");
    }
  }
```

14.41.2.3 getX()

```
template<typename... Args>
std::function<double(const double&)> potential_function< 2, ThermodynamicsPotentialDiscretization↩
::SemiImplicit >::getX (
              Args... args ) [inline]
```

Second derivative of the identity potential X(x)=x with semi-implicit scheme (as implicit/explicit schemes)

Template Parameters

Args	

Parameters

args	

Returns

std::function<double(const double&)>

Definition at line 526 of file PhaseFieldPotentials.hpp.

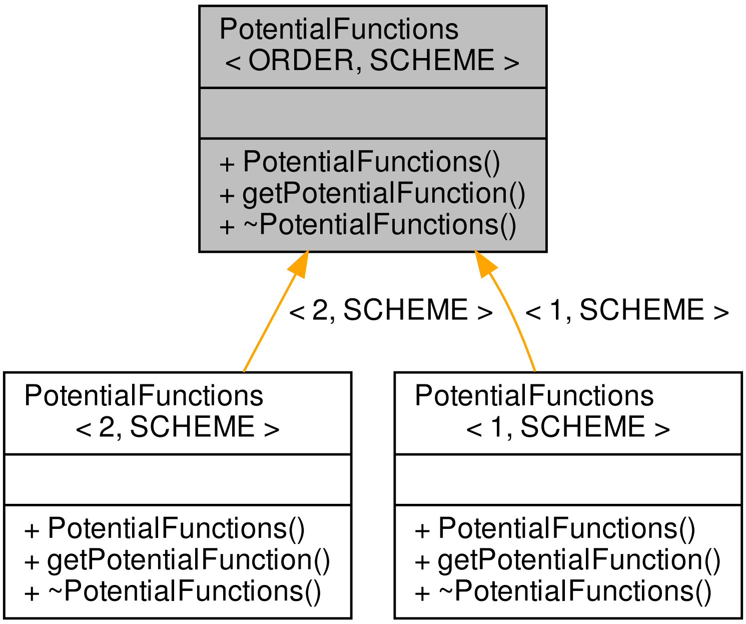
```
                                                        {
    return std::function<double(const double&)>([](double x) {
      const auto pot = 0.;
      return pot;
    });
  }
```

The documentation for this struct was generated from the following file:

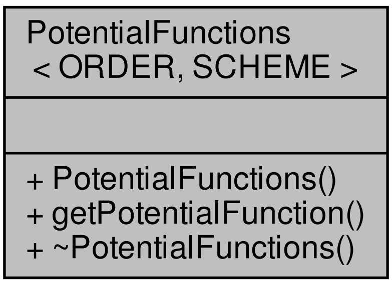
- PhaseFieldPotentials.hpp

14.42 PotentialFunctions< ORDER, SCHEME > Class Template Reference

Inheritance diagram for PotentialFunctions< ORDER, SCHEME >:



Collaboration diagram for PotentialFunctions< ORDER, SCHEME >:



Public Member Functions

- PotentialFunctions ()

    Construct a new potential function:: potential function object.
- template<class... Args>
  std::function< double(const double &)> getPotentialFunction (const std::string &analytical_function_name, Args... args)

    return the function associated with the potential_name and its ORDER of derivative
- ~PotentialFunctions ()

    Destroy the potential function :: potential function object.

14.42.1 Detailed Description

template<int ORDER, ThermodynamicsPotentialDiscretization SCHEME>
class PotentialFunctions< ORDER, SCHEME >

Definition at line 23 of file PhaseFieldPotentials.hpp.

14.42.2 Member Function Documentation

14.42.2.1 getPotentialFunction()

```
template<int ORDER, ThermodynamicsPotentialDiscretization SCHEME>
template<class... Args>
std::function< double(const double &)> PotentialFunctions< ORDER, SCHEME >::getPotential←
Function (
            const std::string & potential_name,
            Args... args )
```

return the function associated with the potential_name and its ORDER of derivative

Parameters

potential_name	

Returns

const double

Definition at line 566 of file PhaseFieldPotentials.hpp.

Referenced by CahnHilliardSpecializedNLFormIntegrator< SCHEME >::AssembleElementGrad(), Allen←CahnSpecializedNLFormIntegrator< SCHEME >::AssembleElementGrad(), CahnHilliardSpecializedNLForm←Integrator< SCHEME >::AssembleElementVector(), and AllenCahnSpecializedNLFormIntegrator< SCHEME >::AssembleElementVector().

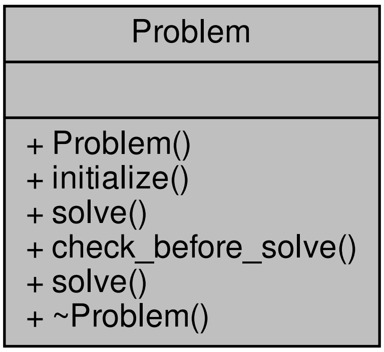
```
                                                    {
  switch (ThermodynamicsPotentials::from(potential_name)) {
    case ThermodynamicsPotentials::W:
      return this->getW(args...);
    case ThermodynamicsPotentials::H:
      return this->getH(args...);
    case ThermodynamicsPotentials::X:
      return this->getX(args...);
    default:
      throw std::runtime_error(
          "PotentialFunctions::getPotentialFunctions: double well, H interpolation and identity "
          "potential function  are available");
      break;
  }
}
```

The documentation for this class was generated from the following file:

- PhaseFieldPotentials.hpp

14.43 Problem Class Reference

Collaboration diagram for Problem:



Public Member Functions

- Problem ()

    Construct a new Problem:: Problem object.
- virtual void **initialize** ()=0
- virtual void **solve** ()=0
- void check_before_solve ()

    check the existance of mandatory objects
- void solve ()

    check the existance of mandatory objects and solve the problem
- ∼Problem ()

    Destroy the Problem:: Problem object.

14.43.1 Detailed Description

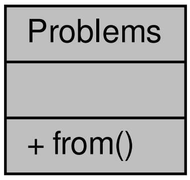
Definition at line 22 of file Problem.hpp.

The documentation for this class was generated from the following file:

- Problem.hpp

14.44 Problems Struct Reference

Collaboration diagram for Problems:



Public Types

- enum **value** { **Diffusion**, **AllenCahn**, **CahnHilliard** }

Static Public Member Functions

- static value **from** (const std::string &)

14.44.1 Detailed Description

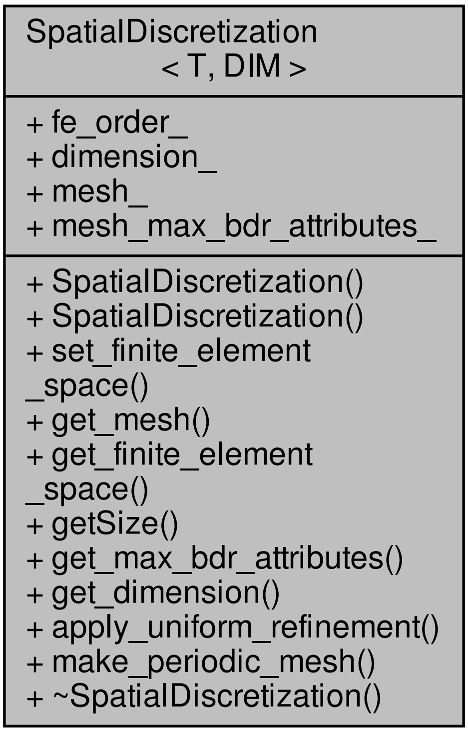
Definition at line 110 of file Utils/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

- Utils/PhaseFieldOptions.hpp

14.45 SpatialDiscretization< T, DIM > Class Template Reference

Collaboration diagram for SpatialDiscretization< T, DIM >:



Public Member Functions

- **SpatialDiscretization** (const std::string &mesh_type, const int &fe_order, const std::string &mesh_file)
- template<class... Args>
  SpatialDiscretization (const std::string &mesh_type, const int &fe_order, std::tuple< Args... > tup_args)
- void set_finite_element_space ()

  Set the FE_Collection, the FE_Space and associated size.
- mfem::Mesh & get_mesh ()

  return a pointer of Mesh
- mfem::FiniteElementSpace ∗ get_finite_element_space ()

  return a pointer toward the finite element space
- std::size_t getSize ()

  get the size of the Finite Element Space
- std::size_t get_max_bdr_attributes ()

  get the maximum number of boundaries
- int get_dimension ()

  get the dimension of the problem
- void apply_uniform_refinement (const int &level)

Apply nb_ref uniform refinement.

- void make_periodic_mesh (std::vector< mfem::Vector >)

  Create the periodic mesh using the vertex mapping defined by the translations vector.

- ∼SpatialDiscretization ()

  Destroy the Spatial Discretization< T>:: Spatial Discretization object.

Data Fields

- int **fe_order_**
- int **dimension_**
- mfem::Mesh **mesh_**
- int **mesh_max_bdr_attributes_**

14.45.1 Detailed Description

template<class T, int DIM>
class SpatialDiscretization< T, DIM >

Definition at line 30 of file Spatial.hpp.

14.45.2 Constructor & Destructor Documentation

14.45.2.1 ∼SpatialDiscretization()

```
template<class T , int DIM>
SpatialDiscretization< T, DIM >::∼SpatialDiscretization ( )
```

Destroy the Spatial Discretization< T>:: Spatial Discretization object.

Template Parameters

T	

Definition at line 423 of file Spatial.hpp.

```
423 {}
```

14.45.3 Member Function Documentation

14.45.3.1 apply_uniform_refinement()

template<class T , int DIM>
void SpatialDiscretization< T, DIM >::apply_uniform_refinement (
const int & *nb_ref*)

Apply nb_ref uniform refinement.

Template Parameters

T	

Parameters

nb_ref	

Definition at line 379 of file Spatial.hpp.

```
379                                                                         {
380   for (auto l = 0; l < nb_ref; l++) {
381     this->mesh_.UniformRefinement();
382   }
383 }
```

14.45.3.2 get_dimension()

template<class T , int DIM>
int SpatialDiscretization< T, DIM >::get_dimension ( )

get the dimension of the problem

Template Parameters

T	

Returns

int

Definition at line 368 of file Spatial.hpp.

Referenced by Variable< T, DIM >::Variable().

```
368                                            {
369   return this->dimension_;
370 }
```

14.45.3.3 get_finite_element_space()

```
template<class T , int DIM>
mfem::FiniteElementSpace * SpatialDiscretization< T, DIM >::get_finite_element_space ( )
```

return a pointer toward the finite element space

Template Parameters

T	

Returns

mfem::FiniteElementSpace∗

Definition at line 335 of file Spatial.hpp.

Referenced by BoundaryConditions< T, DIM >::BoundaryConditions(), PhaseFieldOperator< T, DIM >::PhaseFieldOperator(), and Variable< T, DIM >::Variable().

```
335                                                                                {
336   return this->fespace_;
337 }
```

14.45.3.4 get_max_bdr_attributes()

```
template<class T , int DIM>
std::size_t SpatialDiscretization< T, DIM >::get_max_bdr_attributes ( )
```

get the maximum number of boundaries

Template Parameters

T	

Returns

int

Definition at line 357 of file Spatial.hpp.

Referenced by BoundaryConditions< T, DIM >::BoundaryConditions().

```
357                                                                {
358   return this->mesh_max_bdr_attributes_;
359 }
```

14.45.3.5 get_mesh()

```
template<class T , int DIM>
mfem::Mesh & SpatialDiscretization< T, DIM >::get_mesh ( )
```

return a pointer of Mesh

Template Parameters

T	

Returns

mfem::Mesh&

Definition at line 311 of file Spatial.hpp.

```
311                                                   {
312   return this->mesh_;
313 }
```

14.45.3.6 getSize()

```
template<class T , int DIM>
std::size_t SpatialDiscretization< T, DIM >::getSize ( )
```

get the size of the Finite Element Space

Template Parameters

T	

Returns

int

Definition at line 346 of file Spatial.hpp.

```
346                                                  {
347   return this->size_;
348 }
```

14.45.3.7 make_periodic_mesh()

```
template<class T , int DIM>
void SpatialDiscretization< T, DIM >::make_periodic_mesh (
             std::vector< mfem::Vector > translations )
```

Create the periodic mesh using the vertex mapping defined by the translations vector.

Template Parameters

T	
DIM	

Parameters

translations	

Definition at line 393 of file Spatial.hpp.

```
393                                                                                   {
394   this->mesh_.mfem::Mesh::GenerateBoundaryElements();
395   const auto tol = 1.e-6;
396   std::vector<int> periodicMap = this->mesh_.CreatePeriodicVertexMapping(translations, tol);
397
398   auto periodic_mesh = mfem::Mesh::MakePeriodic(this->mesh_, periodicMap);
399   auto j = 0;
400   for (int i = 0; i < static_cast<int>(periodicMap.size()); i++) {
401     if (i != periodicMap[i]) {
402       j++;
403       std::cout << "... periodicMap[" << i << "] = " << periodicMap[i] << " j " << j << std::endl;
404     }
405   }
406   this->mesh_ = mfem::Mesh(periodic_mesh, true);  // replace the input mesh with the periodic one
407
408   mfem::Vector vert_coord;
409   this->mesh_GetVertices(vert_coord) for (int i = 0; i < static_cast<int>(vert_coord.size()); i++) {
410     if (i != periodicMap[i]) {
411       j++;
412       std::cout << "... vert_coord[" << i << "] = " << vert_coord[i] << " j " << j << std::endl;
413     }
414   }
415 }
```

14.45.3.8 set_finite_element_space()

```
template<class T , int DIM>
void SpatialDiscretization< T, DIM >::set_finite_element_space ( )
```

Set the FE_Collection, the FE_Space and associated size.

Template Parameters

T	

Returns

mfem::FiniteElementSpace∗

Definition at line 322 of file Spatial.hpp.

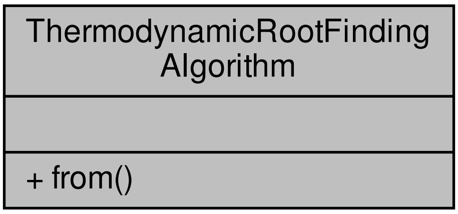
```
322                                                       {
323   this->fecollection_ = new T(this->fe_order_, this->dimension_);
324   this->fespace_ = new mfem::FiniteElementSpace(&this->mesh_, this->fecollection_);
325   this->size_ = this->fespace_->GetTrueVSize();
326 }
```

The documentation for this class was generated from the following file:

- Spatial.hpp

14.46 ThermodynamicRootFindingAlgorithm Struct Reference

Collaboration diagram for ThermodynamicRootFindingAlgorithm:



Public Types

- enum **value** { **MU**, **DeltaMU** }

Static Public Member Functions

- static value **from** (const std::string &)

14.46.1 Detailed Description

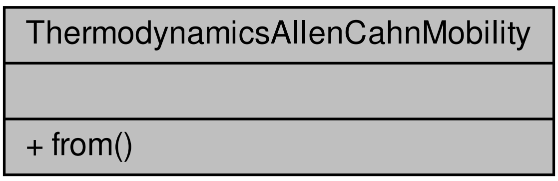
Definition at line 159 of file Coefficients/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

- Coefficients/PhaseFieldOptions.hpp

14.47 ThermodynamicsAllenCahnMobility Struct Reference

Collaboration diagram for ThermodynamicsAllenCahnMobility:



Public Types

- enum **value** { **Given**, **Logarithmic**, **LogarithmicMean** }

Static Public Member Functions

- static value **from** (const std::string &)

14.47.1 Detailed Description

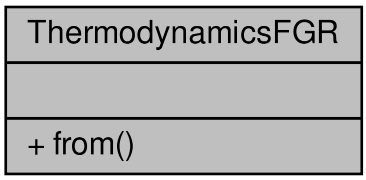
Definition at line 121 of file Coefficients/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

- Coefficients/PhaseFieldOptions.hpp

14.48 ThermodynamicsFGR Struct Reference

Collaboration diagram for ThermodynamicsFGR:



Public Types

- enum **value** { **No**, **REP**, **RNR** }

Static Public Member Functions

- static value **from** (const std::string &)

14.48.1 Detailed Description

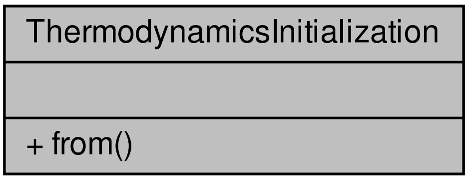
Definition at line 39 of file Coefficients/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

- Coefficients/PhaseFieldOptions.hpp

14.49 ThermodynamicsInitialization Struct Reference

Collaboration diagram for ThermodynamicsInitialization:



Public Types

- enum **value** { **UserDefined**, **UserDefinedControlledAtmosphere**, **Cesar**, **Prodhel** }

Static Public Member Functions

- static value **from** (const std::string &)

14.49.1 Detailed Description

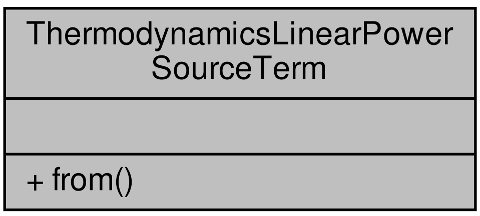
Definition at line 21 of file Coefficients/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

- Coefficients/PhaseFieldOptions.hpp

14.50 ThermodynamicsLinearPowerSourceTerm Struct Reference

Collaboration diagram for ThermodynamicsLinearPowerSourceTerm:



Public Types

- enum **value** { **Constant**, **TimeDependent**, **ModifiedBesselFunction** }

Static Public Member Functions

- static value **from** (const std::string &)

14.50.1 Detailed Description

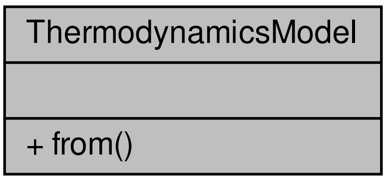
Definition at line 103 of file Coefficients/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

- Coefficients/PhaseFieldOptions.hpp

14.51 ThermodynamicsModel Struct Reference

Collaboration diagram for ThermodynamicsModel:



Public Types

- enum **value** {
  No, **OPENCALPHAD**, **OPENCALPHADwithOXITRAN**, **OPENCALPHADwithOXIRED**,
  OPENCALPHADPHASEFIELD, **OPENCALPHADPHASEFIELD_SPECIFICTIME** }

Static Public Member Functions

- static value **from** (const std::string &)

14.51.1 Detailed Description

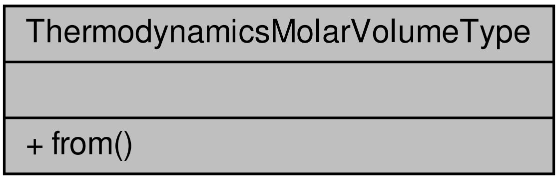
Definition at line 44 of file Coefficients/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

- Coefficients/PhaseFieldOptions.hpp

14.52 ThermodynamicsMolarVolumeType Struct Reference

Collaboration diagram for ThermodynamicsMolarVolumeType:



Public Types

- enum **value** { **One**, **Constant** }

Static Public Member Functions

- static value **from** (const std::string &)

14.52.1 Detailed Description

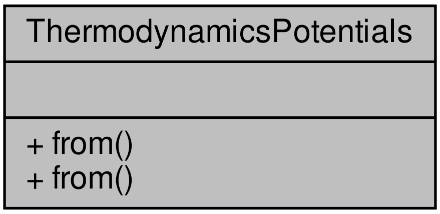
Definition at line 109 of file Coefficients/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

- Coefficients/PhaseFieldOptions.hpp

14.53 ThermodynamicsPotentials Struct Reference

Collaboration diagram for ThermodynamicsPotentials:



Public Types

- enum **value** {
  W, **H**, **X**, **W**,
  H, **X** }
- enum **value** {
  W, **H**, **X**, **W**,
  H, **X** }

Static Public Member Functions

- static value **from** (const std::string &)
- static value **from** (const std::string &)

14.53.1 Detailed Description

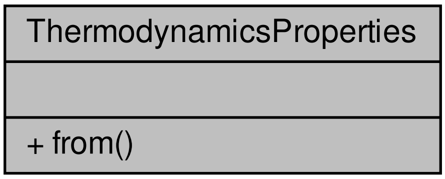
Definition at line 116 of file Coefficients/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

- Coefficients/PhaseFieldOptions.hpp

14.54 ThermodynamicsProperties Struct Reference

Collaboration diagram for ThermodynamicsProperties:



Public Types

- enum **value** {
  WPC, **WTM**, **WME**, **WSD**,
  WLD, **OSD**, **WSHP**, **RSHP**,
  KSHP, **CMV**, **WLC**, **WSC**,
  WSR, **WLR**, **WLCP**, **WSCP** }

Static Public Member Functions

- static value **from** (const std::string &)

14.54.1 Detailed Description

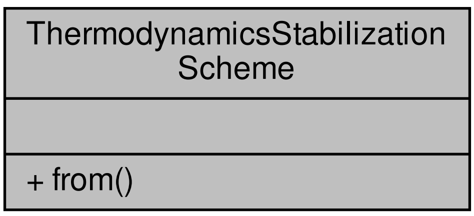
Definition at line 136 of file Coefficients/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

- Coefficients/PhaseFieldOptions.hpp

14.55 ThermodynamicsStabilizationScheme Struct Reference

Collaboration diagram for ThermodynamicsStabilizationScheme:



Public Types

- enum **value** { **No**, **Laplacian1**, **Laplacian2** }

Static Public Member Functions

- static value **from** (const std::string &)

14.55.1 Detailed Description

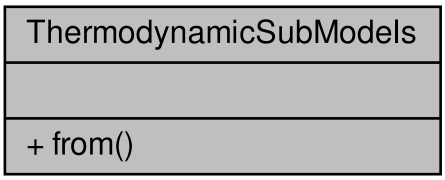
Definition at line 78 of file Coefficients/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

- Coefficients/PhaseFieldOptions.hpp

14.56 ThermodynamicSubModels Struct Reference

Collaboration diagram for ThermodynamicSubModels:



Public Types

- enum **value** {
  No, **OnlyAllenCahn**, **ThermalDiffusion_U_O**, **ThermalDiffusion_U_PU_O**,
  PhaseField_U_O, **PhaseField_U_PU_O**, **PhaseField_Welland** }

Static Public Member Functions

- static value **from** (const std::string &)

14.56.1 Detailed Description

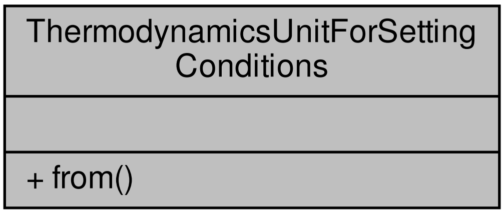
Definition at line 26 of file Coefficients/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

- Coefficients/PhaseFieldOptions.hpp

14.57 ThermodynamicsUnitForSettingConditions Struct Reference

Collaboration diagram for ThermodynamicsUnitForSettingConditions:



Public Types

- enum **value** { **MoleNumbers**, **MoleFractions**, **MoleNumbers2MolesFractions**, **OneMoleNumbers2MolesFractions** }

Static Public Member Functions

- static value **from** (const std::string &)

14.57.1 Detailed Description

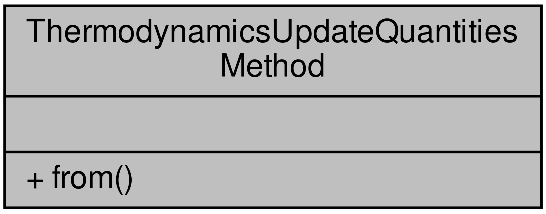
Definition at line 61 of file Coefficients/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

- Coefficients/PhaseFieldOptions.hpp

14.58 ThermodynamicsUpdateQuantitiesMethod Struct Reference

Collaboration diagram for ThermodynamicsUpdateQuantitiesMethod:



Public Types

- enum **value** { **MoleFractions**, **MoleNumbers**, **UnconservedMoleNumbers** }

Static Public Member Functions

- static value **from** (const std::string &)

14.58.1 Detailed Description

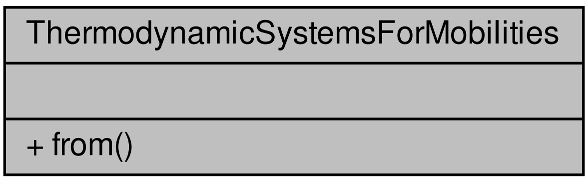
Definition at line 56 of file Coefficients/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

- Coefficients/PhaseFieldOptions.hpp

14.59 ThermodynamicSystemsForMobilities Struct Reference

Collaboration diagram for ThermodynamicSystemsForMobilities:



Public Types

- enum **value** { **UO2**, **UPUO2** }

Static Public Member Functions

- static value **from** (const std::string &)

14.59.1 Detailed Description

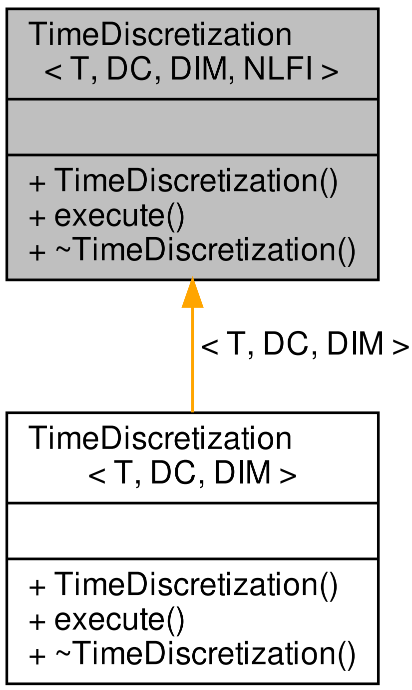
Definition at line 165 of file Coefficients/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

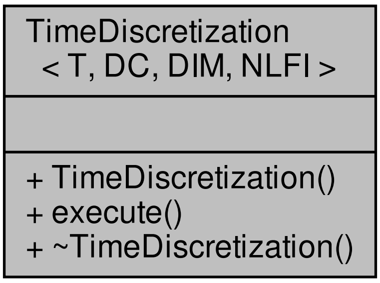
- Coefficients/PhaseFieldOptions.hpp

14.60 TimeDiscretization< T, DC, DIM, NLFI > Class Template Reference

Inheritance diagram for TimeDiscretization< T, DC, DIM, NLFI >:



Collaboration diagram for TimeDiscretization< T, DC, DIM, NLFI >:



Public Member Functions

- TimeDiscretization (const std::string &ode_solver, const PhaseFieldOperator< T, DIM, NLFI > &oper, const Parameters &params, const Variables< T, DIM > &variables, PostProcessing< T, DC, DIM > &pst)

  Construct a new Time Discretization:: Time Discretization object.
- void execute ()

  Run the calculation.
- ~TimeDiscretization ()

  Destroy the Time Discretization:: Time Discretization object.

14.60.1 Detailed Description

template<class T, class DC, int DIM, class NLFI>
class TimeDiscretization< T, DC, DIM, NLFI >

Definition at line 23 of file Time.hpp.

14.60.2 Constructor & Destructor Documentation

14.60.2.1 TimeDiscretization()

```
template<class T, class DC, int DIM, class NLFI>
TimeDiscretization< T, DC, DIM, NLFI >::TimeDiscretization (
            const std::string & ode_solver,
            const PhaseFieldOperator< T, DIM, NLFI > & oper,
            const Parameters & params,
            const Variables< T, DIM > & variables,
            PostProcessing< T, DC, DIM > & pst )
```

Construct a new Time Discretization:: Time Discretization object.

Parameters

ode_solver	
unknown	
with_save	

Definition at line 81 of file Time.hpp.

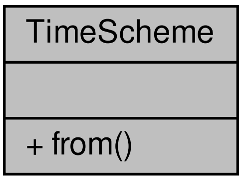
```
      : oper_(oper), variables_(variables), pst_(pst) {
    this->initial_time_ = params.get_parameter_value("initial_time");
    this->final_time_ = params.get_parameter_value("final_time");
    this->time_step_ = params.get_parameter_value("time_step");
    this->setODESolver(ode_solver);
  }
```

The documentation for this class was generated from the following file:

- Time.hpp

14.61 TimeScheme Struct Reference

Collaboration diagram for TimeScheme:



Public Types

- enum **value** { **EulerImplicit**, **EulerExplicit**, **RungeKutta4** }

Static Public Member Functions

- static value **from** (const std::string &)

14.61.1 Detailed Description

Definition at line 75 of file Utils/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

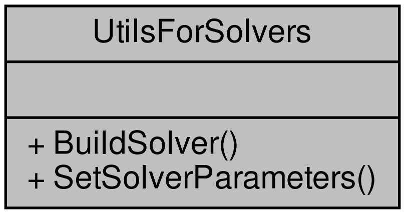
- Utils/PhaseFieldOptions.hpp

14.62 UtilsForSolvers Class Reference

Useful methods for managing solvers.

```
#include </home/ci230846/home-local/MyGitProjects/COMPONENT/PF-MFEM/Numerical↩
Methods/UtilsForSolvers.hpp>
```

Collaboration diagram for UtilsForSolvers:



Public Member Functions

- void BuildSolver (mfem::IterativeSolver &solver, mfem::Solver &solv_method, mfem::Operator &ope)

  Build solver depending on given preconditionner and operator.
- void SetSolverParameters (mfem::IterativeSolver &solver, const int &_print_level, const bool _iterative_mode, const int &_n_iter_max, const double &_n_rel_tol, const double &_n_abs_tol)

  Set all parameters used by the solver.

14.62.1 Detailed Description

Useful methods for managing solvers.

Definition at line 17 of file UtilsForSolvers.hpp.

14.62.2 Member Function Documentation

14.62.2.1 BuildSolver()

```
void UtilsForSolvers::BuildSolver (
            mfem::IterativeSolver & solver,
            mfem::Solver & preconditionner,
            mfem::Operator & ope )
```

Build solver depending on given preconditionner and operator.

Parameters

solver	IterativeSolver
preconditionner	Preconditionner
ope	Operator

Definition at line 32 of file UtilsForSolvers.hpp.

Referenced by PhaseFieldOperator< T, DIM >::SetConstantParameters(), and PhaseFieldOperator< T, DIM >↩
::SetTransientParameters().

```
                                                    {
  solver.SetPreconditioner(preconditionner);
  solver.SetOperator(ope);
}
```

14.62.2.2 SetSolverParameters()

```
void UtilsForSolvers::SetSolverParameters (
            mfem::IterativeSolver & solver,
            const int & _print_level,
            const bool _iterative_mode,
            const int & _n_iter_max,
            const double & _n_rel_tol,
            const double & _n_abs_tol )
```

Set all parameters used by the solver.

Parameters

solver	IterativeSolver
_print_level	printing level
_iterative_mode	flag to activate the iterative mode (initialization of the Iterative Solver, False by default)
_n_iter_max	maximum number of iterations used by the IterativeSolver
_n_rel_tol	relative tolerance used by the IterativeSolver convergence test
_n_abs_tol	absolute tolerance used by the IterativeSolver convergence test

Definition at line 48 of file UtilsForSolvers.hpp.

Referenced by PhaseFieldOperator< T, DIM >::SetConstantParameters(), and PhaseFieldOperator< T, DIM >↩
::SetTransientParameters().

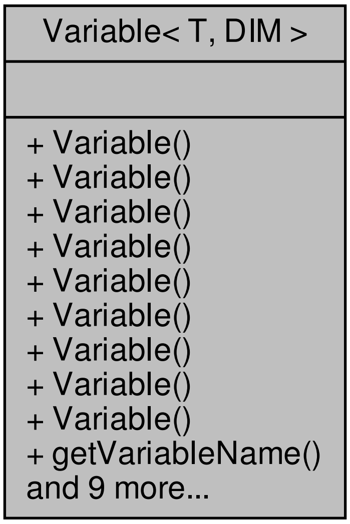
```
                                                                                                  {
  solver.SetPrintLevel(_print_level);
  solver.iterative_mode = _iterative_mode;
  solver.SetMaxIter(_n_iter_max);
  solver.SetRelTol(_n_rel_tol);
  solver.SetAbsTol(_n_abs_tol);
}
```

The documentation for this class was generated from the following file:

- UtilsForSolvers.hpp

14.63 Variable< T, DIM > Class Template Reference

Collaboration diagram for Variable< T, DIM >:



Public Member Functions

- template<class... Args>
  Variable (SpatialDiscretization< T, DIM > *spatial, const BoundaryConditions< T, DIM > &bcs, const std::string &variable_name, const std::string &type, const std::string &initial_condition_name, std::tuple< Args... > args1)

  Construct a new Variable:: Variable object.
- template<class... Args1, class... Args2>
  Variable (SpatialDiscretization< T, DIM > *spatial, const BoundaryConditions< T, DIM > &bcs, const std::string &variable_name, const std::string &type, const std::string &initial_condition_name, std::tuple< Args1... > args1, const std::string &analytical_solution_name, std::tuple< Args2... > args2)

  Construct a new Variable:: Variable object.
- template<class... Args>
  Variable (SpatialDiscretization< T, DIM > *spatial, const BoundaryConditions< T, DIM > &bcs, const std::string &variable_name, const std::string &type, const std::string &initial_condition_name, std::tuple< Args... > args1, const mfem::FunctionCoefficient &analytical_solution_function)

  Construct a new Variable:: Variable object.
- Variable (SpatialDiscretization< T, DIM > *spatial, const BoundaryConditions< T, DIM > &bcs, const std::string &variable_name, const std::string &type, const mfem::FunctionCoefficient &initial_condition_function)

  Construct a new Variable<T>:: Variable object.
- template<class... Args>
  Variable (SpatialDiscretization< T, DIM > *spatial, const BoundaryConditions< T, DIM > &bcs, const std::string &variable_name, const std::string &type, const mfem::FunctionCoefficient &initial_condition_function, const std::string &analytical_solution_name, std::tuple< Args... > args1)

  Construct a new Variable<T>:: Variable object.

- Variable (SpatialDiscretization< T, DIM > *spatial, const BoundaryConditions< T, DIM > &bcs, const std::string &variable_name, const std::string &type, const mfem::FunctionCoefficient &initial_condition_function, const mfem::FunctionCoefficient &analytical_solution_function)

  Construct a new Variable<T>:: Variable object.
- Variable (SpatialDiscretization< T, DIM > *spatial, const BoundaryConditions< T, DIM > &bcs, const std::string &variable_name, const std::string &type, const double &initial_condition_value)

  Construct a new Variable<T>:: Variable object.
- template<class... Args>
  Variable (SpatialDiscretization< T, DIM > *spatial, const BoundaryConditions< T, DIM > &bcs, const std::string &variable_name, const std::string &type, const double &initial_condition_value, const std::string &analytical_solution_name, std::tuple< Args... > args1)

  Construct a new Variable<T>:: Variable object.
- Variable (SpatialDiscretization< T, DIM > *spatial, const BoundaryConditions< T, DIM > &bcs, const std::string &variable_name, const std::string &type, const double &initial_condition_value, const mfem::FunctionCoefficient &analytical_solution_function)

  Construct a new Variable<T>:: Variable object.
- std::string getVariableName () const

  Get the name of the Variable.
- VariableType::value getVariableType ()

  Get the Type of the Variable.
- std::shared_ptr< AnalyticalFunctions< DIM > > **getInitialCondition** ()
- void update (const mfem::Vector &unk)

  update the GridFunction on the basis of its associated unknown vector
- mfem::Vector get_unknown ()

  return the unkown vector
- mfem::GridFunction get_gf () const

  return the gridfunction associated to the unknown
- mfem::GridFunction **get_igf** () const
- mfem::GridFunction get_analytical_solution ()

  return the gridfunction associated to the analytical solution
- BoundaryConditions< T, DIM > * get_boundary_conditions ()

  return the boundary condition object associated to the variable
- ∼Variable ()

  Destroy the Variable:: Variable object.

14.63.1 Detailed Description

template<class T, int DIM>
class Variable< T, DIM >

Definition at line 24 of file Variable.hpp.

14.63.2 Constructor & Destructor Documentation

14.63.2.1 Variable() [1/9]

```
template<class T , int DIM>
template<class... Args>
Variable< T, DIM >::Variable (
            SpatialDiscretization< T, DIM > * spatial,
            const BoundaryConditions< T, DIM > & bcs,
            const std::string & variable_name,
            const std::string & type,
            const std::string & initial_condition_name,
            std::tuple< Args... > args1 )
```

Construct a new Variable:: Variable object.

Parameters

fespace	
variable_name	
type	
initial_condition_name	

Definition at line 122 of file Variable.hpp.

References SpatialDiscretization< T, DIM >::get_dimension(), and SpatialDiscretization< T, DIM >::get_finite_↩ element_space().

```
  : bcs_(bcs), variable_name_(variable_name) {
this->fespace_ = spatial->get_finite_element_space();
this->setVariableType(type);

this->uh_.SetSpace(fespace_);
const auto dim = spatial->get_dimension();

std::apply(
    [dim, initial_condition_name, this](Args... args) {
      Variable<T, DIM>::setInitialCondition(dim, initial_condition_name, args...);
    },
    args1);
// mfem::ConstantCoefficient cc(0.);
// this->uh_.ProjectCoefficient(cc);
// this->uh_.GetTrueDofs(this->unk_);
}
```

14.63.2.2 Variable() [2/9]

```
template<class T , int DIM>
template<class... Args1, class... Args2>
Variable< T, DIM >::Variable (
            SpatialDiscretization< T, DIM > * spatial,
            const BoundaryConditions< T, DIM > & bcs,
            const std::string & variable_name,
            const std::string & type,
            const std::string & initial_condition_name,
            std::tuple< Args1... > args1,
            const std::string & analytical_solution_name,
            std::tuple< Args2... > args2 )
```

Construct a new Variable:: Variable object.

Parameters

fespace	
variable_name	
type	
initial_condition_name	
analytical_solution_name	

Definition at line 154 of file Variable.hpp.

References SpatialDiscretization< T, DIM >::get_dimension(), and SpatialDiscretization< T, DIM >::get_finite_↩
element_space().

```
  : bcs_(bcs), variable_name_(variable_name) {
this->fespace_ = spatial->get_finite_element_space();
this->setVariableType(type);

this->uh_.SetSpace(fespace_);
this->uh_ex_.SetSpace(fespace_);
const auto dim = spatial->get_dimension();
std::apply(
    [dim, initial_condition_name, this](Args1... args) {
      this->setInitialCondition(dim, initial_condition_name, args...);
    },
    args1);
std::apply(
    [dim, analytical_solution_name, this](Args2... args) {
      this->setAnalyticalSolution(dim, analytical_solution_name, args...);
    },
    args2);
}
```

14.63.2.3 Variable() [3/9]

```
template<class T , int DIM>
template<class... Args>
Variable< T, DIM >::Variable (
            SpatialDiscretization< T, DIM > * spatial,
            const BoundaryConditions< T, DIM > & bcs,
            const std::string & variable_name,
            const std::string & type,
            const std::string & initial_condition_name,
            std::tuple< Args... > args1,
            const mfem::FunctionCoefficient & analytical_solution_function )
```

Construct a new Variable:: Variable object.

Parameters

fespace	
variable_name	
type	
initial_condition_name	
analytical_solution_function	

Definition at line 189 of file Variable.hpp.

References SpatialDiscretization< T, DIM >::get_dimension(), and SpatialDiscretization< T, DIM >::get_finite_element_space().

```
  : bcs_(bcs), variable_name_(variable_name) {
this->fespace_ = spatial->get_finite_element_space();
this->setVariableType(type);

this->uh_.SetSpace(fespace_);

this->uh_ex_.SetSpace(fespace_);
const auto dim = spatial->get_dimension();
std::apply([dim, initial_condition_name, this](
               Args... args) { this->setInitialCondition(dim, initial_condition_name, args...); },
           args1);
this->setAnalyticalSolution(analytical_solution_function);
}
```

14.63.2.4 Variable() [4/9]

```
template<class T , int DIM>
Variable< T, DIM >::Variable (
            SpatialDiscretization< T, DIM > * spatial,
            const BoundaryConditions< T, DIM > & bcs,
            const std::string & variable_name,
            const std::string & type,
            const mfem::FunctionCoefficient & initial_condition_function )
```

Construct a new Variable<T>:: Variable object.

Parameters

fespace	
variable_name	
type	
initial_condition_function	

Definition at line 217 of file Variable.hpp.

References SpatialDiscretization< T, DIM >::get_finite_element_space().

```
  : bcs_(bcs), variable_name_(variable_name) {
this->fespace_ = spatial->get_finite_element_space();
this->setVariableType(type);

this->uh_.SetSpace(fespace_);
this->setInitialCondition(initial_condition_function);
}
```

14.63.2.5 Variable() [5/9]

```
template<class T , int DIM>
template<class... Args>
Variable< T, DIM >::Variable (
            SpatialDiscretization< T, DIM > * spatial,
            const BoundaryConditions< T, DIM > & bcs,
            const std::string & variable_name,
            const std::string & type,
            const mfem::FunctionCoefficient & initial_condition_function,
            const std::string & analytical_solution_name,
            std::tuple< Args... > args1 )
```

Construct a new Variable<T>:: Variable object.

Parameters

fespace	
variable_name	
type	
initial_condition_function	
analytical_solution_name	

Definition at line 240 of file Variable.hpp.

References SpatialDiscretization< T, DIM >::get_dimension(), and SpatialDiscretization< T, DIM >::get_finite_element_space().

```
  : bcs_(bcs), variable_name_(variable_name) {
this->fespace_ = spatial->get_finite_element_space();
this->setVariableType(type);

this->uh_.SetSpace(fespace_);
this->uh_ex_.SetSpace(fespace_);
const auto dim = spatial->get_dimension();
this->setInitialCondition(initial_condition_function);
std::apply(
    [dim, analytical_solution_name, this](Args... args) {
      this->setAnalyticalSolution(dim, analytical_solution_name, args...);
    },
    args1);
}
```

14.63.2.6 Variable() [6/9]

```
template<class T , int DIM>
Variable< T, DIM >::Variable (
            SpatialDiscretization< T, DIM > * spatial,
            const BoundaryConditions< T, DIM > & bcs,
            const std::string & variable_name,
            const std::string & type,
            const mfem::FunctionCoefficient & initial_condition_function,
            const mfem::FunctionCoefficient & analytical_solution_function )
```

Construct a new Variable<T>:: Variable object.

Parameters

fespace	
variable_name	
type	
initial_condition_function	
analytical_solution_function	

Definition at line 270 of file Variable.hpp.

References SpatialDiscretization< T, DIM >::get_finite_element_space().

```
    : bcs_(bcs), variable_name_(variable_name) {
  this->fespace_ = spatial->get_finite_element_space();
  this->setVariableType(type);

  this->uh_.SetSpace(fespace_);
  this->uh_ex_.SetSpace(fespace_);
  this->setInitialCondition(initial_condition_function);
  this->setAnalyticalSolution(analytical_solution_function);
}
```

14.63.2.7 Variable() [7/9]

```
template<class T , int DIM>
Variable< T, DIM >::Variable (
            SpatialDiscretization< T, DIM > * spatial,
            const BoundaryConditions< T, DIM > & bcs,
            const std::string & variable_name,
            const std::string & type,
            const double & initial_condition_value )
```

Construct a new Variable<T>:: Variable object.

Parameters

fespace	
variable_name	
type	
initial_condition_value	

Definition at line 294 of file Variable.hpp.

References SpatialDiscretization< T, DIM >::get_finite_element_space().

```
    : bcs_(bcs), variable_name_(variable_name) {
  this->fespace_ = spatial->get_finite_element_space();
  this->setVariableType(type);
  this->uh_.SetSpace(fespace_);
  this->setInitialCondition(initial_condition_value);
}
```

14.63.2.8 Variable() [8/9]

```
template<class T , int DIM>
template<class... Args>
Variable< T, DIM >::Variable (
            SpatialDiscretization< T, DIM > * spatial,
            const BoundaryConditions< T, DIM > & bcs,
            const std::string & variable_name,
            const std::string & type,
            const double & initial_condition_value,
            const std::string & analytical_solution_name,
            std::tuple< Args... > args1 )
```

Construct a new Variable<T>:: Variable object.

Parameters

fespace	
variable_name	
type	
initial_condition_value	
analytical_solution_name	

Definition at line 314 of file Variable.hpp.

References SpatialDiscretization< T, DIM >::get_dimension(), and SpatialDiscretization< T, DIM >::get_finite_↩ element_space().

```
    : bcs_(bcs), variable_name_(variable_name) {
  this->fespace_ = spatial->get_finite_element_space();
  this->setVariableType(type);

  this->uh_.SetSpace(fespace_);
  this->uh_ex_.SetSpace(fespace_);
  const auto dim = spatial->get_dimension();
  this->setInitialCondition(initial_condition_value);
  std::apply(
      [dim, analytical_solution_name, this](Args... args) {
        this->setAnalyticalSolution(dim, analytical_solution_name, args...);
      },
      args1);
}
```

14.63.2.9 Variable() [9/9]

```
template<class T , int DIM>
Variable< T, DIM >::Variable (
            SpatialDiscretization< T, DIM > * spatial,
            const BoundaryConditions< T, DIM > & bcs,
            const std::string & variable_name,
            const std::string & type,
            const double & initial_condition_value,
            const mfem::FunctionCoefficient & analytical_solution_function )
```

Construct a new Variable<T>:: Variable object.

Parameters

fespace	
variable_name	
type	
initial_condition_value	
analytical_solution_function	

Definition at line 343 of file Variable.hpp.

References SpatialDiscretization< T, DIM >::get_finite_element_space().

```
  : bcs_(bcs), variable_name_(variable_name) {
this->fespace_ = spatial->get_finite_element_space();
this->setVariableType(type);

this->uh_.SetSpace(fespace_);
this->uh_ex_.SetSpace(fespace_);
this->setInitialCondition(initial_condition_value);
this->setAnalyticalSolution(analytical_solution_function);
}
```

14.63.3 Member Function Documentation

14.63.3.1 get_analytical_solution()

```
template<class T , int DIM>
mfem::GridFunction Variable< T, DIM >::get_analytical_solution ( )
```

return the gridfunction associated to the analytical solution

Returns

mfem::GridFunction

Definition at line 480 of file Variable.hpp.

```
480                                                             {
481   return this->uh_ex_;
482 }
```

14.63.3.2 get_boundary_conditions()

```
template<class T , int DIM>
BoundaryConditions< T, DIM > * Variable< T, DIM >::get_boundary_conditions ( )
```

return the boundary condition object associated to the variable

Template Parameters

T	
DIM	

Returns

BoundaryConditions<T, DIM>

Definition at line 492 of file Variable.hpp.

Referenced by PhaseFieldOperator< T, DIM >::initialize().

```
492                                                                        {
493   return &this->bcs_;
494 }
```

14.63.3.3 get_gf()

```
template<class T , int DIM>
mfem::GridFunction Variable< T, DIM >::get_gf ( ) const
```

return the gridfunction associated to the unknown

Returns

mfem::GridFunction

Definition at line 470 of file Variable.hpp.

```
470                                                   {
471   return this->uh_;
472 }
```

14.63.3.4 get_unknown()

```
template<class T , int DIM>
mfem::Vector Variable< T, DIM >::get_unknown ( )
```

return the unkown vector

Returns

mfem::Vector

Definition at line 460 of file Variable.hpp.

Referenced by TimeDiscretization< T, DC, DIM >::execute(), and PhaseFieldOperator< T, DIM >::initialize().

```
460                                            {
461   return this->unk_;
462 }
```

14.63.3.5 getVariableName()

```
template<class T , int DIM>
std::string Variable< T, DIM >::getVariableName ( ) const
```

Get the name of the Variable.

Returns

std::string name of the variable

Definition at line 502 of file Variable.hpp.

```
502                                                 {
503   return this->variable_name_;
504 }
```

14.63.3.6 getVariableType()

```
template<class T , int DIM>
VariableType::value Variable< T, DIM >::getVariableType ( )
```

Get the Type of the Variable.

Returns

VariableType::value type of the variable

Definition at line 512 of file Variable.hpp.

```
512                                                   {
513   return this->variable_type_;
514 }
```

The documentation for this class was generated from the following file:

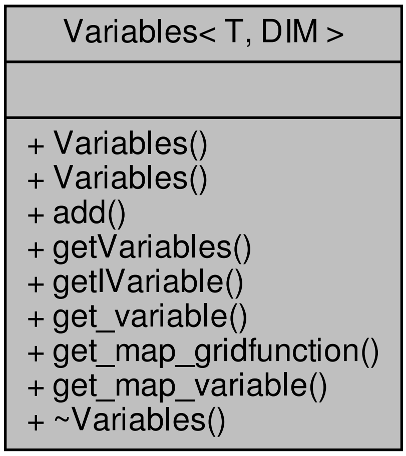
- Variable.hpp

14.64 Variables< T, DIM > Class Template Reference

Class used to manage a list of Variable.

```
#include </home/ci230846/home-local/MyGitProjects/COMPONENT/PF-MFEM/Variables/↩
Variables.hpp>
```

Collaboration diagram for Variables< T, DIM >:



Public Member Functions

- template<class... Args>
  Variables (Args... args)

  Construct a new Variables:: Variables object.
- Variables ()

  Construct a new Variables:: Variables object.
- void add (Variable< T, DIM > var)

  Add a new variable.
- std::vector< Variable< T, DIM > > getVariables () const

  get vector of variables
- Variable< T, DIM > & getIVariable (const int &i)

  get i-th variable
- Variable< T, DIM > & get_variable (const std::string &name)

  return the variable called vname
- std::map< std::string, mfem::GridFunction > get_map_gridfunction () const

  return a map of GridFunction for each variable name
- std::map< std::string, Variable< T, DIM > > get_map_variable () const

  return a map of variables for each variable name
- ∼Variables ()

  Destroy the Variables:: Variables object.

14.64.1 Detailed Description

template<class T, int DIM>
class Variables< T, DIM >

Class used to manage a list of Variable.

Definition at line 28 of file Variables.hpp.

14.64.2 Constructor & Destructor Documentation

14.64.2.1 Variables()

```
template<class T , int DIM>
template<class... Args>
Variables< T, DIM >::Variables (
            Args... args ) [explicit]
```

Construct a new Variables:: Variables object.

Template Parameters

Args	

Parameters

args	

Definition at line 63 of file Variables.hpp.

```
63                                         {
64   this->vect_variables_ = std::vector<Variable<T, DIM>>{args...};
65 }
```

14.64.3 Member Function Documentation

14.64.3.1 add()

```
template<class T , int DIM>
void Variables< T, DIM >::add (
            Variable< T, DIM > var )
```

Add a new variable.

Parameters

var	variable to add

Definition at line 73 of file Variables.hpp.

```
73                                              {
74   this->vect_variables_.emplace_back(var);
75 }
```

14.64.3.2 get_map_gridfunction()

```
template<class T , int DIM>
std::map< std::string, mfem::GridFunction > Variables< T, DIM >::get_map_gridfunction ( )
const
```

return a map of GridFunction for each variable name

Returns

std::map<std::string, mfem::GridFunction>

Definition at line 104 of file Variables.hpp.

Referenced by PostProcessing< T, DC, DIM >::save_variables().

```
104                                                                                  {
105   std::map<std::string, mfem::GridFunction> map_var;
106   for (auto vv : this->vect_variables_) {
107     const std::string& name = vv.getVariableName();
108     auto gf = vv.get_gf();
109     map_var.try_emplace(name, gf);
110   }
111   return map_var;
112 }
```

14.64.3.3 get_map_variable()

```
template<class T , int DIM>
std::map< std::string, Variable< T, DIM > > Variables< T, DIM >::get_map_variable ( ) const
```

return a map of variables for each variable name

Returns

std::map<std::string, Variable>

Definition at line 119 of file Variables.hpp.

```
119                                                                              {
120   std::map<std::string, Variable<T, DIM>> map_var;
121   for (const auto& vv : this->vect_variables_) {
122     const std::string& name = vv.getVariableName();
123     map_var.try_emplace(name, vv);
124   }
125   return map_var;
126 }
```

14.64.3.4 get_variable()

```
template<class T , int DIM>
Variable< T, DIM > & Variables< T, DIM >::get_variable (
              const std::string & vname )
```

return the variable called vname

Parameters

vname	

Returns

Variable&

Definition at line 135 of file Variables.hpp.

Referenced by PhaseFieldOperator< T, DIM >::PhaseFieldOperator().

```
135                                                                     {
136   int id = 0;
137   const auto vect_size = static_cast<int>(this->vect_variables_.size());
138   for (auto i = 0; i < vect_size; i++) {
139     const std::string& name = this->vect_variables_[i].getVariableName();
140     if (name == vname) {
141       id = i;
142       break;
143     }
144   }
145   return this->vect_variables_[id];
146 }
```

14.64.3.5 getIVariable()

```
template<class T , int DIM>
Variable< T, DIM > & Variables< T, DIM >::getIVariable (
              const int & i )
```

get i-th variable

Parameters

i	

Returns

Variable&

Definition at line 94 of file Variables.hpp.

```
94                                                            {
95   return this->vect_variables_[i];
96 }
```

14.64.3.6 getVariables()

```
template<class T , int DIM>
std::vector< Variable< T, DIM > > Variables< T, DIM >::getVariables ( ) const
```

get vector of variables

Returns

std::vector<Variable>

Definition at line 83 of file Variables.hpp.

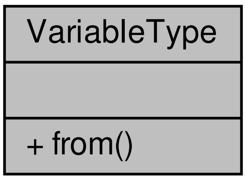
```
83                                                                    {
84   return this->vect_variables_;
85 }
```

The documentation for this class was generated from the following file:

- Variables.hpp

14.65 VariableType Struct Reference

Collaboration diagram for VariableType:



Public Types

- enum **value** { **Conserved**, **Unconserved** }

Static Public Member Functions

- static value **from** (const std::string &)

14.65.1 Detailed Description

Definition at line 63 of file Utils/PhaseFieldOptions.hpp.

The documentation for this struct was generated from the following file:

- Utils/PhaseFieldOptions.hpp

Index