

ASTEROPE

0.1

Generated by Doxygen 1.8.13

Contents

1	Main Page	1
1.1	Getting Started	1
1.1.1	Installation guide	1
1.1.2	Basics features for building phase-field applications	1
1.1.3	User examples	1
1.1.4	Doxygen documentation	1
1.1.5	Licence	2
1.1.6	Contact	2
1.1.7	Contributors	2
2	installation	3
3	Code overview	5
4	How to build of a phase-field application?	7
5	Diffusion problems	9
5.1	1D Problem	9
6	Allen-Cahn problems	11
6.1	1D Problem	11
7	Cahn-Hilliard problems	13
7.1	1D Problem	13
8	SpatialDiscretization	15
8.1	Description	15

9 Variables	17
10 Namespace Index	19
10.1 Namespace List	19
11 Hierarchical Index	21
11.1 Class Hierarchy	21
12 Data Structure Index	23
12.1 Data Structures	23
13 Namespace Documentation	25
13.1 MassDefaultConstant Namespace Reference	25
13.1.1 Detailed Description	25
13.2 NewtonDefaultConstant Namespace Reference	25
13.2.1 Detailed Description	25
14 Data Structure Documentation	27
14.1 AllenCahnSpecializedNLFormIntegrator< SCHEME > Class Template Reference	27
14.1.1 Detailed Description	28
14.1.2 Constructor & Destructor Documentation	28
14.1.2.1 AllenCahnSpecializedNLFormIntegrator()	29
14.1.3 Member Function Documentation	29
14.1.3.1 AssembleElementGrad()	29
14.1.3.2 AssembleElementVector()	30
14.2 AnalyticalFunctions< DIM > Class Template Reference	32
14.2.1 Detailed Description	32
14.2.2 Member Function Documentation	32
14.2.2.1 getAnalyticalFunctions()	33
14.3 AnalyticalFunctionsType Struct Reference	33
14.3.1 Detailed Description	34
14.4 Boundaries Class Reference	34
14.4.1 Detailed Description	34

14.5 Boundary Class Reference	35
14.5.1 Detailed Description	35
14.5.2 Constructor & Destructor Documentation	35
14.5.2.1 Boundary()	35
14.5.3 Member Function Documentation	36
14.5.3.1 get_boundary_index()	36
14.5.3.2 get_boundary_value()	37
14.5.3.3 is_essential_boundary()	37
14.6 BoundaryConditions< T, DIM > Class Template Reference	37
14.6.1 Detailed Description	38
14.6.2 Constructor & Destructor Documentation	38
14.6.2.1 BoundaryConditions()	38
14.6.3 Member Function Documentation	39
14.6.3.1 GetEssentialDofs()	39
14.6.3.2 SetBoundaryConditions()	39
14.7 BoundaryConditionType Struct Reference	40
14.7.1 Detailed Description	41
14.8 CahnHilliardSpecializedNLFormIntegrator< SCHEME > Class Template Reference	41
14.8.1 Detailed Description	42
14.8.2 Constructor & Destructor Documentation	42
14.8.2.1 CahnHilliardSpecializedNLFormIntegrator()	43
14.8.3 Member Function Documentation	43
14.8.3.1 AssembleElementGrad()	43
14.8.3.2 AssembleElementVector()	44
14.9 ConductionOperator Class Reference	47
14.9.1 Detailed Description	49
14.9.2 Member Function Documentation	49
14.9.2.1 ImplicitSolve()	49
14.10 DiffusionNLIntegrator Class Reference	50
14.10.1 Detailed Description	52

14.10.2 Constructor & Destructor Documentation	52
14.10.2.1 DiffusionNLFItegrator() [1/5]	52
14.10.2.2 DiffusionNLFItegrator() [2/5]	52
14.10.2.3 DiffusionNLFItegrator() [3/5]	53
14.10.2.4 DiffusionNLFItegrator() [4/5]	53
14.10.2.5 DiffusionNLFItegrator() [5/5]	53
14.11 EnergyCoefficient Class Reference	54
14.11.1 Detailed Description	55
14.12 InterfacialCoefficient Class Reference	55
14.12.1 Detailed Description	56
14.13 MeltingCoefficient Class Reference	57
14.13.1 Detailed Description	58
14.14 Meshes Struct Reference	58
14.14.1 Detailed Description	58
14.15 PhaseFieldPrivate::mmap< EType > Struct Template Reference	59
14.15.1 Detailed Description	60
14.16 MobilityCoefficient Class Reference	60
14.16.1 Detailed Description	61
14.17 multidimension_function< DIM > Struct Template Reference	61
14.17.1 Detailed Description	62
14.18 multidimension_function< 1 > Struct Template Reference	62
14.18.1 Detailed Description	62
14.19 multidimension_function< 2 > Struct Template Reference	63
14.19.1 Detailed Description	63
14.20 multidimension_function< 3 > Struct Template Reference	64
14.20.1 Detailed Description	64
14.21 my_best_constructor< T, DIM > Struct Template Reference	65
14.21.1 Detailed Description	65
14.22 my_best_constructor< T, 1 > Struct Template Reference	65
14.22.1 Detailed Description	66

14.23my_best_constructor< T, 2 > Struct Template Reference	66
14.23.1 Detailed Description	67
14.24my_best_constructor< T, 3 > Struct Template Reference	67
14.24.1 Detailed Description	67
14.25NonlinearCoefficient Class Reference	68
14.25.1 Detailed Description	69
14.26Parameter Class Reference	70
14.26.1 Detailed Description	70
14.26.2 Member Function Documentation	70
14.26.2.1 getName()	70
14.26.2.2 getValue()	71
14.27Parameters Class Reference	71
14.27.1 Detailed Description	72
14.27.2 Constructor & Destructor Documentation	72
14.27.2.1 Parameters()	72
14.27.3 Member Function Documentation	72
14.27.3.1 add()	72
14.27.3.2 get_parameter_value()	73
14.27.3.3 getMapParameters()	73
14.28PhaseFieldOperator< T, DIM, NLFI > Class Template Reference	74
14.28.1 Detailed Description	77
14.28.2 Constructor & Destructor Documentation	77
14.28.2.1 PhaseFieldOperator()	77
14.28.3 Member Function Documentation	78
14.28.3.1 ImplicitSolve()	78
14.28.3.2 initialize()	79
14.28.3.3 Mult()	79
14.28.3.4 PhaseFieldEnergy()	80
14.28.3.5 SetConstantParameters()	81
14.28.3.6 SetTransientParameters()	81

14.28.4 Field Documentation	82
14.28.4.1 reduced_oper	82
14.29PhaseFieldReducedOperator Class Reference	83
14.29.1 Detailed Description	84
14.30PoissonNLFIegrator Class Reference	84
14.30.1 Detailed Description	85
14.31PostProcessing< T, DC, DIM > Class Template Reference	86
14.31.1 Detailed Description	87
14.31.2 Constructor & Destructor Documentation	88
14.31.2.1 PostProcessing()	88
14.31.3 Member Function Documentation	88
14.31.3.1 get_frequency()	88
14.31.3.2 need_to_be_saved()	89
14.31.3.3 save_variables()	89
14.32potential_function< ORDER, SCHEME > Struct Template Reference	90
14.32.1 Detailed Description	90
14.33potential_function< 0, ThermodynamicsPotentialDiscretization::Explicit > Struct Template Reference	91
14.33.1 Detailed Description	91
14.33.2 Member Function Documentation	91
14.33.2.1 getH()	91
14.33.2.2 getW()	92
14.33.2.3 getX()	93
14.34potential_function< 0, ThermodynamicsPotentialDiscretization::Implicit > Struct Template Reference	93
14.34.1 Detailed Description	94
14.34.2 Member Function Documentation	94
14.34.2.1 getH()	94
14.34.2.2 getW()	95
14.34.2.3 getX()	95
14.35potential_function< 0, ThermodynamicsPotentialDiscretization::SemImplicit > Struct Template Reference	96
14.35.1 Detailed Description	97

14.35.2 Member Function Documentation	97
14.35.2.1 getH()	97
14.35.2.2 getW()	97
14.35.2.3 getX()	98
14.36potential_function< 1, ThermodynamicsPotentialDiscretization::Explicit > Struct Template Reference	99
14.36.1 Detailed Description	99
14.36.2 Member Function Documentation	99
14.36.2.1 getH()	100
14.36.2.2 getW()	100
14.36.2.3 getX()	101
14.37potential_function< 1, ThermodynamicsPotentialDiscretization::Implicit > Struct Template Reference	102
14.37.1 Detailed Description	102
14.37.2 Member Function Documentation	102
14.37.2.1 getH()	102
14.37.2.2 getW()	103
14.37.2.3 getX()	104
14.38potential_function< 1, ThermodynamicsPotentialDiscretization::SemImplicit > Struct Template Reference	104
14.38.1 Detailed Description	105
14.38.2 Member Function Documentation	105
14.38.2.1 getH()	105
14.38.2.2 getW()	106
14.38.2.3 getX()	106
14.39potential_function< 2, ThermodynamicsPotentialDiscretization::Explicit > Struct Template Reference	107
14.39.1 Detailed Description	108
14.39.2 Member Function Documentation	108
14.39.2.1 getH()	108
14.39.2.2 getW()	109
14.39.2.3 getX()	109
14.40potential_function< 2, ThermodynamicsPotentialDiscretization::Implicit > Struct Template Reference	110
14.40.1 Detailed Description	111

14.40.2 Member Function Documentation	111
14.40.2.1 getH()	111
14.40.2.2 getW()	111
14.40.2.3 getX()	112
14.41 potential_function< 2, ThermodynamicsPotentialDiscretization::SemiImplicit > Struct Template Reference	113
14.41.1 Detailed Description	113
14.41.2 Member Function Documentation	113
14.41.2.1 getH()	114
14.41.2.2 getW()	114
14.41.2.3 getX()	115
14.42 PotentialFunctions< ORDER, SCHEME > Class Template Reference	116
14.42.1 Detailed Description	117
14.42.2 Member Function Documentation	117
14.42.2.1 getPotentialFunction()	117
14.43 Problem Class Reference	118
14.43.1 Detailed Description	119
14.44 Problems Struct Reference	119
14.44.1 Detailed Description	119
14.45 SpatialDiscretization< T, DIM > Class Template Reference	120
14.45.1 Detailed Description	121
14.45.2 Constructor & Destructor Documentation	121
14.45.2.1 ~SpatialDiscretization()	121
14.45.3 Member Function Documentation	121
14.45.3.1 apply_uniform_refinement()	122
14.45.3.2 get_dimension()	122
14.45.3.3 get_finite_element_space()	123
14.45.3.4 get_max_bdr_attributes()	123
14.45.3.5 get_mesh()	124
14.45.3.6 getSize()	124
14.45.3.7 make_periodic_mesh()	124

14.45.3.8 set_finite_element_space()	125
14.46ThermodynamicRootFindingAlgorithm Struct Reference	126
14.46.1 Detailed Description	126
14.47ThermodynamicsAllenCahnMobility Struct Reference	126
14.47.1 Detailed Description	127
14.48ThermodynamicsFGR Struct Reference	127
14.48.1 Detailed Description	127
14.49ThermodynamicsInitialization Struct Reference	128
14.49.1 Detailed Description	128
14.50ThermodynamicsLinearPowerSourceTerm Struct Reference	128
14.50.1 Detailed Description	129
14.51ThermodynamicsModel Struct Reference	129
14.51.1 Detailed Description	130
14.52ThermodynamicsMolarVolumeType Struct Reference	130
14.52.1 Detailed Description	130
14.53ThermodynamicsPotentials Struct Reference	131
14.53.1 Detailed Description	131
14.54ThermodynamicsProperties Struct Reference	132
14.54.1 Detailed Description	132
14.55ThermodynamicsStabilizationScheme Struct Reference	132
14.55.1 Detailed Description	133
14.56ThermodynamicSubModels Struct Reference	133
14.56.1 Detailed Description	134
14.57ThermodynamicsUnitForSettingConditions Struct Reference	134
14.57.1 Detailed Description	134
14.58ThermodynamicsUpdateQuantitiesMethod Struct Reference	135
14.58.1 Detailed Description	135
14.59ThermodynamicSystemsForMobilities Struct Reference	135
14.59.1 Detailed Description	136
14.60TimeDiscretization< T, DC, DIM, NLFI > Class Template Reference	136

14.60.1 Detailed Description	137
14.60.2 Constructor & Destructor Documentation	137
14.60.2.1 TimeDiscretization()	137
14.61 TimeScheme Struct Reference	138
14.61.1 Detailed Description	138
14.62 UtilsForSolvers Class Reference	139
14.62.1 Detailed Description	139
14.62.2 Member Function Documentation	139
14.62.2.1 BuildSolver()	139
14.62.2.2 SetSolverParameters()	140
14.63 Variable< T, DIM > Class Template Reference	141
14.63.1 Detailed Description	142
14.63.2 Constructor & Destructor Documentation	142
14.63.2.1 Variable() [1/9]	143
14.63.2.2 Variable() [2/9]	143
14.63.2.3 Variable() [3/9]	144
14.63.2.4 Variable() [4/9]	145
14.63.2.5 Variable() [5/9]	146
14.63.2.6 Variable() [6/9]	146
14.63.2.7 Variable() [7/9]	147
14.63.2.8 Variable() [8/9]	148
14.63.2.9 Variable() [9/9]	148
14.63.3 Member Function Documentation	149
14.63.3.1 get_analytical_solution()	149
14.63.3.2 get_boundary_conditions()	149
14.63.3.3 get_gf()	150
14.63.3.4 get_unknown()	150
14.63.3.5 getVariableName()	151
14.63.3.6 getVariableType()	151
14.64 Variables< T, DIM > Class Template Reference	152
14.64.1 Detailed Description	153
14.64.2 Constructor & Destructor Documentation	153
14.64.2.1 Variables()	153
14.64.3 Member Function Documentation	153
14.64.3.1 add()	153
14.64.3.2 get_map_gridfunction()	154
14.64.3.3 get_map_variable()	154
14.64.3.4 get_variable()	155
14.64.3.5 getlVariable()	155
14.64.3.6 getVariables()	156
14.65 VariableType Struct Reference	156
14.65.1 Detailed Description	156

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:

MassDefaultConstant	
Default constant used by Mass Solver	25
NewtonDefaultConstant	
Defaukt constant used by Newton Solver	25

Chapter 11

Hierarchical Index

11.1 Class Hierarchy

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

AnalyticalFunctions< DIM >	32
AnalyticalFunctionsType	33
Boundaries	34
Boundary	35
BoundaryConditions< T, DIM >	37
BoundaryConditionType	40
Coefficient	
EnergyCoefficient	54
InterfacialCoefficient	55
MeltingCoefficient	57
MobilityCoefficient	60
DC	
PostProcessing< T, DC, DIM >	86
GridFunctionCoefficient	
NonlinearCoefficient	68
Meshes	58
multidimension_function< DIM >	61
multidimension_function< 1 >	62
multidimension_function< 2 >	63
multidimension_function< 3 >	64
my_best_constructor< T, DIM >	65
my_best_constructor< T, 1 >	65
my_best_constructor< T, 2 >	66
my_best_constructor< T, 3 >	67
NonlinearFormIntegrator	
AllenCahnSpecializedNLFormIntegrator< SCHEME >	27
CahnHilliardSpecializedNLFormIntegrator< SCHEME >	41
DiffusionNLFIntegrator	50
PoissonNLFIntegrator	84
Operator	
PhaseFieldReducedOperator	83
Parameter	70
Parameters	71
potential_function< ORDER, SCHEME >	90
potential_function< 0, ThermodynamicsPotentialDiscretization::Explicit >	91

potential_function< 0, ThermodynamicsPotentialDiscretization::Implicit >	93
potential_function< 0, ThermodynamicsPotentialDiscretization::SemImplicit >	96
potential_function< 1, ThermodynamicsPotentialDiscretization::Explicit >	99
potential_function< 1, ThermodynamicsPotentialDiscretization::Implicit >	102
potential_function< 1, ThermodynamicsPotentialDiscretization::SemImplicit >	104
potential_function< 2, ThermodynamicsPotentialDiscretization::Explicit >	107
potential_function< 2, ThermodynamicsPotentialDiscretization::Implicit >	110
potential_function< 2, ThermodynamicsPotentialDiscretization::SemImplicit >	113
PotentialFunctions< ORDER, SCHEME >	116
PotentialFunctions< 1, SCHEME >	116
PotentialFunctions< 2, SCHEME >	116
Problem	118
Problems	119
SpatialDiscretization< T, DIM >	120
ThermodynamicRootFindingAlgorithm	126
ThermodynamicsAllenCahnMobility	126
ThermodynamicsFGR	127
ThermodynamicsInitialization	128
ThermodynamicsLinearPowerSourceTerm	128
ThermodynamicsModel	129
ThermodynamicsMolarVolumeType	130
ThermodynamicsPotentials	131
ThermodynamicsProperties	132
ThermodynamicsStabilizationScheme	132
ThermodynamicSubModels	133
ThermodynamicsUnitForSettingConditions	134
ThermodynamicsUpdateQuantitiesMethod	135
ThermodynamicSystemsForMobilities	135
TimeDependentOperator	
ConductionOperator	47
PhaseFieldOperator< T, DIM, NLFI >	74
PhaseFieldOperator< T, DIM >	74
TimeDiscretization< T, DC, DIM, NLFI >	136
TimeDiscretization< T, DC, DIM >	136
TimeScheme	138
UtilsForSolvers	139
Variable< T, DIM >	141
Variables< T, DIM >	152
VariableType	156
vector	
PhaseFieldPrivate::mmap< EType >	59

Chapter 12

Data Structure Index

12.1 Data Structures

Here are the data structures with brief descriptions:

AllenCahnSpecializedNLFormIntegrator< SCHEME >	27
AnalyticalFunctions< DIM >	32
AnalyticalFunctionsType	33
Boundaries	34
Boundary	35
BoundaryConditions< T, DIM >	
Class used to manage boundary conditions	37
BoundaryConditionType	40
CahnHilliardSpecializedNLFormIntegrator< SCHEME >	41
ConductionOperator	47
DiffusionNLFIIntegrator	50
EnergyCoefficient	54
InterfacialCoefficient	55
MeltingCoefficient	57
Meshes	58
PhaseFieldPrivate::mmap< EType >	59
MobilityCoefficient	60
multidimension_function< DIM >	61
multidimension_function< 1 >	62
multidimension_function< 2 >	63
multidimension_function< 3 >	64
my_best_constructor< T, DIM >	65
my_best_constructor< T, 1 >	65
my_best_constructor< T, 2 >	66
my_best_constructor< T, 3 >	67
NonlinearCoefficient	68
Parameter	70
Parameters	
Class used to manage a list of Parameter	71
PhaseFieldOperator< T, DIM, NLFI >	
PhaseFieldOperator class	74
PhaseFieldReducedOperator	83
PoissonNLFIIntegrator	84
PostProcessing< T, DC, DIM >	86
potential_function< ORDER, SCHEME >	90

potential_function< 0, ThermodynamicsPotentialDiscretization::Explicit >	91
potential_function< 0, ThermodynamicsPotentialDiscretization::Implicit >	93
potential_function< 0, ThermodynamicsPotentialDiscretization::SemImplicit >	96
potential_function< 1, ThermodynamicsPotentialDiscretization::Explicit >	99
potential_function< 1, ThermodynamicsPotentialDiscretization::Implicit >	102
potential_function< 1, ThermodynamicsPotentialDiscretization::SemImplicit >	104
potential_function< 2, ThermodynamicsPotentialDiscretization::Explicit >	107
potential_function< 2, ThermodynamicsPotentialDiscretization::Implicit >	110
potential_function< 2, ThermodynamicsPotentialDiscretization::SemImplicit >	113
PotentialFunctions< ORDER, SCHEME >	116
Problem	118
Problems	119
SpatialDiscretization< T, DIM >	120
ThermodynamicRootFindingAlgorithm	126
ThermodynamicsAllenCahnMobility	126
ThermodynamicsFGR	127
ThermodynamicsInitialization	128
ThermodynamicsLinearPowerSourceTerm	128
ThermodynamicsModel	129
ThermodynamicsMolarVolumeType	130
ThermodynamicsPotentials	131
ThermodynamicsProperties	132
ThermodynamicsStabilizationScheme	132
ThermodynamicSubModels	133
ThermodynamicsUnitForSettingConditions	134
ThermodynamicsUpdateQuantitiesMethod	135
ThermodynamicSystemsForMobilities	135
TimeDiscretization< T, DC, DIM, NLFI >	136
TimeScheme	138
UtilsForSolvers	
Useful methods for managing solvers	139
Variable< T, DIM >	141
Variables< T, DIM >	
Class used to manage a list of Variable	152
VariableType	156

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

Default 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

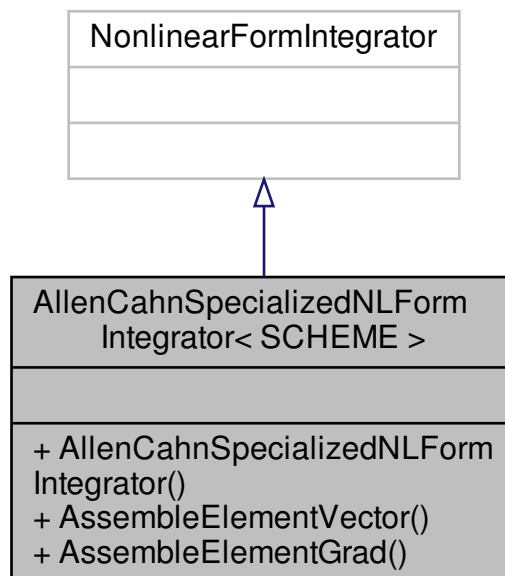
Default constant used by Newton Solver.

Chapter 14

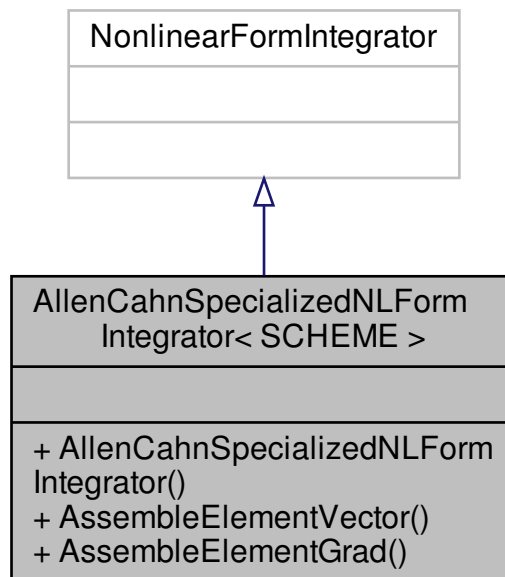
Data Structure Documentation

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

<i>SCHEME</i>	
---------------	--

Parameters

<i>_u_old</i>	
<i>_omega</i>	
<i>_lambda</i>	
<i>_alpha</i>	
<i>_mob</i>	
<i>_w_scheme</i>	
<i>_h_scheme</i>	

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

<i>SCHEME</i>	
---------------	--

Parameters

<i>el</i>	
<i>Tr</i>	
<i>elfun</i>	
<i>elmat</i>	

Definition at line 144 of file AllenCahnSpecializedNLFormIntegrator.hpp.

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

```

146                                     {
147     int nd = el.GetDof();
148     int dim = el.GetDim();
149     int spaceDim = Tr.GetSpaceDim();
150     bool square = (dim == spaceDim);
151     double w;
152
153     shape.SetSize(nd);
154     dshape.SetSize(nd, dim);
155     dshapedxt.SetSize(nd, spaceDim);
156     elmat.SetSize(nd);
157
158     const mfem::IntegrationRule* ir =
159         &mfem::IntRules.Get(el.GetGeomType(), 2 * el.GetOrder() + Tr.OrderW());
160
161     elmat = 0.0;
162     for (int i = 0; i < ir->GetNPoints(); i++) {
163         const mfem::IntegrationPoint& ip = ir->IntPoint(i);
164         el.CalcDShape(ip, dshape); // dphi
165         const auto u = elfun * shape;
166         const auto un = u_old.GetValue(Tr, ip);
167
168         const auto W = this->second_derivative_potential_.getPotentialFunction("W", un);
169         const auto H = this->second_derivative_potential_.getPotentialFunction("H", un);
170         const auto Wsecond = W(u);
171         const auto Hsecond = H(u);
172         const auto Mphi = mob.Eval(Tr, ip);
173
174         Tr.SetIntPoint(&ip);
175         w = Tr.Weight(); // det(J)
176         // std::cout << " SQUARE ? " << square << std::endl;
177         w = ip.weight / (square ? w : w * w * w);
178         // AdjugateJacobian = / adj(J), if J is square
179         // \ adj(J^t.J).J^t, otherwise
180
181         // Tr.AdjugateJacobian() det(J)J-1
182
183         // w = w* Mphi * lambda
184         w *= Mphi * this->lambda;
185
186         // dshapedxt = det(J)J-1 dshape
187         Mult(dshape, Tr.AdjugateJacobian(), dshapedxt);
188         // elmat += w * dshapedxt * dshapedxt^T
189         AddMult_a_AAt(w, dshapedxt, elmat);
190
191         // (this->omega * secondDerivativeDoubleWellPotential(elfun * shape) +
192         // this->alpha * secondDerivativeInterpolationPotential(elfun * shape)) *
193         // Compute w'(u)*(du,v), v is shape function
194         double fun_val =
195             Mphi * (this->omega * Wsecond + this->alpha * Hsecond) * ip.weight * Tr.Weight(); // w'(u)
196         // elmat += fun_val * shape * shape^T
197         AddMult_a_VVt(fun_val, shape, elmat); // w'(u)*(du, v)
198     }
199 }

```

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

<i>SCHEME</i>	
---------------	--

Parameters

<i>el</i>	
<i>Tr</i>	
<i>elfun</i>	
<i>elvect</i>	

Definition at line 82 of file AllenCahnSpecializedNLFormIntegrator.hpp.

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

```

84     {
85     int nd = el.GetDof();
86     int dim = el.GetDim();
87     int spaceDim = Tr.GetSpaceDim();
88     dshape.SetSize(nd, dim);
89     shape.SetSize(nd);
90     invdfdx.SetSize(dim, spaceDim);
91     vec.SetSize(dim);
92     pointflux.SetSize(spaceDim);
93
94     elvect.SetSize(nd);
95     const mfem::IntegrationRule* ir =
96         &mfem::IntRules.Get(el.GetGeomType(), 2 * el.GetOrder() + Tr.OrderW());
97
98     elvect = 0.0;
99     for (int i = 0; i < ir->GetNPoints(); i++) {
100         const mfem::IntegrationPoint& ip = ir->IntPoint(i);
101         el.CalcDShape(ip, dshape); // dphi
102         el.CalcShape(ip, shape); // phi
103         Tr.SetIntPoint(&ip);
104
105         const auto u = elfun * shape;
106         const auto un = u_old.GetValue(Tr, ip);
107
108         const auto W = this->first_derivative_potential_.getPotentialFunction("W", un);
109         const auto H = this->first_derivative_potential_.getPotentialFunction("H", un);
110         const auto Wprime = W(u);
111         const auto Hprime = H(u);
112         const auto Mphi = mob.Eval(Tr, ip);
113
114         CalcAdjugate(Tr.Jacobian(), invdfdx); // invdfdx = adj(J)
115
116         dshape.MultTranspose(elfun, vec);
117         invdfdx.MultTranspose(vec, pointflux);
118
119         const auto fun_val = Mphi * (this->omega * Wprime + this->alpha * Hprime);
120
121         // Given phi, compute (w'(phi)-f, v), v is shape function
122         const double ww = ip.weight * Tr.Weight() * fun_val;
123         add(elvect, ww, shape, elvect);
124
125         // Laplacian : given u, compute (grad(u), grad(v)), v is shape function.
126         double w;
127         w = Mphi * ip.weight * this->lambda / Tr.Weight();
128         pointflux *= w;
129         invdfdx.Mult(pointflux, vec);
130         dshape.AddMult(vec, elvect);
131     }
132 }

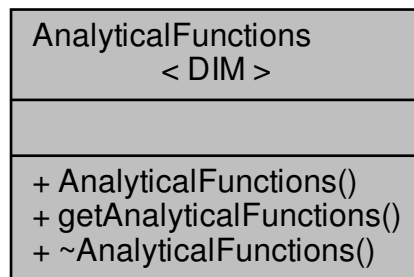
```

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

<i>analytical_function_name</i>	
---------------------------------	--

Returns

const double

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

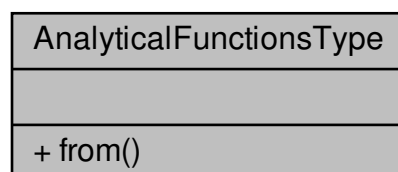
```
233
234     switch (AnalyticalFunctionsType::from(analytical_function_name)) {
235     case AnalyticalFunctionsType::Heaviside:
236         return this->getHeaviside(args...);
237     case AnalyticalFunctionsType::HyperbolicTangent:
238         return this->getHyperbolicTangent(args...);
239     case AnalyticalFunctionsType::Uniform:
240         return this->getUniform(args...);
241     default:
242         throw std::runtime_error(
243             "AnalyticalFunctions::getAnalyticalFunctions: Heaviside, HyperbolicTangent and Uniform "
244             "analytical function are available");
245         break;
246     }
247 }
```

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

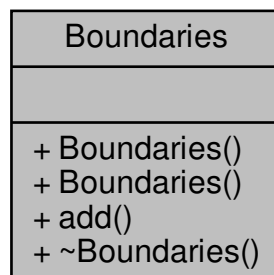
Definition at line 67 of file Utils/PhaseFieldOptions.hpp.

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

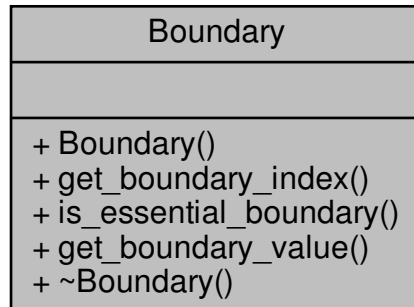
Definition at line 206 of file BoundaryConditions.hpp.

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

<i>boundary_name</i>	
<i>boundary_index</i>	
<i>boundary_type</i>	
<i>boundary_value</i>	

Definition at line 49 of file BoundaryConditions.hpp.

```

51     : boundary_name_(boundary_name),
52       boundary_index_(boundary_index),
53       boundary_value_(boundary_value) {
54     switch (BoundaryConditionType::from(boundary_type)) {
55     case BoundaryConditionType::Dirichlet:
56         this->is_essential_boundary_ = 1;
57         break;
58     case BoundaryConditionType::Neumann:
59     case BoundaryConditionType::Periodic:
60     case BoundaryConditionType::Robin:
61         this->is_essential_boundary_ = 0;
62         break;
63     default:
64         throw std::runtime_error(
65             "Boundary::Boundary(): only Dirichlet, Neumann, Periodic and Robin BoundaryConditionType "
66             "are available");
67         break;
68     }
69 }
```

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.

```
76 { 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:

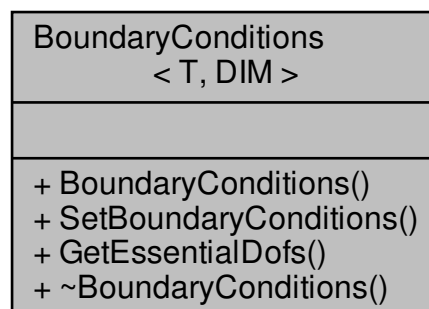
- BoundaryConditions.hpp

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

<i>Args</i>	
-------------	--

Parameters

<i>fespace</i>	
<i>mesh_max_bdr_attributes</i>	
<i>boundaries</i>	

Definition at line 138 of file BoundaryConditions.hpp.

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

```

139                                     {
140   this->fespace_ = spatial->get_finite_element_space();
141   const auto &mesh_max_bdr_attributes = spatial->get_max_bdr_attributes();
142
143   auto bdrs = std::vector<Boundary>{boundaries...};
144
145   Dirichlet_bdr_.SetSize(mesh_max_bdr_attributes);
146   Dirichlet_value_.SetSize(mesh_max_bdr_attributes);
147   if (mesh_max_bdr_attributes == bdrs.size()) {
148     for (const auto &bdr : bdrs) {
149       const auto &id = bdr.get_boundary_index();
150       if (bdr.is_essential_boundary()) {
151         Dirichlet_bdr_[id] = 1;
152       } else {
153         Dirichlet_bdr_[id] = 0;
154       }
155       Dirichlet_value_[id] = bdr.get_boundary_value();
156     }
157     this->fespace_->GetEssentialTrueDofs(this->Dirichlet_bdr_, this->ess_t dof_list_);
158
159   } else {
160     throw std::runtime_error(
161       "BoundaryConditions::BoundaryConditions(): the number of user-defined boundaries is "
162       "different from the total number of boundaries associated to the mesh ");
163   }
164 }

```

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_t dof_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

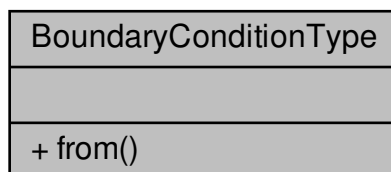
```

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

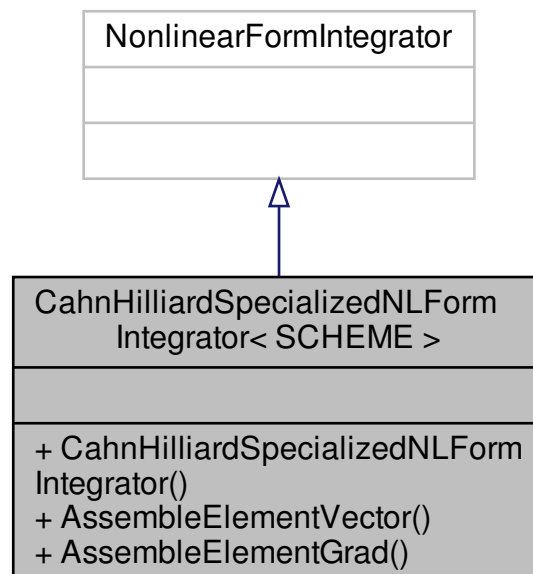
Definition at line 102 of file Utils/PhaseFieldOptions.hpp.

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

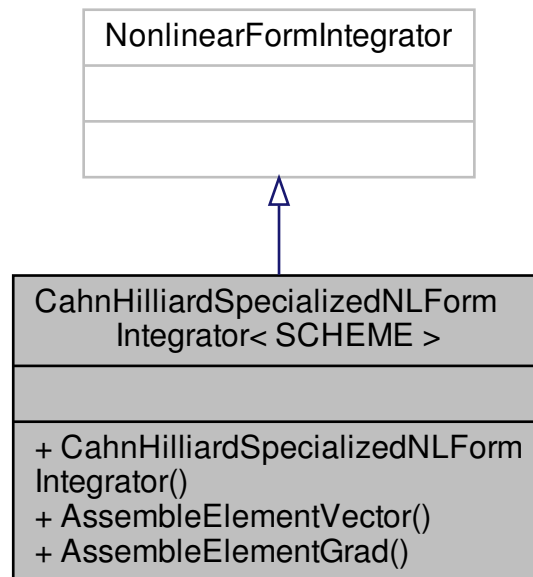
- Utils/PhaseFieldOptions.hpp

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

<i>SCHEME</i>	
---------------	--

Parameters

<i>_u_old</i>	
<i>_omega</i>	
<i>_lambda</i>	
<i>_alpha</i>	
<i>_mob</i>	

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

<i>SCHEME</i>	
---------------	--

Parameters

<i>el</i>	
<i>Tr</i>	
<i>elfun</i>	
<i>elmat</i>	

Definition at line 140 of file CahnHilliardSpecializedNLFormIntegrator.hpp.

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

```

142                                     {
143     int nd = el.GetDof();
144     int dim = el.GetDim();
145     int spaceDim = Tr.GetSpaceDim();
146     bool square = (dim == spaceDim);
147     double w;
148
149     shape.SetSize(nd);
150     dshape.SetSize(nd, dim);
151     dshapedxt.SetSize(nd, spaceDim);
152     elmat.SetSize(nd);
153
154     const mfem::IntegrationRule* ir =
155         &mfem::IntRules.Get(el.GetGeomType(), 2 * el.GetOrder() + Tr.OrderW());
156
157     elmat = 0.0;
158     for (int i = 0; i < ir->GetNPoints(); i++) {
159         const mfem::IntegrationPoint& ip = ir->IntPoint(i);
160         el.CalcDShape(ip, dshape); // dphi
161         const auto u = elfun * shape;
162         const auto un = u_old.GetValue(Tr, ip);
163         const auto W = this->second_derivative_potential_.getPotentialFunction("W", un);
164         const auto H = this->second_derivative_potential_.getPotentialFunction("H", un);
165         const auto Wsecond = W(u);
166         const auto Hsecond = H(u);
167         const auto Mphi = mob.Eval(Tr, ip);
168
169         Tr.SetIntPoint(&ip);
170         w = Tr.Weight(); // det(J)
171         // std::cout << " SQUARE ? " << square << std::endl;
172         w = ip.weight / (square ? w : w * w * w);
173         // AdjugateJacobian = / adj(J),           if J is square
174         //                   \ adj(J^t.J).J^t, otherwise
175
176         // Tr.AdjugateJacobian() det(J)J-1
177
178         // w = w * Mphi * lambda
179         w *= Mphi * this->lambda;
180
181         // dshapedxt = det(J)J-1 dshape
182         Mult(dshape, Tr.AdjugateJacobian(), dshapedxt);
183         // elmat += w * dshapedxt * dshapedxt^T
184         AddMult_a_AAt(w, dshapedxt, elmat);
185
186         // (this->omega * secondDerivativeDoubleWellPotential(elfun * shape) +
187         //  this->alpha * secondDerivativeInterpolationPotential(elfun * shape)) *
188         // Compute w'(u)*(du,v), v is shape function
189         double fun_val =
190             Mphi * (this->omega * Wsecond + this->alpha * Hsecond) * ip.weight * Tr.Weight(); // w'(u)
191         // elmat += fun_val * shape * shape^T
192         AddMult_a_VVt(fun_val, shape, elmat); // w'(u)*(du, v)
193     }
194 }

```

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

<i>SCHEME</i>	
---------------	--

Parameters

<i>el</i>	
<i>Tr</i>	
<i>elfun</i>	
<i>elvect</i>	

Definition at line 78 of file CahnHilliardSpecializedNLFormIntegrator.hpp.

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

```

80     {
81     int nd = el.GetDof();
82     int dim = el.GetDim();
83     int spaceDim = Tr.GetSpaceDim();
84     dshape.SetSize(nd, dim);
85     shape.SetSize(nd);
86     invdfdx.SetSize(dim, spaceDim);
87     vec.SetSize(dim);
88     pointflux.SetSize(spaceDim);
89
90     elvect.SetSize(nd);
91     const mfem::IntegrationRule* ir =
92         &mfem::IntRules.Get(el.GetGeomType(), 2 * el.GetOrder() + Tr.OrderW());
93
94     elvect = 0.0;
95     for (int i = 0; i < ir->GetNPoints(); i++) {
96         const mfem::IntegrationPoint& ip = ir->IntPoint(i);
97         el.CalcDShape(ip, dshape); // dphi
98         el.CalcShape(ip, shape); // phi
99         Tr.SetIntPoint(&ip);
100
101         const auto u = elfun * shape;
102         const auto un = u_old.GetValue(Tr, ip);
103
104         const auto W = this->first_derivative_potential_.getPotentialFunction("W", un);
105         const auto H = this->first_derivative_potential_.getPotentialFunction("H", un);
106         const auto Wprime = W(u);
107         const auto Hprime = H(u);
108         const auto Mphi = mob.Eval(Tr, ip);
109
110         CalcAdjugate(Tr.Jacobian(), invdfdx); // invdfdx = adj(J)
111
112         dshape.MultTranspose(elfun, vec);
113         invdfdx.MultTranspose(vec, pointflux);
114
115         const auto fun_val = Mphi * (this->omega * Wprime + this->alpha * Hprime);
116
117         // Given phi, compute (w'(phi)-f, v), v is shape function
118         const double ww = ip.weight * Tr.Weight() * fun_val;
119         add(elvect, ww, shape, elvect);
120
121         // Laplacian : given u, compute (grad(u), grad(v)), v is shape function.
122         double w;
123         w = Mphi * ip.weight * this->lambda / Tr.Weight();
124         pointflux *= w;
125         invdfdx.Mult(pointflux, vec);
126         dshape.AddMult(vec, elvect);
127     }
128 }

```

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::DSmoothen **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.

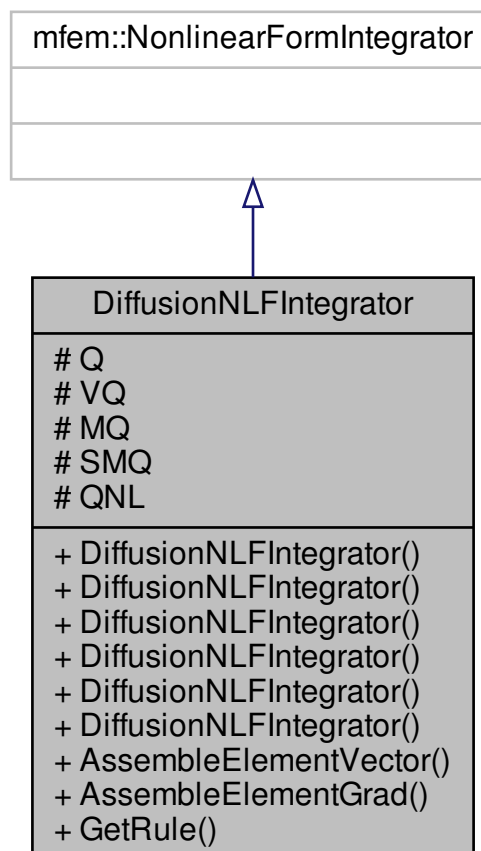
```
138                                     {
139     // Solve the equation:
140     //   du_dt = M^{-1} * [-K(u + dt*du_dt)]
141     // for du_dt, where K is linearized by using u from the previous timestep
142     if (!T) {
143         T = Add(1.0, Mmat, dt, Kmat);
144         current_dt = dt;
145         T_solver.SetOperator(*T);
146     }
147     MFEM_VERIFY(dt == current_dt, ""); // SDIRK methods use the same dt
148     Kmat.Mult(u, z);
149     z.Neg();
150
151     T_solver.Mult(z, du_dt);
152     du_dt.SetSubVector(ess_t dof_list, 0.0);
153 }
```

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

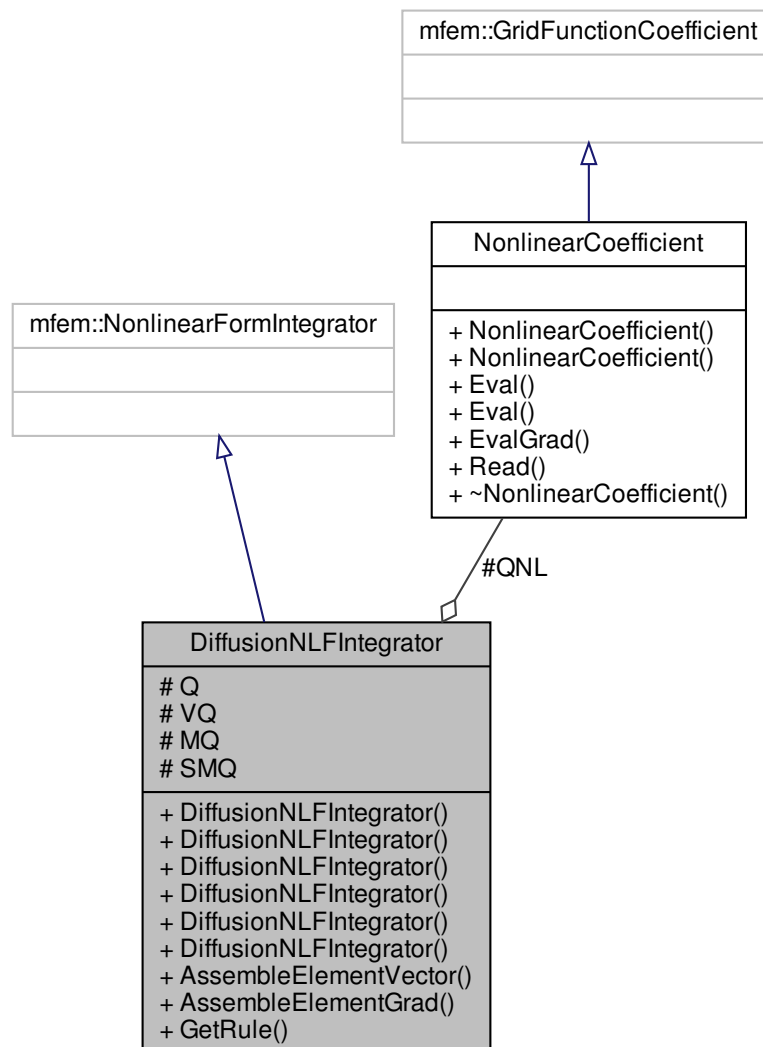
- ConductionOperator.hpp

14.10 DiffusionNLFItegrator Class Reference

Inheritance diagram for DiffusionNLFItegrator:



Collaboration diagram for DiffusionNLFIIntegrator:



Public Member Functions

- [DiffusionNLFIIntegrator](#) ()
Construct a diffusion nonlinear integrator with coefficient $Q = 1$.
- [DiffusionNLFIIntegrator](#) (mfem::ConstantCoefficient &q)
- [DiffusionNLFIIntegrator](#) (mfem::ConstantCoefficient &q, [NonlinearCoefficient](#) &qq)
- [DiffusionNLFIIntegrator](#) (mfem::VectorCoefficient &q, [NonlinearCoefficient](#) &qq)
- [DiffusionNLFIIntegrator](#) (mfem::MatrixCoefficient &q, [NonlinearCoefficient](#) &qq)
- [DiffusionNLFIIntegrator](#) (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 DiffusionNLFItegrator.hpp.

14.10.2 Constructor & Destructor Documentation

14.10.2.1 DiffusionNLFItegrator() [1/5]

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

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

Definition at line 69 of file DiffusionNLFItegrator.hpp.

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

14.10.2.2 DiffusionNLFItegrator() [2/5]

```
DiffusionNLFItegrator::DiffusionNLFItegrator (
    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 DiffusionNLFItegrator.hpp.

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

14.10.2.3 DiffusionNLFItegrator() [3/5]

```
DiffusionNLFItegrator::DiffusionNLFItegrator (
    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 DiffusionNLFItegrator.hpp.

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

14.10.2.4 DiffusionNLFItegrator() [4/5]

```
DiffusionNLFItegrator::DiffusionNLFItegrator (
    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 DiffusionNLFItegrator.hpp.

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

14.10.2.5 DiffusionNLFItegrator() [5/5]

```
DiffusionNLFItegrator::DiffusionNLFItegrator (
    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 DiffusionNLFItegrator.hpp.

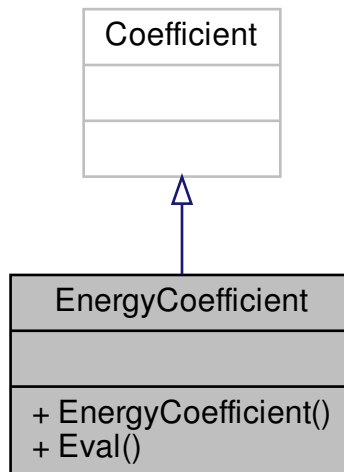
```
90      : Q(NULL), VQ(NULL), MQ(NULL), SMQ(&q), QNL(&qq) {}
```

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

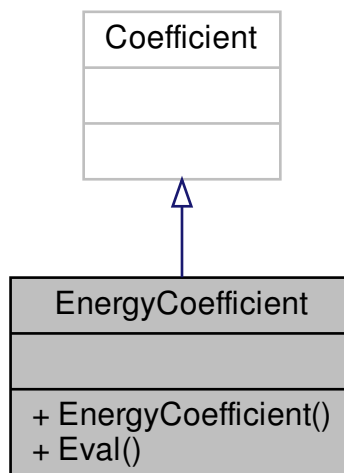
- DiffusionNLFItegrator.hpp

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

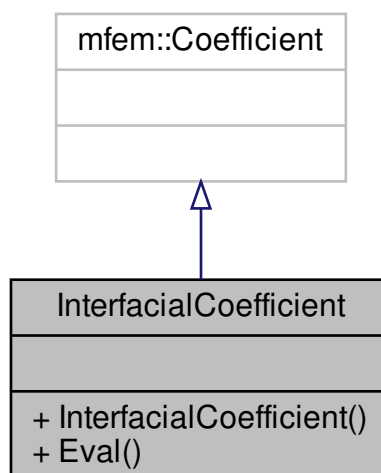
Definition at line 37 of file EnergyCoefficient.hpp.

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

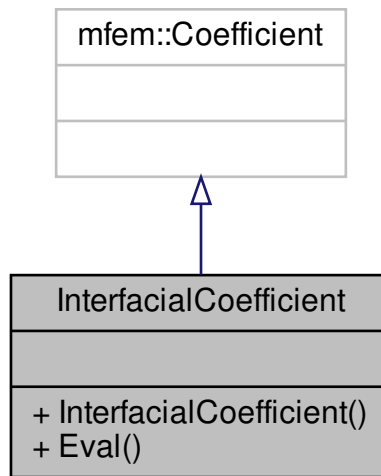
- EnergyCoefficient.hpp

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

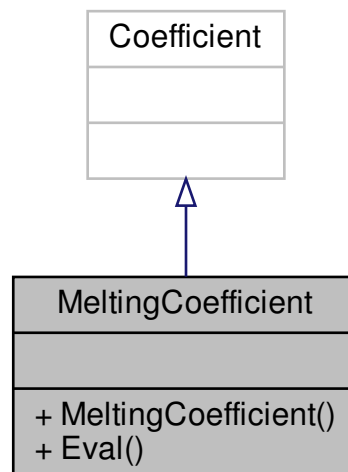
Definition at line 15 of file `EnergyCoefficient.hpp`.

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

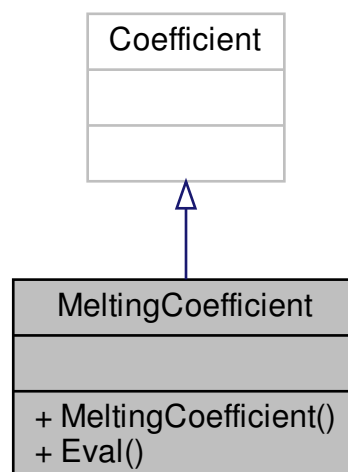
- `EnergyCoefficient.hpp`

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

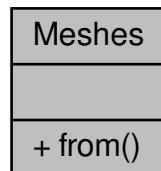
Definition at line 62 of file EnergyCoefficient.hpp.

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

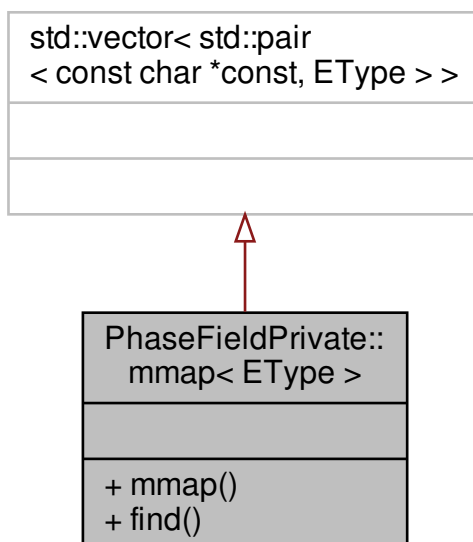
Definition at line 87 of file Utils/PhaseFieldOptions.hpp.

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

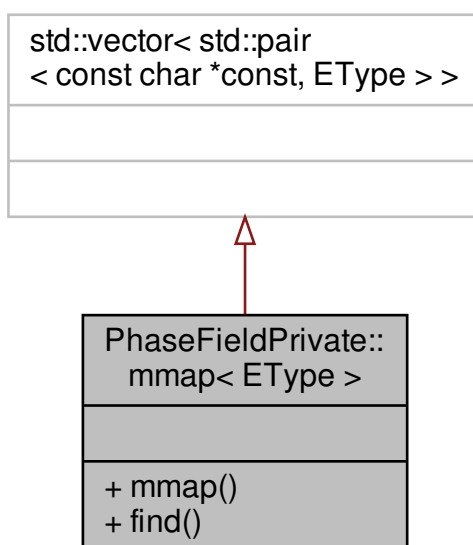
- Utils/PhaseFieldOptions.hpp

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 >
```

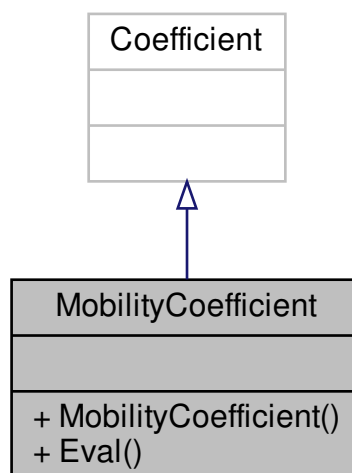
Definition at line 21 of file Utils/PhaseFieldOptions.hpp.

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

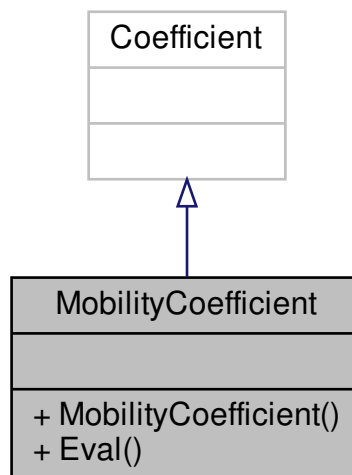
- Utils/PhaseFieldOptions.hpp

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

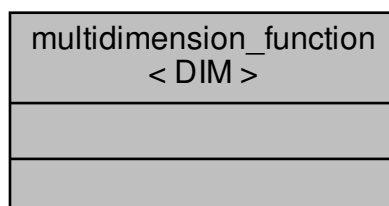
Definition at line 12 of file MobilityCoefficient.hpp.

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 >
```

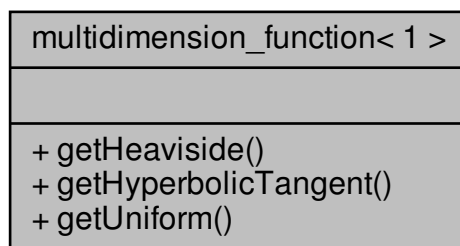
Definition at line 18 of file Utils/AnalyticalFunctions.hpp.

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 >
```

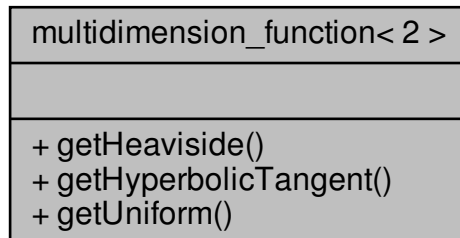
Definition at line 55 of file Utils/AnalyticalFunctions.hpp.

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 >
```

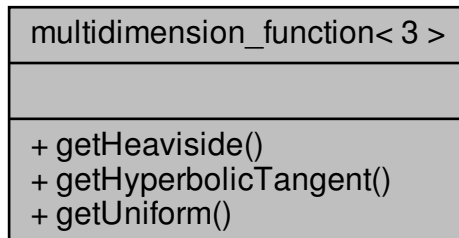
Definition at line 103 of file `Utils/AnalyticalFunctions.hpp`.

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 >
```

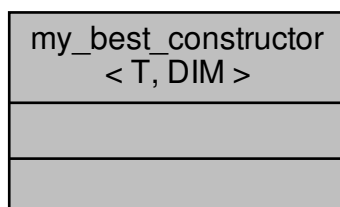
Definition at line 154 of file Utils/AnalyticalFunctions.hpp.

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 >
```

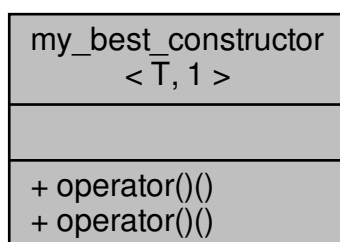
Definition at line 27 of file Spatial.hpp.

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 >
```

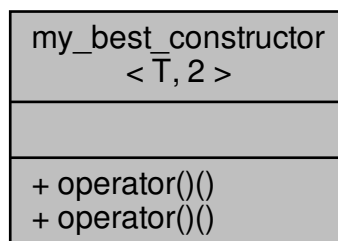
Definition at line 76 of file Spatial.hpp.

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 >
```

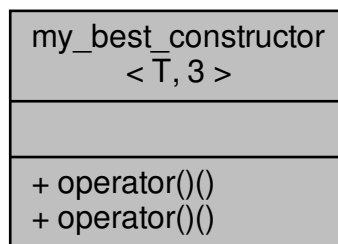
Definition at line 141 of file Spatial.hpp.

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.

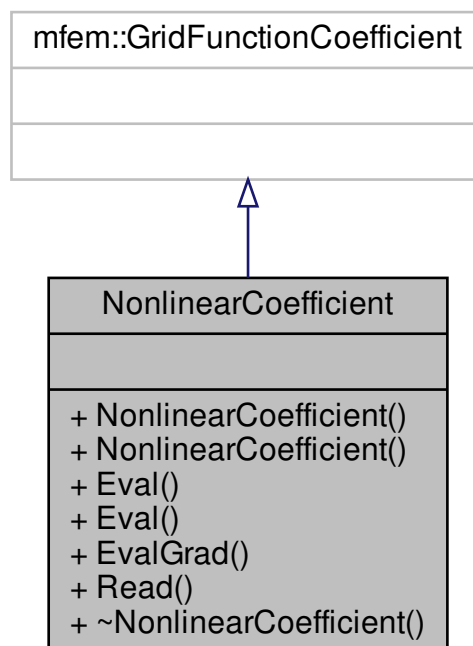
The documentation for this struct was generated from the following file:

- Spatial.hpp

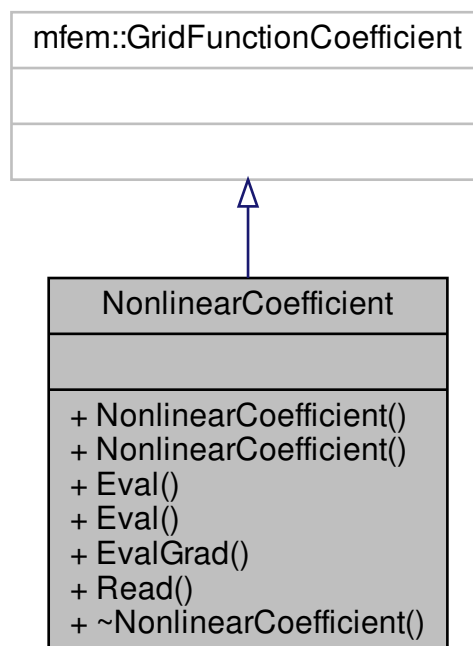
14.25 NonlinearCoefficient Class Reference

```
#include </home/ci230846/home-local/MyGitProjects/COMPONENT/PF-MFEM/Integrators/↵  
DiffusionNLFItegrator.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](#).

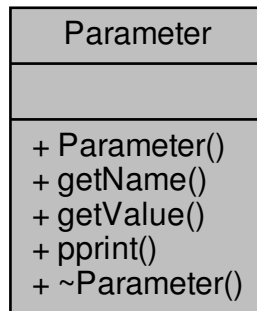
Definition at line 13 of file `DiffusionNLFIntegrator.hpp`.

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:

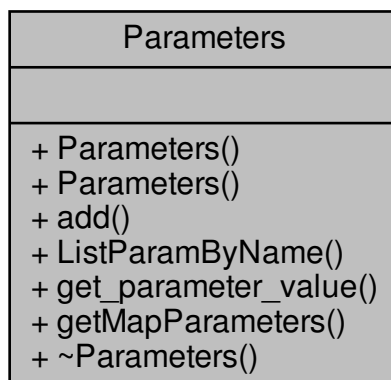
- `Parameter.hpp`

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](#) ¶m)
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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

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

<i>param</i>	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

<i>name</i>	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:

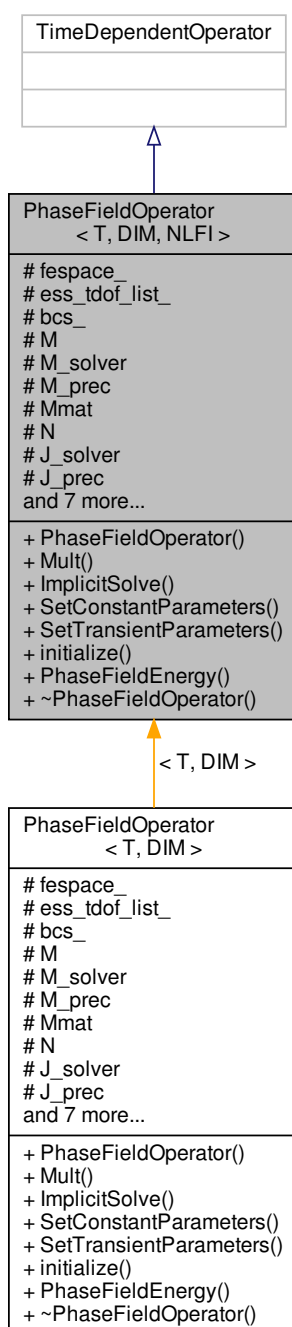
- Parameters.hpp

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

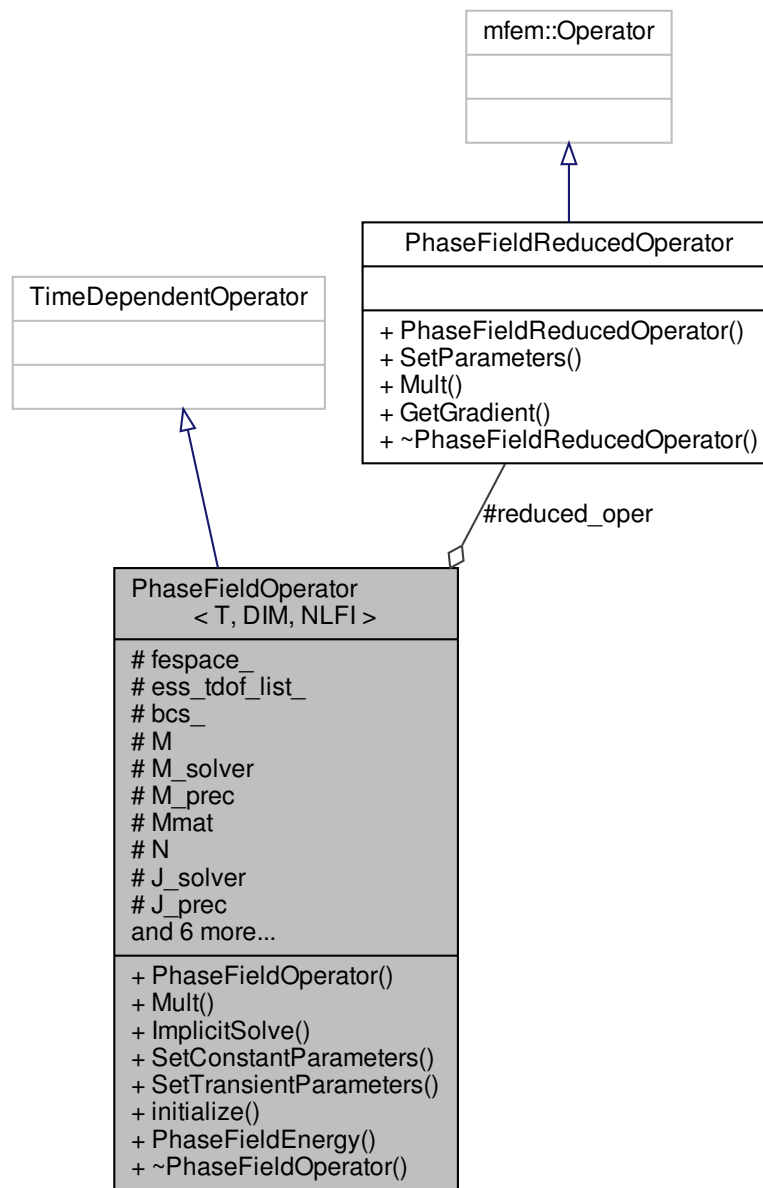
[PhaseFieldOperator](#) class.

```
#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](#) ¶ms, [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: $k = f(\phi + dt \cdot k, t)$, 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::DSmoothen **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

<i>fespace</i>	Finite Element space
<i>params</i>	list of Parameters
<i>u</i>	unknown vector

Definition at line 141 of file PhaseFieldOperator.hpp.

```

144 : mfem::TimeDependentOperator(spatial->getSize(), 0.0),
145   M(NULL),
146   N(NULL),
147   current_dt_(0.0),
148   z(height) {
149   this->fespace_ = spatial->get_finite_element_space();
150   this->omega_ = params.get_parameter_value("omega");
151   this->lambda_ = params.get_parameter_value("lambda");
152   this->mobility_coeff_ = params.get_parameter_value("mobility");
153
154   auto &vv = vars.get_variable("phi");
155   this->initialize(vv);
156   // auto u = vv.get_unknown();
157   // this->ess_tdof_list_ = this->bcs_.GetEssentialDofs();
158   // this->bcs_.SetBoundaryConditions(u);
159   // this->SetConstantParameters(this->current_dt_, u);
160   // this->SetTransientParameters(this->current_dt_, u);
161   // vv.update(u);
162 }
```

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: $k = f(\text{phi} + dt * k, t)$, 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

<i>dt</i>	current time step
<i>u</i>	unknown vector
<i>du_↔</i> <i>_dt</i>	unkwon time derivative vector

Definition at line 280 of file PhaseFieldOperator.hpp.

```

281
282   const auto sc = height;
```

```

283  mfem::Vector v(u.GetData(), sc);
284  mfem::Vector dv_dt(du_dt.GetData(), sc);
285  // // Solve the equation:
286  // //   du_dt = M^{-1} * [-K(u + dt*du_dt)]
287  // // for du_dt
288
289  this->bcs_->SetBoundaryConditions(v);
290  this->SetTransientParameters(dt, v);
291
292  reduced_oper->SetParameters(dt, &v);
293
294  mfem::Vector zero; // empty vector is interpreted as zero r.h.s. by NewtonSolver
295  dv_dt = v;
296  dv_dt *= (1. / dt);
297  this->newton_solver_.Mult(zero, dv_dt);
298  dv_dt.SetSubVector(this->ess_tdof_list_, 0.0); // pour Dirichlet ... uniquement?
299  // std::cout << " PhaseFieldOperator this->newton_solver_->Mult " << std::endl;
300
301  MFEM_VERIFY(this->newton_solver_.GetConverged(), "Nonlinear solver did not converge.");
302 }

```

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
$\frac{du}{dt}$	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

<i>dt</i>	time-step
<i>u</i>	unknown vector
<i>ess_tdof_list</i>	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
238     // Mass matrix
239     M = new mfem::BilinearForm(this->fespace_);
240     M->AddDomainIntegrator(new mfem::MassIntegrator());
241     M->Assemble(0);
242     mfem::SparseMatrix tmp;
243     M->FormSystemMatrix(this->ess_tdof_list_, Mmat);
244     this->ut_solver_.SetSolverParameters(
245         M_solver, MassDefaultConstant::print_level, MassDefaultConstant::iterative_mode,
246         MassDefaultConstant::iter_max, MassDefaultConstant::rel_tol, MassDefaultConstant::abs_tol);
247     this->ut_solver_.BuildSolver(M_solver, M_prec, Mmat);
248 }
249
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

<i>dt</i>	time-step
<i>u</i>	unknown vector

Definition at line 191 of file PhaseFieldOperator.hpp.

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

```

192
193     delete N;
194     delete reduced_oper;
195
196     // PhaseField reduced operator N
197     N = new mfem::NonlinearForm(this->fespace_);
198     mfem::GridFunction un_gf(this->fespace_);
199     un_gf.SetFromTrueDofs(u);
200     mfem::GridFunction un(this->fespace_);
201     un.SetFromTrueDofs(u);
202
203     MobilityCoefficient mob(un_gf, this->mobility_coeff_, 0);
204     // SourceCoefficient
205     auto dh = 0.; // 7.e4;
206     N->AddDomainIntegrator(new NLFI(un, this->omega_, this->lambda_, dh, mob));
207     N->SetEssentialTrueDofs(this->ess_tdof_list_);
208
209     reduced_oper = new PhaseFieldReducedOperator(M, N);
210
211     // Newton Solver
212     this->ut_solver_.SetSolverParameters(
213         this->newton_solver_, NewtonDefaultConstant::print_level,
214         NewtonDefaultConstant::iterative_mode, NewtonDefaultConstant::iter_max,
215         NewtonDefaultConstant::rel_tol, NewtonDefaultConstant::abs_tol);
216     // TODO(ci230846) : cette partie devra etre generalisee pour un solveur iteratif
217     J_solver = new mfem::UMFPackSolver;
218     this->ut_solver_.BuildSolver(this->newton_solver_, *J_solver, *
219         reduced_oper);
220 }

```

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.

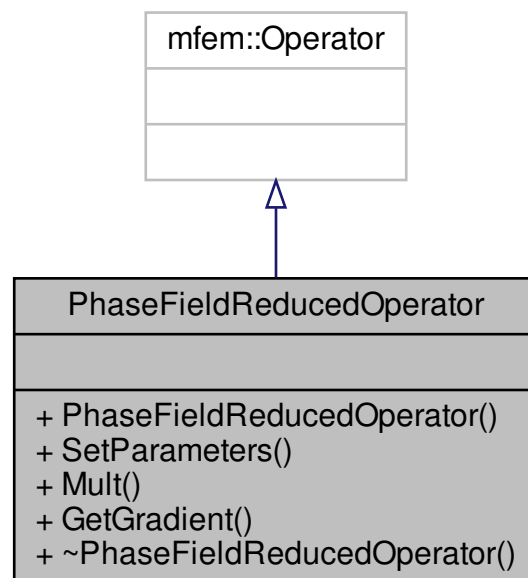
Referenced by PhaseFieldOperator< T, DIM >::ImplicitSolve(), PhaseFieldOperator< T, DIM >::SetConstantParameters(), PhaseFieldOperator< T, DIM >::SetTransientParameters(), and PhaseFieldOperator< T, DIM >::~~PhaseFieldOperator().

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

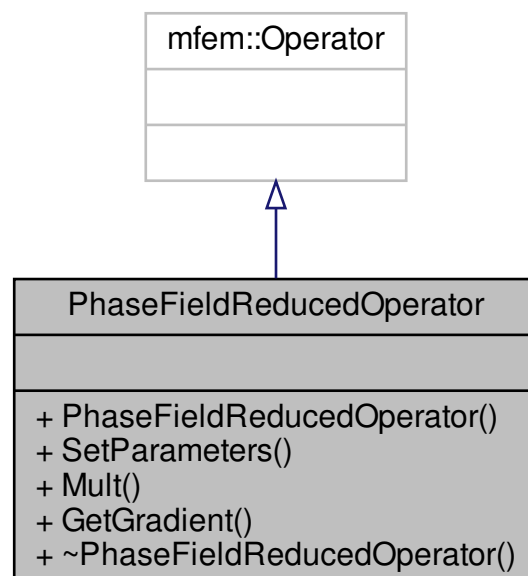
- PhaseFieldOperator.hpp

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

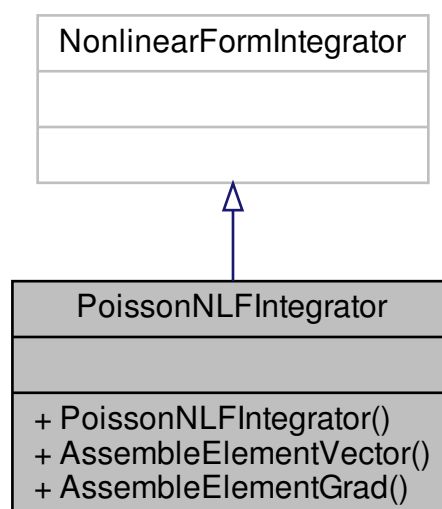
Definition at line 22 of file ReducedOperator.hpp.

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

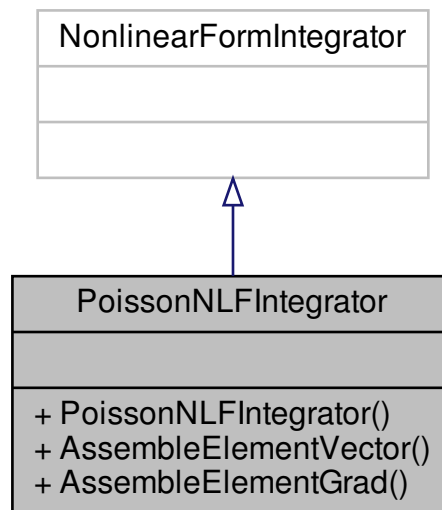
- ReducedOperator.hpp

14.30 PoissonNLFIntegrator Class Reference

Inheritance diagram for PoissonNLFIntegrator:



Collaboration diagram for PoissonNLFItegrator:



Public Member Functions

- **PoissonNLFItegrator** (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

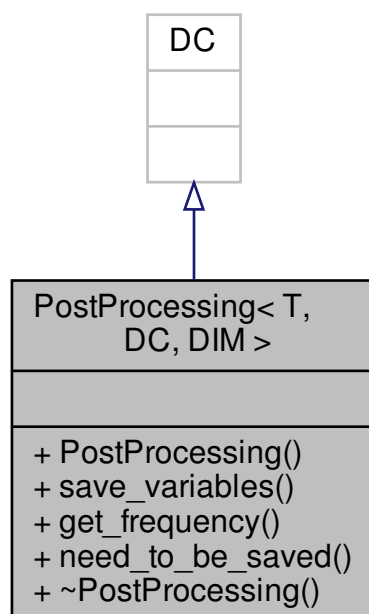
Definition at line 12 of file `PoissonNLFItegrator.hpp`.

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

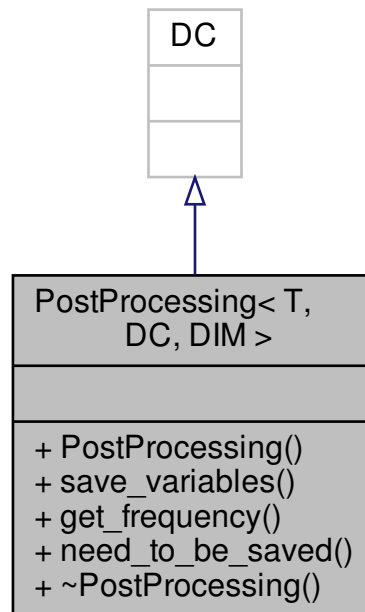
- `PoissonNLFItegrator.hpp`

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

<i>main_folder_path</i>	
<i>calculation_path</i>	
<i>mesh</i>	
<i>level_of_detail</i>	

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

<i>iteration</i>	
------------------	--

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

<i>vars</i>	
<i>iter</i>	
<i>time</i>	

Definition at line 75 of file postprocessing.hpp.

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

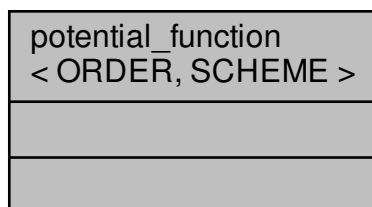
```
76
77     this->SetCycle(iter);
78     this->SetTime(time);
79     auto map_var = vars.get_map_gridfunction();
80     for (auto [name, gf] : map_var) {
81         this->RegisterField(name, &gf);
82         this->Save();
83     }
84 }
```

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 >
```

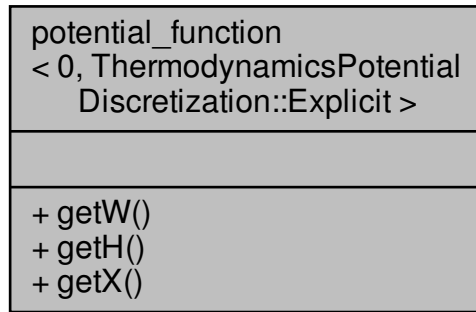
Definition at line 20 of file `PhaseFieldPotentials.hpp`.

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

Returns

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

Definition at line 244 of file PhaseFieldPotentials.hpp.

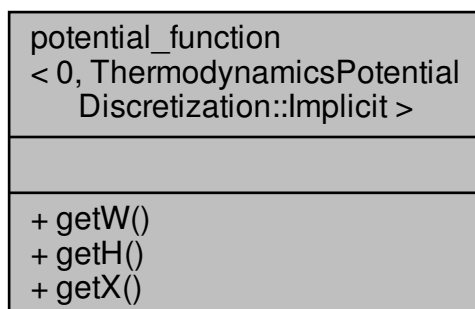
```
244                                     {
245     return std::function<double(const double&)>([](double x) {
246         const auto pot = x;
247         return pot;
248     });
249 }
```

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

Returns

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

Definition at line 83 of file PhaseFieldPotentials.hpp.

```

84     return std::function<double(const double&)>([](double x) {
85         const auto pot = x * x * x * (6.0 * x * x - 15.0 * x + 10.0);
86         return pot;
87     });
88 }

```

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 $W(x)=x^2 * (1-x)^2$

Template Parameters

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

Returns

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

Definition at line 97 of file PhaseFieldPotentials.hpp.

```

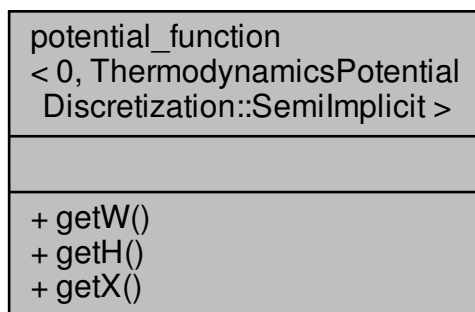
97                                     {
98     return std::function<double(const double&)>([](double x) {
99         const auto pot = x;
100         return pot;
101     });
102 }
```

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

- PhaseFieldPotentials.hpp

14.35 `potential_function< 0, ThermodynamicsPotentialDiscretization::SemImplicit >` Struct Template Reference

Collaboration diagram for `potential_function< 0, ThermodynamicsPotentialDiscretization::SemImplicit >`:



Public Member Functions

- `template<typename... Args>`
`std::function< double(const double &)>` [`getW`](#) (`Args... args`)
*Double Well potential $W(x)=x^2 * (1-x)^2$ with semi-implicit scheme (as implicit/explicit schemes)*
- `template<typename... Args>`
`std::function< double(const double &)>` [`getH`](#) (`Args... args`)
*Interpolation potential $H(x)=x^3 * (6x^2-15x+10)$ 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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

Returns

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

Definition at line 370 of file PhaseFieldPotentials.hpp.

```

370                                     {
371     return std::function<double(const double&)>([](double x) {
372         const auto pot = x * x * (1.0 - x) * (1.0 - x);
373         return pot;
374     });
375 }
```

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

Returns

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

Definition at line 400 of file PhaseFieldPotentials.hpp.

```

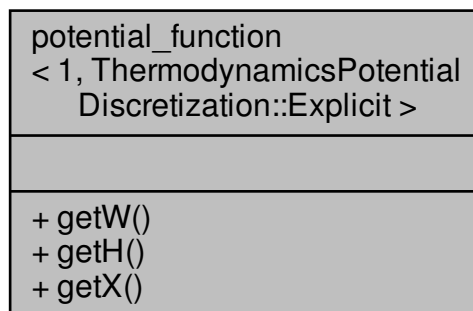
400                                     {
401     return std::function<double(const double&)>([](double x) {
402         const auto pot = x;
403         return pot;
404     });
405 }
```


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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

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 >`:

<code>potential_function< 1, ThermodynamicsPotentialDiscretization::Implicit ></code>
<ul style="list-style-type: none"> + <code>getW()</code> + <code>getH()</code> + <code>getX()</code>

Public Member Functions

- `template<typename... Args>`
`std::function< double(const double &)>` `getW` (`Args... args`)
*First derivative of the double Well potential $W(x)=x^2 * (1-x)^2$*
- `template<typename... Args>`
`std::function< double(const double &)>` `getH` (`Args... args`)
*First derivative of the interpolation potential $H(x)=x^3 * (6x^2-15x+10)$*
- `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 $H(x)=x^3 * (6x^2-15x+10)$

Template Parameters

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

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 $W(x)=x^2 * (1-x)^2$

Template Parameters

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

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 >:

potential_function < 1, ThermodynamicsPotential Discretization::SemiImplicit >
+ getW() + getH() + getX()

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

Template Parameters

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

Returns

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

Definition at line 444 of file PhaseFieldPotentials.hpp.

```

444                                     {
445     return std::function<double(const double&)>([](double x) {
446         const auto pot = 30. * x * x * (1.0 - x) * (1.0 - x);
447         return pot;
448     });
449 }
```

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

Returns

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

Definition at line 459 of file PhaseFieldPotentials.hpp.

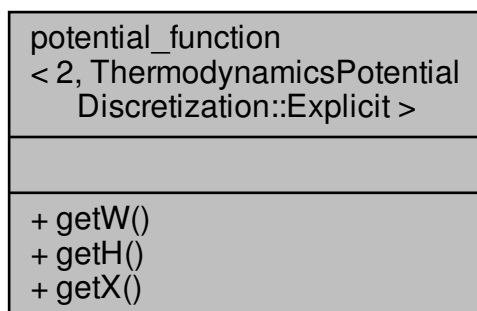
```
459                                     {  
460     return std::function<double(const double&)>([](double x) {  
461         const auto pot = 1.;  
462         return pot;  
463     });  
464 }
```

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

Returns

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

Definition at line 331 of file PhaseFieldPotentials.hpp.

```

332     return std::function<double(const double&)>([](double x) {
333         const auto pot = 60. * x * (1.0 - x) * (1.0 - 2. * x);
334         return pot;
335     });
336 }

```

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 $W(x)=x^2 * (1-x)^2$ with explicit scheme (as implicit scheme)

Template Parameters

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

Returns

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

Definition at line 316 of file PhaseFieldPotentials.hpp.

```

316                                     {
317     return std::function<double(const double&)>([](double x) {
318         const auto pot = 2. * (1. - 6. * x + 6. * x * x);
319         return pot;
320     });
321 }

```

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

Returns

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

Definition at line 346 of file PhaseFieldPotentials.hpp.

```

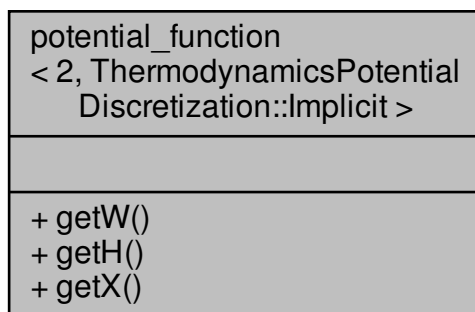
346                                     {
347     return std::function<double(const double&)>([](double x) {
348         const auto pot = 0.;
349         return pot;
350     });
351 }
```

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 $W(x)=x^2 * (1-x)^2$*
- `template<typename... Args>`
`std::function< double(const double &)>` `getH` (`Args... args`)
*Second derivative of the interpolation potential $H(x)=x^3 * (6x^2-15x+10)$*
- `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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

Returns

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

Definition at line 179 of file PhaseFieldPotentials.hpp.

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

Returns

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

Definition at line 193 of file PhaseFieldPotentials.hpp.

```

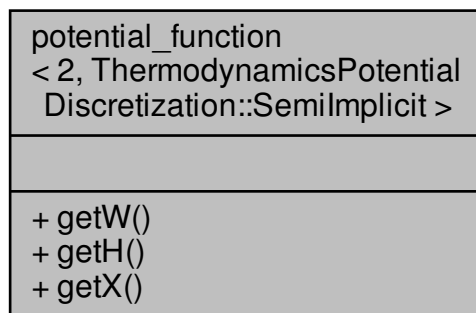
193                                     {
194     return std::function<double(const double&)>([](double x) {
195         const auto pot = 0.;
196         return pot;
197     });
198 }
```

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

- PhaseFieldPotentials.hpp

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

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



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::SemImplicit >
```

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

Returns

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

Definition at line 503 of file PhaseFieldPotentials.hpp.

```
503                                     {
504     auto v = std::vector<double>{args...};
505
506     if (v.size() == 1) {
507         const auto xn = v[0];
508         return std::function<double(const double&)>([xn](double x) {
509             const auto pot = 30. * (1.0 - x - xn) * (x + xn - x * x - xn * xn);
510             return pot;
511         });
512     } else {
513         throw std::runtime_error(
514             "potential_function::getH: only one argument is expected for smei-implicit scheme");
515     }
516 }
```

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

Returns

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

Definition at line 480 of file PhaseFieldPotentials.hpp.

```

480                                     {
481     auto v = std::vector<double>{args...};
482
483     if (v.size() == 1) {
484         const auto xn = v[0];
485         return std::function<double(const double&)>([xn](double x) {
486             const auto pot = ((1.0 - x - xn) * (1.0 - 2.0 * x) - (x + xn - x * x - xn * xn));
487             return pot;
488         });
489     } else {
490         throw std::runtime_error(
491             "potential_function::getW: only one argument is expected for smei-implicit scheme");
492     }
493 }
```

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

Returns

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

Definition at line 526 of file PhaseFieldPotentials.hpp.

```

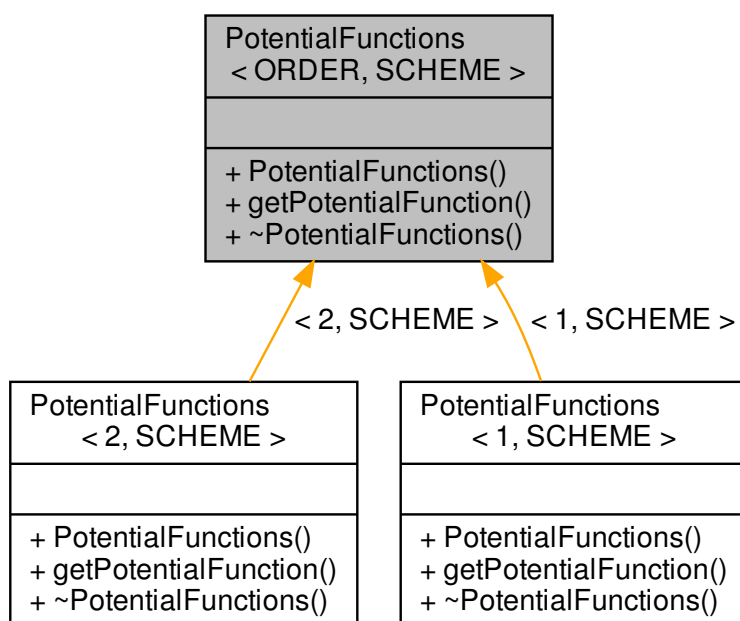
526                                     {
527     return std::function<double(const double&)>([](double x) {
528         const auto pot = 0.;
529         return pot;
530     });
531 }
```

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

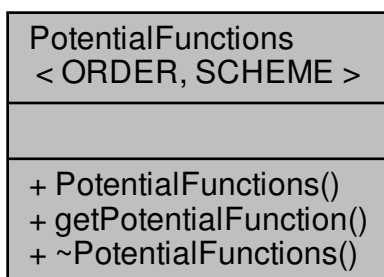
- PhaseFieldPotentials.hpp

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 >::getPotentialFunction (
    const std::string & potential_name,
    Args... args )
```

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

Parameters

<i>potential_name</i>	
-----------------------	--

Returns

const double

Definition at line 566 of file PhaseFieldPotentials.hpp.

Referenced by `CahnHilliardSpecializedNLFormIntegrator< SCHEME >::AssembleElementGrad()`, `AllenCahnSpecializedNLFormIntegrator< SCHEME >::AssembleElementGrad()`, `CahnHilliardSpecializedNLFormIntegrator< SCHEME >::AssembleElementVector()`, and `AllenCahnSpecializedNLFormIntegrator< SCHEME >::AssembleElementVector()`.

```

567                                     {
568     switch (ThermodynamicsPotentials::from(potential_name)) {
569     case ThermodynamicsPotentials::W:
570         return this->getW(args...);
571     case ThermodynamicsPotentials::H:
572         return this->getH(args...);
573     case ThermodynamicsPotentials::X:
574         return this->getX(args...);
575     default:
576         throw std::runtime_error(
577             "PotentialFunctions::getPotentialFunctions: double well, H interpolation and identity "
578             "potential function are available");
579         break;
580     }
581 }

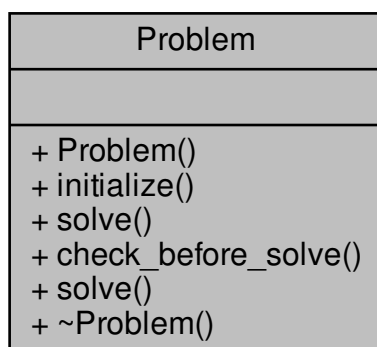
```

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 existence of mandatory objects
- void [solve](#) ()
check the existence of mandatory objects and solve the problem
- [~Problem](#) ()
Destroy the [Problem](#):: [Problem](#) object.

14.43.1 Detailed Description

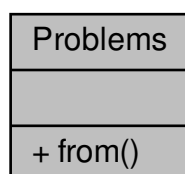
Definition at line 22 of file Problem.hpp.

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

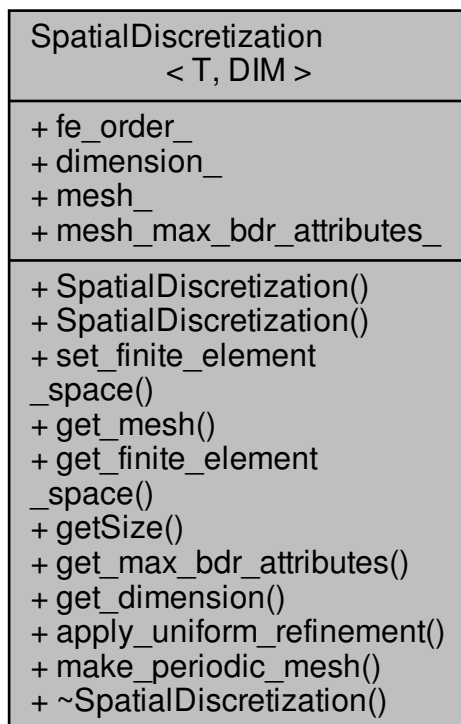
Definition at line 110 of file Utils/PhaseFieldOptions.hpp.

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

<i>T</i>	
----------	--

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

<i>T</i>	
----------	--

Parameters

<i>nb_ref</i>	
---------------	--

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

<i>T</i>	
----------	--

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

<i>T</i>	
----------	--

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

<i>T</i>	
----------	--

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

<i>T</i>	
----------	--

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

<i>T</i>	
----------	--

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

<i>T</i>	
<i>DIM</i>	

Parameters

<i>translations</i>	
---------------------	--

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

<i>T</i>	
----------	--

Returns

mfem::FiniteElementSpace*

Definition at line 322 of file Spatial.hpp.

```

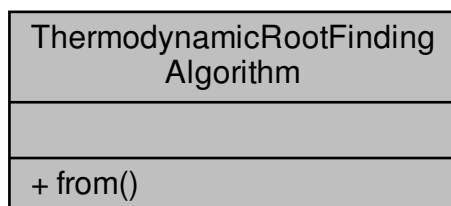
322                                                                 {
323     this->fecollection_ = new T(this->fe_order_, this->dimension_);
324     this->fespace_ = new mfem::FiniteElementSpace(&this->mesh_, this->fecollection_);
325     this->size_ = this->fespace_->GetTrueVSize();
326 }
```

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

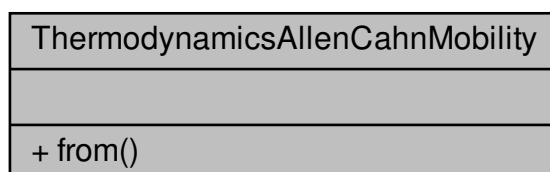
Definition at line 159 of file Coefficients/PhaseFieldOptions.hpp.

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

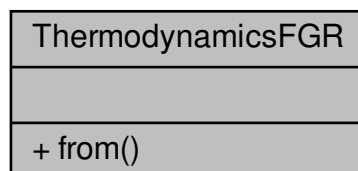
Definition at line 121 of file Coefficients/PhaseFieldOptions.hpp.

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

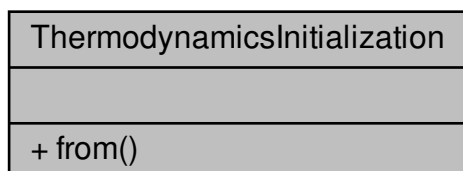
Definition at line 39 of file Coefficients/PhaseFieldOptions.hpp.

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

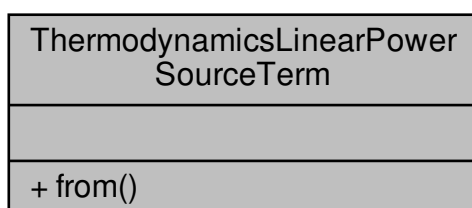
Definition at line 21 of file `Coefficients/PhaseFieldOptions.hpp`.

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

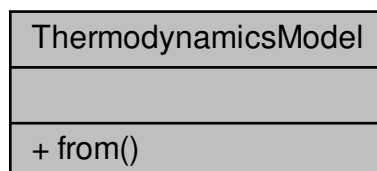
Definition at line 103 of file Coefficients/PhaseFieldOptions.hpp.

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

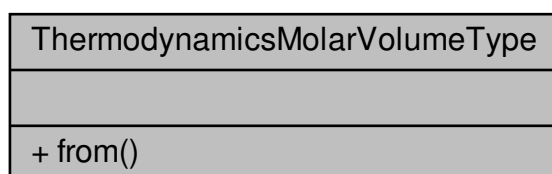
Definition at line 44 of file Coefficients/PhaseFieldOptions.hpp.

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

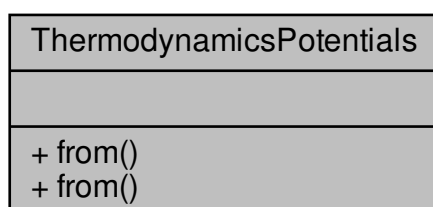
Definition at line 109 of file Coefficients/PhaseFieldOptions.hpp.

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

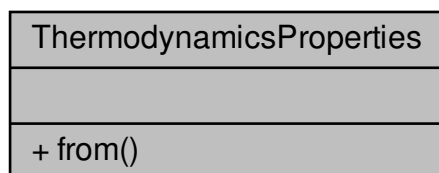
Definition at line 116 of file Coefficients/PhaseFieldOptions.hpp.

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

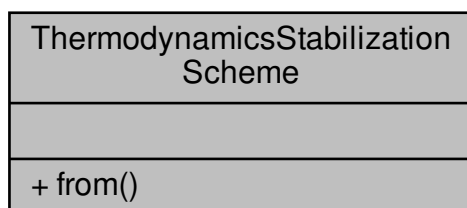
Definition at line 136 of file Coefficients/PhaseFieldOptions.hpp.

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

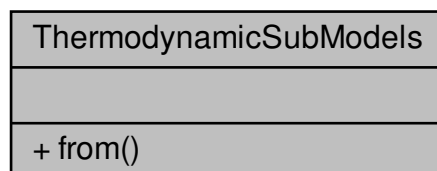
Definition at line 78 of file Coefficients/PhaseFieldOptions.hpp.

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

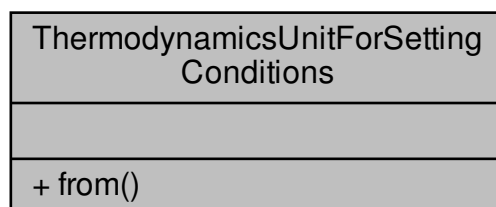
Definition at line 26 of file Coefficients/PhaseFieldOptions.hpp.

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

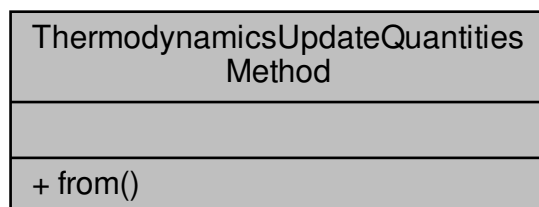
Definition at line 61 of file Coefficients/PhaseFieldOptions.hpp.

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

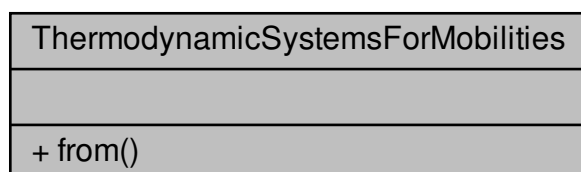
Definition at line 56 of file Coefficients/PhaseFieldOptions.hpp.

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

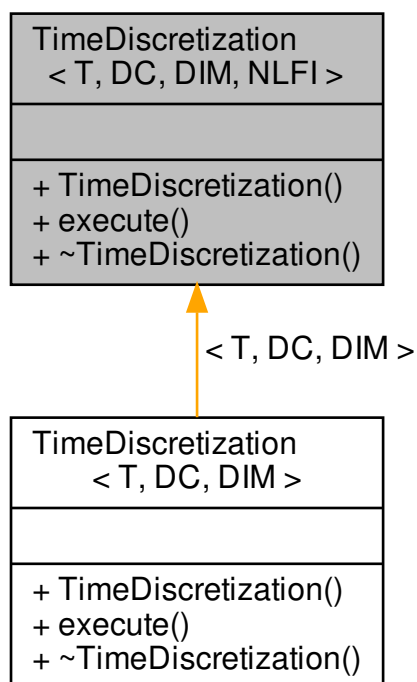
Definition at line 165 of file Coefficients/PhaseFieldOptions.hpp.

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

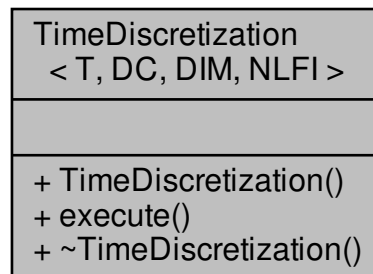
- Coefficients/PhaseFieldOptions.hpp

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](#) ¶ms, 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

<i>ode_solver</i>	
<i>unknown</i>	
<i>with_save</i>	

Definition at line 81 of file Time.hpp.

```

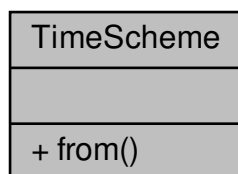
84     : oper_(oper), variables_(variables), pst_(pst) {
85   this->initial_time_ = params.get_parameter_value("initial_time");
86   this->final_time_ = params.get_parameter_value("final_time");
87   this->time_step_ = params.get_parameter_value("time_step");
88   this->setODESolver(ode_solver);
89 }
```

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:

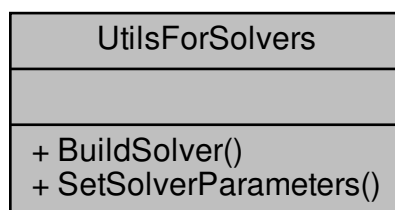
- Utils/PhaseFieldOptions.hpp

14.62 UtilsForSolvers Class Reference

Useful methods for managing solvers.

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

Collaboration diagram for UtilsForSolvers:



Public Member Functions

- void [BuildSolver](#) (mfem::IterativeSolver &solver, mfem::Solver &solv_method, mfem::Operator &o)pe)
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

<i>solver</i>	IterativeSolver
<i>preconditionner</i>	Preconditionner
<i>ope</i>	Operator

Definition at line 32 of file UtilsForSolvers.hpp.

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

```

33                                     {
34     solver.SetPreconditionner(preconditionner);
35     solver.SetOperator(ope);
36 }
```

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

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

```

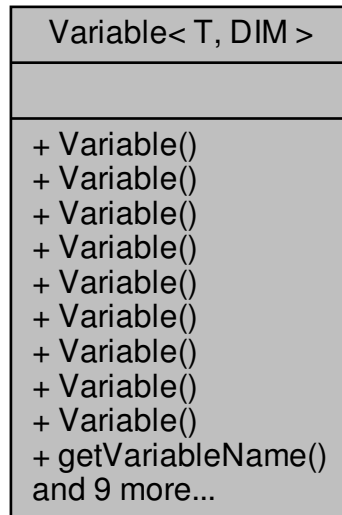
50                                     {
51     solver.SetPrintLevel(_print_level);
52     solver.iterative_mode = _iterative_mode;
53     solver.SetMaxIter(_n_iter_max);
54     solver.SetRelTol(_n_rel_tol);
55     solver.SetAbsTol(_n_abs_tol);
56 }
```

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 unknown 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

<i>fespace</i>	
<i>variable_name</i>	
<i>type</i>	
<i>initial_condition_name</i>	

Definition at line 122 of file Variable.hpp.

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

```
126 : bcs(bcs), variable_name_(variable_name) {
127   this->fespace_ = spatial->get_finite_element_space();
128   this->setVariableType(type);
129
130   this->uh_.SetSpace(fespace_);
131   const auto dim = spatial->get_dimension();
132
133   std::apply(
134     [dim, initial_condition_name, this](Args... args) {
135       Variable<T, DIM>::setInitialCondition(dim, initial_condition_name, args...);
136     },
137     args1);
138   // mfem::ConstantCoefficient cc(0.);
139   // this->uh_.ProjectCoefficient(cc);
140   // this->uh_.GetTrueDofs(this->unk_);
141 }
```

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

<i>fespace</i>	
<i>variable_name</i>	
<i>type</i>	
<i>initial_condition_name</i>	
<i>analytical_solution_name</i>	

Definition at line 154 of file Variable.hpp.

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

```

159     : bcs_(bcs), variable_name_(variable_name) {
160     this->fespace_ = spatial->get_finite_element_space();
161     this->setVariableType(type);
162
163     this->uh_.SetSpace(fespace_);
164     this->uh_ex_.SetSpace(fespace_);
165     const auto dim = spatial->get_dimension();
166     std::apply(
167         [dim, initial_condition_name, this](Args1... args) {
168             this->setInitialCondition(dim, initial_condition_name, args...);
169         },
170         args1);
171     std::apply(
172         [dim, analytical_solution_name, this](Args2... args) {
173             this->setAnalyticalSolution(dim, analytical_solution_name, args...);
174         },
175         args2);
176 }
```

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

<i>fespace</i>	
<i>variable_name</i>	
<i>type</i>	
<i>initial_condition_name</i>	
<i>analytical_solution_function</i>	

Definition at line 189 of file Variable.hpp.

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

```

194     : bcs_(bcs), variable_name_(variable_name) {
195     this->fespace_ = spatial->get_finite_element_space();
196     this->setVariableType(type);
197
198     this->uh_.SetSpace(fespace_);
199
200     this->uh_ex_.SetSpace(fespace_);
201     const auto dim = spatial->get_dimension();
202     std::apply([dim, initial_condition_name, this](
203         Args... args) { this->setInitialCondition(dim, initial_condition_name, args...); },
204         args1);
205     this->setAnalyticalSolution(analytical_solution_function);
206 }
```

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

<i>fespace</i>	
<i>variable_name</i>	
<i>type</i>	
<i>initial_condition_function</i>	

Definition at line 217 of file Variable.hpp.

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

```

221     : bcs_(bcs), variable_name_(variable_name) {
222     this->fespace_ = spatial->get_finite_element_space();
223     this->setVariableType(type);
224
225     this->uh_.SetSpace(fespace_);
226     this->setInitialCondition(initial_condition_function);
227 }
```

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

<i>fespace</i>	
<i>variable_name</i>	
<i>type</i>	
<i>initial_condition_function</i>	
<i>analytical_solution_name</i>	

Definition at line 240 of file Variable.hpp.

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

```

245     : bcs(bcs), variable_name_(variable_name) {
246     this->fespace_ = spatial->get_finite_element_space();
247     this->setVariableType(type);
248
249     this->uh_.SetSpace(fespace_);
250     this->uh_ex_.SetSpace(fespace_);
251     const auto dim = spatial->get_dimension();
252     this->setInitialCondition(initial_condition_function);
253     std::apply(
254         [dim, analytical_solution_name, this](Args... args) {
255             this->setAnalyticalSolution(dim, analytical_solution_name, args...);
256         },
257         args1);
258 }

```

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

<i>fespace</i>	
<i>variable_name</i>	
<i>type</i>	
<i>initial_condition_function</i>	
<i>analytical_solution_function</i>	

Definition at line 270 of file Variable.hpp.

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

```

275     : bcs_(bcs), variable_name_(variable_name) {
276     this->fespace_ = spatial->get_finite_element_space();
277     this->setVariableType(type);
278
279     this->uh_.SetSpace(fespace_);
280     this->uh_ex_.SetSpace(fespace_);
281     this->setInitialCondition(initial_condition_function);
282     this->setAnalyticalSolution(analytical_solution_function);
283 }
```

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

<i>fespace</i>	
<i>variable_name</i>	
<i>type</i>	
<i>initial_condition_value</i>	

Definition at line 294 of file Variable.hpp.

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

```

297     : bcs_(bcs), variable_name_(variable_name) {
298     this->fespace_ = spatial->get_finite_element_space();
299     this->setVariableType(type);
300     this->uh_.SetSpace(fespace_);
301     this->setInitialCondition(initial_condition_value);
302 }
```

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

<i>fespace</i>	
<i>variable_name</i>	
<i>type</i>	
<i>initial_condition_value</i>	
<i>analytical_solution_name</i>	

Definition at line 314 of file Variable.hpp.

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

```

318     : bcs(bcs), variable_name_(variable_name) {
319     this->fespace_ = spatial->get_finite_element_space();
320     this->setVariableType(type);
321
322     this->uh_.SetSpace(fespace_);
323     this->uh_ex_.SetSpace(fespace_);
324     const auto dim = spatial->get_dimension();
325     this->setInitialCondition(initial_condition_value);
326     std::apply(
327         [dim, analytical_solution_name, this](Args... args) {
328             this->setAnalyticalSolution(dim, analytical_solution_name, args...);
329         },
330         args1);
331 }

```

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

<i>fespace</i>	
<i>variable_name</i>	
<i>type</i>	
<i>initial_condition_value</i>	
<i>analytical_solution_function</i>	

Definition at line 343 of file Variable.hpp.

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

```

347     : bcs_(bcs), variable_name_(variable_name) {
348   this->fespace_ = spatial->get_finite_element_space();
349   this->setVariableType(type);
350
351   this->uh_.SetSpace(fespace_);
352   this->uh_ex_.SetSpace(fespace_);
353   this->setInitialCondition(initial_condition_value);
354   this->setAnalyticalSolution(analytical_solution_function);
355 }
```

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

<i>T</i>	
<i>DIM</i>	

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:

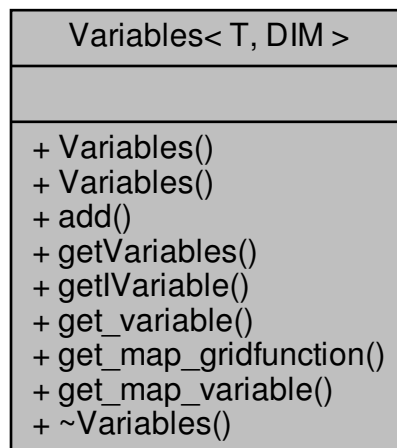
- Variable.hpp

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

<i>Args</i>	
-------------	--

Parameters

<i>args</i>	
-------------	--

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

<i>var</i>	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

<i>vname</i>	
--------------	--

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>i</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.

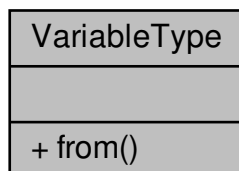
```
83                                     {
84   return this->vect_variables_;
85 }
```

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

- ~SpatialDiscretization
 - SpatialDiscretization, [121](#)
- add
 - Parameters, [72](#)
 - Variables, [153](#)
- AllenCahnSpecializedNLFormIntegrator
 - AllenCahnSpecializedNLFormIntegrator, [28](#)
 - AssembleElementGrad, [29](#)
 - AssembleElementVector, [30](#)
- AllenCahnSpecializedNLFormIntegrator< SCHEME >, [27](#)
- AnalyticalFunctions
 - getAnalyticalFunctions, [32](#)
- AnalyticalFunctions< DIM >, [32](#)
- AnalyticalFunctionsType, [33](#)
- apply_uniform_refinement
 - SpatialDiscretization, [121](#)
- AssembleElementGrad
 - AllenCahnSpecializedNLFormIntegrator, [29](#)
 - CahnHilliardSpecializedNLFormIntegrator, [43](#)
- AssembleElementVector
 - AllenCahnSpecializedNLFormIntegrator, [30](#)
 - CahnHilliardSpecializedNLFormIntegrator, [44](#)
- Boundaries, [34](#)
- Boundary, [35](#)
 - Boundary, [35](#)
 - get_boundary_index, [36](#)
 - get_boundary_value, [36](#)
 - is_essential_boundary, [37](#)
- BoundaryConditionType, [40](#)
- BoundaryConditions
 - BoundaryConditions, [38](#)
 - GetEssentialDofs, [39](#)
 - SetBoundaryConditions, [39](#)
- BoundaryConditions< T, DIM >, [37](#)
- BuildSolver
 - UtilsForSolvers, [139](#)
- CahnHilliardSpecializedNLFormIntegrator
 - AssembleElementGrad, [43](#)
 - AssembleElementVector, [44](#)
 - CahnHilliardSpecializedNLFormIntegrator, [42](#)
- CahnHilliardSpecializedNLFormIntegrator< SCHEME >, [41](#)
- ConductionOperator, [47](#)
 - ImplicitSolve, [49](#)
- DiffusionNLFItegrator, [50](#)
- DiffusionNLFItegrator, [52](#), [53](#)
- EnergyCoefficient, [54](#)
- get_analytical_solution
 - Variable, [149](#)
- get_boundary_conditions
 - Variable, [149](#)
- get_boundary_index
 - Boundary, [36](#)
- get_boundary_value
 - Boundary, [36](#)
- get_dimension
 - SpatialDiscretization, [122](#)
- get_finite_element_space
 - SpatialDiscretization, [122](#)
- get_frequency
 - PostProcessing, [88](#)
- get_gf
 - Variable, [150](#)
- get_map_gridfunction
 - Variables, [154](#)
- get_map_variable
 - Variables, [154](#)
- get_max_bdr_attributes
 - SpatialDiscretization, [123](#)
- get_mesh
 - SpatialDiscretization, [123](#)
- get_parameter_value
 - Parameters, [73](#)
- get_unknown
 - Variable, [150](#)
- get_variable
 - Variables, [154](#)
- getAnalyticalFunctions
 - AnalyticalFunctions, [32](#)
- GetEssentialDofs
 - BoundaryConditions, [39](#)
- getIVariable
 - Variables, [155](#)
- getMapParameters
 - Parameters, [73](#)
- getName
 - Parameter, [70](#)
- getPotentialFunction
 - PotentialFunctions, [117](#)
- getSize
 - SpatialDiscretization, [124](#)
- getValue
 - Parameter, [70](#)

- getVariableName
 - Variable, [150](#)
- getVariableType
 - Variable, [151](#)
- getVariables
 - Variables, [156](#)
- getH
 - potential_function< 0, ThermodynamicsPotential↔
Discretization::Explicit >, [91](#)
 - potential_function< 0, ThermodynamicsPotential↔
Discretization::Implicit >, [94](#)
 - potential_function< 0, ThermodynamicsPotential↔
Discretization::SemImplicit >, [97](#)
 - potential_function< 1, ThermodynamicsPotential↔
Discretization::Explicit >, [99](#)
 - potential_function< 1, ThermodynamicsPotential↔
Discretization::Implicit >, [102](#)
 - potential_function< 1, ThermodynamicsPotential↔
Discretization::SemImplicit >, [105](#)
 - potential_function< 2, ThermodynamicsPotential↔
Discretization::Explicit >, [108](#)
 - potential_function< 2, ThermodynamicsPotential↔
Discretization::Implicit >, [111](#)
 - potential_function< 2, ThermodynamicsPotential↔
Discretization::SemImplicit >, [113](#)
- getW
 - potential_function< 0, ThermodynamicsPotential↔
Discretization::Explicit >, [92](#)
 - potential_function< 0, ThermodynamicsPotential↔
Discretization::Implicit >, [95](#)
 - potential_function< 0, ThermodynamicsPotential↔
Discretization::SemImplicit >, [97](#)
 - potential_function< 1, ThermodynamicsPotential↔
Discretization::Explicit >, [100](#)
 - potential_function< 1, ThermodynamicsPotential↔
Discretization::Implicit >, [103](#)
 - potential_function< 1, ThermodynamicsPotential↔
Discretization::SemImplicit >, [106](#)
 - potential_function< 2, ThermodynamicsPotential↔
Discretization::Explicit >, [109](#)
 - potential_function< 2, ThermodynamicsPotential↔
Discretization::Implicit >, [111](#)
 - potential_function< 2, ThermodynamicsPotential↔
Discretization::SemImplicit >, [114](#)
- getX
 - potential_function< 0, ThermodynamicsPotential↔
Discretization::Explicit >, [92](#)
 - potential_function< 0, ThermodynamicsPotential↔
Discretization::Implicit >, [95](#)
 - potential_function< 0, ThermodynamicsPotential↔
Discretization::SemImplicit >, [98](#)
 - potential_function< 1, ThermodynamicsPotential↔
Discretization::Explicit >, [101](#)
 - potential_function< 1, ThermodynamicsPotential↔
Discretization::Implicit >, [103](#)
 - potential_function< 1, ThermodynamicsPotential↔
Discretization::SemImplicit >, [106](#)
 - potential_function< 2, ThermodynamicsPotential↔
Discretization::Explicit >, [109](#)
 - potential_function< 2, ThermodynamicsPotential↔
Discretization::Implicit >, [112](#)
 - potential_function< 2, ThermodynamicsPotential↔
Discretization::SemImplicit >, [115](#)
- ImplicitSolve
 - ConductionOperator, [49](#)
 - PhaseFieldOperator, [78](#)
- initialize
 - PhaseFieldOperator, [79](#)
- InterfacialCoefficient, [55](#)
- is_essential_boundary
 - Boundary, [37](#)
- make_periodic_mesh
 - SpatialDiscretization, [124](#)
- MassDefaultConstant, [25](#)
- MeltingCoefficient, [57](#)
- Meshes, [58](#)
- MobilityCoefficient, [60](#)
- Mult
 - PhaseFieldOperator, [79](#)
- multidimension_function< 1 >, [62](#)
- multidimension_function< 2 >, [63](#)
- multidimension_function< 3 >, [64](#)
- multidimension_function< DIM >, [61](#)
- my_best_constructor< T, 1 >, [65](#)
- my_best_constructor< T, 2 >, [66](#)
- my_best_constructor< T, 3 >, [67](#)
- my_best_constructor< T, DIM >, [65](#)
- need_to_be_saved
 - PostProcessing, [88](#)
- NewtonDefaultConstant, [25](#)
- NonlinearCoefficient, [68](#)
- Parameter, [70](#)
 - getName, [70](#)
 - getValue, [70](#)
- Parameters, [71](#)
 - add, [72](#)
 - get_parameter_value, [73](#)
 - getMapParameters, [73](#)
 - Parameters, [72](#)
- PhaseFieldEnergy
 - PhaseFieldOperator, [80](#)
- PhaseFieldOperator
 - ImplicitSolve, [78](#)
 - initialize, [79](#)
 - Mult, [79](#)
 - PhaseFieldEnergy, [80](#)
 - PhaseFieldOperator, [77](#)
 - reduced_oper, [82](#)
 - SetConstantParameters, [81](#)
 - SetTransientParameters, [81](#)
- PhaseFieldOperator< T, DIM, NLFI >, [74](#)
- PhaseFieldPrivate::mmap< EType >, [59](#)
- PhaseFieldReducedOperator, [83](#)

- PoissonNLFIntegrator, [84](#)
- PostProcessing
 - get_frequency, [88](#)
 - need_to_be_saved, [88](#)
 - PostProcessing, [88](#)
 - save_variables, [89](#)
- PostProcessing< T, DC, DIM >, [86](#)
- potential_function< 0, ThermodynamicsPotential↔
 - Discretization::Explicit >, [91](#)
 - getH, [91](#)
 - getW, [92](#)
 - getX, [92](#)
- potential_function< 0, ThermodynamicsPotential↔
 - Discretization::Implicit >, [93](#)
 - getH, [94](#)
 - getW, [95](#)
 - getX, [95](#)
- potential_function< 0, ThermodynamicsPotential↔
 - Discretization::Semimplicit >, [96](#)
 - getH, [97](#)
 - getW, [97](#)
 - getX, [98](#)
- potential_function< 1, ThermodynamicsPotential↔
 - Discretization::Explicit >, [99](#)
 - getH, [99](#)
 - getW, [100](#)
 - getX, [101](#)
- potential_function< 1, ThermodynamicsPotential↔
 - Discretization::Implicit >, [102](#)
 - getH, [102](#)
 - getW, [103](#)
 - getX, [103](#)
- potential_function< 1, ThermodynamicsPotential↔
 - Discretization::Semimplicit >, [104](#)
 - getH, [105](#)
 - getW, [106](#)
 - getX, [106](#)
- potential_function< 2, ThermodynamicsPotential↔
 - Discretization::Explicit >, [107](#)
 - getH, [108](#)
 - getW, [109](#)
 - getX, [109](#)
- potential_function< 2, ThermodynamicsPotential↔
 - Discretization::Implicit >, [110](#)
 - getH, [111](#)
 - getW, [111](#)
 - getX, [112](#)
- potential_function< 2, ThermodynamicsPotential↔
 - Discretization::Semimplicit >, [113](#)
 - getH, [113](#)
 - getW, [114](#)
 - getX, [115](#)
- potential_function< ORDER, SCHEME >, [90](#)
- PotentialFunctions
 - getPotentialFunction, [117](#)
- PotentialFunctions< ORDER, SCHEME >, [116](#)
- Problem, [118](#)
- Problems, [119](#)
- reduced_oper
 - PhaseFieldOperator, [82](#)
- save_variables
 - PostProcessing, [89](#)
- set_finite_element_space
 - SpatialDiscretization, [125](#)
- SetBoundaryConditions
 - BoundaryConditions, [39](#)
- SetConstantParameters
 - PhaseFieldOperator, [81](#)
- SetSolverParameters
 - UtilsForSolvers, [140](#)
- SetTransientParameters
 - PhaseFieldOperator, [81](#)
- SpatialDiscretization
 - ~SpatialDiscretization, [121](#)
 - apply_uniform_refinement, [121](#)
 - get_dimension, [122](#)
 - get_finite_element_space, [122](#)
 - get_max_bdr_attributes, [123](#)
 - get_mesh, [123](#)
 - getSize, [124](#)
 - make_periodic_mesh, [124](#)
 - set_finite_element_space, [125](#)
- SpatialDiscretization< T, DIM >, [120](#)
- ThermodynamicRootFindingAlgorithm, [126](#)
- ThermodynamicSubModels, [133](#)
- ThermodynamicSystemsForMobilities, [135](#)
- ThermodynamicsAllenCahnMobility, [126](#)
- ThermodynamicsFGR, [127](#)
- ThermodynamicsInitialization, [128](#)
- ThermodynamicsLinearPowerSourceTerm, [128](#)
- ThermodynamicsModel, [129](#)
- ThermodynamicsMolarVolumeType, [130](#)
- ThermodynamicsPotentials, [131](#)
- ThermodynamicsProperties, [132](#)
- ThermodynamicsStabilizationScheme, [132](#)
- ThermodynamicsUnitForSettingConditions, [134](#)
- ThermodynamicsUpdateQuantitiesMethod, [135](#)
- TimeDiscretization
 - TimeDiscretization, [137](#)
- TimeDiscretization< T, DC, DIM, NLFI >, [136](#)
- TimeScheme, [138](#)
- UtilsForSolvers, [139](#)
 - BuildSolver, [139](#)
 - SetSolverParameters, [140](#)
- Variable
 - get_analytical_solution, [149](#)
 - get_boundary_conditions, [149](#)
 - get_gf, [150](#)
 - get_unknown, [150](#)
 - getVariableName, [150](#)
 - getVariableType, [151](#)
 - Variable, [142–148](#)
- Variable< T, DIM >, [141](#)

VariableType, [156](#)

Variables

 add, [153](#)

 get_map_gridfunction, [154](#)

 get_map_variable, [154](#)

 get_variable, [154](#)

 getIVariable, [155](#)

 getVariables, [156](#)

 Variables, [153](#)

Variables< T, DIM >, [152](#)